Proposed ApplicableFramework for Extracting Rootkits Features and Clustering through Dynamic Analysis for Incident Handling Systems

Cho Cho San, Mie Mie Su Thwin
University of Computer Studies, Yangon.
chochosan@ucsy.edu.mm, miemiesuthwinster@gmail.com

Abstract

*Today's threats have become complex multi-*module *systems using sophisticated techniques to target and attack vulnerable systems. The use of rootkits and rootkit technologies in malware and cybercrime is increasing. To remain undetected, malware creators incorporate rootkit components to maximize their stealth capabilities. The main reason to develop this research is the longer the malware can remain undetected on a compromised machine, the more the cybercriminal can profit. Therefore, the proposed system will focus on analyzing the kernel and user level rootkits based on Window operating system with Cuckoo sandbox. This system performs automated and manual analysis for ensuring the important of their characteristics. The objectives are to identify the rootkits based on their natures and complexity, and to propose feature extraction algorithm for improving the detection model.Effective MalwareFeature Extraction Algorithm(EMFEA) is proposed in this framework for detecting the future malware in Incident Handling Systems. Moreover, the proposed system categorizes the rootkits based on their relevant and prominent features by using Hierarchical Clustering algorithm in WEKA.*

Keywords: Rootkit, feature extraction, Hierarchical Clustering

1. Introduction

The term rootkit originates from the composition of the individual terms root, referring to the highest privilege of access that can be obtained in a traditional Unix-based operating system, and kit, referring to a set of programs that are designed to exploit a target system, gain root access and then maintain it without tripping any alarms [1].

Rootkits differ in a few ways from other malware such as viruses or worms. Rootkit is not a self-propagating code. It requires typically three snippets of code known as dropper, loader and rootkit. Rootkit technologies have been used in malware to conceal the malicious behavior. Deploying rootkit technology buries the malware deep within the computer making it much more difficult to detect and complex to remove [2].

There are two main reasons to conduct research in the area of Window rootkits:

1. Modern rootkits are highly obfuscated to confuse forensics and frustrate reverse engineering, incorporate covert channel, encrypted files, and a modular design that allows different types of malware from different designers to work together by exporting malicious APIs and syscalls [3].

2. Most research avoid the major challenge of dealing with encrypted or packed malicious samples.

Therefore, analyzing the nature of rootkits is very important because they are very effective tools for hiding malicious software, and attackers will continue to use them as long as there is profit to be made. Moreover, the proposed system expects to identify the rootkits depend on their obfuscation techniques.

The rest of the paper is organized as follows:

- Section 2summarizes the related work,
- Section 3 discusses technical background of Rootkits,
- Section 4 describes our proposed system,
- Section 5 provides the contribution,
- Section 6 highlights system experiment setupand
- Section 7shows the experimental results respectively,
- Finally, section 8 we provide a conclusion and the outline of our future research plans.

2. Related Works

Rootkits primarily can be found into three main types, namely patching (replacement of code sections), hooking (altering execution paths), and data structure manipulation (altering data structures) [4]. Malware analysis can be classified into two ways such as static and dynamic analysis. The static feature uniquely identifies the signature of malware or malware families. Static analysis is vulnerable to code obfuscation techniques. Dynamic analysis is test the program real time by actual execution in controlled environment. In dynamic analysis behavior of malicious software is monitored in emulated environment and traces are obtained from the reports generated by sandbox. It can deal with code evasion techniques [14].

In Ramani et.al [5], ***"Rootkit (malicious code) prediction through data mining methods and techniques"***,**2013,** McAfee's Rootkit Detective was used to detect the hooks created by rootkits and to extract the data from log files Parser-o-matic was used. 1377 hooks were detected after analyzing the log files of 87 samples. Correlation Bayes algorithm was found to attain the maximum level of prediction accuracy than others. The rootkit records were categorized according to their attribute values as Inline or other. However, they do not describe exactly the others are Import Address Table (IAT) Hooking or System Service Descriptor Table (SSDT) hooking or Interrupt Descriptor Table (IDT) or Direct Kernel Object Manipulation (DKOM) Hooking.

In Lobo et.al [6], ***"RBACS: Rootkit Behavioral Analysis and Classification System," 2010,*** they analyzed the inline function hooking techniques by using McAfee's rootkit detector and expectation-maximization (EM) algorithm for clustering the dataset. Parser-o-matic was used to extract the features from log files. However, this paper strictly focused on the rootkit that use inline function hooking techniques only. But, in [7], ***"Identifying Rootkit Infections Using Data Mining," 2010,*** they extended their system by conducting IATs and SSDTs hooking techniques. The CLOPE (Clustering with sLOPE) algorithm was used for analyzing their dataset and ID algorithm was used for identifying the system infected by using Wakaito Environment for Knowledge Analysis (WEKA). But they avoided the major challenge of dealing with encrypted or packed samples. And then in [8], ***"A new procedure to help system/network administrators identify multiple rootkit infections", 2010,***they aimed to detect multiple rootkit infections by trying many different combinations of rootkits through dynamic analysis for inline function hooking techniques only.

