Threat Spotlight: Group 72, Opening the ZxShell

This post was authored by Andrea Allievi, Douglas Goddard, Shaun Hurley, and Alain Zidouemba.

Recently, there was a blog post on the takedown of a botnet used by threat actor group known as Group 72 and their involvement in Operation SMN. This group is sophisticated, well funded, and exclusively targets high profile organizations with high value intellectual property in the manufacturing, industrial, aerospace, defense, and media sector. The primary attack vectors are watering-hole, spear phishing, and other web-based attacks.

Frequently, a remote administration tool (RAT) is used to maintain persistence within a victim's organization. These tools are used to further compromise the organization by attacking other hosts inside the targets network.

ZxShell (aka Sensocode) is a Remote Administration Tool (RAT) used by Group 72 to conduct cyber-espionage operations. Once the RAT is installed on the host it will be used to administer the client, exfiltrate data, or leverage the client as a pivot to attack an organization's internal infrastructure. Here is a short list of the types of tools included with ZxShell:

- Keylogger (used to capture passwords and other interesting data)
- Command line shell for remote administration
- Remote desktop
- Various network attack tools used to fingerprint and compromise other hosts on the network
- Local user account creation tools

For a complete list of tools please see the MainConnectionIo section.

The following paper is a technical analysis on the functionality of ZxShell. The analysts involved were able to identify command and control (C2) servers, dropper and installation methods, means of persistence, and identify the attack tools that are core to the RAT's purpose. In addition, the researchers used their analysis to provide detection coverage for Snort, Fireamp, and ClamAV.

Table of Contents

Background

ZxShell has been around since 2004. There are a lot of versions available in the underground market. We have analyzed the most common version of ZxShell, version 3.10. There are newer versions, up to version 3.39 as of October 2014.

Distribution and Delivery

An individual who goes by the name LZX in some online forums is believed to be the original author of ZxShell. Since ZxShell has been around since at least 2004, numerous people have purchased or obtained the tools necessary to set up ZxShell command and control servers (C&C) and generate the malware that is placed on the victim's network. ZxShell has been observed to be distributed through phishing attacks, dropped by exploits that leverage vulnerabilities such as CVE-2011-2462, CVE-2013-3163, and CVE-2014-0322.

Analysis of the Main ZxShell Module

To illustrate the functionality of main ZxShell module, Let's take a look at the following sample:

- MD5: e3878d541d17b156b7ca447eeb49d96a
- SHA256: 1eda7e556181e46ba6e36f1a6bfe18ff5566f9d5e51c53b41d08f9459342e26c

It exports the following functions, which are examined in greater detail below:

- DllMain
- Install
- UnInstall
- ServiceMain
- ShellMain
- ShellMainThread
- zxFunction001
- zxFunction002

DllMain

DllMain performs the initialization of ZxShell. It allocates a buffer of 0x2800 bytes and copies the code for the ZxGetLibAndProcAddr function. To copy memory, the memcpy function is invoked. It is not directly used from msvcrt.dll but is instead copied to another memory chunk before being called. Finally, the trojan Import Address Table (IAT) is resolved and the file path of the process that hosts the dll is resolved and saved in a global variable.

Install

ZxShell.dll is injected in a shared SVCHOST process. The Svchost group registry key HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost is opened and the netsvc group value data is queried to generate a name for the service.

Before the malware can be installed a unique name must to be generated for the service. The malware accomplishes this through querying the netsvc group value data located in the svchost group registry key which is HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost.

At startup, Svchost.exe checks the services part of the registry and constructs a list of services to load. Each Svchost session can contain multiple shared services that are organized in groups. Therefore, separate services can run, depending on how and where Svchost.exe is started.

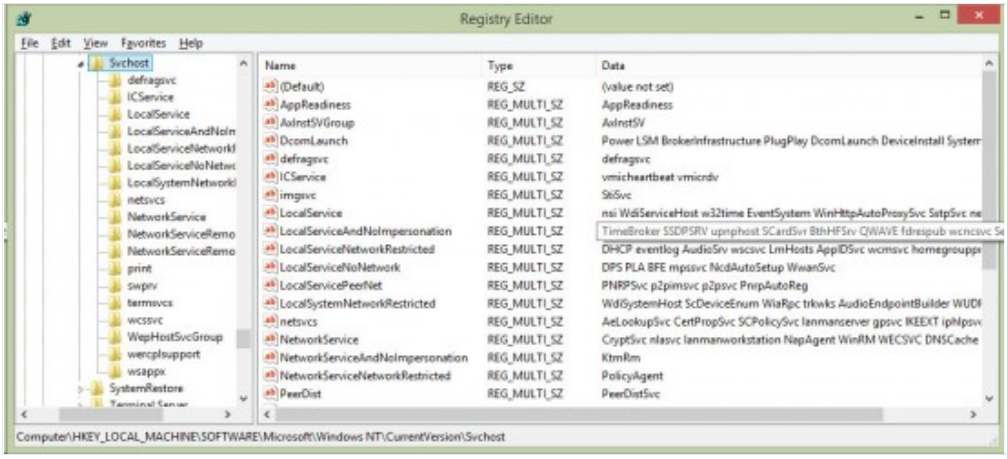


Image 1. Svchost Groups registry key

Svchost.exe groups are identified in the above registry key. Each value under this key represents a separate Svchost group and appears as a separate instance when you are viewing active processes. Each value is a REG_MULTI_SZ value and contains the services that run under that Svchost group. Each Svchost group can contain one or more service names that are extracted from the following registry key, whose Parameters key contains a ServiceDLL value:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Service
```

On a Windows machine, the netsvc group contains names of both existing and non-existing services. ZxShell exploits this fact by cycling between each of the names, verifying the existence of the real service. The service's existence is verified with the ServiceExists function, which attempts to open the relative registry sub-key in HKLM\SYSTEM\CurrentControlSet\Services. The first service name that is not installed on the system becomes the ZxShell service name.

A new service is then created using the service parser function ProcessScCommand. ZxShell implemented its own version of the Windows SC command. There are minor differences between the ZxShell implementation of this command and the original Windows one, but they are irrelevant for the purpose of the analysis The command used to install the service is:

