Malware Analysis: Citadel

2012. 12. 14

AhnLab ASEC (AhnLab Security Emergency response Center)
Analysis Team

The following is a detailed analysis report of the malware Citadel. As with similar malware such as Zeus and SpyEye, Citadel functions as an agent that sets up a botnet and an info-stealer that extracts authentication data. Citadel is also a bot agent that downloads and executes files to install a variety of malware

AhnLab

1. Similarity between Citadel and Zeus

A. Overview

The Citadel Trojan is malware created by a malicious code generating program. It is very similar to the Zeus Trojan in terms of logical structure as well as physical data. This means Citadel is also designed to steal personal information used in financial transactions likes Zeus did. Moreover, it allows attacker do DDoS that paralyze large-scale systems and infrastructure, because Zeus and Citadel basically constructs the extensive botnet that consists of large number of infected computers. In addition, the attacker can arbitrarily execute any malicious codes such as ransomeware and scareware on the infected computer that has already installed bot agent(Backdoor).

The dangers of the Citadel Trojan have been brought into the limelight from the start of 2012. It is known to have been created based on the source code of Zeus. A number of malware (Citadel, Gameover Zeus, Ice IX, Licat, Murofet) have been created after the source code of Zeus was made public, but the team creating Citadel is the most organized and continuing to provide service among them.

B. Similarities

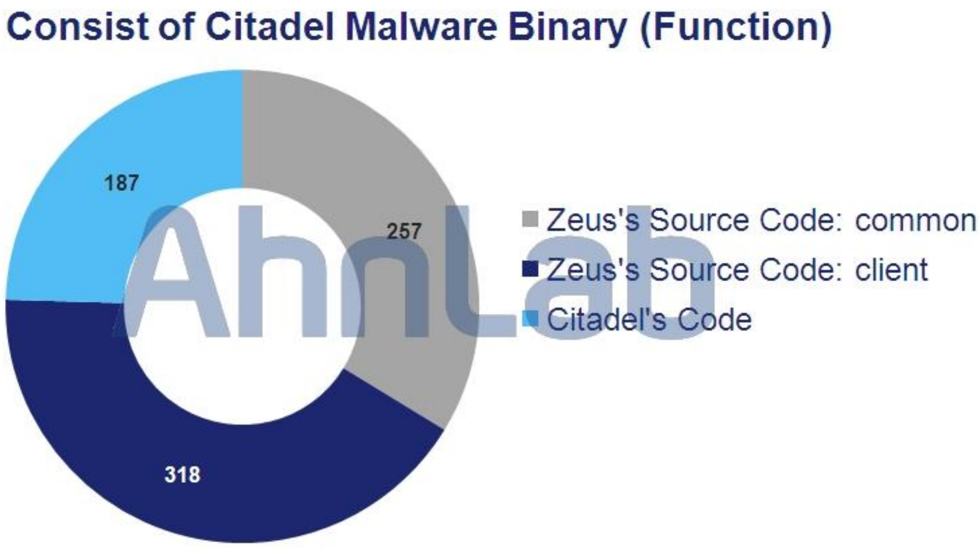


Fig. 1. Code-level Similarity: Citadel vs. Zeus

Citadel (bot agent) is 213,504 bytes and made up of 762 functions. Citadel shares 575 identical functions with the Zeus source code. Of those, 318 are designed to mimic the function of the Zeus source code and the other 257 are utility routines such as strings, memory, network and encryption.

In short, Citadel physically matches Zeus by approximately 75%. The other 25% of routines consist of new manage functions(e.g. Main, Initialization and Finalization), new encryption libraries and utility functions.

2. Citadel: Common with Zeus

A. Citadel Malware Installation

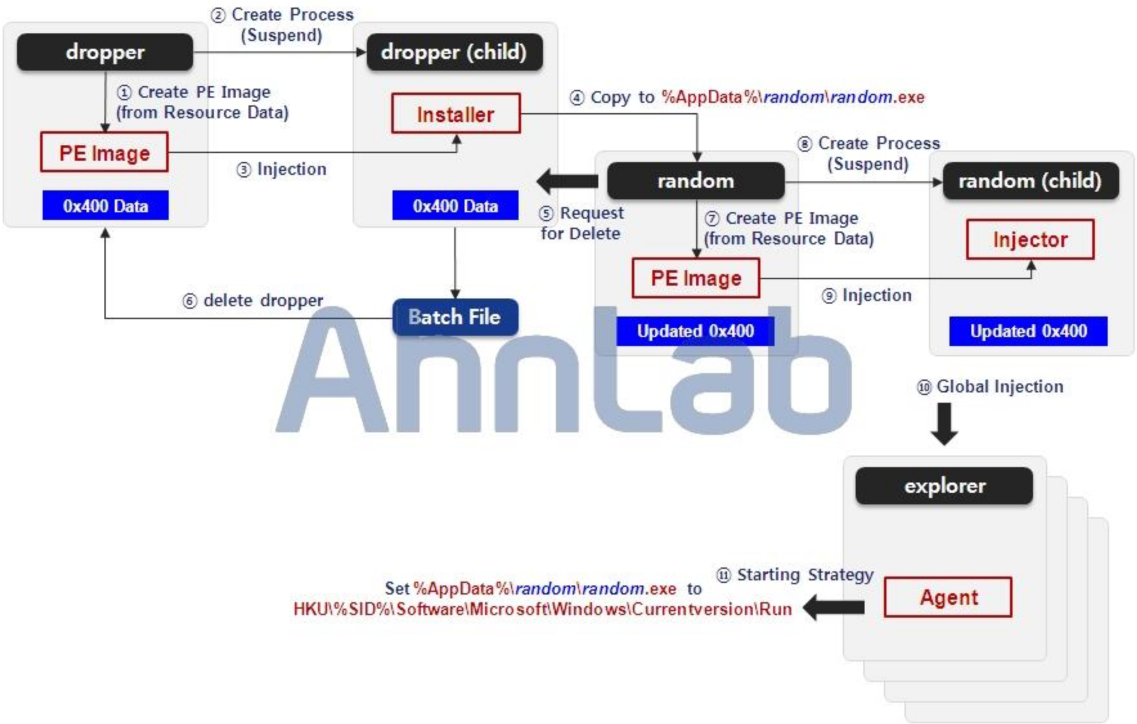


Fig. 2. Citadel Malware Installation Process

When Citadel is executed, the Citadel Agent is installed as shown in Fig. 2. ① to ③ show the image-dropping technique frequently employed by malware. The dropper (child) process appears to have an image of dropper.exe, but what actually results is a process with a completely different image generated by the dropper process. Malware creators freqently use the image-dropping technique because it allows more covert PE generation and execution than file-dropping. In other words, it is used to mask their presence to evade detection by security software.

The newly-generated PE image does not engage in malicious activity immediately; instead, it copies dropper.exe to a target location. The PE image seems to simply copy itself below the path %AppData% in a random name, but debugging reveals that the data of the 0x400 bytes at the end of the file is restructured and overwritten at the end of random.exe. This encrypted data of 0x400 bytes subsequently control the routine flow of the PE image. For further technical details, refer to [Attachment:Technical – 0x400data Decryption Process].

Once the installed Citadel malware (random.exe) is executed, a batch file is used to delete the dropper (⑤, ⑥). More precisely, the random process sends a signal to the dropper (child) process that remains active. The dropper (child) process then performs the command by generating and executing a batch file for deleting dropper.exe before ending itself. At this point, dropper.exe is deleted without issues because the very first dropper process has already terminated.

⑦ to ⑨ are identical to ① to ③. The random process executes the exact same activity executed by the dropper process except the deletion process.

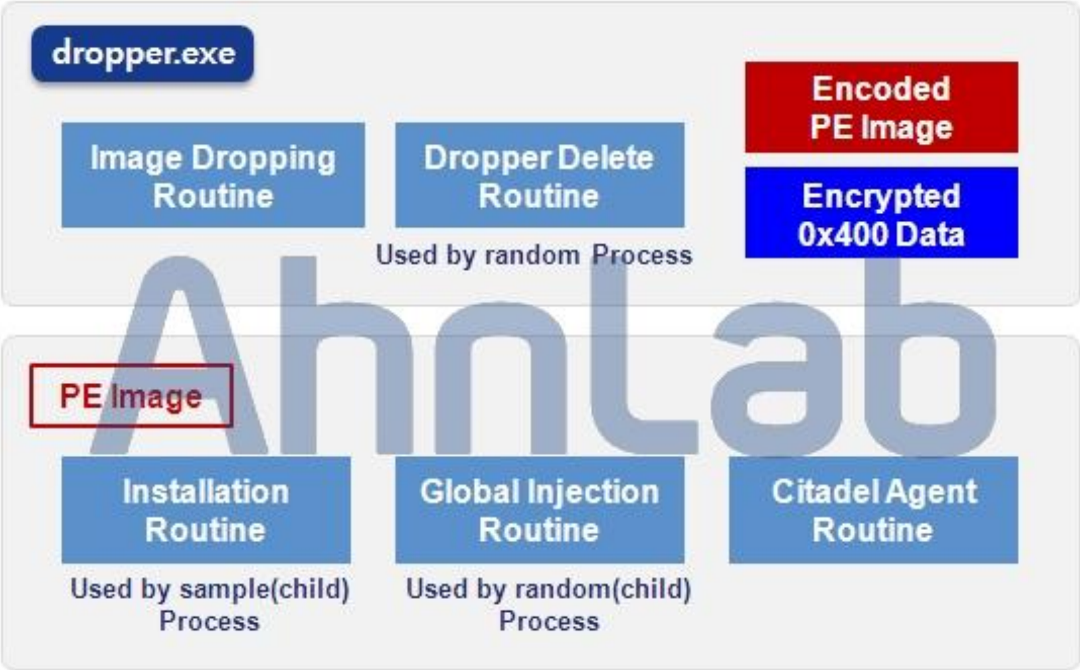


Fig. 3. Logical internal routine of dropper.exe and the PE image

The random (child) process upon completion of image-dropping, however, appears to be quite different from the dropper (child) process. The PE image data injected in the virtual memory are completely identical, but an entirely different routine is executed due to the 4-byte flag in the 0x400-byte data described above. The random (child) process injects the PE image in the explorer process and all of its child processes (i.e. ⑩: global injection). The injected PE image (Citadel agent) functions in each process as an independent thread.

