Automatic Clustering of Malware Variants

Rima Asmar Awad
Oak Ridge National Laboratory
Oak Ridge, Tennessee
awadr@ornl.gov

Kirk D. Sayre
Oak Ridge National Laboratory
Oak Ridge, Tennessee
sayrekd@ornl.gov

Abstract—The emergence of malware creation tools in recent years has facilitated the creation of new variations of existing malware instances. Typically, Anti-Virus companies process new malware instances manually to determine their maliciousness and generate their signatures. However, with the overwhelming number of new malware variants that are created automatically to evade pattern based detection, manual analysis is becoming a bottleneck that hinders the process of responding to new threats. This paper proposes a novel method to automatically cluster malware variants into malware families based on the structured control flow graphs of the malware instances. Our final results demonstrate high effectiveness in terms of accuracy, an average of 94% accuracy, and speed in clustering malware variants.

I. INTRODUCTION

The growing popularity of automatic malware creation toolkits has had a great effect on increasing malware proliferation. Malware writers with minimal skills can now create variants of previously seen malware by using malware variant generation tools. Such tools automatically alter the source code or the binary of existing malware instances to evade Anti-Virus (AV) detectors [13]. Consequently, slight variants of known malware instances are often able to fool current detection techniques, while operating with the same functionality as the original malware instance.

According to Symantec, 400 million new variants of malware were created in 2011, an average of 33 million new variants of malware per month [23]. There are only a few hundred new malware families observed in a year, but there are thousands of variants generated for each malware family [20][24]. For example, Agobot, an instance of malware in the computer worm family, has been observed to have more than 580 variants since its initial release in 2002 [2].

When a new malware instance is released, malware analysts have to analyze it to determine if it is malicious or benign, then identify the family to which it belongs, and finally generate its signature and add it to the signature database. The average time required to manually analyze a new malware instance and generate its signature is between 2 - 48 hours [1][16][15]. Based on an average of one million new malware instances released daily, a minimum of two million hours a day to a maximum of 48 million hours a day are needed to manually analyze all newly released malware and respond to them in a timely manner. With the scarcity and high costs of human resources associated with the manual analysis, AV companies have become less responsive to the threats[6].

The problem stems from the fact that variants of previously seen malware are typically generated automatically with a low cost and at a fast pace, while analysis and classification of malware variants is still done manually [12] [13]. Manual analysis remains an expensive, tedious and time consuming process. To combat the malware variants, it is necessary to develop an accurate and automatic malware variant clustering method that speeds up the clustering of existing malware variants into families, and, eventually, facilitates the classification of new variants of known malware families [12]. Previous efforts to automatically cluster malware variants focused mainly on binary-based signatures. Unfortunately, the automatic creation of malware variants alters the binary signature which leads such efforts to prove ineffective.

In this paper, we propose a novel method that addresses the limitations of existing automatic clustering methods. The proposed method uses the structured control flow graphs (SCFG) of malware variants, generated by Hyperion binary static analysis tool[18], as the unique and indicative of a specific malware family. To estimate the similarity between malware instances of our sample, we calculate the
string edit distance between the string representations of the SCFG of malware instances. We then use the Quality Threshold (QT) clustering algorithm to cluster similar SCFG strings based on a suitable clustering threshold value.

To evaluate the effectiveness of our method, we used a sample of malicious unpacked Windows console applications. According to our results, we were able to accurately identify variants of the same malware family.

The rest of the paper is structured as follows: Section 2 presents an overview of previous malware variant identification research. Section 3 demonstrates the general methodology of our clustering process. Section 4 presents the results of our proposed method. Section 6 concludes and summarizes the paper, and section 7 discusses future works.

II. RELATED WORK

In an effort to automate malware analysis and the clustering of malware variants, computer scientists have been using two basic approaches to analyze malware and extract the unique features of malware families: Static analysis and dynamic analysis.

