COPDA: CONCEALED PROCESS AND SERVICE DISCOVERY ALGORITHM TO REVEAL ROOTKIT FOOTPRINTS

K.Muthumanickam[1], E.Ilavarasan.E[2]

[1, 2]Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry - 605 014, India

Email: kmuthoo@pec.edu[1], eilavarasan@pec.edu[2]

ABSTRACT

The current online world is constantly affected by malicious software such as viruses, Trojans, worms, spyware and botnets. When such a malicious software integrates with the rootkit technique, it becomes a serious threat to end users. Rootkits themselves do not cause damage to a computer. Instead, they mask their footprints either from antivirus software or anti-rootkit tools to allow a remote attacker to conduct computer crimes for a long time. This property makes malicious code attacks difficult to detect. Traditional techniques that aim to reveal rootkit footprints suffer from false alarm rate and also fail to detect unknown stealthy malicious code attacks. The proposed Concealed Process and Service Discovery Algorithm (CoPDA) introduces a novel cross-view comparison technique that can effectively detect the concealed processes and services of a malicious software in Windows operating system. Compare to existing anti-rootkit detection tools, the experimental results show that CoPDA can be effectively used to discover hidden process and service and deserved 99.02% detection accuracy, 100% true positive rate and 1.82% false positive rate. Additionally, CoPDA is portable across various operating systems with only negligible tweaking.

Keywords: Crosscheck, Hidden-process, Rootkit, Windows, Virtualization.

1.0 INTRODUCTION

There are many forms of malicious software that can constantly affect a computer. Today, more advanced malicious software is incorporated with rootkit techniques to make detection more difficult. A rootkit is a technique which is designed with the intent of allowing the remote attacker to maintain highest privilege over the resources in the victim operating system. It has been in the wild for more than 15 years [1]. Different malware adopts different masquerading methods to avoid its detection. As a result, rootkits can dynamically defy detection either by hiding from the view of anti-virus software. Because of these characteristics, rootkits are potentially dangerous to the integrity of user data.

Rootkits can be used for either legitimate purpose, such as debugging or malicious purpose when combined with malicious software. There are two basic classes of rootkit which are classified based on the mode of operation, such as user-mode rootkits and kernel-mode rootkits. As the latter operates in the Windows kernel, they are more powerful than the former. As both could cause damage to the end user, malicious software defenders have created different tools to discover the presence of rootkits. In order to execute different pre-coded tasks, malicious software needs to perform some initial operations such as enumerating processes and services, opening a port, or establishing a network connection on the victim computer. A malicious rootkit can use either

user-space Application Programming Interface (API) hooking or kernel-space API hooks in order to remain hidden. Table 1 lists some important API function names which are targeted by malicious rootkit families to execute their operations.

Table 1. API functions hooked by malicious rootkits

Rootkit Malware	Hooked Functions
Zbot	ZwCreateThread
Win32/Cutwails	ZwOpenKey, ZwEnumerateKey ZwSetValueKey
WinNT/Omexo	ZwReadFile
Win32/Ursnif	CreateProcessA CreateProcessW
WinNT/Ramnit.gen!A	ZwOpenKey ZwCreateKey
Win32/Dorkbot	CreateFileA/W CopyFileA/W ZwEnumerateValueKey
Win32/Eyestye	ZwEnmerateValueKey

There are many different techniques [23] which have already been proposed to reveal a rootkit footprint.

(i) Signature based – This approach uses unique signature, i.e., the byte sequence of known rookits. Additionally, heuristics and behavior models based on certain actions are used to identify a certain family of rootkits.

(ii) Detecting detours – Windows operating systems are comprised of many important data structures such as Import Address Table (IAT), Export Address Table (EAT), and Interrupt Descriptor Table (IDT) in user-space and System Service Descriptor Table (SSDT) in kernel-space which allows a programmer to carry out task execution. Rootkits can either modify or alter these data structures to execute their own code. As a result, when there is a request for system related activities, it will always be executed by the detoured rootkit code.

(iii) Crosscheck approach – In order to decept rootkit presence, they may alter particular data when returned to the defender. Therefore, one of the common approaches to detecting rootkit traces is by comparing the results returned from a high-level system call and low-level system call.

Another distinguishing feature of this method is based on evaluating process memory with the content of a file stored on the hard disk. However, this method does not recognize precisely what was interrupted.

(iv) Integrity check – A digital signature is created using a cryptographic hash function when the system is installed with a clean operating system. Then, each library call is checked for code alternation.

(v) Alternative trusted medium - A rootkit can dynamically conceal its existence only when it is running. Therefore, the finest trustworthy technique for kernel-level rootkit revealing is to shutdown the infected computer and then checks its storage space by booting from an alternative trusted medium.

(vi) Memory dumps – In order to capture a live rootkit, this method analyzes the virtual memory of the underlying operating system in offline state using a debugging tool.