```
sc create <service name> <service name> "%SystemRoot%\System32\svchost.exe -k netsvcs"
```

where <service name> is the chosen infected service name.

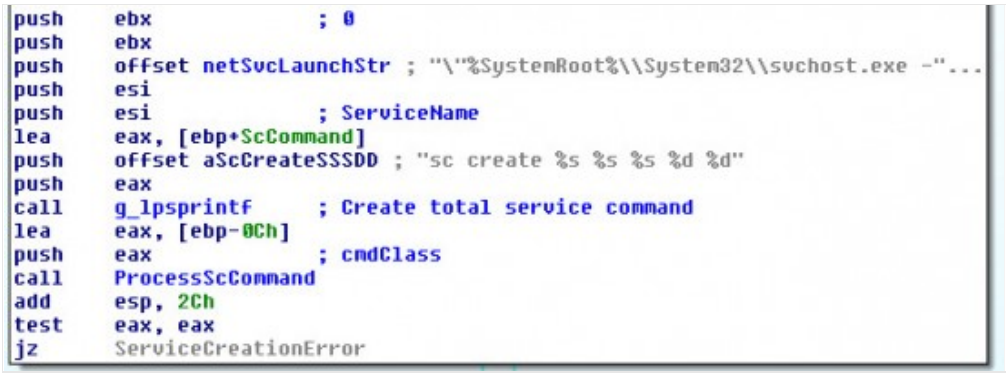


Image 2. “SC” command used to create the target service,and parsed by “ProcessScCommand” routine

The installed service registry key is opened and the 2 values under its *Parameter* subkey are created. These 2 values, *ServiceDll* and ServiceDllUnloadOnStop are needed for services that run in a shared process.

Before the service is started ChangeServiceConfig is called to modify the service type to shared and interactive. If the service fails to start then a random service name formatted as netsvc_xxxxxxxx, where xxxxxxxx represent an 8-digit random hex value, is added to the netsvc group and the entire function is repeated.

ServiceMain

This function is the entry point of the service. It registers the service using the RegisterServiceCtrlHandler Windows API function. The ZxShell service handler routine is only a stub: it responds to each service request code, doing nothing, and finally exits. It sets the service status to RUNNING and finally calls the ShellMain function of ZxShell.

ShellMain

The ShellMain function is a stub that relocates the DLL to another buffer and spawns a thread that starts from *ShellMainThreadInt* at offset +0xC0CD. The *ShellMainThreadInt* function gets the HeapDestroy Windows API address and replaces the first 3 bytes with the RET 4 opcode. Subsequently, it calls the *FreeLibrary* function to free its own DLL buffer located at its original address. Because of this, the allocated heaps will not be freed. It re-copies the DLL from the new buffer to the original one using the memcpy function. Finally, it spawns the main thread that starts at the original location of *ShellMainThread* procedure, and terminates. At this point, the ZxShell library is no longer linked in the module list of the host process. This is important because if any system tool tries to open the host process it will never display the ZxShell DLL.

ShellMainThread

This thread implements the main code, responsible for the entire botnet DLL. First, it checks if the DLL is executed as a service. If so, it spawns the service watchdog thread. The watchdog thread checks the registry path of the ZxShell service every 2 seconds, to verify that it hasn’t been modified. If a user or an application modifies the ZxShell service registry key, the code restores the original infected service key and values.

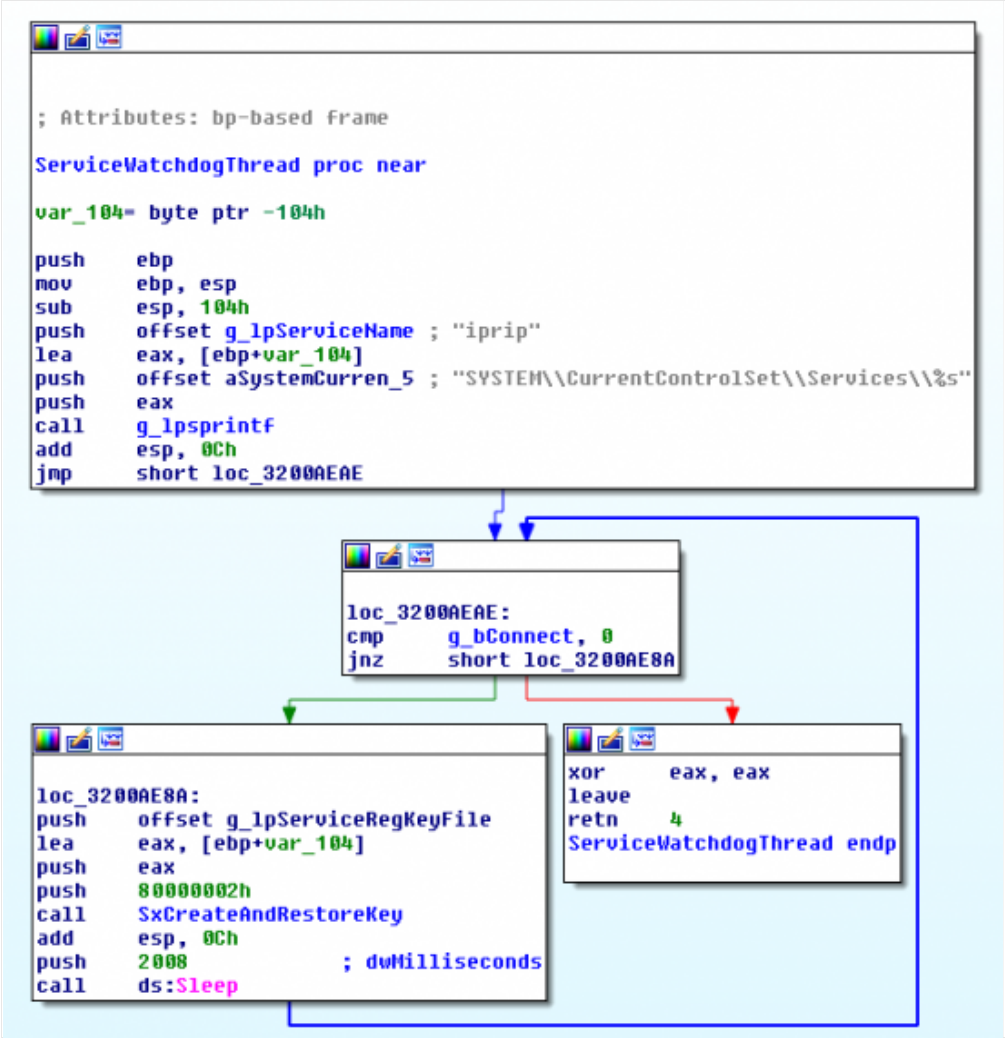


Image 3. The watchdog thread of ZxShell service

The buffer containing the ZxShell Dll in the new location is freed using the VirtualFree API function. A handle to the DLL file is taken in order to make its deletion more difficult. The ZxShell mutex is created named *@_ZXSHELL_@*.

ZxShell plugins are parsed and loaded with the *AnalyseAndLoadPlugins* function. The plugin registry key *HKLM\SYSTEM\CurrentControlSet\Control\zxplug* is opened and each value is queried. The registry value contains the plugin file name. The target file is loaded using the *LoadLibrary* API function, and the address of the exported function *zxMain* is obtained with *GetProcAddress*.

If the target filename is incorrect or invalid the plugin file is deleted and the registry value is erased. That is performed by the function *DeleteAndLogPlugin*. Otherwise, the plugin is added to an internal list. Here is the data structure used to keep track of the plugins:

```
typedef struct _ZX_PLUGINS_STRUCT {
    LPSTR lpStrRegKey;         // + 0x00 - ZxShell Plugins registry key string
// (like 'SYSTEM\CurrentControlSet\Control\zxplug')
    DWORD dwUnknown2;         // + 0x04 - Unknown DWORD value
    LPVOID lp138hBuff;         // + 0x08 - Plugins list
    DWORD dwZero;             // + 0x0C - Always zero
    HANDLE hReg;             // + 0x10 - Handle to plugin registry key
```

```
} ZX_PLUGINS_STRUCT, *PZX_PLUGINS_STRUCT;
```

The thread *KeyloggerThread* is spawned and is responsible for doing keylogging on the target workstation. We will take a look at the keylogger later on. Finally the main network communication function *GetIpListAndConnect* is called.

GetIpListAndConnect

This function is at the core of the RAT's network communication. It starts by initializing a random number generator and reading 100 bytes inside the ZxShell Dll at a hardcoded location. These bytes are XOR encrypted with the byte-key 0x85 and contains a list of remote hosts where to connect. The data is decrypted, the remote host list is parsed and verified using the *BuildTargetIpListStruct* function. There are 3 types of lists recognized by ZxShell: plain ip addresses, HTTP and FTP addresses.
If the list does not contain any item, or if the verification has failed, the ZxShell sample tries to connect to a hardcoded host with the goal of retrieving a new updated list.

Otherwise, ZxShell tries to connect to the first item of the list. If ZxShell successfully connects to the remote host, the function *DoHandshake* is called. This function implements the initial handshake which consists of exchanging 16 bytes, 0x00001985 and 0x00000425, with the server. The function *GetLocalPcDescrStr* is used to compose a large string that contains system information of the target workstation. That information is the following:

- local hostname
- organization
- owner
- operating system details
- CPU speed
- total physical memory

The string is sent to the remote host and the response is checked to see if the first byte of the response is 0xF4, an arbitrary byte. If it is, the botnet connection I/O procedure is called through the *MainConnectionIo* function.

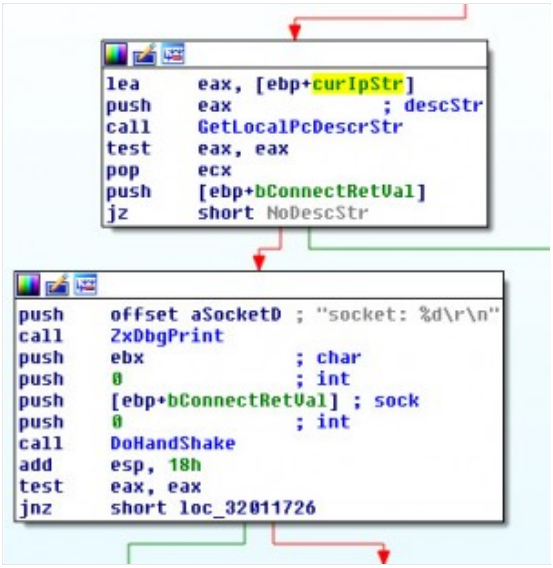


Image 4. The GetLocalPcDescrStr and DoHandshake functions called beforestarting the command processing

Otherwise, the ZxShell code closes the socket used and sleeps for 30 seconds. It will then retry the connection with the next

remote host, if there is one.

It is noteworthy that this function includes the code to set the ZxShell node as a server: if one of the hardcoded boolean value is set to 1, a listening socket is created. The code waits for an incoming connection. When the connection is established a new thread is spawned that starts with the *MainConnectionIo* function.

MainConnectionIo

The *MainConnectionIo* function checks if the Windows Firewall is enabled, sets the Tcp Keep Alive value and Non-blocking mode connection options and receives data from the remote host through the *ReceiveCommandData* function. If the communication fails, ZxShell disables the firewall by modifying the registry key:

```
HKLM\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile
```

Then the connection is retried. The received command is then processed by the ZxShell function with the *ProcessCommand* function.

The command processing function starts by substituting the main module name and path in the hosting process PEB, with the one of the default internet browser. The path of the main browser of the workstation is obtained by reading the registry value:

```
HKLM\SOFTWARE\Classes\HTTP\shell\open\command
```

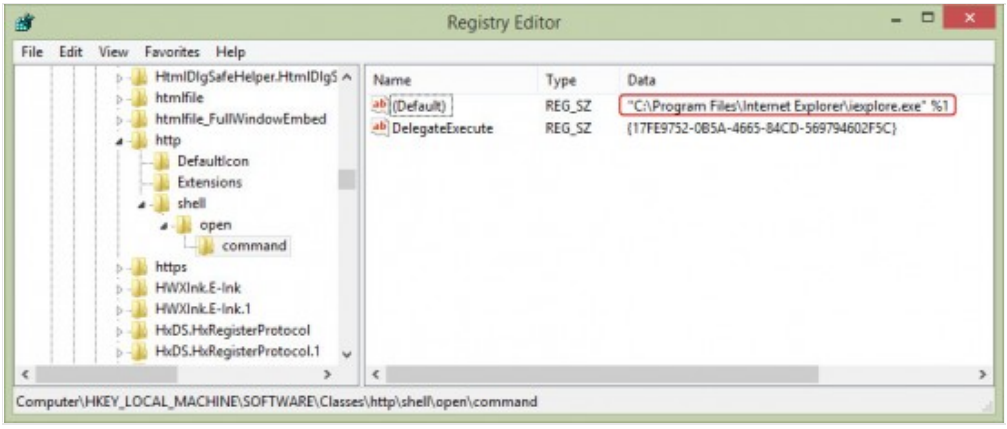


Image 5. Our test workstation use Windows Internet Explorer as default browser

This trick renders identification by firewall more cumbersome. A host firewall will recognize the outgoing connection as originated by the browser instead of the ZxShell service host process. The browser process always performs outgoing connections and the firewall shouldn't block them.

The command processing is straightforward. Here is the list of common commands:

COMMAND	MEANING
Help / ?	Get help
Exit / Quit	Exit and shut down the botnet client
SysInfo	Get target System information
SYNFlood	Perform a SYN attack on a host
Ps	Process service Unix command implementation
CleanEvent	Clear System Event log

FindPass	Find login account password
FileTime	Get time information about a file
FindDialPass	List all the dial-up accounts and passwords
User	Account Management System
TransFile	Transfer file in or from remote host
Execute	Run a program in the remote host
SC	Service control command, implemented as the Windows one
CA	Clone user account
RunAs	Create new process as another User or Process context.
TermSvc	Terminal service configuration (working on Win Xp/2003)
GetCMD	Remote Shell
Shutdown	Logout, shutdown or restart the target system
ZXARPS	Spoofing, redirection, packet capture
ZXNC	Run ZXNC v1.1 -- a simple telnet client
ZXHttpProxy	Run a HTTP proxy server on the workstation
ZXSockProxy	Run a Sock 4 & 5 Proxy server
ZXHttpServer	Run a custom HTTP server
PortScan	Run TCP Port MultiScanner v1.0
KeyLog	Capture or record the remote computer's keystrokes. The implementation is a userland keylogger that polls the keymap with each keystroke.
LoadDll	Load a DLL into the specified process
End	Terminate ZxShell DLL
Uninstall	Uninstall and terminate ZxShell bot DLL
ShareShell	Share a shell to other
CloseFW	Switch off Windows Firewall
FileMG	File Manager
winvnc	Remote Desktop
rPortMap	Port Forwarding
capsrv	Video Device Spying
zxplug	Add and load a ZxShell custom plugin

This set of functionality allows the operator complete control of a system. Being able to transfer and execute files on the infected system means the attacker can run any code they please. Further, the keylogging and remote desktop functionality allows the operator to spy on the infected machine, observing all keystrokes and viewing all user actions.

Uninstall

Unloads ZxShell and deletes all of the active components. This simply deletes the ZxShell service key from the Windows registry (using *SHDeleteKey* Api) and all of the subkeys. Finally, it marks ZxShell main Dll for deletion with the *MoveFileEx* Windows API.

ZxFunction001

This function is the supporting functionality for WinVNC. To allow the VNC session to connect, the current network socket WSAProtcol_Info structure is written to a named pipe prior to calling zxFunction001. Once the named pipe has been created, CreateProcessAsUserA is called with the following as the CommandLine parameter :

```
<systemroot>\\rundll32.exe <zxshell dll name>,zxFunction001 <name of NamedPipe>
```

zxFunction001 modifies the current process memory, uses data contained in the named pipe to create a socket, and then executes the code that sends the remote desktop session to the server controller.

ZxFunction002

This function will either bind the calling process to a port or has the calling process connect to a remote host. The function is called in the following manner:

```
<systemroot>\\rundll32.exe <zxshell dll name>,zxFunction002 <name of NamedPipe>
```

The functionality (connect or bind) depends on the data contained within the named pipe. Unlike zxFunction001, this is not used by any of the RAT commands in the zxshell.dll.

Kernel Device Driver LoveUSD

Apart from user-mode ZxShell droppers mentioned earlier, there is a file (SHA256: 1e200d0d3de360d9c32e30d4c98f07e100f6260a86a817943a8fb06995c15335) that installs a kernel device driver called *loveusd.sys*. The architecture of this dropper is different from the others: it starts extracting the main driver from itself. It adds the SeLoadDriver privilege to its access token and proceeds to install the driver as a fake disk filter driver. ZxShell opens the registry key that describes the disk class drivers:

```
SYSTEM\CurrentControlSet\Control\Class\{4D36E967-E325-11CE-BFC1-08002BE10318}
```

It then adds the “Loveusd.sys” extracted driver name to the upper filter list. In our analysed sample the “Loveusd.sys” driver is installed with the name “USBHPMS”. Finally the driver is started using the ZwLoadDriver native API.

The ZxShell driver starts by acquiring some kernel information and then hooking “ObReferenceObjectByHandle” API. Finally it spawns 2 system threads.

The first thread is the “communication” thread. ZxShell employs a strange method for communication: it hooks the NtWriteFile API and recognizes 5 different special handle values as commands:

- 0x111111111 -- Hide “Loveusd” driver from the system kernel driver list
- 0x22222222 -- Securely delete an in-use or no-access target file-name
- 0x44444444 -- Unhook the ZwWriteFile API and hook KiFastCallEntry
- 0x55555555 -- Remove the ZxShell Image Load Notify routine
- 0x88888888 -- Set a special value called “type” in Windows registry key HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\DriverMain

The second Loveusd system thread does a lot of things. Its principal duties are to create the ZxShell main DLL in “c:\Windows\System32\commhlp32.dll” and to install the Kernel “Load Image Notify routine”. The code then tries to kill each process and service that belongs to the following list of AV products:

- Symantec Firewall
- Norton
- ESET
- McAfee
- Avast

- Avira
- Sophos
- Malwarebytes

Next, the ZxShell *Load-Image Notify* function prevents the AV processes from restarting.

The installation procedure continues in the user-mode dropper. The ZxShell service is installed as usual, and the in-execution dropper is deleted permanently using the special handle value 0x22222222 for the WriteFile API call. This handle value is invalid: all the windows kernel handle values are by design a multiple of 4. The ZxShell hook code knows that and intercept it.

ObReferenceObjectByHandle is a Kernel routine designed to validate a target object and return the pointer to its object body (and even its handle information), starting from the object handle (even the user-mode one). The hook installed by ZxShell implements one of its filtering routine. It filters each attempt to open the ZxShell protected driver or the main DLL, returning a reference to the "netstat.exe" file. The protection is enabled to all processes except for ones in the following list: Svchost.exe, Lsass.exe, Winlogon.exe, Services.exe, Csrss.exe, ctfmon.exe, Rundll32.exe, mpnotify.exe, update.exe.

If the type of the object that the system is trying to validate is a process, the hook code rewrites again the configuration data of the ZxShell service in the windows registry.

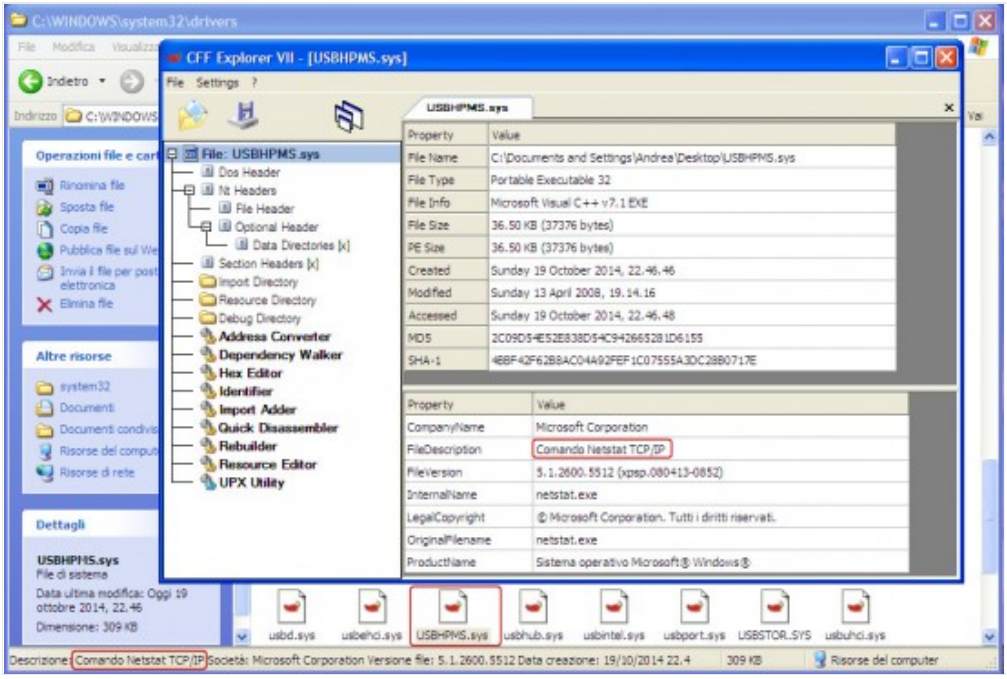


Image 6. Our test Windows XP workstation trying to open the sys file of ZxShell LOVEUSD driver

