ANZ: Digital Investigation Using Wireshark

Forage - ANZ Cyber Security Mangement Job Simulation

25/06/2025

Paige Haines

Contents

1 Introduction

1.1 Case Overview

This forensic investigation aims to examine a laptop that has been flagged on security systems as a result of unusual internet traffic. Using a **pcap** file, and **Okteta**, a Linux hex editor, I will investigate the user's network traffic to establish the cause of the suspicious traffic, and what files this user accessed and downloaded.

1.2 Scope

This investigation will be split into eight distinct sections.

- Examine **anz-logo.jpg** and **bank-card.jpg**.
- Examine **ANZ1.jpg** and **ANZ2.jpg**.
- Examine **how-to-commit-crimes.docx**.
- Examine **ANZ_Document.pdf, ANZ_Document2.pdf,** and **evil.pdf**.
- Examine the Contents Within **hiddenmessage2.txt**.
- Examine **atm-image.jpg**.
- Examine **broken.jpg**.
- Examine **securepdf.pdf**.

2 Section 1: Examine anz-logo.jpg and bank-card.jpg.

2.1 Tools Used

- **Wireshark**: a tool that sniffs web traffic for analysis

I started this investigation by opening my Wireshark to analyse the provided **pcap** file.

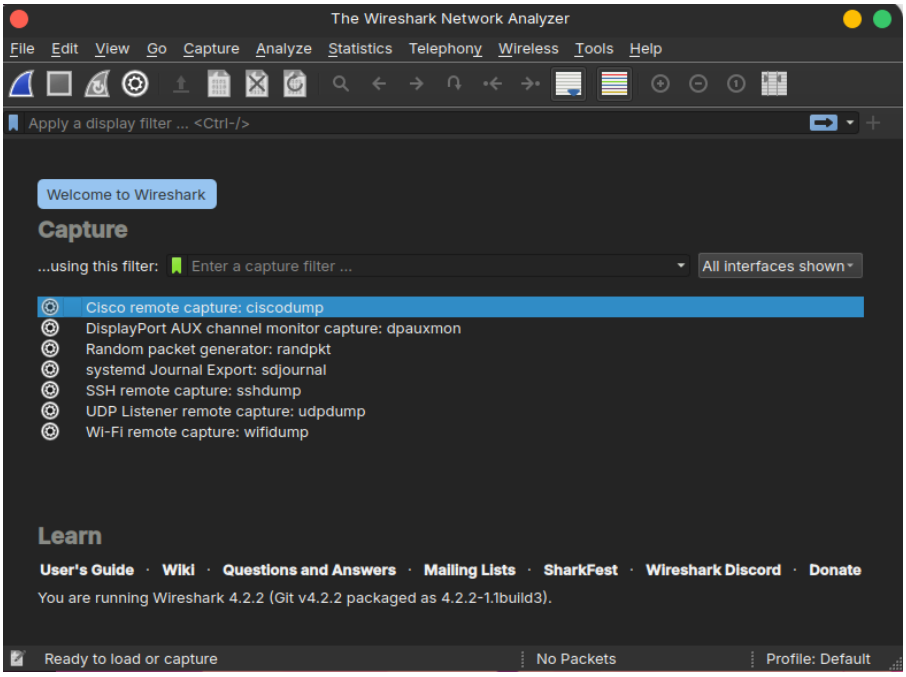

Figure 1: *Wireshark UI.*

From this, I opened the **pcap** file from the file manager.

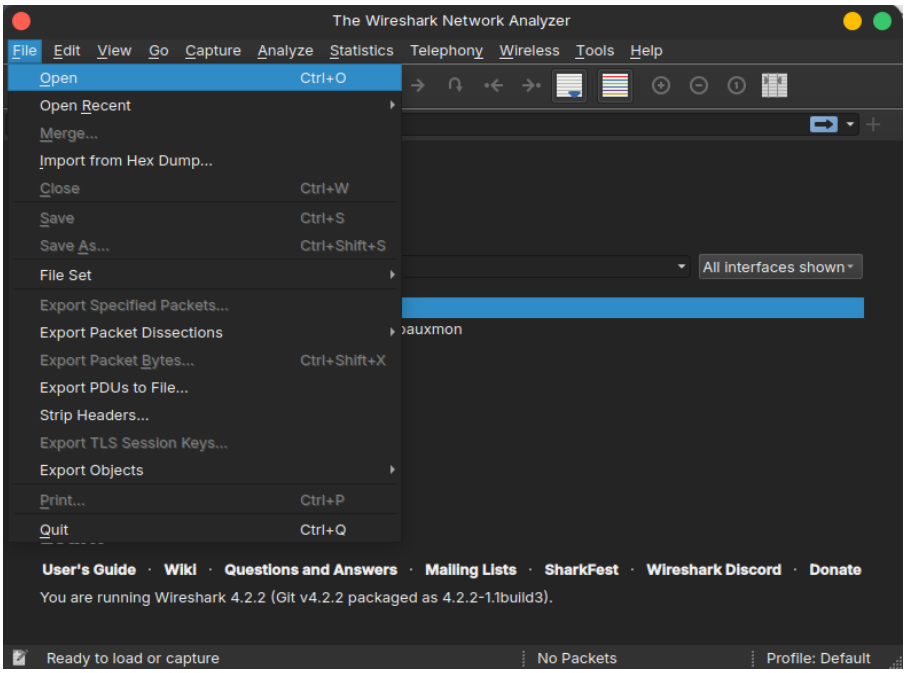

Figure 2: *Opening File.*

Navigating to `/home/b1ondie/Downloads`, I selected the file.

