

PPSAM: Proactive PowerShell Anti-Malware

Customizable Comprehensive Tool to Supplement Commercial AVs

Alejandro Villegas and Lei Chen

Department of Computer Science, Sam Houston State University, Huntsville, TX 77341, USA

Abstract - *This research first explores the different types of Anti-Malware solution approaches, evaluating the pros and cons, and concentrating on their potential weaknesses and drawbacks. The malware technologies analyzed include Windows Direct Kernel Object Manipulation (DKOM), Kernel Patch Protection, Data Execution Prevention, Address Space Layout Randomization, Driver Signing, Windows Service Hardening, Ghostbuster, Assembly Reverse Analysis, and Virtual CloudAV. Furthermore, a proactive comprehensive solution is provided by utilizing the Windows PowerShell 2.0 utility that is available for Windows Vista, 7, 2008 and 2008 R2. The proposed Proactive PowerShell Anti-Malware (PPSAM) is a utility that monitors the system via health checks with shell scripts that can be fully customized and have the ability to be executed on remote systems. PPSAM is designed to be a proactive complement that attempts to promote early discovery of intrusions and malicious applications, and to provide triggers and reports utilizing the scripts' output.*

Keywords: PowerShell, malware, anti-virus, proactive, customizable, security

1 Introduction

Majority of end users already have a preferred Anti-Virus (AV) solution such as Windows Defender, McAfee or Norton. In the meanwhile, anti-rootkits like VICE, GMER, and Rootkit Unhooker [8] have become fairly popular. Most of the anti-malware products take a reactive, rather than proactive, approach to detection. The first and most common strategy is to compare applications with common malware signatures stored in a database and flag them as suspicious malware, whether a virus, Trojan or rootkit. Malware will continue to evolve and make AV applications obsolete is simply the nature of the game.

Some computing essentials remain the same for all different types of malware: for instance, the best malware is the one that goes undetected (at least until is found or discovered either manually or by a new AV tool). Unfortunately, when a virus becomes too noticeable or by the time is discovered it probably has already caused a lot of

damage. Average end users could have a rootkit installed and don't even realize so, as they rely on the anti-virus to detect the compromise, or expects a call from the system or network administrator reporting "unusual" activities. In addition, hackers are well aware of major AV companies too, and they are used to code malware that detects AVs and formulate a scripting workaround to avoid detection. If the AV or Anti-Malware solution is not even able to detect the illicit computing transaction, it becomes useless.

The rest of the paper is structured as follows. Next in Section 2, we survey the existing vulnerabilities that can be used by malware to hide themselves in the system. Section 3 discusses the motivation of this research, followed by the technical details of PPSAM in Section 4 including four levels of script repository, differences from existing solutions, and hardware and software requirements. Section 5 shows a few examples how PPSAM can work along with the AVs to help monitor the system. We draw conclusion in Section 6 and propose future work in Section 7.

2 Background

The research performed by Woei-Jiunn Tsaur [8] clearly exposes five potential vulnerabilities that rootkit developers can exploit to maintain their applications undetected.

2.1 Windows DKOM

A lot of the current research focuses on kernel data schemes that aim to detect hooking driven virtual machine rootkits. Nevertheless, DKOM has been proven to be strategy inefficient [8]. Woei-Jiunn Tsaur et al. in [8] go beyond analyzing the traditional rootkits that typically are traceable within registry keys, questionable drivers or malicious API injections [4]. DKOM style rootkits exploit the kernel object implementation in Windows systems by altering EPROCESS objects [8]. The DKOM detection approach by Woei-Jiunn Tsaur et al. is to install a hidden driver as an object in order to detect DKOM activities. The proposed rootkit named hookzw.sys is a driver format (composed using Borland TASM 5.0) which executes on the Windows XP SP2 platform. One of the DKOM rootkit

techniques is to exploit 'PsLoadedModuleList' in order to hide processes. The core data structures that are modified by a DKOM rootkit are: List_Entry data structures of Object Directory, Object Driver, Object Device and PsLoadedModuleList [8]. Woei-Jiunn Tsauro et al. outline the following five tips to detect DKOM rootkits: Removing Object Drivers and Object Devices from Object Dir, Removing Object Drivers from Driver Object_Type, Removing Object Devices from Device Object_Type, Removing Drivers from PsLoadedModuleList, and Altering Object Driver Appearance.