In [9],***"An Effective Method for Protecting Native API Hook Attacks in User-mode",2015***, they tried to monitor and detect native API hooking in user space by intercepting native API calls in user mode and looking the traces of IAT entry modification and inline hooking. To test the precision rate of their approach, they have run the malware samples with existing tools R3 Hook scanner, BlackLight, IceSword, and VICE. Ten legitimate API hooks have been taken to determine False Positive. However, their system could not be detected as malicious if any hook which attempts to alter some bytes randomly instead of the first five bytes of a DLL functions. Moreover, this method could not detect malicious code attacks that directly target kernel-mode data structures, specifically System Service Dispatch Table (SSDT).

In ***"Comparative Analysis of Feature Extraction Methods of Malware Detection",2015,***they provide a comparative assessment of features and an overview of malware detection techniques based on static, dynamic and hybrid analysis of executables. The authors proposed general framework of malware detection systemand pinpoints strengths and weaknesses of each method[14].

In ***"Analysis of Malware Behavior: Type Classification using Machine Learning", 2015,*** the authors developed a distributed malware testing environment by extending Cuckoo Sandbox and achieved a high classificationrate with weighted average AUC value of 0.98 using Random Forests classifier [15].

In [17], their objectives are to propose a simple static analysis technique for kernel level rootkit detectionwith the aim of detecting malicious driver and to helpin extracting rootkit driver samples from a largecorpus. They proposed a set of features to distinguishbetween malicious and legitimate drivers based on astudy of modern kernel-level rootkit behaviors.Furthermore, they evaluated the proposed features bygathering 2200 kernel level rootkit drivers and 2220legitimate drivers. Employing a C5classifier, this system obtained accuracy of 98.15% inclassifying the malicious and legitimate drivers.

In our proposed system, we will categorize the rootkit samples based on their relevant families. Because some rootkits could use not only the combination of both user level and kernel level hooking techniques but also more than one kernel level hooking techniques to evade the detection.

3. Technical Background of Rootkits

Normally, each OS consists of a user-mode, called ring 3 and a kernel-mode, called ring 0. The classification of user mode and kernel mode rootkits are as shown in Figure 1 [10].

3.1. User level Rootkits

There are two types of user level rootkit hooking techniques. (a) Inline function hooking technique uses Application Programming Interfaces (APIs) imported from user mode DLLs, while kernel mode rootkits inline hook the native APIs functions that reside in the kernel space. It just overwrites the first five bytes of the code in the API function with trampoline code. (b) Import Address Table (IAT) Hooking is usually achieved via DLL injection. From there it can rewrite the IAT entries, pointing them to handlers within the DLL.

3.2. Kernel level Rootkits

SSDThooking is very powerful because instead of hooking a single program like an IAT hook does, this technique installs a system wide hook that affects every process [16].Haxdoor is an example of SSDT hooking rootkits.**IDT hooking** rootkit modifies the IDT in the memory and then gains the control of the complete system. A stealthier version of idt-hook rootkit could keep the original IDT unchanged by copying it to a new location and altering it. Next attacker could change the IDTR register to point to a new location. Alipop rootkits hook the Global Descriptor Table (GDT). The **layered drivers** use the I/O Request Packet (IRP) for communicating to each other. If a rootkit driver is successfully inserted between the driver chains, it is possible to steal the IRP, which has a lot of important information such as USB data, keyboard data and so on [11]. It is called the Layer Driver Hooking. Popular examples of layered driver rootkits are Turla and Uroburos. **DKOM Hooking** rootkits can manipulate kernel structures and can hide processes and ports, change privileges, and fool the Windows event viewer without many problems. This type of rootkit hides processes by manipulating the list of active processes of the operating system, changing data inside the EPROCESS structures [12]. FU is a popular example of a rootkit that uses DKOM tricks. However, FU does not include a remote communication channel. FU can hide processes and device drivers. It can also elevate the privilege and groups of any Windows process token [13].

