

# CS5332 – 2018, Individual Assignment 3 – Rootkit Deployment and Analysis

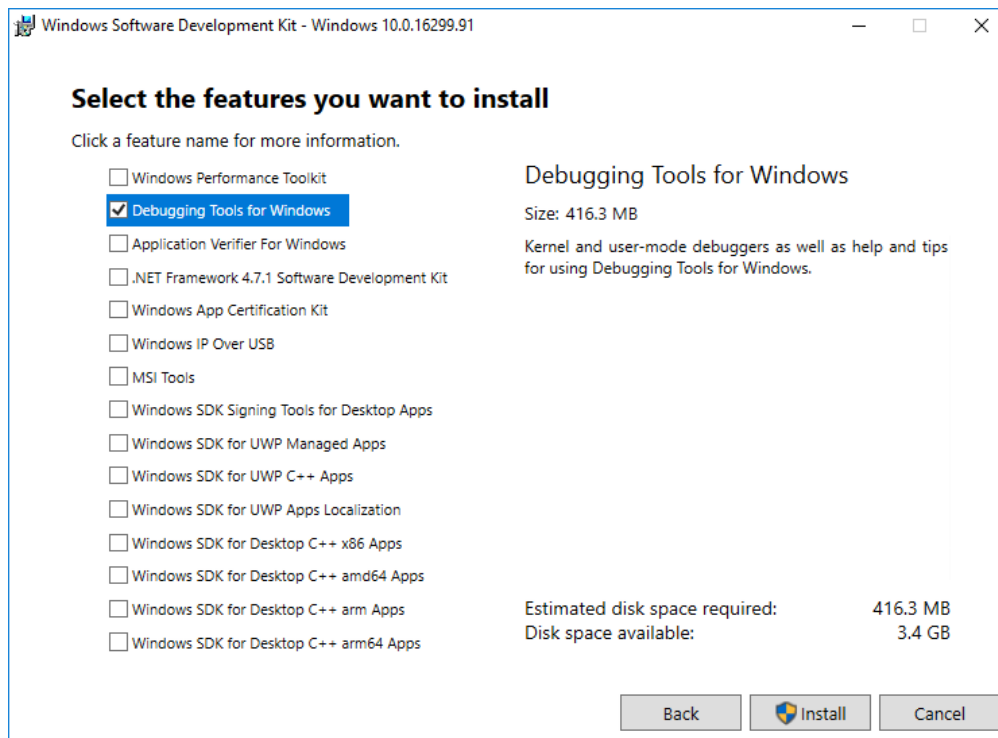
February 2018

## Python Remote Administration Tool ViRu5

This is a full undetectable python RAT which can bypass almost all antivirus and open a backdoor inside any windows machine which will establish a reverse https Metasploit connection to your listening machine.

## Memory dump on Windows 10 1709 x64

LiveKd seems to be the state-of-the-art tool to capture memory dumps of running Windows systems. To use LiveKd, the Debugging Tools for Windows have to be installed first:



Which seems to save the necessary tools to C:\Program Files (x86)\Windows Kits\10\Debuggers\x64. Next, download `livekd64.exe` to the same dir and call it from an admin prompt in the same dir; accept to load the debugging symbols:

```
Select Administrator: Command Prompt - livekd64.exe
C:\>cd C:\Program Files (x86)\Windows Kits\10\Debuggers\x64
C:\Program Files (x86)\Windows Kits\10\Debuggers\x64>livekd64.exe

LiveKd v5.62 - Execute kd/windbg on a live system
Sysinternals - www.sysinternals.com
Copyright (C) 2000-2016 Mark Russinovich and Ken Johnson

Symbols are not configured. Would you like LiveKd to set the _NT_SYMBOL_PATH
directory to reference the Microsoft symbol server so that symbols can be
obtained automatically? (y/n) y

Enter the folder to which symbols download (default is c:\symbols):
Launching C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\kd.exe:

Microsoft (R) Windows Debugger Version 10.0.16299.91 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Loading Dump File [C:\Windows\livekd.dmp]
Kernel Complete Dump File: Full address space is available

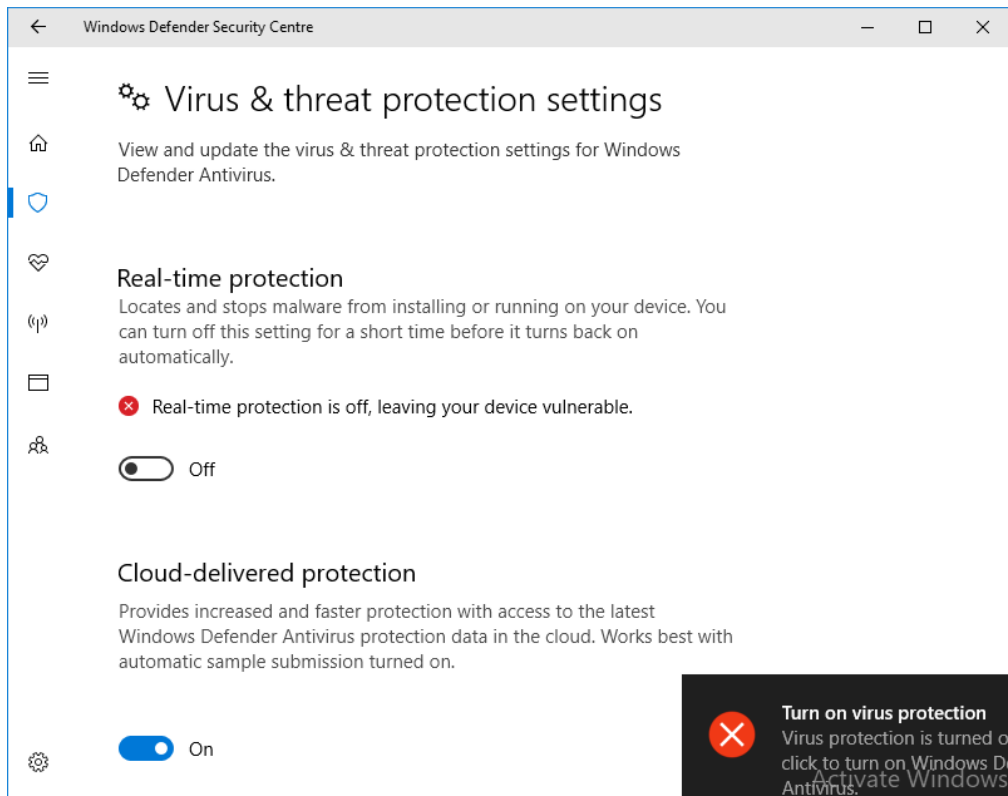
Comment: 'LiveKD live system view'

***** Path validation summary *****
Response          Time (ms)      Location
Deferred          1000          srv*c:\Symbols*http://msdl.microsoft.com/download/symbols
Symbol search path is: srv*c:\Symbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
```