There is no doubt that DKOM rootkits are a threat, and a comprehensive tool for its prevention needs to be developed by a commercial Research and Development (R&D) entity. Furthermore, this approach is designed to discover unknown rootkits on the wild and does not necessarily provide a mechanism to create a signature based database for further detection.

2.2 KPP, DEP, ASLR, DS, and WSH

There is a plethora of different third party rootkit detectors for the Windows platforms such as VICE, GMER, Rootkit Unhooker, among others [8]. Nonetheless, Microsoft introduced five different software utilities to fight rootkits and malware in general for Windows Vista and newer versions [1]. These five utilities are briefly introduced as follows [1]:

- Kernel Patch Protection (KPP) – provides protection of the Windows kernel (formerly known as PatchGuard) at the System Service Descriptor Table (SSDT) level; prevents malware from hooking into system APIs.
- Date Execution Prevention (DEP) – deals with Buffer Overflow prevention.
- Address Space Layout Randomization (ASLR) – randomly arranges the positions of key data areas in a process's address space.
- Driver Signing (DS) – signs legit drivers in order to prevent the installation of new malicious drivers.
- Windows Service Hardening (WSH) – increases restriction to Windows background process.

Albeit the mentioned solutions have made the Windows operating systems more secure and stable, there are exploits such as DKOM among others that still represent a threat to the Windows kernel, not to mention that malware keeps evolving, and Windows only utilizes two of the four Intel's architecture layers leaving the OS still vulnerable [1].

2.3 Strider Ghostbuster

The Strider Ghostbuster [9] is a cross-view difference based approach Ghostware detector. The technique utilized

compares a high-level infected scan with a low-level clean scan. Furthermore, it runs an inside-the-box versus an outside-the-box clean scan [9].

The Strider Ghostbuster approach is essentially a differentiation of potentially infected systems with a known clean one. The term "Ghostware" refers to programs such as rootkits and Trojans with stealth hiding capabilities [9]. The main areas where Ghostbuster runs its diff approach is in the "Master File Table", the "Raw Hive Files", and the "Kernel Process List" [9]. Common Ghostware detected by Ghostbuster includes Urbin, Mersting, Vanquish, Aphex, etc. [9].

While Ghostbuster is considered a good approach to determine whether the files/registry values/drivers were modified or altered, there are some rootkits like DKOM that may bypass the diff check. In addition, it utilizes a lot of system resources when handling the system components comparison. Ghostbuster also has the capability of providing a Virtual Machine (VM) in order to run diff check scans on virtual environments [9]. It may be a good solution with questionable scalability when it comes to large IT environments.

2.4 Assembly Reverse Analysis

The Assembly Reverse Analysis is essentially backward engineering the rootkit assembly code, utilizing tools such as MASM, ASM, and TASM [10]. The Windows debugger Ollydbg is also useful to analyze the content of a given malware [10]. This approach is for expert computer users who are able to decompile the malware/rootkits, analyze the contents and restore the system to its original stage as applicable. It is a very hands-on strategy and does not provide the users with a sustainable solution, which varies on a case by case basis and is not recommended for the average end users.

2.5 Virtual CloudAV

Cloud computing has revolutionized the deployment of hardware infrastructure. There are different cloud solutions such as Amazon's Web Services and Google's AppEngine. The concept of Infrastructure as a Service (IaaS) refers to offering access to remote computer resources [5]. The architecture of this virtual cloud solution consists of running a Kernel Agent that gathers information from each virtual machine and passes the data into a ProxyScan that analyzes the data and seeks for potential malware or kernel rootkits [5]. It is considered an excellent solution for cloud providers as they typically grant root/administrator access to their VM clients, yet remain liable for their actions as they own the hardware infrastructure. Implementation with similar functionality can be costly for small and medium businesses (SMBs).

There is also a behavior based approach in [4] that hooks Native APIs in the kernel mode, however it does have an impact on system performance in addition to the standard AVs system utilization.