All the detection methods listed above implemented different techniques with the intention to assist the

defenders in ascertaining rootkit footprints. These techniques range from identifying for unique signature pattern in the impending malware sample to supervising system behavior. The important issue with live analysis is the authentic information such as files and functions returned by the Operating System (OS). Most existing anti-rootkit detection tools crosscheck information generated by tainted system calls against system information generated by its own for identifying rootkit traces. A stealthy malware conceals its footprints by controlling OS function calls which cannot be hidden. But, using offline investigation to reveal hidden traces of a malware is very difficult. In short, most existing techniques suffer from issues such as lack of integration, high false positive rate, overhead produced by complex configurations and scalability and performance issues.

In this paper, we proposed crosscheck-based comparison approach that aims to ascertain hidden processes and services concealed by stealthy malicious softwares. Our approach maintains a separate process list by continuously monitoring the underlying system that replicates the authentic running processes. Then, another list is maintained by dynamically analyzing lower-level processes and services and finally the two extracted lists of data are cross-checked to obtain hidden processes and services. The CoPDA has three key advantages over existing approaches. First, it is more flexible compared to typical process and service hiding techniques. Second, it utilized guest OS information implicitly to avoid guest evasion attacks. Third, our approach does not depend on the underlying operating system specific data structures; it is independent of hiding techniques exploited by Windows rootkits.

The rest of the paper is arranged as follows. Section 2 describes the related work and Section 3 presents background information about Windows stealthy malicious softwares. Section 4 presents the problem outline and Section 5 explains the aspects of our algorithm with pseudo code. The implementation environment is discussed in Section 6 and the performance evaluation is analyzed in Section 7. Finally, we conclude in Section 8.

2.0 LITERATURE SURVEY

A complex problem which forces the anti-malware software to run longer time is the huge quantity of data being generated by malware on a daily basis. The cross-view validation technique for detecting hidden traces of stealthy malware have been learned and implemented for testing user- applications [5], within the kernel of the OS [24], inside the virtual machine [25] and using coprocessor hardware techniques [15]. In order to detect hidden processes, Kumar et al. [5] have presented an approach to crosscheck two different process lists generated by calling higher-level user-mode APIs and lower-level APIs. The discrepancy between the two lists shows hidden processes. However, if the application were to be run by a limited user with no administrative privileges, it would fail to enumerate several system processes and threads, as well as fail to read memory pages of processes. Saur et al. [6] discussed an algorithm to locate paging structures of impending processes which are concealed by malicious software. Schuster et al. [7] developed a search prototype to scan an entire memory dump to reveal hidden or terminated processes and threads. Burdach et al. [8] described an approach to enumerate unseen processes. This approach is actually implemented in the Windows memory forensics toolkit. Betz et al. [9] developed a tool, called MemParser to enumerate the active running processes of the underlying operating system. Their tool can also dump the process memory. George et al. [10] has programmed Kntlist to analyze and evaluate kernel's internal data structures such as list and table to extract important processes,

threads, and other data. The Windows kernel keeps many tables and list to manage all of its resources. By inspecting them, we can catalog all items which are helpful for looking for malware footprint. However, this approach cannot detect kernel objects that are influenced by the OS and processes have already been terminated but not entirely erased from the memory.

As many rootkits adopt a hooking technique to hide their traces, Yin et al. [11] programmed HookFinder, a tool to find hooked activities of unfaithful binaries using fine-grained crash analysis approach. But, it has not been revised since 2008. Another similar tool, HookMap [12] has a utilized backward data segmentation technique to trace all possible memory addresses which can be abused by rootkits to embed hooks. The crosscheck view approach was used in GhostBuster [13] to detect rootkits by comparing two different sets of information, inside-the-box and outside-the-box. However, rebooting the OS during an external scan would produce a complexity overhead. Microsoft's RootkitRevealer [14] is a rootkit detection tool which generates two different lists of information in the same underlying system to reveal the presence of rootkits. However, it can detect persistent rootkits that can only hide files and registry-related settings in Windows operating systems. But it does not detect hidden processes and services. In [16], the authors proposed a reliable PCI-based kernel framework to expose rootkit existence. However, both approaches suffer from false negatives and also not being able to identify rootkits, which do not work in the kernel's memory. Baliga et al. [20] showed the feasibility of rootkits that modify low-level data within kernel objects to perform some malicious action, such as disabling a firewall or hiding a process. The concept of checking multiple views of processes running in the system has been explained in [21]. To detect hidden processes, they compare the process scheduling list to the list of all processes within the kernel. Though their discussion is on Linux, the same concept can be applied to Windows. Rootkit detection systems require inspection from outside the potentially compromised operating system. For example, virtual machine introspection [22] runs a security service within a privileged domain and uses memory introspection APIs exposed by the hypervisor to analyze the state of a guest system.

