VinSelf - A new backdoor in town!

November 23, 2010 | by Atif Mushtaq

I recently came across a new piece of Modern Malware found to be involved in a highly targeted attack. My initial exploration into the malware revealed it to be a powerful backdoor with the capability to provide an attacker complete control over the infected system.

What's happening at the moment? A few weeks ago, we saw a powerful backdoor Pirpi exploiting the IE 0-day as part of some targeted attacks. Now comes Vinself. The emergence of new and powerful backdoors and their use in the targeted attacks is evidence showing that modern malware is not only used to steal user's credit cards or send spam. There is much more at stake as well.

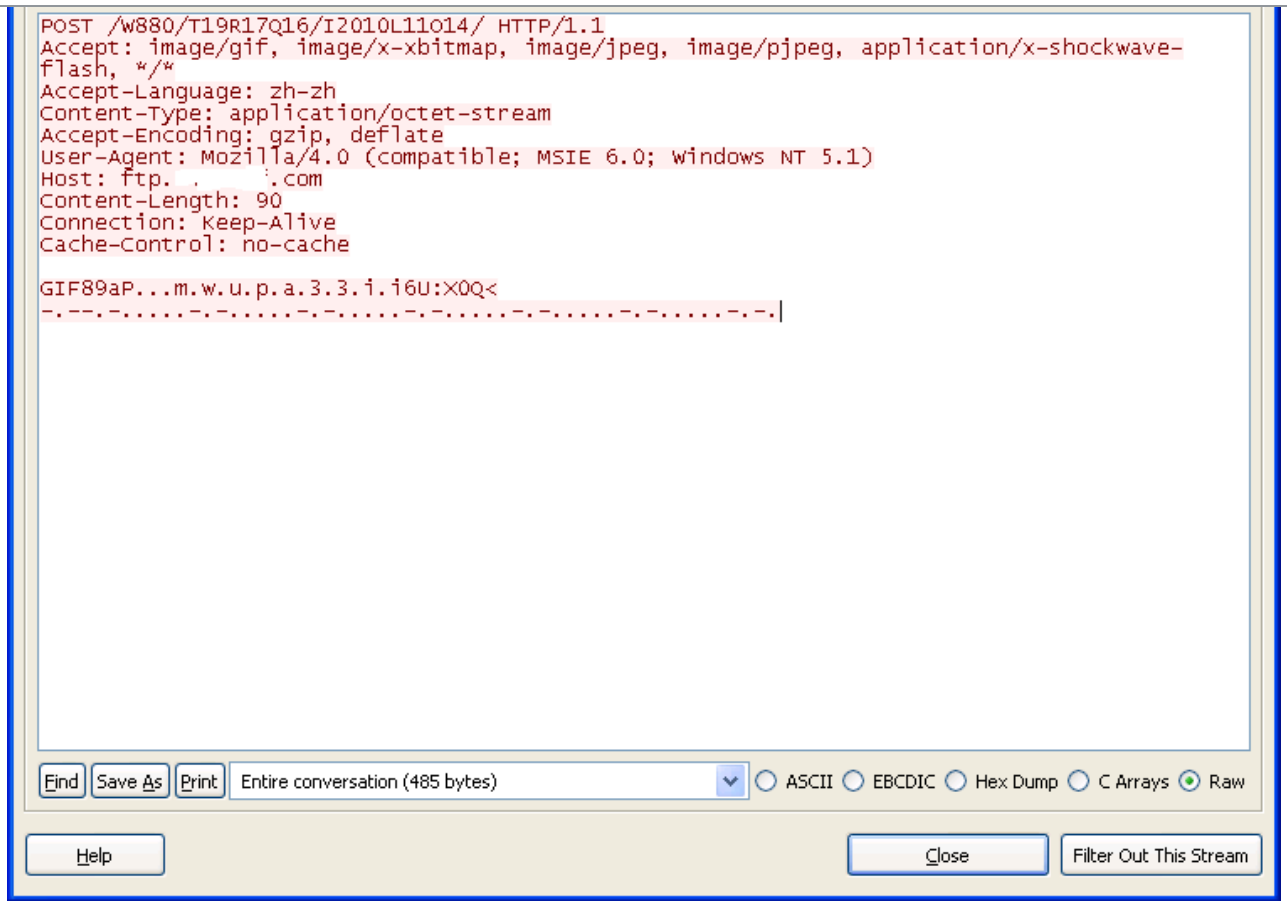
There are many out-and-out criminal gangs (some with potential political affiliations) who are after something more than material gains. They develop targeted malware to get into sensitive networks and then loiter wating for the chance to snatch confidential documents and/or intellectual property. Cases like Vinself (where malware can fully function even if behind proxy firewalls) are also an indication that the main target here is not desktop users.

