# Detecting Windows Based Exploit Chains by Means of Event Correlation and Process Monitoring

Muhammad Mudassar Yamiun
Norwegian University of Science and Technology,
Department of Information Security and Communication Technology
Gjøvik, Norway
muhammad.m.yamin@ntnu.no

Basel Katt
Norwegian University of Science and Technology,
Department of Information Security and Communication Technology
Gjøvik, Norway
basel.katt@ntnu.no

Vasileios Gkioulos
Norwegian University of Science and Technology,
Department of Information Security and Communication Technology
Gjøvik, Norway
vasileios.gkioulos@ntnu.no

**This article presents a novel algorithm for the detection of exploit chains in a Windows based environment. An exploit chain is a group of exploits that executes synchronously, in order to achieve the system exploitation. Unlike high-risk vulnerabilities that allow system exploitation using only one execution step, an exploit chain takes advantage of multiple medium and low risk vulnerabilities. These are grouped, in order to form a chain of exploits that when executed achieve the exploitation of the system. Experiments were performed to check the effectiveness of developed algorithm against multiple anti-virus/anti-malware solutions available in the market.**

***Keywords—Exploit Chain, Event Correlation, Process Monitoring, Windows, process correlation***

## I. INTRODUCTION

Recently, the Pwn2own 2018 researchers introduced multiple Zero Day exploits, which were primarily based on a chain of multiple exploits for the exploitation of systems and services [1]. Traditional anti-virus and anti-malware software uses process monitoring and process isolation techniques for detection, according to suspicious process behavior pattern [2]. Yet, as we see in the Pwn2Own 2018 results, the researchers were able to break such process isolation and sandboxing process protection techniques. Examples of exploits that cannot be detected using traditional techniques are the *guest-to-host* exploits [3], and the macro-less DDE (dynamic data execution) in an MS office application [4]. In this article, we present a novel technique for the detection of such exploits using process execution monitoring my means of event correlation. The technique performs detection in a signature free and fully autonomous manner, using only the process names for monitoring and detection of exploitations. We use event correlation with respect to events extracted from process monitoring logs to create a chain of suspicious processes generated by the application to identify a detection. This article is organized in six sections. The first section introduces the problem, while in the second section we discuss related work and provide additional information about the problem background. The following sections present the proposed algorithm and initial experimentation results, while in the last sections we provide a discussion, future work and conclude the article.

## II. RELATED WORK

The authors were not able to identify in the literature any viable existing technique for the detection of complex exploit chains such as *guest-to-host* exploits, while most cloud security vendors use defense in depth architectures to avoid security incidents involving *guest-to-host* exploits [5]. Existing techniques for securing a host from *guest-to-host* exploits use a multistep approach. Initially, an external process hook or agent is added in each virtual machine, which is then updated for malware and virus definitions from an external source [6]. Another technique used for securing virtual machines, relies on VMI (Virtual Machine Introspection) based process monitoring, for malware detection on a virtual machine from an external source [7]. Graph based event correlation (on the virtual machine) for anomaly detection using machine learning techniques [8]. The problem faced by existing techniques, is that they mostly focus on the protection of the virtual machine, without taking into account the new *guest-to-host* exploits, which exploit guest isolation using an exploit chain and allow the guest virtual machine to access the host operating system. Furthermore, in respect to the macro-less DDE in MS office applications [4], the research focus is on using malicious PowerShell commands for exploiting the system. For the detection of malicious PowerShell commands, researchers are currently using machine-learning techniques [9]. Yet, such existing detection techniques are vulnerable to the use of command line obfuscation for avoiding detection [10].

## III. PROBLEM BACKGROUND

To further explain the problem a brief technical background is given.

### A. *Exploit Chains*

In a normal IT security environment one vulnerability is enough to compromise the security of a system. However, due to continue system security improvements finding such vulnerabilities is becoming harder day by day. On the other hand low impact vulnerabilities are usually easy to find, researcher demonstrated multiple exploits which use these low impact vulnerabilities [4][5], and chain them together to compromise system security. To further explain the flow of a single exploit and an exploit chain we created a simple flow chart for easy understating. Flow chart showing comparison of traditional exploits and an exploit chain flow is given in figure 1:

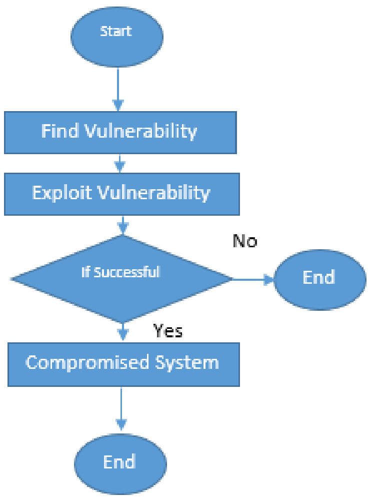


Fig (1) Example of traditional exploit with a single vulnerability

In comparison to a vulnerability that is exploited by a single exploit, in an exploit chain multiple vulnerabilities are involved. Each exploit uses the output of the previous to accomplish the objectives. A flow chart representation of exploit chain is seen in the figure 2:

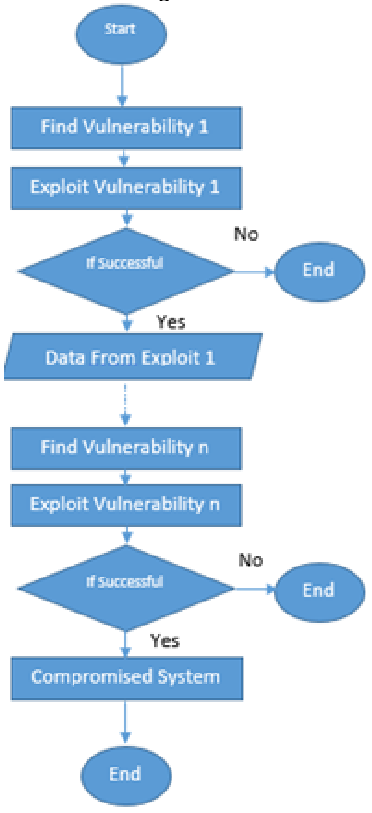


Fig (2) Example of exploit chain with multiple vulnerability

Similar concepts exists in literature such as attack chain or attack paths which is set of possible steps that an attacker could take to compromise a system, involving multiple nodes on which exploitation is performed. In contrast, exploit chaining is the process of linking multiple vulnerabilities of one node which are present in a system and executing them in a specific order to compromise security.

### B. *Window Event logging Mechanism*

The Microsoft Security Event logging mechanism is present in every new release of Windows since Windows XP. This event logging mechanism allows the identification of the type of computer events happening in Windows based systems when an exploit is executed. Researchers at JPCERT [15] provided details of such security events in there technical report. In this research paper we focus on Event ID 4688 [11]. Which is a Windows new process creation event. Each 4688 event contains the following fields

- **SubjectUserSid :** Security id of account from where the process is executed
- **SubjectUserName :** Account name from where the process is executed
- **SubjectDomainName :** Domain Name
- **SubjectLogonId :** Logon id of account from where the process is executed
- **NewProcessId :** Unique hexadecimal new process identifier
- **NewProcessName :** New process name executed by parent process
- **ProcessId :** Unique hexadecimal process identifier
- **CommandLine :** Command which is executed
- **TargetUserSid :** Security id of account on which process executed
- **TargetUserName :** User name
- **TargetDomainName :** Computer name
- **TargetLogonId :** Login id of account on which process executed
- **ParentProcessName :** Name of process which executes new process
- **MandatoryLabel :** Secure object control integrity label assigned to new process

From the information present in the fields of 4688 event we used *NewProcessId, ProcessId, TargetDomainName* in our detection algorithm. The *ProcessId* is a unique identifier issued by computer operating system to a running process. *NewProcessId* is a unique identifier issued by computer operating system to a process that is executed by another running process.TargetDomainName is the unique name of the computer on the domain.

### C. *Guest-to-host exploit*

A recent report from SpiceWork [13] shows that server virtualization adoption reached 85% in comparison to 15% of physical IT infrastructure in 2017, as seen in figure 3.

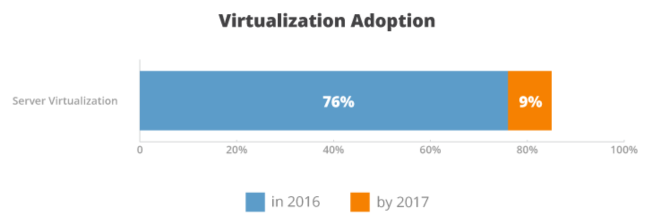


Fig (3) [13] Server virtualization trends

This trend leads security researchers to develop exploits that can break guest isolation and compromise the host machine. A list of few vulnerabilities is given below

- CVE-2017-4924: An out of bound memory corruption vulnerability in Vmware 12.x to 12.5.7 Implementation of SVGA(Virtual graphic card) allows attackers to execute code host system
- CVE-2017-4934: An heap buffer overflow vulnerability in Vmware 12.x to 12.5.8 Implementation of VMNET (virtual machine network) allows attackers to execute code host system
- CVE-2017-4936: An out-of-bounds read vulnerability in Vmware 12.x to 12.5.8 JPEG2000 parser in the TPView.dll allows guest to execute code or perform a DOS (Denial of Service) on the Windows OS.

A detailed list of *guest-to-host* escape vulnerabilities can be found online [14]. An example of these vulnerabilities is CVE-2017-4924 in which an out of bound memory corruption in vmwar-vmx.exe with incorrect memory mapping exists. This allows Data Execution Prevention bypass which leads to code execution on host from virtual machine.

Exploit writers were able to exploit this vulnerability and they created a POC (Proof of Concept) [17] for its exploitation. In the POC first the guest isolation is escaped by out of bound memory corruption and then CMD is executed by exploiting host Windows task registry. From CMD, PowerShell is executed to achieve remote shell level access on host. Schematically the exploit chain presented in figure 4.