The key point of cross-view comparison that uniquely distinguishes these efforts is the method used to extract authentic lower-level kernel information of interest. Compared to other approaches, CoPDA is more flexible to typical process and service hiding technique and it utilized guest OS information implicitly to avoid guest evasion attacks.

3.0 BACKGROUND INFORMATION

"Rootkits: Subverting the Windows kernel" [2] defines a rootkit as a program that ensures persistent, robust, and undetectable presence in the computer. Though the first rootkit was designed with the intent of attacking the Unix OS, Windows rootkits are very popular today. The first creation of rootkits tried to modify a few system utilities with Trojanized binaries on the victim computer. As they modify files on the disk, they can easily be detected by calculating the digital signature of the file using cryptographic hash functions such as Tripwire [3]. In the next generation, rootkits infect process memory. In order to evade detection, they try to divert the predefined execution path of a system call. A rootkit has many places to inject malicious code in the underlying operating system. One such popular attack is an IAT hook in user-mode and SSDT hook in the Kernel-mode. VICE [4] is a rootkit detection tool to identify these kinds of hooks. The next generation of rootkit runs at ring 0 with highest privilege. They are designed to insert into device drivers and can directly affect kernel objects

based on Direct Kernel Object Manipulation (DKOM) [26]. This technique involves altering kernel objects manipulated by the OS in memory. The hypervisor is a micro kernel which is responsible for controlling the distribution of resources between multiple OS. More recent rootkits can intercept a hardware request of the OS to launch their illegal activities.

Though all rootkits are integrated with different techniques, their main intention is to hide their traces and stay undetected for an extensive period of time. Bearing this common character in mind, we have devised a crosscheck-based method to categorize hidden traces of a rootkit in the user-space.

4.0 PROBLEM OUTLINE AND ASSUMPTION

From the survey we have done, we assume that almost all malware hide their footprints such as processes and services rather than other kinds of resources such as the registry, files, etc., to stay undetected and perform their illegal activities for a longer time. Therefore, we focus on locating these footprints to stop them before causing damage. This crucial point is mainly used to develop CoPDA effectively. As many data guard programs might use techniques such as encryption or access control to hide files locally, we do not consider this issue.
Rootkits may hide their traces and activities either from the user or antivirus software. We have taken this vital point to determine whether the underlying operating system includes hidden rootkit traces or not. The hidden character of a rootkit can be formalized as follows.

Let $G_a(t)$ be the set of system-wide objects of type *a* at time *t*, and $V_a(t)$ be the set of visible objects of type *a* at time *t*. Let o be an object of type *a*. At time *t*, if thereexist o such that $o \in G_a(t) \wedge o \notin V_a(t)$, then o is caught as hidden at time *t*. Otherwise, if $(G_a(t) \wedge V_a(t) = \phi)$ then the underlying system is in a stable state (with respect to *a* at time *t*). Let A represent all the available objects in the underlying system. Then,

$G = \bigcup_{a \in A} G_a(t)$ and $V = \bigcup_{a \in A} V_a(t)$ where G and V are the global view and visible view of objects in the underlying system, respectively. If any object is in the global view, but not in the visible view, then it is caught as a hidden object. As a result, by comparing G and V, we can identify and list hidden objects in the target computer.

5.0 PROPOSED ALGORITHM

The proposed algorithm, CoPDA, is a practical tool that point out the user-level hidden rootkit processes and services. In order to produce hard evidence about the hidden processes and hidden services of a malicious rootkit, it is mandatory to have a global view and visible view of objects in the core system. CoPDA is a crosscheck-based approach to discover and listing hidden processes and services by comparing the global view and visible view. The pseudo code of the proposed algorithm is given in Algorithm 1.

Algorithm. Concealed Process and Service Discovery
Output: Hidden processes and services

```
Function1 getVisibleviewlist()
processEntry32 e; wchar_t c, rs;
handle s;
begin
        s←CreateToolhelp32Snapshot();
            if(Process32First(s&e) == TRUE)
       do
           pcount←ProcessIds;
           wcsncpy(ProcessName[pcount],e);
            pcount++;
               if(bTryOpen)
            handle h=OpenProcess();
               endif
             wcsncpy(e,L"csrss.exe");
               wcsncpy(rs,L"services.exe");
            if(wcscmp(e,c)==0)
                cId=e.th32ProcessID;
            endif
             if(wcscmp(e,rs) ==0)
                sId=e.the32ProcessID;
             endif
      while(Process32Next(s,e)==TRUE);
                endif
end
endfunction1

Function2 getGlobalViewlist()
begin     s←NtQuerySystemInformation(SystemhandleInformation, HandleTable,dwsize,0)
   *hInfo=( SYSTEM_HANDLE_INFORMATION *) HandleTable;
    Handle cs=OpenProcess(PROCESS_ALL-ACCESS,GetCsrssProcessId());
    for (int i=0;i<hInfo→ListedHandle;i++)
    int pId=hInfo→Handles[i].uniqueProcessId;
    if(hprocess)
    begin     if(Dhandle(hprocess,(HANDLE)hInfo→Handles[i].cs,GetCurrentProcess(), &tProcess,
        PROCESS_QUERY_INFORMATION,FALSE,0))
        begin
            tpId=GetProcessId(tprocess);
        end
```

