Rootkit Pre-Class Writeup

1 Primary Tools Used

The following tools were primarily used to locate and identify various malware located on the given virtual machine:

- **GMER 1.0.15.15570.** GMER is a tool that scans for rootkits using common techniques, such SSDT or IRP hooking. It also has the ability to identify hidden files and, optionally, copy them to an unhidden location for further analysis.
- **Volatility 1.4-rc1.** Volatility is a memory forensics framework that scans a memory image for various objects, such as loaded modules, socket connections, etc. For simplicity, I just used the `.vmem` file from the given virtual machine image as input to Volatility. In a non-virtualized environment, forcing a crash dump or using a a tool such as `Win32dd.exe` from Moonsols or `LiveKd.exe` from Sysinternals could be used to create the memory image.
- `psservice.exe`. This tool from Sysinternals lists details about currently installed Windows system services, such as their names, descriptions and current status.
- `strings.exe`. This tool, also from Sysinternals, scans a binary for ASCII or Unicode strings of some given minimum length and displays them. It can be useful for quickly identifying the rootkit to which a binary file belongs.
- `listdlls.exe` Displays DLLs loaded for a particular process.
- `handle.exe` Displays open handles owned by a particular process.

Another technique that was easy and somewhat useful was simply looking at file modification times of `.sys` files located in `C:\WINDOWS\system32\drivers`. If a driver has a modification date significantly different than the rest of the drivers in that directory (e.g. `sysenter.sys`), then it might be worth looking into further. Of course, crafty rootkits can (and did) prevent files from being visible in Windows Explorer, so this certainly isn't an exhaustive approach. IDA Pro was also used as a last resort to analyze binary files extracted from the virtual machine's memory image that couldn't be identified via easier methods.

2 Identified Malicious Rootkits

2.1 Vanquish Autoloader v0.2.1

Perhaps the most easily visible rootkit is the Vanquish v0.2.1 rootkit. It is installed as a service and can be viewed via Windows's system services management console (`services.msc`) or by sifting through the output of Sysinternals's `psservice` command. Since Vanquish is installed as a system service, Windows will automatically execute `C:\WINDOWS\vanquish.exe` every time the system starts up. Convienently, the system also contains the folder `C:\vanquish-0.2.1`, which contains additional information on the rootkit's capabilities.

In particular, the rootkit attempts to inject `vanquish.dll` into certain processes on the system and modify their IAT to intercept calls to certain system functions related to the registry (e.g., `RegEnumKey*`), service listings (e.g., `EnumServicesStatus*`), and user authentication (e.g., `LogonUser*`). The rootkit will use this capability to capture usernames and passwords containing the magic string "vanquish", hide certain files and folders, and prevent deletion of files or folders that start with the string "`D:\MY`" by injected processes.If we look at `C:\vanquish.log`, we see that the rootkit claims to have successfully injected itself into `winlogon.exe`. We can confirm this by using Volatility to dump the loaded DLLs for `winlogon.exe` (PID 720) and analyzing the outputted DLLs:

```
>python vol.py dlllist -p 720 -f "rootkitclassvm.vmem"
winlogon.exe pid:    720
Command line : winlogon.exe
Service Pack 3

Base         Size         Path
0x01000000   0x081000     \??\C:\WINDOWS\system32\winlogon.exe
0x7c900000   0x0b2000     C:\WINDOWS\system32\ntdll.dll
   ...         ...        ...
0x01480000   0x00c000
0x014f0000   0x00c000
0x01520000   0x00c000
   ...         ...        ...

> python vol.py dlldump -p 720 --dump-dir=. -f "rootkitclassvm.vmem"
 ...
Dumping             , Process: winlogon.exe, Base:  1480000 output: module.720.1f2fbe0.1480000.dll
Dumping             , Process: winlogon.exe, Base:  14f0000 output: module.720.1f2fbe0.14f0000.dll
Dumping             , Process: winlogon.exe, Base:  1520000 output: module.720.1f2fbe0.1520000.dll
...

> strings.exe module.720.1f2fbe0.1480000.dll
...
Vanquish DLL v0.2.1
...
```

2.2 HackerDefender 1.0.0

Slightly less obvious than the Vanquish rootkit was the HackerDefender 1.0.0 (or 100?) rootkit. Rather than injecting itself as a DLL into certain processes, HackerDefender (or HxDef for short) is installed as both a kernel-mode driver (`hxdefdrv.sys`) and a hidden system service (`hxdef100.exe`). The rootkit will hide all files, folders and processes starting with the string "hxdef", so I used GMER to view the hidden files and copy their contents to a renamed, non-hidden directory. A list of hooked functions is also given in the hidden documentation directory. User-land code can communicate with the driver by issuing IOCTls to the `\\.\HxDefDriver` device. The rootkit also opens a mailslot for IPC with a semi-randomly generated name. For example, again using Volatility and the `files` plugin, we can see the HxDef service has open handles to `\hxdef-rk100s6825727F`.

