Flexible Analyst Defined Viewpoint for Malware Relationship Analysis

by

James Edward Holmes

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved November 2014 by the
Graduate Supervisory Committee:

Gail-Joon Ahn, Chair
Partha Dasgupta
Adam Doupe

ARIZONA STATE UNIVERSITY

December 2014

ABSTRACT

The rate at which new malicious software (Malware) is created is consistently increasing each year. These new malwares are designed to bypass the current anti-virus countermeasures employed to protect computer systems. Security Analysts must understand the nature and intent of the malware sample in order to protect computer systems from these attacks. The large number of new malware samples received daily by computer security companies require Security Analysts to quickly determine the type, threat, and countermeasure for newly identified samples. Our approach provides for a visualization tool to assist the Security Analyst in these tasks that allows the Analyst to visually identify relationships between malware samples.

This approach consists of three steps. First, the received samples are processed by a sandbox environment to perform a dynamic behavior analysis. Second, the reports of the dynamic behavior analysis are parsed to extract identifying features which are matched against other known and analyzed samples. Lastly, those matches that are determined to express a relationship are visualized as an edge connected pair of nodes in an undirected graph.

To my parents, friends, and colleagues. Thank you for your patience and support.

To Dr. Gail-Joon Ahn, thank you for your guidance, and steadfast support over the years.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Problem Statement

Malware is a common portmanteau derived from malicious software. It is a general term that describes any software that is designed to infiltrate a computer system for misuse and or hostile gains [17]. The rate at which new malware is detected is steadily accelerating. Figure 1.1 below shows McAfee reporting that nearly 12 million new malware samples were reported in Q4 of 2012 alone, with an average rate of 100,000 new malwares per day.

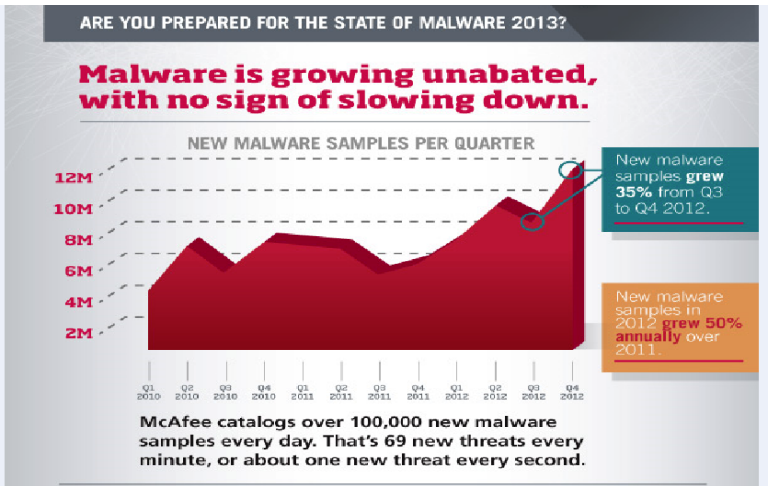

Figure 1.1: State of Malware 2013 [25]

The enormous volume of malware samples that must be categorized, analyzed, and ultimately protected against is overwhelming considering the work that must be performed by an Analyst for each sample. More tools are required to empower Security Analysts in their efforts to process, and if necessary analyze, the deluge of malware samples that are received daily.

Additionally, it is essential that the Analyst's time be spent examining those samples that are of the higher priority, rather than replicating work on derivative malwares or samples that are already handled by the current threat defense strategies. [6] echoes the need for tools that can assist the Analyst in the decision to focus on those malware samples that require more attention, such as the ones that are found to be more novel and not mere mutations of known malwares that have been disguised using obfuscation techniques.

1.2 Approach

The focus of this research has been to create a tool that can aid an Analyst in the process of examining large volumes of malware samples. The proposed approach is to enable the Analyst with a visualization tool to examine the relationship between multiple malware samples.

The visualization of the relationships is generated by clustering together the selected features extracted from the dynamic behavior analysis of the malware samples. The dynamic analysis is done within a Sandbox environment that has been configured to monitor the activity of applications as they are installed and execute on the system.

The relationships are then presented using an undirected graph. From that graph the Analyst may navigate and infer the significance of the relationships that are found within the samples. Unique patterns are automatically extracted to bring them to the analyst attention such as Complete Graphs and the set of samples that show zero correlation.

The added ability to detect the relationships between malwares and readily display this information to the Analyst will aid in the decision making process and in determining which malwares must be assigned a higher analysis priority.

[17] makes the following observation regarding the detection of unique malwares, "based on insight that most new malicious samples are simple syntactic variations of existing malware, and hence, a way to ascertain the maliciousness of an unknown sample is to check if it is sufficiently similar to any known malware".

The approach has been designed with the objective of being a flexible tool with the use case dynamically defined by the Analyst. The relationships that are identified are not meant to classify all known families in the data set. As stated by [23], the relationships are meant to "...aid the investigator in obtaining qualitative and quantitative understanding of large amounts of ... data by providing him with reasonably good similarity groups. The method should be used in close interaction with theory and intuition." As is ever the case with tools, the tool is not meant to accomplish the task without interaction from the operator. The tool is meant to enhance the ability of the operator or Analyst.

Chapter 2

BACKGROUND

2.1 Malware

As previously stated, Malware is a common term to describe malicious software that is designed to exploit and harm computer system. There are multiple types of malware that differ based on their intended targets and attacks. [5] categorizes common malware types as the following: Virus, Trojan Horse, Worm, Adware, Spyware, Zombie(Bot-ware).

As noted in the introduction, the rate of malware creation is accelerating. In addition to the repacking of existing malware to prevent their detection by signature based deterrent systems, new malwares are becoming increasing easy to create with the use of automated builders that allow users of limited ability to create new malware samples with a tailored attack vector and objectives. The Zeus crimeware toolkit and Spy Eye are two builders that are easily obtained for a small fee to allow an attacker to create new malwares [34].

2.2 Malware Analysis Techniques

In order to analyze a malware sample an analyst has two options, either static analysis or dynamic analysis.

2.2.1 Static Analysis

Static analysis consists of any analysis that can be done to identify and understand the malware sample without allowing the sample to execute. Common tools used

during static malware analysis, that will be introduced below, are hash functions, string extraction, packer identification, and disassembly.

A common step when analyzing a malware sample is to compute a hash value for the sample. The hash function computes a unique number based on the malware sample. This can be used to identify the malware in the future regardless of the file name, and as such is very useful when recording results, compiling notes, and comparing with online databases. A hash value computed using a method such as MD5 or SHA1 will positively identify only those files that are exactly the same as the file from which the hash value was generated. However if the malware sample is altered, the hash file will no longer be valid. Fuzzy hashing has been implemented to attempt to address the case in which an analyst would like to know if two programs are nearly the same. Fuzzy hashing creates a series of hash values from different locations within the binary data of the sample [12]. The result is that two fuzzy hashes can be compared to easily determine if two pieces of software are identical or very similar. A popular tool used to perform the fuzzy hashing computation is SSDEEP [20].

String detection is another simple tool that can help to identify useful information contained within the sample. String detect can be run in Unix environments from the command line using the *strings* command, or on Windows using the Strings app from Sysinternals from Microsoft. Strings returned from this analysis can help the Analyst determine if the file has been encrypted, or give details about IP addresses to which the malware is attempting to connect.

Additionally, the Analyst can use PEiD to determine what type of packer, cryptor, or compiler has been used to try to hide the contents of the malware, these are discussed in more detail in section 2.3. If the packer or compiler are correctly de-

tected, the Analyst can use this information help unpack and disassemble the malware sample.

Disassembly is the most difficult of the static analysis approaches discussed in this section. It consists of using a tool such as IDA Pro to break the program down into its assembly instructions. The Analyst must then examine these instructions to understand what action the program is attempting to execute. Add-ons to IDA Pro such as Fast Library Identification and Recognition Technology (FLIRT) can identify standard library functions in the assemble code [15]. This helps the Analyst to more easily read and parse the disassembled program.

2.2.2 Dynamic Analysis

Dynamic analysis extends what can be learned from the static analysis of the malware by allowing it to execute and observing what the malware does during execution.

Precautions should be taken when executing a malware sample. The objective of the malware is to do harm, as such the execution and observation of the malware is typically done within a sandbox environment to isolate the sample. The sandbox will typically present the malware with a virtual machine on which to execute. The sandbox is used to limit the risk of infection to other computers on the local network. The virtual machine is used because it is simple to reset a virtual machine to its previous snapshot state. This allows all the samples to be tested under similar conditions. There are multiple different Sandboxes available to the user such as CWSandbox [9], Anubis [3],and Truman [35].

During the observation, the Analyst may choose which changes are observed in the system to determine how the malware is infecting or attacking the virtual ma-

chine. The tools typically consist of system call monitors, file monitors, and network monitors.

The system calls are made when an application makes requests to the OS. [32] discusses how the CWSandbox is set up to hook the system calls that are made by the executing malware sample. Each call is first recorded to a log file and then passed on to the kernel to allow the execution to take place. Following execution, this log file can be examined to determine the behavior of the sample as it executed in the virtual machine.

A file monitor can be used to determine which files have been accessed or changed by the malware during execution. This can either be accomplished by examining the change of state of the virtual machine prior to infection and following infection, or by using tools such as Tripwire that look for changes in hash values of the files in the virtual machine.

A network monitor can be installed inside or outside of the virtual machine. The role of this monitor is to determine what requests are made to the network and what traffic is received.

These generated logs can be combined to create an understanding of the changes the malware is affecting on the system. This analysis is often times more time costly than static analysis because the malware must be given time and resources to execute. However, the advantage of using the dynamic analysis over the static analysis is that many of the techniques that are used to hide and protect the malware from analysis, as discussed in section 2.3 below, are not effective against dynamic analysis because the malware will ultimately need to run on a system and effect changes if it is to do harm.

With that advantage in mind, dynamic analysis typically performs poorly in the presence of triggers. These triggers can be as simple as a date or time. If the malware

is executed and monitored during a period that does not coincide with a previously determined date or time the sample can perform harmless operations that would not be flagged as malicious. Furthermore, the sample may lay dormant and await an external trigger from a command and control node as is often found in botnet operation. Unless the malware is being actively analyzed when, or possibly following, the broadcast of the trigger command, no malicious activity may be observed. [27] utilizes a system to perform dynamic execution of an executable. During execution the program creates a snapshot when it encounters a branching point. If it is found that the executable attempts to exit prematurely due to the lack of an external trigger, the program is able to back track and mimic the presence of the trigger. By performing this process multiple times a more complete analysis can be generated with or without the presence of the external triggers.

Generally, the techniques discussed below in section 2.3 are very effective at disguising the true identify of the malware application when only static analysis is used. [28] was able to reliably mask the detection of a set of worms using a series of obfuscation steps. The authors concluded that using the obfuscation was sufficient to bypass the static detection, however they do state that the presence of the jumbled code produced after the obfuscation steps are applied would also serve to identify that the sample is malicious. While it may be different than the remainder of the samples in the set, it is not sufficient to infer malicious intent. Many commercial applications employ obfuscation to protect their intellectual property[22]; it was for this purpose that obfuscation was created. These obfuscated programs are not malicious, simply protected from open attempts to disassemble. [28] concedes that the dynamic analysis is preferable to static analysis when handling of an obfuscated sample.