end
endfor
endfunction2

Many advanced stealthy rootkits adopt different methods to hide their traces and evade detection. One of the methods user-mode rootkits can use to hide their detection is to hook a native API function and filter its traces. A Windows process will open many handles associated with a process which can be used to detect hidden processes. One way to enumerate handles is to scroll through the process handles of CSRSS.EXE which holds the handles to all running processes.

Furthermore, in Windows operating system, services can only run with highest privilege mode. As a result, rootkits integrate services not only to keep control over the entire system, but also to manage its tasks and to remain undiscovered. Since services are visible to the end user, it is important for a rootkit to conceal its services to prevent itself from being detected. CoPDA has the ability to locate all hidden services by extracting information from Services.exe which is an important process in Windows operating systems for revealing rootkit footprints.

A list of the global view (G) is generated which contains the handles to all running processes by going through the process handle of the CSRSS.EXE process of the physical operating system. As rootkits hook normal process enumeration functions, the finest way to identify all processes is to make use of the NtQuerySystemInformation function. In order to produce a list of the visible view (V), we generate a list of all running processes by taking a snapshot of currently running processes in the system using CreateToolhelp32Snapshot in the guest operating system with which we try to discover hidden rootkit processes and services. Then the child processes that are manipulated by Services.exe are separated. Finally, the entries in the global view are compared against the entries in the visible view. Any difference in these two data lists reveal hidden traces of malware.

6.0 IMPLEMENTATION ENVIRONMENT

This section describes the implementation environment settings and malware samples in use for assessing the accuracy and performance of CoPDA. Our experiments were conducted on a physical machine built with an ACER Core Duo 2.93 GHz CPU and 2 GB RAM which was running Microsoft Windows 7. The guest operating system runs Microsoft Windows XP SP1 with no service pack. We have implemented CoPDA with 77 lines of C++ code using the Microsoft Visual Studio 2008 development environment.

Different anti-rootkit detectors are opted for in to verify the strengths of the proposed algorithm. Table 2 summarizes a description of the version and revealing techniques of various anti-rootkit detection tools which were used for the evaluation procedure. The hidden rootkit samples were executed and the detection capabilities of each anti-rootkit detection tool were noted. A few notable rootkit malware samples are listed in Table 3.

Table 2. Anti-Rootkit detection tools characteristics including CoPDA

Sl.No	Tool Name	Version	Hooking	Cross-View
1	GMER	1.0.15.15281	Yes	No
2	HeliosLite	1.005	Yes	Yes
3	IceSword	1.22en	Yes	Yes
4	Rootkit Unhooker	3.8.388.480 SR2	Yes	Yes
5	HiddenFinder	1.5.6	Yes	Yes
6	RootkitRevealer	1.7	Yes	Yes
7	BlackLight	2.2.1046	Yes	Yes
8	CoPDA	--	Yes	Yes

Table 3. Rootkit malware samples

Rootkit malware	Hiding Technique
AFX Rootkit(2005)	API Hooking
Hack Defender	API Hooking
Vanquish	DLL injection & API Hooking
FU	API Hooking
FUTo	DKOM

The evaluation and validation of the CoPDA was done by considering both legitimate and malicious processes and services. Without legitimate processes and services, the total number of false alarms generated by CoPDA cannot be identified. Therefore, we have considered few legitimate samples for validating the effectiveness of the CoPDA.

7.0 PERFORMANCE ANALYSIS

To study and get better insight about different malware families, we collected 1000 malware samples that belong to different infecting technique such as virus, worms, Trojans, rootkits and backdoors. We carefully read, analyzed and recorded its technical details. Finally, we have taken a dataset which consists of 100 malware samples [17], [18], [19] that mainly utilized API hooking, registry hiding and process hiding technique. This study helped us to design our CoPDA to react against unknown malware. In order to determine the effectiveness of CoPDA in detecting hidden rootkit footprints, mainly, hidden processes and services, different samples with cross validation technique is used. The dataset is divided into ten groups – a group on the average contains 10 malwares and 9-10 benign samples. Then the tools are trained on nine groups and the remaining one group is used for testing them. The results after testing each group and its average results are tabulated in Table 4.

