GRIFFITH UNIVERSITY

Identifying Rootkit Infections Using Data Mining

Author

Wu, Xin-Wen, Lobo, Desmond, Watters, Paul

Published

2010

Conference Title

Proceedings of The 2010 International Conference on Information Science and Applications (ICISA)

DOI

10.1109/ICISA.2010.5480359

Downloaded from

http://hdl.handle.net/10072/37518

Griffith Research Online

https://research-repository.griffith.edu.au

Identifying Rootkit Infections Using Data Mining

Desmond Lobo, Paul Watters and Xin-Wen Wu
Internet Commerce Security Laboratory
Graduate School of Information Technology and Mathematical Sciences
University of Ballarat, Australia
desmondlobo@students.ballarat.edu.au, {p.watters, x.wu}@ballarat.edu.au

Abstract - **Rootkits refer to software that is used to hide the presence and activity of malware and permit an attacker to take control of a computer system. In our previous work, we focused strictly on identifying rootkits that use inline function hooking techniques to remain hidden. In this paper, we extend our previous work by including rootkits that use other types of hooking techniques, such as those that hook the IATs (Import Address Tables) and SSDTs (System Service Descriptor Tables). Unlike other malware identification techniques, our approach involved conducting dynamic analyses of various rootkits and then determining the family of each rootkit based on the hooks that had been created on the system. We demonstrated the effectiveness of this approach by first using the CLOPE (Clustering with sLOPE) algorithm to cluster a sample of rootkits into several families; next, the ID3 (Iterative Dichotomiser 3) algorithm was utilized to generate a decision tree for identifying the rootkit that had infected a machine.**

Keywords - computer security; rootkits; data mining

I. INTRODUCTION

Rootkits refer to software that is used to hide the presence and activity of malware (such as viruses, worms and trojans) and permit an attacker to take control of a computer system [1]. Installing a rootkit is usually the first thing that an attacker will do after gaining access to a system, as this will ensure that the attack will remain undetected [2]. The attacker can then proceed to capture personal data, such as bank account details, passwords, and credit card numbers.

There are clearly two reasons why it is very important to conduct research in the area of rootkits:

1. It is estimated that 85% of malicious software is being written today with the intention of generating profit for the malware's author [3]. We are no longer dealing with script kiddies just trying to create malware for fun, but instead are targeted by organized criminal gangs that want to steal money. The Symantec Corporation even claims that "cyber crime has surpassed illegal drug trafficking as a criminal moneymaker" [4].
2. There has been an increase of several hundred percent in both the number and complexity of rootkits over the last few years [5]. Malicious software is already a very big worldwide problem and the proliferation of rootkits is only going to serve to escalate this problem.

These two trends are illustrated in Figure 1.

Rootkits use various types of hooking techniques in order to remain hidden and there are several tools available, such as McAfee's Rootkit Detective, that can be used to detect the hooks that have been created by a rootkit on a computer system. Each time that such a tool is run, a log file is generated that contains a list of the detected hooks. The amount of data in these log files is overwhelming as they hold information about each and every hook that had been detected on the system. On average, each of these log files contains several hundred lines of data.

The main contribution of this paper is that we have devised a new procedure that can be used to make sense of the vast amount of information in these log files. This procedure can identify the rootkit that has infected a machine, based on the hooks that have been detected. In our previous work [7, 8], we focused strictly on identifying rootkits that use inline function hooking techniques to remain hidden. In this paper, we extend our previous work by including rootkits that use other types of hooking techniques, such as those that hook the IATs (Import Address Tables) and SSDTs (System Service Descriptor Tables).

The rest of this paper is organized as follows. We describe in the next section three hooking techniques that rootkits use to remain hidden. In Section III, we explain the methodology that was used to identify the rootkit that has infected a computer system. We demonstrated our identification process on a sample of rootkits and describe the results of our experiment in Section IV. A discussion of some of the related work is provided in Section V. Finally, a conclusion to the paper can be found in Section VI.

