Win32/Napolar – A new bot on the block

welivesecurity.com/2013/09/25/win32napolar-a-new-bot-on-the-block/

September 25, 2013

There is a new bot on the block. ESET identifies it as Win32/Napolar while its author calls it solarbot. This piece of malware came to our attention mid-August because of its interesting anti-debugging and code injection techniques.

25 Sep 2013 - 06:46PM

There is a new bot on the block. ESET identifies it as Win32/Napolar while its author calls it solarbot. This piece of malware came to our attention mid-August because of its interesting anti-debugging and code injection techniques.

There is a new bot on the block. ESET identifies it as Win32/Napolar while its author calls it solarbot. This piece of malware came to our attention mid-August because of its interesting anti-debugging and code injection techniques. It recently attracted general attention when it was discussed on various reverse engineering forums.

This malware can serve multiple purposes. The three main ones are to conduct Denial of Service attacks, to act as a SOCKS proxy server, and to steal information from infected systems. The malware is able to hook into various browsers to steal information that is submitted in web forms.

We have uncovered many details about this bot since it became active at the end of July, with in-the-wild infections starting mid-August. There have been reports of thousands of infections, many of them in South America. The countries with the most infections are Peru, Ecuador, and Columbia. More information on the geographical distribution for this threat can be found on virusradar.

The author of Win32/Napolar uses a website to promote it. The website looks very professional and contains detailed information about the bot, including the cost ($200 USD for each build) and even a complete change-log of the evolution of the code.

Although we have not yet directly seen Win32/Napolar being distributed in the wild, it seems likely that this threat has been spread through Facebook. Since malware has the ability to steal Facebook credentials, its operator can reuse those credentials to send messages from compromised accounts and try to infect the victim's friends. Below is a list of filenames we have seen used by this malware family:

- *Photo_032.JPG_www.facebook.com.exe*
- *Photo_012-WWW.FACEBOOK.COM.exe*

- *Photo_014-WWW.FACEBOOK.COM.exe*

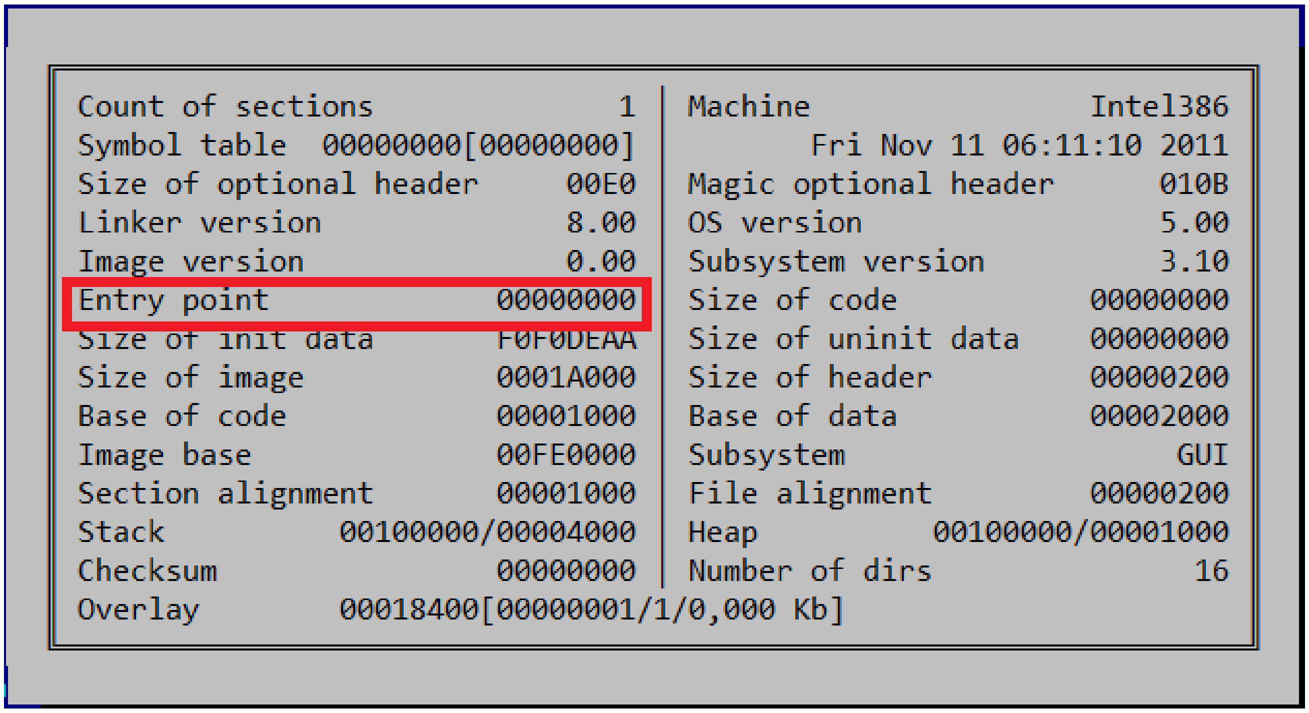
Interestingly enough, the use of doubled file extensions (*.JPG.EXE, *.TXT.EXE and so forth) to obfuscate a file's true extension is an old trick, dating back to Windows 95, but apparently still in use. What is funny about the usage in this particular instance is that the author of Win32/Napolar does not seem to realize that .COM is a valid, if somewhat old, extension for executable files and that these filenames would have allowed their execution without the added .EXE extension. A very recent blog by our colleagues at AVAST confirms they have also seen similar infection vectors.