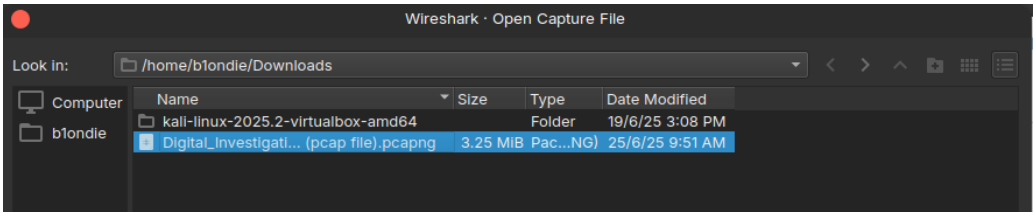

Figure 3: *Selecting Given File.*

Within the **pcap** file, I was able to view the network traffic generated by the user.

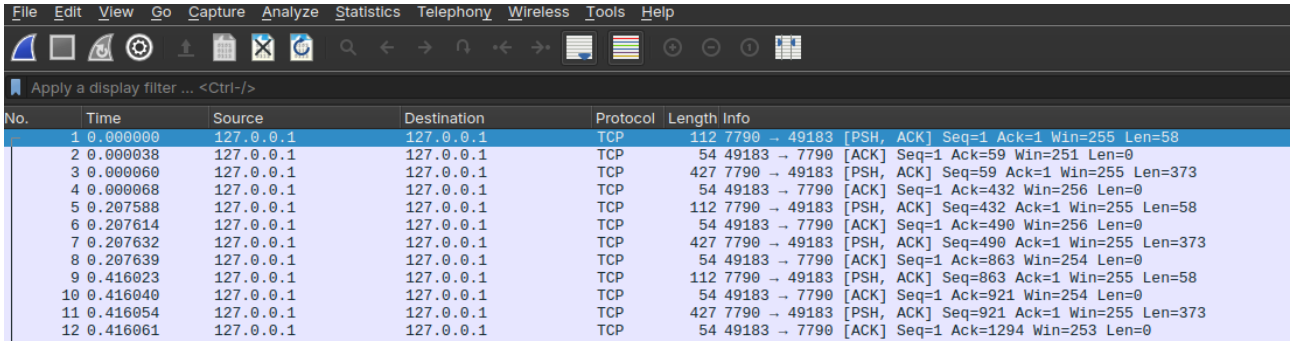

Figure 4: *Viewing **pcap** File.*

As per the request, I filtered the traffic to **http** to ensure I could analyse only HTTP traffic.

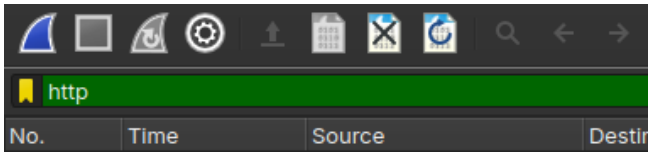

Figure 5: *Filtering Traffic as HTTP.*

I am now able to see all of the HTTP traffic generated by the user.

```
  402 GET /anz-logo.jpg HTTP/1.1
 1065 HTTP/1.1 200 OK  (JPEG JFIF image)
  403 GET /bank-card.jpg HTTP/1.1
  348 HTTP/1.1 200 OK  (JPEG JFIF image)
  401 GET /anz-png.png HTTP/1.1
  790 HTTP/1.1 200 OK  (PNG)
  389 GET /how-to-commit-crimes.docx HTTP/1.1
  488 HTTP/1.1 200 OK  (application/vnd.openxmlformats-officedocument.wordprocessingml.document)
  619 GET /hiddenmessage2.txt HTTP/1.1
 1453 HTTP/1.1 200 OK  (text/plain)
  609 GET /evil.pdf HTTP/1.1
 1486 HTTP/1.1 200 OK  (application/pdf)
  403 GET /atm-image.jpg HTTP/1.1
  352 HTTP/1.1 200 OK  (JPEG JFIF image)
  617 GET /ANZ_Document.pdf HTTP/1.1
 1284 HTTP/1.1 200 OK  (application/pdf)
  618 GET /ANZ_Document2.pdf HTTP/1.1
  744 HTTP/1.1 200 OK  (application/pdf)
  398 GET /ANZ1.jpg HTTP/1.1
 1471 HTTP/1.1 200 OK  (JPEG JFIF image)
  398 GET /ANZ2.jpg HTTP/1.1
```

Figure 6: *Analysing First Image.*

In the first HTTP frame, I can see the user requested the **anz-logo.jpg** resource.

```
pl  Length Info
      402 GET /anz-logo.jpg HTTP/1.1
     1065 HTTP/1.1 200 OK  (JPEG JFIF imag
```

Figure 7: *GET Request from Client.*

Inspecting this packet, I can see that the server response is within frame **140**.

```
[Full request URI: http://localhost:8000/anz-logo.jpg]
[HTTP request 1/1]
[Response in frame: 140]
```

Figure 8: *Response Note.*

I navigate to this frame, and inspect its contents.

```
→  131 6.132470    ::1    ::1    HTTP    402 GET /anz-logo.jpg HTTP/1.1
←  140 6.363216    ::1    ::1    HTTP    1065 HTTP/1.1 200 OK  (JPEG JFIF image)
   505 22.697209   ::1    ::1    HTTP    403 GET /bank-card.jpg HTTP/1.1
```

Figure 9: *Response Frame from Server.*

After doing some quick research [1], I found the appropriate file signature for a **.jpg** file, which is **FF D8 FF E0**.