During the investigation, I found many interesting facts about this malware like the CnC protocol, the obfuscation in use and the backdoor capability etc. Today I would like to share some high level characteristics of this malware.

Command & Control protocol

This backdoor uses HTTP to carry its custom obfuscated protocol. To evade signature-based IPS/IDS, the URLs are generated randomly to be highly dynamic based on the the current time.

Here is how one URL instance would look: (click to see full size)



where the URL

POST /W880/T19R17Q16/I2010L11O14/

is randomly generated like this:

POST /[%c1][%d2]/[**%c3][%d4][%c5][%d6][%c7][%d8]**/[**%c9][%d10][%c11][%d12][%c13][%d14]**/ HTTP/1.

where

%c1 = SystemTime.wMilliseconds % 26 + 'A' = A value between 'A' to 'Z'
%d2 = SystemTime.wMilliseconds = A number ranging from 0 to 999
%c3 = SystemTime.wHour + 'A' = A value between 'A' to 'X'
%d4 = SystemTime.wHour = A number ranging from 0 to 23
%c5 = SystemTime.wMinute % 26 + 'A' = A value between 'A' to 'Z'
%d6 = SystemTime.wMinute = A number ranging from 0 to 59
%c7 = SystemTime.wSecond % 26 + 'A' = A value between 'A' to 'Z'
%d8 = SystemTime.wSecond = A number ranging from 0 to 59
%c9 = SystemTime.wYear % 26 + 'A' = A value between 'A' to 'Z'
%d10 = SystemTime.wYear = A number ranging from 1601 to 30827
%c11 = SystemTime.wMonth % 26 + 'A' = A value between 'A' to 'M'
%d12 = SystemTime.wMonth = A number ranging from 1 to 12
%c13 = SystemTime.wDay % 26 + 'A' = A value between 'A' to 'Z'
%d14 = SystemTime.wDay = A number ranging from 1 to 31