The last type of Kernel modification that ZxShell rootkit performs is the system call dispatcher (KiFastCallEntry) hook. In this manner, ZxShell is able to completely hide itself, intercepting the following Kernel API calls: *ZwAllocateVirtualMemory*, *ZwOpenEvent*, *ZwQueryDirectoryFile*, *ZwWriteFile*, *ZwEnumerateKey*, and *ZwDeviceIoControlFile*.

Command and Control Server

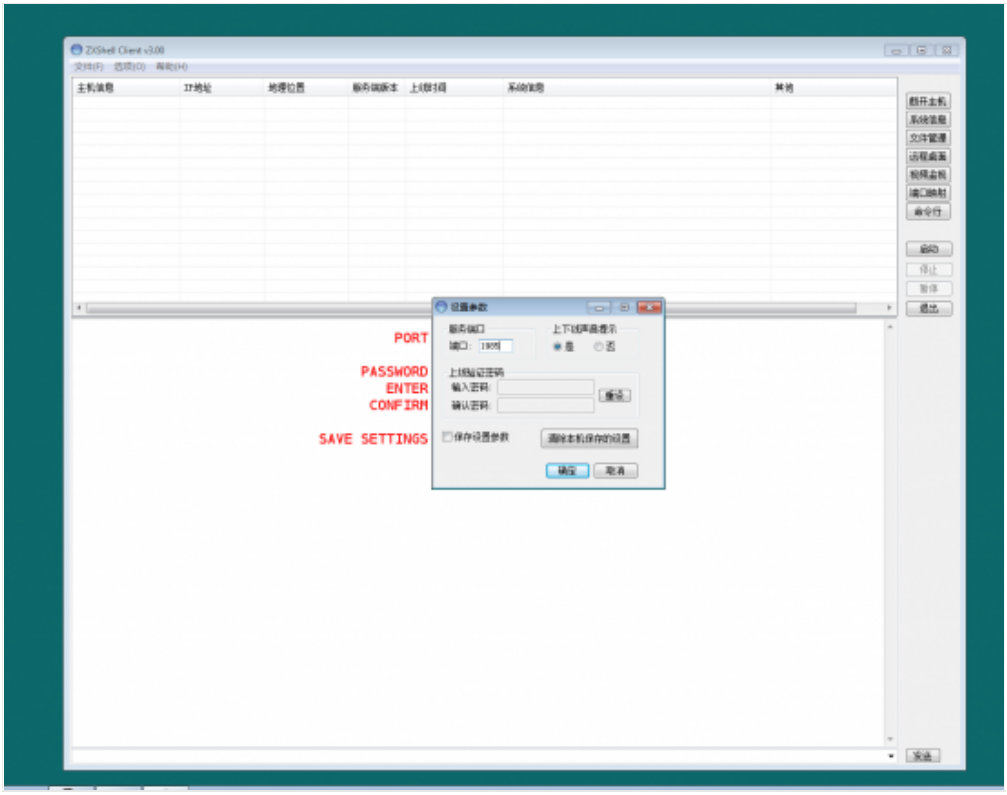
Sample (SHA256: 1eda7e556181e46ba6e36f1a6bfe18ff5566f9d5e51c53b41d08f9459342e26c) is configured to act as a server. The symbol "g_bCreateListenSck" is set to 1. This means that, as seen above, the ZxShell Dll is started in listening mode. It

connects to the first remote C&C that tries to contact it and succeeds in the handshake. The encrypted IP address is "127.0.0.2" (used as loopback) and no connection is made on that IP address (due to the listening variable set to 1).

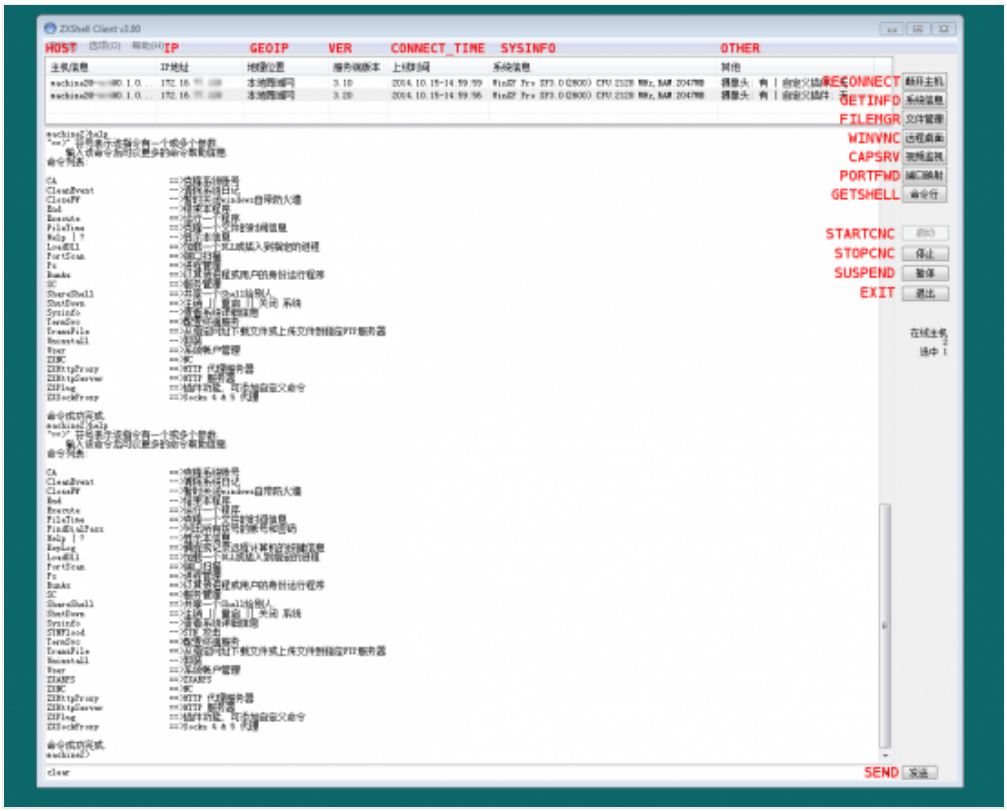
Malware Package

We used the ZxShell package for version 3.10 (SHA256: 1622460afbc8a255141256cb77af61c670ec21291df8fe0989c37852b59422b4).The convenient thing about this is that the CNC panel worked with any version, 3.10 and above. The buttons are all in Chinese, with the help of Google Translate and keen detective skills (read: button clicking), we've deciphered the functionality.

When you start the controller, you need to set the port you want to listen on and if you've set a password, add it here.



Once an infected machine connects, you see its information displayed in a selection box at the top. There are some built in functions on the side for the more common features. These include remote desktop, webcam spying, remote shell, and file management. You can also select a host and type help for a full list of commands.



I have the same machine infected with two different version of ZxShell. Sending the help command for each, you can see the extra features added between version 3.1 and 3.2.

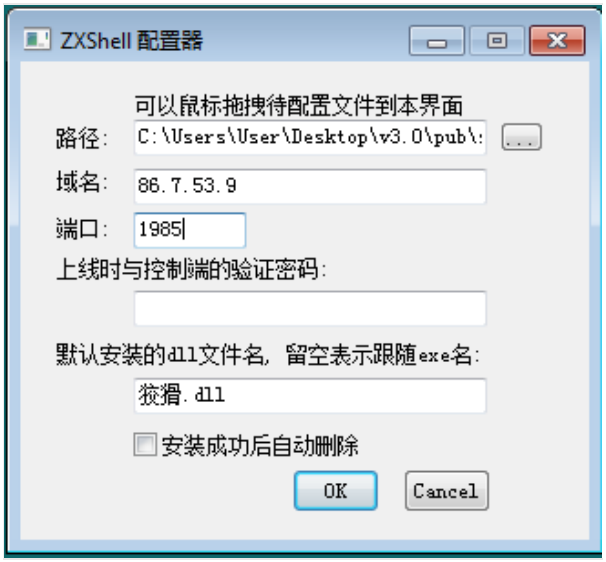
Keylogging, ZXARPS (IP and URL spoofing), and SYNFlood are some of the interesting features added to version 3.2.

Version Information

We wrote a script to extract version info from the binaries we have.

- 3.100 : 914
- 3.200 : 152
- 3.210 : 118
- 3.220 : 14
- 3.390 : 3

In versions 3.1 -- 3.21, the configuration info is xor encoded with 0x85. This configuration info can be changed with a tool included in the ZxShell package.



In versions 3.22 and 3.39 the routine changes. The new xor encoding byte is 0x5B. The data is stored in the last 0x100 bytes of the file. The first 8 bytes of data are static. Then there is the dll install name, the domain, and the port.

Extracted URL Analysis

Knowing the obfuscation routines for this data we wrote a script to extract the URLs / IPs and ports stored.

The most common ports used are, 80, 1985, 1986, and 443. 1985 is the default port for the malware, 1986 is the lazy variation of that port. Port 80 and 443 are the default ports for HTTP and HTTPS traffic. The next most common is port 53. This is used in some of the newer 3.22 and 3.39 samples. After that, the count for each port starts declining sharply. The choices are interesting though, many correspond to what looks like the birth year of the controller (ie. years in the late 1980s and early 1990s), and others seem to match what year the malware was launched in (ie. in the 2000s, relatively close to the current year).

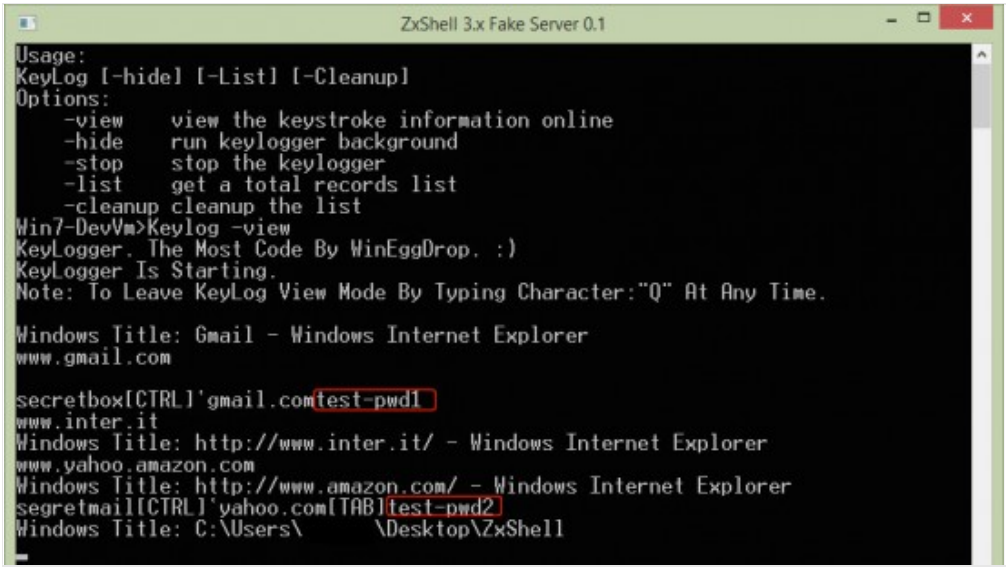
