Global Information Assurance Certification Paper

Detecting DLL Search Order Hijacking:

How using a purple team approach can help create better defensive techniques and a more tactical SIEM

GIAC (GCDA) Gold Certification

Author: Lasse Hauballe Jensen, Lassehauballe@protonmail.com
Advisor: Robert Vandenbrink

Accepted: April 30th, 2020

Abstract

Many SIEM analysts will recognize the feeling of being overwhelmed with security logs and alerts, and having to deal with them using a SIEM that gets slower and slower. For many, it may even seem that the SIEM has transitioned into being an overpriced log storage system. Figuring out how to make the SIEM faster, more tactical, and defensive-oriented will also be a way to make the analysts better and happier. It will also provide more accurate reporting for managers, and lastly, it will reduce storage and processing requirements reducing the overall cost of running a SIEM.

This paper will show that by utilizing a purple team approach in combination with the MITRE ATT&CK™ framework, it is possible to systematically generate precise and simple detection techniques that will work in any SIEM. To exemplify this, this paper will focus on the post-exploitation technique called *DLL Search Order Hijacking*, which is used for persistence, privilege escalation, and evasion. By gathering an understanding of what the technique is and running the actual attack, it will be possible to create a reliable detection technique combining the use of Powershell, Sysmon, Windows Task Scheduler, and ELK. This approach will not only result in a minimum of logs to analyze, but it will also provide analysts with knowledge of specific offensive techniques and tactics, making the security team/SOC even more capable of detecting evil in their environment.

1. Introduction

The purpose of this paper is to outline how to optimize a SIEM using thought-out detection techniques. It thus seems appropriate first to define the technology behind the acronym: SIEM. According to Gartner:

"Security information and event management (SIEM) technology supports threat detection, compliance and security incident management through the collection and analysis (both near real time and historical) of security events, as well as a wide variety of other event and contextual data sources. " (Gartner, n.d.-a)

"SIEM technology aggregates event data produced by security devices, network infrastructure, systems and applications. The primary data source is log data, but SIEM technology can also process other forms of data, such as network telemetry. Event data is combined with contextual information about users, assets, threats and vulnerabilities." (Gartner, n.d.-b)

From the above definitions, it is understandable why an organization would decide to buy a SIEM. It is very beneficial for any security team to have centralized log collection, and the ability to correlate security logs from various infrastructure components along with Netflow data from network equipment like routers and switches. More than that, a SIEM can be utilized to create sophisticated alerting rules, provide better reporting for the managerial staff, and, most importantly, gain visibility into the entire network. It is, therefore, no surprise that the market for SIEMs has been growing for the last couple of years, and will most likely continue to do so in the years to follow (Research and Markets, 2019).

The purpose of this paper is not to debate which SIEM is the best. Instead, the focus will be on how to optimize any SIEM by only collecting and using the logs needed for detecting malicious activity.

The approaches used in this paper are twofold; The MITRE ATT&CK™ Framework and purple teaming. MITRE describes The MITRE ATT&CK™ framework or *matrix for enterprises* in the following way:

Lasse Hauballe Jensen, lassehauballe@protonmail.com

"*MITRE's Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) is a curated knowledge base and model for cyber adversary behavior, reflecting the various phases of an adversary's attack lifecycle and the platforms they are known to target…* (Strom et al., 2018)

The framework is essentially a matrix where each cell represents an attack technique, and each column, a tactic. One way of using it to optimize the SIEM is to first pick-out a technique from the matrix, second try to learn as much as possible about the attack method, and lastly, create a way of detecting it and visualizing it in the SIEM. This way of structurally approaching the task of creating detection techniques, and mapping those on to the ATT&CK framework is a way for the security team to visualize which attacks they can and cannot yet detect in their network. Thus, ATT&CK can become a foundation for the rules in the SIEM and the corresponding detection techniques that triggers them. Using ATT&CK in this way will be a stable path to making the SIEM more tactical, faster, and defensive-oriented.

The technique in focus for this paper is *DLL Search Order Hijacking*, which, according to the ATT&CK framework, is used for evasion, privilege escalation, and persistence. The question now becomes, how does one create a viable detection method for a given offensive technique?

The approach used in this paper, for creating detection techniques is called *purple teaming*. The name derives from combining the two traditional methodologies in IT-security, namely *Red Teaming* and *Blue Teaming*. The first being the offensive-oriented pentesting team, the latter being the SIEM-focused defensive team. Traditional in-house security teams will often focus on the defensive side exclusively. Such groups might never find it necessary to spin-up a Kali Linux host to test a vulnerability themselves. Instead, they will rather rely on outside red team consultants that will routinely scan the network, try different techniques, write a report, and hand it back to the in-house blue team. This division of labor has been accepted for many years, but it might not be the most effective way of approaching the SIEM:

"*In our current security climate, the traditional adversarial relationship does not work. Pitting the Red Team against the Blue Team is a thing of the past. A better path*

forward is to intertwine the two units to create a 'Purple Team' to allow the two teams to cooperate in much greater detail and enable much better results. " (Dale, 2019)

The purple team approach used in this paper will hopefully demonstrate Dale's point of view. First, creating a proof of concept (POC) will provide insight into how DLLs Search Order Hijacking functions and what indicators of compromises (IOC's) the attack leaves behind. Second, these IOCs will then be used by the blue team to create a better and more thorough detection technique.

Note that detection methods should be continuously retested and reevaluated; thus, the Purple Team approach should be an ongoing process.

The first chapter of this paper will be an in-depth look into how DLL Search Order Hijacking functions, followed by a POC of running an attack on Windows 10 to gain privilege-escalation and persistence. In chapter 2, the IOC's from the POC are highlighted, and it is discussed why specific logs could be helpful, and others might not be. Then follows the defensive part of the paper, making sure that the logs are correct and shipped to the SIEM, where they will be processed, enriched, and visualized.

This paper will use the *ELK-stack,* commonly known as *ELK,* as the SIEM. According to Elastic's website, ELK is: "*... the acronym for three open source projects: Elasticsearch, Logstash, and Kibana. Elasticsearch is a search and analytics engine. Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch. Kibana lets users visualize data with charts and graphs in Elasticsearch.*" (Elastic, n.d.-a) ELK is unique in that it is highly customizable and can be used for a wide variety of use-cases other than a SIEM. The ELK stack is not branded as a SIEM; however, it is very capable of operating as such.

2. DLL Search Order Hijacking (T1038)

2.1. What are DLLs and the DLL Search Order?

Before showing how DLL Search Order Hijacking works, it is crucial first to explain the terms *DLL* and *DLL Search Order*. It will be challenging to understand why hijacking a DLL file would be possible without understanding those terms in-depth.