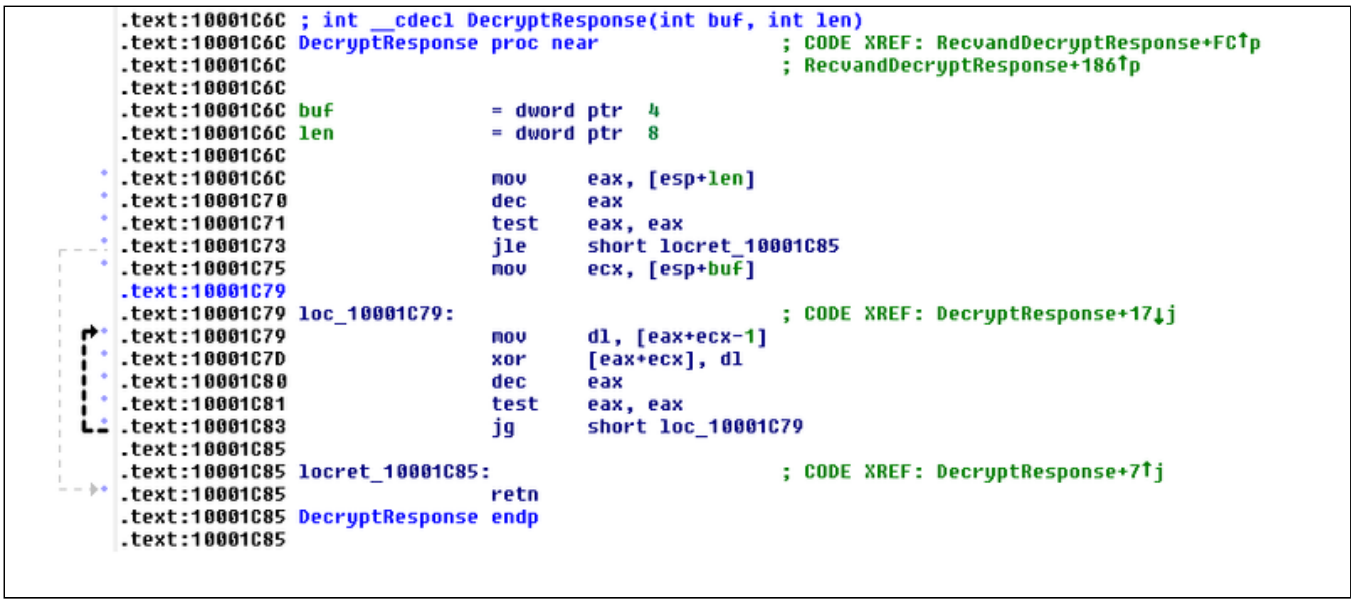
The result of this much randomization is that one will see random HTTP requests like:

POST /X699/O14Y24E4/I2010L11M12/

etc

The next part of the CnC communication is the HTTP stub data. One can see from the above screen shot that this stub data starts with a GIF header (GIF89a). Data after the GIF header is encrypted and can be decrypted like this:

First the 6 bytes long GIF header is skipped. Then starting from the end every leading byte is xored with trailing byte and result is stored back on the same offset and so on. (click for full size image).



An approximate C translation would like this:

```
DecryptResponse(BYTE * buf, int len)
{
  int offset;

 buf = buf + 6;

  for ( offset = len - 1; offset > 0; --offset )
    *(BYTE *)(offset + buf) ^= *(BYTE *)(offset + buf - 1);
}
```

For example after decrypting one of the stub data, I found it to be uploading critical system information to its master(s) like this:

HostName:mycomputer Flag:LLL_IIAA02E UserName:AdministratorOpenTime:2Day-9:30 LocalTime:[11-07 19:17:32] BackTime:NULLC:\Program Files\Internet Explorer\IEXPLORE.EXE [REMOVED]bs.b[REMOVED]k.net

where

'LLL_IIAA02E' is a hard coded string inside the data section. I guess this is served as the malware build identifier, as different VinSelf binaries contain different types of string ids.

other fields are self explanatory.

Similarly in another case, it sent the running process details to its CnC like this:

0[System Process] 01 4System 056 352smss.exe 43 732csrss.exe 35213 756winlogon.exe 35220 800services.exe 75615 812lsass.exe 75618 992svchost.exe 80014 ..

etc.

Command & Control commands:

This backdoor supports a rich set of CnC commands, all there to help an attacker take full control over the infected system. These commands are received in response to the POST request as explained above. These responses are encrypted exactly the way stub requests are done. The scariest part comes when using these responses, bot master(s) logs onto the victim computer to issue different commands. In a sample run in my lab, I saw bot herder(s) silently logging into the infected system in order to grab important system information.

it works like this:

The server response (with a special op code) will contain the DOS compatible command in encrypted form. The backdoor will decrypt and execute this command onto the windows command shell (cmd.exe) and will send back the response after encrypting it.

Some commands which were executed onto my sandnet machine were:

cmd.exe /c "dir c:\ /a /s >c:\windows\system32\soo

dir /q c:\windows\system32\soo.t

ping -a -n 1 207.179.75.114

tasklist /m

etc.