The Citadel agent injected in the explorer process and its child processes is identical to that of the dropper (child) and random (child), but functions as an entirely different program as the entry point is different. Fig. 3. shows the logical structure (overall function) of the Citadel Trojan, which has been designed to execute the routine required by context in the PE file and image. Here, it can be seen that the core function (bot agent and info-stealer functions) of Citadel is executed in the Citadel agent routine. System fix for as a malware re-execution strategy is also executed in the Citadel agent routine (⑪). The strategy used is the registration of absolute path random.exe in the registry key shown below.

- **HKU\%SID%\Software\Microsoft\Windows\Currentversion\Run**

Design of the Citadel dropper is primarily focused on security program and analysis evasion. For example, the code is made to appear as a GUI program instead of a malware dropper and the API actually used by the code is assigned dynamically. As expected, the library name and function name to be used are encrypted. Also, binary obfuscation technology is used to hinder analysis through reverse engineering and API calls take place using a dynamically-written shellcode. For technical details on the anti-analysis technique described above, refer to [Attachment: Technical – Citadel Dropper: Anti-analysis Method].

B. Citadel Agent Functionality

The Citadel agent (injected code) has two functions: a passive function automatically executed on an infected system and an active function executed upon receipt of a command from the C&C server. The passive function is executed by the hooked code that set through hooking in the Citadel agent initialization functions when required. While the active function is also executed by the hooked code set in the network API, it is not executed without a command (network packet) from the C&C.

• Passive: User-level API Inline Hooking

Inline hooking of Citadel is executed in the function {.text:0041b7ee}. The gethostbyname and getaddrinfo APIs are hooked by the injector as a test. The APIs shown in [Table 1] are actually hooked when the Citadel agent is initialized. As the table shows, the majority of the hooked codes are identical to those of the Zeus source code. As such, these codes will be categorized by class file (*.cpp) and explained. For newly-added or edited hooked codes, refer to Chapter 3.

API Name	Hooked Code	Same as Zeus's Hooked Code
NtCreateUserProcess	.text:00419AEA	corehook.cpp / hookerNtCreateUserProcess()
NtCreateThread	.text:00419A34	corehook.cpp / hookerNtCreateThread()
LdrLoadDll	.text:00419C0F	Citadel Unique
ExitProcess	.text:00419E37	Citadel Unique
GetFileAttributesExW	.text:00419E78	corehook.cpp / hookerGetFileAttributesExW()
CreateProcessAsUserA	.text:00419EDE	Citadel Unique
CreateProcessAsUserW	.text:00419EF5	Citadel Unique
PlaySoundA	.text:00419F0C	Citadel Unique
PlaySoundW	.text:00419F33	Citadel Unique
HttpOpenRequestW	.text:0041D72A	Citadel Unique (But Similar as Zeus')
HttpOpenRequestA	.text:0041D768	Citadel Unique (But Similar as Zeus')
HttpSendRequestW	.text:0041D7A6	wininethook.cpp / httpSendRequestBodyW()
HttpSendRequestA	.text:0041D7FB	wininethook.cpp / httpSendRequestBodyA()
HttpSendRequestExW	.text:0041D850	wininethook.cpp / httpSendRequestExBodyW()
HttpSendRequestExA	.text:0041D8ED	wininethook.cpp / httpSendRequestExBodyA()
HttpEndRequestA	.text:0041D98A	Citadel Unique (But Similar as Zeus')
HttpEndRequestW	.text:0041D9D5	Citadel Unique (But Similar as Zeus')
InternetCloseHandle	.text:0041DA20	Citadel Unique (But Similar as Zeus')
InternetReadFile	.text:0041DA8D	wininethook.cpp / hookerInternetReadFile()
InternetReadFileExA	.text:0041DABB	wininethook.cpp / hookerInternetReadFileExA()
InternetSetFilePointer	.text:0041DB3A	Citadel Unique (But Similar as Zeus')
InternetQueryDataAvailable	.text:0041DB94	wininethook.cpp / hookerInternetQueryDataAvailable()
HttpQueryInfoA	.text:0041DBC0	wininethook.cpp / wininethook_hookerHttpQueryInfoA()
closesocket	.text:004249ED	sockethook.cpp / hookerCloseSocket()
Send	.text:00424A25	sockethook.cpp / hookerSend()
WSASend	.text:00424A46	sockethook.cpp / hookerWsaSend()
OpenInputDesktop	.text:00411A9E	vncserver.cpp / hookerOpenInputDesktop()
SwitchDesktop	.text:00411AEE	vncserver.cpp / hookerSwitchDesktop()

DefWindowProcW	.text:00411B0C	vncserver.cpp / hookerDefWindowProcW()
DefWindowProcA	.text:00411B52	vncserver.cpp / hookerDefWindowProcA()
DefDlgProcW	.text:00411B98	vncserver.cpp / hookerDefDlgProcW()
DefDlgProcA	.text:00411BDE	vncserver.cpp / hookerDefDlgProcA()
DefFrameProcW	.text:00411C24	vncserver.cpp / hookerDefFrameProcW()
DefFrameProcA	.text:00411C6D	vncserver.cpp / hookerDefFrameProcA()
DefMDIChildProcW	.text:00411CB6	vncserver.cpp / hookerDefMDIChildProcW()
DefMDIChildProcA	.text:00411CFC	vncserver.cpp / hookerDefMDIChildProcA()
CallWindowProcW	.text:00411D42	vncserver.cpp / hookerCallWindowProcW()
CallWindowProcA	.text:00411D8B	vncserver.cpp / hookerCallWindowProcA()
RegisterClassW	.text:00411E10	vncserver.cpp / hookerRegisterClassW()
RegisterClassA	.text:00411E5D	vncserver.cpp / hookerRegisterClassA()
RegisterClassExW	.text:00411EAA	vncserver.cpp / hookerRegisterClassExW()
RegisterClassExA	.text:00411EFC	vncserver.cpp / hookerRegisterClassExA()
BeginPaint	.text:00423EF7	vncserver.cpp / hookerBeginPaint()
EndPaint	.text:00423F67	vncserver.cpp / hookerEndPaint()
GetDCEx	.text:00423FA7	vncserver.cpp / hookerGetDcEx()
GetDC	.text:00424002	vncserver.cpp / hookerGetDc()
GetWindowDC	.text:00424041	vncserver.cpp / hookerGetWindowDc()
ReleaseDC	.text:00424080	vncserver.cpp / hookerReleaseDC()
GetUpdateRect	.text:004240C0	vncserver.cpp / hookerGetUpdateRect()
GetUpdateRgn	.text:00424153	vncserver.cpp / hookerGetUpdateRgn()
GetMessagePos	.text:00417479	vncserver.cpp / hookerGetMessagePos()
GetCursorPos	.text:004174AB	vncserver.cpp / hookerGetCursorPos()
SetCursorPos	.text:004174F2	vncserver.cpp / hookerSetCursorPos()
SetCapture	.text:0041752F	vncserver.cpp / hookerSetCapture()
ReleaseCapture	.text:00417589	vncserver.cpp / hookerReleaseCapture()
GetCapture	.text:004175D9	vncserver.cpp / hookerGetCapture()
GetMessageW	.text:00417678	vncmouse.cpp / hookerGetMessageW()
GetMessageA	.text:004176A0	vncmouse.cpp / hookerGetMessageA()
PeekMessageW	.text:004176C8	vncmouse.cpp / hookerPeekMessageW()
PeekMessageA	.text:004176F3	vncmouse.cpp / hookerPeekMessageA()
TranslateMessage	.text:00415C38	userhook.cpp / hookerTranslateMessage()

GetClipboardData	.text:00415DAE	userhook.cpp / hookerGetClipboardData()
PFXImportCertStore	.text:00413F13	certstorehook.cpp / _hookerPfxImportCertStore()
gethostbyname	.text:00424585	Citadel Unique
getaddrinfo	.text:004245FE	Citadel Unique

[Table 1] Windows API Hooking Point (Common)

Hooked Code Defined in corehook.cpp

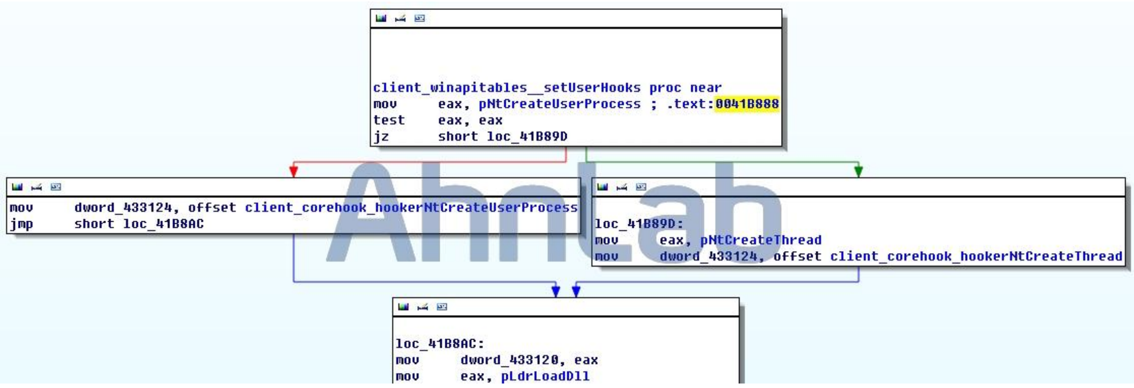


Fig. 4. Entry Point of Hooking Code

Before the hooked codes defined in the {coreHook.cpp} file of Zeus are explained, the hooking code must be explained. Although [Table 1] shows that both NtCreateUserProcess and NtCreateThread APIs are hooked, NtCreateUserProcess and NtCreateThread are not hooked simultaneously as shown in Fig. 4. The hooking codes are programmed in sequence so that NtCreateThread is hooked if the address of NtCreateUserProcess is not attained.

The functions of hooked codes hookerNtCreateUserProcess() and hookerNtCreateThread() are identical. When a new process is generated, the Citadel agent injects itself by calling its injector routine. Subsequently, the global injection executed by the injector is maintained.

Hooked Code Defined in wininethook.cpp

The hooked codes set in network APIs are designed for the following two functions.