Dynamic Link Libraries or DLLs are a standard filetype in Windows operating systems. According to Microsoft, a DLL "*is a library that contains code and data that can be used by more than one program at the same time. For example, in Windows operating systems, the Comdlg32 DLL performs common dialog box related functions*." (Microsoft, n.d.) For a programmer in need of a common dialog box, he or she can just load the DLL file and all of its functionalities without having to program it themselves. Naturally, this makes DLLs precious building blocks for any program running in a Windows operating system.

Having an understanding of what a DLL file is, it is now possible to go into details regarding how Windows finds and loads the exact DLL that a specific program is trying to use. Again, looking at the Microsoft documentation:

"*A system can contain multiple versions of the same dynamic-link library (DLL). Applications can control the location from which a DLL is loaded by specifying a full path or using another mechanism such as a manifest. If these methods are not used, the system searches for the DLL at load time…*" (Microsoft, 2018a)

So first off, the programmer could decide to hard-code the location of the DLL that he or she would like to use. Loading a DLL in C++ is done using the function LoadLibrary. In C++, hardcoding the path to the DLL comdlg32 would thus look like this:

```
LoadLibrary(L“C:\Windows\SysWOW64\Comdlg32.dll”);
```

One problem with hardcoding the path is that a version of Comdlg32.dll is located in C:\Windows\System32, which would be needed if the system was a 32-bit Windows version. Hardcoding the path would, therefore, make the program unavailable for 64-bit versions or vice-versa. Instead, what programmers tend to do is write:

LoadLibrary(L"Comdlg32.dll");

In this case, Windows will go through a couple of steps to try and find the correct DLL. First, Windows will check to see if the system already has another DLL loaded with the same name. If this is true, the system will run the function from the DLL in memory. However, if this is not the case, Windows will instead look into the registry key called *KnownDLLs* (Microsoft, 2018a). According to Microsoft's documentation regarding known DLLs:

"If the DLL is on the list of known DLLs for the version of Windows on which the application is running, the system uses its copy of the known DLL (and the known DLLs dependent DLLs, if any) instead of searching for the DLL." (Microsoft, 2018a)

Lastly, if the KnownDLLs registry key does not contain the DLL, Windows will invoke the so-called *DLL Search Order*. The DLL Search Order is just a list of locations where the operating system will look for a DLL with the name corresponding to the filename requested by the program. The DLL Search Order is as follow:

1. The directory from which the application loaded.
2. The system directory.
3. The 16-bit system directory.
4. The Windows directory.
5. The current directory.
6. The directories listed in the PATH environment variable. (Microsoft, 2018b)

The DLL Search Order listed above assumes two things to be true: 1) *SafeDLLSearchMode* is enabled, which has been the default since Windows XP Service Pack 2. If this is not the case, then the current directory will be checked before the 16-bit system directory. 2) The process looking for the DLL is a *Desktop Application* and not a *Windows Store App*. This paper will only focus on Desktop Applications running in Microsoft Windows 10 since most organizations and enterprises will most likely not be using Windows Store Apps.

It is essential to highlight that Windows is only trying to match the name of the requested DLL. In other words, if the system stumbles upon an incorrect DLL with the

same name, then the system will load the incorrect one regardless. The system will not check for either the MD5-sum of the file, the publisher, file version, or whether the desired function is found within the DLL.

That means that the attacker will need the following: 1) a process requesting a DLL using the DLL Search Order, 2) write-permissions to a folder within the DLL Search Order, and 3) the name of the DLL required by the process. It might seem unlikely for an attacker to have all three things; however, it is quite common for a process to look for a DLL but never find it, and to have directories within the DLL Search Order that unprivileged users can modify. Therefore, this makes it a very valid path for attackers to take when looking to create persistence, gain privilege escalation, and evade detection. Lastly, an interesting fact to keep in mind is that many processes will run fine without ever finding all their required DLLs, and many will even function having loaded a wrong DLL. However, it also occurs that a process crashes or even makes the entire system crash, so be aware of testing for this in an environment that can handle a crash.

Placing a malicious DLL within a directory in the DLL Search Order is known as *DLL Search Order Hijacking*[1] and is labeled T1038 in the MITRE ATT&CK Framework (Kanthak et al., 2019). MITRE also describes the technique in the following way:

"*If a search order-vulnerable program is configured to run at a higher privilege level, then the adversary-controlled DLL that is loaded will be executed at the higher level. In this case, the technique could be used for privilege escalation from user to administrator or SYSTEM or from administrator to SYSTEM, depending on the program.*" (Kanthak et al., 2019).

Knowing the terms DLL, DLL Search Order, and having an understanding as to why attackers exploit this normal Windows behavior, it is time to look at a proof of concept. The purpose of this POC is not only to demonstrate a DLL Search Order Hijacking attack but also to help answer the following questions: At what stage would a SIEM analyst be able to detect it? What IOCs does the attack leave behind? Does the

[1] It is also referred to as *DLL preloading* or *binary planting attack.*

system need any specific configurations? Furthermore, what defensive technologies will the security team need to implement to detect DLL Search Order Hijacking?

2.2. POC: Hijacking a DLL for svchost.exe and gaining SYSTEM on Windows 10

From the previous chapter, we learned that the attacker would first need to locate a process that is invoking the DLL Search Order. One way to find such a process is to use the Windows Sysinternals tool called *Process Monitor* (Microsoft, 2019), which can help visualize how processes are behaving on the system. Process Monitor will monitor multiple things, including whether a process is trying to load a file, reading or editing a registry key. Still, most importantly, it will show if a process is looking for a DLL and whether it loaded correctly or never found it. Process Monitor is a valuable tool to have as a security team; though, it is unlikely that an attacker would use it since offensive toolkits like PowerSploit have this feature integrated (Schroeder, 2020).

Process Monitor comes with a feature called *Boot Logging*, which allows Process Monitor to log everything happening while the system is booting. It is during the booting process that many high privileged services start in the context of NT Authority\SYSTEM. Boot Logging thus makes for an excellent way to find privilege escalation opportunities on the system. Figure 1 is the result of having enabled Boot Logging within Process Monitor. Note that the result shown is filtered: Only show paths ending in ".dll" and only show results if the field *Result* is equal to "NAME NOT FOUND."