2.3 Malware Anti-Analysis Techniques

Creators of malware are aware that their software must compete with Antivirus technologies. In order to allow their malware programs to continue to be affective, the malware needs to be able to infect in spite of the presence of Antivirus software on the target computers. Antivirus tools typically seek to identify and quarantine a malware program using a combination of signature detect, static analysis, and behavior heuristics. In order to evade these defensive strategies the malware community has produced tools to allow the malware creator to easily manipulate the malware application allowing for the creation of many variants of a single application without altering the underlying execution of the original malware. The creation of a large number of variants allows the malware creator to periodically release different variants that are unique to the view of the antivirus software. This allows the malware creator to stay ahead of the detection and response of the antivirus response. The following techniques are common evasion tactics that are used in the creation and hardening of variants of malware applications [29].

2.3.1 Metamorphism / Polymorphism

These techniques seek to alter the execution and structure of the program without affecting the outcome of the applications. By altering the structure and execution flow of the program the malware is able to evade antivirus software using pattern recognition designed to detect the malware such as Intrusion Prevention Systems (IPS) and Data Leakage Prevention (DLP) systems [29].

In order to bypass the pattern recognition the malware can change the structure of the original application [16]. This can be done by changing the types of the loops used in the software, swapping and renaming registers, reordering instructions, and

changing the methods of variable assignments. Figure 2.1 shown below, illustrates how a simple while loop can be easily converted into three different loop types without modifying the outcome.

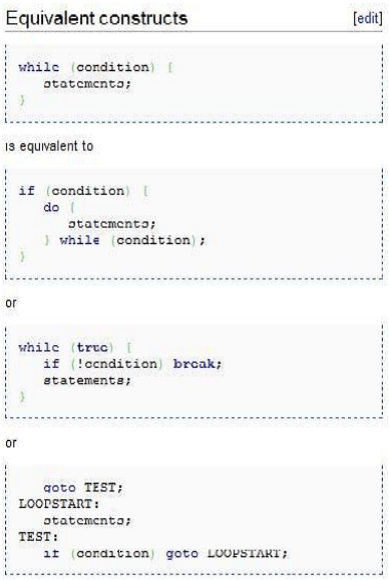

Figure 2.1: Equivalent Loop Types [29]

By combining different techniques between each variant the author creates code that is different than the previous version. These changes make it difficult to detect the original pattern of the program and alter the signature that is known to the antivirus software.

Additionally, the author can add a polymorphic step. A polymorphic malware seeks to further evade antivirus systems by combining an encrypted code block with multiple different decryptors. This allows the malware to constantly change its appearance which is fairly frustrating to static analysis [16]. The polymorphic decryptor block is mutated for new targets using code obfuscation techniques such as junk code insertion [30]. The process of generating new decryptor blocks is illustrated in Figure 2.2

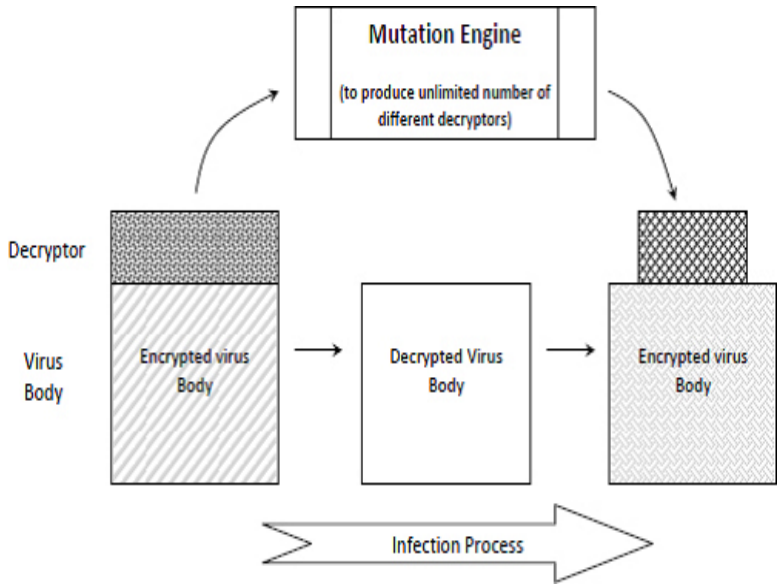

Figure 2.2: Polymorphic Virus Structure [30]

2.3.2 Noise Insertion

Noise insertion is utilized in standard code obfuscation. The effect is to add garbage code that is either redundant, performs no function, or has no impact on the code [29][30]. For example the following operations presented in Figure 2.3 are equivalent to having performed no operation.

Instruction		Operation
ADD	Reg,0	Reg←Reg+0
MOV	Reg,Reg	Reg←Reg
OR	Reg,0	Reg←Reg | 0
AND	Reg,-1	Reg←Reg&-1

Instruction		*Comments*
PUSH	***CX***	*It push value of AX into stack, later it must be turned back to AX before any effects on AX or stack memory*
...	...	
POP	***CX***	
INC	***AX***	*The value of DX increases by 14, and later before any usage of DX, its value must be changed back to its previous value*
...	...	
SUB	***AX, 1***	

Figure 2.3: No Op and Reversible Dead Code Samples [30]

In spite of having no effect on the function of the originally authored code these garbage statements modify the binary sequence of the malware. These techniques, borrowed from code obfuscation, alter the signature of the malware and make the process of reverse engineering difficult; thereby increasing the demand on the antivirus systems and the Analysts.

2.3.3 Crypters

Crypters refers to the approach of encrypting malware's binary data and only decrypting the data during runtime [29], while this technique is very similar to the approach used in polymorphism it is not the same. Crypters can be thought of as subset of polymorphism because the mutating decryption step is not assumed. The decryption step can be refined to only execute on the portions of the code that are currently required for execution. This allows the code to remain hidden and only reveal portions of the code in memory during execution in an attempt to frustrate memory inspection/dump analysis.

Figure 2.4: Examples of Crytper Tools [29]

In addition to offering encryption, Crypter tools generally offer the packers, as described below, and binders which allow for additional payloads to be carried by the malware [14]. The tools are simple to use often only requiring the author to select options [29] as shown in above in Figure 2.4. This enables an author to develop a malware package that is able to penetrate target systems and then reuse that application as a delivery vehicle for various payloads.

2.3.4 Protectors

Protectors is a general label for tools that add "anti-debugging" protection to an application. [29] states that these tools were originally developed by commercial companies to protect online games and DRM technology. A popular tool, that has been developed to protect commercial software, that has now found a home among malware creators is the Themida Advanced Windows Software Protection System from Orleans Technologies [29]. This and other tools allow the application to identify the presence of virtual machines or other debugging technologies. When these are detected the malware has the option of handling their presence by choosing to execute alternate routines. This behavior can defeat the attempts of an Analyst using a sandbox to perform a dynamic behavior analysis of the malware sample [29].

2.3.5 Packers

Packers are another example of tools that were originally developed to serve a benign, or commercial application. The Packer is used to shrink the size of the application's binary file. The packer compresses the binary file which requires the analyst to unpack the file prior to attempting to perform a static analysis of the file. The Packer also has the added benefit of making the malware package smaller which makes it easier to hide within other applications [29]. According to [29] the UPX is

currently the most popular packer in use. UPX is also a commercial application that was not originally developed for the malware community such as Themida mentioned above.

Aside from the ability to shrink the application's binary, some packers also enable polymorphism where "...the malware binary is structurally different every time the packer version is executed" [29]. The advantage the Packer gives the malware sample over the antivirus system is the ability to subvert the signature and hash based detection mechanisms.

Chapter 3

RELATED APPROACHES

Presented below are two approaches that have been previously proposed to cluster together malware samples. While the idea of grouping samples together is shared by our approach, the scale, implementation, and objectives of these approaches differ from our own.

3.1 SMIT

The Symantec Malware Indexing Tree (SMIT) [17] recognizes the need for tools to help deal with the overwhelming number of new malware samples that are received daily by antivirus companies. The first step in analyzing a new sample is to determine whether or not the sample is malicious. SMIT attempts to make this classification by determining if the new malware sample is similar to a previously know malicious malware samples.

The similarity metric is based on a function call graph. The function call graph was chosen because "a program's functionality is mostly determined by the library or system calls it invokes, its function-call graph provides a reasonable approximation to the program's run-time behavior [17]." The call graphs are generated using a disassembler during static analysis. The static analysis approach was chosen based on the time penalty incurred in using the dynamic analysis. The function call graph is chosen as the feature due to its resilience in the presence of obfuscation and evasion techniques. [17] states that the use of high-level structure, such as the call graph, is one way to overcome the obfuscation that has a greater effect on the syntax of the program.

The function call graph is traversed using a customized version of the "Hungarian method". This method was modified to apply a bias to nodes that have neighboring nodes that had already been matched. The metric is ultimately computed using an approach similar to inter-graph edit distance.

The results of the graph traversal is to return the closest match to the already know samples in the database. If the result is found to be sufficiently similar then it can be implied that the new malware sample is indeed malicious.

3.2 peHash

The peHash [38] approach attempts to solve the problem of how to identify and cluster multiple samples of a single malware "specimen". [38] states that the honey pots generate massive amounts of unique samples because "self-modifying malware is able to generate new polymorphic instances of itself with different message digest sums." The object of the peHash is to generate a hash function that is based on the structural data that can be used to cluster malware samples in order to identify those samples that are generated from the same polymorphic malware base.

The structural hash is generated using a combination of the image characteristics, target subsystem, stack commit size, and heap commit size [38]. The image characteristics refer to general settings like whether or not the PE is a DLL. The target subsystem identifies which subsystem will run the malware; such as GUI or CLI. The stack and heap commit sizes are both based on the initial allocation. Only those attributes that can be easily derived from the binary without detailed analysis are used in the hash to ensure that the computation complexity is low to allow for fast operation.

The authors of [38] show that by clustering based on this simple hashing function they are able correctly cluster samples from large data sets. The peHash clustering was shown to perform on par with current antivirus signature approaches.

The paper does concede the following, "It is, however, obvious that peHash cannot be used to cluster variants of malware families as for that the code structure itself would have to be analyzed." The code structure was assumed to be packed and or encrypted. Due to the previously mentioned performance constraint, peHash does not attempt to perform more in depth analysis, such as those that could be provided from a dynamic behavior analysis.

Chapter 4

APPROACH

As stated in the introduction, our objective is to provide a tool to empower the Security Analyst/Researcher. In order for the Analyst to better understand the interaction and evolution of malware, we propose the following system to detect and visualize potential relationships between malware samples. The system is guided and defined by the Analyst. The analyst is allowed to define, update, and or modify the parameters that define the relationships, the data sets used to determine relationships, and the layout and emphasis of the resulting graphs.