- **HTTP Session Redirection**
- **Web Injection (MITB Attack)**

The hooked code set in the HttpOpenRequest API separately manages new HTTP sessions created. Sessions to be managed are not selected; all sessions are managed from the middle. The hooked code set in the HttpEndRequest and IneternetClose APIs is an upgraded version of that of the Zeus source code. (which explains the similarity to the Zeus code, but they are technically different.) These hooked codes delete each session that ends from the data structure established for management. The following is the structure used for session management. Each session has the structure shown in Fig. 5.

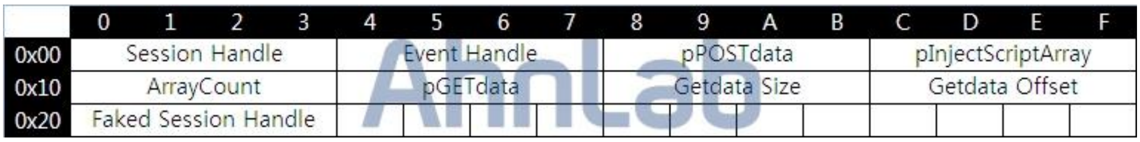


Fig. 5. HTTP Session Management Structure

The Session Handle field shows the session managed by the structure and the Event Handle field is used for synchronization. The pPOSTdata field and pGETdata field are pointers that indicate spoofed GET/POST data. The pInjectScriptArray field is a pointer that indicates the embedded script to be used for web injection (MITB attack). A session handle that will connect to a destination set by the attacker instead of the normal destination is saved in the Fake Session Handle field(used for http session redirection).

Hooked Code Defined in sockethook.cpp

The hooked code set in the WSASend, Send and closesocket APIs become the trigger for executing the following info-stealing functions.

- **FTP Credential Theft**
- **POP3 Credential Theft**
- **Macromedia Flash Files Control**

[Table 2] shows the routines called by the hooked codes for each FTP/POP3 software from which credentials are attempted to be stolen. Each routines are required to be different by each software because it have different credential locations and security algorithms(e.g. encoding or encryption).

Software	Code for Credential Theft	Zeus's Source Code
Total Commander	.text:00420272	ftpTotalCommanderReadIni()
	.text:00420415	ftpTotalCommanderProc()
	.text:00420107	ftpTotalCommanderDecrypt()
	.text:00420230	ftpTotalCommanderBasicSearch()
	.text:004200E2	randTotalCommander()
	.text:0042045E	_ftpTotalCommander()
WSFTP	.text:0042066D	ftpWsFtpProc()
	.text:00420633	ftpWsFtpBasicSearch()
	.text:0042089C	_ftpWsFtp()
WinSCP	.text:00420FD3	ftpWinScpDecrypt()
	.text:0042108D	_ftpWinScp()
SmartFTP	.text:00421D3D	ftpSmartFtpProc()
	.text:00421849	ftpSmartFtpDecrypt()
	.text:00421CF8	ftpSmartFtpBasicSearch()
	.text:00421F91	_ftpSmartFtp()
FTP Commander	.text:004212EF	ftpFtpCommanderProc()
	.text:004212D5	ftpFtpCommanderMarkStringEnd()
	.text:0042157E	_ftpFtpCommander()
FlashFXP	.text:0041FDF0	ftpFlashFxp3Proc()
	.text:0041FCE3	ftpFlashFxp3Decrypt()

	.text:0041FD96	ftpFlashFxp3BasicSearch()
	.text:0041FFDE	_ftpFlashFxp3()
FileZilla	.text:004209A8	ftpFileZillaProc()
	.text:00420C4D	_ftpFileZilla()
Far Manager	.text:00420CF0	ftpFarManagerDecrypt()
	.text:00420D4A	_ftpFarManager()
Core FTP	.text:00421621	_ftpCoreFtp()
Macromedia Flash	.text:0041ECF8	getFlashPlayerPath()
	.text:0041ED87	_removeMacromediaFlashFiles()
	.text:0041ED3C	_getMacromediaFlashFiles()
Outlook Express	.text 0041F5C5	windowsMailAccountProc()
	.text 0041F3BB	getWindowsMailString()
	.text:0041EDAD	enumWindowsMailMessagesAndFolders()
	.text:0041F40E	appendWindowsMailInfo()
	.text:0041F02A	appendOutlookExpressInfo()
	.text:0041EF2E	_emailWindowsMailRecipients()
	.text:0041F7D8	_emailWindowsMail()
	.text:0041FB0E	_emailWindowsContacts()
	.text:0041F8BF	_emailWindowsAddressBook()
	.text:0041F16A	_emailOutlookExpress()
Common	.text:0041ECC1	writeReport()
	.text:0042207F	_ftpAll()
	.text:0041FCB3	_emailAll()

[Table 2] Credential Theft Routines

Hooked Code Defined in vncserver.cpp and vncmouse.cpp

The functions found in {vncserver.cpp} and {vncmouse.cpp} allow remote control of a client on which the Citadel agent is installed. While a normal remote control program is an agent program that functions in a single normal process, Citadel's remote control takes place through the dynamic synchronization of the hooked codes of related APIs.

It should be noted that a remote control function initialization process is required for the valid operation of this hooked code. Such a process is executed by option –v in the Main routine of the injector as shown in Fig. 6.

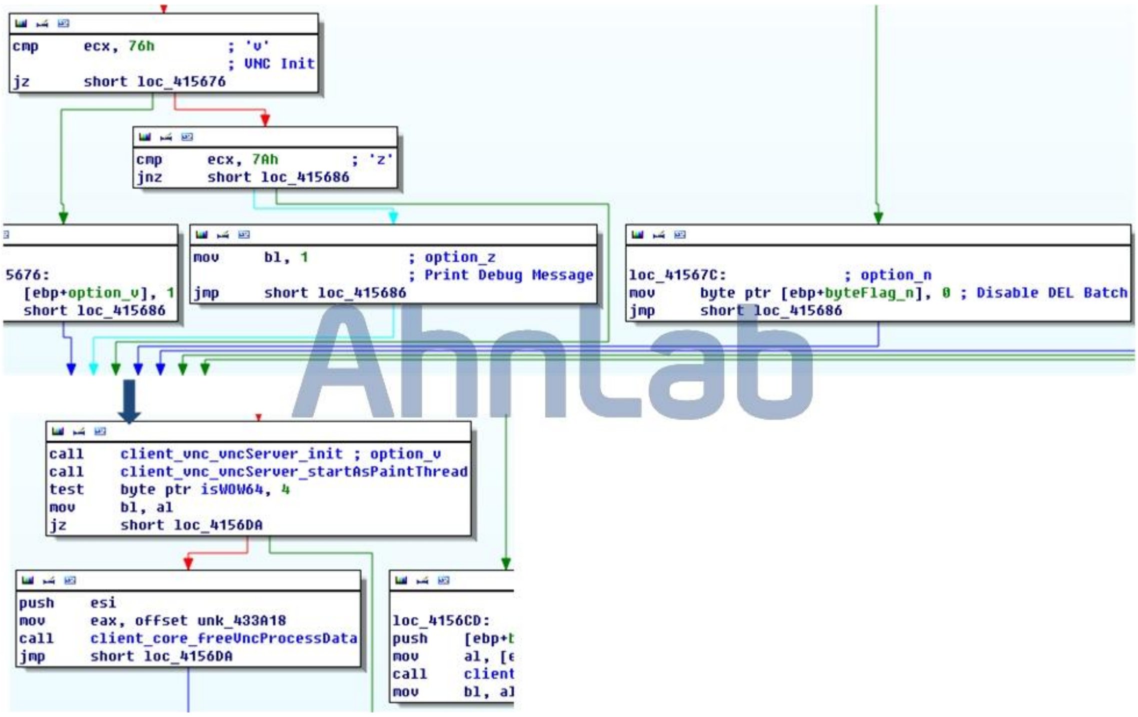


Fig. 5. Injector Routine Control Flow with v option

Hooked Code Defined in userhook.cpp

The hooked code for TranslateMessage and GetClipboardData found in {userhook.cpp} performs the functions of key stroke and screen scrapper as shown in Fig. 7. String data saved in the clipboard is also a target for theft.