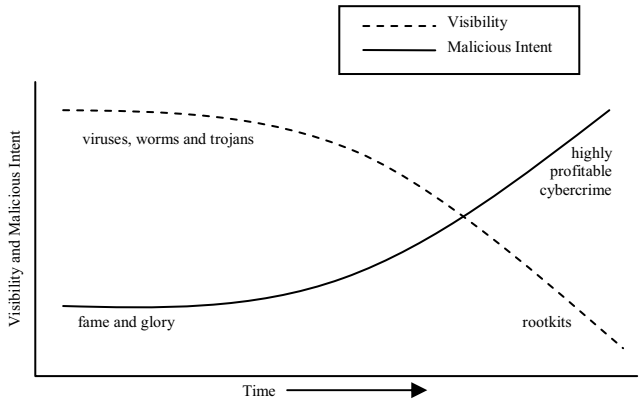

Figure 1. Visibility of Malware versus Malicious Intent [6]

This research was supported in part by the Westpac Banking Corporation, IBM Australia and the Australian Government.

II. ROOTKIT HOOKING TECHNIQUES

As mentioned earlier, rootkits use different types of hooking techniques in order to remain hidden. In this paper, we focus on three types: inline function hooking, hooking the IAT and hooking the SSDT.

```
McAfee(R) Rootkit Detective 1.1 scan report
On 16-11-2009 at 03:17:08
OS-Version 5.1.2600
Service Pack 3.0
==================================
Object-Type: IAT/EAT-hook
PID: 444
Details: Export : Function :
  kernel32.dll!Process32Next =>
  C:\WINDOWS\system32\kernel32.dll:7C80029C
Object-Path: C:\WINDOWS\system32\kernel32.dll
Status: Hooked
```

Figure 2. Inline Function Hook Created by a Qukart Rootkit

```
McAfee(R) Rootkit Detective 1.1 scan report
On 16-11-2009 at 03:25:37
OS-Version 5.1.2600
Service Pack 3.0
==================================
Object-Type: IAT/EAT-hook
PID: 1496
Details: Import : Function :
  SHDOCVW.dll:ADVAPI32.dll!RegEnumValueA
  Should be : ADVAPI32.dll:77DF9BBF But is : :10414044
Object-Path: :10414044
Status: Hooked
```

Figure 3. IAT Hook Created by an Alman Rootkit

```
McAfee(R) Rootkit Detective 1.1 scan report
On 16-11-2009 at 03:41:17
OS-Version 5.1.2600
Service Pack 3.0
==================================
Object-Type: IAT/EAT-hook
PID: 1496
Details: Import : Function :
  Explorer.EXE:KERNEL32.dll!LoadLibraryA
  Should be : KERNEL32.dll:7C801D7B But is    :
  C:\DOCUME~1\DES\LOCALS~1\Temp\VCab.DLL:00D884E8
Object-Path: C:\DOCUME~1\DES\LOCALS~1\Temp\VCab.DLL
Status: Hooked
```

Figure 4. IAT Hook Created by a Bacalid/DetNat Rootkit

```
McAfee(R) Rootkit Detective 1.1 scan report
On 16-11-2009 at 03:55:05
OS-Version 5.1.2600
Service Pack 3.0
==================================
Object-Type: SSDT-hook
Object-Name: ZwCreateProcess
Object-Path: C:\WINDOWS\system32\vdnt32.sys
```

Figure 5. SSDT Hook Created by a Haxdoor Rootkit

A. *Inline Function Hooking*

Figure 2 describes one of the many inline function hooks that had been created by a Qukart rootkit. These types of hooks are created when a rootkit overwrites the first five bytes of an API (Application Programming Interface) function with a JUMP instruction. The first byte in the function is replaced with the value E9, the opcode for a JUMP in assembly language, and the remaining four bytes contain a 32-bit address of some malicious code.

From Figure 2, it is evident that the Process32Next API function in the Kernel32 DLL (Dynamic-Link Library) file had been hooked. This API function had been exported by the Kernel32 DLL file to the MsgSys Windows process (Process IDentifier 444). Whenever this particular API function was called, some malicious code at address 7C80029C was run.

B. *Hooking the IAT*

Figures 3 and 4 describe two of the many IAT hooks that had been created by an Alman rootkit and by a Bacalid/DetNat rootkit, respectively. These types of hooks are created when a rootkit overwrites the address of an API function in the IAT. Figure 3 is an example of a hook in which the IAT of a DLL file had been modified and Figure 4 is an example of a hook in which the IAT of an EXE (EXEcutable) process had been modified.