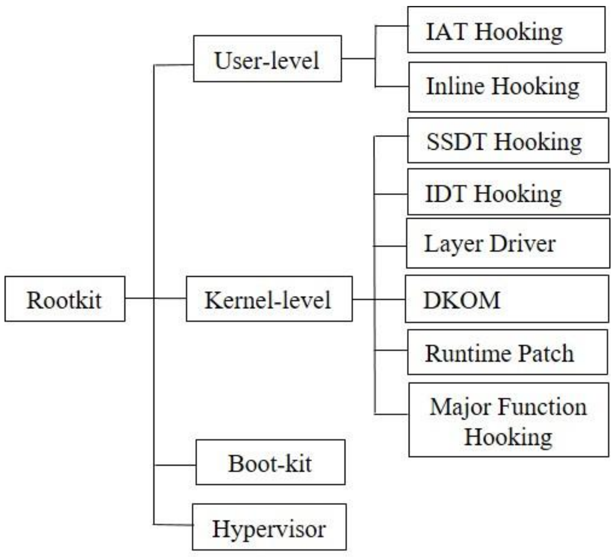


Figure 1:Classification of rootkit

4. Proposed Applicable Process Flow of the System

Nowadays, cybercrimes are committed by using the technology and electronic devices such as computers, mobile phones, and USB devices. The most salient artifact within cyber security is malicious software. The use of rootkits and rootkit technologies in malware and cybercrime is increasing. Moreover, malware creators incorporate rootkit components to maximize their stealth capabilities and to remain undetected in a compromised machine. Deploying rootkit technology buries the malware deep within the computer making it much more difficult to detect and complex to remove. The longer the malware can remain undetected on a compromised machine, the more the cybercriminal can profit. For the above reasons and problems, therefore, features extraction from generated log files in rootkit analysis is very important to reduce the cybercrimes for detection malicious code. So, we proposed the rootkit feature extraction algorithm to point out the dominant features based on the generated log files. The process flow diagram shows the step by step procedure of the rootkits analysis.

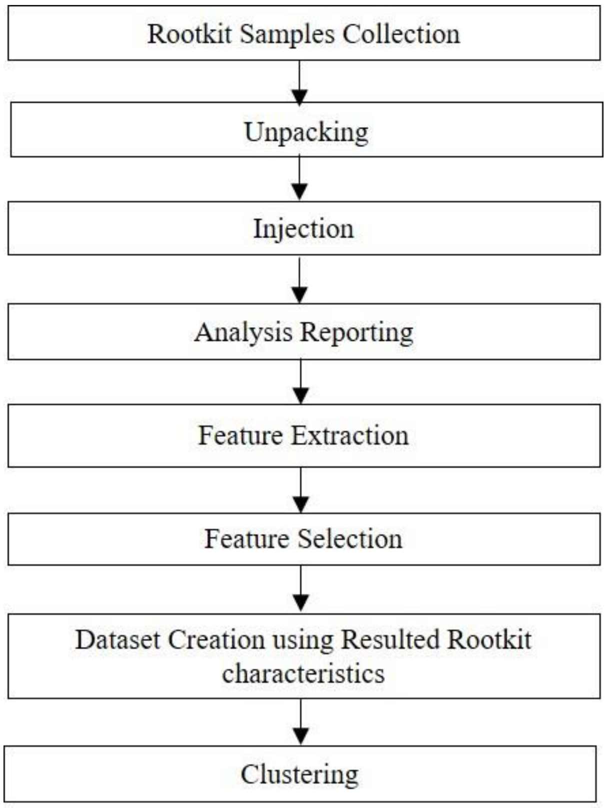

Rootkit Samples Collection
Unpacking
Injection
Analysis Reporting
Feature Extraction
Feature Selection
Dataset Creation using Resulted Rootkit characteristics
Clustering

Figure 2. Overall architecture of the dynamic rootkit analysis

A. Samples Collection Phase

In the samples collection phase, we can get the rootkit samples from the online websites such as offensivecomputing.net, opensecuritytraining, and contagiodump, etc.

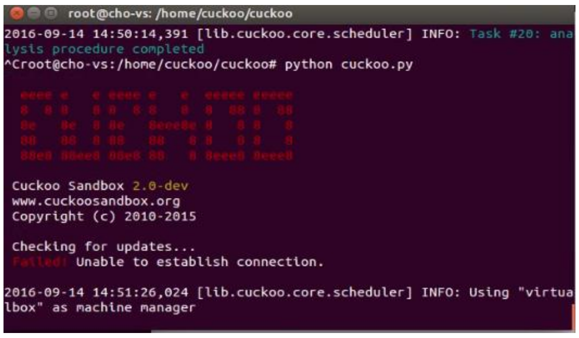
B. Unpacking Phase

For unpacking phase, we use the UPX unpacker for unpacking the samples because malware creators use the packing techniques for obfuscating malware code in some malware. The UPX unpacker plug-in works on packed malware executables and can handle a file even if it has been packed with UPX and modified manually so that UPX cannot be used directly to unpack the file, because internal structures have been modified, for example the names of the sections have been changed from UPX to XYZ, or the version number of the UPX format has been changed from 1.20 to 3.21. This technique often is used by malware authors to make unpacking and reverse engineering harder (www.heaventools.com/PE_Explorer_plug-ins.htm).

C. Injection Phase

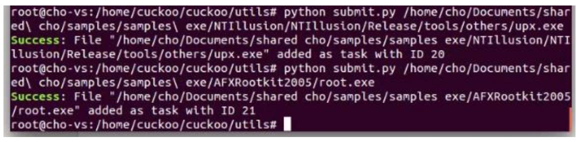
In this phase, cuckoo sandbox will be used as automated rootkits analysis system in our proposed system. Cuckoo Sandbox is one of the open-source projects that has gained popularity in the recent years. It can available at the onlinefreely [19]. It is widely used by academic and independent researchers as well as small to large companies and enterprises. It can analyze many different malicious files (executable, document exploits, Java applets) as well as malicious websites, trace API calls and general behavior of the file, dump and analyze network traffic using Tcpdump, even when encrypted and perform advanced memory analysis of the infected virtualized system with integrated support for Volatility.