In this blog post, we will show some of the anti-debugging tricks used by Win32/Napolar. These tricks were seen in early versions of this malware family. Most recent variants also use third party packers to evade antivirus detection and slow down manual reverse engineering.

We will then explain the Win32/Napolar command and control (C&C) protocol. Finally, we will show some of the information that was retrieved from the promotional website before it was taken offline.

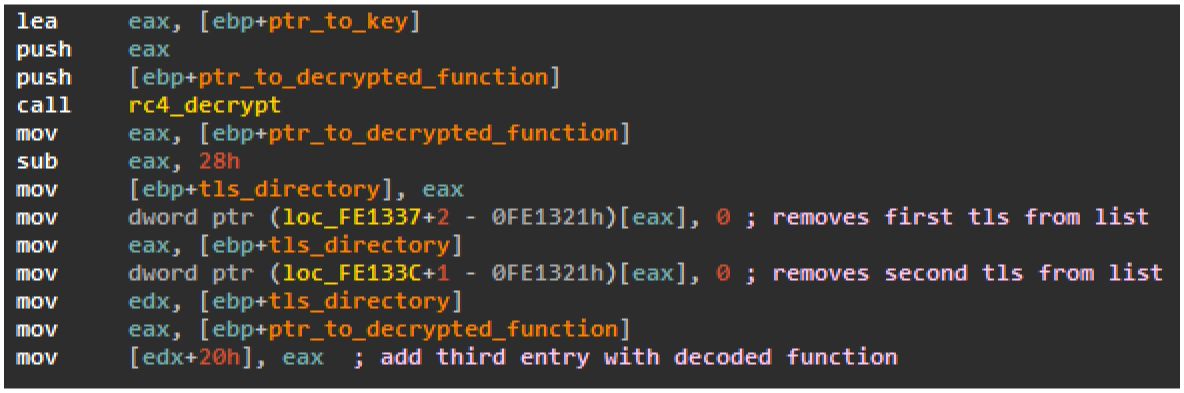
Anti-debugging Techniques

When analyzing Win32/Napolar binaries, the first thing to notice is that there is no valid entry point in the PE header, as shown in the figure below.



```
Count of sections               1 │ Machine                   Intel386
Symbol table  00000000[00000000] │          Fri Nov 11 06:11:10 2011
Size of optional header      00E0 │ Magic optional header         010B
Linker version               8.00 │ OS version                    5.00
Image version                0.00 │ Subsystem version             3.10
Entry point              00000000 │ Size of code              00000000
Size of init data        F0F0DEAA │ Size of uninit data       00000000
Size of image            0001A000 │ Size of header            00000200
Base of code             00001000 │ Base of data              00002000
Image base               00FE0000 │ Subsystem                      GUI
Section alignment        00001000 │ File alignment            00000200
Stack           00100000/00004000 │ Heap             00100000/00001000
Checksum                 00000000 │ Number of dirs                  16
Overlay         00018400[00000001/1/0,000 Kb]
```

The first instructions that are executed when the binary is started are saved in the Thread Local Storage (TLS) functions. There are two TLS functions registered. The first TLS function does not do anything. The second function decrypts more code using the RC4 encryption algorithm and the key 0xDEADBEEF. The decrypted code is registered as a third TLS function before the second function returns, as shown in the code extract below.



