2016 Updates to Shifu Banking Trojan

researchcenter.paloaltonetworks.com/2017/01/unit42-2016-updates-shifu-banking-trojan/

Dominik Reichel

January 6, 2017

By Dominik Reichel

January 6, 2017 at 12:00 PM

Category: Unit 42

Tags: banking, Shifu, threat research, Trojan

This post is also available in: 日本語 (Japanese)

Overview

Shifu is a Banking Trojan first discovered in 2015. Shifu is based on the Shiz source code which incorporated techniques used by Zeus. Attackers use Shifu to steal credentials for online banking websites around the world, starting in Russia but later including the UK, Italy, and others.

Palo Alto Networks Unit 42 research has found that the Shifu authors have evolved Shifu in 2016. Our research has found that Shifu has incorporated multiple new techniques to infect and evade detection on Microsoft Windows systems. Some of these include:

- Exploitation of CVE-2016-0167 a Microsoft Windows Privilege Escalation vulnerability to gain SYSTEM level privileges. Earlier versions of Shifu exploited CVE-2015-0003 to achieve the same goal
- Use of a Windows atom to identify if the host is already infected with Shifu in addition to the mutex used by previous versions
- Use of “push-calc-ret” API obfuscation to hide function calls from malware analysts
- Use of alternative Namecoin .bit domains

We have also identified new links between Shifu and other tools which suggest Shifu isn’t simply based on the Shiz Trojan, but is probably the latest evolution of Shiz.

The primary goal of this report is to introduce Shifu’s new features to other malware analysts who may encounter this Trojan in the future. The following sections give an overview of the new features, and the appendix at the end includes the technical details on the overall functionality of Shifu.

New Developments and Features in Shifu

The Shifu version discussed in this analysis is comprised of several stages of payloads and was compiled in June 2016. The following image illustrates the different files included in the initial loader which get decrypted after execution:

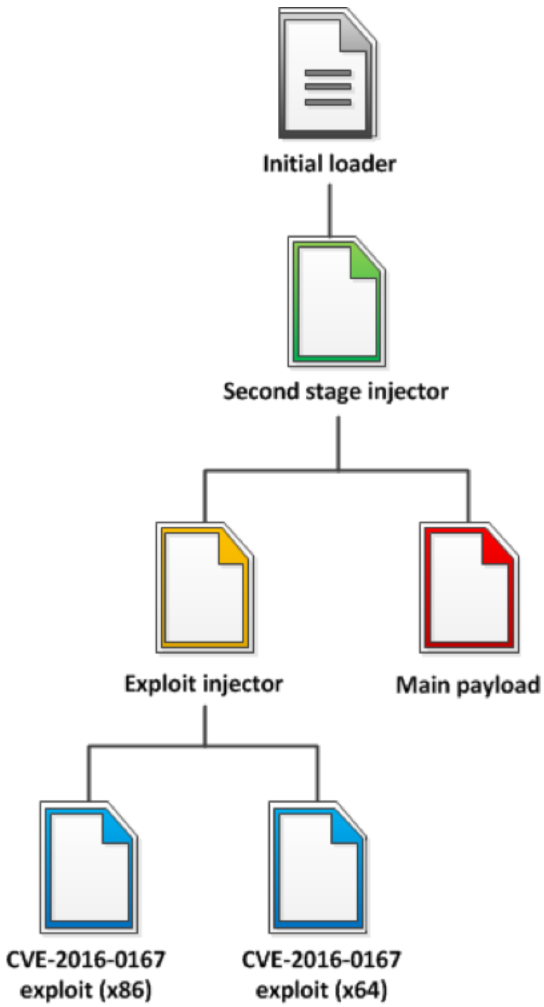


Figure 1. File structure of Shifu

The initial obfuscated loader (x86 exe) contains the encrypted second stage injector (x86 exe). It uses three layers for decryption by subsequently allocating memory via VirtualAlloc() for the next layer. The second stage injector gets decrypted into memory and the original loader process is then overwritten with it. Next, the section flags are adjusted and the IAT addresses are resolved. The final decryption layer then jumps to the entry point of the second stage injector.

The second stage injector contains two exploits for CVE-2016-0167 (x86/x64) that have a compilation time stamp dated February, 2016. At the time of compilation, patches were not yet available for this vulnerability. However, the malware's compilation time stamp dates June 2016. This may indicate the people behind this Shifu version had access to the zero-day exploit at that time or gained access to it afterwards. The exploit uses an interesting technique which makes it possible to just copy the raw disk file into memory. To make the file executable in memory, it uses a custom PE loader shellcode appended to both versions of the exploit as an PE overlay. The shellcode takes care of all the adjustments needed to get a proper executable memory image and executes the exploit. By doing so, the file just needs to be copied into a memory buffer and execution needs to be passed to the shellcode.

We have also found multiple other variants of the exploit, standalone versions (x86/64), but also versions which are embedded in an injector like in Shifu. Additionally, we identified a version of Vawtrak which contains an earlier version of the exploit dating back to November 2015, according to the compilation time stamp. The compilation time stamp of this Vawtrak sample itself dates January 2016 and thus is effectively the first malware known to us to use this exploit.

The second stage injector contains several anti-analysis tricks similar to the previous version. It also contains two command line parameters with functionality that indicate the malware is still in development. Further, the second stage injector uses an atom to check if the system is already infected, instead of using a mutex like most of the malware today. The use of atoms is not a new technique, but still not very widespread.

The main payload is encrypted and packed inside the .tls section of the second stage injector. It first gets decrypted and then unpacked with the aPLib compression library. As persistence method, the main payload copies the initial loader to the AppData folder and creates a Jscript file inside the Startup folder which points to it. The second stage injector injects the main payload inside a x86 instance of svchost and patches its API function calls with an obfuscation technique to make static and dynamic analysis of the malware more difficult.

Compared to the previous version, the main payload contains some updates. This includes the strings to search on the victim's system, the browser target list, and the bot commands. The main payload uses .bit top-level domains to contact its C&C server. The domain names, the user-agent string and the URL parameters are encrypted with a modified RC4 encryption algorithm. The domain names indicate that the attackers may be either located in Ukraine or have a Ukrainian background.

Unfortunately, at the time of the analysis the C&C server didn't respond with any commands and thus further analysis of the targeted financial institutions wasn't possible. This information would be normally downloaded into a configuration file on the victim's disk. For some of its functionality, the main payload hooks some API functions inside the svchost.exe process into which it is injected. Further, it uses the Apache web server for the web injections. If it was successfully downloaded from the C&C server, the malware makes use of a layered service provider to hook into the Winsock API for intercepting and modifying inbound and outbound Internet traffic. It also contains the normally used methods to hook into the browsers networking functions found in many other banking Trojans.

Both the second stage injector and the main payload contain a lot of strings which are never used. This indicates the author(s) were either in a rush to build the malware or the development was done in a sloppy way.

Instead of the string "IntelPowerAgent6" seen in the last version, this sample contains the string "IntelPowerAgent32" which is never used. In addition to the atom created by the second stage injector to check if the system is already infected, the main payload also creates a mutex with a name based on the same procedure to create the name for the atom (see Appendix). However, the mutex uses a hardcoded prefix named "DAN6J0-" before the byte sequence that is also used for the atom string: "{DAN6J0-ae000000d2000000e100}"