The proposed system performs dynamic analysis in the secure virtual environments with Window-7 Operating System by injecting the malicious samples into guest OS from Ubuntu host via cuckoo agent. Cuckoo 2.0 dev version has been used in our malware analysis. Firstly, we must start the cuckoo with python cuckoo.py command as shown in Figure 3. And then submit the sample into the guest operating system with the help of cuckoo agent that installed in the Window-7 guest OS.



```
root@cho-vs: /home/cuckoo/cuckoo
2016-09-14 14:50:14,391 [lib.cuckoo.core.scheduler] INFO: Task #20: ana
lysis procedure completed
^Croot@cho-vs:/home/cuckoo/cuckoo# python cuckoo.py

  [illegible]

 Cuckoo Sandbox 2.0-dev
 www.cuckoosandbox.org
 Copyright (c) 2010-2015

 Checking for updates...
 Failed! Unable to establish connection.

2016-09-14 14:51:26,024 [lib.cuckoo.core.scheduler] INFO: Using "virtua
lbox" as machine manager
```

Figure 3. Starting cuckoo sandbox



```
root@cho-vs:/home/cuckoo/cuckoo/utils# python submit.py /home/cho/Documents/shar
ed\ cho/samples/samples\ exe/NTIllusion/NTIllusion/Release/tools/others/upx.exe
Success: File "/home/cho/Documents/shared cho/samples/samples exe/NTIllusion/NTI
llusion/Release/tools/others/upx.exe" added as task with ID 20
root@cho-vs:/home/cuckoo/cuckoo/utils# python submit.py /home/cho/Documents/shar
ed\ cho/samples/samples\ exe/AFXRootkit2005/root.exe
Success: File "/home/cho/Documents/shared cho/samples/samples exe/AFXRootkit2005
/root.exe" added as task with ID 21
root@cho-vs:/home/cuckoo/cuckoo/utils# 
```

Figure 4. Injecting the malware into guest operating system

In this phase, some malware escape from the analysis because of obfuscation. So, at that time we will perform this type of malware by analyzing manually using some rootkit detectors such as McAfee and GMER, otherwise we will perform automatically by using cuckoo sandbox.

Surface analysis will also be performed before dynamic analysis step in manual rootkit analysis. A surface analysis gets information of malware before execution such as file type, file name, file size, time stamp, strings (meaningful word) and hash value. Some surface analysis tools are String, Digest, HexWorkshop, PEiD and RDG Packer Detector, etc. Figure 5 shows the process flow of the dynamic rootkit analysis manually.

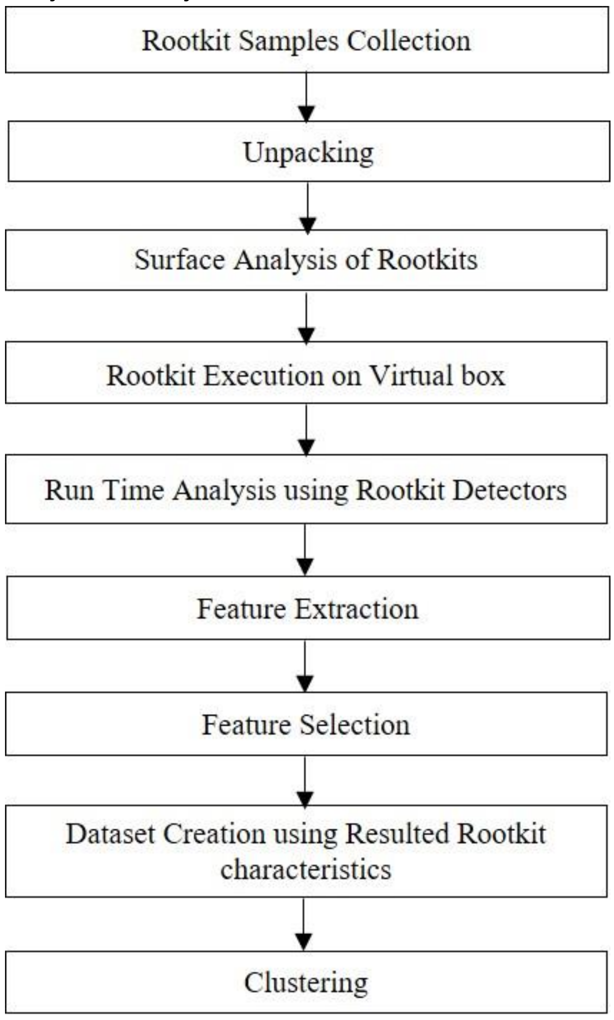

Rootkit Samples Collection
Unpacking
Surface Analysis of Rootkits
Rootkit Execution on Virtual box
Run Time Analysis using Rootkit Detectors
Feature Extraction
Feature Selection
Dataset Creation using Resulted Rootkit characteristics
Clustering

Figure 5. Manual dynamic rootkit analysis procedure

D. Analysis Reporting

This phase describes the reporting of the output from analysis. It generates the analysis result with HTML, JSON (Java Script Object Notation) and Malware Attribute Enumeration and Characterization (MAEC) formats as a report.