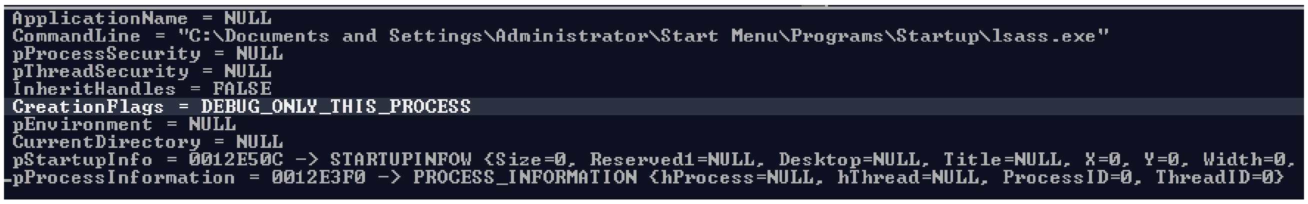
```
lea     eax, [ebp+ptr_to_key]
push    eax
push    [ebp+ptr_to_decrypted_function]
call    rc4_decrypt
mov     eax, [ebp+ptr_to_decrypted_function]
sub     eax, 28h
mov     [ebp+tls_directory], eax
mov     dword ptr (loc_FE1337+2 - 0FE1321h)[eax], 0 ; removes first tls from list
mov     eax, [ebp+tls_directory]
mov     dword ptr (loc_FE133C+1 - 0FE1321h)[eax], 0 ; removes second tls from list
mov     edx, [ebp+tls_directory]
mov     eax, [ebp+ptr_to_decrypted_function]
mov     [edx+20h], eax  ; add third entry with decoded function
```

The third TLS function decrypts the rest of the code before calling the main body of the malware. The malware uses other tricks to make itself harder to analyze:

- All imports are resolved at runtime using hashes instead of the import names.
- Interactions with the operating system are mostly done by directly calling undocumented functions of the NTDLL library instead of using the standard APIs.
- All the code is position-independent.

To find the offset of its own code that will be decrypted, Win32/Napolar searches through its memory for the opcode *0x55*. This opcode represents "*push ebp*", the first instruction of the current function in assembly language. If this instruction is replaced by *0xCC*, the opcode for a software breakpoint, the decryption of the code will not work. This is a clever way of altering the behavior of the malware if it is being analyzed with a debugger and if a software breakpoint is put on the first instruction of the TLS.