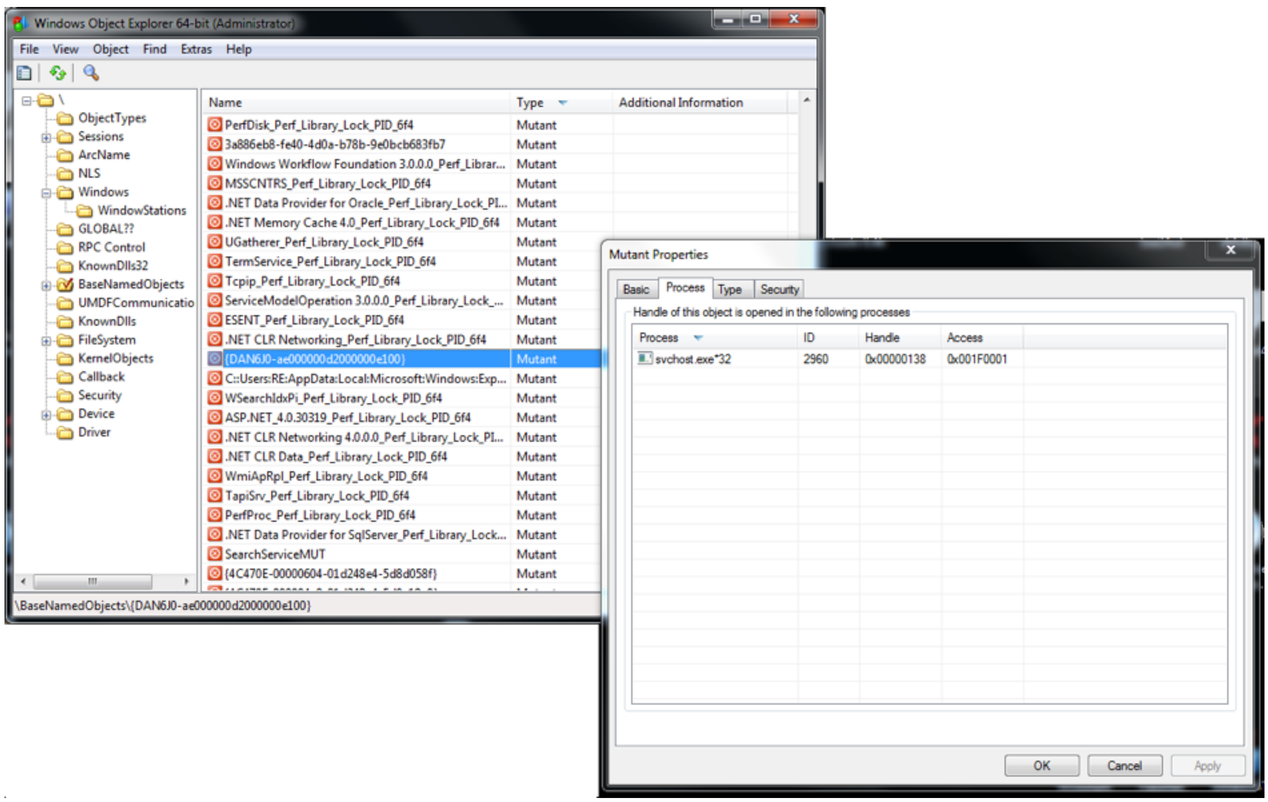


Figure 2. Shifu mutex and the associated svchost process

Shifu, Shiz and Other Related Tools

The Shifu banking Trojan is mainly based on the Shiz/iBank source code, which is one of the oldest banking Trojans still in the wild today. Shiz was first discovered in 2006 and has been through several stages of development since that time. It began as a banking Trojan which only focused on Russian financial institutions. Later, it also began targeting an Italian bank which may have set the stage for a more international focus. The internal versions we have tracked over the last five years ranged from generation 2 to 4 (2011) and 5 (2013/2014). The fifth generation of Shiz was the last one we saw in the wild in 2014 (last internal version was 5.6.25) and it differs from the 4th generation in the coding style. It looks like it was developed by another coder, which could indicate the source code was sold or shared. The query string used to contact the C&C server of one of the very first versions of the fifth generation supports our theory:

botid=%s&ver=5.0.1&up=%u&os=%03u&ltime=%s%d&token=%d&cn=reborn&av=%s

We can see that the campaign name (cn) contains the string "reborn".

Shifu was first discovered in the wild in the middle of 2015 and we believe it's the evolution of the 5th generation of Shiz with a more international focus.

We have not only tracked the Shiz banking Trojan over the last couple of years, but also found several additional malware tools allegedly from the same author(s). Collected samples indicate the author(s) have developed a whole set of financially related malware. It’s not clear if the author works as part of a group or uses the malware themselves. These tools are mainly based on the source code of the fifth generation of Shiz.

We have connected these tools together because they all contain a PDB path that has the same root folder:

Z:\coding\...

Furthermore, most of the tools are based on the Shiz source code, because the coding style and used API functions are very similar. Also, comparing the code between the tools with BinDiff shows a high degree of similarity. Moreover, those tools with network functionality contain query strings similar to the one in Shiz to contact their C&C server.

As our colleagues from FireEye described last year, the PDB path found in Shifu is as follows:

Z:\coding\project\main\payload\payload.x86.pdb

Other tools we have identified have the following PDB paths and are likely from the same author(s):

Z:\coding\cryptor\Release\crypted.pdb

Z:\coding\malware\tests\Release\cryptoshit.pdb

Z:\coding\malware\RDP\output\Release\rdp_bot.pdb

Z:\coding\malware\ScanBot\Release\bot.pdb

The malware internally named "cryptor" contains an encrypted sample of BifitAgent, the first malware known to attack the financial software from BIFIT. While it's possible that BifitAgent is developed from the same person, we haven't found any indications for that. According to the compilation time stamps, most of the samples were created in October/November 2013.

The malware with the name "rdp_bot" is a small bot which uses the RDP protocol to gain full access to a computer. It uses the same modified RC4 encryption algorithm as the Shifu version discussed in this article. This tool was probably used along the Shiz banking Trojan, because the attacker is able to do his fraudulent activities directly from the victim’s computer. By doing so, one could fool bank antifraud systems which check for the IP address, browser footprints or keyboard layouts. The tool is based on the research about RDP performed by Alisa Esage. The samples date from June to November 2013.

The tool which is named "cryptoshit" contains an encrypted sample of rdp_bot and also uses the same modified RC4 algorithm as the Shifu version described here. The samples date September/October 2013 and January 2014 according to the compilation time stamp.

The malware with the internal name "ScanBot" is a small backdoor which uses the Super Light Regular Expression library (SRLE) for scanning a victim’s computer for files via commands from its operator. The samples date June 2013 according to the time stamp.

Protection Against Shifu

Palo Alto Networks customers are protected from Shifu in the following ways:

- Wildfire classifies Shifu files as malicious and signatures are loaded into Threat Prevention
- AutoFocus customers can track malware using the Shifu tag
- Command and Control domains used by Shifu are blocked through Threat Prevention

SHA256 Hashes of Samples Discussed

Initial obfuscated loader

d3f9c4037f8b4d24f2baff1e0940d2bf238032f9343d06478b5034d0981b2cd9
368b23e6d9ec7843e537e9d6547777088cf36581076599d04846287a9162652b
e7e154c65417f5594a8b4602db601ac39156b5758889f708dac7258e415d4a18
f63ec1e5752eb8b9a07104f42392eebf143617708bfdd0fe31cbf00ef12383f9

Second stage injector

003965bd25acb7e8c6e16de4f387ff9518db7bcca845502d23b6505d8d3cec01
1188c5c9f04658bef20162f3001d9b89f69c93bf5343a1f849974daf6284a650

Exploit injector

e7c1523d93154462ed9e15e84d3af01abe827aa6dd0082bc90fc8b58989e9a9a

CVE-2016-0167 exploit (x86)

5124f4fec24acb2c83f26d1e70d7c525daac6c9fb6e2262ed1c1c52c88636bad

CVE-2016-0167 exploit (x64)

f3c2d4090f6f563928e9a9ec86bf0f1c6ee49cdc110b7368db8905781a9a966e

Main payload

e9bd4375f9b0b95f385191895edf81c8eadfb3964204bbbe48f7700fc746e4dc
5ca2a9de65c998b0d0a0a01b4aa103a9410d76ab86c75d7b968984be53e279b6

Appendix - Technical details

Second Stage Injector Analysis

The second stage injector contains an exploit injector (x86 DLL) which in turn has two embedded exploits (x86/64 DLL) for CVE-2016-0167. The second stage injector also contains the encrypted and aPLib packed main payload module (x86 DLL) in its .tls section. For decryption, it uses a modified version of the RC4 encryption algorithm with a salt that is stored in the .rsrc section. Significant strings in the second stage injector's .data section were XORed with the key 0x8D and get decrypted on-the-fly. Decrypted strings:

```
AddMandatoryAce
ADVAPI
Advapi32.dlladvapi32.dllws2_32.dll
WPUCloseEvent
WPUCloseSocketHandleWPUCreateEvent
WPUCreateSocketHandle
WPUFDIsSet
WPUGetProviderPath
WPUModifyIFSHandle
WPUPostMessage
WPUQueryBlockingCallbackWPUQuerySocketHandleContext
WPUQueueApc
WPUResetEvent
WPUSetEvent
WPUOpenCurrentThreadWPUCloseThread
WSPStartup
> %1\r\ndel %0
software\\microsoft\\windows\\currentversion\\run
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/echo
rundll32.exe shell32.dll, ShellExec_RunDLL %s
Microsoft\\Microsoft AntimalwareSoftware\\Coranti
Software\\risingSoftware\\TrendMicroSoftware\\Symantec
Software\\ComodoGroup
Software\\Network Associates\\TVD
Software\\Data Fellows\\F-SecureSoftware\\Eset\\Nod
Software\\Softed\\ViGUARD
Software\\Zone Labs\\ZoneAlarm
Software\\Avg
Software\\VBA32
Software\\Doctor WebSoftware\\G DataSoftware\\Avira
Software\\AVAST Software\\Avast
Software\\KasperskyLab\\protected
Software\\Bitdefender
Software\\Panda SoftwareSoftware\\Sophos.bat\\\\.\\%C:
|$$$}rstuvwxyz{$$$$$$$>?@ABCDEFGHIJKLMNOPQRSTUVW$$$$$$XYZ[\\]^_`abcdefghijklmnopq
conhost
CreateProcessInternalW
ConvertStringSecurityDescriptorToSecurityDescriptorWContent-Type: multipart/form-data; boundary=--------------------------%s\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Host: %s\r\n%d.%d.%d.%d
%d.%d.%d.%d.%x
%temp%\\debug_file.txt
[%u][%s:%s:%u][0x%x;0x%x] %sDnsFlushResolverCache
\\*.*
dnsapi.dll
DnsGetCacheDataTable.dll.exedownload.windowsupdate.com
vk.com
yandex.ru
HTTP/1.1https://http://%s
IsWow64Process
kernel
kernel32.dllLdrGetProcedureAddress
Microsoft
NtAllocateVirtualMemory
CLOSED
LAST_ACKTIME_WAIT
```

```
57  DELETE_TCB
58  LISTEN
59  SYN_SENTSYN_RCVDESTAB
60  FIN_WAIT1
61  FIN_WAIT2
62  CLOSE_WAIT
63  CLOSING
64  TCP\t%s:%d\t%s:%d\t%s\n
65  netstat\nProto\tLocal address\tRemote address\tState\n
66  ntdll.dll
67  NtResumeProcess
68  NtSuspendProcess\\\\?\\globalroot\\systemroot\\system32\\drivers\\null.sys
69  NtWriteVirtualMemoryopenRegisterApplicationRestart
70  RtlCreateUserThread
71  ResetSR
72  RtlComputeCrc32
73  rundll32SeDebugPrivilegeSystemDrive
74  \\StringFileInfo\\%04x%04x\\ProductName
75  software\\microsoft\\windows nt\\currentversion\\winlogon
76  shell
77  Sleep
78  srclient.dllSeShutdownPrivilege
79  \"%s\"
80  %d\t%s\ntaskmgr\nPID\tProcess name\nnet user\n
81  the computer is joined to a domain\n..
82  \\VarFileInfo\\Translation
83  %windir%\\system32\\%windir%\\syswow64\\POST*.exe
84  %SystemDrive%\\
85  *SYSTEM*%02x%s:Zone.Identifier
86  GetProcessUserModeExceptionPolicy
87  SetProcessUserModeExceptionPolicy
88  %ws\\%ws\n
89  WORKGROUP
90  HOMESoftware\\Microsoft\\Windows\\CurrentVersion\\Policies\\ExplorerDisableCurrentUserRun
91  %s.dat
92  software\\microsoft\\windows%OS%_%NUMBER_OF_PROCESSORS%
93  S:(ML;;NRNWNX;;;LW)D:(A;;GA;;;WD)
94  S:(ML;;NRNWNX;;;LW)D:(A;;GA;;;WD)(A;;GA;;;AC)
95  \\\\.\\AVGIDSShim
96  FFD3\\\\.\\NPF_NdisWanIpc:\\sample\\pos.exe
97  ANALYSERS
98  SANDBOX
99  VIRUS
100 MALWARE
101 FORTINETMALNETVMc:\\analysis\\sandboxstarter.exec:\\analysisc:\\insidetmc:\\windows\\system32\\drivers\\vmmouse.sys
102 c:\\windows\\system32\\drivers\\vmhgfs.sys
103 c:\\windows\\system32\\drivers\\vboxmouse.sys
104 c:\\iDEFENSEc:\\popupkiller.exe
105 c:\\tools\\execute.exe
106 c:\\Perlc:\\Python27api_log.dll
107 dir_watch.dll
108 pstorec.dll
109 dbghelp.dll
110 Process32NextW
111 Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\Zones\\3
112 1406.bitMiniDumpWriteDump
113 \r\nReferer: %s\r\n
114 \\Google\\Chrome\\User Data\\Default\\Cache
115 var %s = new ActiveXObject("WScript.Shell"); %s.Run("%s");
116 IntelPowerAgent32
117 %OS%_%NUMBER_OF_PROCESSORS%
118 %s\cmd.exe
119 ComSpec
120 ConsoleWindowClass
121 .exekernel32.dllntdll.dll
122 ZwQuerySystemInformationZwAllocateVirtualMemory
123 PsLookupProcessByProcessId
124 PsReferencePrimaryToken
125 Class
126 Window
127 open "%s" -q%windir%\\system32\\sdbinst.exe
128 /c "start "" "%s" -d"
129 %windir%\\system32\\sndvol.exe
130 "%s" -u /c "%s\\SysWOW64\\SysSndVol.exe /c "start "" "%s" -d""
131 %temp%\\%u
132 %u.tmp
133 Wow64DisableWow64FsRedirection
134 Wow64RevertWow64FsRedirection
```

```
runas.exe
%systemroot%\\system32\\svchost.exe
%systemroot%\\system32\\wscript.exe
snxhk.dll
sbiedll.dll
/c start "" "%s" " "
cmd.exe
runas
--crypt-test
It work's!
--vm-test
```

