**Packet Capture Analysis**

(how-to-commit-crimes.docx)Graphical user interface, text, application

Description automatically generated

I opened the .pcap file on WireShark and looked at the three panes:

1. Packet List
2. Packet Details
3. Packet Bytes

I focused my attention on the packet list pane and configured the display to highlight HTTP network protocol as blue. I navigated from ‘View’ to ‘Colouring Rules’ and chose blue as the background.

Graphical user interface, application

Description automatically generated

To filter for only HTTP requests and responses, I typed in the display filter: ‘http.request’ or ‘HTTP’. Also, although all HTTP traffic is displayed, the focus should be on websites that were visited. To view only websites that were accessed, you can also type the following filter command: ‘http.request.method == “GET”’

Breakdown:

* ‘http.request.method’ to look for what the user is requesting to see
* ‘== “GET”’ states to only return the GET method, it is used to get HTML data from the web server so the user’s browser can display the information

Graphical user interface, text, application

Description automatically generated

I immediately noticed in the ‘Source Port’ column displayed the same source and destination ports: 1, meaning the address from which the specific packet came from and the address to which the particular packet is going to is the same.

The ‘Length’ column measures the packets in bytes, and I did not find any immediate signs of malicious activity from the information provided.

However, the ‘Info’ column provides additional information about the packet, which is typically dependent on the protocol used. The ‘Info’ column was the most informative by displaying names that alluded to malicious activity (e.g., ‘how-to-commit-crimes.docx’ & ‘evil.pdf’).

I right-clicked on the potentially malicious file, scrolled down to ‘Follow’ and clicked ‘TCP Stream’. WireShark converted the binary data to a readable text using ASCII and confirmed the vulnerability.

Graphical user interface, text, application

Description automatically generated

(evil.pdf)

Graphical user interface, application, Word

Description automatically generated

The second malicious network packet was the ‘Evil.pdf’, found on the ‘Info’ column on WireShark. The name states it is a .pdf file and I confirmed it is by looking at the ‘TCP Stream’ and scrolled to the ‘Content-Type’, result: “application/pdf”. Further, I showed the data as ‘Raw’, searched for the pdf’s file signature (25 50 44 46 2d) and copy and pasted it onto Hex editor, HxD. I saved the file and opened it with my web browser (because it is .pdf compatible) and found the results above.

Sub-task 1:

Logo

Description automatically generatedGraphical user interface, text

Description automatically generated

Sub-task 2:

The file signature of a .jpeg file is ‘ff d8’ and ends with ‘ff d9’.

Clicking ‘TCP Stream’, choosing to show ‘Raw’ data, I copied the information between ‘ff d8’ and ‘ff d9’, pasted it onto HxD and saved the file. I used Paint to open the image, and the content matched the name displayed on WireShark.

WireShark discovered the cryptographic technique of steganography by placing a hidden message within a file or image. This is important because of the possibility of an insider trading secret information. In regards to a bank, the insider could be trading/selling customer identity to the black market. The hacker may have used ‘Steghide’, a forensive command-line tool used to apply steganography techniques.

Results:

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Difference: You’ve found a hidden message in this file! Include it in your write up

Text

Description automatically generatedA picture containing text

Description automatically generated

Difference: You’ve found the hidden message! Images are sometimes more than they appear

Sub-task 3:

Graphical user interface, text, application

Description automatically generated

Sub-task 4:

The .pdf file signature is ’25 50 44 46 2d’

TCP Stream -> Content-Type: pdf -> show raw data -> copied data between file signature (‘25 50 44 46 2d’) -> pasted on HxD -> saved file -> open on Google Chrome (.pdf compatible)

Results:





Sub-task 5:

A picture containing shirt

Description automatically generated

Content-Type: text/plain; charset=UTF-8

Show data as: UTF-8

According to the ASCII data translation on TCP stream, the JFIF string at the beginning helps me infer the content is an image. However, the title of the network packet alludes to a hidden message. Upon further notice, the content-type is encoded with UTF-8. When de-coded, the result:

Graphical user interface, text, application, email

Description automatically generated

I have read through the content and was unable to find a hidden message. I further tried decoding the UTF-8 information on other software with no success.

Sub-task 6:

A group of people in a room

Description automatically generated

Unlike the jpeg images on sub-task 2, there is no hidden message when data is decoded with ASCII. This teaches cybersecurity professionals to think like a forensic examiner and read the entire traffic message for hidden cryptographic messages known as steganography. However, because the “atm-image” did not have any, the difference about this traffic is the lack steganography.

Sub-task 7:

The .png file has a file signature of ‘0d 0a 0d 0a’

TCP Stream -> show raw data -> copied data between file signature (‘0a 0d 0d 0a’) -> saved file -> opened on Google Chrome

According to sub-task 7, the “broken.png” file is an image. However, based on my research, it is pcapng file because it has the file signature of “0d 0a 0d 0a”. Also, the file signature for png is ’89 50 4E 47 0D 0A 1A 0A’ and I was unable to find the signature on the TCP Stream. Further, I looked for gif (47 49 46 38 37), jpeg ( ffd8 + ffd9), zip (50 4b 03 04) and pdf (25 50 44 46 2d), with no success.

The pcapng is a PCAP file format, meaning it is not an image.

Result:

Graphical user interface, application, Word

Description automatically generated

Sub-task 8:

Graphical user interface, application

Description automatically generated

The file is in a zip format with a signature of ’50 4B 03 04’. After copying and pasting the raw data onto HxD, I saved the file and opened it with Google Chrome. The browser immediately downloaded a zip file (image above) and I extracted it. However, I needed a password and discovered it on TCP Stream using the ASCII format (image below)

Graphical user interface, text, application

Description automatically generated

Afterwards, I typed in the password, successfully extracted the zip file, found the pdf and opened it using Google Chrome.