Figure 3 shows that the Explorer Windows process (Process IDentifier 1496) depends on the Shdocvw DLL file. This Shdocvw DLL file, in turn, needs to import the RegEnumValueA API function from the Advapi DLL file. The Alman rootkit had managed to create a hook by overwriting the address of the RegEnumValueA function in the IAT of the Shdocvw DLL file: the address should have been 77DF9BBF but had been changed to 10414044.

From Figure 4, it is clear that the Explorer Windows process (Process IDentifier 1496) needs to import the LoadLibraryA API function from the Kernel32 DLL file. The Bacalid/DetNat rootkit had managed to create a hook by overwriting the address of the LoadLibraryA function in the IAT of the Explorer Windows Process: the address should have been 7C801D7B but had been changed to 00D884E8.

C. *Hooking the SSDT*

Figure 5 describes one of several SSDT hooks that had been created by a Haxdoor rootkit. System services refer to undocumented API functions for the Windows operating system that are callable from user mode [9]. From Figure 5, for example, ZwCreateProcess is the internal system service that the CreateProcess API function calls to create a new process. A system service call is thus a mechanism that allows a user mode application to access the operating system's kernel [10].

The SSDT contains a list of pointers with the addresses of the internal kernel function that implements the corresponding service [9, 10, 11]. A rootkit can intercept calls that are made to a specific system service by replacing the SSDT entry with the address of its own code. After this rootkit code is executed, the original system service can be called or some fabricated data can be returned instead [11, 12].

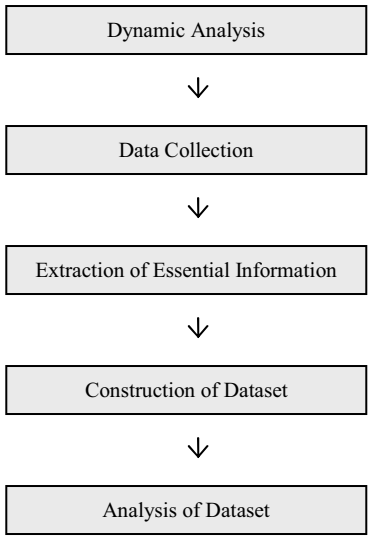

Figure 6. Methodology

III. METHODOLOGY

Having described the various rootkit hooking techniques in the previous section, we now explain our procedure that can be used to identify the rootkit that has infected a computer system. There are several steps involved in the process and these are illustrated in Figure 6.

A. Controlled Environment for Conducting Dynamic Analyses of Rootkits

It was first necessary to set up a controlled environment for conducting a dynamic analysis of rootkits. Since the Windows family accounts for approximately 90% of the operating systems in use today [13], we decided to focus exclusively on rootkits that target those systems. Ubuntu was chosen as the host operating system as it was felt that it was unlikely that a Windows rootkit could cause damage to a Linux-based operating system.

Within the Windows family of operating systems, Windows XP is by far the most popular [13]; thus, Sun Microsystem's VirtualBox was installed to create a virtual environment for testing rootkits and we installed XP as a guest operating system inside that virtual environment.

We obtained 100 rootkit samples from the Offensive Computing website (http://www.offensivecomputing.net) and ran each sample, one by one, in the virtual environment. After running each sample, we restored the system back to its original settings.

B. Rootkit Hook Detector for Collecting Data

There are several tools that can be used to detect the hooks that have been created by a rootkit on a Windows machine; we chose to use McAfee's Rootkit Detective (available from http://vil.nai.com) to accomplish this task. After running each of the rootkit samples in the Windows XP virtual environment, the Rootkit Detective was then run to detect the hooks that had been created and a log file containing data about each of these hooks was generated. Figures 2, 3, 4 and 5 are extracts from these log files.

C. Parser for Extracting Essential Information

The log files contained a fair amount of redundant information and the next step involved extracting just the essential information about each of the hooks from the files. The exact information that was extracted depended on the type of hook that was detected. Pyroto's Parse-O-Matic parser (available from http://www.parse-o-matic.com) was used to extract this data from the log files.