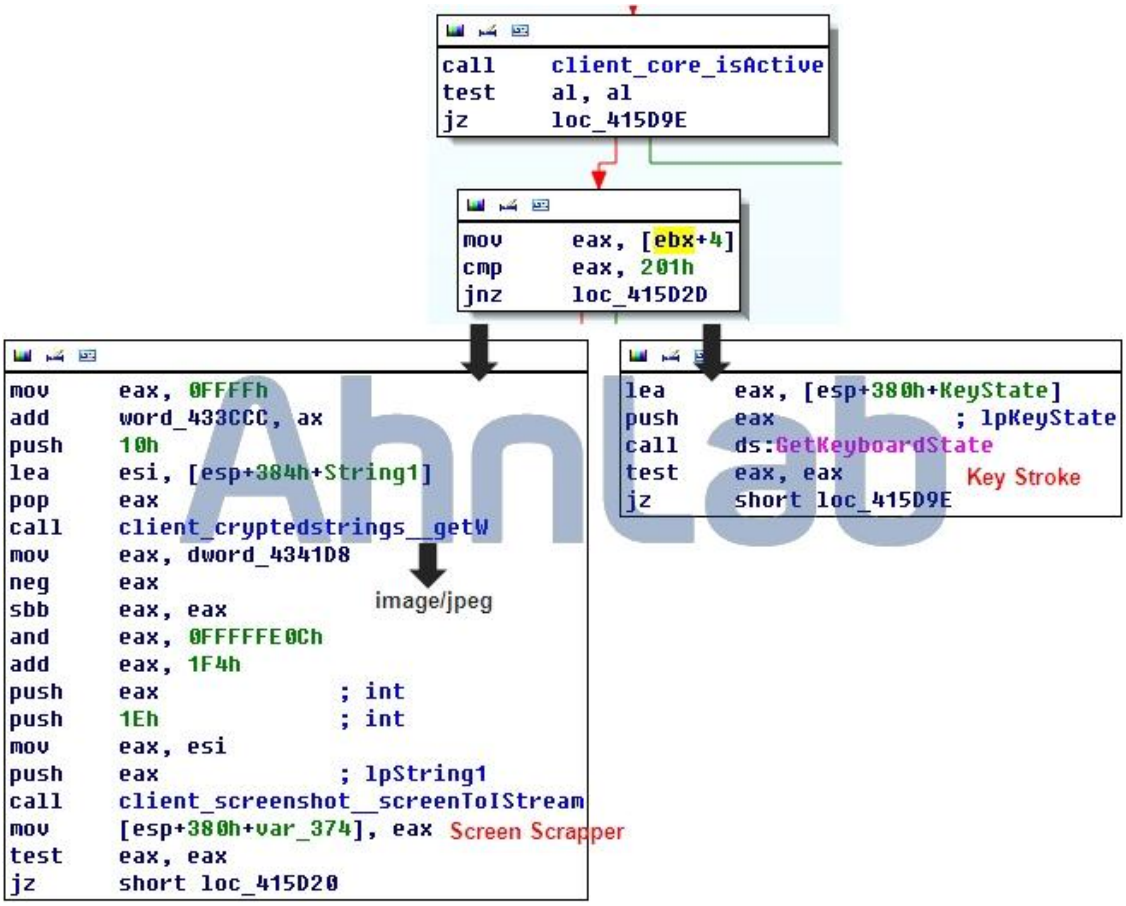


Fig. 6. Hooked Code: TranslateMessage API

Hooked Code Defined in certstorehook.cpp

The hooked code found in {certstorehook.cpp} is used to steal the private keys related to certificates saved on the client. To load this certificate, the PFXImportCertStore API must be called. By hooking the PFXImportCertStore API, theft of both the certificate and its key is attempted when the certificate is loaded.

• Web Browser API Inline Hooking

If web browser Firefox or Opera is used, network communication is executed through the Firefox or Opera API instead of the Windows API. Thus malicious activities such as web injection and http session redirection cannot be performed with the same hooked code if these two web browsers are used. As such, Citadel directly hooks the network communication library (nspr4.dll) of each web browser. [Table 3] shows the export functions of nspr4.dll hooked.

Export Function Name	Hooked Code	Same as Zeus's Hooked Code
PrClose	.text:004134D4	nspr4hook.cpp / hookerPrClose()
	.text:0041B332	nspr4hook.cpp / hookerPrClose()
	.text:0041294E	nspr4hook.cpp / hookerPrClose()
PrOpenTcpSocket	.text:0041349A	nspr4hook.cpp / hookerPrOpenTcpSocket()

	.text:0041B2DC	nspr4hook.cpp / hookerPrOpenTcpSocket()
PrRead	.text:0041AFA6	nspr4hook.cpp / hookerPrOpenTcpRead()
PrWrite	.text:0041ADB0	nspr4hook.cpp / hookerPrOpenTcpWrite()

[Table 3] nspr4.dll Library Hooking Point

PrOpenTcpSocket is a function that performs the same function as HttpOpenRequest in the Windows API. Thus the hooked code set here separately manages created http sessions by creating the structure shown in Fig. 8. PrClose performs the same function as HttpEndRequest of the Windows API. As with HttpEndRequest, it removes the structure when a session is ended.

	0 1 2 3	4 5 6 7	8 9 A B	C D E F	
0x00	Session Handle	pURL	skipBytesWrite	pInjectScriptArray	
0x10	ArrayCount	pBufferFromServer	BufferSize	*buf	PendingRequest
0x20	bufSize	position	realSize	*buf	PendingResponse
0x30	bufSize	position			

Fig. 7. HTTP Session Management Structure (nspr4.dll)

The Session Handle field shows the handle returned by the function PrOpenTcpSocket. The second field, pURL, is a pointer that indicates the URL string of a connected website. Saved in skipBytesWirte are data sizes to be ignored by PR_WRITE. The fourth field is identical to that of the Windows API (string array pointer in which the script to be used for web injection is saved). ArrayCount indicates how many arrays exist. The pBufferFromServer pointer indicate the location where data received from the server is saved and BufferSize indicates the size of the saved location. PendingRequest and PendingResponse are structures that each contains network request and response data.

• Active: Citadel C&C Command List

Citadel C&C Command List	
os_shutdown	**os_reboot**
url_open	
bot_uninstall	**bot_update**
dns_filter_add	**dns_filter_remove**
bot_bc_add	**bot_bc_remove**
bot_httpinject_disable	**bot_httpinject_enable**
fs_path_get	**fs_search_add**
fs_search_remove	
user_destroy	**user_logoff**
user_execute	**user_cookies_get**
user_cookies_remove	**user_certs_get**
user_certs_remove	**user_url_block**

user_url_unblock	user_homepage_set
user_ftpclients_get	user_emailclients_get
user_flashplayer_get	user_flashplayer_remove
module_execute_enable	module_execute_disable
module_download_enable	module_download_disable
info_get_software	info_get_antivirus
info_get_firewall	
search_file	upload_file
download_file	
ddos_start	ddos_stop

[Table 4] Citadel C&C Command List derived from Encrypted Data

3. Citadel: Additional Factors

A. Windows API Hooking

Citadel malware attempts inline hooking to the Windows API to covertly perform malicious activity. It is identical to Zeus in that regard. However, Citadel has other hooking points in addition to the APIs hooked by Zeus. Also, some functions of Zeus' hooked codes have been edited. [Table 5] shows hooking points and hooked codes that have been added or edited.

Windows API	Hooked Code	Description
LdrLoadDll	.text:00419C0F	Hooking Trigger for nspr4.dll, chrom.dll
ExitProcess	.text:00419E37	Citadel Finalization Code
CreateProcessAsUserA	.text:00419EDE	Integrity Up (Code for upper Windows VISTA)
CreateProcessAsUserW	.text:00419EF5	
PlaySoundA	.text:00419F0C	Noise made on the client when remote control takes place is removed
PlaySoundW	.text:00419F33	
gethostbyname	.text:00424585	Pharming (DNS Redirection)
getaddrinfo	.text:004245FE	

[Table 5] Windows API Hooking Point (Citadel)

The hooked code set in the LdrLoadDll API monitors the times when nspr4.dll (Firefox, Opera) and chrome.dll (Chrome) are loaded. When loading is detected, the code attempts to hooks network function provided by nspr4.dll and chrome.dll. Functions that hooking is attempted for and the functions of the set hooked codes are shown in Chapter 2.B(Web Browser API Inline Hooking) for nspr4.dll and Chapter 3.B for chrome.dll.

The hooked code set in the ExitProcess API performs finalization of the Citadel malware. While most of Citadel’s functions are identical to those of Zeus, the modules that control the functions (e.g. main and initialization routines) were developed independently and thus a different finalization code is required.

The hooked code set in the CreateProcessAsUser API is for platforms higher than Windows Vista. It increases the integrity level of generated processes.

The hooked code set in the PlaySound API is intrinsically related to VNC service. It is a code that prevents noise from being made on a PC infected by Citadel when remote control is taking place. It is suspected that the code has been designed to prevent a clicking noise that might be heard by nearby individuals if speakers are connected to an infected PC.

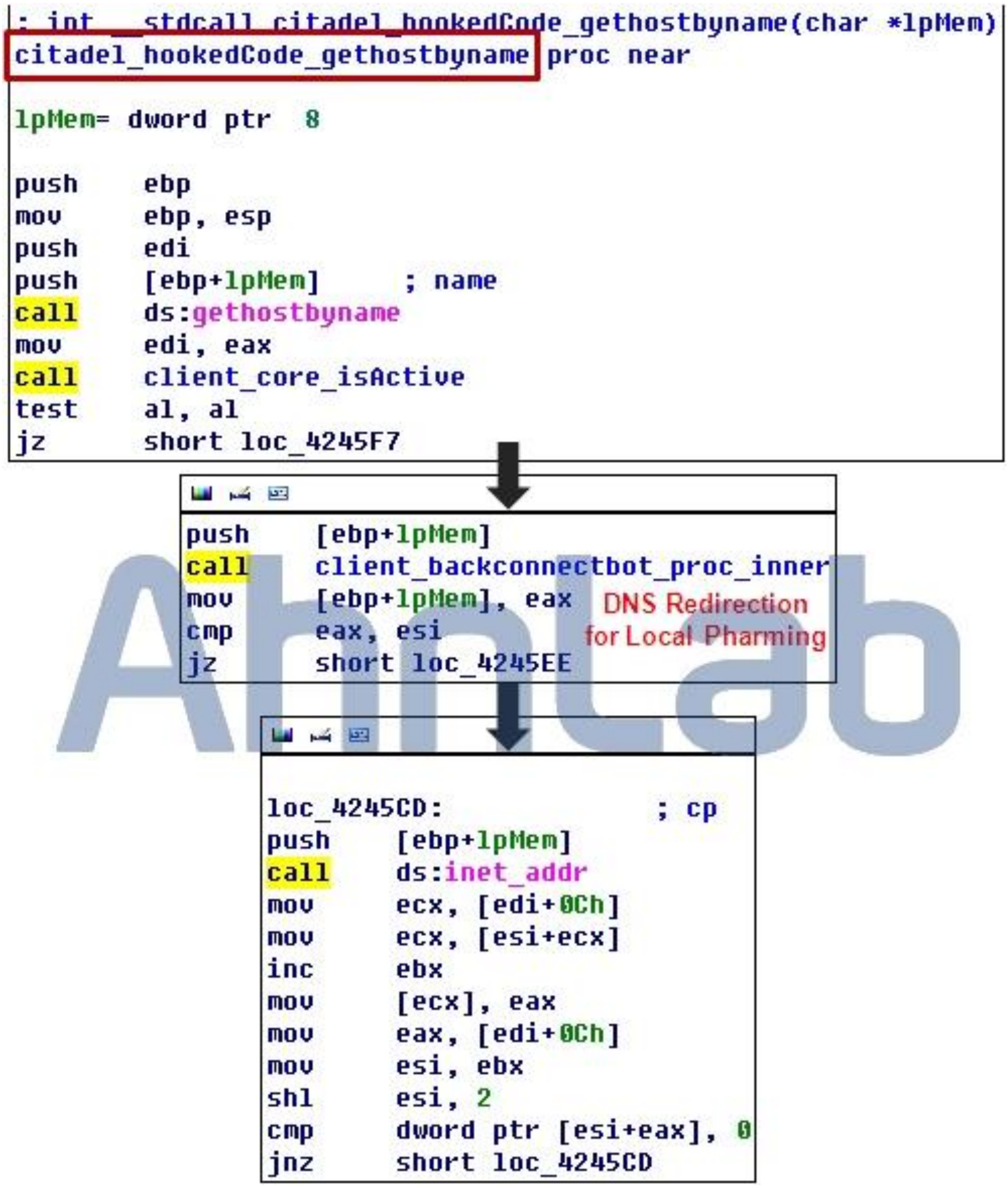


Fig. 8. Hooked Code: gethostbyname API

The hooked codes set in the gethostbyname and getaddrinfo APIs are for local pharming. This is a new malware function unique to Citadel. For pharming, DNS redirection through Windows API hooking takes place. However, Fig. 9 shows that codes provided by Zeus have been used to achieve this function. What this means is that Citadel uses Zeus' physical modules to achieve the new logical function of local pharming.