7.1. Overall Detection Accuracy of CoPDA

From the usability point of view, allowing CoPDA to generate a low false alarm rate is an important issue; otherwise CoPDA will not be competitor against existing anti-rootkit detection tools if it stopped legitimate applications frequently. CoPDA identifies malicious hidden rootkit process and service by generating and matching two dissimilar views. The creation of the visible view is manipulated in the user-space of the guest operating system. Then, the production of the global view and the matching of the two views are completed in the physical operating system. Before talking about the final results of the research, we wish to point out the metrics that have been used for evaluating CoPDA.

- True Positive (TP) - It occurs when CoPDA correctly detects a hidden process or service.
- False Positive (FP) - It is computed as the ratio between the number of legitimate entries which are misclassified as hidden and the total number of legitimate entries.
- The Precision Rate (PR) and Overll Detection Accuracy (DA) are calculated as follows.

$$\text{PR} = \frac{TP}{TP + FP}$$

$$\text{DA} = \frac{TP + TN}{TP + TN + FP + FN}$$

- F-Measure – It denotes the measurement of a test's accurateness by adding recall value and precision value into a single measure of performance. Normally, the result closer to 100% is good. F-Measure is computed as follows.

$$F - Measure = 2 * \frac{PR * \text{Re}call}{PR + \text{Re}call}$$

Table 4. Performance Metrics Calculation

Tool Name	Weighted Average (%)					
	TPR	FPR	PR	Recall	F-Measure	DA
Hellioslite	88.14	5.93	93.67	88.14	90.76	91.03
GMER	85.53	6.00	94.33	85.53	89.45	89.42
HiddenFinder	85.52	8.13	91.98	85.52	86.82	86.83
RootkitRevealer	0.0	0.0	0.0	0.0	0.0	0.0
IceSword	96.18	3.82	96.18	96.18	96.08	96.12
BlackLight	96.0	3.0	97.27	96.0	96.47	96.50
Rootkit Unhooker	84.53	7.55	91.85	84.53	87.95	88.50
CoPDA	100	1.82	98.09	100	98.99	99.02

As, we can see from Table 4, all anti-rootkit detection tools achieved different detection accuracy rate, i.e. detecting hidden processes and services, except RootkitRevealer, with different FPR. We realized that CoPDA achieved an average of 98.09% PR with 99.02% DA of locating hidden processes and services. As few benign applications imitate operations performed by a malicious software, CoPDA produced 1.82% FPR. When restarting the system, CoPDA correctly identified all hidden processes and services with no false alarm. Overall, CoPDA had outperformed all other anti-rootkit detection tools in all measures. The feature supplied with Receiver Operating Characteristic (ROC) is commonly used to evaluate the performance of malware detection

tools in computer security. It is especially used by researchers to understand how detection accuracy changes with respect to different false alarm rate.

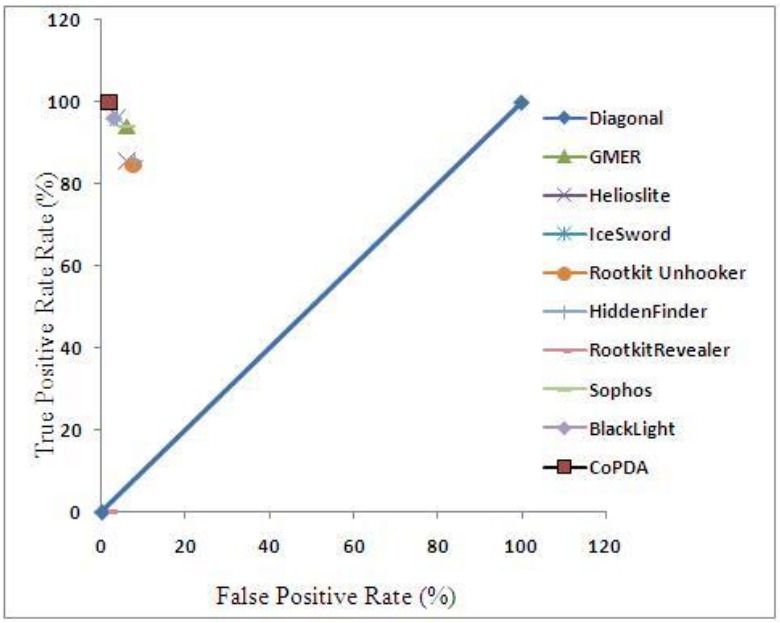

Fig 1. Realization of ROC Curve

Figure 1 depicts the generated ROC curve of eight anti-rootkit detection tools including CoPDA. As the curve bypasses the upper-left region, CoPDA achieved 100% TPR, indicated that CoPDA offers the best result among the other anti-rootkit detection tools.