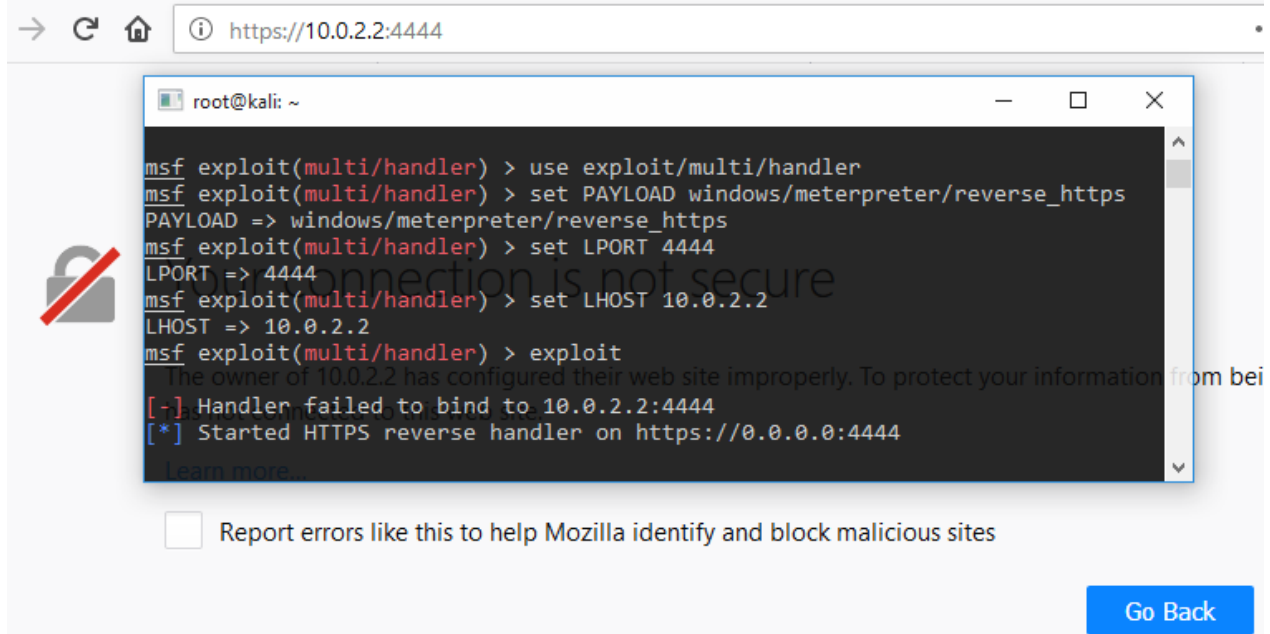
**Important:** Installation of a “Cumulative Update for Windows 10 1709” breaks the memory dump tool. Memory dumps can now be created with `.dump /o /f C:\full.dump`

## Building

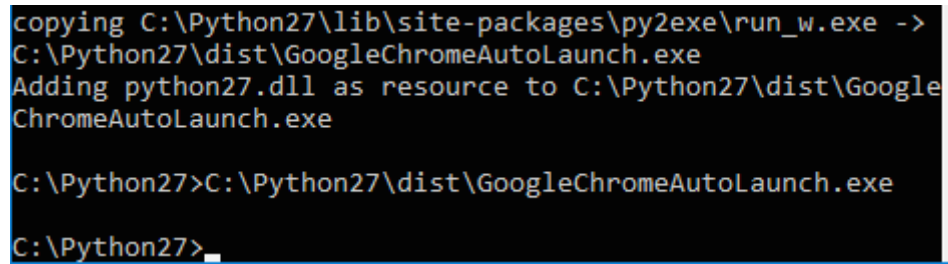
Download and extract the the Python Remote Administration Tool ViRu5. As Windows defender seems to recognize the rootkit as malicious, it needs to be disabled:



The rootkit opens a meterpreter HTTPS session, so first that needs to be started and reachable from the target machine. The server listens in a VM whose port 4444 is exposed to the host (which in turn is reachable from the Windows VM as 10.0.2.2). The certificate warning in Firefox demonstrates that the general setup works:



But then building the rootkit keeps failing. After installing a couple of Python modules, the final error is:

A screenshot of a Windows command prompt window with a black background and white text. The text shows the process of using py2exe to create a Windows executable from a Python script. The first line shows the command: 'copying C:\Python27\lib\site-packages\py2exe\run\_w.exe -> C:\Python27\dist\GoogleChromeAutoLaunch.exe'. The second line shows the output: 'Adding python27.dll as resource to C:\Python27\dist\GoogleChromeAutoLaunch.exe'. The third line shows the command being executed: 'C:\Python27>C:\Python27\dist\GoogleChromeAutoLaunch.exe'. The fourth line shows the prompt: 'C:\Python27>\_'.

```
copying C:\Python27\lib\site-packages\py2exe\run_w.exe ->
C:\Python27\dist\GoogleChromeAutoLaunch.exe
Adding python27.dll as resource to C:\Python27\dist\Google
ChromeAutoLaunch.exe

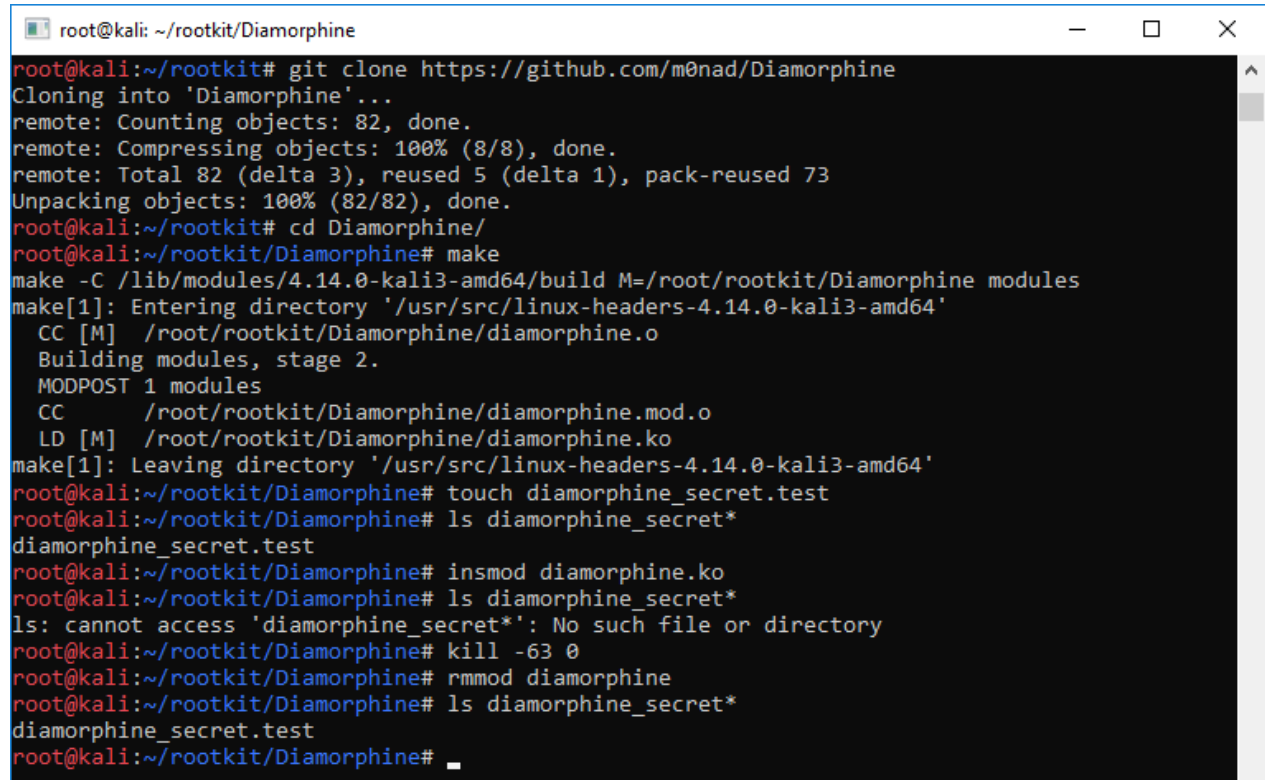
C:\Python27>C:\Python27\dist\GoogleChromeAutoLaunch.exe

C:\Python27>_
```

At this point, I decided not to waste any more time on this poorly documented (and apparently not updated) project but look for another rootkit. So I opened every single one of the ~ 75 rootkit pages linked on the list provided. Turns out, that the vast majority is either at least 3 years old, not or sparsely documented, just an unfinished experiment or all at once. Of the hand full of rootkits that do not match the above criteria, all target Linux, so most likely all the hours invested in the approach above are to be booked under “personal experience”...

# Diamorphine

Because of its rather detailed installation instructions, simple usage and a recent commit saying “make compile in kernels >4.12”, I decided to try the Diamorphine LKM rootkit next. That seems to be a good choice, the setup is very easy and the rootkit actually does something:

A terminal window titled 'root@kali: ~/rootkit/Diamorphine' showing the following commands and output:

```
root@kali:~/rootkit# git clone https://github.com/m0nad/Diamorphine
Cloning into 'Diamorphine'...
remote: Counting objects: 82, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 82 (delta 3), reused 5 (delta 1), pack-reused 73
Unpacking objects: 100% (82/82), done.
root@kali:~/rootkit# cd Diamorphine/
root@kali:~/rootkit/Diamorphine# make
make -C /lib/modules/4.14.0-kali3-amd64/build M=/root/rootkit/Diamorphine modules
make[1]: Entering directory '/usr/src/linux-headers-4.14.0-kali3-amd64'
  CC [M]  /root/rootkit/Diamorphine/diamorphine.o
Building modules, stage 2.
MODPOST 1 modules
  CC      /root/rootkit/Diamorphine/diamorphine.mod.o
  LD [M]  /root/rootkit/Diamorphine/diamorphine.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.14.0-kali3-amd64'
root@kali:~/rootkit/Diamorphine# touch diamorphine_secret.test
root@kali:~/rootkit/Diamorphine# ls diamorphine_secret*
diamorphine_secret.test
root@kali:~/rootkit/Diamorphine# insmod diamorphine.ko
root@kali:~/rootkit/Diamorphine# ls diamorphine_secret*
ls: cannot access 'diamorphine_secret*': No such file or directory
root@kali:~/rootkit/Diamorphine# kill -63 0
root@kali:~/rootkit/Diamorphine# rmmod diamorphine
root@kali:~/rootkit/Diamorphine# ls diamorphine_secret*
diamorphine_secret.test
root@kali:~/rootkit/Diamorphine# _
```

## Behavior

Diamorphine (among other things) hides files starting with a definable prefix (`diamorphine_secret` by default) and supposedly also hides itself (the `kill -63 0` un-hides it). The file hiding is demonstrated above.

## Memory dump on Kali Linux (4.14.0-kali3-amd64)

fmem doesn't compile:

```
/root/fmem/lkm.c:289:44: error: 'struct inode' has no member named 'i_mutex'; did you mean 'i_mode'?  
    mutex_lock(&file->f_path.dentry->d_inode->i_mutex);  
                                     ^~~~~~  
                                     i_mode
```

LiME on the other hand seems to work perfectly fine:

```
root@kali:~/rootkit# git clone https://github.com/504ensicsLabs/LiME
Cloning into 'LiME'...
remote: Counting objects: 192, done.
remote: Total 192 (delta 0), reused 0 (delta 0), pack-reused 192
Receiving objects: 100% (192/192), 1.57 MiB | 593.00 KiB/s, done.
Resolving deltas: 100% (86/86), done.
root@kali:~/rootkit# cd LiME/
root@kali:~/rootkit/LiME# cd src/
root@kali:~/rootkit/LiME/src# make
make -C /lib/modules/4.14.0-kali3-amd64/build M="/root/rootkit/LiME/src" modules
make[1]: Entering directory '/usr/src/linux-headers-4.14.0-kali3-amd64'
  CC [M] /root/rootkit/LiME/src/tcp.o
  CC [M] /root/rootkit/LiME/src/disk.o
  CC [M] /root/rootkit/LiME/src/main.o
  LD [M] /root/rootkit/LiME/src/lime.o
Building modules, stage 2.
MODPOST 1 modules
  CC /root/rootkit/LiME/src/lime.mod.o
  LD [M] /root/rootkit/LiME/src/lime.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.14.0-kali3-amd64'
strip --strip-unneeded lime.ko
mv lime.ko lime-4.14.0-kali3-amd64.ko
root@kali:~/rootkit/LiME/src# insmod ./lime-4.14.0-kali3-amd64.ko "path=$(pwd)/full-dump.lime format=lime dio=1"
root@kali:~/rootkit/LiME/src# ls -lah full-dump.lime
-r--r--r-- 1 root root 768M May  1 22:22 full-dump.lime
root@kali:~/rootkit/LiME/src#
```

To create the next dump, the kernel module has to be unloaded first:

```
rmmod lime
```

## Memory dump from VirtualBox

Another interesting memory dump source is VirtualBox itself.