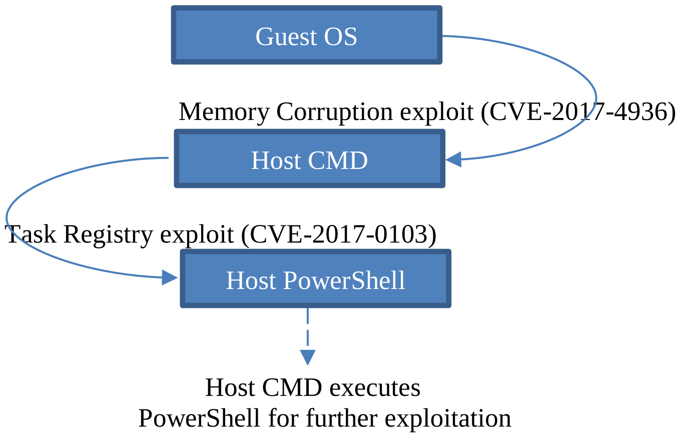

Guest OS
Memory Corruption exploit (CVE-2017-4936)
Host CMD
Task Registry exploit (CVE-2017-0103)
Host PowerShell
Host CMD executes
PowerShell for further exploitation

Fig (4) Guest to host escape exploit chain

Ideally the virtualization provide isolation between Guest OS and Host OS, where only the relevant services are shared as seen in the figure 5:

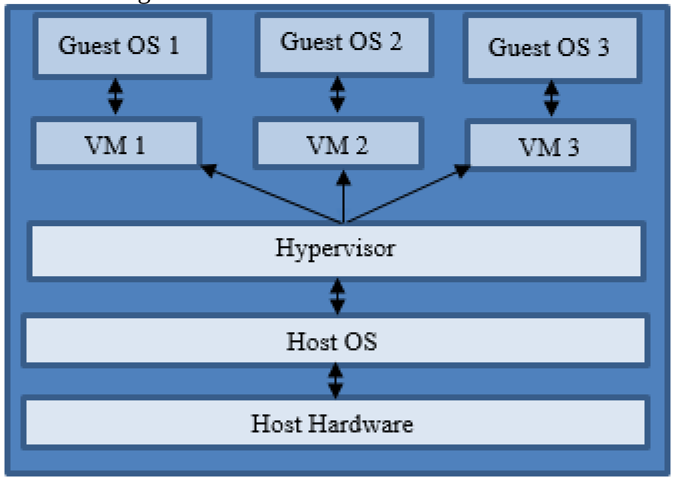


Fig (5) Isolated guest and host in virtualized environment

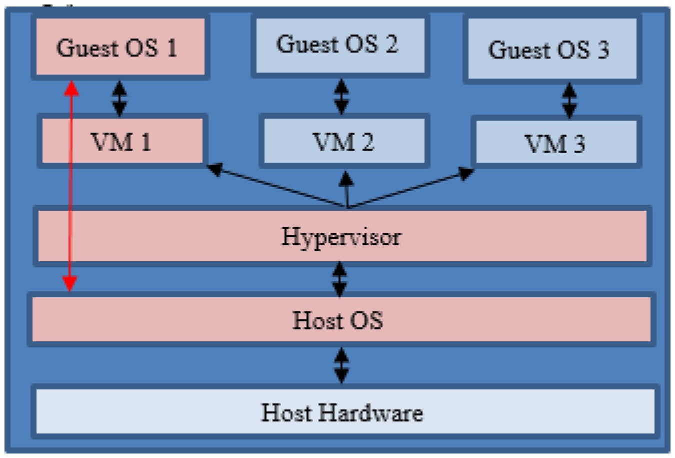


Fig (6) Broken isolation between guest and host

## D. *Macro-less DDE Attacks*

To transfer data between different applications Windows provides the functionality of *Dynamic Data Execution*. The communication or COM Objects of Microsoft word and Microsoft excel, have public access to this DDE functionality. The functionality allows Microsoft Word and Excel to execute system commands legitimately. Exploit writers misused this functionality and were able to develop complex exploits such as macro-less DDE code execution [4]. It is also very difficult to detect with traditional detection techniques since the functionality is legitimate feature and is not blocked and patched by Microsoft [4]. Anti-virus and anti-malware solutions are using signature-based detection mechanism for the detection of macro-less DDE but the signature-based detection was also easily bypass able using command obfuscation techniques [9]. The exploit execution of macro-less DDE is similar to guest-to-host escape but in this case Microsoft Word or Excel is used to create exploit chain. First DDE on Microsoft Word or Excel is exploited which allows the exploited process to use COM object in Windows and pass data related to secondary logon elevation vulnerability in windows through which CMD is executed. Now when the CMD process is started a command line argument containing malicious PowerShell script is passed to obtain a remote shell of the host a schematic repression of the explication chain is seen in figure 7:

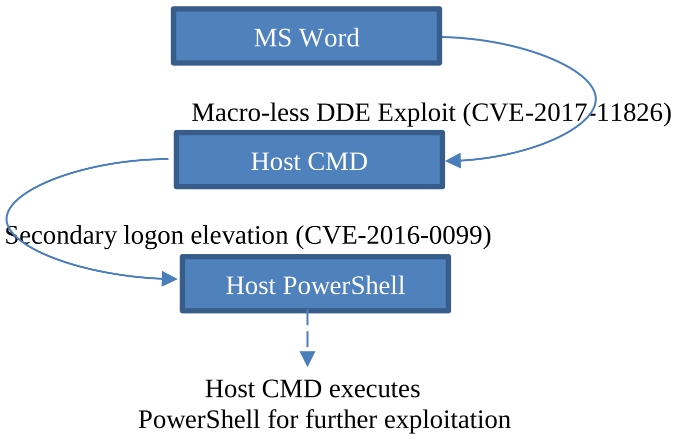
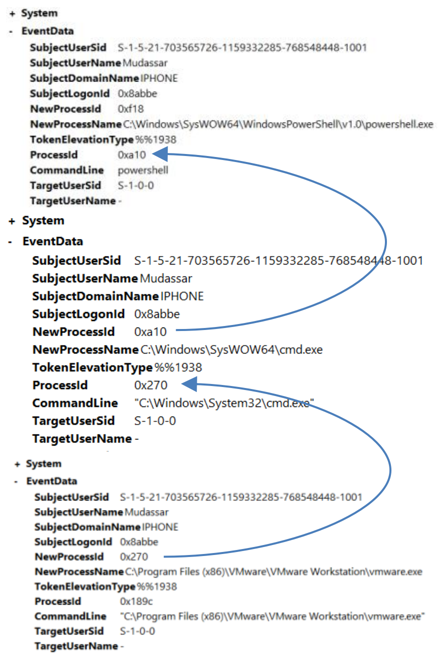

MS Word
Macro-less DDE Exploit (CVE-2017-11826)
Host CMD
Secondary logon elevation (CVE-2016-0099)
Host PowerShell
Host CMD executes
PowerShell for further exploitation

Fig (7) Guest to host escape exploit chain

## IV. Detection Methodolgh

The detection algorithm is developed by analyzing Windows security logs. Consider the following Windows security logs of Vmware *guest-to-host* Escape exploit. It breaks the Guest isolation, executes a CMD command on the host to run a PowerShell Exploit. The logs generated by the exploit can be seen in the figure 8:



```
+ System
- EventData
    SubjectUserSid    S-1-5-21-703565726-1159332285-768548448-1001
    SubjectUserName Mudassar
    SubjectDomainName IPHONE
    SubjectLogonId    0x8abbe
    NewProcessId      0xf18
    NewProcessName C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
    TokenElevationType %%1938
    ProcessId         0xa10
    CommandLine       powershell
    TargetUserSid     S-1-0-0
    TargetUserName -
+ System
- EventData
    SubjectUserSid    S-1-5-21-703565726-1159332285-768548448-1001
    SubjectUserName Mudassar
    SubjectDomainName IPHONE
    SubjectLogonId    0x8abbe
    NewProcessId      0xa10
    NewProcessName C:\Windows\SysWOW64\cmd.exe
    TokenElevationType %%1938
    ProcessId         0x270
    CommandLine       "C:\Windows\System32\cmd.exe"
    TargetUserSid     S-1-0-0
    TargetUserName -
+ System
- EventData
    SubjectUserSid    S-1-5-21-703565726-1159332285-768548448-1001
    SubjectUserName Mudassar
    SubjectDomainName IPHONE
    SubjectLogonId    0x8abbe
    NewProcessId      0x270
    NewProcessName C:\Program Files (x86)\VMware\VMware Workstation\vmware.exe
    TokenElevationType %%1938
    ProcessId         0x189c
    CommandLine       "C:\Program Files (x86)\VMware\VMware Workstation\vmware.exe"
    TargetUserSid     S-1-0-0
    TargetUserName -
```

Fig (8) Windows event logs generated from a *guest to host* exploit

After analyzing the logs a clear link is established between the processes generated by the exploit, as the ProcessID of a process is the NewProcessID of previous process involved in the exploit chain. We identified that by co-relating multiple events based upon the relation of ProcessID and NewProcessID we can create a process execution chain of the exploit. Accordingly a detection algorithm has been developed based on this finding. The proposed algorithm works in the following manner:

**Exploit Chain Detector (ECD) Algorithm**

**Input:** a list of ordered Windows event logs *A*; a list of process names to be monitored *B*

**/*** an event logs has the following attributes: *NewProcessId, ProcessId, ProcessName, TargetDomainName*/*

**/*** *B* contains a list of process names that are executed after a vulnerability is exploited retrieved from report[1] [15] */

**Output:** a list of string stacks *D*, a Boolean represents if exploit chains are detected *c*

**/*** *D* will contain all exploit chains detected by the algorithm, and *c* is true if one chain is found*/

**Initialization:** create an empty event log *a* ; initialize *c* with the value false ; create integer *m* with initial value 0

```
1  for (i=0; i<Size(A); i++) do
2      if ($A_i$.ProcessId $\in$ B) then
3          $a=A_i$
4          for (j=i; i<Size(A); j++) do
5              if (a. ProcessId == $A_j$.NewProcessId && a.TargetDomainName == $A_j$.TargetDomainName) then
6                  $D_m$.Push(a.ProcessName)
7                  $a=A_j$
8                  if($A_{(j+m)}$.NewProcessId==Null) then
9                      c=true
10                     m=m+1
11                 end if
12             end if
13         end for
14     end if
15 end for
```