This tool allows the Analyst to quickly visualize,understand, and make conclusions concerning the relational information of many malware samples. [21] reported that 30,000 variants of the Beagle malware were recorded in a three month time during 2007. Due to the high volume of data that must be analyzed and considered when studying malware relationships it is imperative that the Analyst be given tools that are able to abstract and present the data for fast digestion, such as the visualization of relationships presented here in. The large number of variants bias the analysis towards a dynamic behavior analysis which is more suited to detection of variants than static analysis [38].

Because the nature of the behavior patterns of malware can shift between types and families the development of the tool has followed an Analyst-centric design in which the Analyst is able to manipulate and direct the detection and visualization of the sample sets. The system presented in Figure 4.1 and in the following sections seeks to fulfill these needs by allowing the Analyst to define the criteria for the

detected relationships while remaining flexible in the display and analysis of those relationships.

Figure 4.1: Approach Flow Diagram

4.1 Data Collection

In order to analyze malware to understand the relationships between samples, a set of samples must be made available. There are generally two methods for collecting a set of malware samples. The samples can either be collected using honeypot methods or collected via malware sharing services.

Honeypots are systems that are set up to lure external attacks. The Honeypot is set up to study the attacks and the tools by listing services with known vulnerabilities [8]. These honeypot systems are segmented into two types; Passive and Active [10]. The Active system is set to delibrately link with malicious URLs to seek out malware infections. The Passive systems wait for the malware to attack the computer. The Passive Honeypot may be configured with a high or low level of interaction. The low-interaction Honeypot runs as an emulated service. The setup varies based on the number of vulnerable services that are emulated. Since the services are emulated they are easy to deploy and scale [4]. The high-interaction Honeypot configures a real system. The systems are more responsive to the malware attack but are more difficult to deploy and maintain [4].

Malware sharing websites provide free sharing services. These websites collect submissions gathered from disk forensics, honeypots and other sources such as web crawlers. The sites include websites such as VXHeaven and VirusSign.

Malware samples have been obtained from two separate locations for testing and evaluation. These samples were obtained from 3rd party repositories. The first set of 151 samples was obtained from safegroup.pl. This is a Polish forum that maintains a repository of malware samples. A further 2000 samples were then obtained from VirusSign.com which contains are large number of samples that are freely available following the acquisition of an account.

4.2 Dynamic Analysis

The samples must first be analyzed in order to identify the features that will be used to determine the relationships. Dynamic analysis was chosen to prevent obfuscation from confounding the analysis, and to provide a more detailed set of features. The dynamic analysis of the malware is performed using the Cuckoo Sandbox. Cuckoo is an open-source project that started in 2010. The software has remained open-source to preserve its flexibility and creativity [14]. For example, the user is able to create additional analysis modules that can be injected into the Cuckoo framework to allow for customized analysis of the malware behavior.

Figure 4.2 shown below provides a brief overview of the layout of the Cuckoo system. The host machine is responsible for starting each of the virtual machines that are used to capture the malware's execution behavior. Once the sample has completed its analysis period on the virtual machine the results are reported back to the host machine to await action from the user. The virtual machines that were created as the analysis environment in this case are running an un-patched version of Windows XP.

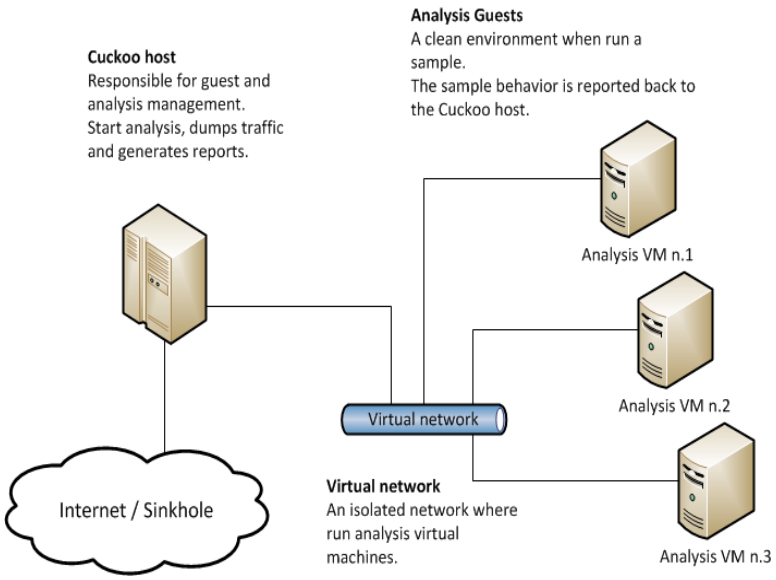

Figure 4.2: Cuckoo's System Arhitecture [14]

The Cuckoo sandbox is set up to monitor the changes made to the operating system running on the virtual machine. Cuckoo records all changes that are made to the file system, memory, network traffic, and periodic screenshots. During the execution the sandbox generates behavior logs. The behavior logs contain the win32 API calls that are executed on the virtual machine while the malware is executing. The memory dump that is retrieved from the virtual machine is processed using Volatility. Volatility is an open source collection of tools used in the analysis of memory files. In addition to the dynamic analysis performed by Cuckoo in this example, the sandbox can also be configured to perform basic static analysis on the malware samples. The static analysis includes extraction of hashes, signatures, and extracted strings.

The analysis reports have been configured to focus on the win32 API calls and file access, creation, and deletion. These attributes of the reported logs were chosen as the focus of the attribute matching because they are the most likely to appear in

the majority of the analysis logs of the malware samples in the test collection, while maintaining a descriptive representation of the malware's execution behavior.

The results returned from Cuckoo are presented in a variety of formats that are configurable by the user. The current setup returns the report in an html format that is simple to read and navigate for an overview of the observed behavior. The first portion of this report is presented below as an example in Figure 4.3.

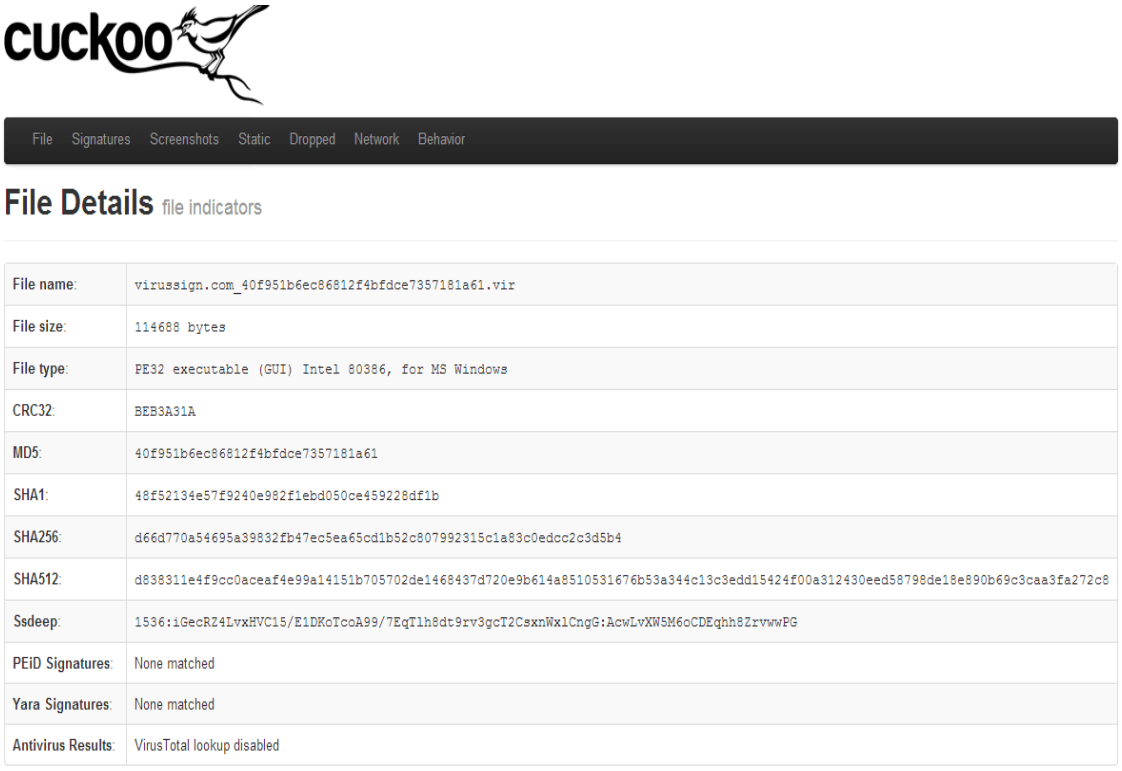

Figure 4.3: Sample Generated HTML Report from Cuckoo

In addition to the html overview, CSV log files, and json files are generated. The features extracted from the Cuckoo reports are retrieved from the json files.

Algorithm 1 Feature Extraction

```
Objective: Extract data from Cuckoo Reports:
while not end of log file do
    read category type
    if category is FileSystem then
        add file name to File Access list
    end if
    if category is System then
        add function name and attributes API call list
    end if
end while
```

The json logs are parsed to extract each element of the behavior logs. The log elements are labeled with categories that denote the type of behavior that is recorded from the dynamic analysis. The parsing focuses on extracting the calls to the "System" and "FileSystem". System types correspond to the function calls made by the malware sample. The FileSystem types are generated for all accesses to the files including opening, creation, modification, and deletion. After each json log is processed the key features that are used during the relationship analysis are logged into a csv file to conserve memory and provide for future auditing.

4.3 Feature Clustering

In order to allow the analysis to be continually flexible and defined by the Analyst the features that are compared are isolated based on the type of feature. The primary features that are used during the clustering process are the file accesses and win32 API calls. This allows the relationships between the samples to be defined solely on the interactions with the file system or the interactions with the API calls. Additionally, these features can be combined to define the relationship based on a weighted average of these features.

The files that are accessed by the malware are clustered based on the file name. The strength of the relationship between two malwares with respect to the file access

is determined by the total number of shared file accesses that are performed by the files. For examples, if two files each perform 10 file accesses during their execution and 8 of those are common to both malware samples then a strong correlation between the samples is assumed. The Analyst defines the minimum coverage threshold. All pairs of samples that are found to exceed this Analyst defined threshold are marked as having a relationship. The relationships are stored as an edge with an associated weight that can be filtered in real time during the visualization phase.

Algorithm 2 Feature Clustering: File Access

```
Objective: Identify similar File Accesses:
procedure MatchFileAccess(FileName)
    matchCount = 0
    for i = 0 to FileAccessListSize do
        if FileName = File at i then
            increment matchCount
        end if
    end for
    if matchCount>threshold then
        Return true
    else
        Return false
    end if
end procedure
for i = 0 to SampleListSize do
    Count shared file accesses MatchFileAccess Procedure
    if sharedFileAccessCount>Threshold then
        Add relation
    end if
end for
```

The file access list that is extracted for each malware sample is compared against all other samples in the set. The comparisons are made between each malware pair. The comparisons are not assumed to be commutative. This implies that a relationship may be identified between samples 1 and 2 but not between 2 and 1. Both cases are checked due to the varying length of the file access lists. Based on the type of malware that the Analyst is attempting to investigate the file access list may contain

the essential information required to define the relationship. One such example is the Sality virus. [31] found that the Sality virus created and deleted files in the Windows System Directory. By monitoring these file access patterns they were able to establish the Behavior pattern that identified the Sality virus.