HxDef is configured on this machine to run `taskmgr.exe` on system startup, but the configuration file (`hxdef100.ini`) could be modified to just as easily run netcat or something executable. When checking the running processes via Windows Task Manager, there appeared to be two `taskmgr.exe` processes executing: one running under the "Student" user account and another under the "SYSTEM" account, the latter of which was launched by HxDef.

The rootkit also includes a backdoor component that doesn't open up any ports of its own, but rather intercepts *all* incoming network traffic and looks for a "magic" password-based packet of a fixed length. Thus, if *any* service has an open port (e.g., port 135 on this VM), an attacker can use the `bdcli100.exe`

executable to connect remotely to the machine with the default password `hxdef-rulez`. For example, the following command will pop open a shell on the VM:

```
bdcli100.exe 127.0.0.1 135 hxdef-rulez
```

2.3 FUTo

GMER also located an additional hidden file, `C:\WINDOWS\system32\drivers\fu.exe`. Copying the hidden `fu.exe` to a new folder, renaming it and running the `strings.exe` tool from Sysinternals on the executable yields a few interesting entries in the resulting output. In particular, we find a reference to the `.pdb` file generated by Visual C++ when compiling the binary:

```
c:\Documents and Settings\user\Desktop\FUTo\_enhanced\FUTo\fu\Debug\fu.pdb
```

From this string and the hidden `fu.exe` binary, we can reasonably assume the system is infected with the FUTo rootkit, which is an updated version of the FU rootkit. Perhaps more usefully, we can even find the command-line options for the program without disassembly or attempting to run the executable directly. The available options are:

```
Usage: fu
[-ph]  \#PID      to hide the process with \#PID
[-phng]  \#PID    to hide the process with \#PID. The process must not have a GUI
[-phd] DRIVER_NAME to hide the named driver
[-pas] \#PID      to set the AUTH\_ID to SYSTEM on process \#PID
[-prl] to list the available privileges
[-prs] \#PID \#privilege_name to set privileges on process \#PID
[-pss] \#PID \#account_name to add \#account\_name SID to process \#PID token
```

This rootkit also has a kernel-mode driver (`msdirectx.sys`, which can receive IOCTLs from `fu.exe` to the `\Device\msdirectx` device. The kernel-mode driver also contains the following string, which further suggests the `msdirectx.sys` drive is from the FUTo rootkit:

```
c:\futo_enhanced\futo\exe\i386\msdirectx.pdb
```

The driver file is apparently named such in an attempt to confuse it with Microsoft's DirectX graphics API. The IOCTL sent to the KMD depends on the command-line switch given to `fu.exe`, and each feature is implemented in the driver using Direct Kernel Object Manipulation (DKOM).

2.4 Shadow Walker

The system has another suspcious driver installed related to the FU rootkit called `mmpc.sys`, located in the `C:\Windows\System32\Drivers`. Running `strings.exe` on the binary again yields another interesting reference to a `.pdb` file:

```
c:\shadowwalker_corey\BINi386\mmpc.pdb
```

Shadow Walker is a proof-of-concept feature added to the FU rootkit above that attempts to control an application's view of the system memory. It does so by hooking the 0x0E (page fault) interrupt in the IDT. The rootkit installs a new page fault handler and filter read, write and execute access to hooked pages.

We can find confirm that `mmpc.sys` has hooked the 0x0E interrupt again using Volatility to dump the IDT from the memory image:

```
...     ...
13      ntoskrnl.exe!0x8053fd90
14      0xf8bf2816
15      ntoskrnl.exe!0x805407c8
...     ...
```

The address `0xf8bf2816` for interrupt 0x0E (14) corresponds to the `mmpc.sys` driver:

```
Start        Size Service key          Name
0xf8bf2000   6144 'mmpc'               'mmpc'       '\\Driver\\mmpc'
```

The driver also opens a `\Device\mmHook` device, which can receive IOCTLs from user-mode applications via `DeviceIOControl`.

