BackSwap Banker Malware Hides Inside Replicas of Legitimate Programs

By sharon

Published: 2018-08-06 · Archived: 2026-04-05 14:56:07 UTC

This post will focus on a deep dive analysis of how the BackSwap banker malware hides its malicious code inside replicas of popular, legitimate computer programs. The Cyberbit Malware Research team analyzed four samples of this banking malware. The fourth sample is a new variant that has not been previously analyzed and was found to contain an interesting method for decrypting its code during runtime.

The recent research on BackSwap banker published by ESET and CERT.PL, uncovered BackSwap was using a technique that hooks the Windows message loop events to look for banking activity, instead of injecting code directly into the browser's process.

We will take a deeper look at another interesting property of this banking malware, the masquerading technique it uses to hide its code inside malicious replicas of legitimate computer programs like FileZilla, 7-zip, OllyDbg and WinGraph32. These maliciously-altered copies of real programs will appear completely legitimate to users during download.

BackSwap plants its code in the initialization phase of the program, in an early stage of the program execution, replacing the normal flow with its malicious instructions. The program will *not* return to its normal execution after the malicious code begins running, in each variant we have seen the code was placed in a different part of the program, making detection and analysis challenging.

Analysis of 4 BackSwap Banking Malware Samples

Sample 1: 7-Zip

SHA256: 16fe4de2235850a7d947e4517a667a9bfcca3aee17b5022b02c68cc584aa6548

This malware sample masquerades itself as the popular free and open-source file compression and archiver program "7-Zip". The icon is identical to the original program, as well as in the other samples infected with this malware. By looking at the properties of this file, nothing looks suspicious:

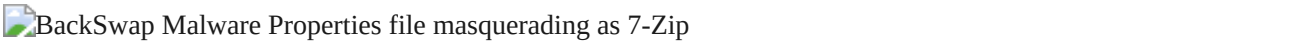

Figure 1 – Properties of the file masquerading as 7-Zip

The _initterm function

This purpose of the _initterm function is to walk through function pointers and execute the functions pointed to by them. For example, it is used to initialize global variables of the program.

Let's look at the function pointers that the original _initterm of the legitimate 7-Zip (version 16.2.0.0) receives:


BackSwap Malware 7zip_initterm

Figure 2 – the function pointers passed to _initterm

Going inside the last pointer, sub_4328f8, we see initialization of variables


BackSwap Malware 7zip inside last pointer

Figure 3 – initialization of global variables in the function pointer passed to _initterm

And it continues:


BackSwap Malware zip inside last pointer

Figure 4 – continue of figure 3 (some lines were truncated)

However, while analyzing the version infected with BackSwap, we can see some malicious code right after the last legitimate command (mov byte_44d5cd, al) instead of the "retn" command:


BackSwap Malware 7zip first malware code

Figure 5 – After the highlighted command, the malicious code begins

Scrolling down a bit, we can see a call to VirtualAlloc with RWX permissions. This hints to us the malware will use it for continuing its execution. If we carefully examine the instructions there are "push" instructions which pass the parameters to the VirtualAlloc.Notice that the push instructions are scattered among many other instructions, which makes the analysis of this malware tricky.

The call and parameters are as follows:

VirtualAlloc(0,0x3e00,0x3000, 0x40)


BackSwap Malware 7zip valloc

Figure 6 – push instructions among the code. In this figure you can see the push instructions of 0x40 (PAGE_EXECUTE_READWRITE), 0x3000 (MEM_COMMIT | MEM_RESERVSE) and 0x3e00 (size of allocation)


BackSwap Malware 7zip_valloc_2

Figure 7 – The last push instruction (0x0 – the optional starting address) is highlighted. The last line in the figure shows the call to VirtualAlloc

The size of allocation is 4 pages (16kb). To find further evidence, we look at a running instance of this malware at process hacker and see this memory region:


BackSwap Malware 7zip phacker

Figure 8 – A running instance of this malware. It first drops itself to the startup folder as "taskslist.exe" and runs it again. The allocated RWX memory region is highlighted.

I will skip the deep drill down into the malware's functionality as it is well documented in the reports we linked to at the top of this post, but let's have a closer look at this code executed from this allocated region:

The first commands indicate that this code is Position-Independent-Code (PIC) as you can see the trick of "call $0" followed by "pop ebx". When using the call instruction, the return address of that call instruction is pushed into the stack. The next pop instruction moves the address at the top of the stack into the ebx register, so the malware knows where it is in the process' memory.


BackSwap Malware 7zip PIC

Figure 9 – Retrieving EIP

We can also see the malware trying to retrieve the address of kernel32.dll from the PEB, since this is a PIC and it doesn't know where the windows libraries are loaded.


BackSwap Malware 7zip PEB

Figure 10 – Checking the PEB for the address of kernel32.dll

Sample 2: FileZilla

SHA256: 2223a93521b261715767f00f0d1ae4e692bd593202be40f3508cb4fd5e21712b

This sample masquerades itself as the free cross-platform FTP application, program "FileZilla". One of the virtual functions of a class called "CWinThread" is patched. During the program initialization, in wWinMain function, there is an initialization of this class using AfxGetThread() – which returns a pointer to the CWinThread object that represents the currently executing thread. After that, one of the virtual functions in that class, at offset 0x50, is executed.