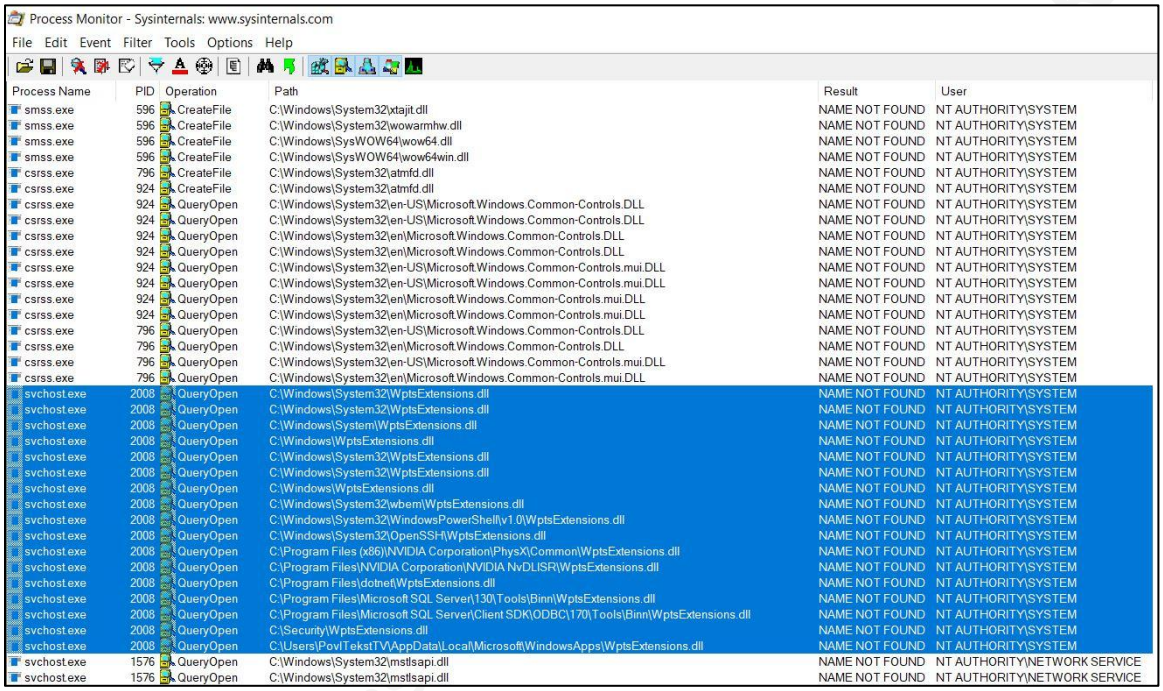

Figure 1: Results from Process Monitors Boot Logging

The highlighted results in figure 1 show that the process named *svchost.exe* is looking for a DLL called WptsExtensions.dll, and not finding it. Figure 1 also shows that the process is being run as NT Authority\SYSTEM, making it a highly valuable process to hijack. Lastly, we see that the process is going through the DLL Search Order as expected. Notice that individual paths look unique for the victim PC. These are the directories listed in the so-called System PATH environment variable, which is the last step of the DLL Search Order. In this case, the victim-PC seems to have multiple directories listed, including directories belonging to Nvidia, Microsoft SQL, and dotnet. It is important to note that two distinct PATH environment variables exist in Windows operating systems. The first is the *system* PATH environment variable, and the second is the *user* PATH environment variable. The *system* PATH variable is for system-wide usage, whereas the *user* variable is user-specific. It is essential to understand this distinction because process running as SYSTEM will only look in the PATH variable, whereas process running as the current user will look in both *system* and *user*. If an attacker is looking to privilege-escalate using DLL Search Order Hijacking, they would have to find a writeable directory in the System PATH variable. However, if the attacker

is only using DLL Search Order Hijacking for evasion purposes, they could rely on both the System and the User environment variable. Thus, they would not need to hijack a process running as SYSTEM; however, it would be preferable for an attacker. The POC in this paper will focus on gaining privilege-escalation, and thus, the focus will be on the System PATH environment variable.

The next step is to determine if the current low-privileged user has permission to write files to a directory in the DLL Search Order. One way to test this is to create a simple PowerShell script like the one showcased in figure 2. Scripting is not necessary, but it can be a swift and practical approach to take. The script used only looks for vulnerable directories within the system PATH environment variable, and not the full DLL Search Order, thus, it assumes that a regular user does not have permission to write files to directories like C:\Windows or C:\Windows\System32.

The script is straightforward. It will go through each directory in the System PATH variable and check to see if users other than those listed in *$ignoreUsers* have one of the following permissions: Modify, Write or FullControl. If this is true, the script will write the directory to the screen.

```
$Reg = "Registry::HKLM\System\CurrentControlSet\Control\Session Manager\Environment"
(Get-ItemProperty -Path "$Reg" -Name PATH).Path.Split(";") | ForEach {
    $currentPath = $_
    if ($_ -ne "") {
        (get-acl $_).access | Where-Object {$_.FileSystemRights -like "Write*" -or
$_.FileSystemRights -like "FullControl" -or $_.FileSystemRights -like "Modify*"} |
ForEach {
            $user = $_.IdentityReference
            $permission = $_.FileSystemRights

            #Ignore the following privileged users.
            $ignoreUsers = @("NT SERVICE\TrustedInstaller", "NT AUTHORITY\SYSTEM",
"BUILTIN\Administrators", "APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES")
            if ($ignoreUsers -notcontains $user) {
                echo "[VULNERABLE] $user can add files to $currentPath"
            }
        }
    }
}
```

Figure 2: Detect vulnerable directories

Figure 3 is the result of running the script on the victim-PC, we see that the group *Authenticated Users* has permission to write files to the folder C:\Security\, which has been added to the System PATH variable for testing purposes.

```
PS C:\Users\PovlTekstTV\Desktop> .\T1038_Detect_Vulnerable_Directories.ps1
[VULNERABLE] NT AUTHORITY\Authenticated Users can add files to C:\Security\
```