3 Motivation

In this research, we decided to develop a boutique style anti-malware tool that would address the key drawbacks of each of the algorithms researched, yet comprehensive, scalable and intuitive. The goal of this research was to target newer Windows operating systems such as Vista and 7. The proposed solution PPSAM takes advantage of the Windows PowerShell supplied on newer Windows Operating Systems to craft simple scripts that monitor suspicious activity on a given system.

PowerShell (PS) has the capability of running commands on remote systems, therefore is ideal for scalability purposes. In addition PowerShell can be utilized to create scripts and functions by combining different cmdlets [7]. PPSAM is a starting point solution that can be customized with additional PS scripts and actualize them as malware evolves. The main purpose is to keep this solution relatively obscure in order to prevent automatic detection by malware components. The script can be located in random locations, named differently, and executed on different manners. Ideally it can be scheduled to run periodically (without user intervention), create reports and flag them accordingly with triggers designed to catch suspicious activities within the system(s). The suggested method to run PPSAM is to execute it from a remote system to avoid affecting performance and ensure that the source system is “clean”.

Before the discussion of the implementation and use of PPSAM, here we list the key drawbacks of each of the algorithms researched:

1. DKOM: potential security holes not currently exploited. Solution algorithm requires corporate level R&D investment.
2. Microsoft Security Implementations: Efficient. However the Windows platform only utilizes two of the four Intel’s architecture rings. OS still exposed to other potential malware/exploits.
3. Strider Ghostbuster: excellent diff approach; requires a lot of resources and limits scalability.
4. Assembly Reverse Analysis: ideal for a malware research lab, not practical for average end users.
5. Virtual CloudAV: comprehensive anti-malware setup for virtual cloud environments. It requires hardware investment and extensive configuration, optimal for ISPs not a promising solution for SMBs.

4 PPSAM

The Windows PowerShell [6] is native to the Operating System and therefore is able to interact with the OS and Microsoft applications flawlessly. PPSAM is designed to operate from the command line via the default PS cmdlets.

4.1 Levels of script repository

The script repository is divided in four levels: Registry, Network, Driver, and Application. These four levels and the corresponding Cmdlets [6] are introduced as follows.

- **Registry Level**

PowerShell cmdlets that interact with the registry:

- **Get-Item** – get a file/registry object (or any other namespace object)
- **Get-ChildItem** – get child items (contents of a folder or reg key)
- **Get-Acl** – get permission settings for a file or registry key
- **Get a registry key** – PS `C:\>get-item hklm:\software\microsoft\exchange`

- **Network Level**

PowerShell cmdlets that retrieve mac-addresses:

- `$strComputer = "."`
- `$colItems = get-wmiobject -class "Win32_NetworkAdapterConfiguration" ` -computername $strComputer | Where{$_.IpEnabled -Match "True"} foreach ($objItem in $colItems) { write-host "Hardware Address:" $objItem.MACAddress}`

- **Driver Level**

PowerShell cmdlets that retrieve drivers:

- **Get-WmiObject -Class Win32_SystemDriver / Format-List Name, Caption, Description, InstallDate, PathName, Started, StartName, Status, SystemName**

- **Application Level**

PowerShell cmdlets that verify exchange health:

- **Test-ServiceHealth** – tests if all required services have started successfully
- **Test-SystemHealth** – gathers data about Microsoft Exchange system and analyzes the data according to best practices
- **Test-UMConnectivity** – tests the operation of a computer that has the Unified Messaging (UM) server role installed
- **Test-WebServicesConnectivity** – tests the functionality of Exchange Web Services

PPSAM takes advantage of the comprehensive utility availability of PowerShell in order to run health checks at different levels and monitor the system for suspicious

activity. PPSAM utilizes a collection of PowerShell scripts fully customizable that can be easily executed and formulate reports in HTML and/or XML format for easy viewing.

4.2 Differences from existing solutions

PPSAM is a simple utility that is instrumental to perform proactive malware scans and flag suspicious activity that could have been overlooked by a commercial AV. PPSAM can be customized and deployed to several systems. While it does not replace traditional AVs or Anti-Rootkit programs, it is designed to complement them. PPSAM PowerShell architecture makes the scripts executable natively without third party application installation requirements. In addition, the code is transparent and there is no need to install malware freeware that might be facilitated from web sites that include malware along with the utility. It is a middle man solution – it might not be as thorough as debugging with Ollydbg or MASM [10], but has enough capabilities to detect malicious activity in a system(s) that could have bypassed a traditional AV.