B. Chrome Web Browser API Hooking

Zeus not only hooks the network APIs of Windows but also the network functions provided by web browsers such as Firefox and Opera. This is because Firefox and Opera use their own functions (not Windows APIs) for network communications. Chrome, the most popular web browser of late, also uses its own functions for network communications. The Citadel malware even hooks those functions for http redirection and web injection (MITB attack) on Chrome.

Finding hooking points on FireFox and Opera is easy because they export network connection/disconnection (PR_OPEN, PR_CLOSE) and IO calculation (PR_READ, PR_WRITE, ...) functions from a library file called nspr4.dll. On the other hand, it is relatively difficult to find a hooking point on Chrome as its main DLL chrome.dll does not export such functions. For that reason, Citadel finds a hooking point by identifying a code pattern in the chrome.dll image loaded on the memory. The address of the hooking code that identifies a code pattern in chrome.dll and executes inline hooking is

{.text:0x0041BEA5}.

Functionality	Function RVA in chrome.dll	Hooked Code
Open	0xC3FA72	.text:41B69C
	0xC422BA	.text:41B69C
Close	0xC3FD27	.text:41B6B8
	0xC43478	.text:41B6B8
Read	0xC3FEEA	.text:41B6CE
	0xC4265A	.text:41B6CE
Write	0xC40056	.text:41B6F2
	0xC426AF	.text:41B6F2

[Table 6] chrome.dll Library Hooking Point

[Table 6] shows the hooking points for Chrome Version 15.0.874.106. The analyzed sample succeeds in hooking by finding the code pattern of Chrome Version 15.x, 16.x and 17.x. The Chrome team promptly responds to such malware attacks and the Citadel team also provides ongoing updates.

C. AES Crypto Algorithm

The Citadel malware performs multi-layered encryption of configuration files (static/dynamic) and stolen data by using Custom Xoring, RC4 encryption algorithm and AES encryption algorithm.

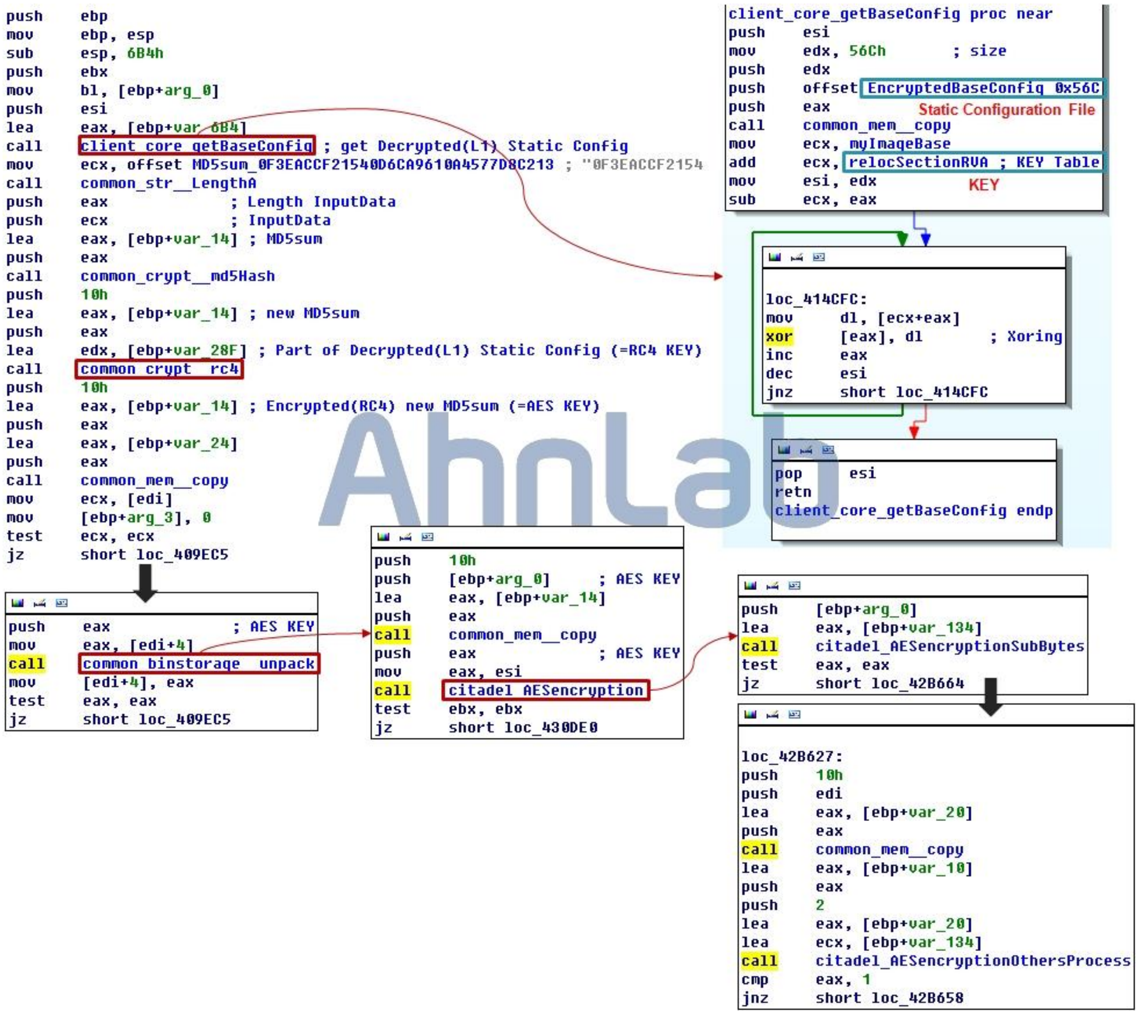


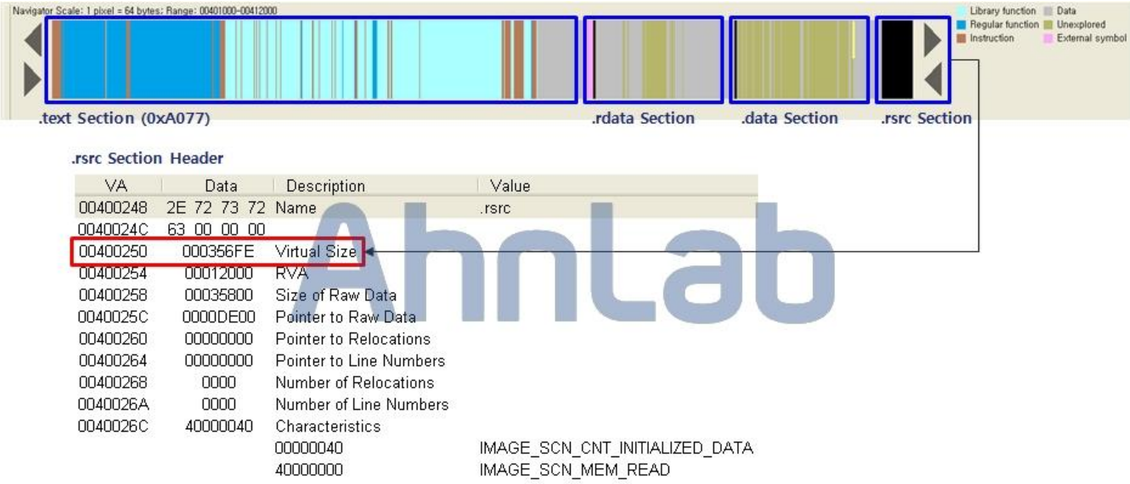
Fig. 9. Citadel Multi-layered Encryption (Decryption Process)

Fig. 10 shows routine {.text:00409DF7}, executed when configuration data update is commanded by the C&C, and is an apt representation of Citadel's multi-layered encryption. More precisely, Fig. 10 shows the process in which the AES round key is obtained by referencing the md5sum data {.text:00401868} included in the static configuration data {.text:004064D0} and Citadel agent. Details of the process are as follows.

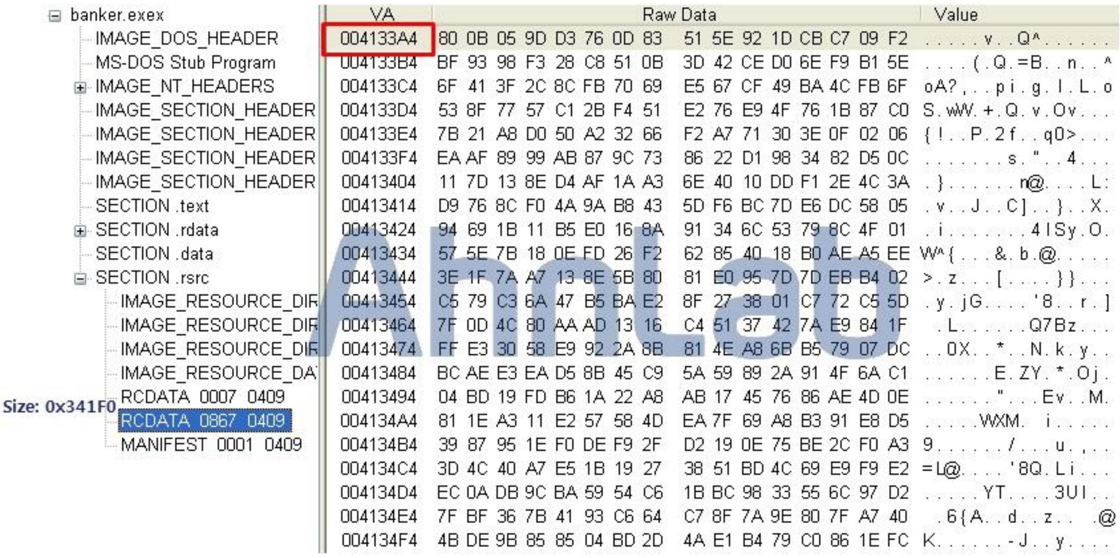
- **A static configuration data (0x56c bytes) is decoded using the Custom Xoring routine**
  Custom XORing routine address: {.text:00414CDA}
- **RC4 key is found in the decrypted static configuration data**
- **md5sum hard-coded in the Citadel agent is input to perform MD5 hashing once**
- **New md5sum is encrypted using the RC4 encryption algorithm; key obtained from the decrypted static configuration data is used**
- **md5sum encrypted using RC4 is used as an AES encryption algorithm SEED**

If a value at an entry of configuration data is required, an AES key is obtained through the process shown above and then only the value is decoded to obtain data. If refreshing is required, the same process is used to decode the entry and plain text is edited and encrypted before being updated on the payload.