The calls to the win32 API functions are grouped together into sequences of calls along with their parameters. The win32 API calls are clustered as groups using n sequential entries. This pattern is referred to as a system access pattern, and is similar to the use of n-gram as mentioned in [1]. The motivation behind using this system access pattern is to minimize the effect of anti-analysis techniques that were previously described such as polymorphism, obfuscation, and NOP code injection. As was also the approach with the file access, the strength of the correlation between two malwares is found by comparing the number of matching system access patterns performed by each malware.

Algorithm 3 Feature Clustering: Win32 API Calls

```
Objective: Identify similar Win32 API call sequences:
procedure NLengthPattern(CallList, n)
    accessPattern = null
    for i = 0 to n do
        extract API call from CallList
        append API call to accessPattern
    end for
    Return accessPattern
end procedure
procedure MatchAccesPatterns(nLengthAccessPattern)
    matchCount = 0
    Generate list of n length API call sequence: NLengthPattern procedure
    for i = 0 to AccessPatternLIstSize do
        testPattern = construct n length API call sequence: NLengthPattern pro-
cedure
        if nLengthAccessPattern = Pattern at i then
            increment matchCount
        end if
    end for
    if matchCount>threshold then
        Return true
    else
        Return false
    end if
end procedure
for i = 0 to SampleListSize do
    Count similar system access patterns: MatchAccesPatterns Procedure
    if matchedAccessPatternCount>Threshold then
        Add relation
    end if
end for
```

The same algorithm structure that was used to compare the file access is repeated when comparing the win32 API calls. The comparison is only modified to use the system access pattern instead of the single file name. Again as with the file access pattern the relationships are not assumed to be commutative.

The system access pattern was introduced following the evaluation of the more naive method employed in the file access pattern. Initial tests attempted to identify the coverage of the individual API calls between two samples. This approach produced

many false relationships. The sequential API calls in the system access pattern were found to be useful to both subvert the attempted obfuscation processes and reduce the occurrences of the false matches based on the API calls. This approach was seemingly vindicated by the results reported by [21], in which the authors found that software plagiarism detection methods were able to produce similar results to the malware clustering techniques presented by [7]. The Plagiarism detectors [2, 37, 39] mentioned by [21] use sequences of API calls that are captured during the program execution.

The correlation strength derived for both the file accesses and the system access patterns are evaluated to determine if a relationship exists between the two malware samples. The presence of a relationship is determined based on the analyst's configuration. The relationship can be defined by the file accesses, or the system access patterns, or both the file accesses and system access patterns. In either case the correlation is compared to the analyst defined threshold values. If both types of attributes are enabled then a weighting value is applied to determine the influence of both attribute sets. The weighting value is set using a vertical slider so that it is simple for the Analyst to visualize the level of influence that one feature type has over the other as seen in Figure 4.4.

Figure 4.4: Analyst Defined Feature Thresholds and Combining Weights

For each identified relationship an entry is added to the malware samples relative log file. This file holds a list of all malware samples with which the current sample has found a relation. In addition to the sample ID the strength of the relationship is stored. These files are later used to generate adjacency matrices for use in the visualization tool and graph analysis. The storage of these log files allow for quicker execution of the visualization and graph analysis portions of the approach.

The initial implementation was developed around a small test set of samples. In this case the extracted features and their derived relationships were held in memory throughout the execution of the program. However as the size of the sample set increased, memory quickly became a limiting factor. To address this issue the features and established relations for each sample are written to the hard drive. This forces the application to be more IO intensive but allows it to handle the data set of 2000+ samples that occupies roughly 80GB of storage space on the hard drive.

These file access and system call features that are currently used in the feature clustering to establish the malware relationships could be readily expanded to include DLL access, network queries, and Registry access. Additionally, mutexes generated by the executing sample can be extracted from the memory snapshot using the Volatility tool. [11] has concluded that the occurrence of known mutexes is sufficient to identify the presence of the Sality virus.

4.4 Visualization Tool

In order to visualize the relationships between the analyzed malware samples, a standard graph view was chosen. The graph view consists of nodes and edges. A node is created for each malware sample added to the tool. The edges are added between each node in which a relationship is detected. This allows the Analyst to easily view the number of samples to which a single malware connects. The graph is presented

with undirected edges to show associated relationships and to prevent the inference of a hierarchy of the relationship between the samples. The graph is derived from an adjacency matrix. An adjacency matrix contains n x n elements. The elements of the matrix are marked with a non zero edge weight where an edge exists between two nodes. A simple adjacency matrix is shown in fig:AdjacenyMatrix.

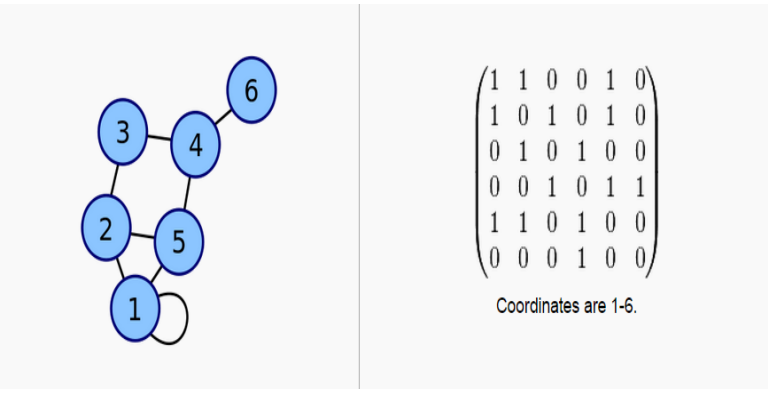

Figure 4.5: Adjaceny Matrix for a 6 Node Graph (Wikipedia)

The adjacency matrix is generated at run time by compiling the relative logs from the Feature Clustering phase. These logs are stored with each sample. The log file is parsed and an edge is added to the adjacency matrix for each sample found to have a connection to the current malware. In addition to using the matrix to generate the visual representation of the sample relations, the matrix can be analyzed when looking to extract patterns from the graph as discussed in the following section.

The visualization of the relationship has been enabled using two approaches. The first approach is integrated in the tool using VTK. The second exports the relationships in real-time to Gephi.

The Visualization Toolkit (VTK) [36] is an open-source library to produce 3D computer graphics. VTK is available from Kitware which also offers a C# wrapped version that was used in this application. Basic navigation has been enabled within the Visualization windows in which the user can zoom using the mouse's scroll wheel

and is able to translate the graph using the middle mouse button. The graphs presented using the VTK windows are limited in the interactivity and are static in that they cannot be further processed once rendered by the application. VTK is a general purpose library for visualization. This means that while VTK is able to render the graph, it is not its strongest feature. For this reason Gephi is the preferred visualization method.

Gephi is free graph visualization platform [13]. Through the use of an extending plugin the nodes and edges of the graph can be written to Gephi using the cURL library. All the features that are available in the VTK renderings are available in the Gephi based viewer. There are numerous advantages gained by adding support for a feature rich application such as Gephi. Among these advantages are the ability to dynamically reorganize the layout, change color schemes, perform filtering, and save the test data. Additionally, the graphs generated in Gephi are subjectively more visually appealing.

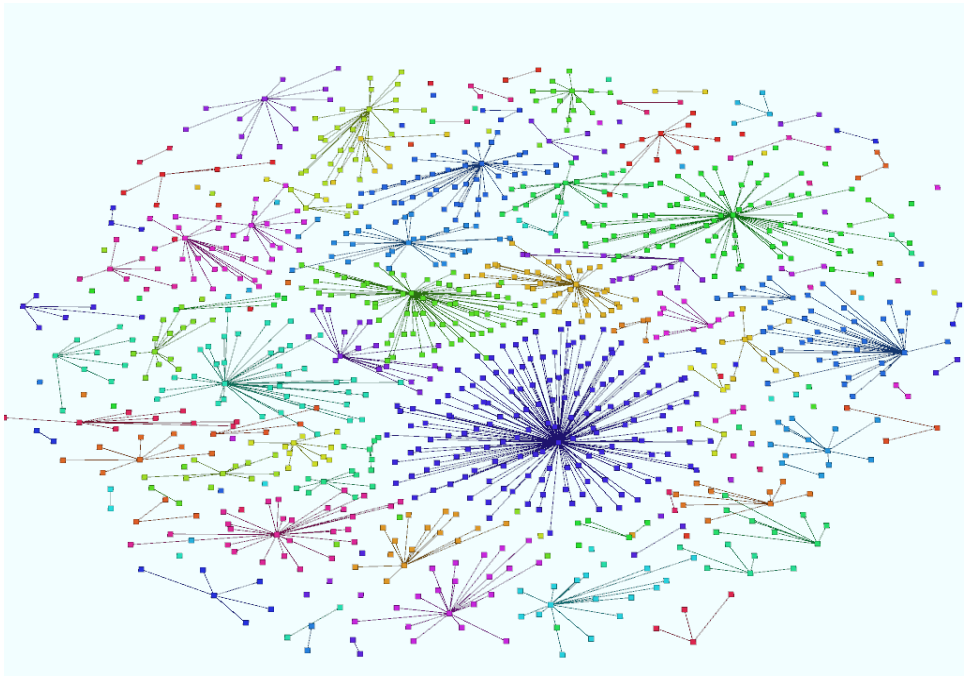

Figure 4.6: Sample Graph with Malwares Samples Color Coded by Family

The relationships between the samples are plotted in a graph to allow the Analyst to visualize how the relationships interact. [33] refers to this as exploring through visualization. She says "Visualization for exploring can be imprecise. It's useful when you're not exactly sure what the data has to tell you and you're trying to get a sense of the relationships and the patters contained within it for the first time." This is the primary motivation behind the development of a tool to visualize the relationships between malwares. In the case of the Analyst trying to understand a new batch of samples; the classification, intent, and countermeasures for each sample may be unknown. The goal of the visualization tool is to provide the Analyst with a fast method to form a hypothesis of the relationships that is at the same time flexible to allow the Analyst to explore how the relationships interact. [33] further states "The best data visualizations are ones that expose something new about the underlying patterns and relationships contained within the data. Understanding those relationships and being able to observe them is key to good decision making."

The speed at which the Analyst is able to get a grasp of the relationships, and patterns of relationships, that exist within the sample sets directly affects the volume and quality of analysis that the Analyst is able to perform. With regard to the value of visualizations [19] said "Finally, and perhaps most importantly, visualizations give us access to actionable insight. It is not until we have access to the knowledge within the data that we can actually act on it."

In order for the Analyst to be able to perceive the relationships drawn by the graph, it is necessary to arrange the graph in a manner that is visually pleasing. When a large number of samples are rendered in the visualization tool using a random placement of nodes and edges the effect is a noisy graph that is not easy to understand or analyze. [18] submits that graphs are more aesthetically pleasing when the number of edge crossings are minimized and the nodes are evenly distributed. Automatic

layout tools are applied to arrange the nodes and edges to enhance the aesthetics of the graph and aid the Analyst in the perception of the underlying relationship interactions.