JPEG-EXIF-SPIFF images	FF D8 FF	JFIF|JPE|JPEG|JPG	Picture	0	FF D9

Figure 10: *JPEG File Header [1].*

Inspecting the frame, I was able to find the **JPEG** file.

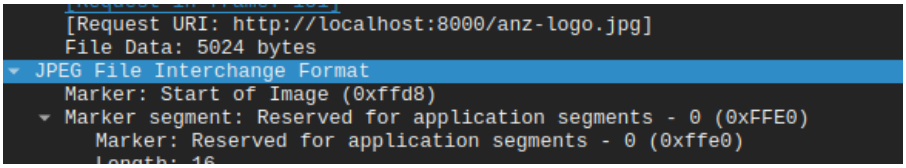

Figure 11: *Packet Details.*

I validated its file type by inspecting the raw bytes, which matched the file signature in my research.

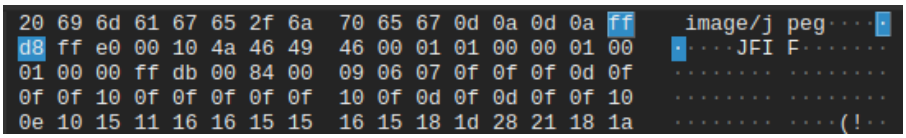

Figure 12: *Packet Hex and ASCII Values.*

I inspected the image itself, by right-clicking the **JPEG Filee Interchange Format** line, and clicking on **Show Packet Bytes**.

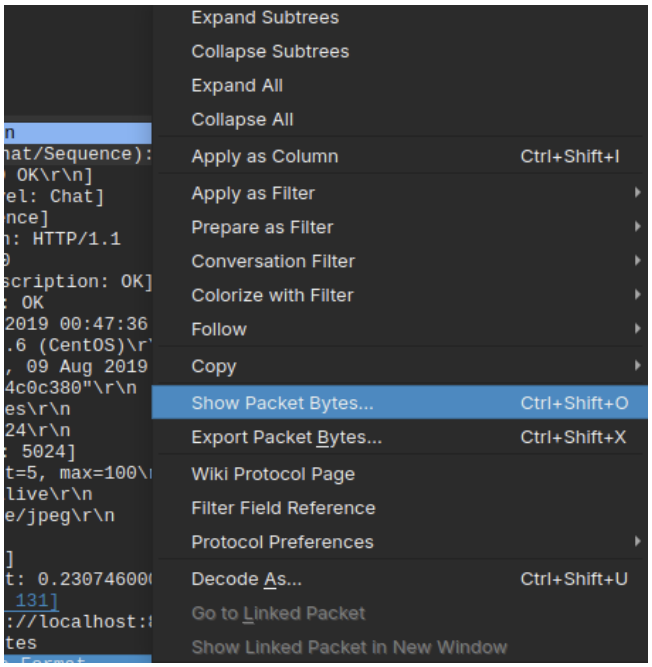

Figure 13: *Show Packet Bytes Option.*

I can now confirm that this is, in fact, an image file.

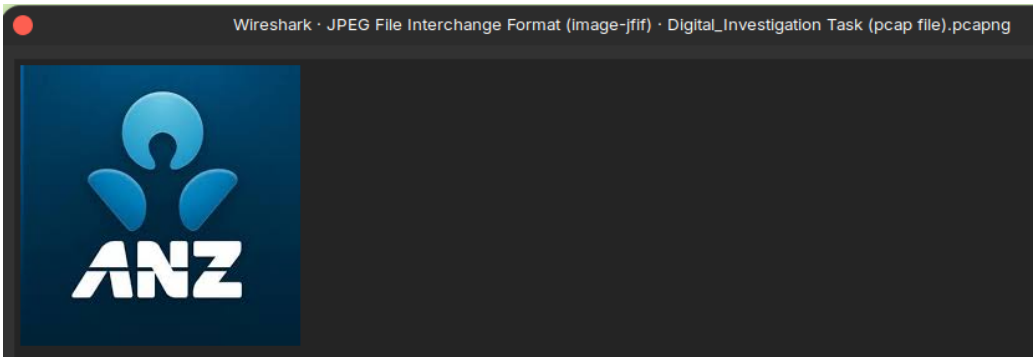

Figure 14: *ANZ Image in Wireshark.*

I proceed to export the image file, by right-clicking the same line, and selecting **Export Packet Bytes**.

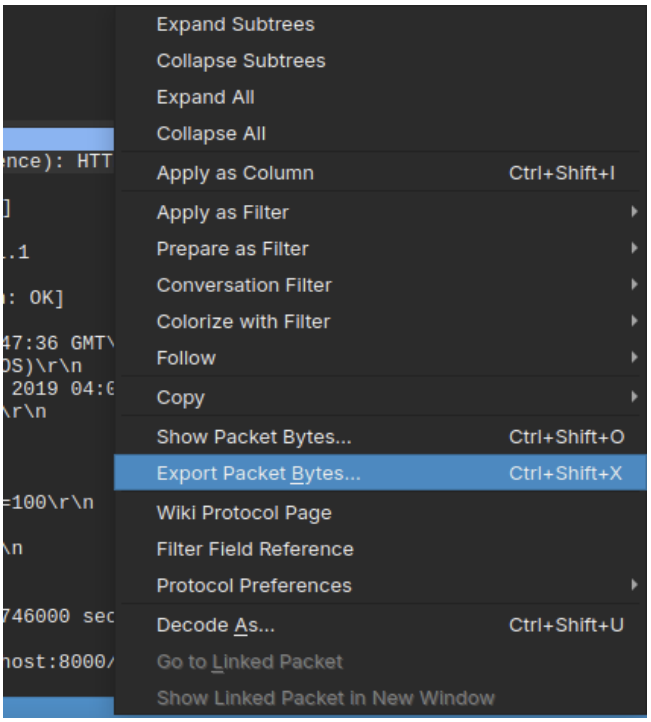

Figure 15: *Export Packet Bytes Option.*

I saved this on my local machine under **anz-logo.jpg**.