In the static analysis, the binary instructions of the malware instance are scanned to determine the properties of the program without the need to run it [9]. Static analysis based malware variant identification approaches, such as approaches using structured control flow graphs [10] [9], instruction frequencies [12], and length of functions [24], have been effective in identifying malware variants without the need to run the malicious software. This is due to the fact that the static analysis examines all the possible execution paths by scanning the binary instructions of the malware executable, even the parts that are not executed when the program runs under specific conditions. Analyzing the binary instructions of the executable allows for a prediction of the full program behavior under all execution conditions[12] [24].

The static analysis offers significant advantages, but it suffers from some limitations. The lack of access to the malware source code [5], which allows only for the analysis of the compiled binary is one of the limitations. Binary analysis is a difficult and time consuming process that can fail when advanced obfuscation techniques are applied to the malware [11]. It has been argued also that the static analysis does not provide a fully predictable behavior of large programs[25].

In dynamic analysis the malware instance is actually executed in a sandbox or emulator (controlled environments) and its behavior is monitored at runtime [27]. Dynamic analysis based malware variant identification approach used in [2] attempted to classify malware based on monitoring the operating system state changes, while the approach proposed in [22] relies on exploiting shared behavioral patterns and using a machine learning algorithm. In [11] [4] the authors build behavioral profiles based on collected tainted information and execution traces.

Dynamic analysis has the advantage of generating fast and accurate analysis results regardless of the level of obfuscation used in the malware executable. While having some advantages, dynamic approaches have the limitation of generating the program's behavior under only specific conditions [14]. Dynamic analysis is also considered dangerous; the execution of malware outside of a sandbox can cause potential harm to the device used for the execution. Dynamic analysis may also be foiled by the anti-emulation/virtualization technology used by some malware to detect emulated or virtual execution environments. Anti-emulator functionality embedded in malware programs prevents the execution of the malicious parts of the program when an emulated environment is detected [28].

III. METHODOLOGY

This section describes our developed clustering method which is based on the analysis results generated by Hyperion, a static binary analysis tool [17] [18]. Our clustering method involves four main phases:

i. Generating Structured Control Flow Graphs

To generate the string representations of the SCFG of the local functions contained in malware variants, we use Hyperion. Hyperion is an automatic static analysis tool that was developed by the Community Emergency Response Team (CERT) organization at Carnegie Mellon University and Oak Ridge National Laboratory (ORNL) to augment the manual malware analysis process with a relatively fast and automatic analysis of malware behavior [17] [18].

Hyperion begins the analysis process by taking the input binary and disassembling its instructions. The assembly instructions are then augmented with their
corresponding functional semantics that accounts for all the effects each instruction has on the state of the hardware executing the instruction.

To generates the true unstructured control flow graph of the executable, Hyperion handles certain forms of control flow obfuscation, such as the arbitrary jumps and the insertion of no-op code, by performing a recursive disassembly of the input executable and generating the control flow graphs minus computed calls and jumps [18].

The unstructured control flow graph is then transformed into a functionally equivalent structured control flow by applying the Structure Theorem [19]. The structured form of a program's control flow graph represents its different execution paths in the form of program primes with arbitrary jumps eliminated. The Structure Theorem guarantees that any program logic can be expressed in nested and sequenced single-entry, single-exit structures (called program primes).

As an optional form of output, the SCFG of the malware sample set is generated by Hyperion in the form of regular expression (regex) strings. Figure 1 show an example of CFG of a local function in a malware instance before and after, being abstracted using regex. In the regex string in figure 1, the assembly instructions of the malware instance have been abstracted in the form of (.) and (+). The (.) symbol in the control flow regex implies that any assembly instruction may be matched. The (+) sign represents that there are one or more assembly instructions. The regex string also includes information about called external functions (that is, functions called from DLLs), which is necessary to determine the malicious behavior of the program. External calls are represented in the form of brackets with the name of the external call inside. In addition to the abstracted assembly instructions and external calls, the regex strings contains information about the control structures of the program.