Force-Directed layouts were found to generate graphs that are simpler to analyze, these layouts are computed using energy minimization to balance attractive and repulsive forces of the system. The attractive forces describe the strength of the force that draws two nodes together. This attraction is determined by the edge weights that are defined by our similarity metric. The repulsive force is used to push the nodes into the unoccupied spaces. The Yifan Hu and OpenOrd implementations of Force-Directed layouts have been employed to improve the analysis of the relationships found during the relationship generation. The Yifan Hu [18] force-directed layout performs edge collapsing to merge two adjacent nodes. When nodes are merged their edges are combined and given a cumulative weight. This allows the graph to be decluttered and more easily presented to the Analyst. The OpenOrd [24] force-directed layout allows for selective edge cutting to reduce the the occurrences of edge crossings. Figure 4.7 provides three examples to illustrate the effect of the Yifan Hu and OpenOrd Force-Directed layouts on a random node arrangement.

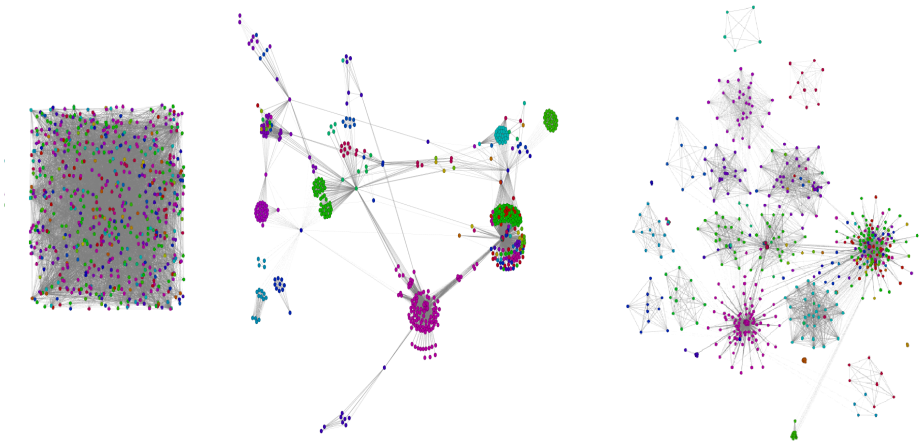

Figure 4.7: Random Node Arrangement, Yifan Hu Force-Directed Layout, OpenOrd Force-Directed Layout

Large sets of thousands of malware samples that have need to be processed daily contain an overwhelming amount of information pertaining to the relationships, attributes, and classifications of the samples. As stated above, the proper exploration of the malware relationships, and the understanding there of, that leads to good and timely decision and actions is facilitated through the use of the flexible visualization tool. This provides the Analyst with the optimal end result; the fast, accurate, actionable decisions that correctly handle the threats posed by the incessant advent of new malwares.

4.5 Analysis Examples

As previously mentioned, a significant goal of the visualization tool is to assist the Analyst in the task of sorting and analyzing the large volume of malware samples. One aspect of this goal is to help the Analyst prioritize which samples should receive a higher analysis priority. This is necessitated by the large number of samples that are received daily that require the analysis and attention of an Analyst. The examples

shown below are extracted from a test set consisting of approximately 300 malware samples.

The malware samples presented in Figure 4.8 show those samples that have an edge count of zero. The lack of the edges are indicative of samples that exhibit no relation to the other samples available in the analysis set. These samples were unique in both their file access patterns and system access patterns. This lack of correlation with other malwares may indicate a unique attack vector. Due to the lack of overlap with the other samples, and previously analyzed malwares, the need for further attention from the Analyst is high. The unique nature of the attack vector may be indicative of a potential danger that the current defense strategies will not overlap the intended target of the malware.

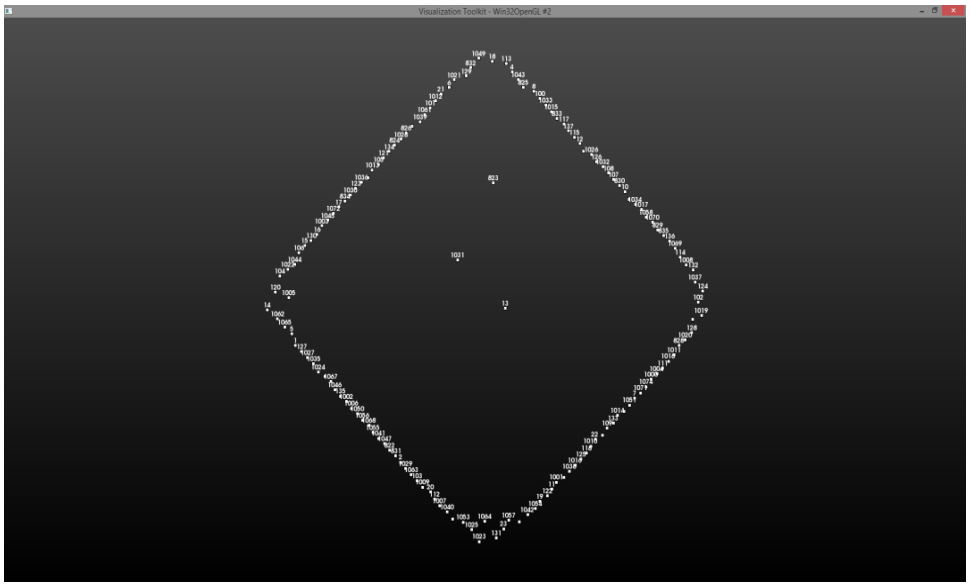

Figure 4.8: Samples That Show No Shared Characteristics with the Sample Set

The following cluster of malware samples shown in Figure 4.9 can be interpreted as an example on which an Analyst needs to focus attention based on the highly connected relationship it exhibits. This example shows a single malware that has performed the same file access as 98 other malware samples. This offers the analyst

two scenarios. In the first scenario none of the samples with which the central node are connected have been analyzed, in this case the Analyst would likely be able to deprioritize the analysis of this malware based on the knowledge that its attack vector is already encapsulated in the known attacks of the previously analyzed malwares. Alternatively, if none of the associated malwares have currently been analyzed then the Analyst would be able to prioritize the analysis of the central node due to the fact that by quarantining the files that are accessed by one malware the effect of the attacks of many could also be neutralized.

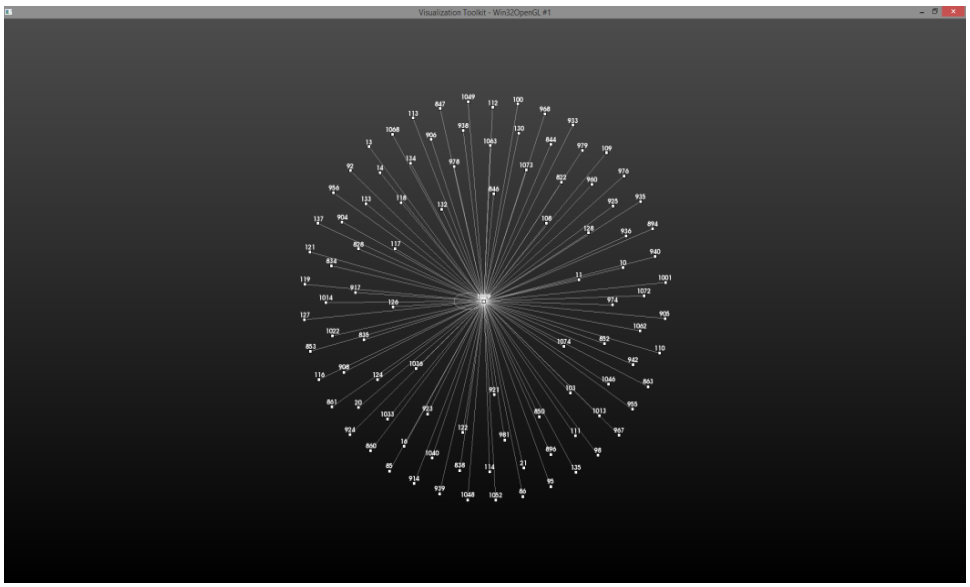

Figure 4.9: Highly Connected Sample

The relationship presented in Figure 4.10 shows a Clique Graph. A Clique is a pattern in which every vertex pair is connected. This special case is indicative of a set of malwares that are highly correlated. This case is of particular interest because it allows for the extraction of highly correlated groups of malwares from the superset. The Clique is a known pattern that can be extracted from graphs. Even though the problem of clique identification is known to be NP-complete, the extraction of these

cliques merits the effort due to the fact that the subsets may be difficult to see or interpret when contained in the larger data set.

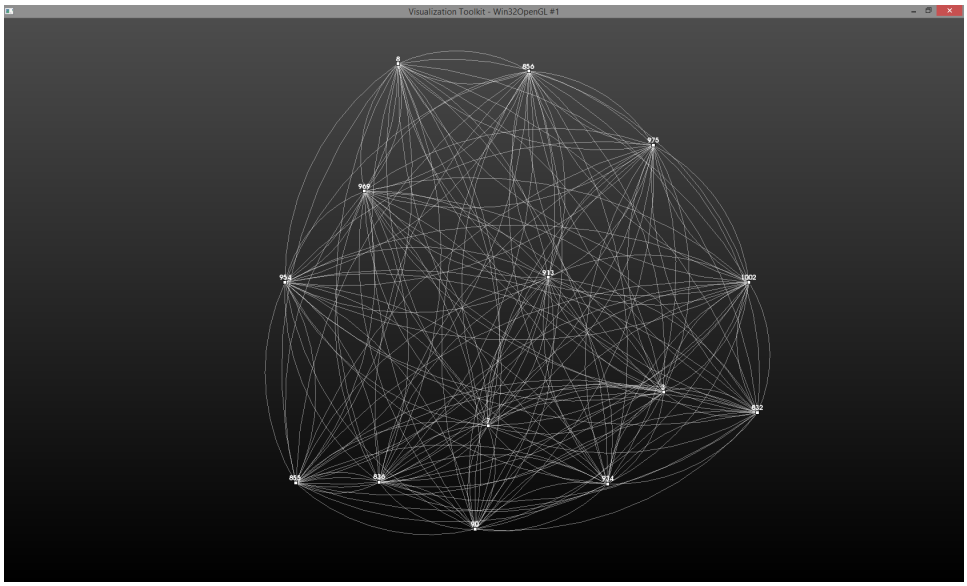

Figure 4.10: Large Clique Relationship Example

Among samples that were found to be members of the clique in Figure 4.10 the following names were identified using Kapersky and Microsoft:

Sample ID	Kapersky Name	Microsoft Name
975	HEUR.Trojan.Win32.Generic	Trojan.Win32/Rimecud.A
969	Trojan.Win32.Agent.adebm	PWS.Win32/Uospry.A
913	HEUR.Trojan.Win32.Generic	Backdoor.Wind32.Bifrose.1Z
1002	Trojan.Win32.Agent.adebm	PWS.Win32/Uospry.A

Table 4.1: Identified Malwares from Clique Graph

In this example, there are two instance where a malware is identified that has two different static signatures. Additionally, the files that are accessed by these samples are shown to be similar which was not apparent based on the report tables generated by automated tools such as the one shown in Figure 4.3.

