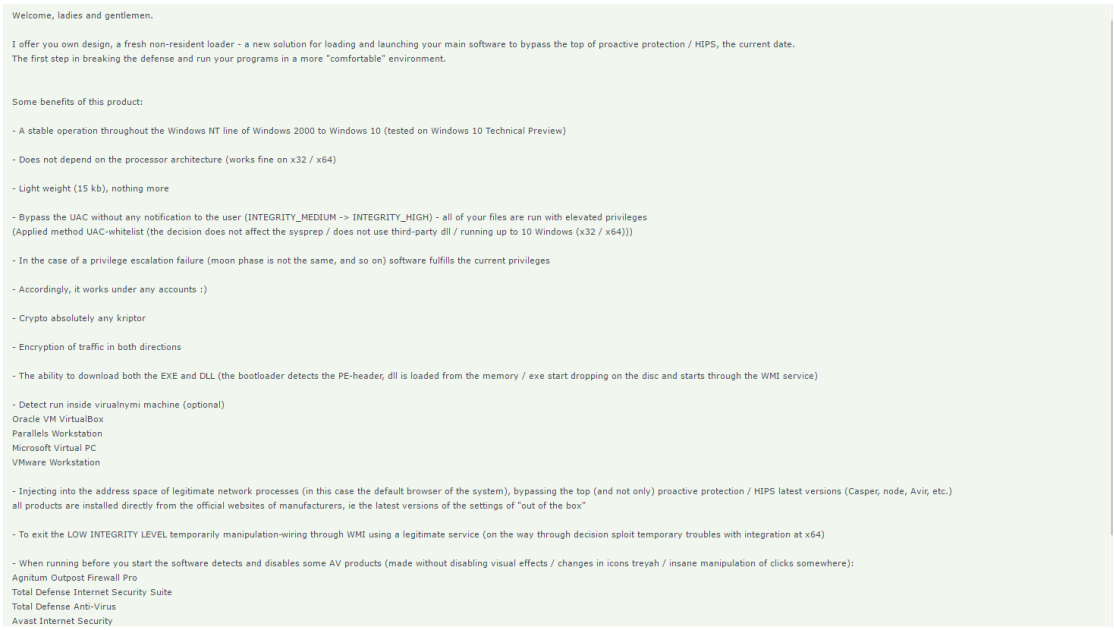
H1N1 LOADER

Nikolaos Pantazopoulos

Introduction

The H1N1 loader is a dropper which is able to load and launch a malware. Its developer has added a lot features on it, features like Virtual Box detection, UAC bypass (Figure 1). Despite the fact that the malware started getting sale on 2015, the developer is updating it until today and sells it in a high price (500$).


Welcome, ladies and gentlemen.

I offer you own design, a fresh non-resident loader - a new solution for loading and launching your main software to bypass the top of proactive protection / HIPS, the current date.
The first step in breaking the defense and run your programs in a more "comfortable" environment.

Some benefits of this product:

- A stable operation throughout the Windows NT line of Windows 2000 to Windows 10 (tested on Windows 10 Technical Preview)

- Does not depend on the processor architecture (works fine on x32 / x64)

- Light weight (15 kb), nothing more

- Bypass the UAC without any notification to the user (INTEGRITY_MEDIUM -> INTEGRITY_HIGH) - all of your files are run with elevated privileges
(Applied method UAC-whitelist (the decision does not affect the sysprep / does not use third-party dll / running up to 10 Windows (x32 / x64)))

- In the case of a privilege escalation failure (moon phase is not the same, and so on) software fulfills the current privileges

- Accordingly, it works under any accounts :)

- Crypto absolutely any kriptor

- Encryption of traffic in both directions

- The ability to download both the EXE and DLL (the bootloader detects the PE-header, dll is loaded from the memory / exe start dropping on the disc and starts through the WMI service)

- Detect run inside virualnymi machine (optional)
Oracle VM VirtualBox
Parallels Workstation
Microsoft Virtual PC
VMware Workstation

- Injecting into the address space of legitimate network processes (in this case the default browser of the system), bypassing the top (and not only) proactive protection / HIPS latest versions (Casper, node, Avir, etc.)
all products are installed directly from the official websites of manufacturers, ie the latest versions of the settings of "out of the box"

- To exit the LOW INTEGRITY LEVEL temporarily manipulation-wiring through WMI using a legitimate service (on the way through decision sploit temporary troubles with integration at x64)

- When running before you start the software detects and disables some AV products (made without disabling visual effects / changes in icons treyah / insane manipulation of clicks somewhere):
Agnitum Outpost Firewall Pro
Total Defense Internet Security Suite
Total Defense Anti-Virus
Avast Internet Security

Figure 1. Sales Thread

Malware File Details

All the hashes are in MD5 algorithm.

H1N1 loader (packed): 6319cd4f40633d91287b0b1ba8e15724

H1N1 loader (unpacked): 0a911aebb309fca049117b1c35cef50d

Main Payload: 96ea83325ad478504cdeafe8d5538294

Main Payload (unpacked): b57ffdcc8bdee01b71c955375e2e9b68

Unpacking

The process of unpacking is fast and simple. Open the malware sample in the Ollydbg and set a breakpoint in the CreateProcessA function. Looking in the *Registers* windows, the ESI

register shows that a possible new executable has been mapped in the memory (Figure2).Save the memory data into a file and open the unpacked file in the Ollydbg.

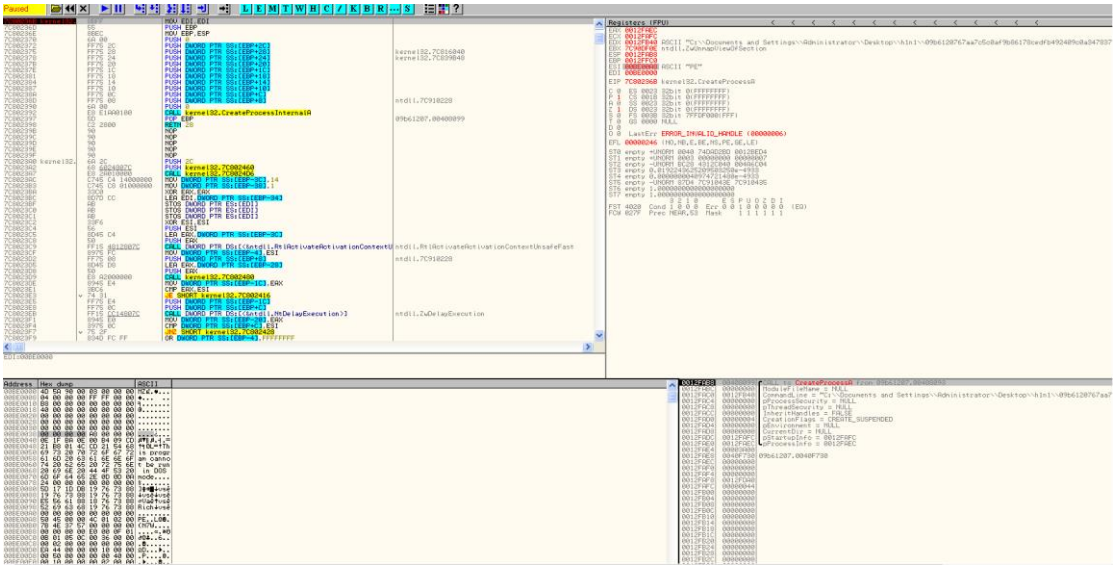


Figure 2. Unpacked

Looking for the payload

Having opened the unpacked loader in the Ollydbg we step over until we reach this point (Figure 3). On this loop, a XOR action is being performed in order to decrypt a new executable in the memory which is the payload (Figure 4).