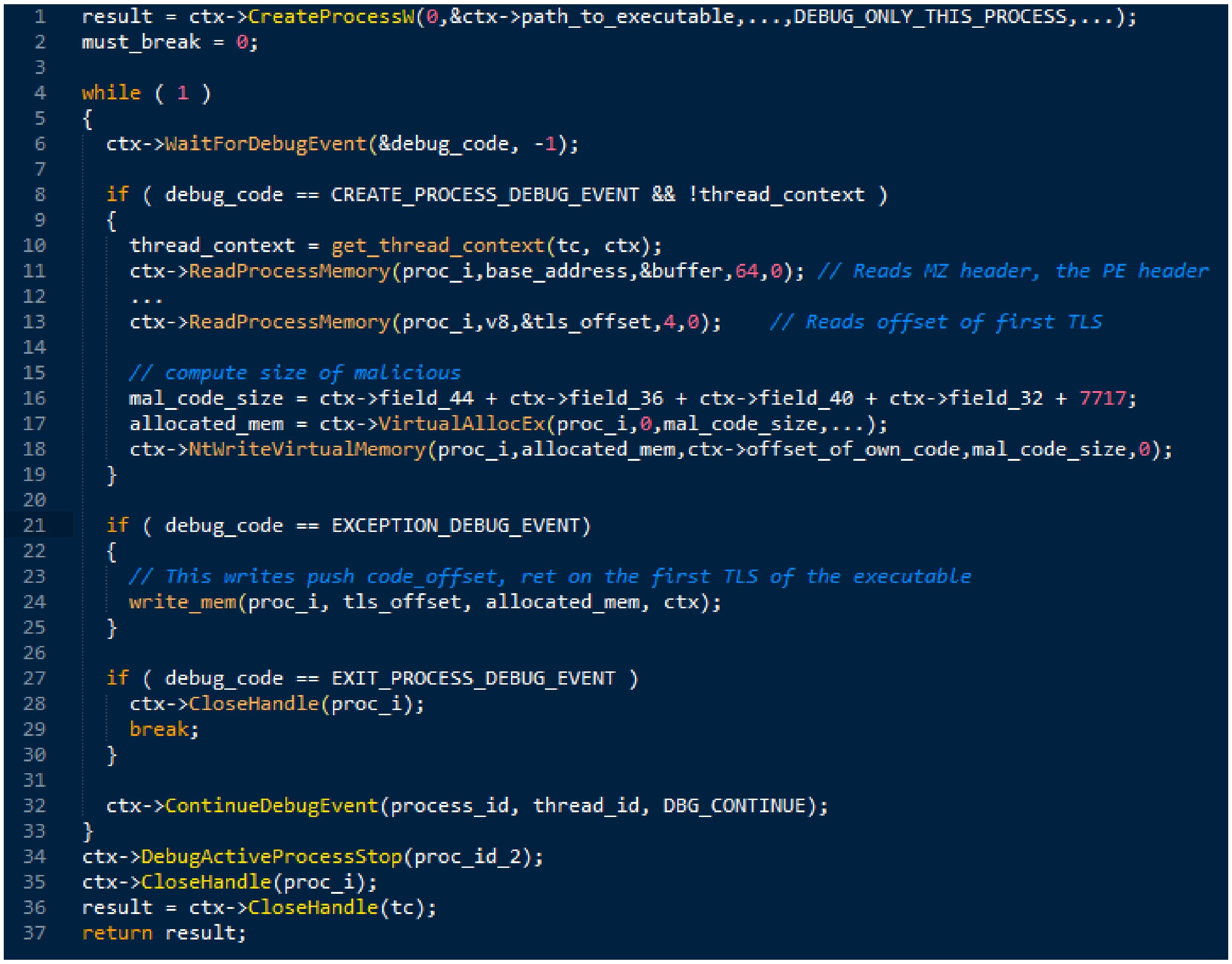
Win32/Napolar has more anti-debugging tricks. To make dynamic analysis harder, Win32/Napolar will create a sub process of itself and will debug this new instance. The screenshot below shows the call to *CreateProcess*.



```
ApplicationName = NULL
CommandLine = "C:\Documents and Settings\Administrator\Start Menu\Programs\Startup\lsass.exe"
pProcessSecurity = NULL
pThreadSecurity = NULL
InheritHandles = FALSE
CreationFlags = DEBUG_ONLY_THIS_PROCESS
pEnvironment = NULL
CurrentDirectory = NULL
pStartupInfo = 0012E50C -> STARTUPINFOW (Size=0, Reserved1=NULL, Desktop=NULL, Title=NULL, X=0, Y=0, Width=0,
pProcessInformation = 0012E3F0 -> PROCESS_INFORMATION (hProcess=NULL, hThread=NULL, ProcessID=0, ThreadID=0)
```

The software protection technique of self-debugging has been seen before but in the case of Win32/Napolar, the trick happens in the main body of the malware, not in the packer.

Once the debugged process is started, Win32/Napolar will enter a loop that handles debugging events returned by the function *WaitForDebugEvent*. Pseudocode for the loop handling debugging events is presented below.