2.5 Other Kernel-mode Drivers

The system also has the following interesting kernel-mode drivers installed (each of the drivers is located in the `C:\Windows\System32\drivers` directory):

- **Ctr12Cap.sys** This kernel mode driver is apparently named to attempt to trick a casual observer into confusing it with `Ctrl2cap.sys`, which is a legitimate driver from Sysinternals that intercepts caps-lock characters and converts them to control characters for people who suck at typing. This driver instead hooks the SSDT to intercept calls to the system function `ZwQueryDirectoryFile`, which will let the driver effectively hide directories or files matching a certain pattern. In this case, the magic string appears to be "_cool_", based on an analysis of the output from `strings.exe`. I then created a file called `_cool_.txt` and verified that it was not visible in Windows Explorer. The file was still visible in GMER, however.

- **BASIC.sys** The most interesting string in this driver's binary file is:

  ```
  z:\binarytransfer\rootkits\___installed\basic_callgate\callgate_driver\objfre_wxp_x86\i386\BASIC.pdb
  ```

  The phrase "callgate" in the `.pdb` path suggests that this particular driver adds a new call-gate descriptor to the Windows global descriptor table (GDT). Doing so would allow code running at a less privileged level to call functions at a higher privilege level (e.g., ring0).

- **sysenter.sys** This driver hooks the `SYSENTER` instruction, which switches from user-mode to kernel-mode using three machine-specific registers (MSRs). The `SYSENTER` hook in this driver doesn't appear to do anything malicious and simply calls the original `KiFastCallEntry` function from the hook.

- **BreakOnThruToTheOtherSide.sys** This Doors-referencing driver doesn't appear to do much except print some debug statements regarding the current value of several registers. Though, it does seem to be from Xeno's Intermediate x86 class:

  ```
  c:\intermediatex86code\breakonthrutotheotherside\i386\BreakOnThruToTheOtherSide.pdb
  ```

  The `BreakOnThruToTheOtherSide` service is not visible within `services.msc`, but one can still manually query, start, stop and remove it from the command line. For example, running `sc stop BreakOnThruToTheOtherSide` will display "Goodbye Kernel! I left things the way I found them :)" in DebugView.

- **azdow88m.sys** The driver `azdow88m.sys` is also currently installed on the system. It creates the following two device paths:

  ```
  \Device\Scsi\azdow88m1
  \Device\Scsi\azdow88m1Port2Path0Target0Lun0
  ```

  Unfortunately, it's not clear to me what this driver actually does. The driver appears to be hidden, since it doesn't show up in Windows Explorer. Instead, I extracted the binary from the memory image using Volatility and opened the resulting file in IDA Pro, but didn't have time to do much analysis on it. Perhaps it's related to the Daemon Tools software running in the system tray, due to the presence of `dtpro.pdb` in the binary?

3 Other "Rootkits"

While not necessarily malicious, the system also contained other software described in this section that utilizes techniques not unlike the rootkits from the previous section.

3.1 Trusteer Rapport

Trusteer Rapport is an attempt at an anti-phishing solution from Trusteer, Ltd., that is starting to be recommended by more banks to their customers. The software installs a`RapportMgmtService` system service and associated kernel-mode drivers `RapportPG.sys`, `RapportCerberus_23645`, and `RapportKELL`. The kernel-mode drivers use SSDT hooking to intercept calls to several system functions, such as `ZwCreateFile`, `ZwDeleteKey`, etc.

3.2 ZoneAlarm

ZoneAlarm is a common software firewall application for Windows. The software has system services for a "ZoneAlarm Toolbar" (`IswSvc`) and "True Vector Internet monitor" (`vsmon`). It installs the `vsdatant.sys` driver that uses SSDT hooking to intercept many other system calls, such as `ZwCreatePort`, `ZwConnnectPort`, `ZwSecureConnectPort`, etc. It also uses IAT hooking to intercept calls to many functions from `NDIS.sys`, such as `NdisOpenAdapter`, `NdisCloseAdapter`, etc. Based on GMER's output, ZoneAlarm also appears to use inline hooking to detour calls to many user-land APIs, such as `ws2_32.dll!WSARecv`, `kernel32.dll!ReadFile`, among many others.