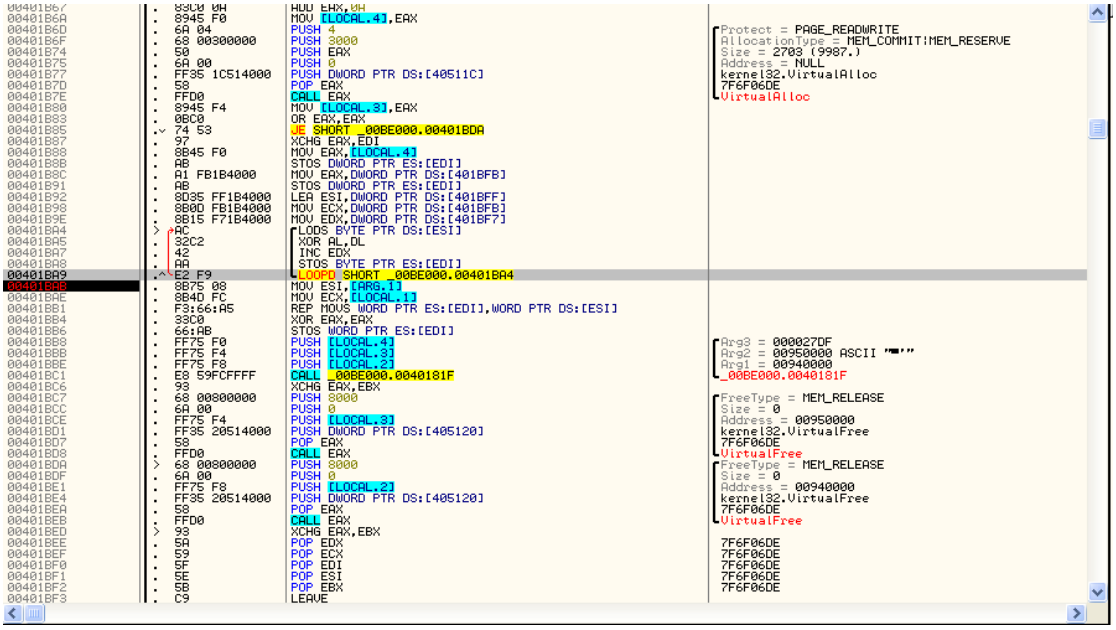


Figure 3

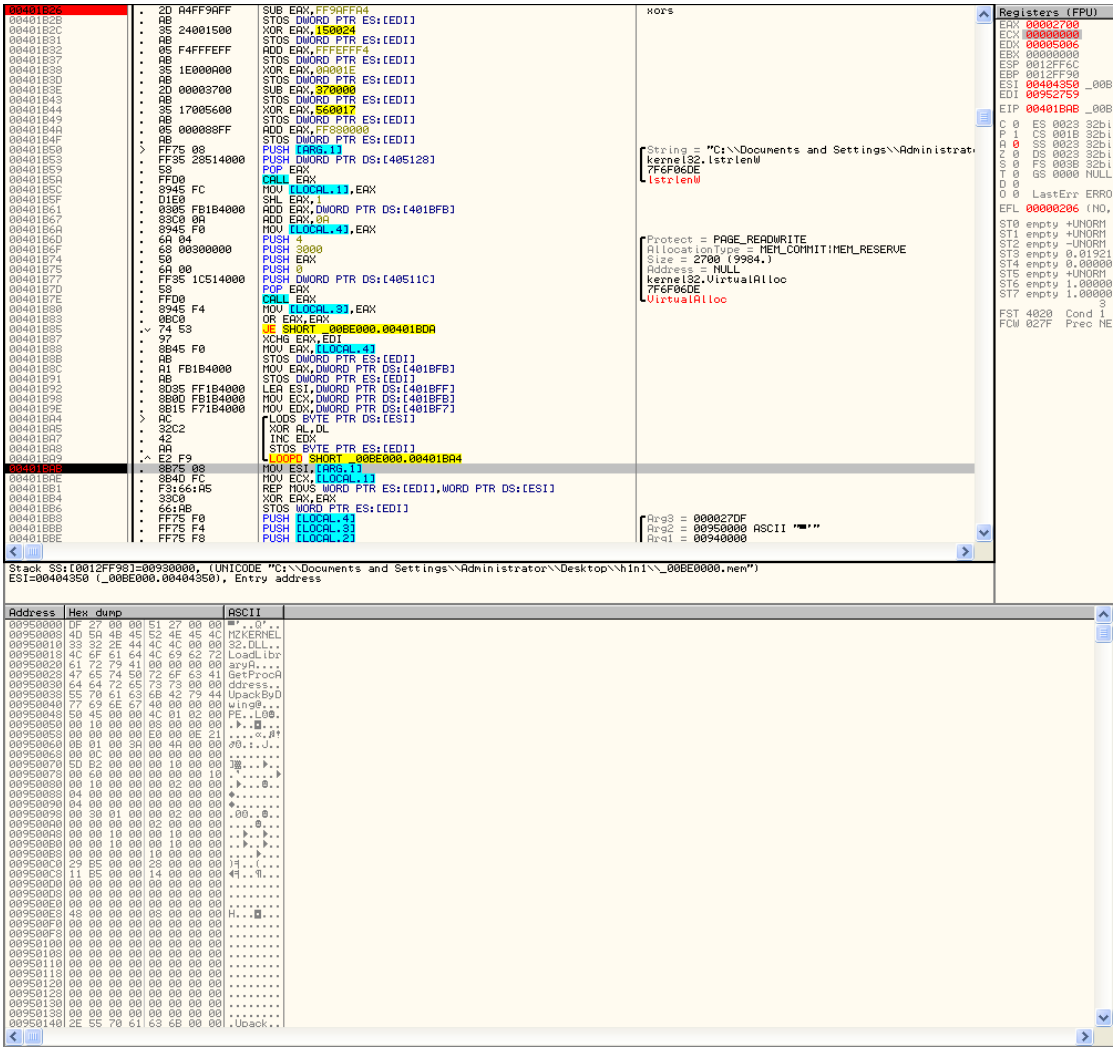


Figure 4. After the XOR decryption

A noticeable thing here is the missing ‘This program cannot be run in DOS mode’ which is not needed for a PE (Portable Executable) to run.