4.6 Types of Relationship Graphs Derived from Clusters

Check boxes have been added to the applications control dialog box to allow the analyst to define which types of relationship graphs should be displayed in the visualizations windows, as seen below in Figure 4.11. Additionally, in order to speed up repeated analysis of data sets the graphs are written to a csv file that can be reloaded. Once reloaded, a new set of graphs can be generated based on the chosen set of relationship graph types selected by the analyst.

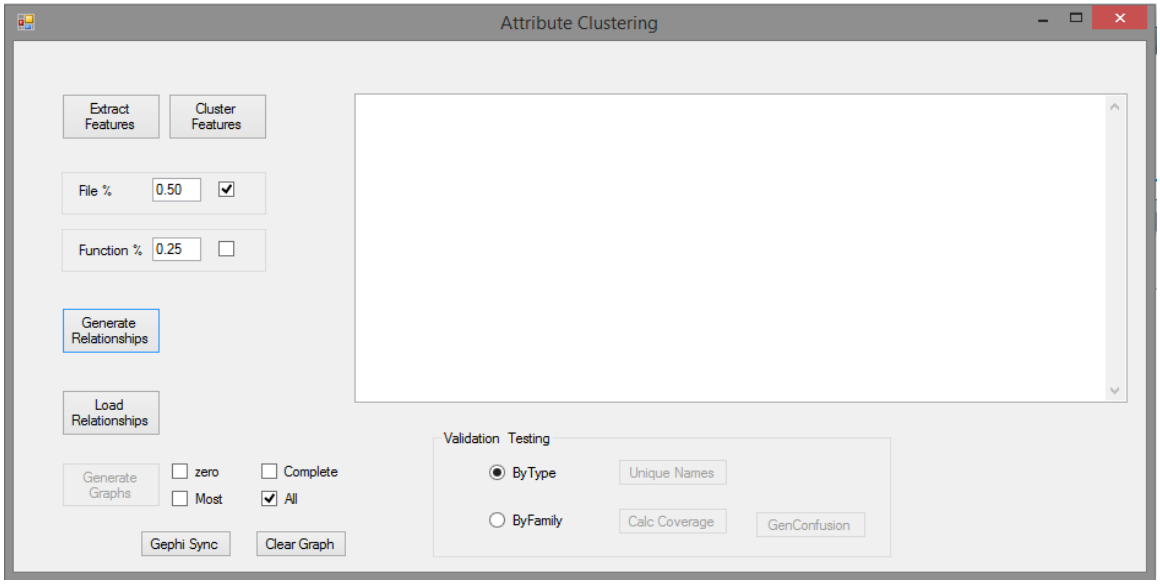

Figure 4.11: Application Dialog Box

The following connection types can be extracted from the graph relations using the applications dialog box controls:

4.6.1 Zero Connection

In this case the graph shows the set of samples that exhibit zero correlations with the other malware samples in the set. This type of relationship may be indicative of a new malware sample. In this case the attack vector is unique in that it does

not overlap with any of the previously known samples in the analysis set. Based on the other malwares that were used in the relationship set, this could indicate that additional attention is required of the Analyst.

4.6.2 Clique

In this relationship each node of the sub graph is connected to all other nodes in the sub graph. This type of relationship is consistent with samples that are the same or similar in nature. Samples which have been modified to subvert known hash signatures (variants) would likely manifest themselves as completely connected graphs.

Algorithm 4 Clique Extraction

```
Objective: Identify fully connected relationships within the network graph.
Place all nodes in the graph with n edges into the clique subset
Identify node in clique subset with least matches
while Disconnected node exists do
    remove disconnected node from clique subset.
    Identify node in subset with least matches.
end while
```

The cliques are identified by filtering the relationships to remove all nodes that do not meet the minimum number of connected edges. The remaining nodes are then exhaustively scanned to remove nodes that are not connected to the remaining nodes in the set. The set is pruned until all non fully connected nodes are removed from the graph.

4.6.3 Most Connected

This graph displays the node that has the most connections with other samples in the analysis set. This can be used to identify samples that have the low or high analysis priority based on the other samples with which it is associated. The low

case would occur when the other samples have been analyzed and the high priority case would be flagged when the majority of the connected samples have not been previously analyzed.

Algorithm 5 Most Connected

Objective: Identify the node with the most identified relationships.
Place all nodes in the graph with n edges into the clique subset
$maxEdgeCount = 0$
for $i = 0$ to $nodeCount$ **do**
  if $nodeEdgeCount > maxEdgeCount$ **then**
    $maxEdgeCount = nodeEdgeCount$
  end if
end for
Extract the node with the most connected edges.

To extract the node with the most edges the full set of nodes is traversed. The node that is found to have the highest number of edges is extracted. In the case of multiple nodes having the max number of edges all nodes are drawn.

4.6.4 All

This is the default mode for the graph relationships. This mode simply displays all of the samples with their relationship connections from analysis set. The Analyst is free to navigate and browse but no special attention is called to any relationship in the set.

Additional relationships that could be further explored from the graphs include Trees, Embeddedness, Centrality, and Dispersion.

4.7 Applications of Relationships

The ability to select the type of relationship from the analysis of the full undirected graph is provided to help the Analyst select subsets of data on which to focus attention. A large volume of data can be represented using the undirected graph. While

this has the advantage of giving an Analyst an overview of the general relationships in the data set, it can also serve as a detriment by hiding the desired information and patterns beneath the rendering of the graph.

Following a perusal of the overview of the full graph, the user may choose to focus attention only on those sample that show a strong relational bond with other samples. In this case the user may select to extract all the cliques of a given strength from the graph.

The ultimate goal of this approach is to develop a tool that could be used to aid a user in the analysis of many malware samples by helping the Analyst to allocate time. The ability to target the desired relationships and extract them from a larger set allows the user to more quickly find the samples required for the analysis, thereby conserving time that can be used to more productively analyze additional malware samples.

Chapter 5

EVALUATION

5.1 Evaluation Approach

The goal of the application is to aid the Analyst in the processing and prioritization of a large amount of malware samples. This is done by allowing an Analyst to quickly view the relationships exhibited by a malware with other samples. By viewing the relationships the Analyst is free to execute the task of analysis and countermeasures with the informed view of how a sample or groups of samples are connected.

In order to evaluate the approach, the relationships identified via the clustering of the extracted execution attributes must be validated to show that they are able to indicate when two or more similar malwares are present and correlated. The relationship of the malwares is treated as a binary classification, in which each malware in the sample set is either related or not related to the next. Treating the relationships as binary decisions allow for the generation of a confusion matrix to evaluate the correctness of the relationships as a whole across the sample set.

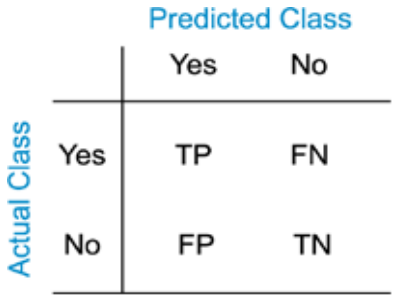

Figure 5.1: 2x2 Confusion Matrix

The confusion matrix consists of four classification: True Positive (TP), False Negative (FN), False Positive (FP), and True Negative (TN). The predicted class in the vertical columns of the matrix is derived from the relationships produced by our attribute clustering approach. The actual class in the horizontal columns of the matrix are derived from the CARO formatted name generated by Microsoft, as reported by VirusTotal. The Computer Antivirus Research Organization (CARO) name is a naming scheme which consists of multiple components, the two primary components considered in this analysis are the Type and the Family.

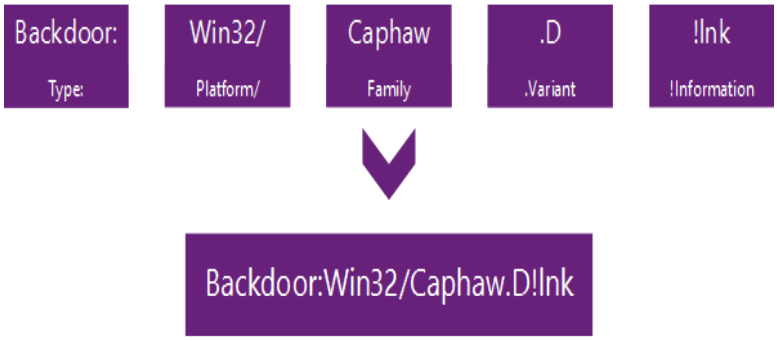

Figure 5.2: Example Naming Scheme for Caphaw from Microsoft [26]

The confusion matrix is populated using the actual and predicted classifications for the relationships in the sample set. All possible combinations of the samples in the test set are compared using the CARO name and edge state. The True Positive is identified when two samples are reported to have the same CARO name and are found to have a relationship established by our clustering analysis. The False Positive case is identified when the approach inaccurately assigns a relationship but the CARO names are different. The False Negative case is generated when the approach fails to identify a relationship yet the CARO names of the samples are the same. The True Negative case is found when the approach correctly determines no relationship is present and this is confirmed by the CARO names being different.

The accumulated results of the confusion matrix are used to compute the Sensitivity and the Specificity. Sensitivity gives a measure of the number of relationships that have been correctly classified by the approach. Sensitivity is computed as follows:

$$Sensitivity = TP/(TP + FN) \tag{5.1}$$

Whereas the Sensitivity focuses on the positive identification cases, the Specificity is used to measure the negative cases. The Specificity is the inverse Sensitivity in that it reports the percentage of the cases that are not related that were reported as such.

The Sensitivity and Specificity are plotted using an ROC space plot. The Receiver Operating Characteristic (ROC) plot is used to show how the Sensitivity and Specificity vary based on the thresholds of the experiments. The Sensitivity is plotted along the y axis also known as the True Positive Rate (TPR). The Specificity is presented along the x axis and as the False Positive Rate (FPR), which is computed as FPR = 1-Specificity. Points that are plotted along the diagonal of the ROC space show a poor classification result. The diagonal of the space represents the results of a random guess.

The Sensitivity and Specificity are measures that are applied to the relationships that are generated for the test set as a whole. The coverage per family and type of malware is also computed as a percentage based on the number of the known samples from each family and type that are found to be related by the approach.

5.2 Evaluation Results

The relationships identified by the approach are evaluated using both the type and the family as identified by the Microsoft name. The test set consists of 1060 samples. In this set there are known to be 164 separate families from 17 different types of malware.

5.2.1 *Evaluation of Relationships by Type*

The results are evaluated across three different types of experiments. The first experiment considers only the similarity between the malwares based on their file accesses. The second experiment considers the similarities based solely on the Win32 API function calls. The third experiment identifies the relationship using both the API function calls and the file accesses.