- For inline function hooks, the Process IDentifier and the API function name were extracted. Thus, the following data would have been obtained from Figure 2:

  444 | Process32Next

- For IAT hooks, the Process IDentifier, the hooked DLL/EXE name, and the API function name were extracted. Thus, the following data would have been obtained from Figure 3:

  1496 | Shdocvw.dll | RegEnumValueA

  and the following data would have been obtained from Figure 4:

  1496 | Explorer.exe | LoadLibraryA

- For SSDT hooks, just the API function name was extracted. Thus, the following data would have been obtained from Figure 5:

  ZwCreateProcess

D. Spreadsheet for Constructing Dataset

After examining the log files from each of the 100 rootkit samples, it was determined that a total of 12472 hooks had been detected. From these 12472 hooks, we were able to find 1357 unique hooks. Thus, using a spreadsheet, a large table with 100 rows (one row for each rootkit) and 1357 columns (one column for each unique hook) was then constructed. The table contained 0/1 binary values, with a 1 meaning that a particular rootkit had managed to create a certain hook. The binary data from the spreadsheet's table was then used to create a dataset. The dataset, therefore, consisted of 100 instances and 1357 attributes.

E. Data Mining Software for Analysing Dataset

Having generated a dataset, we subsequently used the Wakaito Environment for Knowledge Analysis (WEKA) to examine it [14]. The results of this analysis are described in the next section.

IV. EXPERIMENTAL RESULTS

Just to reiterate, the primary objective of this research was to be able to identify the rootkit that has infected a system. Prior to explaining how one can identify the rootkit, it is first essential that we demonstrate how a sample of rootkits can be clustered into different families. This step is necessary as it will

show that the variants within each rootkit family have similar hooking patterns.

Thus, in subsection A, we reveal a procedure that can be used to cluster the 100 instances in the dataset (which represent the 100 rootkit samples) into different families. Having some knowledge of the hooking patterns of each family, we then show how a decision tree could be used to identify the rootkit that has infected a system, and this is described in subsection B.

A. Clustering

Dividing the 100 rootkit samples into different families was a two-step process. The first step involved breaking up the samples into groups based on the type of hooks that each particular rootkit had created. The samples were broken up into the following five groups:

- Inline function hooks only (67 samples)
- IAT hooks only (7 samples)
- SSDT hooks only (9 samples)
- Inline function and SSDT hooks (11 samples)
- IAT and SSDT hooks (6 samples)

It should be pointed out that we did not find any rootkits that had created both inline function and IAT hooks.

TABLE I. CLUSTERING RESULTS

Family Number	Family Name	Type of Hooks	Number of Samples
F01	Papras	Inline & SSDT	4
F02	Haxdoor-A	SSDT	1
F03	Bacalid/DetNat	IAT	6
F04	Haxdoor-B	Inline & SSDT	7
F05	Agent	IAT	1
F06	Alman-A	SSDT	8
F07	Feebs-A	Inline	14
F08	Qukart	Inline	18
F09	Virut	Inline	21
F10	Feebs-B	Inline	1
F11	Alman-B	IAT & SSDT	6
F12	ProAgent	Inline	5
F13	DNSChanger	Inline	6
F14	Banker	Inline	2

The second step involved breaking up these five groups into smaller groups using a clustering algorithm. We intended to use an unsupervised clustering model since the instances had not been labeled. One of the most common clustering algorithms is the k-means algorithm, but a disadvantage of this algorithm is that the number of clusters needs to be specified *a priori*.

The EM (Expectation-Maximization) clustering algorithm, on the other hand, can find the optimum number of clusters automatically using a cross validation procedure. We attempted to cluster the 100 rootkit samples using this EM algorithm but were not too pleased with the results. This algorithm resulted in too few clusters: some clusters contained rootkits from two or more different families and should have been broken up into even smaller clusters.

The CLOPE (Clustering with sLOPE) algorithm, however, did provide some much better results. Like the EM algorithm, the CLOPE algorithm can also find the most suitable number of clusters automatically. The CLOPE algorithm proved to be an appropriate choice for analyzing our dataset. As mentioned earlier,

- Our dataset consisted of 0/1 binary values.
- Over 90% of the values in the dataset were zero.
- Each instance in the dataset contained 1357 attributes.