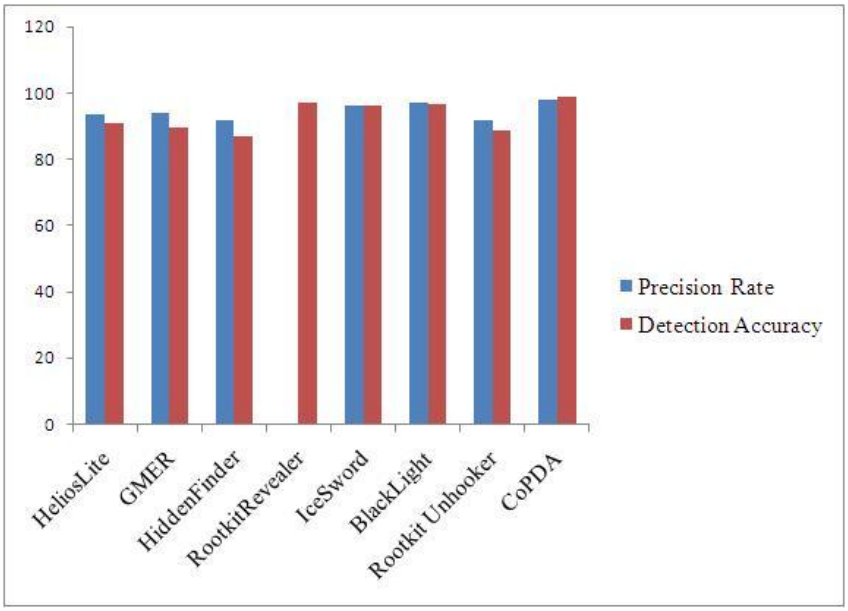

Fig 2. Comparison of Precision Rate and Detection Accuracy

From Figure 2 GMER scans for hidden processes, services, threads, files, as well as the detection of several different types of hooks. In our test, it had a precision rate of 94.33%. One important downfall of GMER is that it cannot allow a user to perform any other task while it is scanning the target systems. HeliosLite utilizes the cross-view approach to detect rootkits and can be executed from a USB drive. Against the rootkit samples that were taken for tests, it achieved a precision rate of 93.67%.

IceSword scans the entire system to categorize hidden processes and services, files, registry settings, ports and startup items. IceSword spotted all the processes and services hidden of all the five rootkit families. It produced 100% precision as it detected all the hidden processes and services. However, while reacting to FU and FUTo rootkit malware, IceSword was unable to discover the rootkit. One important limitation of IceSword is that it only works effectively within the Windows XP OS, and fail to function in another environment. By scanning the entire target system, the BlackLight anti-rootkit tool identified hidden processes when executed against FU and FUTo rootkits. But fail to discover the rootkit. However, BlackLight achieved good results, detecting all the rootkits in our experiments.

Rootkit Unhooker detects hidden processes, and files, but fail to discover hidden services and registry key values. As a result, it produced a 88.5% detection rate. HiddenFinder also uses cross-view technique to identify hidden processes, but only attained a 91.85% of the precision rate although it has not been revised since 2008. RootkitRevealer was the first rootkit detection application that used cross-view mechanism to detect persistent rootkits that can only hide files and registry-related settings in Windows operating systems although it does not detect hidden processes and services. Therefore, it has no impact on the tested rootkit samples. RootkitRevealer recorded no inconsistency when performing against FU and FUTo rootkit samples since they only hide processes. As it was not updated periodically, it failed to detect modern rootkit malware. CoPDA produces a total high detection precision rate of 98.09%. No other tested anti-rootkit detection tools achieved better TPR, FAR, PR and DA than CoPDA.

7.2. Detection through Hindrance

We evaluate the timeliness and precision of CoPDA when spotting a single hidden process. When the victim computer contained more than one hidden process, the discrepancy between the global view list and visible view list is larger which leads to much easier detection. Therefore, enabling CoPDA to detect a single hidden process implies an unconditional detection state. We utilized the synthetic process creator which is a tool for procreating processes randomly. In [27], the authors point out that process appearances are exploding than obsession. Hence, we select a Pareto distribution with k=1 (shape metric) for process inter-landing time. By varying the Pareto locality metric, we stabilize the average rate of process creation which is directed to sizable process creation. This dissemination makes the detection process difficult. We set a process lifetime interval target between 0 and 1 second based on uniform process distribution to keep the test system in a stable state. To keep hiding processes in Windows operating system, we made use of the rootkit malware fu.exe that hides its traces by directly modify a process token used by the OS. We also use a guest process reporting tool to fabricate hidden processes in Linux operating systems.

