Malware Detected in Preliminary Assignment for Rootkits Class

Vanquish

What about the system was maliciously changed?

No visible changes were evident in the system other than the presence of installation directory vanquish-0.2.1 and log file vanquish.log in C:\ and vanquish.exe and vanquish.dll in C:\WINDOWS.

According to readme-style notes in the vanquish-0.2.1 folder, Vanquish is a “DLL-Injection based rootkit that hides files, folders, registry entries and logs passwords.”

How was the change caused?

vanquish.log states:

```
***Application: C:\WINDOWS\System32\svchost.exe
***Time: 7:33:27 PM
***Date: 3/13/2011
0x00000005: Access is denied.
Vanquish - DLL injection failed:
WriteProcessMemory

***Application: C:\WINDOWS\System32\svchost.exe
***Time: 7:33:27 PM
***Date: 3/13/2011
Failed to inject VANQUISH!
hProcess: 0x000017a0
hThread: 0x00001728
dwProcessId: 0x000007ac
dwThreadId: 0x000006c8

***Application: \??\C:\WINDOWS\system32\winlogon.exe
***Time: 8:16:13 PM
***Date: 3/13/2011
```

From this I hypothesize that Vanquish injected itself into winlogon.exe, so when a user logs onto the computer, vanquish hooks into the interrupt table, executes itself, and then passes control on to the authentic winlogon.exe process. According to the readme, the system passwords should be listed in the log files above; Vanquish is supposed hook the msgina dll function WlxLoggedOutSAS to obtain the password. However, the passwords are not present perhaps because Vanquish is not working correctly.

According to the readme, Vanquish:

> hooks API calls and processes them. If needed it calls the REAL Windows API for additional processing. The injection method is described in . . . "Executing arbitrary code in a chosen process(or advanced dll injection)" on [www.rootkit.com]. The effective API replace is done by overwriting the first 5 bytes of the API prolog with a 32bit offset jmp instruction to our NEW Vanquish API wich resides in the injected

DLL(vanquish.dll); if needed Vanquish will rebuild the first 5 bytes of the REAL API and call it, then write the jmp once again. Vanquish DLL employs a number of thread-synchronization tehniques to be safe. The only problem with this method is that lengthy functions could be called again while being un-hooked for original api forwarding. Testing showed that in about 4% - 5% of the cases (on a nonblocking function) calls would not go to the NEW Vanquish API.

The Vanquish DLL is initially injected into all windowed exe images by Vanquish Autoloader. The injection code is also present in Vanquish DLL and is activated when a call to CreateProcess(AsUser)A/W is intercepted.

a) Vanquish Autoloader
When run, vanquish.exe searches for these command-line arguments:

-install	Add vanquish.exe to Services; inject DLL into running apps.
-remove	Remove vanquish from Services; needs restart to remove DLL.

It is not recommended to use directly. Use setup.cmd script instead. When no recognizable command-line argument is found, vanquish.exe defaults to injecting the DLL into running apps and exit. Thus, you will never see Vanquish Autoloader as 'STARTED' in the Services MMC. This is so that no one will spot it's *[sic]* process in Task Manager.

b) Vanquish DLL
Vanquish DLL is internally divided into SIX submodules:

1)DllUtils
Inject Vanquish DLL into new processes. (CreateProcess(AsUser)A/W)
Make sure nobody will unload Vanquish DLL. (FreeLibrary)

Utilites related to dll functionality and propagation.

2)HideFiles
Hide files/folders containing the magic string "vanquish" (FindFirstFileExW, FindNextFileW)

A hidden file/folder won't get reported by windowze as occupying space and cannot be found with "Search For Files or Folders..." or similar, and a folder containing hidden files/folders cannot be erased.

3)HideReg
Hide registry entries containing the same magic string.
(RegCloseKey, RegEnumKeyA/W, RegEnumKeyExA/W,
RegEnumValueA/W, RegQueryMultipleValuesA/W)

Uses an advanced cached system to keep track of enumerated keys/values and hide the ones that need to be hidden, but keeping the high-performance of registry.

4)HideServices
Hide service entries containing the magic string in their name.
(EnumServicesStatusA/W)

Uses the same logic as in HideReg but without the complicated caching system.

5)PwdLog
Logs username, passwords and domain.
(LogonUserA/W, WlxLoggedOutSAS)

First it intercepts calls to LogonUserW (probably 'runas' commands, switch user and similar). To get the logon prompt password we hook the msgina dll function WlxLoggedOutSAS. The passwords are present in the logfile.

Other features:
Error logging - everything (including passwords) will be shown in a file called 'vanquish.log' in the root of c:

How can you remove the changes?

Vanquish Autoloader not present for the preferred method of removal, so I switche to cmd and ran “C:\vanquish-0.2.1>setup do remove.” According to the resulting screen output, this successfully removed vanquish from the system.

Tools and techniques used

Vanquish was discovered by visual inspection of the C:\ drive.

asr_pfu and fu (suspect these to be the same)

What about the system was maliciously changed?

asr_pfu.exe found in C:\WINDOWS\system32 on initial load (to snapshot). On reboot, fu.exe is additionally found in C:\WINDOWS\system32. Internet sources[1] indicate asr_pfu.exe is a rootkit with many aliases. Based on the commonality of the names, asr_pfu.exe may be the launching program for fu.exe, which as previously stated, launches and installs on reboot.