ii. Calculating String Edit Distance

To determine the similarity between SCFG regex strings, we have implemented the Sift3 string edit distance algorithm, which performs 10 to 25 times faster than the Levenshtein string edit distance algorithm [8], and has a time complexity of:

\[
O(\min(n,m)\maxOffset)
\]

To decrease time costs, regex strings of structured control flow were initially grouped based on their size; strings with significant differences between their lengths will not cluster together as their edit distance exceeds the QT threshold value. This prevents the system from performing all pairwise comparisons between regex strings. The control flow regex strings roughly clustered by length were then passed to the clustering function of our automatic clustering application.

iii. Clustering Strings of Local Functions SCFG

Our developed tool applies the Quality Threshold Clustering (QTC) algorithm to cluster the generated structured control flow regex strings. QTC is a clustering algorithm that groups similar objects based on a pre-defined quality threshold value that controls the maximum distance allowed between any two elements within a cluster [7].

QTC is perceived to be computationally expensive, but it was an attractive option for our work because, unlike other clustering algorithms, it does not require a priori knowledge of the number of clusters. The lack of a predefined number of clusters into which to cluster the data ensures the quality of the output clusters, and produces non-overlapping clusters with diameter that does not exceed the predetermined threshold value [7]. The clustering output consists of clusters of related elements, while the elements that not related to other elements remain outside and are not forced into a cluster.
To estimate a suitable quality threshold value that should be used by the QTC algorithm to accurately cluster regex strings of structured control flow, we have calculated the smallest regex string length of each group and divided it by a configurable constant value. We noticed that more accurate results are generated when the quality threshold is set to the average regex string length divided by a higher value, that is, when the threshold is set so that there is a high degree of similarity between elements in a cluster. The final result of this phase was clusters of local functions that share very similar SCFG regex strings.

iv. Clustering Malware Variants Based on the Number of Shared Functions

During this phase, malware local functions that have very similar structured control flows, were mapped back to the malware instance containing the local functions. The result of this mapping was clusters of malware instances that we believed share at least one local function based on the structured control flow of the local functions. Because malware instances that share higher number of functions are more likely to be variants of the same family, we worked on calculating the number of shared functions between every pairwise combination of malware instances in the sample set.

To improve accuracy, we generated an accurate relation between the pair combination of malware instances of the sample set, in the form of percentage of shared local functions between each pair of malware instance. We then run the clustering tool using different percentage's values of shared function to get the best results, and added the edges of our final graph, that represent the relationship between the malware instances, based on the used percentage. The generated directed graph consisted of 303 nodes, representing the malware instances, and edges representing the relationship between malware instances.

To visualize our results, we have used Gephi, an open source tool for visualizing and analyzing graphs [3]. We used Gephi to calculate the modularity of the graph, which describes how the graph is compartmentalized into sub-graphs based on the strength between node clusters. Finally, we obtained a graph that groups malware instances with a high similarity into clusters and presents them using unique colors as shown in figure 2.

IV. RESULTS

To evaluate the developed method, we used Hyperion to analyze a sample set consisting of 303 Windows unpacked console malware instances.

To verify the accuracy of our results, we randomly picked clusters, and scanned the malware instances of the cluster using VirusTotal [26]. The analysis generated by the 40 scanners used by VirusTotal confirmed that most of the malware instances of the clusters belong to the same family.

We have noticed that the quality of the clusters generated by QTC is dependent on the predetermined cluster membership threshold value provided by the user. To provide the best estimation of threshold value and the percentage of shared functions that we should use, we ran our experiment three times using different calculated threshold values and different percentages of shared functions.

In experiment one, the threshold value was calculated by dividing the smallest regex string length in each group by five. Edges between pairs of malware variants were added when a pair of malware instances shared 50%, 70%, and 95% or more of their total local functions. Our results reflected a percentage of agreement with the commercial AV scanners that ranged between 61% - 85%.