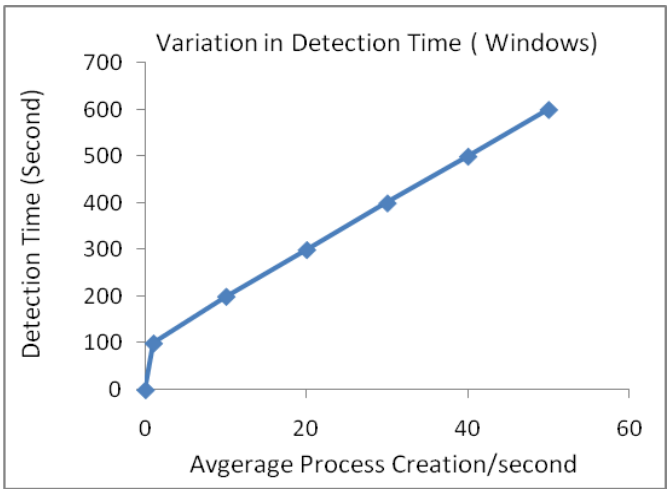

Fig 3 (a). Variation in hidden process detection time depends on the process creation activity in Windows.

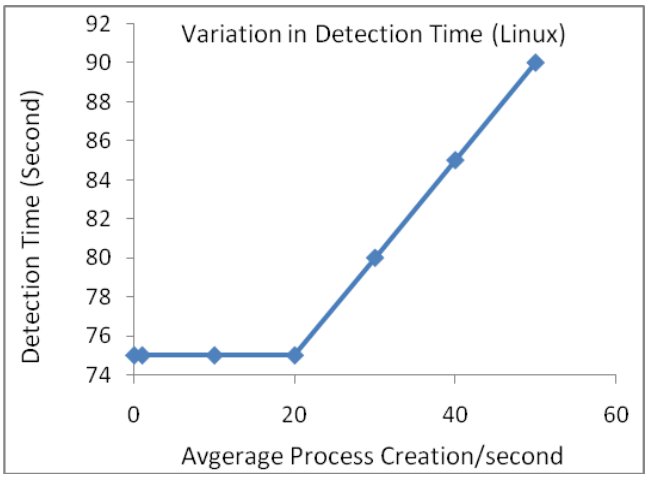

Fig 3 (b). Variation in hidden process detection time depends on the process creation activity in Linux.

We measure process creation time for different sets of process activity levels in Windows operating systems and Linux operating systems, the same is given in Figure 3 where the x-axis represents different process activity levels and the y-axis represents the detection time. Figure 3 also states that each process creation and depart action increases the detection time. When the legitimate process consignment is reasonable, we can accurately spot the single malicious rootkit process every time. However, we can observe a bigger discrepancy in detection time under intense process creation that introduces false positives. However, CoPDA reduces the result of false positive identification.

7.3. Runtime Overhead and Limitation

While CoPDA runs continuously for detecting hidden processes, we must ensure that whether it produces significant runtime overhead. In order to assess the runtime overhead of the prototype, we executed three benchmarks in Windows and in the Xen hypervisor including CoPDA. We take five test samples and list its

average in Table 5. The CreateProc is capable of creating and destroying 1000 processes rapidly. The MemAlloc allots a 200 MB fragment of memory and performs page table switching. CoPDA generates a 5.1 % runtime overhead for CreateProc benchmark and a 3.5 % runtime overhead for MemAlloc. For the Compile benchmark which includes the bash shell source, CoPDA causes a tiny runtime overhead of 0.6%.

Table 5. Runtime overhead detection

Name of the Benchmark	Runtime in seconds		% of overhead
	CoPDA	Xen	
Compile	21.164	21.041	0.6
CreateProc	5.601	5.303	5.1
MemAlloc	5.523	5.220	3.5

The CoPDA was tested and tweaked principally around 32-bit Windows operating systems and produced positive outcomes. CoPDA is not tested against malware samples that target the Windows 64-bit operating systems and x86 platforms. Secondly, the proposed CoPDA does not work in kernel-mode, when a kernel-mode hook attack directly invokes ntoskrnl.exe. Identifying and preventing such attacks is a challenging task which we aim to address our future work.

8.0 CONCLUSION AND FUTURE WORK

Many modern malicious computer software has been integrated with the rootkit technique to evade rootkit detection tool and anti-virus software. Therefore in-depth research into prevention of stealthy malicious code attacks is absolutely essential. In this paper, we devised CoPDA, a practical tool that runs in the user-space of Windows systems to discover hidden rootkit traces. Our approach maintains a separate process list that contains all running processes and services by continuously monitoring the victim computer. Then, another list is generated by dynamically analyzing lower-level processes and services and then the two lists of data are cross-checked to discover hidden processes and services. The experimental results show that all tested rootkits were successfully identified by CoPDA with a 99.02% detection accuracy. The proposed algorithm not only detected user-mode hidden processes and services but also some of the rootkit processes which adopt kernel-mode rootkit techniques without taking care of hidden process handles within the CSRSS.EXE process. We thus conclude that the proposed prototype can reliably detect malicious rootkit software and thus it represents a useful tool for the detection of hidden processes and services. In the future, we plan to design a kernel level process authentication mechanism to validate the originality of every suspected hidden process to prevent malicious code attacks from misusing system resources.