The CLOPE algorithm is an appropriate choice for examining sparse datasets with high dimension and containing boolean values [15]. The algorithm was available for implementation in WEKA.

The CLOPE algorithm contained a repulsion parameter, which is used to control the level of intra-cluster similarity [14, 15]. We used the default repulsion value of 2.6 for our analysis. Using this algorithm to cluster each of the five groups resulted in a total of 14 smaller groups. We refer to these smaller groups as families and list them in Table I.

It was then necessary to validate these results. We needed to demonstrate the effectiveness of this clustering algorithm; in other words, we needed to confirm that the rootkit samples within each family had something in common. Had the algorithm generated meaningful families? To answer this question, we proceeded by labeling the rootkit samples using several different antivirus scanners.

There are numerous online antivirus file scanners that can be used to label suspicious files. We chose to use VirusTotal's Online Virus and Malware Scan (available from http://www.virustotal.com). Once a file is uploaded to this website, the file is quickly scanned and the website then returns labels from 39 different antivirus scanners for that file. A list of these scanners is provided in Table II.

Bailey et al. [16] had pointed out that for each malware sample in the wild, there exist a variety of labels that have been chosen by the various antivirus vendors. Thus, when presented with 39 labels for each of our rootkit samples, we decided to choose the label that appeared most frequently. Once we had a label for each rootkit sample, we then verified that all the samples within each family had identical labels. This label that

was common to all members of the family was obviously selected as the name of the family and recorded in Table I. It is worth noting that other researchers, such as Bayer et al. [17], also used the labels from several different antivirus scanners to validate their clustering results.

TABLE II. VIRUSTOTAL'S ANTIVIRUS SCANNERS

AhnLab (V3)	Eset Software (ESET NOD32)	Norman (Norman Antivirus)
Antiy Labs (Antiy-AVL)	Fortinet (Fortinet)	Panda Security (Panda Platinum)
Aladdin (eSafe)	FRISK Software (F-Prot)	PC Tools (PCTools)
ALWIL (Avast! Antivirus)	F-Secure (F-Secure)	Prevx (Prevx1)
Authentium (Command Antivirus)	G DATA Software (GData)	Rising Antivirus (Rising)
AVG Technologies (AVG)	Hacksoft (The Hacker)	Secure Computing (SecureWeb)
Avira (AntiVir)	Hauri (ViRobot)	BitDefender GmbH (BitDefender)
Cat Computer Services (Quick Heal)	Ikarus Software (Ikarus)	Sophos (SAV)
ClamAV (ClamAV)	INCA Internet (nProtect)	Sunbelt Software (Antivirus)
Comodo (Comodo)	K7 Computing (K7AntiVirus)	Symantec (Norton Antivirus)
CA Inc. (Vet)	Kaspersky Lab (AVP)	VirusBlokAda (VBA32)
Doctor Web, Ltd. (DrWeb)	McAfee (VirusScan)	Trend Micro (TrendMicro)
Emsi Software GmbH (a-squared)	Microsoft (Malware Protection)	VirusBuster (VirusBuster)

This same procedure to verify the effectiveness of the CLOPE algorithm had also been used to confirm that the EM algorithm had NOT generated proper clusters. We noticed, for example, that the EM algorithm had inappropriately grouped the samples from families F03 and F05 into the same cluster.

B. *Decision Tree*

Having successfully completed the intermediate step of categorizing the 100 rootkit samples into 14 families, the next objective was to develop a procedure that would identify the rootkit that had infected a system. With some knowledge of the hooking patterns of each rootkit family, we could now construct a decision tree that could be used to find the attributes that would differentiate between the different families of rootkits.

To construct the decision tree, it was first necessary to make some slight modifications to the original dataset. The original dataset had consisted of 100 instances and 1357 attributes. We added one more attribute to our dataset: this 1358th attribute contained the name of the family that the rootkit belonged to. We also added one more instance to the dataset: this 101st instance contained all zero values and represented the situation in which none of the 1357 hooks had been detected. This 101st instance was placed in and was the only member of a family called F00.