Figure 3: Output from the Powershell script

The next step is to create a custom DLL that will showcase that the privilege escalation happened. The DLL will be created using Microsoft Visual Studio 2017. The DLL will have a single function: To output the user running the DLL (i.e. SYSTEM) to a text file called "C:\Security\whoami.txt." The DLL is simple, yet it will show that code-execution happened and that privilege escalation was possible. Figure 4 is the code used for this POC.

```
// dllmain.cpp : Defines the entry point for the DLL application.
#include "pch.h"
#include <stdio.h>
#include <windows.h>
#include <Lmcons.h>
#include <fstream>

BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
    )
{
    std::ofstream outdata;
    TCHAR username[UNLEN + 1];
    DWORD size = UNLEN + 1;
    GetUserName((TCHAR*)username, &size);

    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        //Write to a file called whoami.txt, who the user is.
        outdata.open("c:\\security\\whoami.txt");
        if (outdata.is_open()) {
            outdata << username;
            outdata.close();
        }

    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

Figure 4: C++ code for the DLL used in the proof of concept

The DLL is compiled and placed in the directory C:\Security\ and named *WptsExtensions.dll*. Afterward, the computer is restarted.

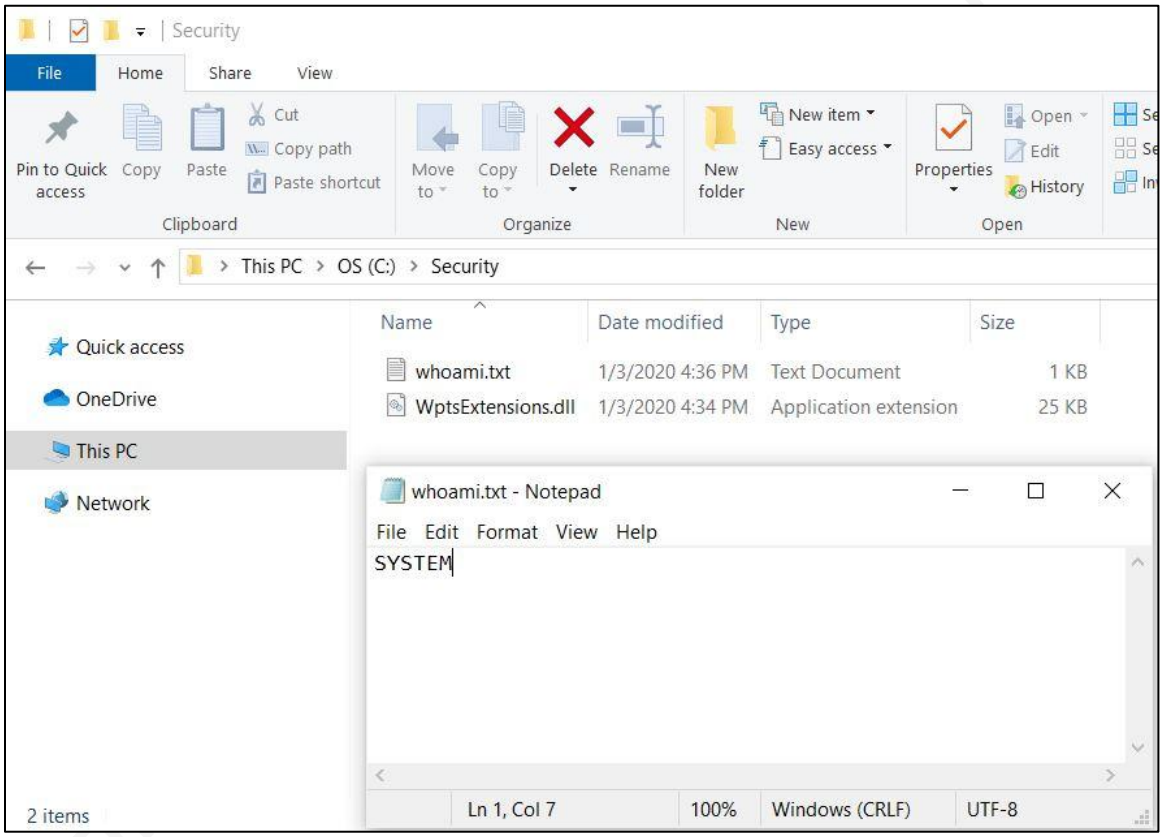

Figure 5: NT SYSTEM ran the DLL and created whoami.txt

Figure 5 should be pretty self-explanatory at this point. Upon restarting, svchost.exe looked for WptsExtensions.dll, it searched the DLL Search Order, and found the malicious DLL in C:\Security\, which it loaded thus executing our code. The execution wrote the name of the current user: SYSTEM to the file whoami.txt. Once again, we begin Process Monitor with boot logging enabled.

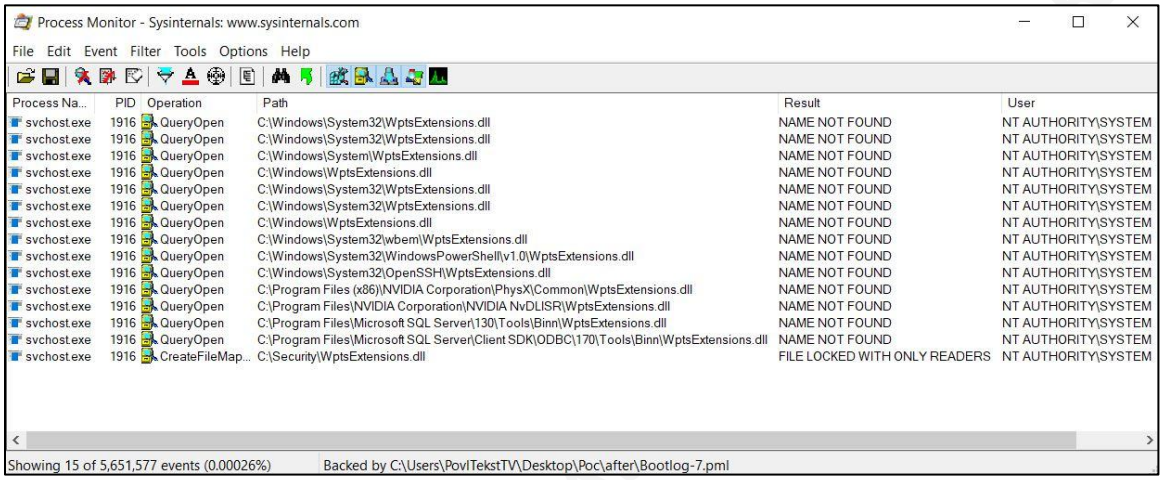

Figure 6: Process monitor showing WptsExtension.dll loaded from the c:\security directory

As expected, we see that the process svchost.exe went through the DLL Search Order until it reached the malicious DLL located in C:\Security. As expected, the system booted with no issues, and so an attacker would be able to run this DLL Search Order Hijacking attack without the user ever noticing.

3. Creating the detection technique

Having a thorough understanding of how DLL Search Order Hijacking works and having gone through the red team exercise of creating a proof of concept, it is now time to figure out how to create a detection technique that will work with the SIEM. To do this, we will try to adhere to the following three key points: First, the analyst looking in the SIEM should have to go through the fewest logs possible. Secondly, the risk of false/positives should be minimal, and thirdly, if possible, it should be evident right away what technique in the MITRE ATT&CK™ matrix was used, in this case, T1038 or DLL Search Order Hijacking. If we manage to meet these critical points, the analyst will be able to react swiftly and appropriately following the appropriate incident response process.

Lasse Hauballe Jensen, lassehauballe@protonmail.com

3.1. Audit the DLL Search Order using Powershell

From chapter 1, we learned that for the attacker to succeed with DLL Search Order Hijacking, he or she would have to place a malicious DLL inside the DLL Search Order. In Windows operating systems, it is possible to create logging on folders and files using *Object Access Logging.* Now, in theory, it would be possible to have Object Access Logging on every folder on the operating system; however, anyone with Object Access Logging experience would properly advise heavily against it due to the number of logs generated. Therefore the goal is only to enable Object Access Logging on those specific directories listed in the DLL Search Order. Furthermore, it is only necessary to enable on those folders, where non-privileged users have write-permissions.

In the proof of concept example, we found that only one directory "C:\Security\" was vulnerable, and so this is the only folder that needs to have Object Access Logging enabled. Two things are necessary to achieve this: 1) Create a Local or Group Policy and make sure to set *Audit File System* to *Success and Failure* under "Computer Configuration/Windows Settings/Security Settings/Advanced Audit Policy." The Audit File System policy allows for the system to generate Object Access Logging events in the Windows Security Log. 2) By default, Windows will not audit any folder without a *SACL* or *System Access Control List*. According to Microsoft, a SACL "*enables administrators to log attempts to access a secured object. Each ACL specifies the types of access attempts by a specified trustee that cause the system to generate a record in the security log.*" (Microsoft, 2018c) SACLs can be created manually in Windows; however, it is also possible to extend the PowerShell script from the proof of concept, to enable auditing on

vulnerable folders. For this to work, make sure the script runs as an administrator.