[Attachment: Technical – Static Analysis: dropper.exe / PE Image]



File dropper.exe has the binary layout shown in the above diagram. The text section's size (virtual size) is 0xA077 bytes; two thirds of that comprises known libraries (sky blue), user codes (blue) are approximately 0x3500 bytes. Because the sample was compiled using the Microsoft Visual C++ 9.0 engine and programmed using OOP (Object-oriented Programming), actual user code size is estimated to be less than 0x3000 bytes. As described in the above report, this user code is divided into two categories: an image-dropping routine that injects a decoded PE image in child processes and a dropper-delete routine that generates a batch file and deletes file dropper.exe.



Besides the above 2 core routines, the encoded PE image and encrypted 0x400 data are included in the rsrc section of sample.exe. It can be deduced that the PE image that will be the Citadel agent is in the rsrc section as that section is larger (0x356FE) in comparison to the text section. The diagram above shows the exact address (VA) where the encoded PE image is saved and a part of its data.

File layout of the PE image generated by dropper.exe is shown in the above diagram. Unlike general PEs, this binary does not have a resource area and its text section accounts for more than 80% of the binary. It can also be seen that the IAT is contained inside the text section. What is also out of the ordinary is that string data that would normally be located in the data section is included in the text section. This kind of binary layout differs from the PE structure generated by an ordinary compiler. However, such a design facilitates injection of binary data in virtual memory as a PE image (because all required data is in a single text section, target address can be changed according to the status of virtual memory in which the image will be injected). Since the PE image generated by dropper.exe is injected and executed in a process(explorer and it's children), this design proves to be efficient.

The basic EP of the PE image is the last part of the text section. As shown in the above diagram, it is written like a shellcode and located away from user codes. This also demonstrates that the binary data was not generated by the compiler alone.

[Attachment: Technical – 0x400data Decryption Process]

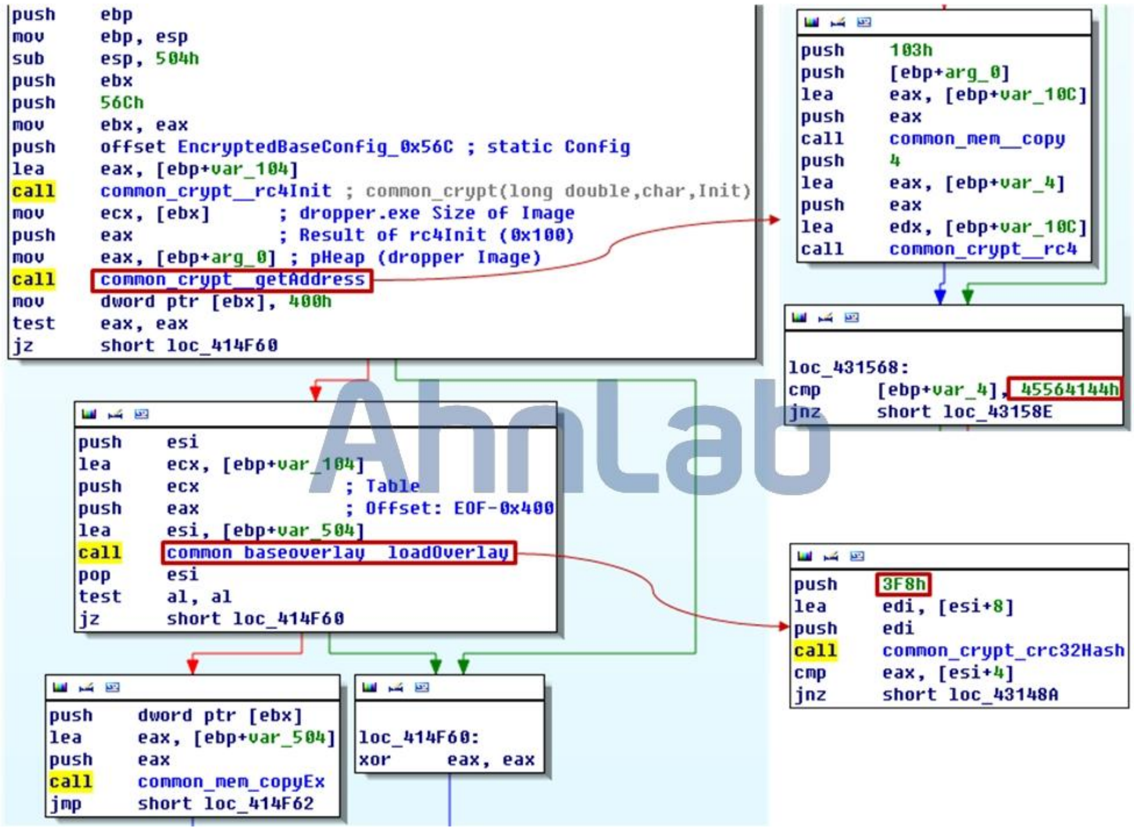
The following data (0x400 bytes) exists behind the file dropper.exe (md5: 97e545aae517a5f816abcd960875ac05).

```
00043600  E3 94 A3 CB BD 22 2C 8E  EF D6 AC E6 1D 66 72 8B  ã▮£Ë½",▮ïÖ¬æ fr▮
00043610  0C DB 62 1F 8B 93 C2 FC  E5 12 1E 1F 95 0D AE A1   Ûb ▮▮Âüå    ▮ ®¡
00043620  7A A9 F6 3D F4 9C 0A 9C  26 3C 4D 49 E0 E9 66 BE  z©ö=ô▮ ▮&<MIàéf¾
00043630  D4 8E 96 AE CA 5D 4C 6F  94 10 0B 4E 77 37 6A 4F  Ô▮▮®Ê]Lo▮  Nw7jO
00043640  BB A7 59 8C 8D 07 95 24  7A 69 8E D9 F5 F6 B4 E2  »§Y▮▮ ▮$zi▮Ùõö´â
00043650  FC F6 B0 02 31 4D B4 5B  92 87 19 B5 C7 40 BD A6  üö° 1M´[’▮ µÇ@½¦
00043660  31 76 18 82 4C 04 BB C5  20 A0 DB A9 89 3A 01 75  1v ▮L »Å  Û©▮: u
00043670  D7 32 3D B5 C9 C6 BF 3E  86 54 D4 B2 72 A1 89 61  ×2=µÉÆ¿>▮TÔ²r¡▮a
00043680  44 29 F3 CC 1C 9B C1 6C  8E FE 48 A9 42 CD 81 C4  D)óÌ ▮Ál▮þH©BÍ▮Ä
00043690  71 B9 D3 E1 6C CD E1 FB  46 98 0D A8 BC 65 2F 30  q¹ÓálÍáûF▮ ¨¼e/0
000436A0  54 69 14 77 BB 3B 35 61  60 94 CF 99 F0 6C 94 05  Ti w»;5a`▮Ï▮ðl▮
000436B0  D2 40 30 D3 A4 EC D4 8B  B0 88 41 37 89 F4 21 96  Ò@0Ó¤ìÔ▮°▮A7▮ô!▮
000436C0  EB 3E AE 6B EC 13 51 ED  BB B3 06 08 32 36 89 9D  ë>®kì Qí»³  26▮▮
000436D0  7B 80 04 0E 6A 4C 19 47  63 47 D1 89 0F 8D F2 0B  {▮  jL GcGÑ▮ ▮ò
000436E0  78 FC 82 13 4B 0D F2 D4  8E 37 8D 01 25 0A CE 97  xü▮ K òÔ▮7▮ % Î▮
000436F0  16 5D 71 BD BD 74 F5 00  5D C1 57 AE E2 EE 8A 1A   ]q½½tõ ]ÁW®âî▮
00043700  DF FB EF 33 E2 3B DF 8C  CE 41 28 99 C7 C5 59 29  ßûï3â;ß▮ÎA(▮ÇÅY)

00043900  81 F4 C7 0F CB 1F 1B 6B  13 ED ED 18 00 E5 B9 AB  ▮ôÇ Ë  k íí  å¹«
00043910  C7 67 C8 91 46 07 17 D3  2C 77 29 18 15 07 08 CB  ÇgÈ‘F  Ó,w)    Ë
00043920  74 33 5C 77 9F 9F CE 6E  64 E7 87 4E C9 F8 70 43  t3\w▮▮Îndç▮NÉøpC
00043930  C0 CB 8B C3 6D 9E 43 88  AA 31 FE 3B 31 4E 4E 5B  ÀË▮Ãm▮C▮ª1þ;1NN[
00043940  AF D7 56 D0 13 E3 6E 93  2A E9 6D F7 1F 48 F1 C5  ¯×VÐ ãn▮*ém÷ HñÅ
00043950  BA 7C 02 1D 1B 15 AD 47  A7 37 C2 CC 54 9E 1C C3  º|    -G§7ÂÌT▮ Ã
00043960  7E ED 9C B5 AD 37 F0 E5  16 93 CA A8 CC 8F 42 2A  ~í▮µ-7ðå ▮Ê¨Ì▮B*
00043970  F2 C5 94 49 3F D9 35 66  70 53 49 3D BD CE FD 04  òÅ▮I?Ù5fpSI=½Îý
00043980  12 BF 58 71 8E 6C 57 32  ED F2 56 E5 4C 8E 50 90   ¿Xq▮lW2íòVåL▮P▮
00043990  96 5D 05 09 ED 9B 53 C6  89 4C B4 8B 85 36 A0 E7  ▮]  í▮SÆ▮L´▮▮6 ç
000439A0  C4 FE 98 F8 D8 00 7F 82  7A DC DA 54 A8 54 8B F8  Äþ▮øØ ▮▮zÜÚT¨T▮ø
000439B0  2E 42 85 1A 8B 09 EE 02  D4 58 BB 27 E1 28 8A D3  .B▮ ▮ î ÔX»'á(▮Ó
000439C0  58 08 F5 25 FB 75 7C 3F  6D 54 C8 3A 3A 9D 58 B9  X õ%ûu|?mTÈ::▮X¹
000439D0  10 42 88 3A 75 CB B6 F6  2B CC CE 0A 54 5F 21 FC   B▮:uË¶ö+ÌÎ T_!ü
000439E0  C7 3D 6A 32 C0 C4 CF 78  2C 65 C6 0C 47 AA A4 8C  Ç=j2ÀÄÏx,eÆ Gª¤▮
000439F0  A1 E1 6A 5C 39 C8 13 97  2B 7B E6 8C BF E8 A2 63  ¡áj\9È ▮+{æ▮¿è¢c
```