1 (The list is created according to the JPCERT report Detecting Lateral Movement through Tracking Event Logs, which suggest the following processes for active tracking: cmd, powershell, regsvr32, rundll32, mshta)

The ECD (Exploit Chain Detector) algorithm requires only two inputs for execution. First is the security monitoring logs on which detection is performed *A*, and second is the list of process names that need to be monitored *B*. *A* is directly retrieved from host which contains individual events with multiple fields like ProcessId, NewProcessId ,ProcessName, TargetDomainName etc. *B* is the list of process names given by JPCERT[15] that are executed after a vulnerability is exploited. In output the algorithm returns whether an exploit chain is detected or not in a boolean variable c. If detected then it also shows the exploit chains in a stack *D*. For initializing the algorithm we need an empty event log *a*, an integer *m* with value 0 and c will be initialized with the value false..

When the algorithm starts processing it reads all the event logs available in *A*, then it start checking one by one if the ProcessName of an event in *A* is present in *B*. If a match is found the single event of *A* is stored in *a* and ProccessName is pushed to the stack D. Now a second comparison is performed on those events of *A* which are present after *a*, in the comparison ProcessId of *a* and ComputerName of *a* is compared with the ProcessId of the next event of *A* and ComputerName of that event in the coming logs. If a match is found ProccessName is pushed to a stack D and value of *a* is updated with current value of event at *A* .This process is performed until there is no NewProcessID in *A*. When this happens true value is assigned to *c* while the stack *D* contains the whole exploit chain. We calculated the algorithm complexity and it was

$$O\,(n \log n)$$

The algorithm complexity is good for detection of exploit chain in environment with small or medium amount of security logs data but in an environment with large amount of event log data the algorithm will take considerable amount of time for detection of exploit chains.

## V. Implementation

We developed our proposed algorithm on a simple python based Windows logging mechanism. It is based on the standard pywin32 library presented at python library blog post [12], while the detection algorithm is built around this logging mechanism. The logs come in a recursive manner, as post exploitation is done after the initial exploitation with respect to time. We developed our detection algorithm POC on Microsoft Visual Studio 2017[2] on python environment 3.6. Our implementation contains the following primary functions.

*1) Get-All-System-Events*

This function takes all event logs from system which include application events, security events, setup events, system events and forwarder events and write them to separate files on disk.

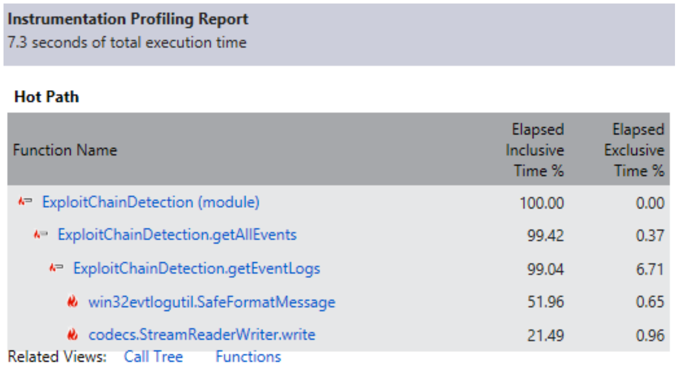
*2) Event-Parser*

Event log parser read the event from the disk and parse them to individual readable events and forward it to next function *Get All Event Logs*

*3) Get-All-Event Logs*

*Get-All-Event-Logs* is the core function of our detection algorithm. It takes parsed events from *Event-parser*, then performs event process comparison and event corelation for the detetction of exploit chains.

To test the algorithm we run the developed tool on a Core i5 3320M 2.60 ghz system with 16 gb of RAM against 17098 Windows secuirty events and two executed exploits *guest-to-host*, *macro-less DDE*. The Execution took 7.3s for the detection of the exploit chains, which can be seen in the figure 9:


**Instrumentation Profiling Report**
7.3 seconds of total execution time

**Hot Path**

| Function Name | Elapsed Inclusive Time % | Elapsed Exclusive Time % |
|---|---|---|
| ExploitChainDetection (module) | 100.00 | 0.00 |
| ExploitChainDetection.getAllEvents | 99.42 | 0.37 |
| ExploitChainDetection.getEventLogs | 99.04 | 6.71 |
| win32evtlogutil.SafeFormatMessage | 51.96 | 0.65 |
| codecs.StreamReaderWriter.write | 21.49 | 0.96 |

Related Views: Call Tree Functions

Fig (9) Implemented algorithm process execution time and function calls

The implementation works without any malware, virus or malicious command signature for the detection of the exploit chain. We performed detailed experimentation on the developed algorithm to check the effeteness of our algorithm. The following section elaborates the experimental details and results:

## VI. ExperimentATAion And Results

Two experiments were performed to check the effectiveness of the developed algorithm one is a *guest-to-host* exploit the other is a macro-less DDE exploit details of which are given below:

### A. *Guest-to-host* exploit

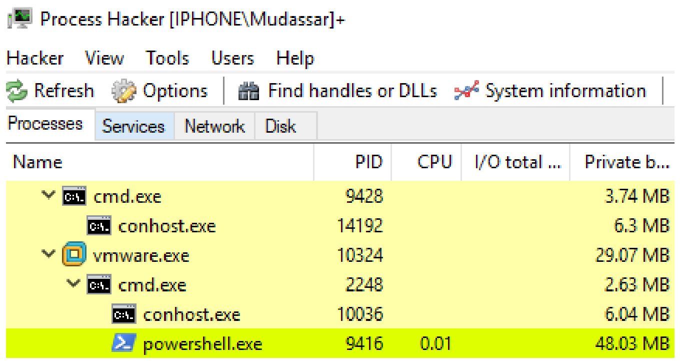
#### 1. Experimental Setup

We created our experimental setup on a 64-bit Windows 10 machine running on a VMware Workstation 12.5.5. A Guest Windows 10 operating system is installed on the Vmware. For detection comparison analysis we installed Bit Defender Home, Avira Home, Kasper Sky Home, Avast Home and Panda Security Suite on the Host OS and deactivated them.

#### 2. Controlled Exploit Execution

We executed a *guest-to-host* proof of concept for CVE-2017-4924 [16], [17] on Guest windows 10 operating system. The exploit breaks the Guest isolation and executed an CMD on host machine and then executed PowerShell. Execution of exploit on Process Hacker can be seen in the figure 10:

2 https://www.visualstudio.com/


Process Hacker [IPHONE\Mudassar]+
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Processes Services Network Disk

| Name | PID | CPU | I/O total ... | Private b... |
|---|---|---|---|---|
| cmd.exe | 9428 | | | 3.74 MB |
| conhost.exe | 14192 | | | 6.3 MB |
| vmware.exe | 10324 | | | 29.07 MB |
| cmd.exe | 2248 | | | 2.63 MB |
| conhost.exe | 10036 | | | 6.04 MB |
| powershell.exe | 9416 | 0.01 | | 48.03 MB |

Fig (10) *Guest to host* exploit execution

3. Scenarios

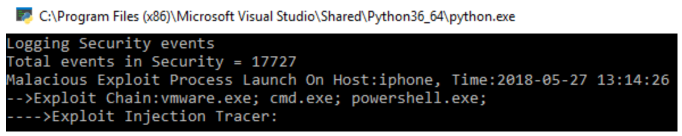
We created two scenarios for comparison of our developed algorithm with different anti-virus/anti-malware solution available in the market. In the first scenario we executed the exploit only against our detection algorithm. In the second scenario we executed the exploit against different anti-virus/anti-malware solution available in the market. Details of which is given below:

a) Detection against developed algorithm

We executed our detection algorithm on the Windows 10 Host OS and executed the exploit on Windows 10 Guest OS and we were able to detect the exploit chain in the first run successfully.

$$\textit{Vmware} \longrightarrow \textit{CMD} \longrightarrow \textit{PowerShell}$$

The detection of the exploit chain by the developed algorithm can be seen in the figure 11:


C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64\python.exe

```
Logging Security events
Total events in Security = 17727
Malacious Exploit Process Launch On Host:iphone, Time:2018-05-27 13:14:26
-->Exploit Chain:vmware.exe; cmd.exe; powershell.exe;
---->Exploit Injection Tracer:
```

Fig (11) *Guest-to-host* exploit detection

The exploit chain detected by our algorithm is according to the process execution tree shown at process hacker. However, due to the event correlation capabilities of the developed algorithm with respect to malicious process monitoring, we are able to mark the chain as being malicious.

b) Detection against anti-virus/anti-malware solutions

We ran Bit Defender Home, Avira Home, Kasper Sky Home, Avast Home and Panda Security Suite on the Windows 10 Host OS one by one while executing the guest-to-host exploit on a Window 10 Guest OS for the possible detection of exploit chain we weren't able to identify any malicious activity.

4. Experimental Results

We ran a comparative analysis of our detection techniques with different anti-virus and anti-malware solution available in the market. The table 1 shows the result of detection by different security software.

| Solution | Detection Yes/No |
|---|---|
| Proposed Algorithm | Yes |
| Windows Defender | No |
| Bit Defender Home | No |
| Avira Home | No |
| Kasper Sky Home | No |
| Avast Home | No |
| Panda Security Suite | No |

Table (1) Result of Comparative Detection Analysis of Developed algorithm and Different Software Security Software

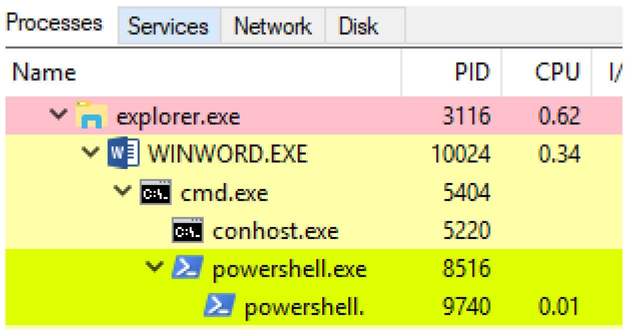
B. Macro-less DDE exploit

1. Experimental Setup

We used Microsoft Office 2013 running on 64-bit Window 10 for the experimentation purpose.