Exploit Injector with Embedded CVE-2016-0167 Exploits

The exploit injector is used to gain SYSTEM privileges on the infected host. The injector contains the actual exploits for both x86 and x64 systems. The magic PE bytes ("MZ") at the beginning of the files are patched will null bytes to prevent them from automatic extraction.

The second stage injector checks for the current process' integrity level and the OS version. If the integrity level of the process is low and the OS version is 6.1 (Windows 7 / Windows Server 2008 R2), the second stage injector writes the exploit injector file into memory. Then, it searches for the magic value 0x99999999 in the exploit injector which marks the beginning of the PE overlay. When the address was found, 12 bytes are added and the second stage injector jumps to this address which is in fact a custom PE loader shellcode. The call to the shellcode looks as follows:

```
00401EF5   pusha
00401EF6   add esi, 0Ch
00401EF9   call esi   -> PE loader shellcode in overlay
00401EFB   popa
```

Custom PE loader shellcode

It first gets the end of the shellcode which is then used to scan the exploit injector file for the magic PE number ("MZ"). The code to get end of the shellcode looks as follows:

```
00077174   jmp short 00077178
00077176   pop eax
00077177   retn
00077178   call 00077176
```

Next, a custom GetProcAddress() function is used together with a hashing function to find the address of VirtualAllocEx(). Then, VirtualAllocEx() is called to allocate a memory buffer of with full access rights into which the exploit injectors sections are written with the appropriate memory alignments. The necessary memory addresses are then adjusted with help of the relocation information, the API function addresses are resolved and the IAT is filled. Finally, the shellcode jumps to the DLL entry point of the freshly created exploit injector module.

Exploit injector

At first, the strings "kernel32.dll", "LoadLibrary" and "GetProcAddress" are created. Next, the image base address for kernel32.dll is searched and the addresses of LoadLibrary() and GetProcAddress() are obtained. With help of these API functions, the IAT addresses of the exploit injector get resolved and the IAT is filled. The purpose of this function is unclear, as it was already done by the second stage injector. Thereafter, a new thread gets created with API function CreateThread().

The thread first calls IsWow64Process() and according to the result either the embedded x86 or x64 version of the exploit file is written into a memory buffer. Next, the PE magic value ("MZ") is written to the beginning of the exploit file. Then, an event named "WaitEventX" is created which is later used by the exploit. Then, the main exploit loading function is called.

The exploit loading function searches for the following process names and if found also the module names for the following strings which are part of Trend Micro security software:

- "uiSeAgnt.exe"
- "PtSessionAgent.exe"
- "PwmSvc.exe"
- "coreServiceShell.exe"

If one of the processes is found, a suspended process of wuauclt.exe is created. Otherwise, a suspended process of svchost.exe is created. In both cases, the command line argument "-k netsvcs" is passed, but can be only used by svchost.exe. It should be noted that this functionality always fails if the x64 version of Trend Micro Internet Security is installed. The code (x86) calls CreateToolhelp32Snapshot() on a x64 process which results in an error (ERROR_PARTIAL_COPY). Moreover, it also fails because the code tries to access a protected Trend Micro process (ERROR_ACCESS_DENIED).

Next, it maps the x86 or x64 file of the exploit into memory with CreateFileMapping() and MapViewOfFile() and fills in the memory with the exploit bytes. Finally, the section gets mapped into the suspended process of svchost.exe or wuauclt.exe by using ZwMapViewOfSection(). It then checks the OS version if it is 5.2 (Windows Server 2003 / Windows XP 64-Bit Edition) and exits the function if so. Afterwards, two memory buffers are created and a shellcode is written to each of them. The first obfuscated shellcode calls the second shellcode, which is a stager for the mapped exploit file. Next, it calls ResumeThread() to execute the suspended process so the exploit is executed.

The second stage injector verifies that the exploit was successful by checking if the integrity level of itself is still SECURITY_MANDATORY_LOW_RID. If not, the exploit successfully elevated privileges to SECURITY_MANDATORY_SYSTEM_RID and continues with the injection of the main payload. If the exploit failed, it tries to execute itself under the SYSTEM user account with help of the Windows command line (cmd.exe) and runas.exe tool.

Atom String Building

Instead of using a mutex like most of today's malware, the second stage injector creates an atom and checks the global atom table to see if an instance of Shifu is already running.

At first, it uses the template string "%OS%_%NUMBER_OF_PROCESSORS%" for the API ExpandEnvironmentStrings() to get the Windows version and number of processors. For example, in Windows 7 with one processor the result would be "Windows_NT_1". This string is then used to calculate four CRC32 hashes with RtlComputeCrc32() and the following initial values:

- 0xFFFFFFFF
- 0xEEEEEEEE
- 0xAAAAAAAA
- 0x77777777

The resulting CRC hashes of the string "Windows_NT_1" are as follows:

- 0x395693AE
- 0xB24495D2
- 0xF39F86E1
- 0xBAE0B5C8

Next, the last byte of each CRC hash is stored as a DWORD value on the stack:

- 0xAE000000 (from 0x395693AE)
- 0xD2000000 (from 0xB24495D2)
- 0xE1000000 (from 0xF39F86E1)
- 0xC8000000 (from 0xBAE0B5C8)

The stack with the hash byte sequence looks as follows:

AE 00 00 00 D2 00 00 00 E1 00 00 00 C8 00 00 00

The atom string is then created by converting first 8 bytes of the hash byte sequence to ASCII characters with snprintf() function. The result in this case would be:

"ae000000d2000000"

At last, it calls GlobalFindAtom() API to check if the atom is present and calls GlobalAddAtom() if not.

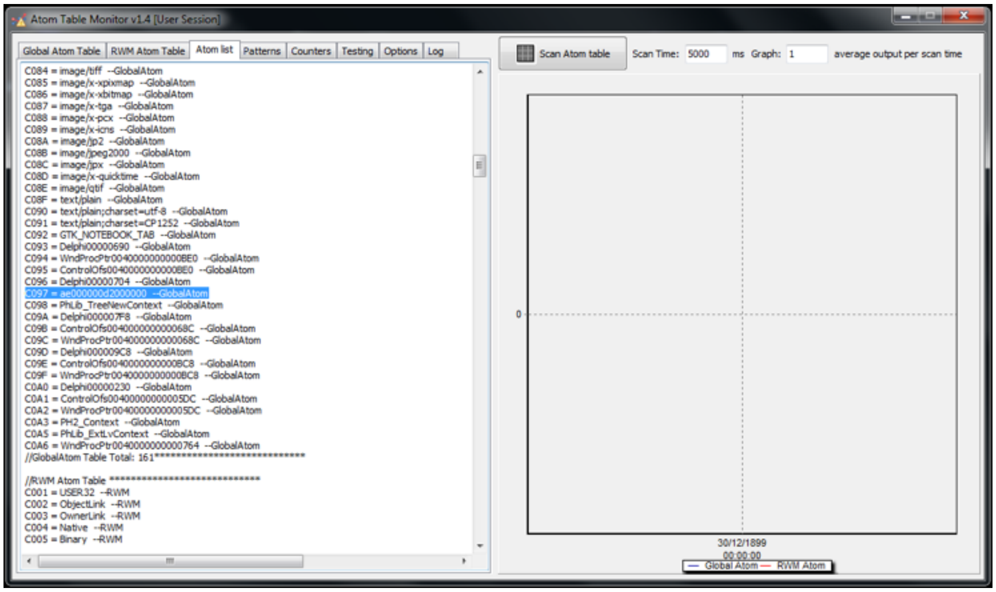


Figure 3. Shifu atom in the global atom table

Command Line Arguments

The second stage injector has two command line parameters of which only one has a functionality. They may be used for an upcoming feature or were just forgotten to be removed.