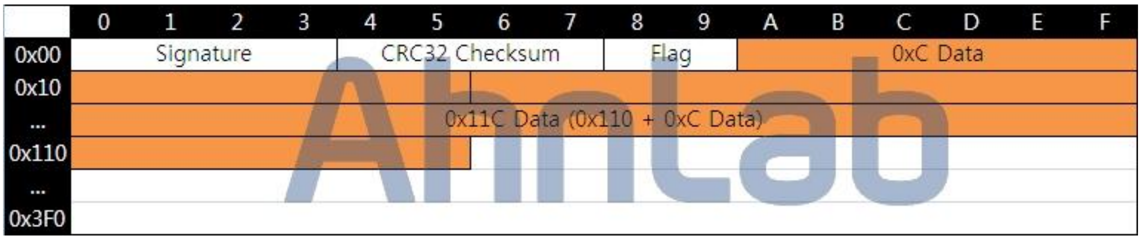
This 0x400 data physically exists in dropper.exe, but the routine that actually uses or updates this data is found in the PE image (installer). The PE image (injector) also references this 0x400 data.

The analysis shows that this 0x400 data is encrypted by the RC4 encryption algorithm and the encryption key is derived from static configuration data (0x56c bytes). As such, when the PE image (installer) is executed, static configuration data is loaded from the inside of the file (text:004064D0) ahead of any other activity and the RC4 key generation process commences (RC4 Init).



RC4 Init generates a key table (0x100 bytes) based on the input seed (static configuration data). The RC4 Init function is located in {.text:0042A7E8}. When key generation is complete, RC4 decryption of the entire dropper.exe image commences. The decryption routine (_rc4) is located in {.text:0042A8D8}.

Decryption result is checked in units of 4 bytes to see if it matches 0x45564144. This data is the first 4 bytes of the decrypted 0x400data (signature of the 0x400 data). In this way, the PE image (installer) determines the starting location of the 0x400 data. What can be inferred here is that the 0x400 data may not always be found at the end of the file. This is because the entire image has been decrypted to locate the 0x400 data.



After the PE image (installer) locates the 0x400 data, an integrity check is performed. The second 4 bytes of the 0x400 data is CRC32 Checksum as can be seen in the above diagram. The data with the Checksum to be checked are Offset 8 to the end of the 0x400 data. The routine that generates CRC32 Checksum can be found in {.text:0042A77E}.

The flag value is checked when the integrity check is complete. The flag of the 0x400 data is 0x000C if dropper.exe has been generated by the Citadel builder. The PE image is designed to function as an installer if this value remains 0x000C and as an injector if the value is changed to 0x011C. Thus it can be deduced that the installer edits this flag in the process of Citadel malware installation to enable the injector to function. The flag value separates the PE image and also represents the size of the data found at the end. If the flag value is 0xC, 0xC worth of data is saved in the heap. If the flag value is 0x11C, 0x11C worth of data is saved in the heap. The exact function of this data could not be identified as the data was not used in the analysis.

[Attachment: Technical – Citadel Dropper: Anti-analysis Method]

• Encryption for Binary Obfuscation

```
mov     esi, offset EncodedDataArray0 ;
                        ; Example)
                        ; Address   Hex dump                                          ASCII
                        ; 0012FE98  55 36 37 6C|33 61 59 57|7A 78 50 6C|58 52 58 51| U67l3aYWzxPlXRXQ
                        ; 0012FEA8  79 51 69 46|73 62 58 73|5A 61 5A 6F|4D 37 71 53| yQiFsbXsZaZoM7qS
                        ; 0012FEB8  44 76 58 38|59 68 4F 44|41 70 72 36|51 6B 50 62| DvX8YhODApr6QkPb
                        ; 0012FEC8  74 4F 7A 39|73 6A 4E 55|64 6F 63 76|41 58 74 00| tOz9sjNUdocvAXt
lea     edi, [ebp+78h+var_94]
rep movsd
mov     esi, offset unk_40DBE8
lea     edi, [ebp+78h+LibFileName]
movsd
movsd
lea     eax, [ebp+78h+LibFileName] ; Setting OutBuf
movsd
mov     [ebp+78h+var_98], eax
lea     eax, [ebp+78h+var_98] ; pOutBuf
push    0Eh             ; PARAM3: Loop Count
push    eax             ; PARAM2: pEncodedData / pOutBuf
lea     ecx, [ebp+78h+var_94] ; PARAM1: SEED (for KeyTable Gen)
movsw
call    CustomDecoding  ; Example)
                        ; [OutBuf]
                        ; Address   Hex dump                                          ASCII
                        ; 0012FF0C  6B 65 72 6E|65 6C 33 32|2E 64 6C 6C|00            kernel32.dll
```

VA of the custom decoding routine of dropper.exe is {.text:00402ad6}. As shown in the above diagram, dropper.exe has the library name and API name it will use in an encoded state and dynamically calls and uses the API needed. The 3 parameters of the decoding routine are as follows.

PARAM1: A value that generates the key table to be used for XORing

PARAM2: Where encoded data (IN) and decoded strings (OUT) are saved

PARAM3: Number of times the loop found inside the routine will be executed (i.e. data size)

This dynamic allocation of APIs to be used by the program is a technique employed to better evade security programs. The function name and library name required for such dynamic allocation are decoded for the same reason. A different seed is used for each set of data to be decoded; the seeds are located in {.rdata:0040d970}.

```
loc_402BFC:             ;
mov     cl, byte ptr [ebp+ecx*4+790h+var_404] ; Load KEY from KeyTable
xor     [edx+ebx], cl   ; Xoring
cdq
push    3
pop     ecx
idiv    ecx
inc     [ebp+790h+var_808] ; offset++
lea     ebx, [eax+2FDh]
mov     eax, [ebp+790h+var_808]
cmp     eax, [ebp+790h+arg_4]
jl      loc_402B95
```

The encoded PE image in the resource area also is decoded using by the same routine. The above diagram shows the core {.text:00402c03} of the decoding routine.

• Indirect API Call Method

Dropper does not call APIs directly; APIs are called indirectly using the CallWindowsProc API. This kind of programming may be due to the characteristics of the Visual C++ 9.0 compiler (Enable obfuscation option), but it is not a normal call method. The following is a call method using CallWindowsProc.

When the API which should be called have zero parameter

```
push    ebx
push    ebp
push    edi
xor     ebx, ebx
push    ebx                 ; PARAM3: Number of Parameter
push    offset ASCII_Comctl32_dll ; PARAM2: "Comctl32.dll" ASCII
mov     ecx, offset ASCII_InitCommonControls ; PARAM1: "InitCommonControls" ASCII
call    FunctionCallMethod
```

When the API which should be called have parameters

```
lea     eax, [ebp+var_438]
push    eax                 ; lpProcessInformation
lea     eax, [ebp+var_490]
push    eax                 ; lpStartupInfo
push    ebx                 ; lpCurrentDirectory
push    ebx                 ; lpEnvironment
push    84h                 ; dwCreationFlags
push    ebx                 ; bInheritHandles
push    ebx                 ; lpThreadAttributes
push    ebx                 ; lpProcessAttributes
push    esi                 ; lpCommandLine
push    ebx                 ; lpApplicationName
push    0Ah                 ; PARAM3: Number of Parameter
mov     ebx, offset ASCII_kernel32_dll
push    ebx                 ; PARAM2: Library Name
mov     ecx, offset ASCII_CreateProcessW ; PARAM1: API Name
call    FunctionCallMethod
```