Windows Vista and later

The H1N1 loader uses a few tricks in order to bypass UAC (User Account Control) which are going to be discussed. The loader calls the *GetVersionEx* to identify the Operating System version, if the OS is Vista or higher then it will use two ways for the UAC bypass. After the OS check, the first thing that the loader does is to identify the integrity level of the process (Figure 5).

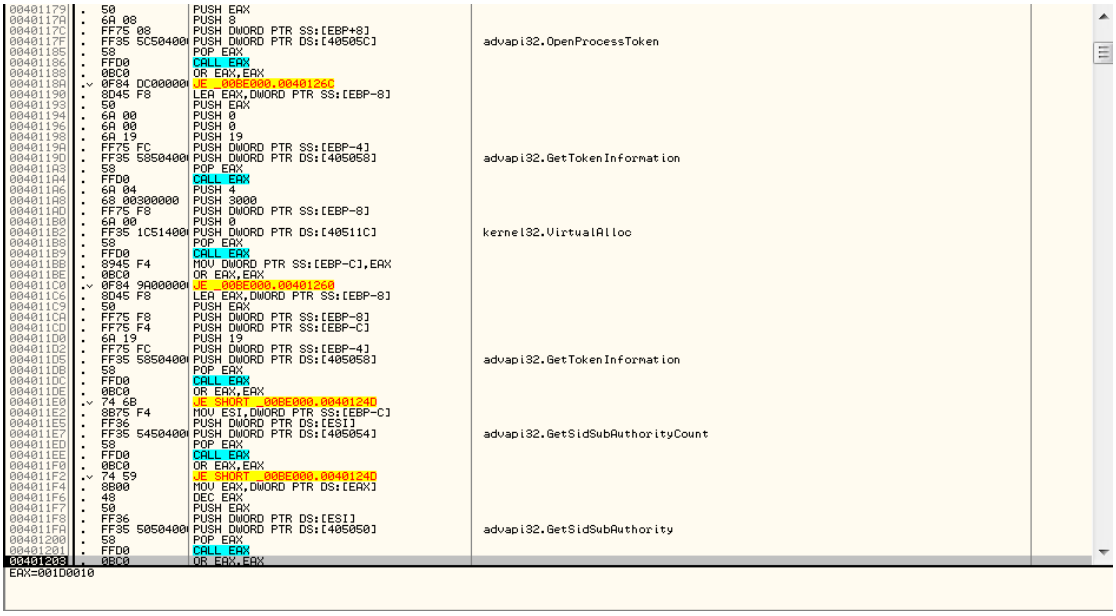


Figure 5. Integrity level check

If the integrity level of the process is not higher or equal of the SECURITY_MANDATORY_MEDIUM_RID level then it will load the *ShellExecuteExW* function and will execute the malware with the WMIC.exe by using ‘runas wmic process call create’ as argument(Figure 6).

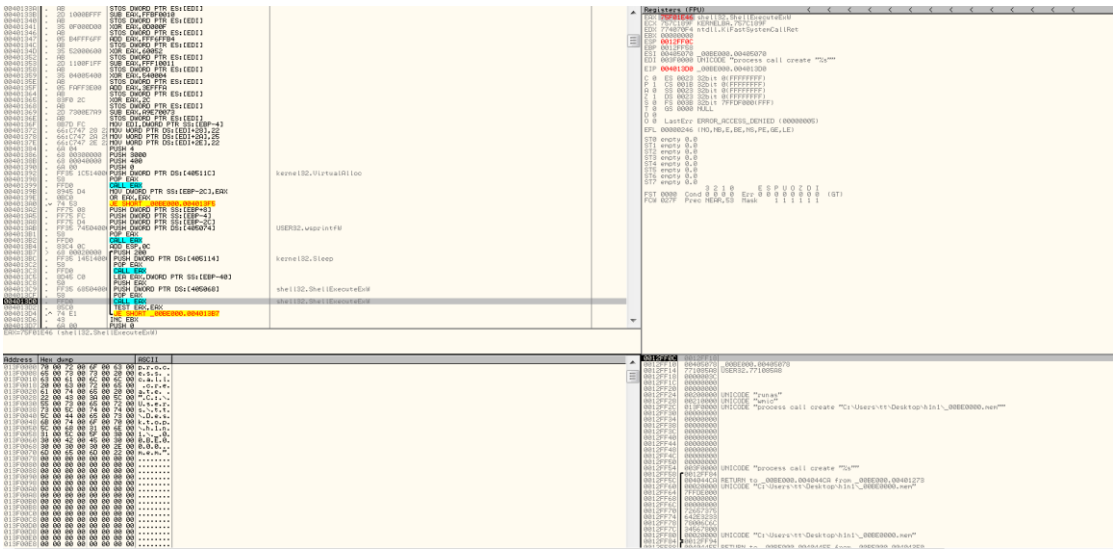


Figure 6. ShellExecuteW execution with parameters

This command will ask from the user to run the malware with privileged rights (Figure 7). The whole trick is to make an unexperienced user click ‘Yes’. However, looking in the details, it is obvious what the malware is trying to do (Figure 8).