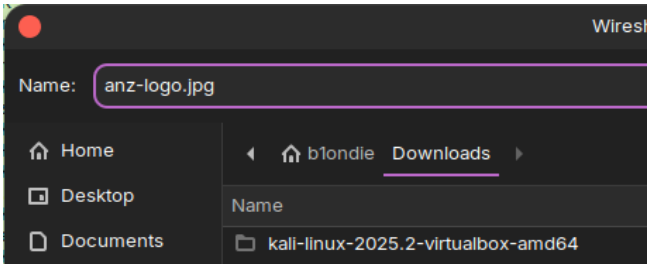

Figure 16: *Saving Logo to File System.*

I can now view this in my file system.

Figure 17: *File Saved.*

Opening the file heeds no errors, indicating that file extraction was a success.

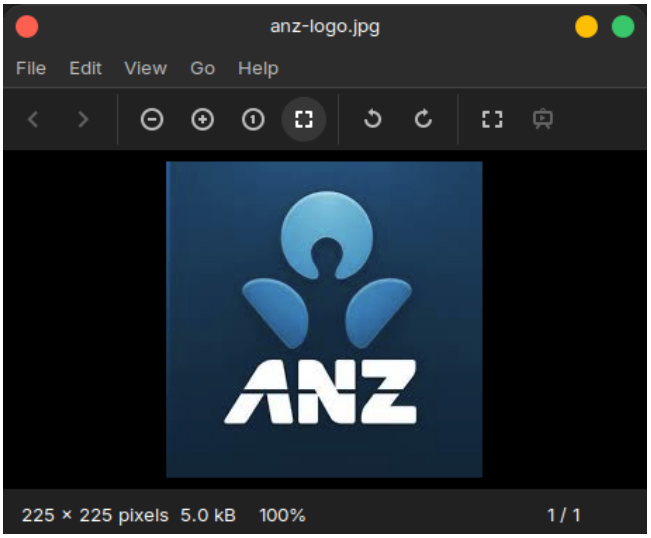

Figure 18: *File Operable.*

The full image contains the ANZ logo, in blue and white.

Figure 19: *Full Image.*

Inspecting the next file, **bank-card.jpg**, I followed the same procedure.

```
HTTP     1065 HTTP/1.1 200 OK  (JPEG JFIF image)
HTTP      403 GET /bank-card.jpg HTTP/1.1
HTTP      348 HTTP/1.1 200 OK  (JPEG JFIF image)
HTTP      401 GET /anz-png.png HTTP/1.1
```

Figure 20: *Second Image GET Request.*

When the user has requested the image resource, the response from the server is provided in **frame 567**.

```
[Full request URI: http://localhost:8000/bank-card.jpg]
[HTTP request 1/1]
[Response in frame: 567]
```

Figure 21: *Response Note.*

I proceed to navigate to this frame is Wireshark.

```
    140 6.363216      ::1
    505 22.697209     ::1
    567 24.333701     ::1
    818 36.266571     ::1
```

Figure 22: *Response Packet.*

Using the same method, I find the **JPEG File Interchange Format** line.

```
[Request URI: http://localhost:8000/bank-card.jpg]
File Data: 11506 bytes
▾ JPEG File Interchange Format
  Marker: Start of Image (0xffd8)
  ▾ Marker segment: Reserved for application segments - 0 (0xFFE0)
```

Figure 23: *Packet Details.*

Right-clicking this, I select **Export Packet Bytes**.

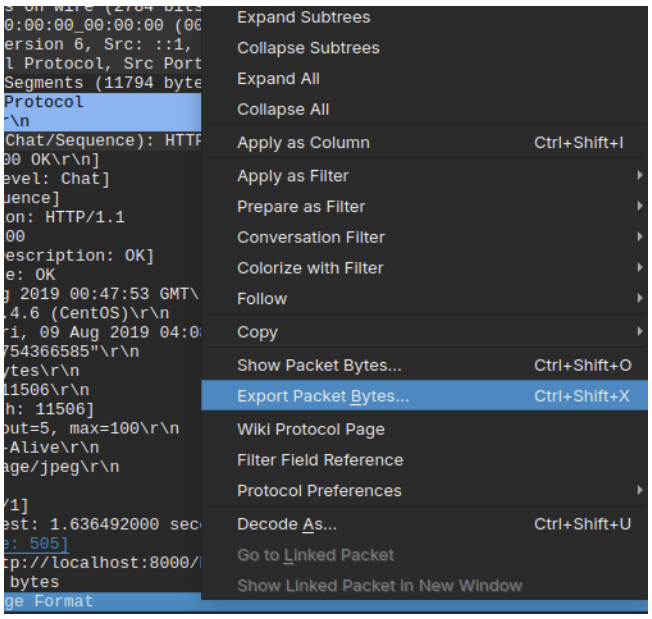

Figure 24: *Export Packet Details Option.*

This image is saved in my local file system under the name **bank-card.jpg**.