2. Controlled Exploit Execution

For macro-less DDE Exploit we developed an obfuscated DDE Exploit for CVE-2017-11826.The exploit first executes CMD from MS Word then from CMD it executes PowerShell for further exploitation. The exploit execution on Process Hacker can be seen in the figure 12:



| Name | PID | CPU |
|---|---|---|
| explorer.exe | 3116 | 0.62 |
| WINWORD.EXE | 10024 | 0.34 |
| cmd.exe | 5404 | |
| conhost.exe | 5220 | |
| powershell.exe | 8516 | |
| powershell. | 9740 | 0.01 |

Fig (12) Macro-less DDE exploit execution

3. Scenarios

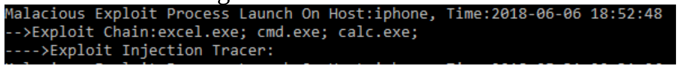
We created two scenarios for the evaluation of our developed algorithm in the first scenario we executed the exploit on the Experimental setup to check the detection against our developed algorithm. In the second scenario we used online service VIrustototal[3] which performed detection analysis against 59 anti-virus/anti-malware solution details of the scenarios is given below:

a) Detection against developed algorithm

We executed our detection algorithm on the Windows 10 running MS Word and we were able to detect the exploit chain in the first run successfully.

$$\textit{Word} \longrightarrow \textit{CMD} \longrightarrow \textit{PowerShell}$$

The detection of the exploit chain by the developed algorithm can be seen in the figure 13:



```
Malacious Exploit Process Launch On Host:iphone, Time:2018-06-06 18:52:48
-->Exploit Chain:excel.exe; cmd.exe; calc.exe;
---->Exploit Injection Tracer:
```

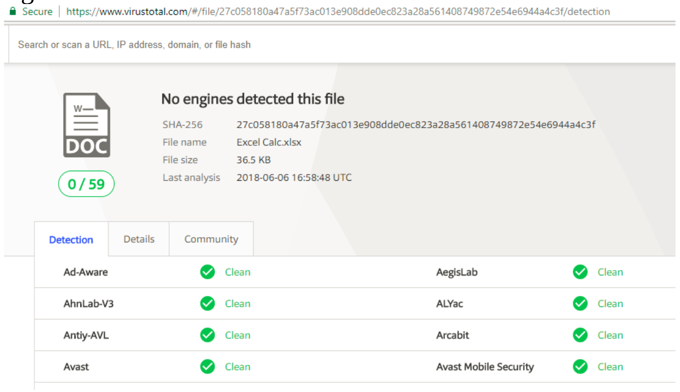
Fig (13) Macro-less DDE exploit detection

3 https://www.virustotal.com/#/file/27c058180a47a5f73ac013e908dde0ec823a28a561408749872e54e6944a4c3f/detection

The exploit chain detected by our algorithm is according to the process execution tree shown at Process Hacker. However, due to the event correlation capabilities of the developed algorithm with respect to malicious process monitoring, we are able to mark the chain as being malicious.

b) Detection against anti-virus/anti-malware solutions

As stated earlier we developed an obfuscated macro-less DDE exploit which have zero detection signature against 59 anti-virus and anti-malware solution on Virus Total[3] as seen in the figure 14:


Secure | https://www.virustotal.com/#/file/27c058180a47a5f73ac013e908dde0ec823a28a561408749872e54e6944a4c3f/detection

Search or scan a URL, IP address, domain, or file hash

No engines detected this file

0 / 59

| | |
|---|---|
| SHA-256 | 27c058180a47a5f73ac013e908dde0ec823a28a561408749872e54e6944a4c3f |
| File name | Excel Calc.xlsx |
| File size | 36.5 KB |
| Last analysis | 2018-06-06 16:58:48 UTC |

Detection | Details | Community

| | | | |
|---|---|---|---|
| Ad-Aware | ✓ Clean | AegisLab | ✓ Clean |
| AhnLab-V3 | ✓ Clean | ALYac | ✓ Clean |
| Antiy-AVL | ✓ Clean | Arcabit | ✓ Clean |
| Avast | ✓ Clean | Avast Mobile Security | ✓ Clean |

Fig (14) Obfuscated macro-less DDE exploit

4. Experimental Results

Our analysis is being performed on 59 anti-virus and anti-malware solution for saving space few results are omitted but details of analysis can be found online*. Table 2 presenting the detection result compare to different anti-virus and anti-malware solution is given below:

| Solution | Detection Yes/No |
|---|---|
| Proposed Algorithm | Yes |
| Ad-Aware | No |
| AegisLab | No |
| AhnLab-V3 | No |
| ALYac | No |
| Antiy-AVL | No |
| Arcabit | No |
| Avast | No |
| Avast Mobile Security | No |
| AVG | No |

Table (2) Result of comparative detection analysis of developed algorithm and different software security software

## VII. Discussion

The key factor of failure of other detection techniques compare to our techniques is that other detection techniques focus on signature and illegitimate behavior of processes that are being executed. As shown above legitimate behavior of an application can be used for malicious purposes. Similarly signatures of malicious code can be obfuscated as well to avoid detection. Our detection technique works completely different in comparison to other techniques it tries to identify the chain of processes that are being executed by a process and then co-relate them for the identification of malicious processes in the chain. Therefore it has the capability to detect those exploits which are not detected by other available solutions.