4.3 Software and hardware requirements

Software Requirements:

- Windows Vista, 7, 2008 or 2008 R2
- Windows Framework Management (Installed by default in Windows 2008 R2 and Windows 7)
- PowerShell 2.0
- WinRM 2.0
- BITS 4.0

Hardware Requirements:

- 1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64) processor
- 1 gigabyte (GB) RAM (32-bit) or 2 GB RAM (64-bit)
- 16 GB available hard disk space (32-bit) or 20 GB (64-bit)
- DirectX 9 graphics device with WDDM 1.0 or higher driver

5 Performance and analysis

This section shows a few examples of how PowerShell cmdlets can be utilized to help AVs monitor the system. The PowerShell has a myriad of different cmdlets that make interacting with a Windows platform smoothly and flawlessly. The get-itemproperty command is useful to obtain the values of a specific registry key as shown in Figure 1. This command can be combined with the invoke-command in order to be executed on a remote system. The PowerShell uses a similar approach than the Linux BASH shell, where the scripts can be coded on a simple text editor such as notepad, saved with the .ps1 file extension and can then be executed accordingly. They can also be loaded via a .bat (batch file). The registry key HKLM:\SOFTWARE\

CLASSES\CLSID\{2781761E-28E1-4109-99FE-B9D127C57AFE} queried in Figure 1 shows that the system is running the Microsoft Malware Protection IOfficeAntiVirus Implementation.

The registry key in Figure 2 shows that Windows Defender is installed as the default AV solution HKLM:\SOFTWARE\CLASSES\CLSID\{2781761E-28E0-4109-99FE-B9D127C57AFE}.

Figure 3 shows the output from the PowerShell get-process * | more command, essentially showing all the running processes on the system. It can also be combined with the invoke-command to be executed remotely.

```

PS HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES\CLSID\{2781761E-28E1-4109-99FE-B9D127C57AFE}

PSPath           : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES\CLSID\{2781761E-28E1-4109-99FE-B9D127C57AFE}
PSParentPath     : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES\CLSID
PSChildName      : {2781761E-28E1-4109-99FE-B9D127C57AFE}
PSDrive          : HKEY
PSProvider       : Microsoft.PowerShell.Core\Registry
(Default)       : Microsoft.MalwareProtection.IOfficeAntiVirusImplementation
  
```

Figure 1. get-itemproperty cmdlet showing Microsoft Malware Protection IOfficeAntiVirus Implementation is running

```

PS HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES\CLSID\{2781761E-28E0-4109-99FE-B9D127C57AFE}

PSPath           : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES\CLSID\{2781761E-28E0-4109-99FE-B9D127C57AFE}
PSParentPath     : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES\CLSID
PSChildName      : {2781761E-28E0-4109-99FE-B9D127C57AFE}
PSDrive          : HKEY
PSProvider       : Microsoft.PowerShell.Core\Registry
(Default)       : Windows Defender.IOfficeAntiVirusImplementation
  
```

Figure 2. get-itemproperty cmdlet showing Windows Defender IOfficeAntiVirus Implementation is running