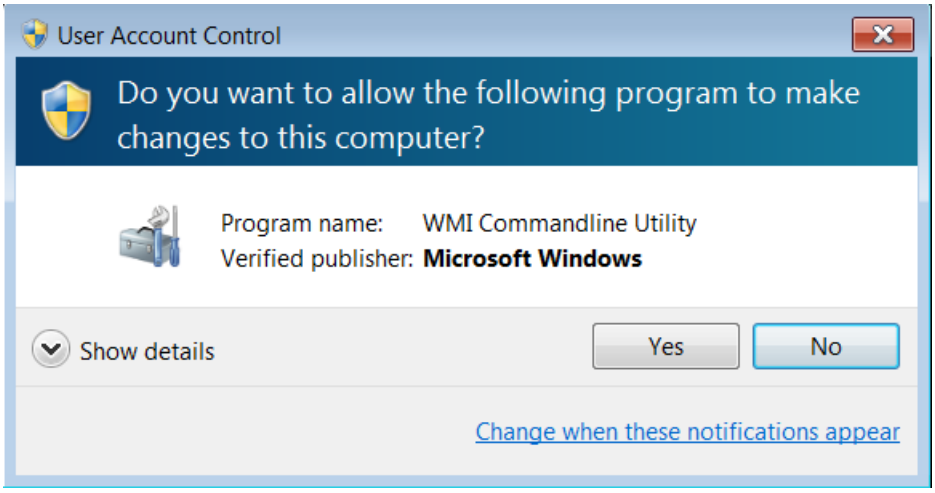


Figure 7

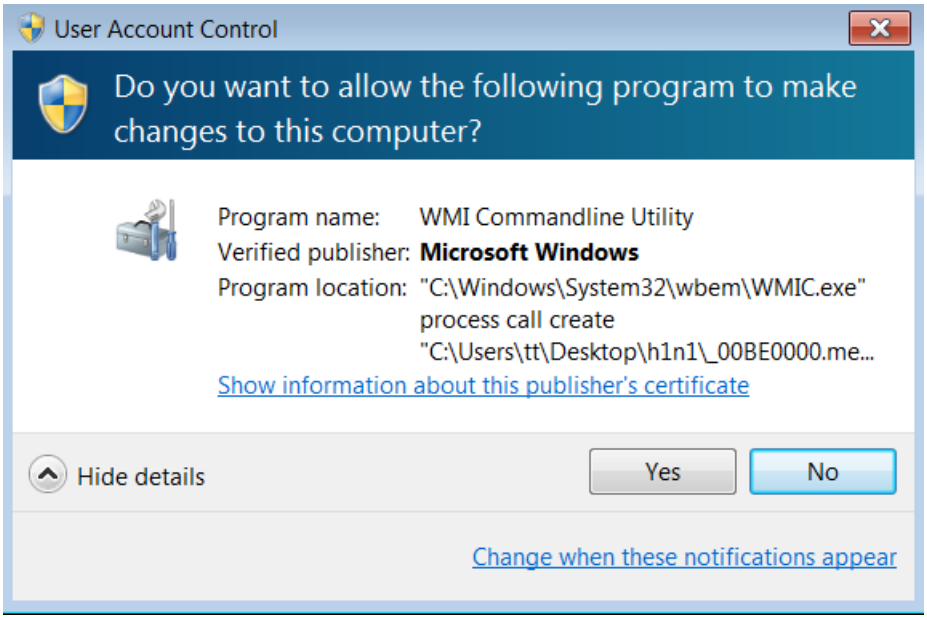


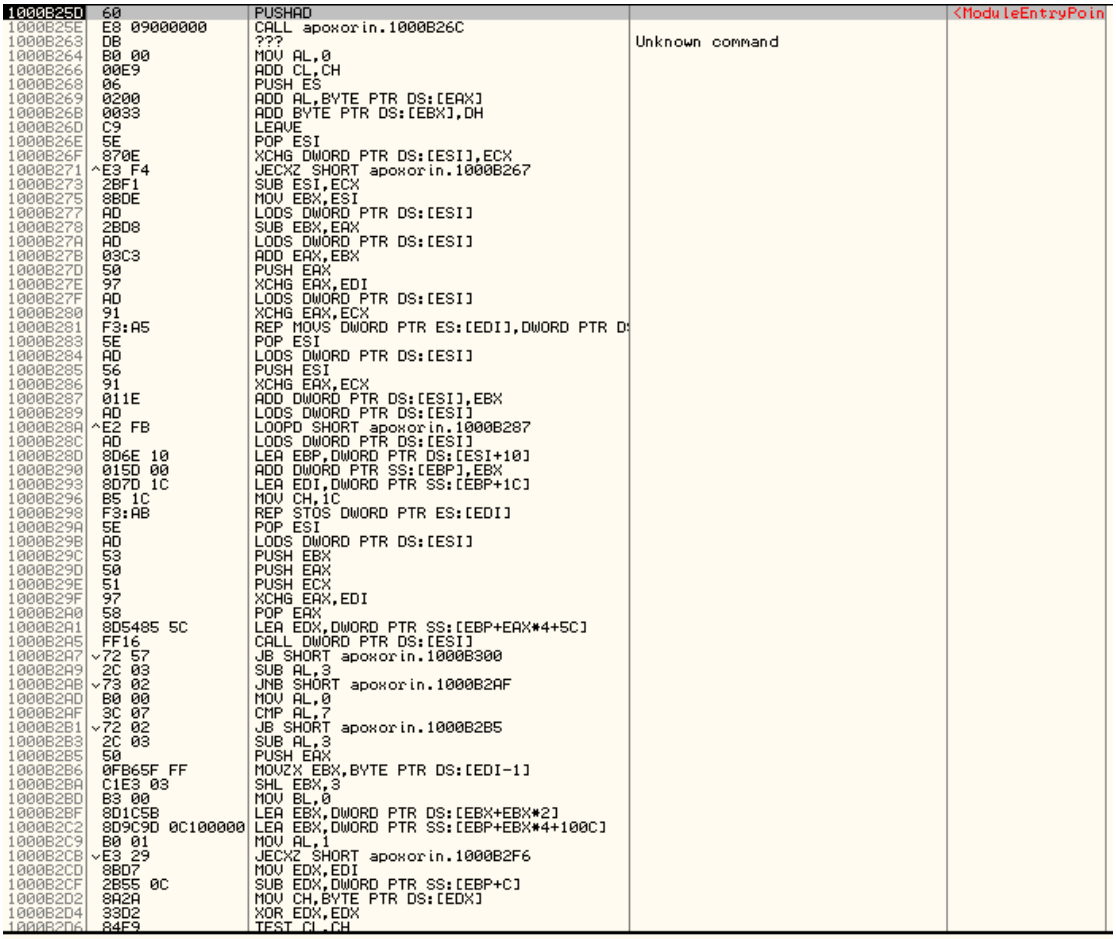
Figure 8

On the other hand, if the integrity level of the process is at least SECURITY_MANDATORY_MEDIUM_RID then it will create a new explorer.exe process in suspend mode (Figure 9) and will inject the payload into it which is decrypted in the XOR loop (Figure 4). The injection is done by using a method that Duqu used. (See the reference links). After the injection, the payload will use the WUSA method in order to bypass UAC (See the next section).