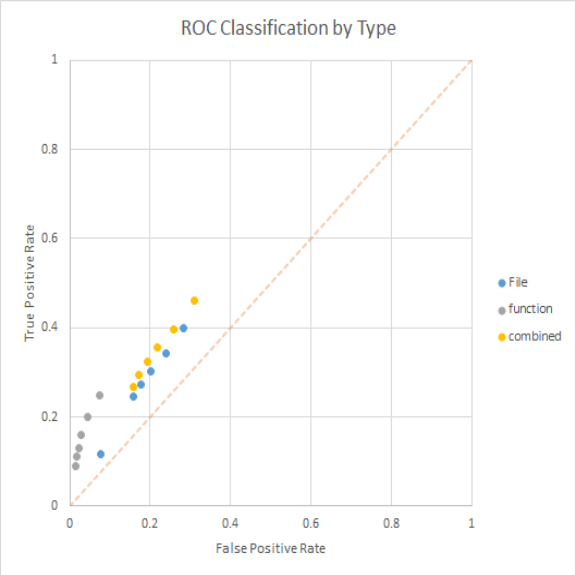

Figure 5.3: ROC Space for Evaluation by Type

The results in the ROC space are presented for each of the three experiment types. Each experiment is conducted over 5 separate thresholds to evaluate the varying degree of the true relationships identified. When evaluating the relationships based on the type, the Sensitivity shows that in all cases the relationships were able to correctly classify the type of malware. However, the strength of the Sensitivity is low showing that the actual proportion of the correctly identified relationships is weak. The varying of the threshold used to determine a relationship in the classifier has little impact on the strength of the classification. This is observed by the fact that

the distance between each point and the diagonal line only changes slightly as the threshold level is modified.

The objective of the approach is primarily focused on the identification of the malwares with similar features that are defined by the Analyst. The strength of the Sensitivity in the ROC serves to show that this similarities can also be used to identify a rough approximation of the type of malware that is being analyzed.

In addition to the values shown in the ROC space, the values in the confusion matrix can be used to compute the Negative Predictive Value (NPV). The NPV shows the proportion of the true negatives that are identified as unrelated. In all cases the NPV was about 85%, showing that when a malware pair is determined to be unrelated it is done so with a high confidence.

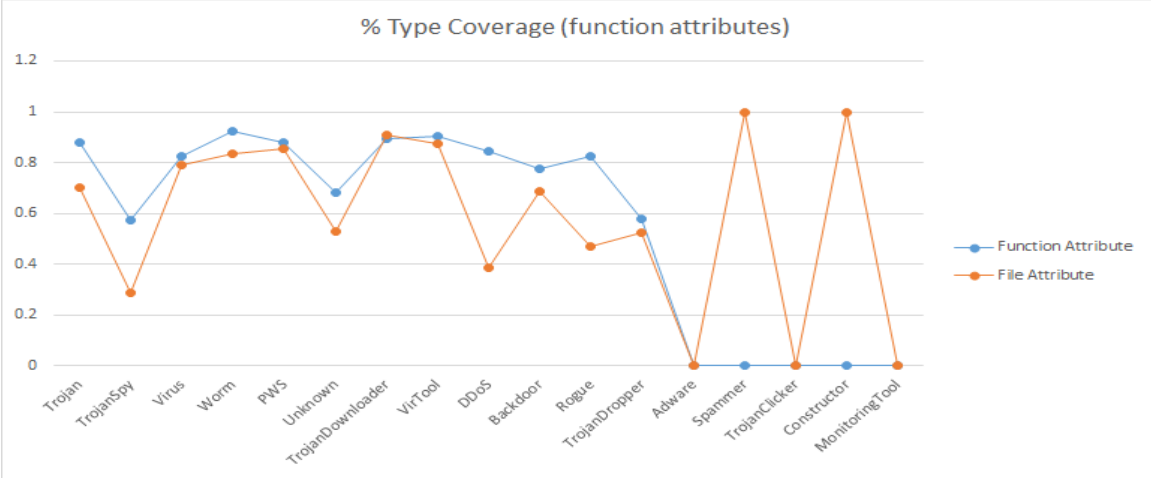

Figure 5.4: Percent of Sample by Types Marked as Related

Additionally, the coverage of the individual groups can be observed by calculating the percentage of each group that were found to be related. The above graph illustrates that coverage using both the file and the function attributes to identify similar malwares. From this it can be seen that the function attributes performed better than the file attributes when trying to identify malwares of a similar type.

5.2.2 *Evaluation of Relationships by Family*

The same approach from the previous section is taken to evaluate the relationships based on the malware family using three different types of experiments. The first experiment considers only the similarity between the malwares based on their file accesses. The second experiment considers the similarities based solely on the Win32 API function calls. The third experiment identifies the relationship using both the API function calls and the file accesses.

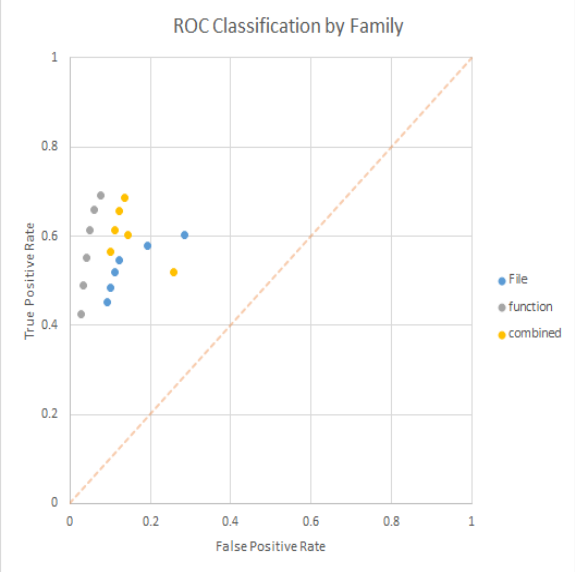

Figure 5.5: ROC Space for Evaluation by Family

The experiments from the type category were repeated using the malware family to evaluate the relationship. The Sensitivity scores increase as compared to those reported when the relationships are evaluated by the type. The scores reported when evaluating based on the malware family show that the relationships produced by our approach are able to more accurately discriminate the family of malware.

In addition to the improved Sensitivity rates reported while evaluating based on the malware family, the NPV rates also improve to over 95% in all cases. Again, as

was seen in the evaluation based on the type, those sample pairs that are found to not be related are most likely accurately classified as such.

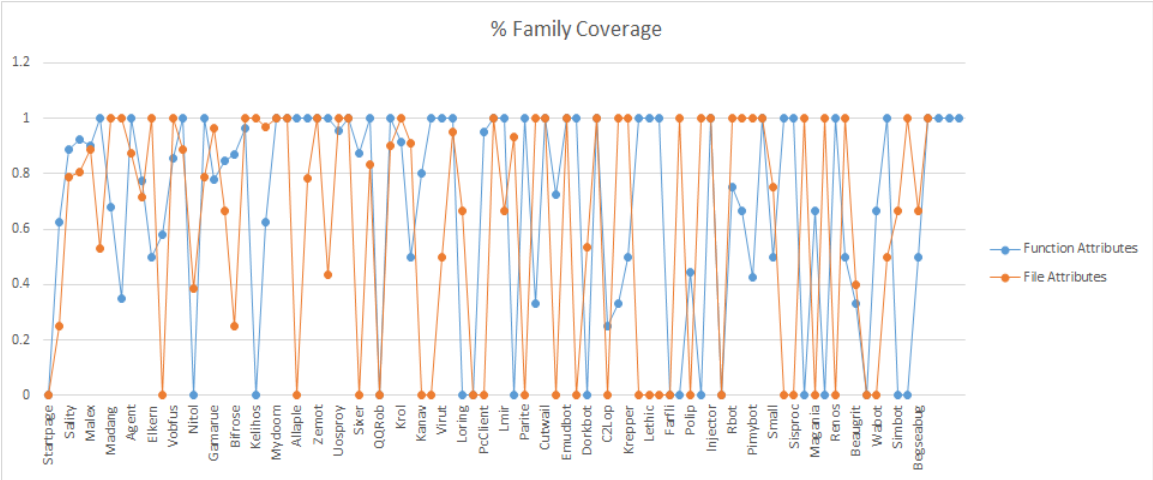

Figure 5.6: Percent of Samples by Family Marked as Related

The coverage of the families in the set shows a high degree of success. However, in many cases the coverage drops to 0%. This is seen in many of the families that are underrepresented in the test set. For example, Rovnix in the figure above, reports a 0% coverage of the detected relationships. Rovnix was only found twice in the sample set. As such, a near miss in identifying the relationship of a single couple prevented the set of Rovnix samples from being linked, where as with other families there exists the possibility of more easily identified relationships through other members ultimately linking the family of samples together via a chain.

Chapter 6

FUTURE WORK

6.1 Limitations

While a motivating factor behind our approach is to present the Analyst with the option to prevent spending time in analysis of redundant samples, the implementation itself has not been designed with execution speed as a primary goal. Other approaches such as [17, 6] were designed to be fast and scalable to be used on large sets of data. The tool created in this implementation requires approximately 720 seconds to perform the relationship analysis on 1000 samples.

A further limitation of the current system depends on the fact that all the samples in the database are malwares that can be executed on Microsoft Windows systems. No testing has been done to attempt to integrate samples from heterogeneous data sets.

Additionally, this phase of the project focused solely on establishing the relationships based on the file accesses and the Win32 API calls. Focusing solely on these two sets of features of the dynamic analysis of the malware samples discards much of the valuable data collected concerning the DLL accessess, Registry modifications, Network queries, and Mutex allocations which have been shown by [11] to contain unique identifying features to malware families.

Furthermore, the current dynamic analysis has only allowed for a single pass per sample. No efforts have been made to verify that the dynamic analysis will generate the same feature sets for each sample if they are allowed to execute multiple times. Indeed [27] indicates that the analysis will record new and unique features based on

the varying conditions found during the dynamic analysis, even though the analysis is performed using a clean virtual machine. Likewise an instance of a sample has not been purposely replicated and matched against itself repeatedly to verify the relationship is consistently identified.

6.2 Recommended Improvements

The current mechanism used during the identification and clustering of the relationships between the malware samples utilizes a series of linked lists. For example, when analyzing the system calls a list is created for each function to store its arguments. Additional lists are then constructed for all the files that are accessed by the samples. The result of loading the analysis information into the memory structures is an excessive use of memory during the loading and analysis phases.

A more appropriate and scalable approach would be to load the features for each sample into a database. This would allow for a more efficient means of storing the information than using the series of linked lists that is currently employed. Additionally, the ability to do queries against the database would allow for a simplification of the attribute clustering steps. Transitioning from the current method to a database approach would likely deliver an improved performance, a reduction in memory consumption, and expandability.

In order to allow the tool to continue to be useful it needs to have the ability to accept new definitions for the attribute clustering and relationship extraction. In order to facilitate this the application needs to allow the user to create plugins that can be added to the tool. This would allow the Analyst to adapt the tool to the specific needs of the analysis. Allowing for this kind of flexibility and modularity would expand the usefulness and life cycle of the visualization tool.