E. Feature Extraction and Selection

In the feature extraction and selection phase, the malware feature extraction algorithm will be proposed in this system. Feature extraction phase is one of the essential part of the rootkit analysis processes because the detection accuracy depends on the nature of features or attributes of malicious codes.Principal Component Analysis (PCA) is one of the famous feature extraction method. PCA method can only extract the linear structure information in the data set, however, it cannot extract this nonlinear structure information. Kernel principal component analysis (KPCA) is an improved PCA, which extracts the principal components by adopting a nonlinear kernel method. Some researchers use n-gram feature extraction algorithm. However,we proposed a feature extraction algorithm for extracting the prominent features of the rootkits without using the existing feature extraction algorithms. And then the feature selection phase is necessary in our proposed system due to the nature of attributes overlapping.Some popular feature selection algorithms are TF-IDF, Information Gain, and Chi Squares. After extracting the important information from all samples of normal guest virtual environment, the dataset will be created using spreadsheet for future rootkits detection to help the system administrator. The proposed feature extraction algorithm has been described as below:

```
Algorithm:Effective Malware Feature Extraction
Algorithm (EMFEA)

Input: R ⟵ A collection of report files
Output: F ⟵ Malware features
1: Attributes A1 = {$a_1$, ……, $a_i$};
2: Features A2= {$a_{1i}$, ……..., $a_{ji}$};
3: Read a report file;
4: Process the feature extraction on the file;
5: Output the generated features;
6:For each A1 in report do
7:      If the R has A1then
8:            For each A2 in R do
9:                  Extract A2 from R;
10:                 Add the new signatures into
                    the feature sets;
11:                 F ⟵ A2;
12:           End for
13:     End if
14: End for
```

F. Clustering

By taking a dynamic analysis approach, the proposed system will create the dataset and then cluster the resulted output by using unsupervised clustering technique. Therefore, the generated dataset has been performed with Hierarchical Clustering algorithm to cluster the resulted our own dataset according to their associate and relevant features by using WEKA [18].In hierarchicalclustering clusters are created either by top-down or bottom-up fashion by recursive partitioning. Hierarchical clusteringisdivided into two types such as hierarchical agglomerative methods, and hierarchical divisive clustering. This paper shows the clustering result with percentage based on agglomerative (i.e. bottom up) hierarchical clustering method on WEKA.

5. Contribution

This system proposed an applicable dynamic malware analysis framework for Cyber Crime Investigation and Incident Handling Systems in Myanmar. Moreover, Effective Malware Feature Extraction Algorithm has also proposed in this paper because feature extraction method affects the performance of the systemin terms efficiency, robustness, and accuracy. Therefore, feature extraction phase is one of the essential parts of the malware analysis in Cyber Crime Investigation. List of points are described as follow:

- Perform Dynamic analysis on the rootkit because polymorphic rootkit (or virus) effectively evades signature based detection of its code body.
- Propose a prominent rootkit features extraction algorithm for improving the accuracy.
- Build rootkit dataset to perform experiments for supporting rootkit detection model.
- Identify significant behaviors from the malware samples.

The purposes of analyzing the variations of rootkits are

- To prevent unauthorized access from the remote attackers,
- To protect sensitive information and to reduce financial lost because rootkits pose a very high level of risk to information and information systems.
- To generate the features of the various rootkits using their data access patterns, and then the dataset will be created for future rootkit detection in Incident Handling Systems.

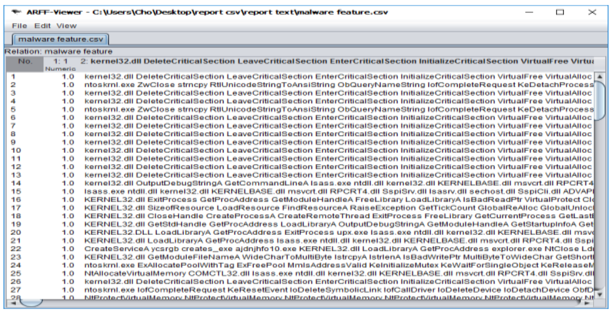
6. Experiment Setup

Although there are several behavior-based rootkits monitoring system that have been conducted in both Linux and Window operating system for user mode and kernel mode, this system will mainly focus only on window user-level and kernel-level rootkits.

Our proposed system uses the following specifications for rootkit analysis. Ubuntu 14.04 LTS-64bit has been used for Host Operating System and Window7-32bit as a Guest Operating System for analyzing the malware sample in Virtual Box. For automated dynamic analysis, Cuckoo Sandbox is used in this system. As for other rootkits detectors, GMER and McAfee's detector will be performed in our proposed system. We will also use other monitoring tools such as Regshot and Process Explorer for detecting the registry, system processes and services.