```
$Reg = "Registry::HKLM\System\CurrentControlSet\Control\Session Manager\Environment"
(Get-ItemProperty -Path "$Reg" -Name PATH).Path.Split(";") | ForEach {
    $currentPath = $_
    if ($_ -ne "") {
        (get-acl $_).access | where-Object {$_.FileSystemRights -like "Write*" -or
        $_.FileSystemRights -like "FullControl" -or $_.FileSystemRights -like "Modify*"}
        ForEach {
            $user = $_.IdentityReference
            $permission = $_.FileSystemRights
            #The following users will have permissions and should be ignored
            $ignoreUsers = @("NT SERVICE\TrustedInstaller", "NT AUTHORITY\SYSTEM",
                             "BUILTIN\Administrators", "APPLICATION PACKAGE
                              AUTHORITY\ALL APPLICATION PACKAGES")

            if ($ignoreUsers -notcontains $user) {
                echo "[VULNERABLE] $user can write to $currentPath"
                #Set Audit Rules
                $ACL = Get-Acl $currentPath
                $AuditRule = New-Object Security.AccessControl.FileSystemAuditRule(
                                        "Everyone","Write","None","None","success")
                $ACL.SetAuditRule($AuditRule)
                $ACL | Set-Acl $currentPath
                echo "SACL has now been added for the directory $currentPath"
            }
        }
    }
}
```

Figure 7: Enable Auditing by adding SACL to the directory

The extended script will go through each directory in the system PATH environment variable and add a SACL to it. The script could be moderated further only to fit certain users; however, for the sake of simplicity and thoroughness, *Everyone* will work as intended. It could also be extended to prevent DLL Search Order Hijacking by restricting access for unprivileged-users. The preventive approach will only make sense if it is clear that this would not disrupt the users' ability to work. Lastly, it should be noted that if a directory is later removed from the system PATH environment variable or if non-privileged users can no longer add files to it, the audit settings would remain. This problem could be solved by having a similar script that, instead of adding audit settings, would remove them if no longer needed.

In a Windows Domain, this PowerShell script could be placed on a shared network folder (with read-only), and have a Group Policy created to run it for every client on startup. This approach would ensure that every Windows workstation and server in the network would only generate the necessary Object Access logs.

With the correct logging in place on the Windows clients, the logs are now ready to be forwarded to the SIEM. How to achieve this often depends on the SIEM in use since most will have a product-specific *log agent* or *log shipper*; QRadar uses WinCollect, Splunk uses Universal Forwarder, and so on. Most SIEMs will, however, receive incoming logs just fine no matter the agent as long as the formatting is correct. The log agent used for this paper is *NXLog*, which is a highly customizable agent, that can be made to only forward specific logs, and do so in the format most suitable for the SIEM like CSV, Syslog, and JSON (NXLog, 2018). In ELK, it is easy to use the JSON-format because the ingestion engine in ELK, called Logstash, can automatically parse JSON. Using Logstash to accept, parse, and enrich the logs will be the focus of the next section.

3.2. Tagging in Logstash and visualizing in Kibana

One of the things that make the ELK-stack stand out from other SIEMs is Logstash: "*a server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to your favorite 'stash'*" (Elastic, n.d.-b). Logstash parses and enriches logs with information like geo and DNS lookup on IPs. It can tag individual logs and add entirely new fields with information that will help the analyst understand the event better. It is important to note that Logstash can work with any SIEM; it is just a matter of placing it in between the log agent and the SIEM. Figure 8 is an example of setting up a Logstash configuration filter for Windows Object Access Log event ID 4663.

```
filter {
    date {
        match => ["EventTime", "YYYY-MM-dd HH:mm:ss"]
    }