Let's have a look at the WinMain function of the original, clean FileZilla:


BackSwap Malware FileZilla wWinMain

Figure 11 – CWinThread is initialized and one of its virtual functions at offset 0x50 is called

From our dynamic analysis, we identified that the executed function resides at 0x410935, which, as expected is in offset 0x50 from the start of the vtable of the CWinThread class. This can be computed, as 0x4BAB0C – 0x4BAABC = 0x50


BackSwap Malware filezilla vtable1

Figure 12 – the vtable of the CWinThread class

Now let's compare the function that starts 0x410935 at the original FileZilla and at the infected FileZilla

Figure 13 shows the clean FileZilla, where there is a call to WSAStartup.

Figure 13 – part of the function at 0x410935 of the original clean FileZilla

Figure 14 shows that the malicious code starts right after "push 101h" instruction, without calling WSAStartup

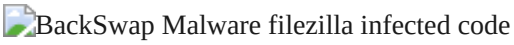

Figure 14 – *part of the function at 0x410935 of the infected FileZilla file, instead of calling WSAStartup, the malicious code takes place*

Another thing to note in this banking malware, compared to the previous sample, is the use of GetProcAddress to get the address of VirtualAlloc, instead of calling it directly. The parameters for VirtualAlloc, in this case, remain the same.

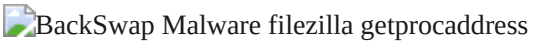

Figure 15 – Using GetProcAddress to get the address of VirtualAlloc. On the last line you can see the "push 40h" command which is the pages' protection parameter – RWX in this case.

Sample 3: OllyDbg

SHA256: f51336e862b891f78f2682505c3d38ea7de5b0673d6ef7a3b0907c0996887c22

This BackSwap malware sample masquerades itself as OllyDbg, the x86 debugger. This is a bit surprising as this program is a favorite of security researchers who use it for dynamic code analysis. Even the most security-aware users are known to obtain unsigned tools from download sites.

This time, the malicious code hides in the WinMain function. In the original clean OllyDbg – the WinMain functions calls another function, CreateWindowExA (Figure 16).

Figure 16 – The clean version of OllyDbg contains a call to CreateWindowExA

In the infected version of OllyDbg, the parameters for CreateWindowExA are pushed into the stack but instead of calling CreateWindowExA, the malicious code starts its execution (Figure 17).

Figure 17 – Instead of calling CreateWindowExA – the malicious code executes

Sample 4: wingraph plugin

SHA256: 6bb85a033a446976123b9aecf57155e1dd832fa4a7059013897c84833f8fbcf7

This sample masquerades itself as the program wingraph32. This sample is the newest we observed and was uploaded to VirusTotal on July 11th, 2018 at 7:14:24.

This time the malicious code is triggered by the call to _init_exit_proc which is also called at the initialization phase of the program. _init_exit_proc executes functions according to the pointer it receives. Functions are executed in a loop according to the pointer which is incremented in each step. One of the functions that _init_exit_proc calls was patched. The address of the function patched is 0x497a38.


BackSwap Malware wingraph init exit proc

Figure 18 – There are 2 loops in _init_exit_proc that go over function pointers. One of the functions executed in the second loop is the one patched by the malware


BackSwap Malware wingraph original

Figure 19 – the function at 0x497a38 of the unpatched version of wingraph, notice that after the "sub" command there is a "retn" command.


BackSwap Malware wingraph infected

Figure 20 – instead of the "retn" command, the malicious code executes

In this sample the allocation size of the first VirtualAlloc by the malicious code is different. It is 0x5912 bytes which translates to allocation of 6 pages (24KB).

In this sample, the malware uses additional protection, by copying an encrypted code to the allocated memory region, and then decrypting it.


BackSwap Malware wingraph algo

Figure 21 – The algorithm used to decrypt the encrypted code (0x582dc3 to 0x582e0e)


BackSwap Malwrae wingraph algo

Figure 22 – The encrypted code, before decryption


BackSwap Malware wingraph algo

Figure 23 – The encrypted code, after decryption, notice the "call $0, pop ebx" commands which are identical to the commands in the previous samples and indicate that this code is position independent.

The decryption algorithm can be translated to the following pseudo-code:


BackSwap Malware wingraph pseudo code

Figure 24 – The pseudo code of the decryption algorithm

Protecting Against BackSwap Malware

Our analysis shows that the BackSwap malware is hidden inside various legitimate and popular programs, often changing its location within the program. The BackSwap malware code also varies from sample to sample and uses different obfuscation techniques each time. We suspect the authors used an automatic tool to infect these

programs with the malware. We expect this masquerading technique to be used in more malware in the future and therefore recommend the following precautions:

- Download software only from official publisher website
- Download open source software only from the original verified GitHub account and verify it is signed by the official publisher before running.

Hod Gavriel is a malware analyst at Cyberbit.

Boris Erbesfeld is a principal software engineer at Cyberbit.

Learn more about Cyberbit EDR Kernel-Based Endpoint Detection vs. Whitelisting

---