7. Experimental Result

In our proposed system, we experiment the total 144 malicious samples with different families respectively. After analyzing the malicious samples, we extract the prominent features from the analysis' report into CSV format. And then we convert the CSV format into ARFF for analyzing the data in WEKA with WEKA tool. Figure 6 shows the conversion of CSV into ARFF format by using the ArffViewerin WEKA tool.


ARFF-Viewer - C:\Users\Cho\Desktop\report csv\report text\malware feature.csv
File Edit View
malware feature.csv

```
Relation: malware feature
No.   1: 1     2: kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree Virtu
      Numeric
1     1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
2     1.0   ntoskrnl.exe ZwClose strncpy RtlUnicodeStringToAnsiString ObQueryNameString IofCompleteRequest KeDetachProcess
3     1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
4     1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
5     1.0   ntoskrnl.exe ZwClose strncpy RtlUnicodeStringToAnsiString ObQueryNameString IofCompleteRequest KeDetachProcess
6     1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
7     1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
8     1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
9     1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
10    1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
11    1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
12    1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
13    1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
14    1.0   kernel32.dll OutputDebugStringA GetCommandLineA lsass.exe ntdll.dll kernel32.dll KERNELBASE.dll msvcrt.dll RPCRT4
15    1.0   lsass.exe ntdll.dll kernel32.dll KERNELBASE.dll msvcrt.dll RPCRT4.dll SspiSrv.dll lsasrv.dll sechost.dll SspiCli.dll ADVAPI
16    1.0   KERNEL32.dll ExitProcess GetProcAddress GetModuleHandleA FreeLibrary LoadLibraryA IsBadReadPtr VirtualProtect Cl
17    1.0   KERNEL32.dll SizeofResource LoadResource FindResourceA RaiseException GetTickCount GlobalReAlloc GlobalUnloc
18    1.0   KERNEL32.dll CloseHandle CreateProcessA CreateRemoteThread ExitProcess FreeLibrary GetCurrentProcess GetLast
19    1.0   KERNEL32.dll GetStdHandle GetProcAddress LoadLibraryA OutputDebugStringA GetModuleHandleA GetStartupInfoA Get
20    1.0   KERNEL32.DLL LoadLibraryA GetProcAddress ExitProcess upx.exe lsass.exe ntdll.dll kernel32.dll KERNELBASE.dll msv
21    1.0   KERNEL32.dll LoadLibraryA GetProcAddress lsass.exe ntdll.dll kernel32.dll KERNELBASE.dll msvcrt.dll RPCRT4.dll Sspi
22    1.0   CreateServiceA ycsrgb creates_exe ajdnjhfo10.exe KERNEL32.dll LoadLibraryA GetProcAddress explorer.exe NtClose Ld
23    1.0   KERNEL32.dll GetModuleFileNameA WideCharToMultiByte lstrcpyA lstrlenA IsBadWritePtr MultiByteToWideChar GetShort
24    1.0   ntoskrnl.exe ExAllocatePoolWithTag ExFreePool MmIsAddressValid KeInitializeMutex KeWaitForSingleObject KeReleaseM
25    1.0   NtAllocateVirtualMemory COMCTL32.dll lsass.exe ntdll.dll kernel32.dll KERNELBASE.dll msvcrt.dll RPCRT4.dll SspiSrv.dl
26    1.0   kernel32.dll DeleteCriticalSection LeaveCriticalSection EnterCriticalSection InitializeCriticalSection VirtualFree VirtualAlloc
27    1.0   ntoskrnl.exe IofCompleteRequest KeResetEvent IoDeleteSymbolicLink IofCallDriver IoDeleteDevice IoDetachDevice ObfD
28    1.0   NtProtectVirtualMemory NtProtectVirtualMemory NtProtectVirtualMemory NtProtectVirtualMemory Nt
```

Figure 6: Conversion extracted features in csv to arff format using ArffViewer

Using a spreadsheet, we created a large table with 144 rows for 144 rootkit samples with the attributes including API calls, services, processes, network, import and export files, and registry and so on. In total, the 144 rootkit samples hadcreated 2371 attributes. In our proposed system, feature reduction method has not yet been performed in each sample because malware create and call the system services, processes and function continuously and sequentially. The final step in the process is clustering the dataset. The dataset, thus, consisted of 2371 attributes and 144 instances.

We attempted to cluster the 144 rootkit samples using expectation-maximization (EM) algorithm but the results are not too pleased. Some clusters contain rootkits from two or more different families and should have been broken up into even smaller clusters. Therefore, we applied the Hierarchical Clustering algorithm on this dataset. The Hierarchical Clustering algorithm was available for implementation in the WEKA. One of the advantages of hierarchical clustering is that can generate smaller clusters which may be helpful for discovery.And Hierarchical Clustering algorithm did provide some much better results than the other clustering algorithms with this dataset. The results of our experiment are described in the following figures with 10 clusters. Figure 7 represents the list of samples with their 10-clusterfamily and figure 8 shows the percentage of each cluster such as F1 is 10% and F3 is 49% respectively.