For experiment two, the threshold value was tightened up; it was calculated using the value of the smallest regex string length divided by twenty, and the edges between nodes of the graph were added when a pair of malware shared %50, %70, and %95 or more of local functions. The results obtained in this experiment indicated a percentage of agreement that ranges between %84 - %92.
Finally, in experiment three we further tightened the threshold value, by calculating the length of the smallest regex string length, and dividing it by fifty. Again, edges between pairs of malware variants were added when a pair of malware shared %50, %70, and %95 or more of local functions. The results of experiment three showed a percentage of agreement that ranges between %92 - %94.

Figure 2 is a graph of clusters generated by Gephi for the third run of experiment three. The Clusters that were randomly picked are highlighted. Table 1 presents the overall results of the three experiments.

### V. DISCUSSION

According to the results obtained by the three experiments in table 1, it is obvious that the value of the threshold passed to the QTC algorithm plays an important role in the accuracy of the malware variant classification. The threshold value is what determines how much difference is allowed between the malware instances local function regex strings, hence, tightening up the threshold value increases the accuracy of the obtained results by requiring greater similarity between local functions before they are classified as the same. On the other hand, using an overestimated threshold value, results in an increased number of unrelated malware instances in the malware clusters.

<table>
<thead>
<tr>
<th>Experiment/Run</th>
<th>Run One 50% Shared Functions</th>
<th>Run Two 70% Shared Functions</th>
<th>Run Three 95% Shared Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment One</td>
<td>QT = Smallest/5</td>
<td>61%</td>
<td>70%</td>
</tr>
<tr>
<td>Experiment Two</td>
<td>QT = Smallest/20</td>
<td>84%</td>
<td>90%</td>
</tr>
<tr>
<td>Experiment Three</td>
<td>QT = Smallest/50</td>
<td>92%</td>
<td>93%</td>
</tr>
</tbody>
</table>

The high accuracy of the results generated by our automatic clustering method provides an evidence that variants of a specific malware family share similar SCFG, while variants of different malware families have different SCFG. In spite of the fact that the variants used in our sample are not new, many AV scanners failed to detect them as malicious. In order for the AV scanners to detect these variants of a specific malware family, specific signatures had to be written for each AV scanner for each malware variant, while our method required only the automatic processing of the regex of SCFG strings and the application of QTC to the regex strings. No special purpose signatures were written. The malware variants families classification was performed automatically.

### VI. CONCLUSION

In this paper, we hypothesized that the structured control flow, generated by the Hyperion static reverse engineering tool, can be used as an invariant feature to automatically cluster malware variants. To present a proof the concept, we developed a method that combines a number of tools and algorithms to automatically cluster malware instances into malware families based on the SCFG of the individual local functions of malware instances.

We tested the effectiveness of our developed method by processing a sample of malicious unpacked Windows console applications. The generated results present a partial proof of the concept of our hypothesis and ascertains that the SCFG of malware instances can be used be the invariant feature used to automatically cluster malware variants into malware families.

### VII. FUTURE WORK

As future work, we plan to run our automatic clustering tool against a large representative sample of malware, and use the clustering results to build a catalog of malware families, with a single most representative malware variant selected from each family. Upon the arrival of an unknown program, the malware family representatives in the catalog of malware families can be used to check the structured control flow of the unknown program against the structured control flow of the candidate malware variants. This will allow us to use the described malware variant detection mechanism as an operational malware detection and classification system.

To speed up the clustering process, we plan to search for substitutions for the string edit distance algorithm and the QTC algorithm used in our method. Although the Sift 3 string edit distance algorithm does a better job than other string edit distance algorithm with regards to time complexity, the expenses of the sift 3 algorithm increase as the size of the sample used to test the developed method increase. Similarly, the QTC algorithm is more accurate than other clustering algorithms, but with
the increased size of the sample, it becomes computationally expensive.

REFERENCES