This malware currently supports 18 commands which are sent using opcodes like 0x2710, 0x2716u, 0x271Au etc. All of these op codes trigger a different function onto the infected system.

Like

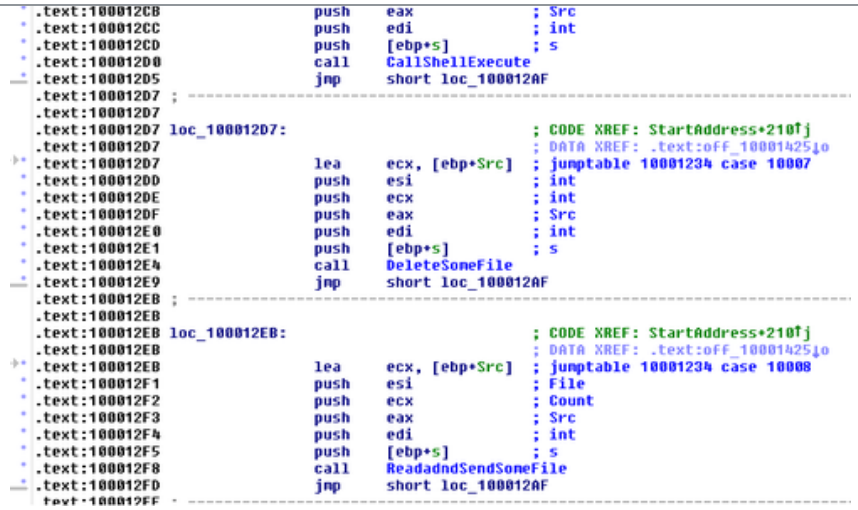
0x2715 is the opcode to invoke the Windows Command Shell (explained above) and get itself ready for issuing the bot masters commands onto the system.

0x271B for killing any existing process.

0x2718 for uploading any file that exists on the file system.

0x271F for launching any program onto the system.

etc.



Architecture:

Mainly this malware is comprised of three components:

1. A watchdog program, responsible for keeping other child components installed onto the system using a powerful rootkit.

2. A dll file containing the main backdoor functionality.

3. A child executable which is responsible for injecting the above mentioned dll file into the Internet Explorer (iexplore.exe) process. Dll injection is done like this:

3.1 Run a brand new instance of Internet Explorer (iexplore.exe)

3.2 Allocate heap memory into the remote process.

3.3 Write the dll path into the external memory region.

3.4 Activate the dll in the remote process using the CreateRemoteThread (Win32 API) trick.

Once the dll gets loaded inside iexplore.exe, it collects vital system information like the machine's NetBIOS name, logged in user's name etc and then encrypts and sends this information as part of HTTP stub data. In response to this, the server can issue back a variety of commands as explained above.

CnC geo location:

At the moment, I can see two CnC servers fully alive.

91.142.208.43, located in Spain; 207.179.75.114, located in USA.

Interesting Facts

1. As I have also mentioned above, one of the critical features of VinSelf is its ability to traverse through non-transparent a.k.a. browser configured proxies. This is without a doubt showing that this malware was developed keeping corporate networks in mind.

2. This malware is capable of hibernating itself for an extended period of time. For example, if the CnC domain is resolving to 127.0.0.1, it will sleep for 12 hours before trying again. There is a time bomb hidden in the code as well. At start up, VinSelf looks for a file named winfont.cpl under "%SYSTEM32". If found, it will try to read an activation date and time from it and won't activate itself until that time comes. In case this file doesn't exist, VinSelf will activate itself right away.

under the radar for a longer period of time. Here in FireEye labs, we are monitoring VirSeff on 24/7 basis. I will soon be releasing a detailed white paper covering different aspects of this malware in quite a detail.

Atif Mushtaq

Detailed Question/Comments : research {@} fireeye DOT COM

This entry was posted on Tue Nov 23 14:08 EST 2010 and filed under Atif Mushtaq.

Technical Support

Incident?

Report Security Issue

Contact Support

Customer Portal

Communities

Documentation Portal

Site Language
English