The current evaluation was done on a limited set of malware samples; approximately 2000 were used as of this writing. Essentially the task of the visualization tool is to perform a basic method of data mining by extracting, clustering, and visualizing the relationship between malware samples attributes. Additional data samples would allow for more in-depth analysis of these relations. Additionally, the presence of many more relations are likely not visible due to their lack of connecting relations that could be observed had additional samples been used in the set. Due to the structure of the application and its heavy memory footprint, the current data set was not actively expanded.

Validation of the current and future relationship patterns could be expedited if the dataset malware samples contained more prior knowledge indicating which samples were in fact variants of each other. The ground truth would allow for a rapid testing of current and new attribute clustering metrics and relational patterns. The current dataset was collected from public databases without prior analysis and as such no knowledge concerning existing variations and malicious intentions were known or assumed.

An additional analysis and visualization that would help the Analyst to understand the evolution of the malware samples would be the construction of a phylogenetic tree. The tree would be comprised of related samples with ancestry determined by the first recorded observation date reported by AV scanners.

A further characteristic that would be useful in the analysis of malware sets would be the context of the sample collection. It could be very illuminating to visualize the composition of collected sets based on the type of institution and geographic location.

Chapter 7

CONCLUSIONS

The objective of the approach that has been presented is to provide a malware Analyst an additional tool that can be used to aid in the identification of relationship between malware samples. The tool does not attempt to classify or identify malware based on the attributes or effects. Rather, the ideal outcome is to provide the Analyst with an easy to identify set of malware samples that share common features. With that in mind the evaluation of the relationships showed a good correlation between the related samples and the known malware families.

Additionally, the evaluation of the relationships showed that the metrics provided a very strong negative predictive value. This is of greater importance to the performance of the tool than the ability to classify a sample into a particular family or type, because it allows the Analyst insight into which malwares may be related outside of the classified families and minimizes the presentation of erroneous relationships.

Similar approaches mentioned in Chapter 3 include SMIT and peHash. Both SMIT and peHash attempt to cluster together similar malware samples. Both of these approaches rely on a static analysis of the malware sample to extract the features that are used to determine the similarity of two samples. A common problem with relying solely on static analysis is that it is more difficult to detect similar malwares samples due to the many obfuscating and other protecting measures that are applied to the malwares as stated in Chapter 2. The authors of peHash acknowledged this difficulty by stating "peHash cannot be used to cluster variants of malware family as for that the code structure itself would have to be analyzed"[38]. SMIT concedes that the similarities that are detected by their approach rely on the function call graph which

attempts to approximate the real time behavior of the sample. Both these approaches acknowledge that more in-depth knowledge of the behavior and structure of a malware sample can be derived from using dynamic behavior analysis. The disadvantage to using the dynamic analysis is time required to execute and analyze the sample in real-time.

Our approach contributes to those previous approaches, such as SMIT and peHash, by utilizing the features extracted from the dynamic analysis of the malware's behavior. This analysis allows our approach to observe patterns of behavior that are obscured from the static analysis by the anti-analysis techniques employed by the authors of the malwares. Rather than replacing the results produced by the static analysis, the relationships identified by the dynamic analysis may be used in tandem to allow for deeper insights by the Analyst. Additionally, the tool presents the Analyst with a graph to visualize all the extracted relationships. The Analyst is also free to manipulate that graph data to extract sub graphs that relate to the desired analysis scenario.

As our approach is based solely on the features and characteristics that are extracted by the dynamic analysis. It is unique in comparison to the other approaches presented in Chapter 3. Other approaches not mentioned in Chapter 3 may also utilize the features extracted from the dynamic analysis of the malware such as [6], however these approaches first attempt to establish an abstraction of the dynamic analysis features to reduce the feature space, this is referred to as a "Behavior Profile" by [6]. We further add to the "Behavior Profile" through the use of the system access patters which were shown to perform nearly as well as the "Behavior Profile" by [21]. These other approaches typically are acting in a classification mode and rely on a pre-trained classifier or rigid structure. The tool presented here has been configured to remain flexible so that the Analyst can control the relationship definitions at

run-time, thereby allowing it to be defined based on the use case presented by each analysis scenario.

To our knowledge the approach presented here is novel in the following aspect. The system preserves the full feature set provided by the dynamic analysis and it is available to be used during the relationship analysis. The system presents a flexible user defined approach to the relationship definition that is able to change with each execution of the application. Our approach doesn't seek to classify all related samples belonging to a given family or variant, but rather, to provide a visualization tool to allow the Analyst to form conclusions from these relationships that may not be apparent given a simple classification. The visualization is able to be manipulated using automatic layout adjustment tools, user defined edges, drag-drop, and group labeling to allow the Analyst to continually define and redefine the viewpoint through which the relationships are examined. Furthermore, the system provides graph extraction tools to allow the Analyst to extract sub-graphs that are found to correlate to desired related groups.

In summary, the system presented here fulfills its objectives through the following actions: malware features are extracted from reports generated using dynamic analysis, flexible attribute clustering is defined by the Analyst at runtime, visualization of sample relationships via an undirected graph, and user defined relationship extraction through graph patterns. The graph based visualization tool has been shown to be able to extract basic relationships from a set of unclassified malware samples. Using this information the Analyst is able to approach the set of samples with additional insight into how to best allocate analysis resources for maximum effectiveness. This knowledge allows the Analyst to avoid the sinking of additional time in the analysis of a set of variants of a single malware by identifying cliques of samples in the graph,

or to identify unique samples with no known correlation to the other samples in the database through the identification of disconnected nodes.

REFERENCES

[1] Abou-assaleh, T., Cercone, N., and Sweidan, R. (2004). R.: Detection of new malicious code using n-grams signatures. In *In: Proc. Second Annual Conference on Privacy, Security and Trust*, pages 13–15.

[2] Aiken, A. (1994). Moss: a system for detecting software plagiarism. `http://http://theory.stanford.edu/~aiken/moss/`.

[3] Anubis (2014). Anubis - malware analsysis for unknown binaries. `https://anubis.iseclab.org/`.

[4] Baecher, P., Koetter, M., Dornseif, M., and Freiling, F. (2006). The nepenthes platform: An efficient approach to collect malware. In *In Proceedings of the 9 th International Symposium on Recent Advances in Intrusion Detection (RAID*, pages 165–184. Springer.

[5] Barraco, L. (2013). What are the most common types of malware. `https://www.alienvault.com/blogs/security-essentials/what-are-the-most-common-types-of-malware`.

[6] Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., and Kirda, E. (2009). Scalable, behavior-based malware clustering.

[7] Bayer, U., Kruegel, C., and Kirda, E. (2006). Ttanalyze: A tool for analyzing malware.

[8] Cavalca, D. and Goldoni, E. (2008). Hive: an open infrastructure for malware collection and analysis. In *IN PROCEEDINGS OF THE 1ST WORKSHOP ON OPEN SOURCE SOFTWARE FOR COMPUTER AND NETWORK FORENSICS*, pages 23–34.

[9] CWSandbox (2009). Cwsandbox - automated online malware analysis. `http://www.rarst.net/web/cwsandbox/`.

[10] dar Lin, Y., Lee, C.-Y., Wu, Y.-S., Ho, P.-H., yu Wang, F., and Tsai, Y.-L. (2014). Active versus passive malware collection. *Computer*, 47(4):59–65.

[11] Falliere, N. (2011). Sality: Story of a peer-to-peer viral network. `http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/sality_peer_to_peer_viral_network.pdf`.

[12] French, D. (2011). Fuzzy hashing techniques in applied malware analysis. `http://blog.sei.cmu.edu/post.cfm/fuzzy-hashing-techniques-in-applied-malware-analysis`.

[13] Gephi (2014). Gephi. `http://gephi.github.io/`.

[14] Guarnieri, C. (2013). Mo malware mo problems. `https://media.blackhat.com/us-13/US-13-Bremer-Mo-Malware-Mo-Problems-Cuckoo-Sandbox-Slides.pdf`.

[15] Hex-Rays (2012). Flirt. https://www.hex-rays.com/products/ida/tech/flirt/index.shtml.

[16] Hosmer, C. (2008). Polymorphic and metamorphic malware. https://www.blackhat.com/presentations/bh-usa-08/Hosmer/BH_US_08_Hosmer_Polymorphic_Malware.pdf.

[17] Hu, X. (2011). *Large-Scale Malware Analysis, Detection, and Signature Generation*. PhD thesis, University of Michigan.

[18] Hu, Y. F. (2005). Efficient and high quality force-directed graph drawing. *The Mathematica Journal*, 10:37–71.

[19] Iliinsky, N. (2012). Why is data visualization so hot. http://blog.visual.ly/why-is-data-visualization-so-hot/.

[20] Kornblum, J. (2011). Ssdeep. http://ssdeep.sourceforge.net/.

[21] Li, P., Liu, L., and Reiter, M. K. (2007). On challenges in evaluating malware clustering.

[22] Linn, C. and Debray, S. (2003). Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 290–299, New York, NY, USA. ACM.

[23] Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297.

[24] Martin, S., Brown, W. M., Klavans, R., and Boyack, K. W. (2011). OpenOrd: an open-source toolbox for large graph layout. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7868 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 6.

[25] McAfee (2013). Infographic: The state of malware 2013. Technical report.

[26] Microsoft (2014). Naming malware. http://www.microsoft.com/security/portal/mmpc/shared/malwarenaming.aspx.

[27] Moser, A., Kruegel, C., and Kirda, E. (2007a). Exploring multiple execution paths for malware analysis. In *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 231–245.

[28] Moser, A., Kruegel, C., and Kirda, E. (2007b). Limits of static analysis for malware detection.

[29] Ollmann, G. (2009). Serial variant evasion tactics. Technical report.

[30] Rad, B. B., Masrom, M., and Ibrahim, S. (2012). Camouflage in malware: from encryption to metamorphism. In *IJCSNS International Journal of Computer Science and Network Security*, pages 74–83.

[31] Rieck, K., Holz, T., Willems, C., and Dssel, P. (2008). Learning and classification of malware behavior. In *In Fifth Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 08*.

[32] Rieck, K., Trinius, P., Willems, C., and Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *J. Comput. Secur.*, 19(4):639–668.

[33] Steele, J. (2012). Why data visualization matters. .

[34] Symantec.com (2010). Spyeye bot versus zeus bot. `http://www.symantec.com/connect/blogs/spyeye-bot-versus-zeus-bot`.

[35] Truman (2014). Truman. `http://www.secureworks.com/cyber-threat-intelligence/tools/truman/`.

[36] VTK (2012). Vtk. `http://www.vtk.org`.

[37] Whale, G. (1990). Identification of program similarity in large populations. *Comput. J.*, 33(2):140–146.

[38] Wicherski, G. (2009). pehash: A novel approach to fast malware clustering. In Lee, W., editor, *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '09, Boston, MA, USA, April 21, 2009*. USENIX Association.

[39] Wise, M. J. (1992). Detection of similarities in student programs: Yap'ing may be preferable to plague'ing. *SIGCSE Bull.*, 24(1):268–271.