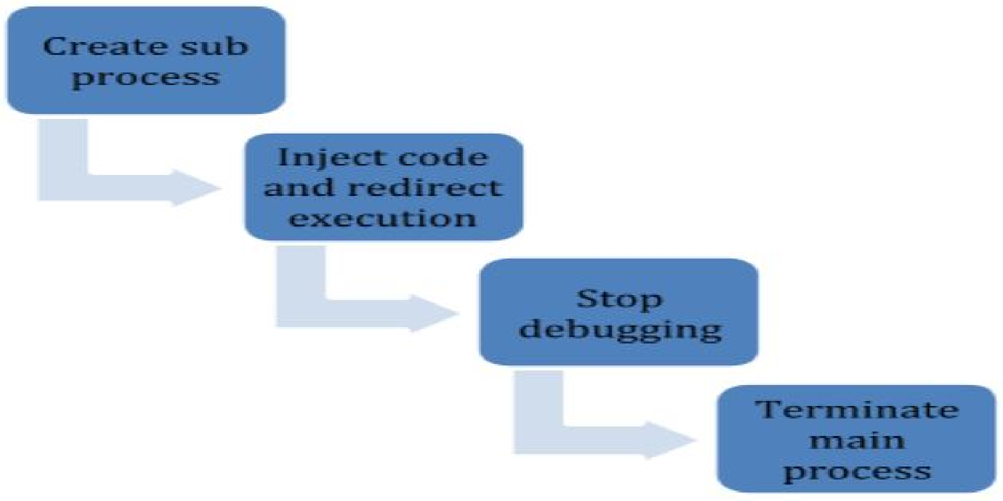


```
result = ctx->CreateProcessW(0,&ctx->path_to_executable,...,DEBUG_ONLY_THIS_PROCESS,...);
must_break = 0;

while ( 1 )
{
  ctx->WaitForDebugEvent(&debug_code, -1);

  if ( debug_code == CREATE_PROCESS_DEBUG_EVENT && !thread_context )
  {
    thread_context = get_thread_context(tc, ctx);
    ctx->ReadProcessMemory(proc_i,base_address,&buffer,64,0); // Reads MZ header, the PE header
    ...
    ctx->ReadProcessMemory(proc_i,v8,&tls_offset,4,0);    // Reads offset of first TLS

    // compute size of malicious
    mal_code_size = ctx->field_44 + ctx->field_36 + ctx->field_40 + ctx->field_32 + 7717;
    allocated_mem = ctx->VirtualAllocEx(proc_i,0,mal_code_size,...);
    ctx->NtWriteVirtualMemory(proc_i,allocated_mem,ctx->offset_of_own_code,mal_code_size,0);
  }

  if ( debug_code == EXCEPTION_DEBUG_EVENT)
  {
    // This writes push code_offset, ret on the first TLS of the executable
    write_mem(proc_i, tls_offset, allocated_mem, ctx);
  }

  if ( debug_code == EXIT_PROCESS_DEBUG_EVENT )
    ctx->CloseHandle(proc_i);
    break;
  }

  ctx->ContinueDebugEvent(process_id, thread_id, DBG_CONTINUE);
}
ctx->DebugActiveProcessStop(proc_id_2);
ctx->CloseHandle(proc_i);
result = ctx->CloseHandle(tc);
return result;
```

The first event handled by this code is CREATE_PROCESS_DEBUG_EVENT. This event takes place when the debugged process is started. In this case, the main process will parse the MZ and PE header of the debugged process in order to retrieve the offset and size of the position-independent code. It will then allocate another area of memory in the debugged process in which to inject the code. This creates two copies of the same code in the same process.

The next event is EXCEPTION_DEBUG_EVENT. In this second event, the main process overwrites the first TLS function of the binary so as to redirect execution at the beginning of the executable, using a push – ret instruction. This, once again, decrypts the main body of the malware and lets it execute within the child process. It is the code of the child process that then proceeds to inject itself into all the processes running sub-processes and hooking various functions to hide its presence on the system and capture desired information.

Finally, the main process receives the EXIT_PROCESS_DEBUG_EVENT event; it stops debugging by calling the function *DebugActiveProcessStop* and terminates its own process using *NtTerminateProcess.*



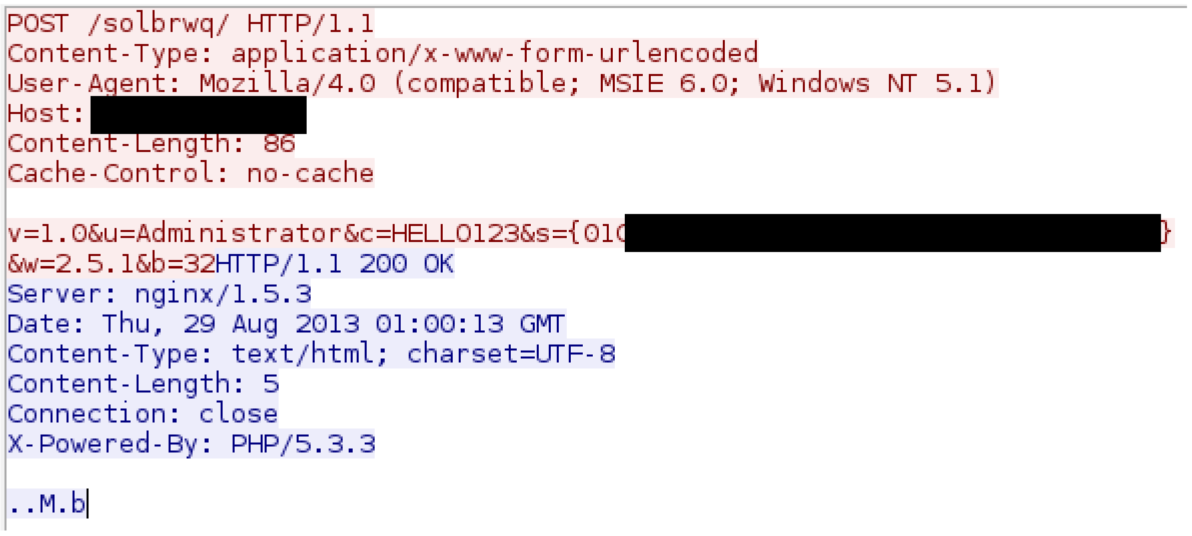
One of the main characteristics of Win32/Napolar is its ability to steal information when a user fills a web form in a web browser. Trusteer's browser protection probably stops the malware from capturing this information. This is why the malware has specific checks for Trusteer products. It will iterate through all the running processes and specifically kill any process that has the string "*trusteer*" in it. We did not perform any test to confirm whether or not this attempt at disabling Trusteer's product is successful or not.