    if [EventID] == 4663 {
        if [AccessMask] == "0x2" {
            if [ObjectName] =~ ".dll" {
                    mutate {
                        add_tag => ["Alert", "MITRE"]
                        add_field => { "MITRE" => "T1308" }
                    }
            }
        }
    }
}
```

Figure 8: Logstash configuration filter

The filter should be quite self-explanatory, ignoring the date filter at the top, the configuration will look for a log with the EventID value of 4663 (An attempt was made to access an object). It then checks to see if the *AcesssMask* is "0x2", which, according to Microsoft, is "WriteData (or Add file)" meaning a file was added to the directory (Microsoft, 2017). Lastly, Logstash will look at the field *ObjectName* and check if the value of the file contains ".dll." If the event meets these criteria, Logstash will then enrich the log with the tags: "Alert" and "MITRE", and create a new field called MITRE with the value "T1308". Configuring the filter to drop events not containing ".dll" is also an option that naturally would free up even more space in the SIEM.

Turning the focus to Kibana, our SIEM frontend, we see that event logs from the Windows client are visible. Figure 9 shows that Logstash parsed the event correctly and created the field "MITRE" with T1308 as the value.

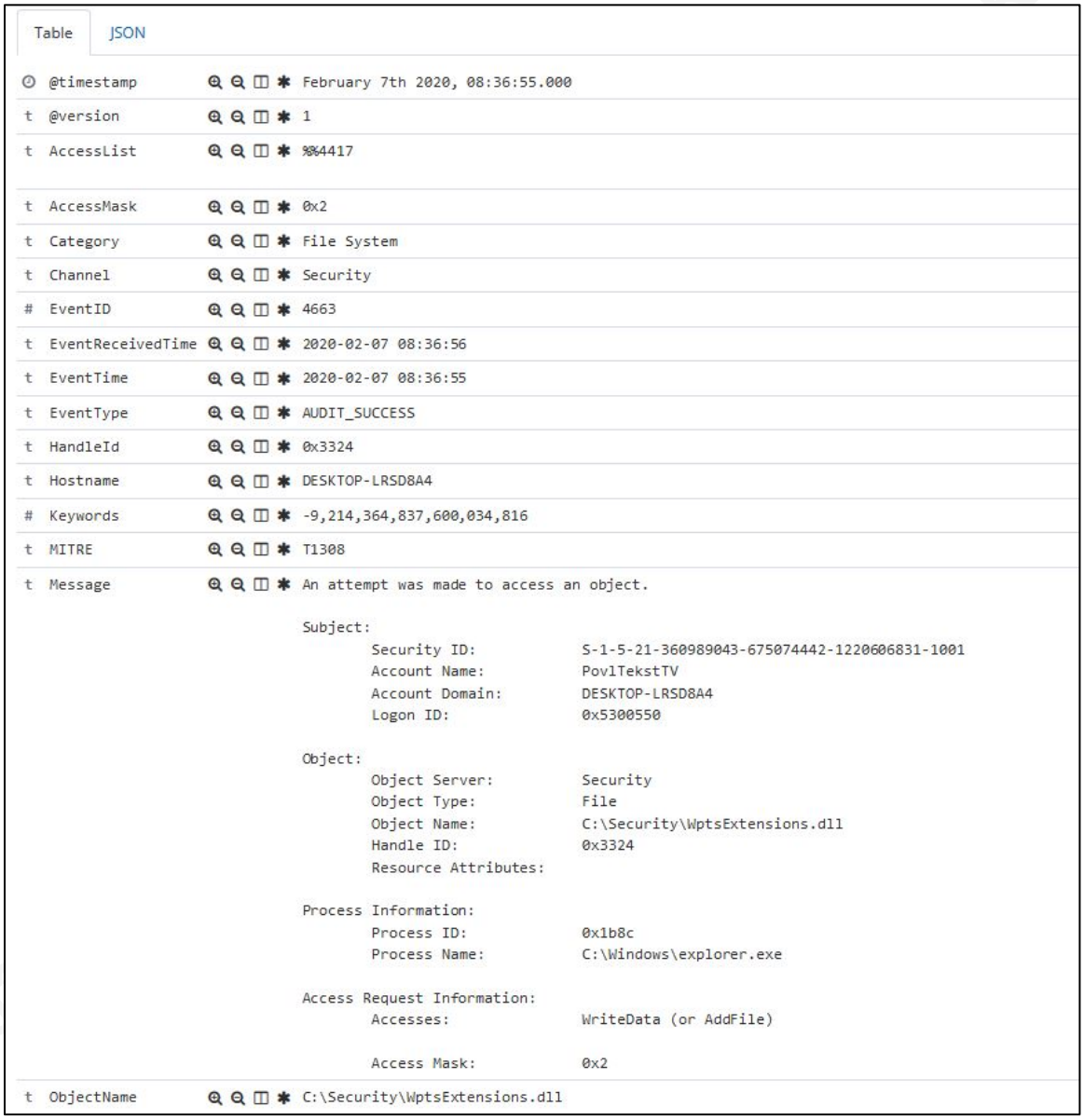

Table JSON

@timestamp		February 7th 2020, 08:36:55.000
@version		1
AccessList		%%4417
AccessMask		0x2
Category		File System
Channel		Security
EventID		4663
EventReceivedTime		2020-02-07 08:36:56
EventTime		2020-02-07 08:36:55
EventType		AUDIT_SUCCESS
HandleId		0x3324
Hostname		DESKTOP-LRSD8A4
Keywords		-9,214,364,837,600,034,816
MITRE		T1308
Message		An attempt was made to access an object. Subject: Security ID: S-1-5-21-360989043-675074442-1220606831-1001 Account Name: PovlTekstTV Account Domain: DESKTOP-LRSD8A4 Logon ID: 0x5300550 Object: Object Server: Security Object Type: File Object Name: C:\Security\WptsExtensions.dll Handle ID: 0x3324 Resource Attributes: Process Information: Process ID: 0x1b8c Process Name: C:\Windows\explorer.exe Access Request Information: Accesses: WriteData (or AddFile) Access Mask: 0x2
ObjectName		C:\Security\WptsExtensions.dll

Figure 9: Event ID 4663 in Kibana

Now, there is no right or wrong way to create dashboards and visualizations. The most important thing is to make sure the analyst has a comfortable and clean overview, and that he or she knows what is happening on the screen. Many SIEMs will come pre-installed with a bunch of cluttered graphs and colorful visuals, which might work when selling the product, but they will most likely only remove focus and cause further confusion for the analyst.

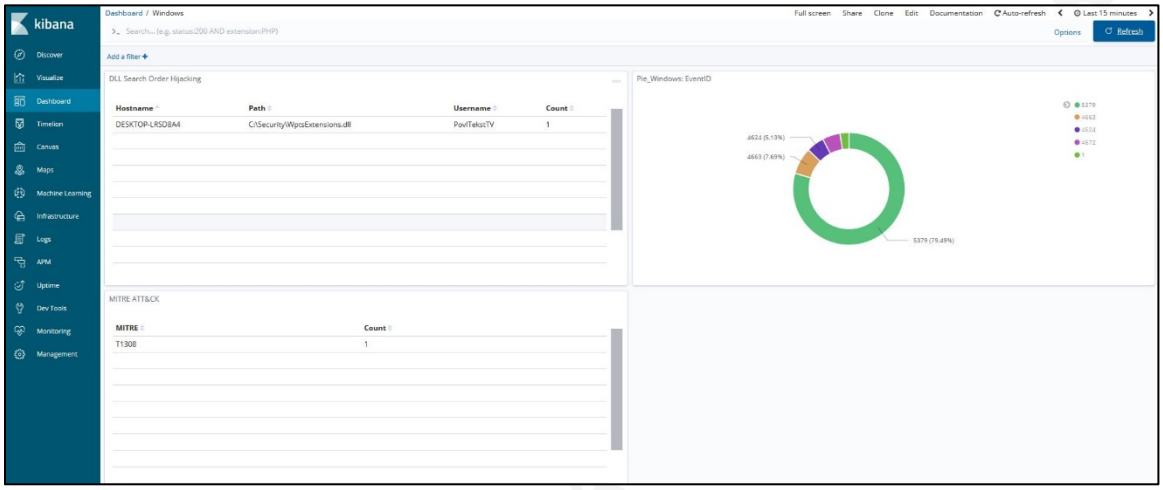

Figure 10: Visualization in Kibana

Figure 10 is an example of a Kibana dashboard called *Windows* created to give a simple overview of the Windows clients in the monitored environment, which in this case, is only a single client. Depending on the number of clients, servers, domains, and networks being monitored, the dashboard might be too simplistic; However, the idea should remain the same. The dashboard contains three visualizations: The top-left visualization is for DLL Search Order Hijacking only. The table will only show events if they contain "MITRE: T1038", in that case, it will show the hostname, the username, and the name and location of the DLL. For an analyst, this makes detecting DLL Search Order Hijacking a lot easier, and he or she can begin the incident response process right away, knowing that this is most likely an actual threat. This particular detection technique does not reveal to the analyst what process was attempted exploited or if it was successful. However, for an analyst charged with managing this incident, those questions should be secondary. The visualization in the bottom-left corner shows how many MITRE ATT&CK detections that have been triggered on Windows hosts. The visualization is great for creating security reports on the condition of the entire Windows environment. Last, the top-right corner will show the analyst which Windows events are most common. This visualization is valuable to look at in the beginning, since Windows Object Access Logging can become very noisy if configured incorrectly. Having an overview close

to the other visualization might help in detecting false/positives or other misconfigurations.

The visualization in the example only monitored a single Windows host. In reality, a security team could be tasked to monitor hundreds, thousands or even tens of thousands making the visualization provided seem very simplistic. Since DLL Search Order Hijacking is not commonly an attack vector used on multiple systems at a time, it really should not be an issue for the analysts to keep up with the number of alerts generated by this technique. Therefore, the detection technique provided is valid even for organizations with thousands of clients and servers.

3.3. Keeping up with the PATH environment variable

The paper up until now has consisted of a red team exercise followed by a blue team exercise: Creating a proof of concept, adapting the findings into blue team knowledge, and finally creating the detection technique. Now the task is to circle back again and ask: "Does this detection technique contain flaws"?

One thing that will inevitably make this detection technique obsolete is the fact that a Windows environment will eventually change, and with it, so will the DLL Search Order since new directories will appear in the PATH system environmental variable. Specific programs will only work if certain directories are added to the System PATH environment variable, and so the DLL Search Order will change.

Luckily, Microsoft Sysinternals have made another tool called *Sysmon* that is gaining popularity rapidly. Sysmon allows for enhancing logging capabilities further in Windows operating systems and makes it possible for administrators to select what to monitor in a way that Windows cannot do on its own. This could include logging when specific processes or subprocesses are running, keeping an eye on network connections, or monitor changes to particular registry keys. Earlier, we learned that the system PATH environment variable is stored in the registry key located in HKLM\System\CurrentControlSet\Control\Session Manger\Environment (Microsoft, 2018d). So to monitor changes to the system environment PATH variable, all one needs to do is set up a Sysmon configuration to track changes made to this key:

```
<Sysmon schemaversion="4.21">
    <HashAlgorithms>md5,sha256</HashAlgorithms>
    <EventFiltering>
        <!-- Audit the following Registry Event -->
        <RegistryEvent onmatch="include">
            <TargetObject condition="end with">HKLM\System\CurrentControlSet\Control\Session Manager\Environment\Path</TargetObject>
        </RegistryEvent>
    </EventFiltering>
</Sysmon>
```

Figure 11: A Sysmon configuration file for monitoring changes to the PATH variable

Now, if someone makes a change to the System PATH environment variable, Sysmon will create event ID 13. Figure 12 is an example of such an event. Here the directory "C:\Security\Sysmon" was added as an example. We also see that the event includes all the directories in the System PATH environment variable.

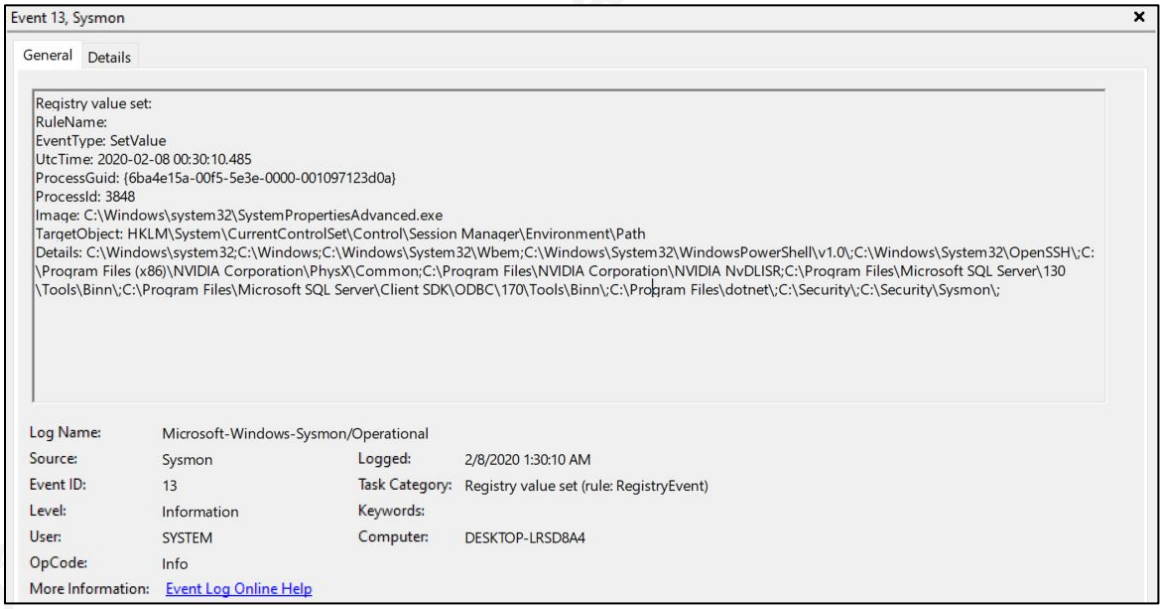

Figure 12: Sysmon Event ID 13: A registry value changed

Knowing that Sysmon can create this event makes it possible only to run the PowerShell audit-script when necessary. New directories added to the system PATH variable would then automatically be given the correct SACL and have object access logging performed. Sysmon could be skipped entirely, and instead, just have the script run on every startup. Now, this would work though it might be a bad idea if the Windows environment is large while at the same time, the security team is trying to pull in every PowerShell log. In that case, the number of logs created every day from running this script could become a problem. To have the PowerShell script execute when a specific event is generated, we turn to an old Windows Application called *Task Scheduler* known

to many since the 1990s. Windows Task Scheduler can trigger an action based on an event in the Windows Event Log, for instance, a Sysmon event containing a specified string. Figure 13 shows how to accomplish this using XPath queries in Task Scheduler.

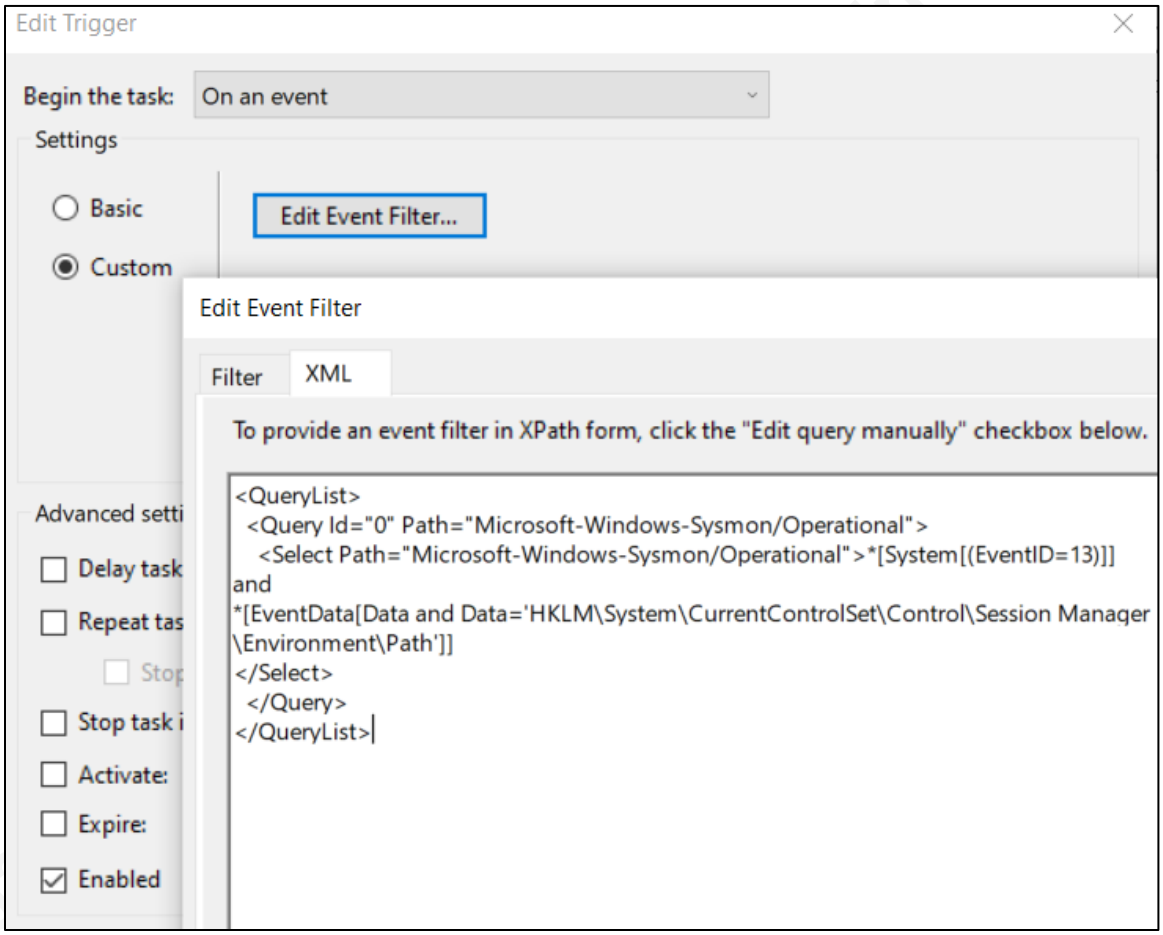

Figure 13: Adding customized Windows event trigger using XPath

The above task will only trigger on Sysmon Event ID 13 and only if the event contains the string: "HKLM\System\CurrentControlSet\Control\Session Manager\Environment\Path". Upon seeing this event, Windows Task Scheduler will run PowerShell as an elevated user and execute the audit-script. With this final step implemented, we now have a complete automatic DLL Search Order Hijacking detection technique.

4. Conclusion

DLL Search Order Hijacking is an attack technique that, without the proper understanding of the intricacies of the attack, can be very hard to detect in a SIEM. The goal of this paper was to use the MITRE ATT&CK framework as a foundation, pick out a single attack technique (in this case T1038), and then approach it using a purple team approach to create a reliable detection technique. The detection technique had to live up specific demands: The number of logs shipped to the SIEM had to be as few as possible. Secondly, the risk of a false/positive should be minimal. Lastly, since the foundation of the approach was the MITRE ATT&CK framework, it should be evident right away for the analyst that the logs were referring to T1038 making for a faster incident response process.