Handles	NPM(K)	PM(K)	WS(K)	UM(M)	CPU(s)	Id	ProcessName
522	47	141336	142328	354	458.71	8632	AcroRd32
51	4	1872	2116	15		2124	AESTSr64
26	6	1256	2320	42	0.00	6432	BluetoothHeadsetProxy
401	40	32760	15132	160	1.42	6948	BTStackServer
211	16	8484	11032	129	6.44	5748	BTTray
121	9	3120	4688	53		2152	btwdins
1470	49	40656	44032	165		3324	CcmExec
101	8	1488	3100	33		7048	ComHQLBEX
1281	85	56876	23940	492	41.12	6608	communicator
64	8	4292	8956	75	1.08	10064	conhost
1023	15	3208	4012	46		592	csrss
837	23	3516	93012	222		708	csrss
641	26	10600	11396	55		2192	DcaSvc
665	40	28532	25236	203	2.40	6600	DcaTray
690	20	18020	9464	118		1644	DisplayLinkManager
157	10	4084	6048	83	0.08	6056	DisplayLinkUI
129	9	3384	6464	80		1704	DisplayLinkUserAgent
176	23	57056	40132	260	532.95	6064	dwm
977	78	56408	67480	341	75.05	6096	explorer
174	14	6532	8284	92	0.05	5036	FEPCClientUI
784	130	222072	251936	457	1,528.40	5424	firefox
87	9	2136	4440	68	0.80	9176	FlashUtil100_ActiveX
601	26	15248	13096	104		2344	FPMAgent
539	21	8976	8848	50		2412	FwcAgent
387	15	5132	5260	81	0.12	5872	FwcHgmt
493	24	7868	2388	103	0.64	5816	GoogleToolbarNotifier
128	12	4964	6236	88	1.70	3748	GoogleToolbarUser_32
120	10	2472	4452	49		6980	hpqWmiEx
82	7	2480	3432	32		1608	hpaservice
0	0	0	24	0		0	Idle
1245	255	351004	351236	745	378.12	968	ieplorex
898	64	24692	31536	195	24.84	3912	ieplorex
347	19	10632	11672	115	1.65	6620	IrmBackground

Figure 3. get-process cmdlet showing all the running processes on the system (partially shown due to length limit)

In order to parse the piped content of the PS scripts, the Out-File cmdlet can be utilized: Get-Process | Out-File c:\output\processes.txt.

PPSAM has a PERL written scripting implementation that controls the execution of the PowerShell commands, parses the data in HTML format and launches a browser window to display the output.

PPSAM requires a simple setup, a folder that contains the ppsam.pl PERL script, PowerShell .ps1 files containing commands to be executed, and the ppsam.html output report as shown in Figure 4.