> --crypt-test

Shows just a message box with the text "It work's!"

> --vm-test

No functionality

Anti-Analysis Tricks

Anti Sandboxie / Avast

Shifu checks if the module snxhk.dll (Avast) or sbiedll.dll (Sandboxie) is present in its own process space by calling GetModuleHandleA() and runs an infinite Sleep() loop if a handle is returned.

All the following anti analysis tricks are only used if Shifu is executed on a 32-bit Windows machine (no Wow64 process).

Process name detection

It enumerates running process names, converts them to lowercase, calculates the CRC32 hashes of those names and compares to the following list:

- 0x99DD4432 - ?
- 0x1F413C1F - vmwaretray.exe
- 0x6D3323D9 - vmusrvc.exe
- 0x3BFFF885 - vmsrvc.exe
- 0x64340DCE - ?
- 0x63C54474 - vboxtray.exe
- 0x2B05B17D - ?
- 0xF725433E - ?
- 0x77AE10F7 - ?
- 0xCE7D304E - dumpcap.exe
- 0xAF2015F2 - ollydbg.exe
- 0x31FD677C - importrec.exe
- 0x6E9AD238 - petools.exe
- 0xE90ACC42 - idag.exe
- 0x4231F0AD - sysanalyzer.exe
- 0xD20981E0 - sniff_hit.exe
- 0xCCEA165E - scktool.exe
- 0xFCA978AC - proc_analyzer.exe

- 0x46FA37FB - hookexplorer.exe
- 0xEEBF618A - multi_pot.exe
- 0x06AAAE60 - idaq.exe
- 0x5BA9B1FE - procmon.exe
- 0x3CE2BEF3 - regmon.exe
- 0xA945E459 - procexp.exe
- 0x877A154B - peid.exe
- 0x33495995 - autoruns.exe
- 0x68684B33 - autorunsc.exe
- 0xB4364A7A - ?
- 0x9305F80D - imul.exe
- 0xC4AAED42 - emul.exe
- 0x14078D5B - apispy.exe
- 0x7E3DF4F6 - ?
- 0xD3B48D5B - hookanaapp.exe
- 0x332FD095 - fortitracer.exe
- 0x2D6A6921 - ?
- 0x2AAA273B - joeboxserver.exe
- 0x777BE06C - joeboxcontrol.exe
- 0x954B35E8 - ?
- 0x870E13A2 - ?

File detection

Shifu checks if the following files or folders exist on the system and runs an infinite Sleep() loop if so:

- c:\sample\pos.exe
- c:\analysis\sandboxstarter.exe
- c:\analysis
- c:\insidetm
- c:\windows\system32\drivers\vmmouse.sys
- c:\windows\system32\drivers\vmhgfs.sys
- c:\windows\system32\drivers\vboxmouse.sys
- c:\iDEFENSE
- c:\popupkiller.exe
- c:\tools\execute.exe
- c:\Perl
- c:\Python27

Debugger detection

It checks if it’s being debugged by calling IsDebuggerPresent(). Also, it calls ZwQueryInformationSystem() with ProcessDebugPort and ProcessDebugObjectHandle to check for a debugger presence. If a debugger is detected it runs an infinite Sleep() loop.

Wireshark detection

Shifu attempts to open \\.\NPF_NdisWanIp with CreateFile() and will enter an infinite Sleep() loop if it is successful.

Self-sanity checks

It checks its own file name length if it is longer than 30 characters and runs an infinite Sleep() loop if so. Also, it checks if its own process name CRC32 hash matches one of the following:

- 0xE84126B8 - sample.exe
- 0x0A84E285 - ?
- 0x3C164BED - ?
- 0xC19DADCE - ?
- 0xA07ACEDD - ?
- 0xD254F323 - ?
- 0xF3C4E556 - ?
- 0xF8782263 - ?
- 0xCA96016D - ?

Furthermore, it checks if one of the following modules from GFI Sandbox is present in its own process address space:

- api_log.dll
- dir_watch.dll

- pstorec.dll

Unknown anti-analysis trick

Shifu uses an anti-analysis trick whose purpose is unknown to us. It retrieves the address of Process32NextW() and compares the first 5 bytes with the sequence 0x33C0C20800 which disassembles to:

```
33C0  XOR EAX,EAX
C2 0800   RETN 8
```

This code is only present in 32-bit Windows XP and not in later Windows versions, because the Unicode version of that function probably wasn't implemented yet. If the code sequence is found meaning that Shifu was executed on 32-bit Windows XP, it runs an infinite Sleep() loop.

Windows domain name check

It checks if the computer workgroup name is either "WORKGROUP" or "HOME" with API functions NetServerGetInfo() and NetWkstaGetInfo() and runs an infinite Sleep() loop otherwise. Next, it checks for the name "ANALYSERS" and runs the infinite loop if found.

Computer and user name check

Shifu gets the computer and user name with GetComputerName() and GetUserName() to check for the following strings:

- SANDBOX
- FORTINET
- VIRUS
- MALWARE
- MALNETVM

If one is found it runs an infinite loop.

Process termination feature

Second stage injector of Shifu enumerates all running processes, converts every name to lower case, calculates the CRC32 hash of it and compares it to the following ones:

- 0xD2EFC6C4 - python.exe
- 0xE185BD8C - pythonw.exe
- 0xDE1BACD2 - perl.exe
- 0xF2EAA55E - autoit3.exe
- 0xB8BED542 - ?

If one matches, it first tries to terminate the process with OpenProcess() and TerminateProcess(). If that fails, it tries to close the main window handle of the process if it is flagged as HANDLE_FLAG_PROTECT_FROM_CLOSE with ZwClose(). Then, it opens the process with full access rights and unmaps it from memory with ZwUnmapViewOfSection(). At last, the main window handle of the unmapped process is closed.

Main Payload Decryption, Unpacking and Injection

To decrypt the main payload, the second stage injector retrieves a salt needed for the decryption algorithm from its .rsrc section. It uses a modified RC4 algorithm where the salt is used to XOR the array of 256 bytes byte after byte at the beginning. The encrypted array is then used to decrypt the main payload located in the .tls section. The decrypted main payload is additionally packed with the aPLib compression library.

If the initial loader runs as a medium or high integrity level process, the routine which calculates the atom string name is called again. This time, only the first 4 bytes are used to build a string, for example "ae000000". Next, the CRC32 hash of this string is calculated and used to XOR another array of 256 bytes starting from 0x0 to 0xFF. This encrypted array is then used to again encrypt the decrypted main payload. The resulting encrypted data are written to registry for persistence purposes under the key "HKCU\software\microsoft\windows" with a random CRC32 hash name, for example "f4e64d63". Also, a second value with the string "ae000000" as name is created and filled up with null bytes and the path of the initial loader, for example "C:\ProgramData\7d5d6044.exe". At last, the temporarily encrypted main payload gets decrypted again.

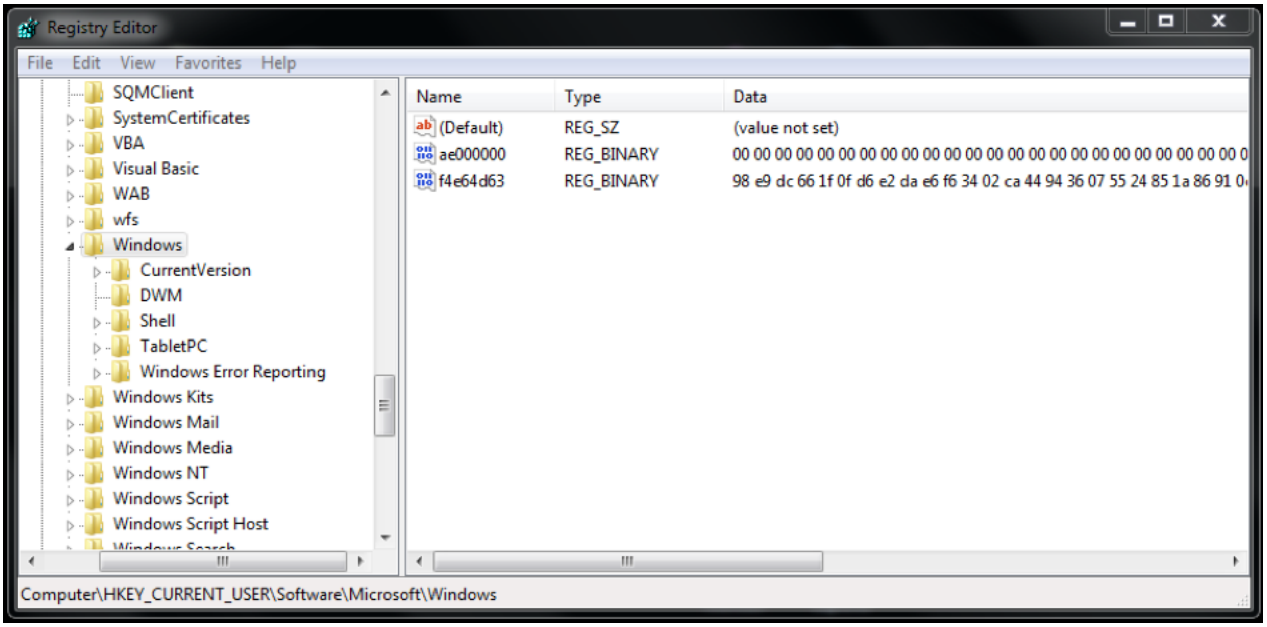


Figure 4. Encrypted main payload and initial loader path stored in the Windows registry

Next, the main payload gets unpacked into memory. Thereafter, a suspended svchost.exe process (x86) is created with the same integrity level as the parent process. The main payload gets mapped into the process and the magic PE value (MZ) patched. The svchost process gets then resumed so the main payload is executed. At last, a batch file is created and executed in the %TEMP% folder. It overwrites the original executed initial loader with a random number of bytes to cover the tracks. The random bytes are always followed by a space character and the CR LF control characters.