The ID3 (Iterative Dichotomiser 3) algorithm was then used to analyze the modified dataset and create a decision tree. The primary objective of any type of decision tree classification is to iteratively partition the dataset into subsets and eventually end up with final subsets in which all elements belong to the same class. With the ID3 algorithm, the basic strategy when splitting the dataset is to choose attributes that have the highest information gain. [18, 19]

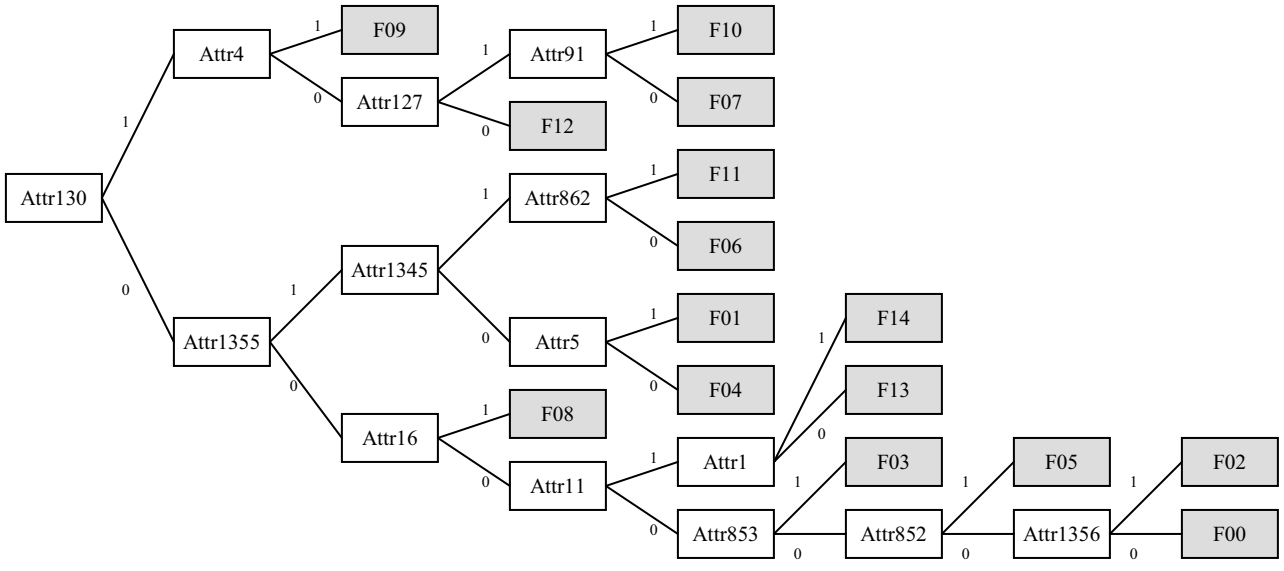

Figure 7. Decision Tree to Identify Rootkit Infections

TABLE III. DESCRIPTION OF ATTRIBUTES

Attribute	Hook Type	Windows Process	Hooked DLL	API Function
130	Inline	Explorer		FindNextFileW
1355	SSDT			ZwQueryDirectoryFile
4	Inline	Svchost		FindNextFileW
16	Inline	Svchost		Process32Next
1345	SSDT			ZwClose
127	Inline	Explorer		FindFirstFileA
11	Inline	Svchost		NtDeleteValueKey
5	Inline	Svchost		HttpSendRequestA
862	IAT	Explorer	Advapi32	FindFirstFileW
91	Inline	DefWatch		FindFirstFileA
853	IAT	Explorer	AcGenral	FindFirstFileW
1	Inline	Svchost		EnumServicesStatusA
852	IAT	Explorer	AcGenral	RegOpenKeyW
1356	SSDT			ZwQuerySystemInformation

Like the CLOPE clustering algorithm, the ID3 algorithm was also suitable for examining a dataset containing many binary attributes [14, 18]. Figure 7 illustrates the decision tree that was generated by the ID3 algorithm after implementation in WEKA. Leaves F01, F02, F03 … F14 in the decision tree correspond to the 14 families in Table I. If one traverses the decision tree and arrives at the F00 leaf, this could mean either one of two things:

1. The rootkit had not created any hooks (the system had possibly not been infected by a rootkit).
2. The hooks that had been created by the rootkit were different from the ones being tested for. This might suggest that we are dealing with a brand new rootkit.