Network behavior

When communicating with its command and control server, Win32/Napolar uses the HTTP protocol. The first query sent by the bot to the command and control server contains the following information:

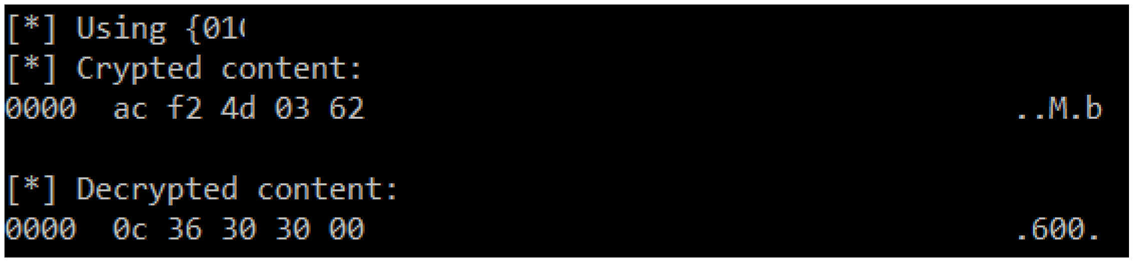
- Version of the bot
- Current windows username of the infected user
- Computer name
- A unique bot identifier
- Version of the operating system
- System type, which can be 32 or 64 bit. Indeed, this bot supports both types of architecture.

The server then responds with commands the bot needs to execute. These commands are encrypted using RC4, The bot unique identifier is used as the encryption key. The bot supports a variety of commands, from information stealing and SOCKS proxying, to denial of service, download, execution and update. Each command has a unique identifier stored as a single byte and the information following this byte contains the command parameters. The following figure shows a traffic dump of the communication between a host infected by Win32/Napolar and its command and control server.



```
POST /solbrwq/ HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: [redacted]
Content-Length: 86
Cache-Control: no-cache

v=1.0&u=Administrator&c=HELLO123&s={01[redacted]}
&w=2.5.1&b=32HTTP/1.1 200 OK
Server: nginx/1.5.3
Date: Thu, 29 Aug 2013 01:00:13 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 5
Connection: close
X-Powered-By: PHP/5.3.3

..M.b
```

The following figure shows the decryption of this command using the proper key. The first byte of the received content is *0xC*, and this instructs the bot to sleep. The parameter is a string, “600”, which represents the number of seconds that the bot needs to sleep.



```
[*] Using {01(
[*] Crypted content:
0000  ac f2 4d 03 62                                  ..M.b

[*] Decrypted content:
0000  0c 36 30 30 00                                  .600.
```