We believe that the algorithm complexity is not ideal and there is a lot of room for improvement. But the approach which the algorithm use is quite unique for detection of malicious exploit chains. We intend to further refine the technique for other detection like the detection malicious activates of user by means of event correlation.

## VIII. Conclusion and Future Wrok

With the proposed detection technique we are able to identify complex exploit chains. We assume that some complex user administration automation scripts may cause false positives due to their complex execution nature, but overall the detection technique is satisfactory in detecting complex exploit chains. A significant benefit of this technique is that it works completely blindly, without any signature and behavior metrics. In the future, we intent to further refine our technique to trace the exploit chain when the process migrates to another process. Furthermore we will perform our experiments in a large network for identification of false positives in our detection algorithm.

## References

[1] Pwn2own 2018 – Day Two Results and Master Of Pwn https://www.zerodayinitiative.com/blog/2018/3/15/pwn2own-2018-day-two-results-and-master-of-pwn Accessed 17 May 2018

[2] Srinivasan, Deepa, Zhi Wang, Xuxian Jiang, and Dongyan Xu. "Process out-grafting: an efficient out-of-vm approach for fine-grained process execution monitoring." In Proceedings of the 18th ACM conference on Computer and communications security, pp. 363-374. ACM, 2011.

[3] Mandal, Debasish, and Yakun Zhang. THE GREAT ESCAPES OF VMWARE: A RETROSPECTIVE CASE STUDY OF VMWARE GUEST-TO-HOST ESCAPE VULNERABILITIES. PDF. London: Blackhat, December, 2017.

[4] Sensepost https://sensepost.com/blog/2017/macro-less-code-exec-in-msword/ Accessed 17 May 2018

[5] Neumann, William C., Thomas E. Corby, and Gerald Allen Epps. "System for secure computing using defense-in-depth architecture." U.S. Patent 7,428,754, issued September 23, 2008.

[6] Win, Thu Yein, Huaglory Tianfield, and Quentin Mair. "Big data based security analytics for protecting virtualized infrastructures in cloud computing." IEEE Transactions on Big Data 4, no. 1 (2018): 11-25.

[7] Wang, Xiaoguang, Yong Qi, Zhi Wang, Yue Chen, and Yajin Zhou. "Design and Implementation of SecPod, A Framework for Virtualization-based Security Systems." IEEE Transactions on Dependable and Secure Computing (2017).

[8] Ucci, Daniele, Leonardo Aniello, and Roberto Baldoni. "Survey on the Usage of Machine Learning Techniques for Malware Analysis." arXiv preprint arXiv:1710.08189 (2017).

[9] Hendler, Danny, Shay Kels, and Amir Rubin. "Detecting Malicious PowerShell Commands using Deep Neural Networks." arXiv preprint arXiv:1804.04177 (2018).

[10] Dosfuscation: Exploring the Depths Of Cmd.exe Obfuscation and Detection Techniques « Dosfuscation: Exploring the Depths Of Cmd.exe Obfuscation and Detection Techniques

Daniel Bohannon - https://www.fireeye.com/blog/threat-research/2018/03/dosfuscation-exploring-obfuscation-and-detection-techniques.html Accessed 19 May 2018

[11] 4688(s) A New Process Has Been Created. (windows 10)

Mir0sh - https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4688 Accessed 19 May 2018

[12] URLhttps://www.blog.pythonlibrary.org/2010/07/27/pywin32-getting-windows-event-logs/ Website TitleThe Mouse Vs. The Python Date Accessed May 27, 2018

[13] Server Virtualization and Os Trends

Spiceworks, Inc - https://community.spiceworks.com/networking/articles/2462-server-virtualization-and-os-trends Accessed 24 May 2018

[14] Virtual Machine Escape

https://en.wikipedia.org/wiki/Virtual_machine_escape Accessed 17 May 2018

[15] Research Report Released: Detecting Lateral Movement Through Tracking Event Logs (version 2)

https://blog.jpcert.or.jp/2017/12/research-report-released-detecting-lateral-movement-through-tracking-event-logs-version-2.htm Accessed 17 May 2018

[16] Comsecuris/vgpu_shader_pocs

Comsecuris - https://github.com/Comsecuris/vgpu_shader_pocs Accessed 18 May 2018

[17] 0patch Blog Luka Treiber - http://blog.0patch.com/2017/10/micropatching-hypervisor-with-running.html Accessed 18 May 2018