The purple team approach used in this paper helped us understand how and why the attack works by going through a full red team proof of concept. Knowing how the attack works, along with the indicators of compromise from the POC, made it possible to create a detection technique that lived up to the demands proposed in this paper. By using tools like PowerShell, Sysmon, NXLog, and Windows Task Scheduler, the SIEM analyst ended up with efficient visualizations capable of detecting DLL Search Order Hijacking. The technique does not identify whether the attacker was successful in loading a malicious DLL; however, it will detect the attempt, which hopefully will be enough for most security teams.

The SIEM is an excellent tool for security teams looking to enhance their network visibility, and strengthen the organization's overall defensive posture, but keep in mind that a SIEM is only as good as the logs it receives.

Lasse Hauballe Jensen, lassehauballe@protonmail.com

References

Dale, Chris. (2019). Red, Blue and Purple Teams: Combining Your Security Capabilities for the Best Outcome. Retrieved from SANS.org: https://www.sans.org/reading-room/whitepapers/analyst/red-blue-purple-teams-combining-security-capabilities-outcome-39190

Elastic. (n.d.-a). What is the ELK Stack? Retrieved from: https://www.elastic.co/what-is/elk-stack

Elastic. (n.d.-b). Logstash. Retrieved from Elastic: https://www.elastic.co/logstash

Gartner. (n.d.-a). Gartner Glossary: S: Security Information and Event Management (SIEM). Retrieved from https://www.gartner.com/en/information-technology/glossary/security-information-and-event-management-siem

Gartner. (n.d.-b). Security Information and Event Management Market. Retrieved from: https://www.gartner.com/reviews/market/security-information-event-management

Microsoft. (n.d.). What is a DLL? Retrieved from: https://support.microsoft.com/da-dk/help/815065/what-is-a-dll

Microsoft. (2017). 4663(S): An attempt was made to access an object. Retrieved from: https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4663

Microsoft. (2018a). Dynamic-Link Library Search Order. Retrieved from: https://docs.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order

Microsoft. (2018b). Dynamic-Link Library Security. Retrieved from: https://docs.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security

Microsoft. (2018c). Access Control Lists. Retrieved from: https://docs.microsoft.com/en-us/windows/win32/secauthz/access-control-lists

Lasse Hauballe Jensen, lassehauballe@protonmail.com

Microsoft. (2018d). Environment Variables. Retrieved from Microsoft: https://docs.microsoft.com/en-us/windows/win32/procthread/environment-variables

Microsoft. (2019). Process Monitor v.3.53. Retrieved from: https://docs.microsoft.com/en-us/sysinternals/downloads/procmon

MITRE. (n.d.). ATT&CK. Retrieved from: https://attack.mitre.org/

NXLog. (2018). NXLog Community Edition Reference Manual v. 2.10.2150. Retrieved from: https://nxlog.co/docs/nxlog-ce/nxlog-reference-manual.html

Research and Markets. (2019). Security Information and Event Management Market to 2027 – Global Analysis and Forecast by Solution; Service; and End User. Retrieved from: https://www.researchandmarkets.com/reports/4762286/security-information-and-event-management-market

Schroeder, Will. (n.d.). PowerSploit Documentation. Retrieved from: https://powersploit.readthedocs.io/en/latest/Privesc/Find-ProcessDLLHijack/

Stefan Kanthak et al. (2019). DLL Search Order Hijacking. Retrieved from: https://attack.mitre.org/techniques/T1038/

Strom, Blake E. et al. (2018). MITRE ATT&CK™: Design and Philosophy. Retrieved from https://www.mitre.org/sites/default/files/publications/pr-18-0944-11-mitre-attack-design-and-philosophy.pdf