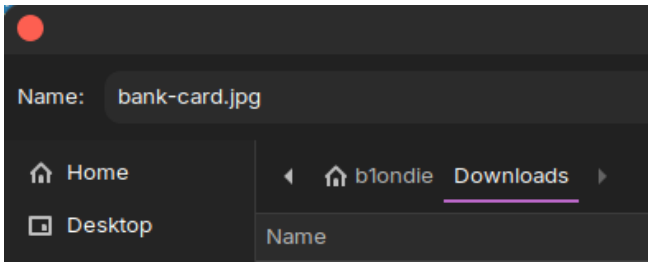

Figure 25: *Saving File.*

I can now view this in my file system.

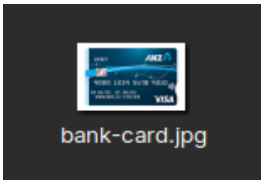

Figure 26: *File Saved to File System.*

I validate its integrity by opening the file. I can see the entire file in my image viewer, which demonstrates file extraction was a success.

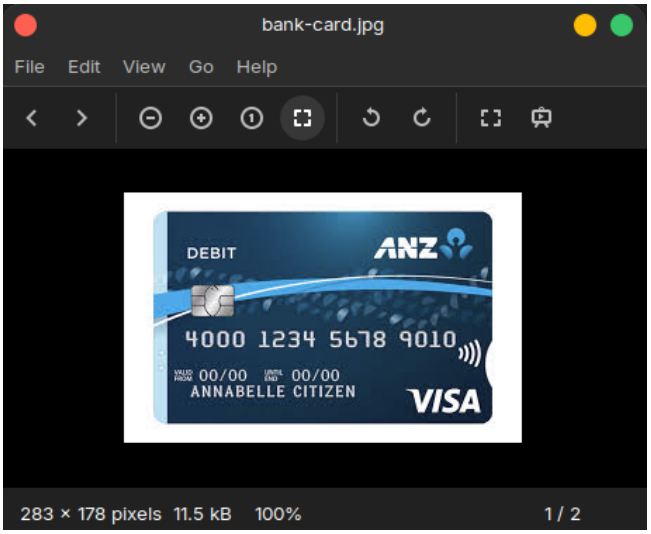

Figure 27: *File Operable.*

The full image appears to be a bank card, addresses to an **Annabelle Citizen**, with the full card number shown, **4000 1234 5678 9010**.

Figure 28: *Full Image.*

3 Section 2: Examine ANZ1.jpg and ANZ2.jpg.

3.1 Tools Used

- **Wireshark**

I first examined a regular image that the server had sent to the user.

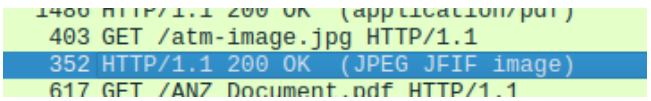

Figure 29: *Image GET Request.*

Upon examining this file, I analysed the file structure as a means of comparison to **ANZ1.jpg**, and **ANZ2.jpg**.

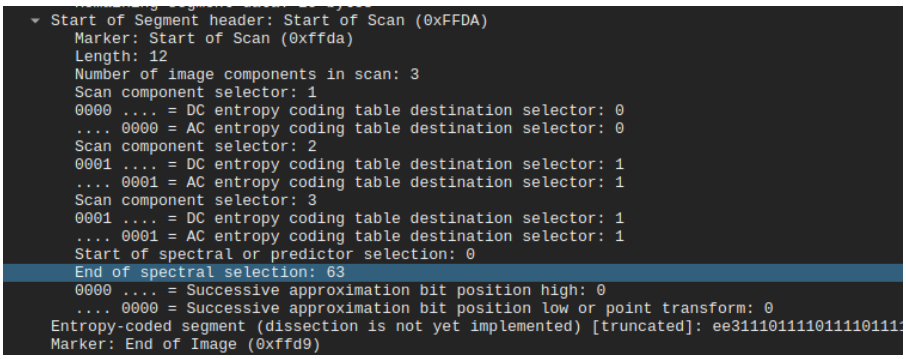

Figure 30: *Packet Details.*

I then navigated to **ANZ1.jpg**.

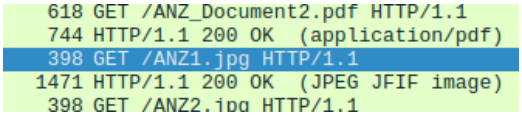

Figure 31: *Image GET Request.*

Within this image, I noticed an additional line at the bottom of the frame data, denoted **Entropy-coded segment (dissection is not yet implemented): 596f7527766520666f756e642061206869646465e206d65737361676520696e2074686973206696c652120496e636c756465207420696e20796f757220777269746520757020e0a**.

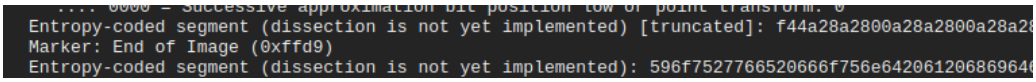

Figure 32: *Hidden String in Packet.*

When analysing the data further, a secret message appears in the **ASCII** panel of Wireshark:

`You've found a hidden message in this file! Include it in your write up.`

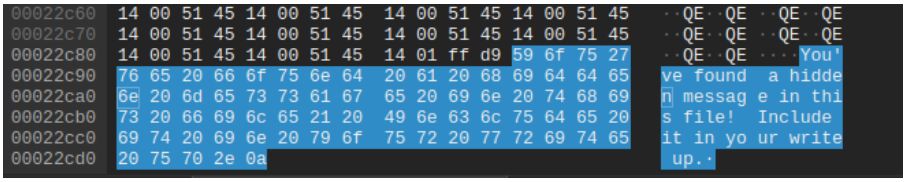

Figure 33: *Hidden String in ASCII Panel.*

I noticed **ANZ2.jpg** also contained a similar file structure.