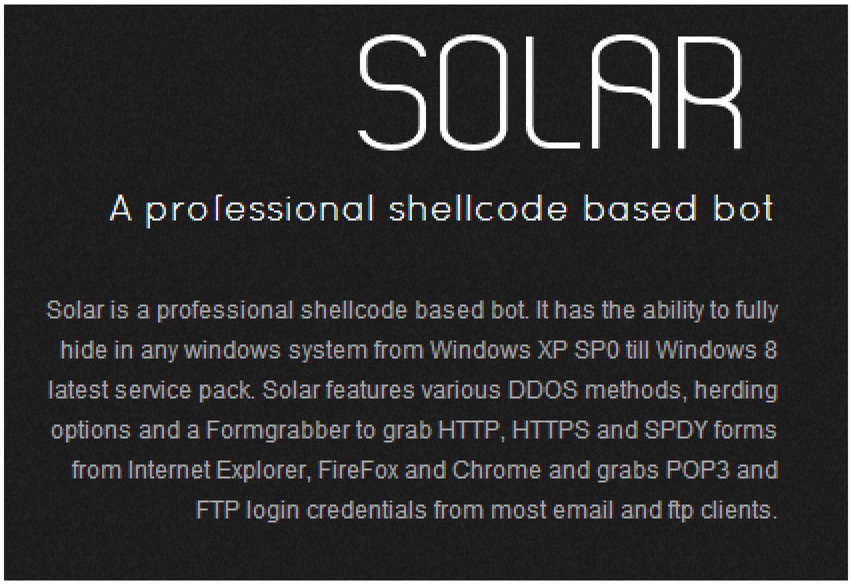
We have seen at least seven different command and control servers used by Win32/Napolar. Most of them only stayed online for a couple of days before the operator moved them to a new network. This might indicate that this bot is being actively used in the wild. Below is a list of domain names where we have recently observed command and control servers:

- *dabakhost.be*
- *terra-araucania.cl*
- *xyz25.com*
- *yandafia.com*
- *elzbthfntr.com*
- *alfadente.com.br*

There are some references to TOR in the malware code. Most precisely, some configuration lines and references to the configuration file for TOR. During our analysis of the malware, it didn’t seem to make any usage of this data. This could be some dormant feature that has not been activated in the samples we have analyzed.

Promotional website

The author of Win32/Napolar seems very frank about wanting to sell his new malware. He has put together a very professional-looking website where he boasts that his bot is a “professional shellcode based bot”, referring to the fact the malware is position-independent.