Main Payload Analysis

The main payload module's IAT function names were XORed with the key 0xFF to make static analysis more difficult. Significant strings in the .data section are also XORed with the key 0x8D and get decrypted on-the-fly. Decrypted strings:

```
AddMandatoryAce
ADVAPI
Advapi32.dlladvapi32.dllws2_32.dll
WPUCloseEvent
WPUCloseSocketHandleWPUCreateEvent
WPUCreateSocketHandle
WPUFDIsSet
WPUGetProviderPath
WPUModifyIFSHandle
WPUPostMessage
WPUQueryBlockingCallbackWPUQuerySocketHandleContext
WPUQueueApc
WPUResetEvent
WPUSetEvent
WPUOpenCurrentThreadWPUCloseThread
WSPStartup
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/echo
> %1\r\ndel %0
rundll32.exe shell32.dll, ShellExec_RunDLL %s
software\\microsoft\\windows\\currentversion\\run
Microsoft\\Microsoft AntimalwareSoftware\\Coranti
Software\\risingSoftware\\TrendMicroSoftware\\Symantec
Software\\ComodoGroup
Software\\Network Associates\\TVD
Software\\Data Fellows\\F-SecureSoftware\\Eset\\Nod
Software\\Softed\\ViGUARD
Software\\Zone Labs\\ZoneAlarm
Software\\Avg
Software\\VBA32
Software\\Doctor WebSoftware\\G DataSoftware\\Avira
Software\\AVAST Software\\Avast
Software\\KasperskyLab\\protected
Software\\Bitdefender
Software\\Panda SoftwareSoftware\\Sophos.bat|$$$}rstuvwxyz{$$$$$$$>?
@ABCDEFGHIJKLMNOPQRSTUVW$$$$$$XYZ[\\]^_`abcdefghijklmnop
q
```

```
\\\\.\\%C:
conhost
CreateProcessInternalW
ConvertStringSecurityDescriptorToSecurityDescriptorWContent-Type: application/x-www-form-urlencoded\r\n
Content-Type: multipart/form-data; boundary=---------------------------%s\r\n
Host: %s\r\n%d.%d.%d.%d
%d.%d.%d.%d.%x
%temp%\\debug_file.txt
[%u][%s:%s:%u][0x%x;0x%x] %sDnsFlushResolverCache
\\*.*
dnsapi.dll
DnsGetCacheDataTable.dll.exedownload.windowsupdate.com
vk.com
yandex.ru
HTTP/1.1https://http://%s
IsWow64Process
kernel
kernel32.dllLdrGetProcedureAddress
Microsoft
NtAllocateVirtualMemory
CLOSED
LAST_ACKTIME_WAIT
DELETE_TCB
LISTEN
SYN_SENTSYN_RCVDESTAB
FIN_WAIT1
FIN_WAIT2
CLOSE_WAIT
CLOSING
TCP\t%s:%d\t%s:%d\t%s\n
netstat\nProto\tLocal address\tRemote address\tState\n
ntdll.dll
NtResumeProcess
NtSuspendProcess\\\\?\\globalroot\\systemroot\\system32\\drivers\\null.sys
NtWriteVirtualMemoryopenRegisterApplicationRestart
RtlCreateUserThread
ResetSR
RtlComputeCrc32
rundll32SeDebugPrivilegeSystemDrive
\\StringFileInfo\\%04x%04x\\ProductName
software\\microsoft\\windows nt\\currentversion\\winlogon
shell
Sleep
srclient.dllSeShutdownPrivilege
\"%s\"
%d\t%s\ntaskmgr\nPID\tProcess name\nnet user\n
the computer is joined to a domain\n..
\\VarFileInfo\\Translation
%windir%\\system32\\%windir%\\syswow64\\POST*.exe
%SystemDrive%\\
*SYSTEM*%02x%s:Zone.Identifier
GetProcessUserModeExceptionPolicy
SetProcessUserModeExceptionPolicy
%ws\\%ws\n
WORKGROUP
HOMEsoftware\\microsoft\\windowsSoftware\\Microsoft\\Windows\\CurrentVersion\\Policies\\ExplorerDisableCurrentUserRun
%s.dat
%OS%_%NUMBER_OF_PROCESSORS%
S:(ML;;NRNWNX;;;LW)D:(A;;GA;;;WD)
S:(ML;;NRNWNX;;;LW)D:(A;;GA;;;WD)(A;;GA;;;AC)
\\\\.\\AVGIDSShim
FFD3\\\\.\\NPF_NdisWanIpc:\\sample\\pos.exe
ANALYSERS
SANDBOX
VIRUS
MALWARE
FORTINETMALNETVMc:\\analysis\\sandboxstarter.exec:\\analysisc:\\insidetmc:\\windows\\system32\\drivers\\vmmouse.sys
c:\\windows\\system32\\drivers\\vmhgfs.sys
c:\\windows\\system32\\drivers\\vboxmouse.sys
c:\\iDEFENSEc:\\popupkiller.exe
c:\\tools\\execute.exe
c:\\Perlc:\\Python27api_log.dll
dir_watch.dll
pstorec.dll
dbghelp.dll
Process32NextW
1406Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\Zones\\3
.bitMiniDumpWriteDump
```

```
\r\nReferer: %s\r\n
\\Google\\Chrome\\User Data\\Default\\Cache
var %s = new ActiveXObject("WScript.Shell"); %s.Run("%s");
GenuineIntelAuthenticAMDCentaurHauls7z
fnbqooqdaixfueangywblgabirdgvkewdyqgfqaioluesyrpryfkjerfsouemaxnavrkguxmcmhckwprunurmhehclermtufwiyjbqhwlunbun
uumeowfjmerxppxrgaxukyx
PowerManager_M5VKII_%d
[type=ftp]\n[botid=%s]\n[proc=%s]\n[data=%s]\n
[type=pop3]\n[botid=%s]\n[proc=%s]\n[data=%s]\n
%OS%_%NUMBER_OF_PROCESSORS%
[type=post]\n[botid=%s]\n[url=%s]\n[ua=%s]\n[proc=%s]\n[ref=%s]\n[keys=%s]\n[data=%s]\n
name=%s&ok=%s&id=%d&res_code=%d&res_text=%s_%x
name=%s&ok=%s&id=%d&res_code=%d&res_text=%s
botid=%s&ver=%s.%u&up=%u&os=%u&ltime=%s%d&token=%d&cn=%s&av=%s&dmn=%s&mitm=%u
java.exe|javaw.exe|plugin-container.exe|acrobat.exe|acrod32.exe
tellerplus|bancline|fidelity|micrsolv|bankman|vanity|episys|jack
henry|cruisenet|gplusmain|silverlake|v48d0250s1Root|TrustedPeople|SMS|Remote Desktop|REQUEST
TREASURE|BUH|BANK|ACCOUNT|CASH|FINAN|MONEY|MANAGE|OPER|DIRECT|ROSPIL|CAPO|BOSS|TRADEactive_bc
----------------------------%s\r\nContent-Disposition: form-data; name=\"pcname\"\r\n\r\n%s!%s\r\n----------------------------
%s\r\nContent-Disposition: form-data; name=\"file\"; filename=\"report\"\r\nContent-Type: text/plain\r\n\r\n%s\r\n--------------
---------------%s--\r\n
%domain%deactivebc
inject
kill_os
loadactive_sk
deactive_sk
wipe_cookiesmitm_modmitm_script
mitm_geterr
get_keylog
get_sols!active_bc\[(\d+)\] (\S+) (\d+)
!deactive_bc\[(\d+)\]
!inject\[(\d+)\] (\S+)
!kill_os\[(\d+)\]
!get_keylog\[(\d+)\]!load\[(\d+)\] (\S+)!update\[(\d+)\] (\S+)
!wipe_cookies\[(\d+)\]
!active_sk\[(\d+)\] (\S+) (\d+)
!deactive_sk\[(\d+)\]
!mitm_mod\[(\d+)\] (\S+) (\d+) (\S+)!mitm_script\[(\d+)\] (\S+)
!mitm_geterr\[(\d+)\]
!get_sols\[(\d+)\]
ATCASH
ATLOCAL
CERTCERTX
COLVCRAIF
CRYPT
CTERM
SCREEN
INTER
ELBALOCAL
ELBAWEB
ELBAWEB
ELBAWEB
PUTTY
VNCVIEW
MCLOCAL
MCSIGN
OPENVPN
PIPEK
PIPEK
PIPEK
PIPEK
POSTSAP
chrome.dll
mxwebkit.dlldragon_s.dlliron.dllvivaldi.dll
nspr4.dll
nss3.dllbrowser.dll
Advapi32.dllrsaenh.dll
kernel32.dllIprivLibEx.dll
cryptui.dll
crypt32.dll
ntdll.dll
ssleay32.dllurlmon.dll
user32.dll
Wininet.dll
Ws2_32.dll
PSAPI.dll
NzBrco.dll
VirtualProtect
```

```
193 LoadLibraryExW
194 ZwQuerySystemInformationWSARecv
195 WSASend
196 ZwDeviceIoControlFile
197 URLDownloadToCacheFileW
198 URLDownloadToFileW
199 TranslateMessageSSL_get_fd
200 SSL_write
201 PFXImportCertStore
202 CryptEncryptCPExportKey
203 CreateProcessInternalW
204 CreateDialogParamW
205 GetClipboardDatagetaddrinfo
206 gethostbyname
207 GetAddrInfoExW
208 GetMessageA
209 GetMessageW
210 DeleteFileA
211 GetModuleBaseNameW
212 bad port value
213 can't find plug-in path
214 can't get bot path
215 can't download file
216 can't encrypt file
217 can't save inject config to filecan't get temp file
218 file is not valid PEcan't delete original file
219 can't replace original file
220 can't close handle
221 can't protect file
222 original file not found
223 can't execute file
224 can't create directory
225 can't unzip file #1
226 can't unzip file #2
227 mitm_mod is inactivehttpd.exe is anactive
228 microsoft.com
229 dropbox.com
230 KEYGRAB
231 PasswordTELEMACOScelta e Login dispositivo
232 TLQ Web
233 db Corporate Banking WebSecureStoreCSP - enter PIN
234 google.com
235 Software\\SimonTatham\\PuTTYreg.txt
236 Software\\Microsoft\\Internet Explorer\\MainTabProcGrowth
237 Temp\\Low
238 crc32[%x]
239 ACCT
240 AUTHINFO PASS
241 AUTHINFO USER
242 Authorization
243 :BA:[bks]
244 %X!%X!%08X
245 btc_path.txtbtc_wallet.dat
246 bitcoin\\wallet.dat
247 %s%s\\%u_cert.pfx
248 cmdline.txt
249 1.3.6.1.5.5.7.3.3
250 CodeSign\n
251 Software\\Microsoft\\Windows NT\\CurrentVersion
252 [del]
253 Default
254 .exeELBA5\\ELBA_dataftp://anonymous:ftp://%s:%s@%s:%d\n
255 HBPData\\hbp.profileHH:mm:ssdd:MMM:yyyy
256 I_CryptUIProtect\\exe\\
257 infected.exx%s%s\\%u_info.txt
258 [ins]
259 InstallDate
260 %02u.jpg%s\\%02d.jpgKEYLOG
261 %s\\keylog.txt
262 [TOKEN ON]
263 \n\n[%s (%s-%s) - %s (%s)]\n[pst]%s[/pst]
264 ltcd_path.txt
265 ltcd_wallet.dat
266 litecoind\\wallet.dat
267 ltc_path.txtltc_wallet.dat
268 litecoin\\wallet.dat\\MacromediaMultiCash@Sign
269 C:\\Omikron\\MCSign
270 [ML][MR]Global\\{4C470E-%08x-%08x-%08x}
```

```
Global\\{DAN6J0-%s}
noneopera.exe
PASS
password.txt\\\\.\\pipe\\%s
pop3://%s:%s@%s:%d\n%PROCESSOR_ARCHITECTURE%Referer
[ret]
%08x\\system32\\rstrui.exe
\\scrs\\send%s%s%s%d%s:%s
sysinfo.txt
[tab]
data.txt<unnamed>
<untitled>
update
USER
User-agent
vkeys
%x\r\n
\r\n%x%x%x.tmp
\\*.txt
%02x%2b
torrent
-config config.vnc
--config
config.ovpn
data.txt[type=post]\n
CreateFileW
pos.exe
bank.exePOS
secure.
.mozgoogle.com
CertVerifyCertificateChainPolicyCertGetCertificateChain
SSL_AuthCertificateHook
USERNAMESoftware\\ESET\\ESET Security\\CurrentVersion\\Info
C8FFAD27AE1BBE28BE24DDF20AF36EF901C609968930ED82CEFBC64808BA34102C4FABA0560523FB4CCBF33684F77C8401DFB
3A7D2D598E872DD78033E7F900B78A0C710CDF0941662FF7745A435D4BC18D5661E0582B21B2DB8FCA1C0CA3401D0FC9F051
85A558AB6A76A010F606CD77B35A480B6B7176F0903299B91F1BBD141B4D33615849C35557357DAB819BC3D4A8722BB433DE
B66C7A326BE859BD94930331B37DEE6EF4C475EA4B33DE4699FFDBCD34E196E19FE630E631D2C612705048620183BCF56709B
484A4380C4B00D8D94D131C31DB53AE6BCDCCC14131BAC99A68C59A604D0AE9116E9196F7FA3EA5F86F67E9B175CC09D3E17
997728B7D
10001
get=1
COMPNAMEAppDataDir
updfiles\\upd.ver
updfiles\\lastupd.ver
SYSTEM\\CurrentControlSet\\services\\Avg\\SystemValues
Local AppData
Avg2015
Avg2014
Avg2013
Avg2012
Avg2011
update
Software\\Microsoft\\Windows\\CurrentVersion\\explorer\\Browser Helper Objects\\{8CA7E745-EF75-4E7B-BB86-
8065C0CE29CA}
Software\\Microsoft\\Windows\\CurrentVersion\\explorer\\Browser Helper Objects\\{BB62FFF4-41CB-4AFC-BB8C-
2A4D4B42BBDC}
Software\\Microsoft\\Internet Explorer\\MainEnable Browser Extensions
httpd.exe
%s\\httpd.exe
connect
data\\index.php
logs\\error.log
error.log
<?\n';\n$bot_id = '
$bot_net = '$key_log_file = '
$process_file = '
127.0.0.1
Listen %s:%u\n
conf\\httpd.confSSL_PORT%u>\n
[type=post]\n
[type=screen]\n
[type=knock]\n
74??834E0440B832FFFFFF
74??834E04405F5EB832FFFFFF
DEBUG
memory.dmp
config.xml
php5ts.dll
```

```
zend_stream_fixup
zend_compile_file
index.php
config.php
content.php
iexplore.exe|firefox.exe|chrome.exe|opera.exe|browser.exe|dragon.exe|epic.exe|sbrender.exe|vivaldi.exe|maxthon.exe|ybr
owser.exe|microsoftedgecp.exe
InternetQueryDataAvailable
InternetReadFileInternetReadFileExA
InternetReadFileExW
InternetSetStatusCallbackA
InternetSetStatusCallbackW
HttpSendRequestAHttpSendRequestExA
HttpSendRequestExW
HttpSendRequestW\r\n0\r\n\r\n
.rdata
\r\n\r\nHTTP/1.
Transfer-Encoding
chunked
Content-Length
close
Proxy-ConnectionHostAccept-Encoding
x-xss-protectionx-content-security-policy
x-frame-options
x-content-type-options
If-Modified-Since
If-None-Match
content-security-policy
x-webkit-cspConnection
http://
https://NSS layer
Content-TypeBasic
PR_ClosePR_Connect
PR_GetNameForIdentity
PR_Read
PR_SetError
PR_WriteReferer:
Accept-Encoding:\r\n1406SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3
data_after\ndata_before\n
data_enddata_inject\n
set_url %BOTID%
%BOTNET%InternetCloseHandle
HTMLc:\\inject.txt
Dalvik/1.6.0 (Linux; U; Android 4.1.2; GT-N7000 Build/JZO54K)
xxx_process_0x%08x
Common.js
```