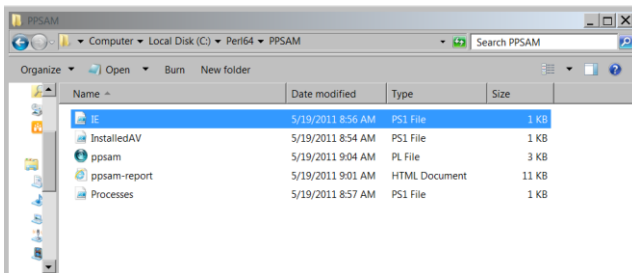


Figure 4. PPSAM file structure

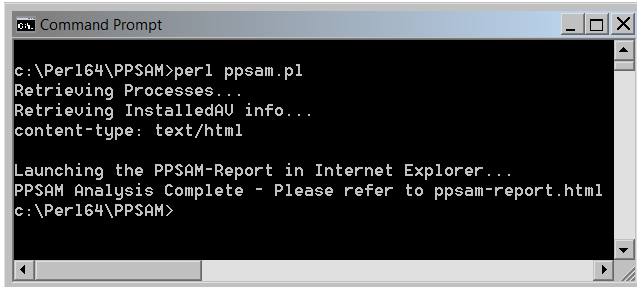
The following is a piece of sample PERL scripting code for PPSAM:

```
#!/usr/local/ActivePerl-5.6/bin/perl -w
#
# PPSAM: Proactive PowerShell Anti-Malware
# Customizable Comprehensive Tool to Supplement
# Commercial AVs
# Analyzes Windows system processes via PowerShell
# commands
#
# ppsam.pl
#
# By: Alejandro Villegas
# Department of Computer Science
# Sam Houston State University
#
# Professor: Dr. Lei Chen
# April 29, 2011
#
#Loading necessary Perl Modules
use strict;
use warnings;
use FindBin ();
use File::Copy qw(copy);
use Fcntl;
#Path to the PowerShell Executable
my $PWSpath =
"C:/Windows/System32/windowspowershell/v1.0/power
shell.exe";
#Variable to store Running-Processes
```

```
my $PS1RunningProcesses =
"C:/Perl64/PPSAM/Processes.ps1";
#Variable to store installed Antivirus Info
my $PS1InstalledAV =
"C:/Perl64/PPSAM/InstalledAV.ps1";
#Retrieve Processes via a PowerShell cmdlet: get-
process * | format-table
chomp(my @RunningProcesses = `$PWSpath -command
$PS1RunningProcesses`);
print "Retrieving Processes... \n";
#Retrieve Processes via a PowerShell cmdlet: Get-
ItemProperty "HKLM:\SOFTWARE\CLASSES\CLSID
\{2781761E-28E0-4109-99FE-B9D127C57AFE}" |
format-list | format-table
chomp(my @InstalledAVs = `$PWSpath -command
$PS1InstalledAV`);
print "Retrieving InstalledAV info... \n";
#Path to the PowerShell cmdlet to load Internet Explorer
my $IEpath = "C:/Perl64/PPSAM/IE.ps1";
#Deleting previous PPSAM-Reports
my $file = "C:/Perl64/PPSAM/ppsam-report.html";
unlink($file);
#Creating PPSAM-Report in HTML
print "content-type: text/html \n\n";
sysopen (HTML, 'ppsam-report.html',
O_RDWR|O_EXCL|O_CREAT);
printf HTML "<html>\n";
printf HTML "<head>\n";
printf HTML "<title>PPSAM: Proactive PowerShell
Anti-Malware</title>";
printf HTML "</head>\n";
printf HTML "<body bgcolor='Silver'>\n";
printf HTML "<b><h2><p align='center'>PPSAM:
Proactive PowerShell Anti-Malware</h2></b>";
printf HTML "By: Alejandro Villegas</p>";
printf HTML "<b>Processes<br></b>";
foreach (@RunningProcesses) {
printf HTML "$_ \n<br>";
}
printf HTML "<b>Anti-Malware Info<br></b>";
foreach (@InstalledAVs) {
printf HTML "$_ \n<br>";
}
printf HTML "</body>\n";
printf HTML "</html>\n";
close (HTML);
print "Launching the PPSAM-Report in Internet
Explorer... \n";
#PowerShell cmdlet to load Internet Explorer: $ie = new-
object -com
"InternetExplorer.Application"; $ie.visible = $true;
$ie.navigate("file:///C:/Perl64/PPSAM/ppsam-
report.html")
`$PWSpath -command $IEpath`;
print "PPSAM Analysis Complete - Please refer to
ppsam-report.html"
```

The PERL script ppsam.pl can be executed from the Windows command prompt as shown in Figure 5.

Lastly, the ppsam.pl script will automatically load the ppsam.html report for viewing and analysis as shown in Figure 6.



```
c:\Per164\PPSAM>perl ppsam.pl
Retrieving Processes...
Retrieving InstalledAU info...
content-type: text/html

Launching the PPSAM-Report in Internet Explorer...
PPSAM Analysis Complete - Please refer to ppsam-report.html
c:\Per164\PPSAM>
```

Figure 5. Execution of ppsam.pl via the command prompt

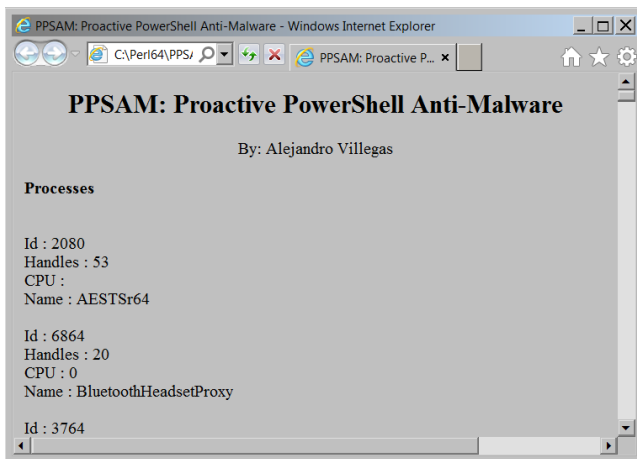


Figure 6. ppsam.html output generated report

The Windows PowerShell is based on the .NET framework and is able to execute any functionality that is available via the traditional GUI. Therefore the complexity of each cmdlet can be customized in order to gather any necessary data to from the system to discover a potential malware proactively.