Figure 9.Create new process in suspend mode

Inside the payload

Opening the payload on OllyDbg, the entry point of it looks suspicious and packed (Figure 10). The unpacking process is similar to the UPX. After stepping over the first instruction, follow in dump the ESP (stack pointer register) and set a hardware on access (Dword) breakpoint, execute the malware, OllyDbg has stopped on the original entry point (Figure 11).



```
1000B25D  60                PUSHAD                                   <ModuleEntryPoin
1000B25E  E8 09000000       CALL apoxorin.1000B26C
1000B263  DB                ???                                      Unknown command
1000B264  B0 00             MOV AL,0
1000B266  00E9              ADD CL,CH
1000B268  06                PUSH ES
1000B269  0200              ADD AL,BYTE PTR DS:[EAX]
1000B26B  0033              ADD BYTE PTR DS:[EBX],DH
1000B26D  C9                LEAVE
1000B26E  5E                POP ESI
1000B26F  870E              XCHG DWORD PTR DS:[ESI],ECX
1000B271 ^E3 F4             JECXZ SHORT apoxorin.1000B267
1000B273  2BF1              SUB ESI,ECX
1000B275  8BDE              MOV EBX,ESI
1000B277  AD                LODS DWORD PTR DS:[ESI]
1000B278  2BD8              SUB EBX,EAX
1000B27A  AD                LODS DWORD PTR DS:[ESI]
1000B27B  03C3              ADD EAX,EBX
1000B27D  50                PUSH EAX
1000B27E  97                XCHG EAX,EDI
1000B27F  AD                LODS DWORD PTR DS:[ESI]
1000B280  91                XCHG EAX,ECX
1000B281  F3:A5             REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS
1000B283  5E                POP ESI
1000B284  AD                LODS DWORD PTR DS:[ESI]
1000B285  56                PUSH ESI
1000B286  91                XCHG EAX,ECX
1000B287  011E              ADD DWORD PTR DS:[ESI],EBX
1000B289  AD                LODS DWORD PTR DS:[ESI]
1000B28A ^E2 FB             LOOPD SHORT apoxorin.1000B287
1000B28C  AD                LODS DWORD PTR DS:[ESI]
1000B28D  8D6E 10           LEA EBP,DWORD PTR DS:[ESI+10]
1000B290  015D 00           ADD DWORD PTR SS:[EBP],EBX
1000B293  8D7D 1C           LEA EDI,DWORD PTR SS:[EBP+1C]
1000B296  B5 1C             MOV CH,1C
1000B298  F3:AB             REP STOS DWORD PTR ES:[EDI]
1000B29A  5E                POP ESI
1000B29B  AD                LODS DWORD PTR DS:[ESI]
1000B29C  53                PUSH EBX
1000B29D  50                PUSH EAX
1000B29E  51                PUSH ECX
1000B29F  97                XCHG EAX,EDI
1000B2A0  58                POP EAX
1000B2A1  8D5485 5C         LEA EDX,DWORD PTR SS:[EBP+EAX*4+5C]
1000B2A5  FF16              CALL DWORD PTR DS:[ESI]
1000B2A7 v72 57             JB SHORT apoxorin.1000B300
1000B2A9  2C 03             SUB AL,3
1000B2AB v73 02             JNB SHORT apoxorin.1000B2AF
1000B2AD  B0 00             MOV AL,0
1000B2AF  3C 07             CMP AL,7
1000B2B1 v72 02             JB SHORT apoxorin.1000B2B5
1000B2B3  2C 03             SUB AL,3
1000B2B5  50                PUSH EAX
1000B2B6  0FB65F FF         MOVZX EBX,BYTE PTR DS:[EDI-1]
1000B2BA  C1E3 03           SHL EBX,3
1000B2BD  B3 00             MOV BL,0
1000B2BF  8D1C5B            LEA EBX,DWORD PTR DS:[EBX+EBX*2]
1000B2C2  8D9C9D 0C100000   LEA EBX,DWORD PTR SS:[EBP+EBX*4+100C]
1000B2C9  B0 01             MOV AL,1
1000B2CB v E3 29            JECXZ SHORT apoxorin.1000B2F6
1000B2CD  8BD7              MOV EDX,EDI
1000B2CF  2B55 0C           SUB EDX,DWORD PTR SS:[EBP+C]
1000B2D2  8A2A              MOV CH,BYTE PTR DS:[EDX]
1000B2D4  33D2              XOR EDX,EDX
1000B2D6  84E9              TEST CL,CH
```

Figure 10. Entry Point

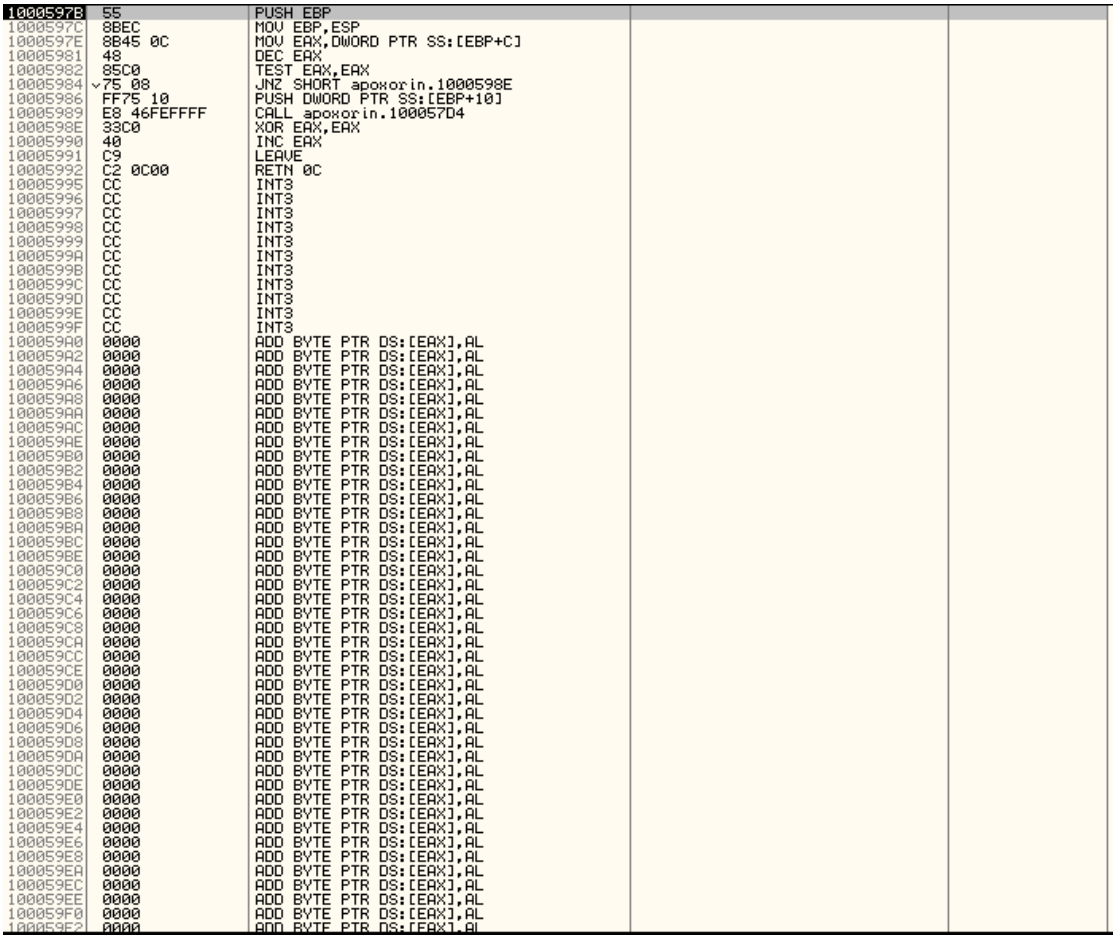


Figure 11. Original entry point

Doing a string search, it becomes obvious that the malware API calls and strings are encrypted. As shown in figure 12, in order to make API calls, the malware doesn't use the GetProcAddress WINAPI, instead it finds manually the appropriate dll Image base address by using the PEB (Process Environment Block). It is important to note that a similar technique has been used by the Zeus botnet and Carberp [1] [2] [3].

Figure 12

In general, the malware will locate the image base address of kernel32.dll and ntdll.dll and will find the functions from the export table (Figure 13) [4].