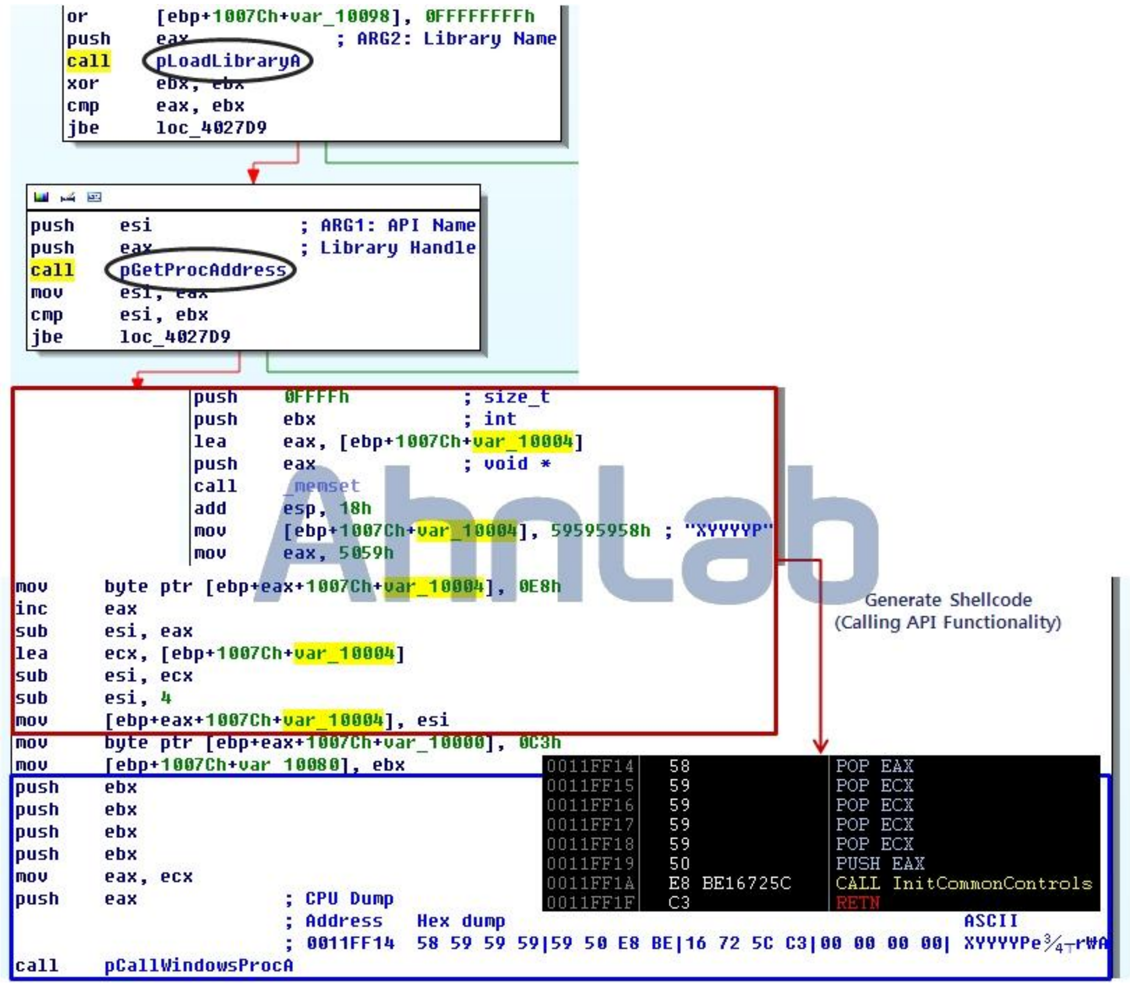
The above diagram shows the two call methods used by the function FunctionCallMehthod, which calls the CallWindowsProc API from the inside. Parameters of this function are as follows and correspond to the red area shown in the above diagram.

PARAM1: API name to call (ASCII)

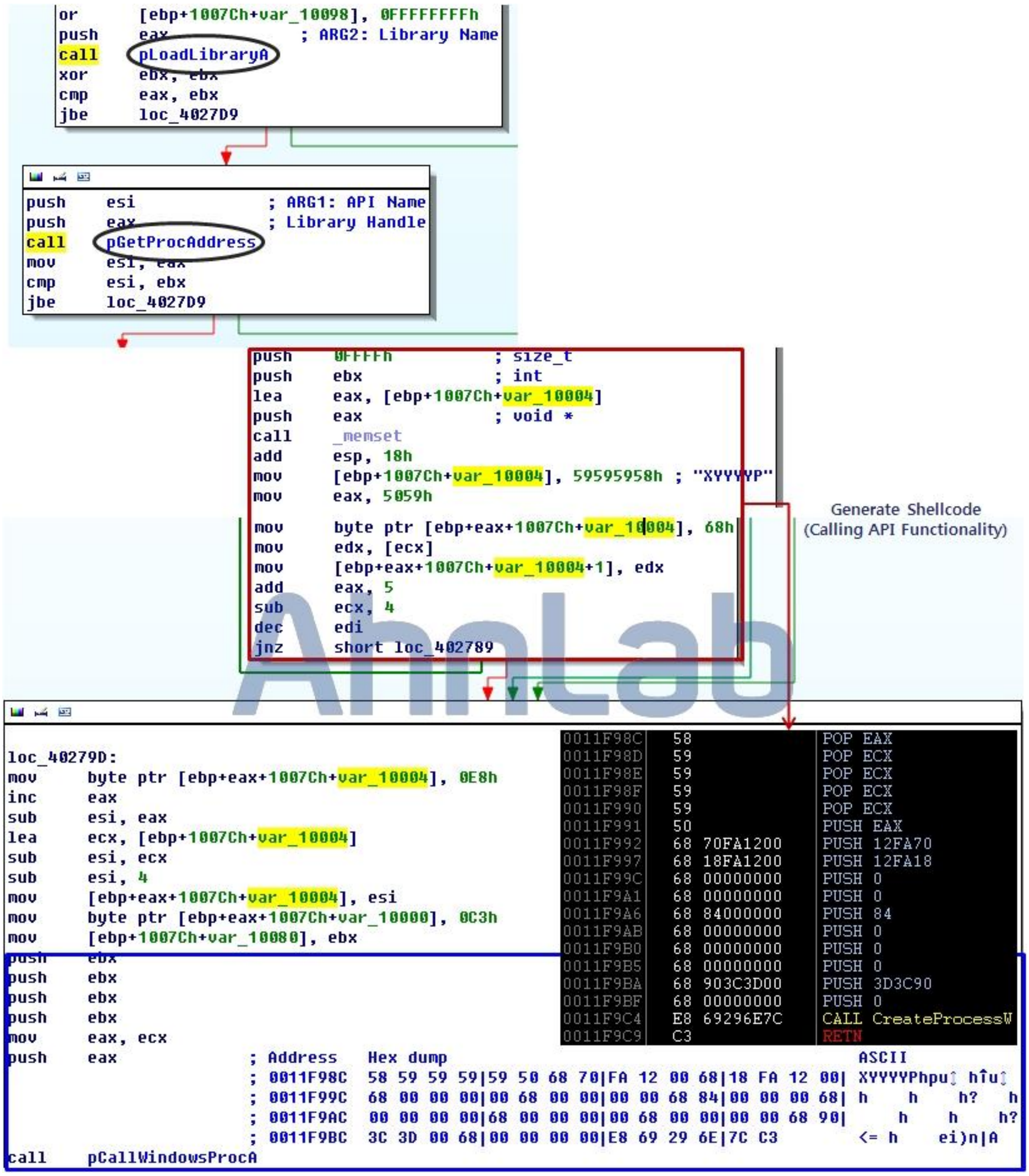
PARAM2: Library name that provides the API to call (ASCII)

PARAM3: Number of parameters the API to call has

If there are parameters to send to the API to call, the required data is saved in the stack (blue area) in advance and the number of data is then inserted in the 3rd parameter.



The above diagram illustrates one of the two call methods through which APIs without parameters are called; core codes of the FunctionCallMethod are displayed. This function obtains the address of the actual API code through the input string (library name, API name) and dynamically writes the code that calls the API in the heap memory. Then, by allocating the code's starting address to the first parameter of the CallWindowsProc API, the desired API is indirect called.

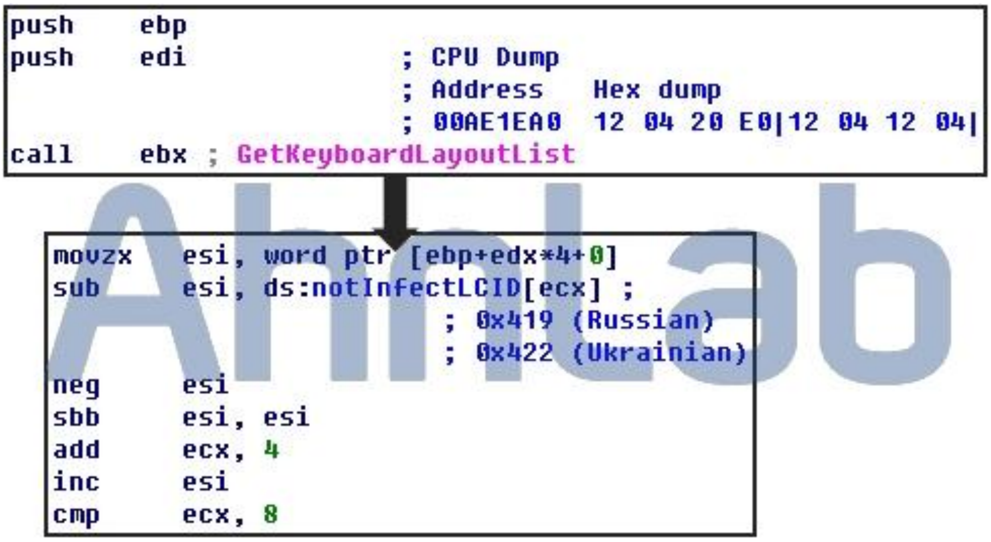


The above diagram shows the core codes of the FunctionCallMethod when APIs with parameters are called. Again, the starting address of the code that calls the API is set in the first parameter of CallWindowsProc, but the code generated in the heap memory is different because the parameters must be handed over.

[Attachment: Technical – Citadel’s Characteristics]

• Infection Area

There are several known characteristics of the Citadel Trojan. One of them is that computers in a certain region, namely Russia and Ukraine, do not get infected as an LCID comparison shows.



The above diagram shows codes that compare the LCID value obtained through the GetKeyboardLayoutList API and the Russian/Ukrainian LCID value. If these LCID values are identical, the terminate routine is executed immediately.

• -z Option: Print Debug Message

The following is another known appearance.

If the PE image (injector) is executed using the -z option, the string ‘Coded by BRIAN KREBS for personal use only. I love my job & wife’ is displayed in the MessageBoxA API. Brian Krebs is the name of a security researcher who researches commercial bots such as Zeus, SpyEye and Citadel.

About AhnLab, inc.

AhnLab develops industry-leading information security solutions and services for consumers, enterprises, and small and medium businesses worldwide. As a leading innovator in the information security arena since 1995, AhnLab's cutting-edge technologies and services meet today's dynamic security requirements, ensure business continuity for our clients, and contribute to a safe computing environment for all

We deliver a comprehensive security lineup, including proven, world-class antivirus products for desktops and servers, mobile security products, online transaction security products, network security appliances, and consulting services.

AhnLab has firmly established its market position and manages sales partners in many countries worldwide.

AhnLab