REFERENCES

[1] A. Shevchenko, “Rootkit evolution,” 28 August 2008. [Online]. Available: http://www.securelist.com/en/analysis/204792016/Rootkit_evolution.

[2] G. Hoglund and J. Butler, Rootkits: Subverting the windows kernel. Addison Wesley, Boston, USA,

2006.

[3] Tripwire. [Online] Available: http://www.tripwire.com/

[4] J. Butler and G. Hoglund, "VICE - Catch the hookers! (Plus new rootkit techniques)", Black Hat USA 2004 Conference, Las Vegas, USA, 2004.

[5] Eric Uday Kumar,"User-mode memory scanning on 32-bit & 64-bit Windows", *Journal in Computer Virology*, vol.6, pp.123-142, May 2010.

[6] Karla Saur, Julian B.Grizzard,"Locating x86 paging structures in memory images", *Digital Investigation*, vol.7, issue1-2, pp.28-37, 2010.

[7] Andreas Schuster,"Searching for processes and threads in Microsoft Windows memory dumps", *Digital Investigation*, vol.3, pp.10-16, 2006.

[8] Burdach Mariusz, An introduction to Windows Memory Forensic, July 2005.

[9] Betz Chris. MemParser, August 2005. [Online]. Available:http://www.dfrws.org/2005/challenge/memparser.html

[10] Garner George M, Mora Robert-Jan. Kntlist, August 2005.[Online].Available: http://www.dfrws.org/2005/challenge/kntlist.html

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

[12] Z.Wang, X.Jiang, W.Cui, X.Wang,"Countering persistent Kernel Rootkits through Semantic Hook Discovery", *in Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection, 2008, pp.21-38*.

[13] Y.M.Wang, D.Beck, B.Vo, R.Roussev, C.Verbowski,"Detecting Stealth Software with Strider GhostBuster", *in Proceedings of the 2005 International conference on Dependable Systems and Networks, 2005, pp.368-377*.

[14] Windows Sysinternals RootkitRevealer. [Online]. Avilable:http://technet.microsoft.com/bb897445.aspx.

[15] J.Nick L. Petroni, T.Fraser, J.Molina, W.A.Arbaugh,"Capilot – a Coprocessor-based Kernel Runtime Integrity Monitor", *in Proceedings of the 13th conference on USENIX Security Symposium, 2004, vol.13, pp.13-13*.

[16] J.Nick L.Petroni, T.Fraser, A.Walters, W.A.Arbaugh,"An Architecture for specification-Based Detection of Semantic Integrity Violations in kernel Dynamic Data", *in Proceedings of the 15th conference on USENIX Security Symposium, 2006, vol.15, Article No.20.*

[17] [Online]. Available: http://www.rootkit.com/

[18] VX Heavens, [Online]. Available: http://vx.netlux.org/

[19] Xfocus Team, [Online]. Available: http://www.xfocus.net/

[20] Baliga, Arati, Pandurang Kamat, and Liviu Iftode,"Lurking in the Shadows:Identifying Systemic Threats to Kernel Data", *in Security and Privacy Symposium Proceedings, IEEE, 2007, pp.246-251.*

[21] Baliga, Arati, Vinod Ganapathy, and Liviu Iftode,"Automatic Inference and Enforcement of kernel Data Structure Invariants", *in Procedings of the Annual Computer Security Applications Conference, IEEE, 2008, pp.77-86.*

[22] Garfinkel, Tal, and Mendel Rosenblum. "A Virtual Machine Introspection Based Architecture for Intrusion Detection", *in Proceedings of the Network and Distribued Systems Security Symposium, 2003.*

[23] J.Butle, S.Sparks,Windows rootkits of 2005: Part three. [Online] Available: www.securityfocus.com/infocus/1854

[24] Leian Liu, Zuanxing Yin, Yuli Shen, Haitao Lin, Hongjiang Wang," Research and Design of Rootkit Detection Method", *Physics Procedia*, vo.33, pp.852-857, 2012.

[25] Yan Wen, Jinjing Zhao, Huaimin Wang," Implicit Detection of Hidden Processes with a Local-Booted Virtual Machine", *International Journal of Security and Its Applications*, vol.2(4), 2008, pp.39-47.

[26] Woei-Jiunn Tsaur, Yuh-Chen Chen, Being-Yu Tsai, "A New Windows Driver-Hidden Rootkit Based on Direct Kernel Object Manipulation", *in Proceedings of the 9th International conference, 2009, pp.202-213*.

[27] M.Harchol Balter and A.B.Downey, "Exploiting process lifetime distribution for dynamic load balancing", *ACM Transactions on Computer Systems*, vol.15(3), 1997, pp.253-285.