API Obfuscation

The main payload uses an API obfuscation technique known as Push-Calc-Ret obfuscation. The calls to the real API functions are patched by the second stage injector after the main payload gets injected into the svchost process. Whenever a Windows API function should have been called, instead the address of a trampoline function is called which calculates the actual function address. All the trampoline function addresses are stored in an array in memory.

For example, the main payload wants to call CreateFile(), but this call is patched. Now, it calls the trampoline function which could look as follows:

```
00846110   PUSH 2B464C25
00846115   PUSHFD
00846116   XOR DWORD PTR SS:[ESP+4], 5DB5E13F
0084611E   POPFD
0084611F   RETN
```

First, a value is pushed to the stack. Next, the EFLAGS register is saved to the stack, because it will be altered by the following XOR instruction (OF, CF flags are cleared and the SF, ZF, and PF flags are set according to the result). Then, the previously pushed value is XORed with another value to calculate the actual API function address. At last, the EFLAGS register gets restored and the real API function address is called via the RETN instruction.

Persistence Method

The main payload copies the initial obfuscated loader file to the %ProgramData% folder with a random file retrieved with GetTickCount(). Then, it creates a JScript file named "Common.js" in the Startup folder of the current user. The file contains the following code which runs the initial loader after the system was rebooted:

```
var yqvltidpue = new ActiveXObject("WScript.Shell");
yqvltidpue.Run("C:\\PROGRA~3\\930d4a6d.exe")
```

Updates of the Main Payload compared to Previous Version

Reports on previous versions of Shifu have been published by FireEye and Fortinet.