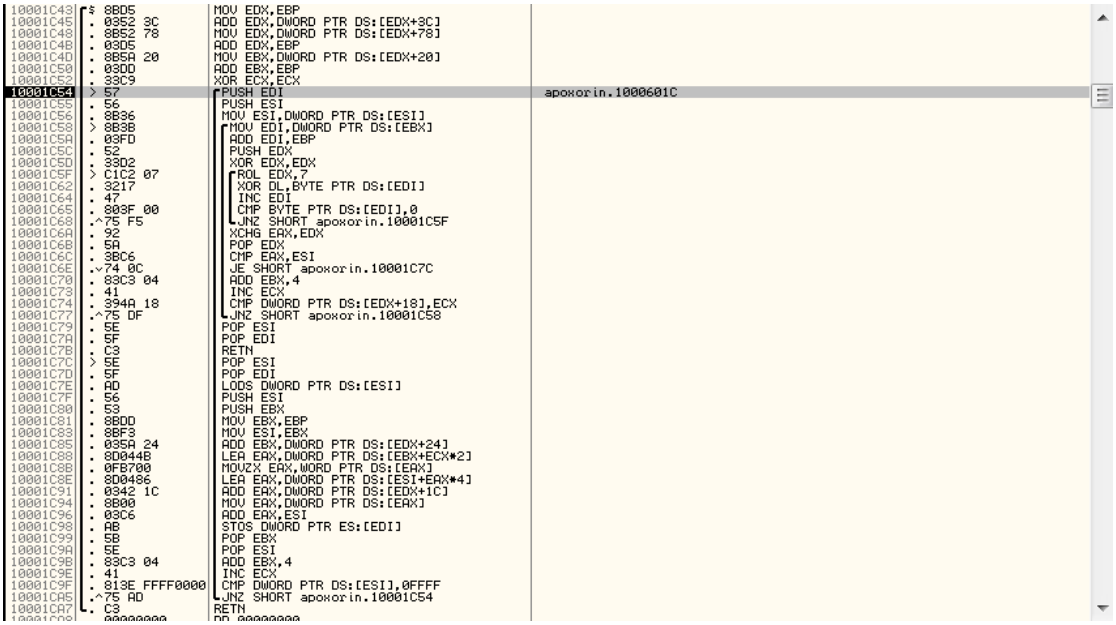


Figure 13

Furthermore, the malware will use the Loadlibrary WINAPI to load the advapi32.dll, shell32.dll, user32.dll, urlmon.dll, wininet.dll, crypt32.dll and will use the same method as before to read the export table. After stepping over a few instructions, step in to the 10004F12 call (See Figure 14), that call is responsible for decrypting the config of the malware and to send HTTP requests to the C&C server. As is shown in figure 14, the VirtualAlloc WINAPI is being called, the allocated memory will store the config file (Figure 15).

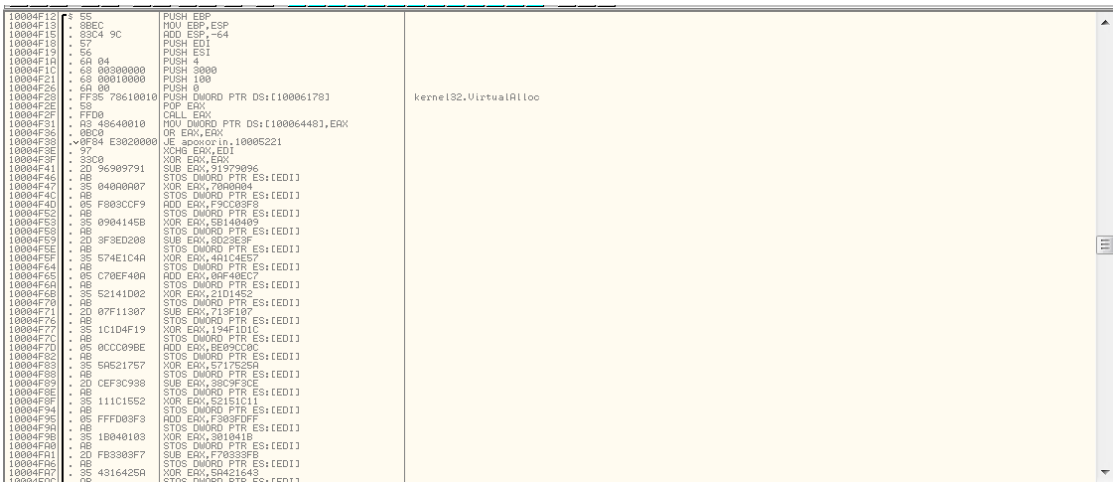


Figure 14

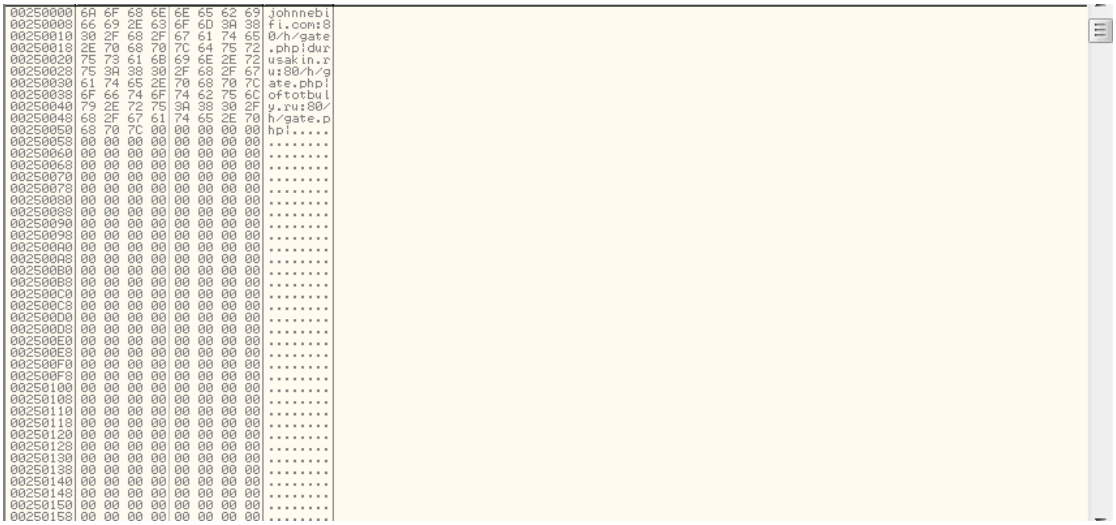


Figure 15

Having send requests to the C&C servers, the malware will check if the process has admin rights. To achieve that goal, it will use the CheckTokenMembership WINAPI [5] (Figure 16).