Before reboot, a Windows search for fu reveals FU.EXE-1D986D16.pf in an invisible folder, C:\WINDOWS\Prefetch. This folder is invisible even with the Tools->Folder Options->View->Show hidden files and folders option activated. After reboot and the presence of fu.exe in C:\WINDOWS\system32, the file FFUNZIP.EXE-20E76EF2.pf similarly appears in this folder. The former file’s “Date Modified” property field changes at or near reboot, so it is likely that, given that both files are in the same hidden folder and the modified timestamp, that FU.EXE-1D986D16.pf generates FFUNZIP.EXE-20E76EF2.pf.

According to http://www.prevx.com/filenames/X24965474357862442-X1/FU.EXE.html:

FU.EXE has been seen to perform the following behavior:

- The Process is packed and/or encrypted using a software packing process
- Enables a COM Object/Server on the Local Machine
- Writes to another Process's Virtual Memory (Process Hijacking)
- Executes a Process
- Injects code into other processes

---

[1] http://www.threatexpert.com/files/asr_pfu.exe.html

- This process creates other processes on disk
- This Process Deletes Other Processes From Disk
- Can make outbound communication to other computers, IM chat rooms and other services using IRC protocols
- Adds products to the system registry
- Found on infected systems and resists interrogation by security products

FU.EXE has been the subject of the following behavior:

- Created as a process on disk
- Executed as a Process
- Enabled as a COM Object/Server on the Local Machine
- Has code inserted into its Virtual Memory space by other programs
- Terminated as a Process
- Changes to the file command map within the registry
- Deleted as a process from disk

How was the change caused?

When using PEview.exe to view memory locations 00044C20 to 000452D0, fu.exe displays the capability for a number of suspect actions:

- Use the process ID of another process to masquerade as that process
- Hide a named driver
- Set the AUTH_ID to System for a particular process (identified by PID)
- Use of
    - lock memory privilege
    - increase quota privilege
    - security privilege
    - take ownership privilege
    - system time privilege

C:\WINDOWS has two suspicious files, fat.bat and fat_init.bat, the first of which seems to use a process named fu to run or manipulate (perhaps hook?) msdirectx.sys and mmpc.sys.

- fat.bat

  ```
  @ECHO OFF
  cd C:\WINDOWS\system32\drivers
  fu -ph 4
  ..\InstDriver.exe -install mmpc mmpc.sys
  ..\InstDriver.exe -start mmpc
  fu -phd msdirectx.sys
  fu -phd mmpc.sys
  sc delete mmpc
  exit
  ```

- fat_init.bat

  ```
  start "" /MIN /HIGH "C:\WINDOWS\fat.bat" > NUL
  ```

I note that fu is an .exe rather than .sys and hypothesize that this is the case because .exe files don't have relocation information and therefore can't be moved around in memory. Since ASLR cannot be used with .exe, it can know precisely what memory it will occupy and calculate offsets to other Windows programs of choice.

How can you remove the changes?

I hypothesize that one could hook the interrupt table before fu.exe does. If fu.exe then subsequently attempts to hook the table, the original program can prevent this.

Tools and techniques used

Visual inspection of the C:\WINDOWS/system32 and C:\WINDOWS/system32\drivers folders located the anomalies. Once located, PEview.exe was used to peruse the file structure.

Additional, unexplained observations

Initial observations

- Notepad open (minimized)
    - When maximized it had nothing
    - Minimized
    - After it was maximized again it had two lines of text, saying “red rum”
    - After it was closed and reopened, notepad was blank

Possible corruption/hooking of ZoneAlarm

- ZoneAlarm states that there is a “New Network Found,” asking for a new connection between it and the Internet or another computer
    - DebugView says:
        - [2176] <<UI>>
        - [2176] Update is active
        - [2176] <blank line>
        - …
    - When links are clicked within ZoneAlarm, the following appears in DebugView:
        - [2176[<<UI>>
        - [2176] Show Detail dialog
        - [2176] <blank line>
- Delay when accessing ZoneAlarm links
- Process 2176 is zlclient.exe (part of ZoneAlarm) and is too big to read through using PEview

Conclusion: Given the DebugView output, the sensitivity of the ZoneAlarm application (monitors internet connections), and the delay when accessing ZoneAlarm links, it is likely that ZoneAlarm has been hooked or otherwise compromised by malware.

Attempted tool use stymied

WinDbg

I could not get WinDbg to connect, or break into, the guest OS. This difficulty may be because malware is interfering with the interrupt table. According to a posting in OSROnline,[2] WinDbg break-in polling happens as a part of time interrupt processing.

BreakOnThruToTheOtherSide

I tried to find a program that would enable me to view the interrupt program. One is staring me in the face, and the driver for it is already loaded on the VM: BreakOnThruToTheOtherSide.sys. However, the driver does not seem to execute the method void MyInterruptHandler(), which is called by

myHandlerPtr = (unsigned int)MyInterruptHandler;

in the main DriverEntry function. When I put DbgPrint statements on either side of the line above, I get those printed out to the kernel view (DebugView) but do not see any of the DbgPrint statements contained within MyInterruptHandler().

I have tried taking the code of MyInterruptHandler out of the function and putting in into the DriverEntry function directly; however, when I do so, the computer crashes. Memory dump analysis in WinDbg states that in crashes at the iretd statement.

---

[2] http://www.osronline.com/showthread.cfm?link=148853