6 Conclusion

While commercial AV and Anti-Malware solutions overall are the first layer of protection against popular viruses, Trojans and rootkits, they typically target malware that is known and utilize a comprehensive signature database. Therefore, a more proactive approach is needed in order to promote early prevention of malware attacks, since in majority of the cases rootkits can be installed without the end users ever acknowledging the systems have been compromised. If the malware has bypassed their AV or altered system binaries, chances are it will take a while before they even discover the compromise. PPSAM provides a solution that utilizes the newly released

Windows PowerShell 2.0 which comes with the Operating System and is capable of executing a plethora of different cmdlets that can check the performance of a system including remote operations for scalability. While not an AV replacement, it is absolutely a tool that can be used in conjunction with most third party software in order to propose a more proactive approach to prevent malware adverse attacks. PPSAM can be adapted to match the requirements of every particular infrastructure. PowerShell is based on .NET, therefore there is also the option to develop new cmdlets based on clients' OS and application security priorities.

7 Future work

In order to provide a more user friendly application, PPSAM will possess a GUI interface coded in PERL and CGI. Such interface will have the capability of displaying, parsing and organizing the cmdlets output. In addition, archiving PPSAM scans will be an option. Furthermore, the utility will offer the feature of loading additional PowerShell scripts, as well as running a syntax and sanity check before uploading. Another proposed implementation is to be able to load PPSAM via a bootable image such as Windows PE; this alternative would be able to be utilized even on compromised systems. Additionally, the Windows Research Kernel (WRK) [2] will be used in order to create more monitoring PowerShell cmdlets at the kernel level, in order to construct a potential DKOM detection antidote. After all, new generation rootkits aim to exploit kernel memory vulnerabilities, hence the importance of kernel memory protection [3].

8 References

- [1] Desmond Lobo, Paul Watters, Xin-Wen Wu, Li Sun, "Windows Rootkits: Attacks and Countermeasures," etc, pp.69-78, 2010 Second Cybercrime and Trustworthy Computing Workshop, 2010.
- [2] Diomidis Spinellis, "A tale of four kernels," icse, pp.381-390, Proceedings of the 30th International Conference on Software Engineering (ICSE '08), 2008.
- [3] Dong Hwi Lee, Jae Myung Kim, Kyong-Ho Choi, Kuinam J. Kim, "The Study of Response Model & Mechanism Against Windows Kernel Compromises," ichit, pp.600-608, 2008 International Conference on Convergence and Hybrid Information Technology, 2008.
- [4] Hung-Min Sun, Hsun Wang, King-Hang Wang, Chien-Ming Chen, "A Native APIs Protection Mechanism in the Kernel Mode against Malicious Code," IEEE Transactions on Computers, 10 Feb. 2011. IEEE computer Society Digital Library. IEEE Computer Society.

- [5] Matthias Schmidt, Lars Baumgartner, Pablo Graubner, David Bock, Bernd Freisleben, "Malware Detection and Kernel Rootkit Prevention in Cloud Computing Environments," Parallel, Distributed, and Network-Based Processing, Euromicro Conference on, pp. 603-610, 2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2011.
- [6] Microsoft. Scripting with Windows PowerShell, 2008. <http://technet.microsoft.com/en-us/scriptcenter/dd742419.aspx>
- [7] Nicolas Bruno, Surajit Chaudhuri, "Interactive physical design tuning," icde, pp.1161-1164, 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), 2010.
- [8] Woei-Jiunn Tsaur, Yuh-Chen Chen, "Exploring Rootkit Detectors' Vulnerabilities Using a New Windows Hidden Driver Based Rootkit," socialcom, pp.842-848, 2010 IEEE Second International Conference on Social Computing, 2010.
- [9] Yi-Min Wang, Doug Beck, Binh Vo, Roussi Roussev, Chad Verbowski, "Detecting Stealth Software with Strider GhostBuster," Dependable Systems and Networks, International Conference on, pp. 368-377, 2005 International Conference on Dependable Systems and Networks (DSN'05), 2005.
- [10] Yong Wang, Dawu Gu, Jianping Xu, Fenyu Zen, "Assembly Reverse Analysis on Malicious Code of Web Rootkit Trojan," Web Information Systems and Mining, International Conference on, pp. 501-504, 2009, International Conference on Web Information Systems and Mining, 2009.