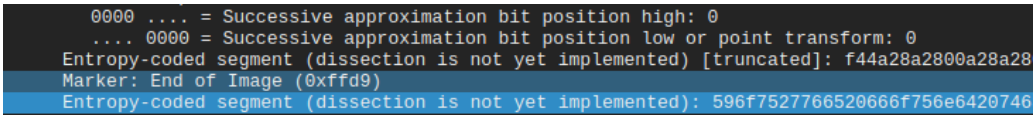

Figure 34: *ANZ2.jpg Presents Same File Structure.*

Scrolling to the last bytes of the file, I notice a similar hidden message to the one above, which reads: `You've found a hidden message!  Images are sometimes more than they appear.`

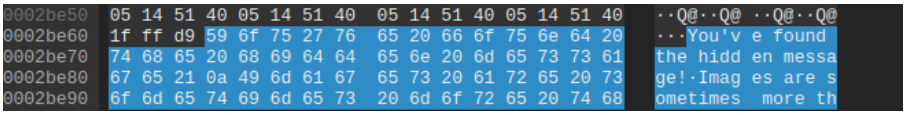

Figure 35: *Hidden String in ASCII Panel.*

To extract the files, I right-clicked the image file in the frame and selected the **Export Packet Bytes** option as usual.

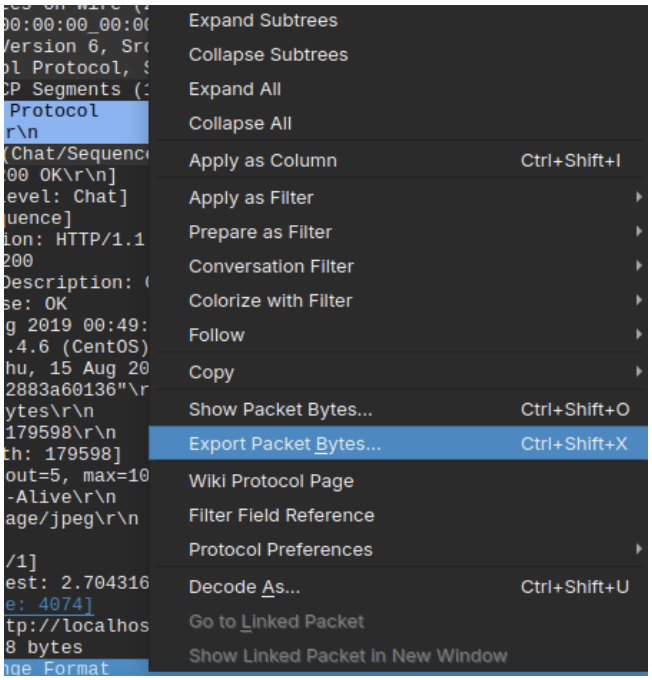

Figure 36: *Export Packet Bytes Option.*

I can now see these images in my file system.

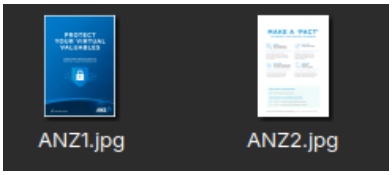

Figure 37: *Two Images in File System.*

Opening the files validates their integrity, as they have not been corruputed in the extraction process, and work as expected.

Figure 38: *Extracted JPG's.*

ANZ1.jpg is an awareness poster made by ANZ prompting the viewer to **protect [their] virtual valuables**.