The following script can be called with a virtual machine name and creates a `.raw` memory dump of the same name in the `pwd`:

```
#!/bin/bash Simple script for VirtualBox memory extraction Usage: vboxmemdump.sh <VM name>

"/mnt/c/Program Files/Oracle/VirtualBox/VBoxManage.exe" debugvm $1 dumpvmcore --filename=$1.elf
fixed number extraction
line=$(objdump -h kali1.elf | grep 'load1 ')
size=$(echo $line | cut -d' ' -f3)
echo 'size: 0x'$size
off=$(echo $line | cut -d' ' -f6)
echo 'offset: 0x'$off

head -c $((0x$size + 0x$off)) $1.elf | tail -c +$((0x$off + 1)) > $1.raw

rm $1.elf
```

This is mostly a copy of `vboxmemdump.sh` by Andrea Fortuna but fixed and adjusted to work with vBox on Windows and WSL.

# Memory dump analysis

## Profile

For `volatility` to be able to read the Kali Linux memory dumps, it needs a profile that matches in distribution, its version, kernel version and bitness. Due to the diversity of Linux systems, the profiles usually have to be created individually for each target.

For and on Kali Linux, this can be done by:

```
$ git clone https://github.com/volatilityfoundation/volatility/
$ cd volatility/tools/linux/
$ make
$ cd ../../..
$ zip profile-Kali-Linux-4.14.0-kali3-amd64.zip volatility/tools/linux/module.dwarf /boot/System.map-4.14.0
```

The resulting script can be copied to an otherwise empty directory on the analyzing machine and used with `--plugins=path/to/containing/dir --profile=Linuxprofile-Kali-Linux-4_14_0-kali3-amd64x64`.

## Analysis

Running

```
volatility --plugins=./vprofiles --profile=Linuxprofile-Kali-Linux-4_14_0-kali3-amd64x64 -f Kali-Linux-2.6.32-5-amd64
```

Fails with an uncaught Python exception. Searching for the error message leads to volatility issue #414 that is not yet fixed (or its fix not released). Both the Windows and Linux releases 2.6 suffer from this issue.

So currently I can not find out anything from the memory dumps I captured.



## Conclusion

Pretty much all of the necessary used for this memory analysis are at least partially broken. The analysis itself might work on Windows, but working rootkits (at least in the provided list) seem to be available only for Linux and volatility for memory analysis of Linux targets seems to be broken at the moment.

I think I spend enough time on this and actually solved a good number of problems, but there seems to be no end to stuff that is broken. So at this point, I'll just check against the requirements of the assignment:

“Choose a rootkit from the list given in the following link:  
<https://github.com/d30sa1/RootKits-List-Download>”

I did that. Tried one for Windows that doesn't compile and one for Linux that works.

“Capture the memory dump while the rootkit is running.”

I captured on Windows without rootkit and on Linux with rootkit using two different approaches.

“And then report an analysis of the rootkit using volatility or any other memory forensics tool.”

I did. Just didn't find anything because volatility is broken.

“In particular, report what the rootkit is doing” ...

I did that for the Diamorphine rootkit.

... “and whether you are able to capture its intended target and behavior.”

I did that as well. And the answer is “no”, at “capture its intended [...] behavior” means finding the rootkit through memory analysis.

“Deliverable: A report along with some snapshots showing.”

See above!

## Post scriptum

A student in class pointed out that he was able to compile win-rootkit on Windows 10. While I can confirm that, I am pretty sure that this rootkit is totally incomplete.

The rootkit includes the `mhook` library with which I actually worked a few years ago. Its basic principle is to inject a `.dll` file into every running process and/or have Windows load that `.dll` into every new process. The code in the `.dll` uses `mhook` itself to replace parts of build in functions with custom code. Thereby, one can for example replace the functions that do process enumeration and hide processes that way. A static analysis of a memory dump would not be affected by the hooking, so I think this would actually make for a quite interesting subject to study in a future class.

The problem is that, in what little code the rootkits repository contains, `mhook` is not actually being referenced (and the compilation succeeds without it). The rootkit also seems to re-implement `mhooks` basic functionality in `hook.cpp`, but doesn't use that either. The only thing the main function in the injected `.dll` does is open message boxes, but they are not created either. The `.dll` does not seem to be injected, it is not listed by *Process Hacker*, and it did not run any code to hide itself.

So bottom line, that rootkit does not work either.