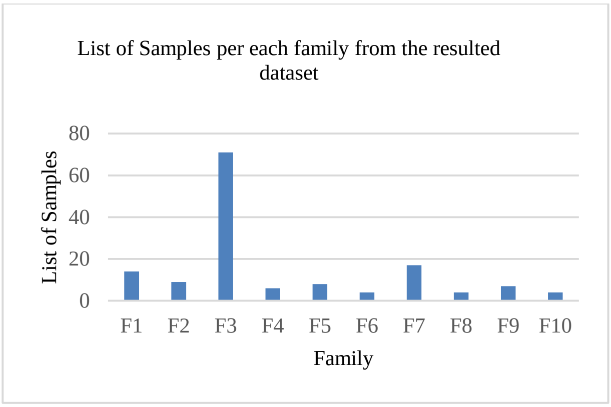


Figure 7. Categorize the resulted dataset with hierarchical clustering algorithm

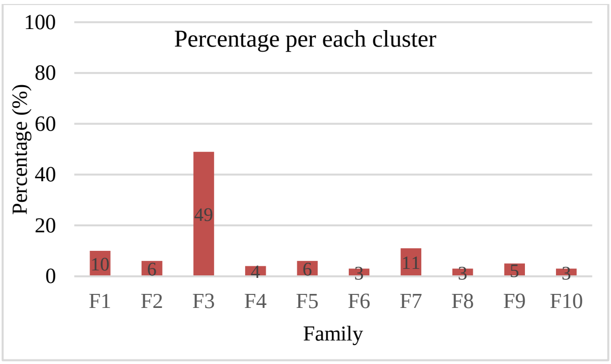


Figure 8. Clustering rootkit family with percentage

We demonstrated the effectiveness of this resulted clusters to validate these results because we needed to confirm that each rootkit family had something in common. Was this algorithm generated meaningful families or clusters for this dataset. To provide this question, we examined by labeling the rootkit samples using numerous online antivirus file scanners that can label suspicious files. VirusTotal'sOnline Virus and Malware Scan has been used to verify the clustering algorithm's result. A list ofthese scanners is provided in Table I with the antivirus labels of families respectively.

Table I. Antivirus Labels with Cluster Family

Family	Antivirus Labels	Antivirus Name	No: of Sample
F1	Win.Trojan.Rootkit-6471 Rootkit.Agent.AJFK Downloader.Darkmegi	ClamAV F-Secure Symantec	14
F2	Win.Trojan.Stuxnet-16 Trojan.Stuxnet.1	ClamAV DrWeb	9
F3	Rootkit.Duqu.A Trojan.Win32.Duqu.a	BitDefender Kaspersky	4
F3	Trojan.Virut.Win32.2137	Zillya	12
F3	Trojan;Win32/Alureon.DX Rootkit.TDSS.BK	Microsoft BitDefender	17
F3	Trojan.Zeroaccess!g31 Rootkit.0Access Backdoor.Win32.ZAccess.apvo	Symantec Malwarebyte Kaspersky	7
F3	Backdoor.Generic.348775 Backdoor.HackDefender	F-Secure, BitDefender Symantec	3
F3	Backdoor.HackDefender Win.Trojan.Hacdef-16	Symantec ClamAV	14
F3	Backdoor.Rustock.Gen.1 Trojan-Clicker.Win32.Costrat.bk	BitDefender Kaspersky	5
F3	Trojan-Dropper.Win32.Smiscer.hf TrojanDropper:Win32/Sirefef.B	Kaspersky Microsoft	9
F4	Rootkit.Agent.AJFK Rootkit.Agent.AJFK	Ad-Aware F-Secure	6
F5	Trojan.Win32.Generic.pak!cobra	AVware	8
F6	Backdoor.Win32.Agent.uq Trojan.Migbot	Kaspersky DrWeb	4
F7	Trojan.NtRootKit.3056 Rootkit.Win32.Agent.uv Win32:Trojan-gen	DrWeb Kaspersky Avast	6
F7	RootKit-NTIllusion Trojan/Hacktool.NTIllusion.a	McAfee TheHacker	5
F7	BackDoor.Ntrootkit.B Win32:NTRootKit [Trj]	AVG Avast	6
F8	Backdoor.Haxdoor.OG	BitDefender	4
F9	Dropped:Trojan.Hider.C	BitDefender	7
F10	Trojan.Rootkit.D	BitDefender, F-Secure	4

The 144 rootkit samples were clustered into ten different families: the first14 samples have been assigned as Darkmegibythe Symantec Antivirus scanner were all groupedinto the same family F1. As a family F2, Win.Trojan.Stuxnet-16 has been labeled by ClamAV. For family F3, 71 rootkit samples have been labeled as Backdoor.Win32 or Trojan.Win32 families. And F4, F5 and F6 are labeled as Rootkit.Agent.AJFK, Trojan.Win32.Generic.pak!cobra, and Trojan.Migbot respectively. In family F7, ithas been labeled as RootKit-NTIllusion or Ntrootkit. The last three families F8, F9, F10 have been labeled as Backdoor.Haxdoor.OG, Dropped:Trojan.Hider.C and Trojan.Rootkit.D. As described above, this paper presented the Effective Malware Feature Extraction Algorithm(EMFEA) and shown that the clustering performance together with Antivirus labels.