As mentioned earlier, the original dataset consisted of 1357 attributes, each representing a particular hook. To simplify matters, it would be sufficient to give the details of just those 14 attributes in Figure 7 that are needed to identify the family of a rootkit that has infected a system, and this information is provided in Table III. The 14 attributes in Table III were chosen by the ID3 algorithm as being the most significant attributes for determining the family of a rootkit infection

In summary, after running a tool such as McAfee's Rootkit Detective, the decision tree in Figure 7 along with the information contained in Table III could then be used to identify the rootkit that has infected a system.

V. RELATED WORK

Schultz et al. [20], Kolter and Maloof [21] and Siddiqui et al. [22] used data mining techniques for the purpose of identifying malicious software. Their approaches involved static analyses of malicious executables files that might arrive, for example, as email attachments. The intention was to be able to catch these files before they had a chance to run on and possibly cause damage to a system. A significant problem for this type of static analysis approach is that 80 to 90 percent of malware in the wild is either encrypted or packed [23, 24].

By taking a dynamic analysis approach to identifying rootkits, we avoided this major challenge of dealing with encrypted or packed malware samples. Our goal, on the other hand, attempted to address a situation that a system administrator might be faced with when he or she finds a machine behaving abnormally because of a rootkit infection. His or her immediate concern would be to try to determine what he or she was dealing with, and our system tries to tackle this concern.

There have been other groups of researchers, such as Bailey et al. [16], Rieck et al. [25] and Bayer et al. [17], whose work was more like ours in that they too collected malware samples, performed a dynamic analysis of each sample in a controlled environment, and eventually ended up with clusters of families. The objective of these researchers was to cluster malware samples based on the behavior of each sample: they attempted to identify the distinctive behavioral features of each malware family. Yin et al. [26] conducted a dynamic analysis of various malware samples with the objective of detecting the hooks that had been created on a system. Their innovative technique could be used to identify new hooking techniques, as well. Our approach was closely related to that of these four groups of researchers but more focused, as we concentrated strictly on rootkits and used the hooks that had been created by these rootkits to differentiate between the various families.

VI. CONCLUSION

With an increasing amount of malware adopting rootkit technologies to evade antivirus software, further research into defenses against rootkit attacks is absolutely essential. In this paper, we focused on rootkits that use inline function, IAT and SSDT hooking techniques to remain hidden. There are several tools available, such as McAfee's Rootkit Detective, that can be used to detect these types of hooks, but these tools cannot identify the rootkits. The main contribution of this paper is that we have developed a new procedure that will enhance these currently available tools for detecting the hooks on a system. This procedure would identify the family of the rootkit as well, and this knowledge would certainly help the administrator of that system in deciding how to proceed.

To demonstrate our approach, we conducted dynamic analyses of 100 rootkit samples in a controlled environment. It was determined that these rootkits had created a total of 12472 hooks. Based on the hooks that had been created by each rootkit, we were then able to categorize these samples into 14 families using an unsupervised clustering algorithm. Next, the labels from 39 different antivirus scanners were used to verify the effectiveness of this algorithm. Having successfully categorized the samples into families, a decision tree was then generated that could be used to identify the family of the rootkit.

Being able to identify rootkit infections would mean that half the battle against these extremely sophisticated forms of malware would be won. Since we have shown that the variants

within a particular rootkit family have similar hooking patterns, the procedure that we have presented in this paper could also be used to very quickly identify newly released variants from that family. Hence, we envision that this procedure will lead to the development of better tools for identifying the rootkit that has infected a system.

ACKNOWLEDGMENT

We would like to thank Prof Lynn Batten and the anonymous reviewers for their helpful comments.