SOLAR

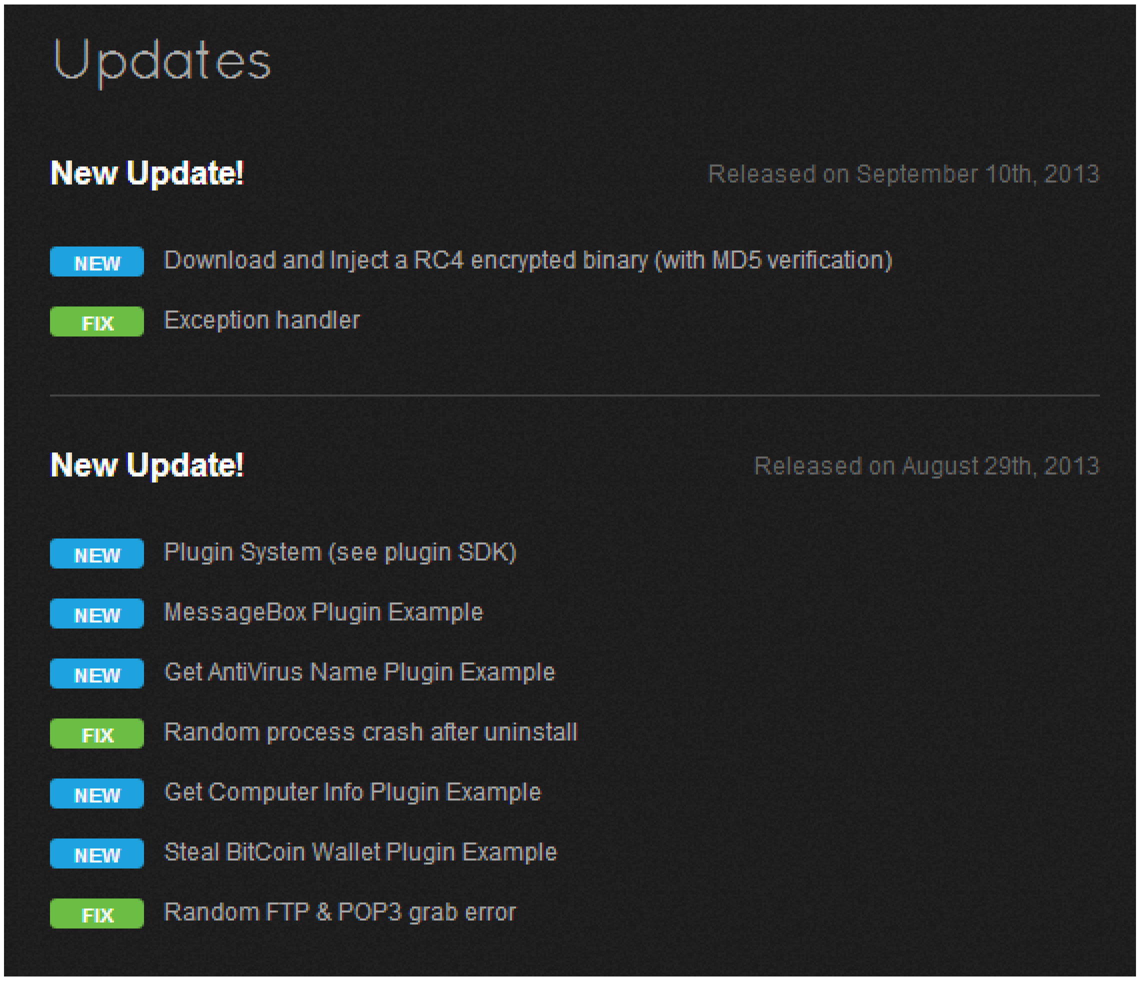
A professional shellcode based bot

Solar is a professional shellcode based bot. It has the ability to fully hide in any windows system from Windows XP SP0 till Windows 8 latest service pack. Solar features various DDOS methods, herding options and a Formgrabber to grab HTTP, HTTPS and SPDY forms from Internet Explorer, FireFox and Chrome and grabs POP3 and FTP login credentials from most email and ftp clients.

The website also provides information for potential customers. For example, the complete code for the command and control server can be found there, a php script running with an SQL database backend. The code of the command and control server confirms of our analysis of the network protocol used by the Win32/Napolar malware.

The promotional website also provides multiple examples of plugins that can be used by malware operators. The plugins must be written using the Delphi programming language. The example plugins show how one can display a message on an infected victim system, find which version of the antivirus is installed on the victim system, and even how to steal Bitcoin wallets.

Finally, the website even presented a complete log of the changes made to the bot’s source code, including information on new features and bug fixes. The website shows the first changelog entry made on July 14th. This fits our timeline since we saw the first instances of this bot in the wild in the beginning of August. The registration date for the domain name where the content is hosted is the first day of August, another indication that the beginning of the promotion is recent.



Updates

New Update! Released on September 10th, 2013

- NEW Download and Inject a RC4 encrypted binary (with MD5 verification)
- FIX Exception handler

New Update! Released on August 29th, 2013

- NEW Plugin System (see plugin SDK)
- NEW MessageBox Plugin Example
- NEW Get AntiVirus Name Plugin Example
- FIX Random process crash after uninstall
- NEW Get Computer Info Plugin Example
- NEW Steal BitCoin Wallet Plugin Example
- FIX Random FTP & POP3 grab error

Conclusions

Win32/Napolar is a new bot that surfaced in July and started to be observed in the wild in August. It has interesting techniques for countering reverse engineering. The most notable point about this malware is how openly it is being promoted on the web by its creator. The advertisement is probably the same that was identified by Dancho Danchev at webroot in July. We have seen many messages on different forums promoting this bot, in addition to the existence of a publicly-accessible website. As it was previously discussed in the Foxxy case, this is another good example of the specialization of cybercrime operations where we now clearly have authors that create malware and sell it to other gangs who will operate it.

Although this bot has functionalities similar to other families like Zeus or SpyEye, it might gain in popularity because its author is actively maintaining it, and because of its ease of use and the simplicity with which plugins can be created.

Analyzed files

The following are MD5 hashes of the analyzed files:

- 85e5a0951182de95827f1135721f73ad0828b6bc

- 9c159f00292a22b7b609e1e8b1cf960e8a4fa795
- a86e4bd51c15b17f89544f94105c397d64a060bb
- ce24ae6d55c008e7a75fb78cfe033576d8416940
- dacfa9d0c4b37f1966441075b6ef34ec8adc1aa6

Acknowledgments

Thanks to Lubomir Trebula and Joan Calvet for their help while analyzing this malware.

Pierre-Marc Bureau
Security Intelligence Program Manager

25 Sep 2013 - 06:46PM