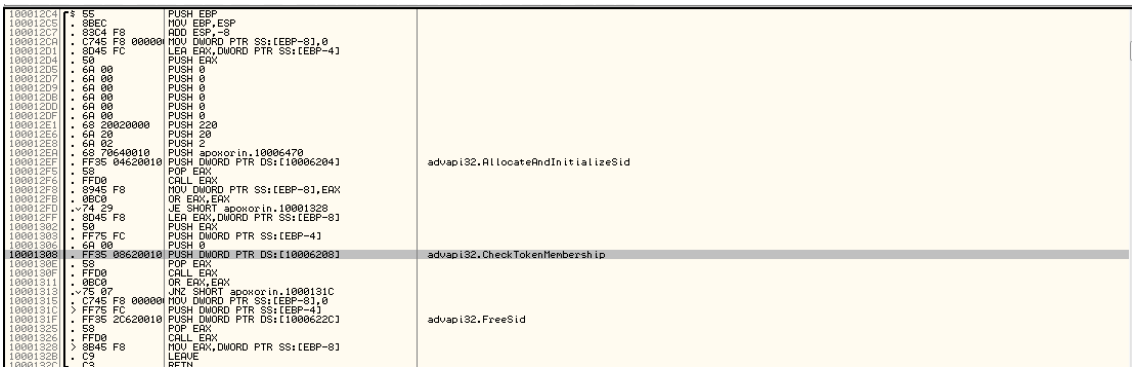


Figure 16

Firefox Credentials Stealer

H1N1 loader comes with browser stealer functions. Firstly, it will try to steal credentials from the Firefox browser (after checking if Firefox is installed). The stealing process starts by loading the Firefox module nss3.dll (This particularly library needs msvcr120.dll and mozglue.dll in order to be loaded) (Figure 17). Next, it will search the logins.json file in all Firefox profiles (%APPDATA%\Mozilla\Firefox\Profiles*.*\). The logins.json file is responsible for keeping all the saved passwords (encrypted) (Figure 18). H1N1 overcomes the encryption problem by using the PK11SDR_Decrypt function (exported from nss3.dll with

the method described above) (Figure 19). The problem is that Firefox encrypts the saved credentials with the *PK11SDR_Encrypt* function which uses a blank password.

Having already located the logins.json file, the next step is to read the file and get the values of the 'hostname', 'Encrypteduser name', 'Encryptedpassword'. In figure 20, it is shown that the malware has finished the decryption successfully.

```
EAX 752C499F kernel32.LoadLibraryA
ECX 7560ADF4 KERNELBA.7560ADF4
EDX 00290001 ASCII ":\Program Files (x86)\Mozilla Firefox\msvcr120.dll"
EBX 00000000
ESP 0018FD44
EBP 0018FD5C
ESI 10006308 apoxorin.10006308
EDI 00290036
```