REFERENCES

[1] A. Emigh, "The Crimeware Landscape: Malware, Phishing, Identity Theft and Beyond", Journal of Digital Forensic Practice, Vol. 1(3), Sept. 2006, pp. 245-260

[2] M. Alvarez, M. Vucelich, and L. Johnson, "IBM Internet Security Systems X-Force Threat Insight Monthly", July 2008, IBM Corporation

[3] L. Wang and P. Dasgupta, "Kernel and Application Integrity Assurance: Ensuring Freedom from Rootkits and Malware in a Computer System", Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops, 2007, IEEE Computer Society

[4] Symantec, "Cyber Crime has Surpassed Illegal Drug Trafficking as a Criminal Moneymaker; 1 in 5 will become a Victim", Symantec Corporation Press Release, Retrieved December 5, 2009 from http://www.symantec.com

[5] McAfee, "Rootkits - Part 1 of 3: The Growing Threat", McAfee Inc., Apr. 2006

[6] OECD, "Malicious Software (Malware): A Security Threat to the Internet Economy", Organization for Economic Co-operation and Development, June 2008, OECD Ministrial Meeting on the Future of the Internet Economy

[7] D. Lobo, P. Watters and X. Wu "RBACS: Rootkit Behavioral Analysis and Classification System", Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, 2010, IEEE Computer Society

[8] D. Lobo, P. Watters and X. Wu (in press) "A New Procedure to Help System/Network Administrators Identify Multiple Rootkit Infections", Proceedings of the Second International Conference on Communication Software and Networks, 2010, IEEE Computer Society

[9] M. E. Russinovich and D. A. Solomon, "Microsoft Windows Internals", 4th Edition, 2005, Microsoft Press

[10] K. Kasslin, M. Stahlberg, S. Larvala and A. Tikkanen, "Hide 'n Seek Revisited – Full Stealth is Back", Proceedings of the Virus Bulletin Conference, 2005

[11] G. Hoglund, and J. Butler, "Rootkits: Subverting the Windows Kernel", 2005, Addison-Wesley Professional

[12] C. Ries, "Inside Windows Rootkits", VigilantMinds, 2006

[13] W3Schools, "OS Platform Statistics", Retrieved October 10, 2009 from http://www.w3schools.com

[14] I. H. Witten and E. Frank, "Data Mining: Practical Machine Learning Tools and Techniques", 2nd Edition, 2005, Morgan Kaufmann

[15] Y. Yang, X. Guan and J. You, "CLOPE: A Fast and Effective Clustering Algorithm fo Transactional Data", Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining, 2002, ACM Special Interest Group on Knowledge Discovery and Data Mining

[16] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian and J. Nazario, "Automated Classification and Analysis of Internet Malware", Recent Advances in Intrusion Detection, Lecture Notes in Computer Science, Vol. 4637, 2007, pp. 178-197, Springer

[17] U. Bayer, P. Milani Comparetti, C. Hlauschek, C. Kruegel and E. Kirda, "Scalable, Behavior-Based Malware Clustering", Proceedings of the 16th Annual Network and Distributed System Security Symposium, Feb. 2009

[18] J. R. Quinlan, "Induction of Decision Trees", Machine Learning, Vol. 1, Mar. 1986, pp. 81-106

[19] M. H. Dunham, "Data Mining: Introductory and Advanced Topics", 2003, Pearson Education

[20] M. G. Schultz, E. Eskin, E. Zadok and S. J. Stolfo, "Data Mining Methods for Detection of New Malicious Executables", Proceedings of the 2001 IEEE Symposium on Security and Privacy, IEEE Computer Society

[21] J. Z. Kolter and M. A. Maloof, "Learning to Detect and Classify Malicious Executables in the Wild", Journal of Machine Learning Research, Vol. 7, Dec. 2006, pp. 2721-2744

[22] M. Siddiqui, M. C. Wang and Joohan Lee, "Detecting Internet Worms Using Data Mining Techniques", Journal of Systemics, Cybernetics and Informatics, Vol, 6, No. 6, 2008, pp. 48-53

[23] M. Morgenstern and T. Brosch, "Runtime Packers: The Hidden Problem?", Proceedings of Black Hat USA, 2006

[24] R. Lyda and J. Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware", IEEE Security and Privacy, Vol. 5, Issue 2, March 2007, pp. 40-45

[25] K. Rieck, T. Holz, C. Willems, P. Dussel and P. Laskov, 2008, "Learning and Classification of Malware Behavior", Detection of Intrusions and Malware, and Vulnerability Assessment, Lecture Notes in Computer Science, Vol. 5137, pp. 108-125, Springer

[26] H. Yin, Z. Liang and D. Song, "HookFinder: Identifying and Understanding Malware Hooking Behaviors", Proceedings of the 15th Annual Network and Distributed System Security Symposium, Feb. 2008