Since this malware dates back to around 2004, there are many samples containing CNC URLs from the 3322.org page. This page used to offer no-ip type hosting and was widely used by malware authors. So much so that Microsoft did a takedown in 2012. A similar service, vicp.net, is also seen in many of the domains.

In the malware, if a domain is configured, it will retrieve domain.tld/myip.txt. This file contains a list of IP addresses for the infected machine to connect back to. Otherwise, if an IP address is configured, it will connect directly to that IP address.

Cloning the ZxShell Server

We have written a simple C++ ZxShell Server that implements the communication and the handshake for the version 3.10 and 3.20 of the ZxShell DLL. The implementation is quite simple: After the handshake, 2 threads that deal with data transfer are spawned. Here we have some screenshots that show the Server and the ZxShell Keylogger in action:

Our server has accepted a connection from a remote host



The ZxShell keylogger has captured 2 user passwords(gmail.com and amazon.com)

The last image shows a very interesting feature of the ZxShell keylogger: once installed and activated, the keylogger is able to catch each password that the user inserts in the login box of each website (like Google, Amazon and so on...). This makes the keylogger a perfect weapons for the attackers. They will be able to steal and resell in the underground market the sensitive data of each victim.

Conclusion

Advanced persistent threats will remain a problem for companies and organizations of all sizes, especially those with high financial or intellectual property value. Group 72's involvement in Operation SMN is another example of what sort of damage that can be done if organizations are not diligent in their efforts to secure their networks. ZxShell is one sample amongst several tools that Group 72 used within their campaign.

ZxShell is a sophisticated tool employed by Group 72 that contains all kinds of functionality. Its detection and removal can be difficult due to the various techniques used to conceal its presence, such as disabling the host anti-virus, masking its

installation on a system with a valid service name, and by masking outbound traffic as originating from a web browser. While other techniques are also utilized to conceal and inhibit its removal, ZxShell's primary functionality is to act as a Remote Administration Tool (RAT), allowing the threat actor to have continuous backdoor access on to the compromised machine.

As our analysis demonstrates, ZxShell is an effective tool that can be ultimately used to steal user credentials and other highly valuable information. The threat posed by ZxShell to organizations is one that cannot be ignored. Organizations with high financial or intellectual property value should take the time to ensure their security requirements are met and that employee's are educated about the security threats their organizations face.

For additional information, please see our blog post.

Protecting Users from These Threats

Product	Protection
AMP	✔
CWS	✔
ESA	✔
Network Security	✔
WSA	✔

Advanced Malware Protection (AMP) is ideally suited to detect the sophisticated malware used by this threat actor.

CWS or WSA web scanning prevents access to malicious websites, including watering hole attacks, and detects malware used in these attacks.

The Network Security protection of IPS and NGFW have up-to-date signatures to detect malicious network activity by threat actors.

ESA can block spear phishing emails sent by threat actors as part of their campaign.

Appendix A: Snort Rules

Initial connection from the infected computer's perspective -- after it connects to the controller -