Figure 17

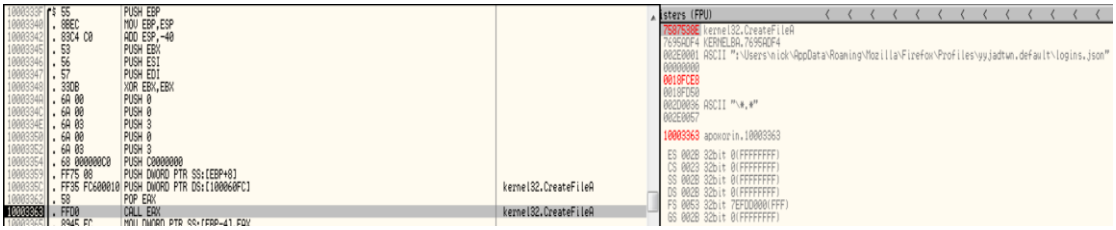


Figure 18

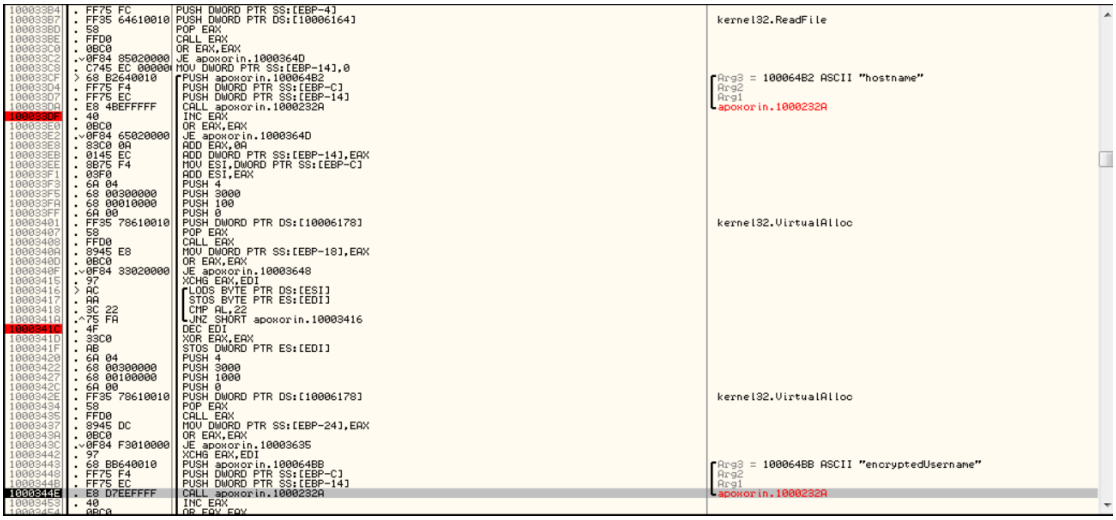


Figure 19

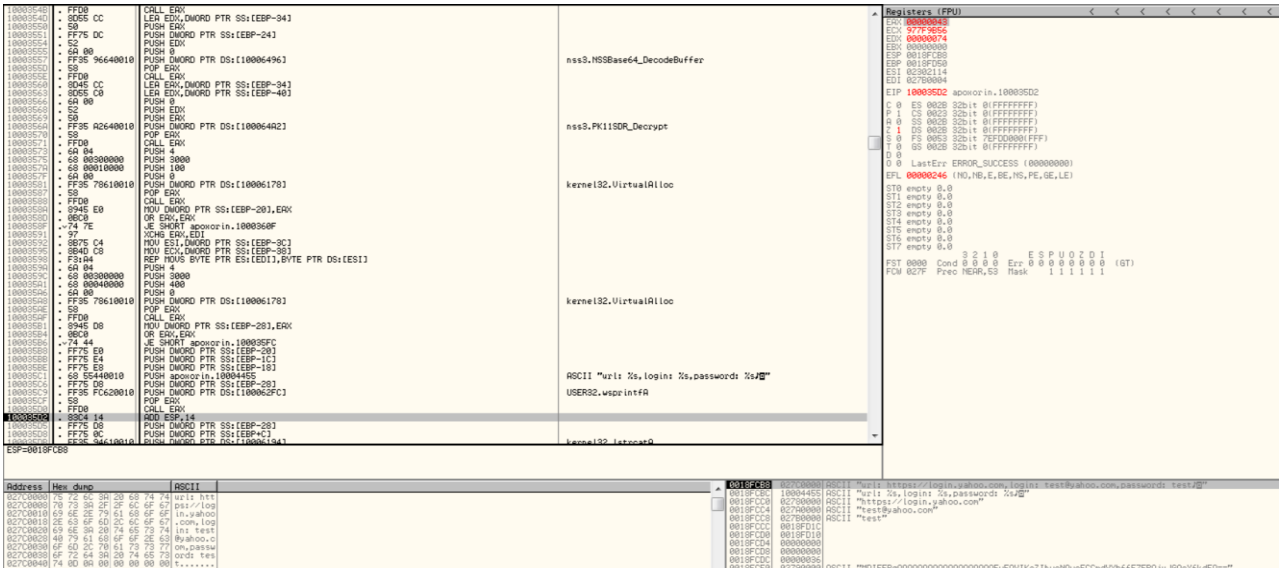


Figure 20

C&C Communication

H1N1 loader communicates with its C&C server by using common WINAPI calls like 'InternetOpenA'. The request which is going to be sent to the server contains information for the OS version, GUID, OS architecture (x86 or x64), the permissions of the process, browsers credentials, and emails credentials. The OS version is obtained using the *GetVersionEx*() WINAPI call. The OS architecture will be obtained by using the *IsWow64Process*() and the *GetSystemInfo*() WINAPI calls. The permissions of the process will be retrieved with the method that described before. The GUID will be calculated by calling the *GetVolumeInformation*() WINAPI call in order to get the serial number of the hard drive, then the result of it is going to be XORed with the hard coded value 0x0BADC0DE (Figure 21).

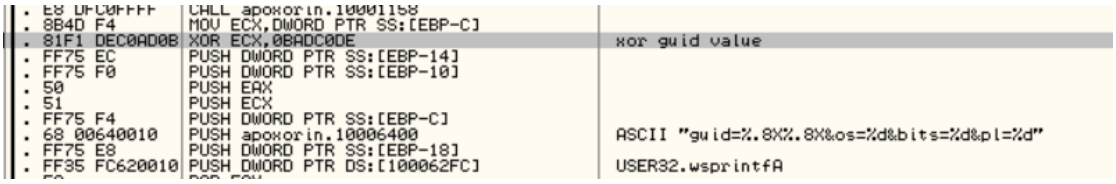


Figure 21

Lastly, the information will be encrypted using RC4 and then encode it with base64. The key for the RC4 encryption is hard-coded and obfuscated (Figure 22 and 23). As for the base64 encoding, the malware uses the URL-safe Base64 encoding which means that the '+' characters will be replaced with '-'and the '/' with '_' [6]. The final result will be like the string in figure 24.

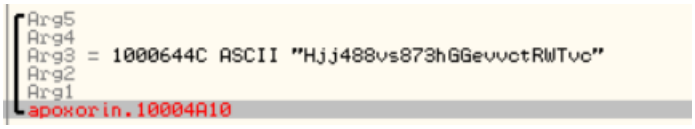


Figure 22

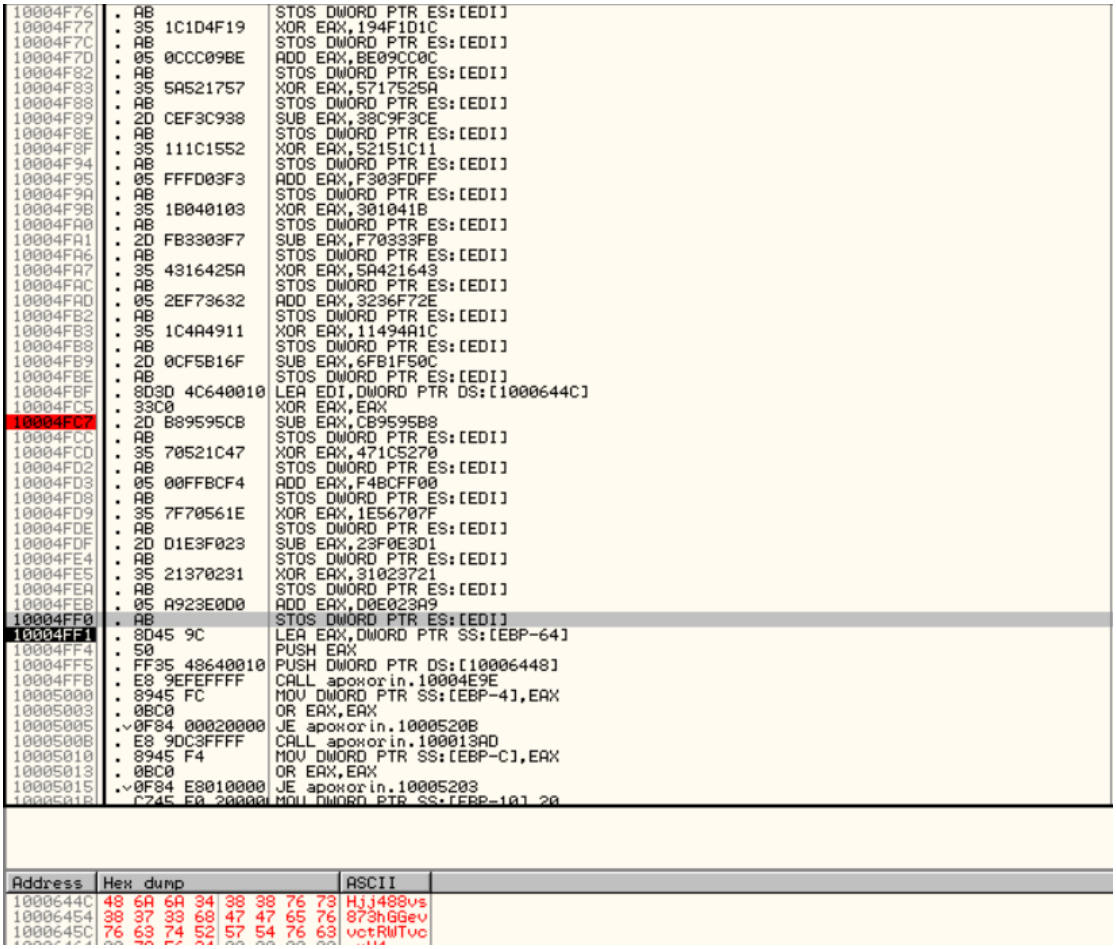


Figure 23

```
"HabRiev3Tve2glxZSlWmuuyTRruuM7MeutGOiRO23JaujGehE5gO9WaOKYRStQM6KbLNcZ5Bq4="
```

Figure 24

References

[1]. http://interestingmalware.blogspot.co.uk/2010/07/find-base-address-of-kernel32dll.html

[2]. http://blog.harmonysecurity.com/2009/06/retrieving-kernel32s-base-address.html

[3]. https://github.com/hzeroo/Carberp/blob/master/source%20-%20absource/pro/all%20source/Locker/src/getapi.cpp

[4]. http://www.rohitab.com/discuss/topic/38717-quick-tutorial-finding-kernel32-base-and-walking-its-export-table/

[5]. https://forum.tuts4you.com/blogs/entry/147-isuseranadministrator/

[6]. http://www.komeil.com/toolbox/base64encoder

[7]. http://blog.w4kfu.com/tag/duqu