In comparison to the previous version, the list of substrings to scan for in the string that gets created with the computer name, user name, install date and system drive volume serial number was expanded:

- TREASURE
- BUH
- BANK
- ACCOUNT
- CASH
- FINAN
- MONEY
- MANAGE
- OPER
- DIRECT
- ROSPIL
- CAPO
- BOSS
- TRADE

Updated command list:

- active_sk
- deactive_sk
- deactivebc
- get_keylog
- get_sols
- inject
- kill_os
- load
- mitm_geterr
- mitm_mod
- mitm_script
- wipe_cookies

Updated list of targeted browsers:

- iexplore.exe
- firefox.exe
- chrome.exe
- opera.exe
- browser.exe
- dragon.exe
- epic.exe
- sbrender.exe
- vivaldi.exe
- maxthon.exe
- ybrowser.exe
- microsoftedgecp.exe

The main payload will download the Apache httpd.exe server file from one of the C&C servers to store it on disk for web injection purposes. Compared to the previous version, the main payload also contains two strings which indicate some functionality for the Zend PHP Framework:

- zend_stream_fixup
- zend_compile_file

Function Hooking in Svchost

Like in the previous version, the malware hooks some API functions to redirect URLs, capture network traffic, the clipboard and to log keystrokes. It uses a technique known as inline function hooking where the first 5 bytes of a function get patched with a jump to the malware's hook handlers. The following functions get hooked:

- NtDeviceIoControlFile (ntdll.dll)
- ZwDeviceIoControlFile (ntdll.dll)
- GetClipboardData (user32.dll)
- GetMessageA (user32.dll)
- GetMessageW (user32.dll)
- TranslateMessage (user32.dll)
- GetAddrInfoExW (ws2_32.dll)
- gethostbyname (ws2_32.dll)
- getaddrinfo (ws2_32.dll)

Network Functionality

The main payload of Shifu uses .bit top-level domains which is a decentralized DNS system based on the Namecoin infrastructure. The malware requests the IP addresses of the domains by subsequently contacting the following hardcoded Namecoin DNS servers:

- 92.222.80.28
- 78.138.97.93
- 77.66.108.93

The C&C domain names, the user-agent string and the URL parameters are encrypted with a modified RC4 encryption algorithm. Decrypted strings:

- klyatiemoskali.bit
- slavaukraine.bit
- Mozilla/5.0 (Windows; U; Windows NT 5.2 x64; en-US; rv:1.9a1) Gecko/20061007 Minefield/3.0a1
- L9mS3THljZylEx46ymJ2eqIdsEguKC15KnyQdfx4RTcVu8gCT
- https://www.bing.com
- /english/imageupload.php
- /english/userlogin.php
- /english/userpanel.php
- 1brz

The encrypted strings are stored in the following format inside the .data section:

<LengthOfString><EncryptedString>

The domain string “klyatiemoskali“ means roughly translated to wish something bad to Muscovites. The second domain string “slavaukraine” means translated “glory to the Ukraine”. The included RC4 key "L9mS3THljZylEx46ymJ2eqIdsEguKC15KnyQdfx4RTcVu8gCT" is used to encrypt the network traffic.

At the time of analysis, only the following Namecoin DNS server was answering with the IP address of the actual C&C server:

77.66.108.93 (ns1.dk.dns.d0wn.biz)

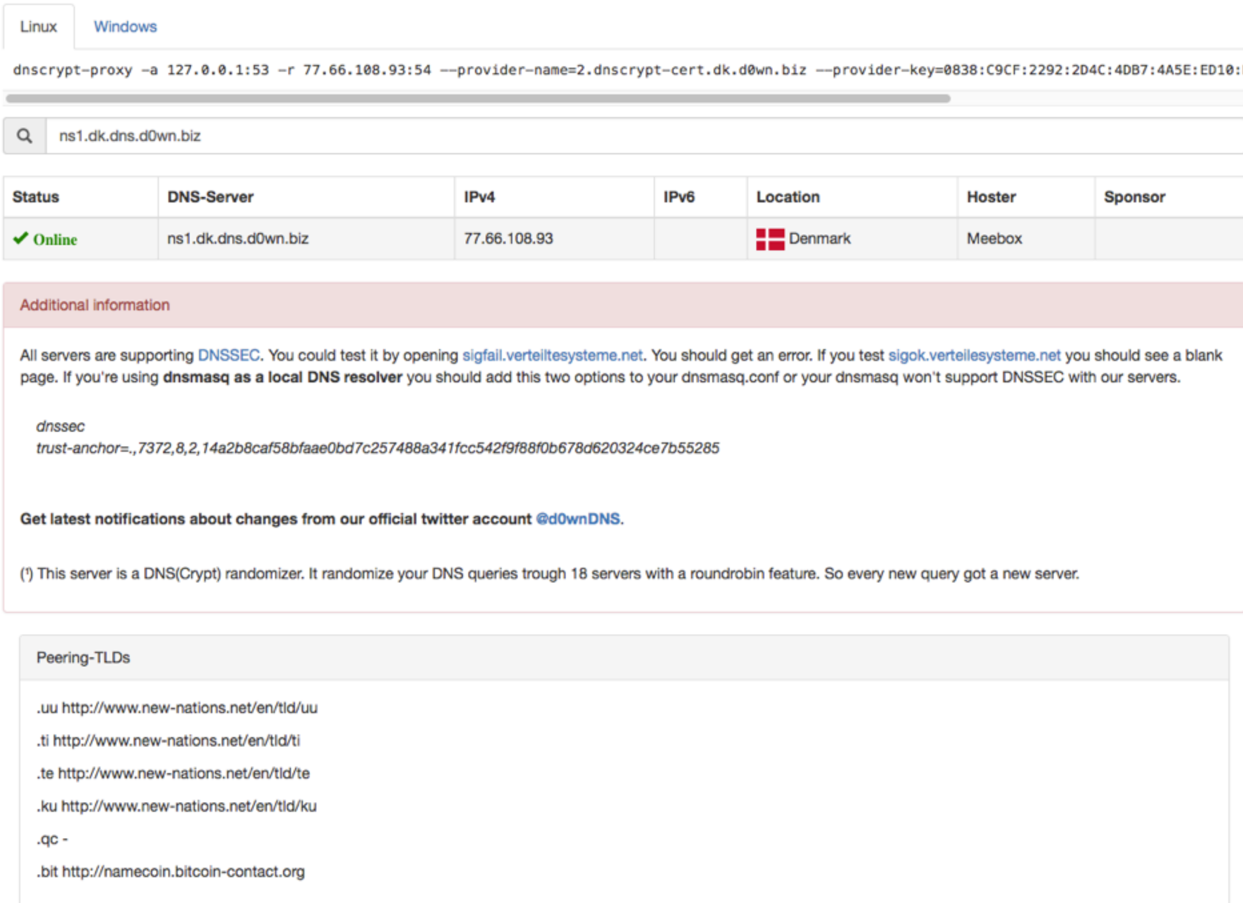


Figure 5. Namecoin DNS server information of 77.66.108.93

The following screenshot shows the captured network traffic during the dynamic analysis of Shifu:

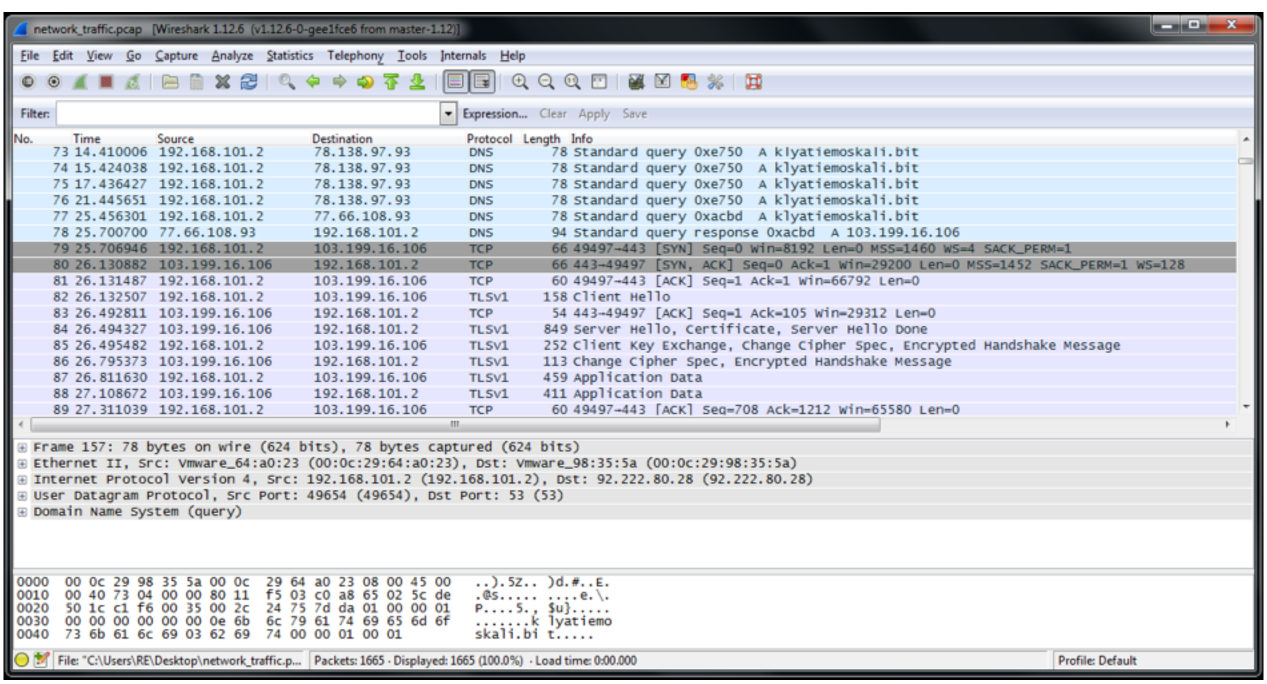
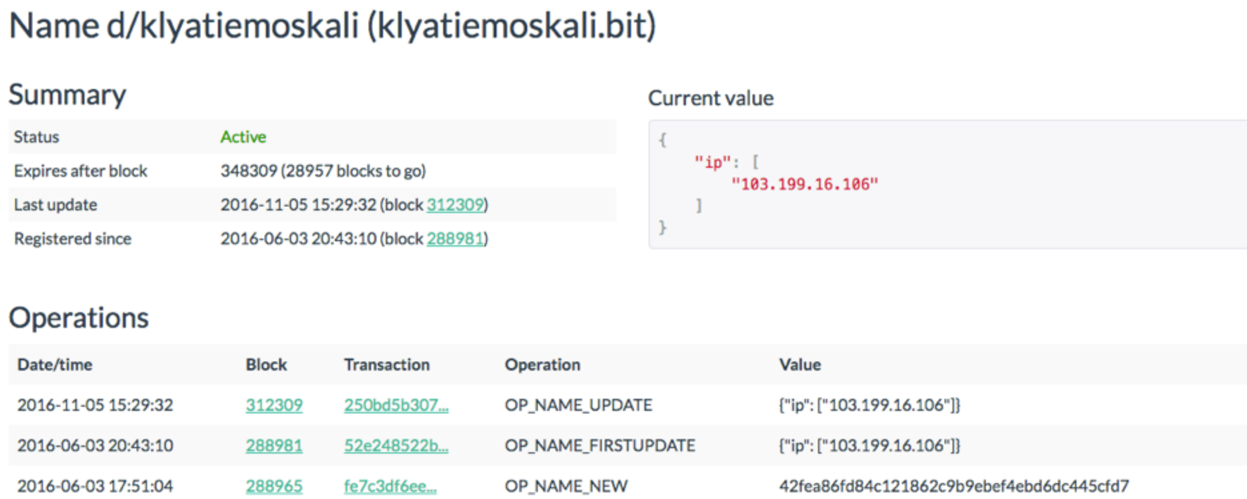