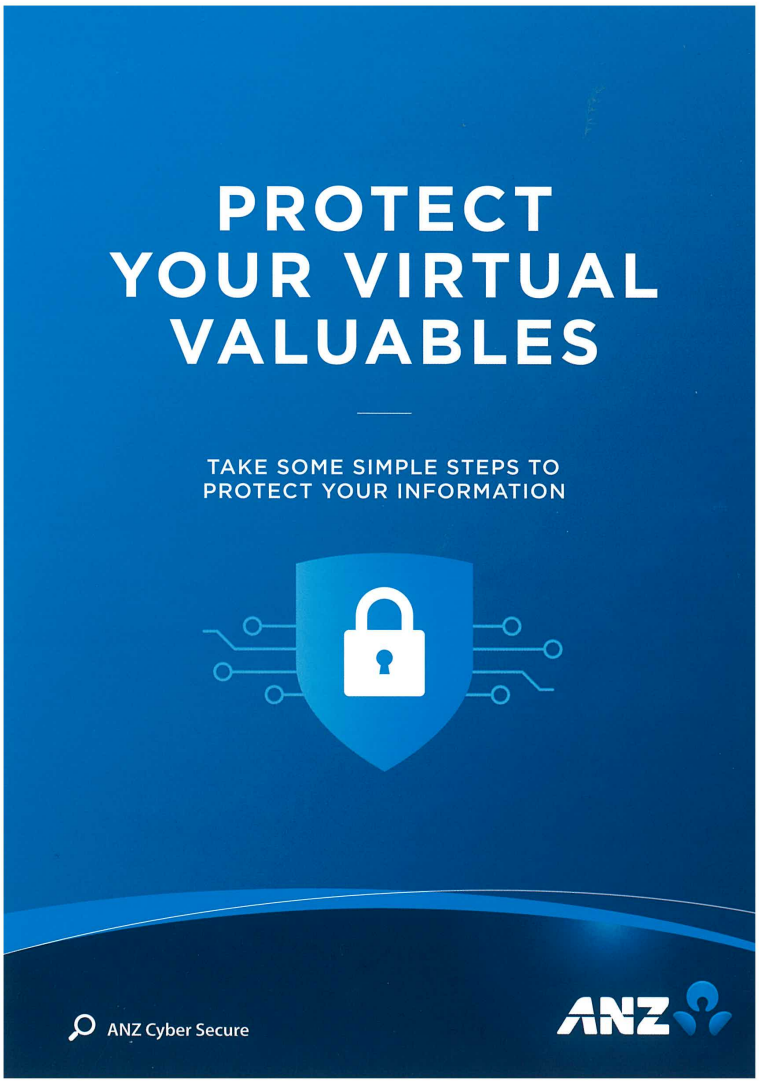

Figure 39: *First JPG.*

ANZ2.jpg is another awareness poster on *how* one can protect their virtual valuables.

Figure 40: *Second JPG.*

4 Section 3: Examine how-to-commit-crimes.docx.

4.1 Tools Used

- **Wireshark**

I searched through the packets in Wireshark until I found **how-to-commit-crimes.docx**.

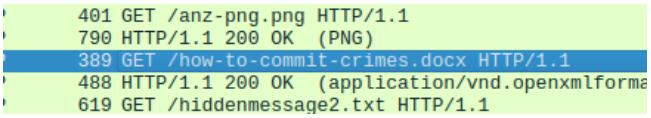

Figure 41: *docx GET Request.*

The very next frame displays the requested content provided by the server.

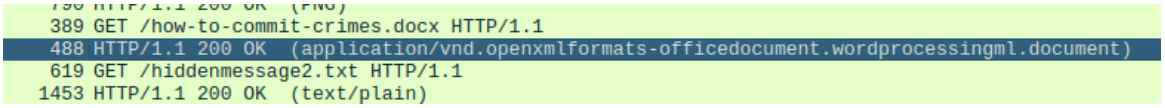

Figure 42: *Response Packet from Server.*

Within this frame, I notice that the document is an Word Processing XML document.

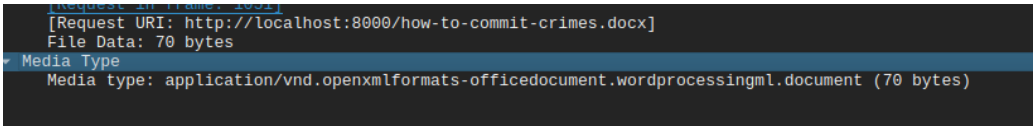

Figure 43: *Media Type of Payload.*

Analysing the contents of the document in the **ASCII/Raw Bytes panel** of Wireshark, I notice the following words:

```
Step 1:  Find target.
Step 2:  Hack them.
This is a suspicious document.
```

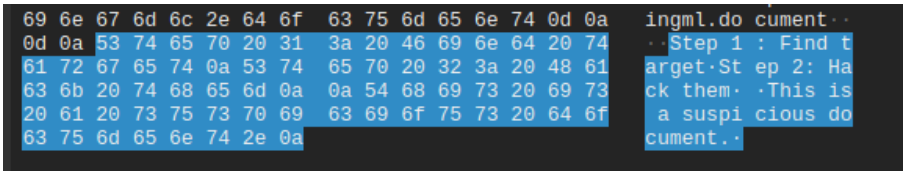

Figure 44: *Suspicious Strings in docx.*

To analyse the file further, I export the bytes using the option **Export Packet Bytes**.

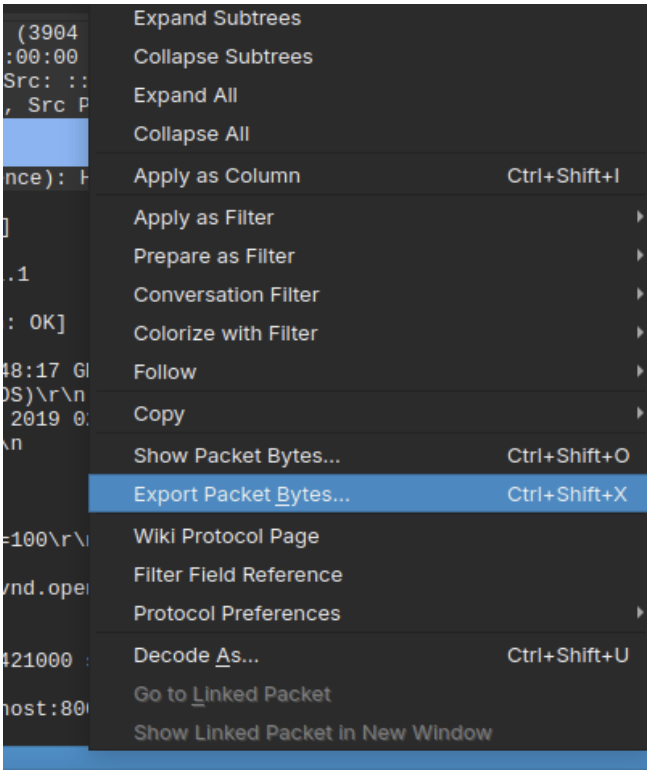

Figure 45: *Export Packet Bytes Option.*

A new window pops up and I input the file name **how-to-commit-crimes.docx** to maintain the integrity of the file type.