```
RECV:  851900002504000000004040000000000
SEND:  861900000401000066664640000000000
RECV:  4edf934078010000000000000000000000
SEND:  00000000000000000000000000000000000
```

The rules are on the first 8 bytes of the first two packets. They are hard coded in the binaries. The rest of the bytes are variable (for example, 66664640 is a floating point version number of ZxShell).

Snort rules:

- sid:32180
- sid:32181

These rules have been released in our community ruleset and can be downloaded and used directly, or via pulledpork from Snort.org

Appendix B: ClamAV Signatures

- Win.Trojan.ZxShell-11
- Win.Trojan.ZxShell-12

- Win.Trojan.ZxShell-13
- Win.Trojan.ZxShell-14
- Win.Trojan.ZxShell-15
- Win.Trojan.ZxShell-16
- Win.Trojan.ZxShell-17
- Win.Trojan.ZxShell-18
- Win.Trojan.ZxShell-19
- Win.Trojan.ZxShell-20
- Win.Trojan.ZxShell-21
- Win.Trojan.ZxShell-22
- Win.Trojan.ZxShell-23
- Win.Trojan.ZxShell-24
- Win.Trojan.ZxShell-25
- Win.Trojan.ZxShell-26

These signatures are available within the ClamAV database. Please run freshclam to ensure you stay updated with the latest coverage.

Appendix C: List of Memory Offsets for Some ZxShell Functions

Here's a list for some ZxShell functions for sample SHA256: 1eda7e556181e46ba6e36f1a6bfe18ff5566f9d5e51c53b41d08f9459342e26c:

FUNCTION NAME	BRIEF DESCRIPTION	OFFSET
ZxGetLibAndProcAddr	ZxShell GetProcAddress implementation	0x12CDA
CopyMemoryFromNewMsvcrt	ZxShell memory copy routine	0x12C4C
ServiceExists	Get if a service is installed in the system or not	0x0A7C7
ProcessScCommand	ZxShell "SC" command implementation	0x0E3EF
AnalyseAndLoadPlugins	Parse the installed plugin list and load each one of them	0x0127B7
DeleteAndLogPlugin	Delete a corrupted plugin and log the problem	0x012597
KeyloggerThread	ZxShell keylogger implementation	0x0D591
GetIpListAndConnect	Analyse the IP list inside the ZxShell PE and tries to connect	0x011496
BuildTargetIpListStruct	Build remote server Ip list structure	0x11419
DoHandshake	Perform initial connection handshake	0xB8E8
GetLocalPcDescrStr	Build a string containing the target workstation data	0x0B627
MainConnectionIo	ZxShell main connection I/O routine	0x1126C
ReceiveCommandData	Receive each byte from the socket until a newline char	0x016DF
ProcessCommand	Main ZxShell command processing routine	0x10C2B

Appendix D: Other Collateral

Here is a non-exhaustive list of ZxShell samples that were analyzed for this report.

Here is a list of Domains organized by port.

Tags: APT, Group 72, malware, Operation SMN, security, SMN, Talos, threats