Figure 6. Shifu network traffic captured with Wireshark

We can see that Shifu subsequently queries the Namecoin DNS servers with the domain name klyatiemoskali.bit to get the IP address. After one name server responds with the IP address of the C&C server, it does a TLS handshake to open an encrypted network channel. Finally, it sends some encrypted data and gets an encrypted answer. However, no further network traffic could have been observed during the time of the analysis. Both domain names, klyatiemoskali.bit and slavaukraine.bit, resolved to the IP address 103.199.16.106 at the time of analysis.

As the .bit top-level domain relies on the Namecoin cryptocurrency which is based on the Bitcoin system, every transaction can be traced back. Thus, we can use a Namecoin block explorer to look when the .bit domains were registered and which IP addresses are connected to it. For example, if we use the web service namecha.in, we can get the following information for klyatiemaskali.bit:



Name d/klyatiemoskali (klyatiemoskali.bit)

Summary

Status	Active
Expires after block	348309 (28957 blocks to go)
Last update	2016-11-05 15:29:32 (block 312309)
Registered since	2016-06-03 20:43:10 (block 288981)

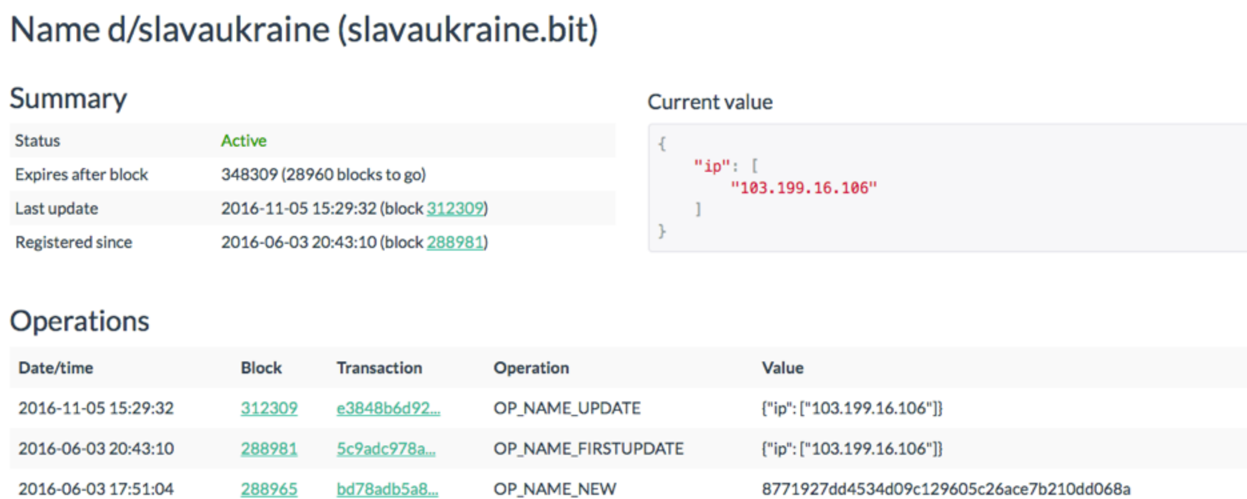
Current value


```
{
    "ip": [
        "103.199.16.106"
    ]
}
```


Operations

Date/time	Block	Transaction	Operation	Value
2016-11-05 15:29:32	312309	250bd5b307...	OP_NAME_UPDATE	{"ip": ["103.199.16.106"]}
2016-06-03 20:43:10	288981	52e248522b...	OP_NAME_FIRSTUPDATE	{"ip": ["103.199.16.106"]}
2016-06-03 17:51:04	288965	fe7c3df6ee...	OP_NAME_NEW	42fea86fd84c121862c9b9ebef4ebd6dc445cfd7

We can see the same information for slavaukraine.bit:



Name d/slavaukraine (slavaukraine.bit)

Summary

Status	Active
Expires after block	348309 (28960 blocks to go)
Last update	2016-11-05 15:29:32 (block 312309)
Registered since	2016-06-03 20:43:10 (block 288981)

Current value


```
{
    "ip": [
        "103.199.16.106"
    ]
}
```


Operations

Date/time	Block	Transaction	Operation	Value
2016-11-05 15:29:32	312309	e3848b6d92...	OP_NAME_UPDATE	{"ip": ["103.199.16.106"]}
2016-06-03 20:43:10	288981	5c9adc978a...	OP_NAME_FIRSTUPDATE	{"ip": ["103.199.16.106"]}
2016-06-03 17:51:04	288965	bd78adb5a8...	OP_NAME_NEW	8771927dd4534d09c129605c26ace7b210dd068a

Both domains were registered on 2016-06-03 and only one IP address is assigned to them. This IP address coincides with the response of the Namecoin DNS server we have seen in the captured network traffic. Moreover, we can see the domain seems to be still active.

URL Query String for C&C Server

The main payload contains a query string template used to send information of the victim to the C&C server:

botid=%s&ver=%s.%u&up=%u&os=%u&ltime=%s%d&token=%d&cn=%s&av=%s&dmn=%s&mitm=%u

We can see that some information is dynamically retrieved (**bot id**entifier, **up**time, **o**perating **s**ystem version, **l**ocal **time**stamp, **token**, **a**nti-**v**irus software, **d**o**m**ai**n** name of workstation, **m**an **i**n **t**he **m**iddle interception detected), while also static values like the bot **ver**sion and the **c**ampaign **n**ame are send. An example of the created query string could look as follows:

botid=26C47136!A5A4B18A!F2F924F2&ver=1.759&up=18294&os=6110&ltime=-8&token=0&cn=1brz&av=&dmn=&mitm=0

We can see that the internal Shifu version is “1.759” and the campaign name is stated “1brz”.

If we compare Shifu's query string with the one of the latest Shiz version we have tracked which dates February 2014 (internal version 5.6.25), we can see the similarity between those two malwares:

botid=%s&ver=5.6.25&up=%u&os=%03u&ltime=%s%d&token=%d&cn=sochi&av=%s

Modified RC4 Encryption Algorithm

Shifu uses a modified version of the RC4 encryption algorithm. We have reconstructed the algorithm in Python and show how the domain name "klyatiemoskali.bit" present in the main payload will be encrypted as an example:

```
import os
import binascii

###initial values##########
string = "klyatiemoskali.bit"
seed =
"fnbqooqdaixfueangywblgabirdgvkewdyqgfqaioluesyrpryfkjerfsouemaxnavrkguxmcmhckwprunurmhehclermtufwi
yjbqhwlunbunuumeowfjmerxppxrgaxukyx"
buffer = [0] * (len(string))
table_encr = [0] * 0x102
table_encr[0x100] = 1
table_encr[0x101] = 0
###########################

###string2buffer###########
i = 0
while (i<len(string)):
    char_1 = string[i]
    int_1 = ord (char_1)
    buffer[i] = int_1
    i += 1
###string2buffer###########

###encryption table########
i = 0
while (i < 0x100):
    table_encr[i] = 0x000000ff&i
    i += 1

i = 0
j = 0
while (i < 0x100):
    char_1 = seed[j]
    int_2 = ord (char_1)
    table_encr[i] ^= int_2
    i += 1
    j += 1
    if (j == len(seed)):
        j = 0
###########################

###encryption##############
size_1 = len(string)
i = 0
while (size_1 != 0):
    byte_buf = buffer[i]
    ind_1 = table_encr[0x100]
    ind_2 = table_encr[ind_1]
    ind_3 = 0x000000ff&(ind_2 + table_encr[0x101])
    ind_4 = 0x000000ff&(table_encr[ind_3])
    table_encr[ind_1] = ind_4
    table_encr[ind_3] = ind_2
    buffer[i] = 0x000000ff&(table_encr[0x000000ff&(ind_2 + ind_4)] ^ byte_buf)
    table_encr[0x100] = 0x000000ff&(ind_1 + 1)
    table_encr[0x101] = ind_3
    i += 1
    size_1 -= 1

i = 0
str_1 = ""
while (i < len(string)):
    str_1 = str_1 + chr(buffer[i])
    i += 1
###########################

###output##################
print ("Cleartext string: %s" % string)
print ("Encrypted: 0x%s" % binascii.hexlify(str_1))
###########################
```