8. Conclusion

Modern rootkits are highly obfuscated toconfuse forensics and frustrate reverse engineering,incorporate encrypted files, encrypted communications,and a modular design that allows different types ofmalware from different designers to work together byexporting malicious APIs and syscalls.Rootkits often try to hide resources such as files, processes, Registry entries, andports in order to remain stealthy.Therefore, malware such as Trojans that have the rootkit hiding functionalities will also be considered in this system.This system will also consider the case that a rootkit disables the anti-virus software or turns off the firewall or connect to the C&C server to inject backdoor or malicious code.This system will also consider the case related with remote command/control by monitoring the network traffic in the future.With an increasing amount of malware adopting rootkit technologies to evade antivirus software, further research into defenses against rootkit attacks is essential. Being able to identify rootkit infections is an essential step in handling the infections appropriately.Moreover, the feature extraction is the core part of the malicious samples analysis for high accuracy in detection. Therefore,this system proposes a feature extraction algorithm for identifying the malware attributes. Beyond that, the proposed system performed the clustering from the resulted dataset by using Hierarchical Clustering algorithm in WEKA. However, the clustering is based on the relevant and prominent features. This research focused on a small number of rootkits samples. As for future work, we plan to expand our system by adding other types of rootkits and test on a larger sample set based on their rootkit hooking techniques.Moreover, we will perform the classification of input as benign or malicious.So, the proposed system may expect to detect for future rootkits infections by using the resulted dataset from the relevant features of 144 rootkit samples and it is applicable to identify the future malicious samples.

References

[1]. Ramaswamy, Ashwin. “Detecting kernel rootkits.” (2008).

[2]. Padakanti, Srikanth. “Rootkits Detection Using Inline Hooking.” PhD diss., Texas A&M University-Corpus Christi, 2012.

[3]. “*Rootkits-investigation-procedures*”, SANS Institute,

[4]. DD. Nerenberg, “A study of rootkit stealth techniques and associated detection methods”, AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH GRADUATE SCHOOL OF ENGINEERING AND MANAGEMENT; 2007 Mar.

[5]. RG Ramani, SS Kumar, SG Jacob, “Rootkit (malicious code) prediction through data mining methods and techniques”, In Computational Intelligence and Computing Research (ICCIC), 2013 IEEE International Conference on 2013 Dec 26 (pp. 1-5). IEEE.

[6]. D. Lobo, P. Watters, X.Wu, “RBACS: Rootkit behavioral analysis and classification system”, In Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on 2010 Jan 9 (pp. 75-80). IEEE.

[7]. D. Lobo, P. Watters, X. Wu, “Identifying rootkit infections using data mining”, In2010 International Conference on Information Science and Applications 2010 Apr 21 (pp. 1-7). IEEE.

[8]. D. Lobo, P. Watters, X. Wu, “A new procedure to help system/network administrators identify multiple rootkit infections”, In Communication Software and Networks, 2010. ICCSN'10. Second International Conference on 2010 Feb 26 (pp. 124-128). IEEE.

[9]. K. Muthumanickam, E. Ilavarasan, “An Effective Method for Protecting Native API Hook Attacks in User-mode”, Research Journal of Applied Sciences, Engineering and Technology. 2015 Jan 5;9(1):33-9.

[10]. G. Hoglund, J. Butler, “Rootkits: subverting the Windows kernel”, Addison-Wesley Professional; 2006.

[11]. S. Kim, J. Park, K. Lee, I. You, K. Yim, “A brief survey on rootkit techniques in malicious codes”, Journal of Internet Services and Information Security. 2012;3(4):134-47.

[12]. E. Florio, “When malware meets rootkits”, Virus Bulletin. 2005 Dec;12.

[13]. J. Butler, S. Sparks, *“Windows rootkits of 2005, part one”,* Updated: 02 Nov 2010.

[14]. S. Ranveer, S. Hiray, “Comparative Analysis of Feature Extraction Methods of Malware Detection”, International Journal of Computer Applications. 2015 Jan 1;120(5).

[15]. RS. Pirscoveanu, SS. Hansen, TM. Larsen, M. Stevanovic, JM. Pedersen, A. Czech, “Analysis of malware behavior: Type classification using machine learning”, In Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), 2015 International Conference on 2015 Jun 8 (pp. 1-7). IEEE.

[16]. J. Butler, S. Sparks, “*Windows rootkits of 2005, part one*”, Updated: 02 Nov 2010.

[17]. S. A. Musavi, and M. Kharrazi, "*Back to Static Analysis for Kernel-Level Rootkit Detection*", IEEE, 2013.

[18]. http://www.cs.waikato.ac.nz/ml/weka/

[19]. https://github.com/cuckoosandbox/cuckoo