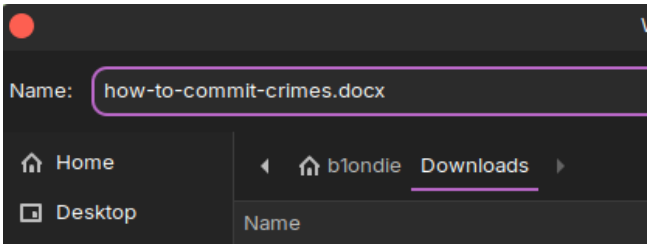

Figure 46: *Extracting File.*

This file is now available to view in my file system.

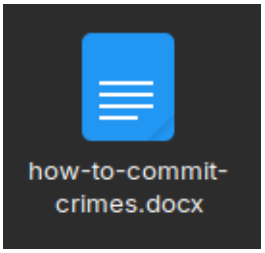

Figure 47: *Extracted File in File System.*

When opened with LibreOffice, I can now see the entirety of the file contents as I first discovered in Wireshark. This confirms that the file contents is indeed:

```
Step 1:  Find target.
Step 2:  Hack them.
This is a suspicious document.
```

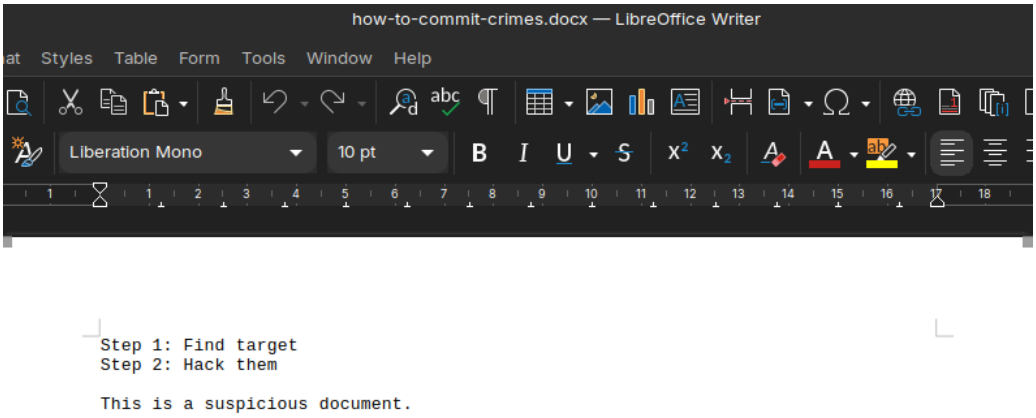

Figure 48: *File is Operable and Displays Secret Message.*

5 Section 4: Examine ANZ_Document.pdf, ANZ_Document2.pdf, and evil.pdf.

5.1 Tools Used

- **Wireshark**

Tracing the clients resource request for **ANZ_Document.pdf**, I was able to locate the server's response.

```
HTTP      352 HTTP/1.1 200 OK  (JPEG JFIF image)
HTTP      617 GET /ANZ_Document.pdf HTTP/1.1
HTTP     1284 HTTP/1.1 200 OK  (application/pdf)
HTTP      618 GET /ANZ_Document2.pdf HTTP/1.1
```

Figure 49: *PDF GET Request.*

As usual, I export the file using **Export Packet Bytes**.

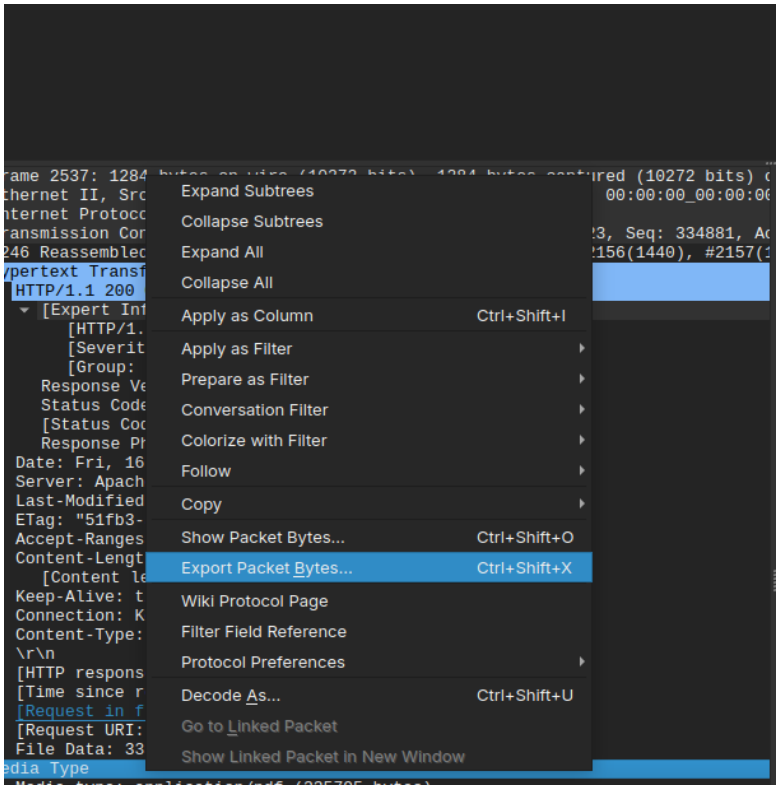

Figure 50: *PDF GET Request.*

I extract this file, naming it **ANZ_Document.pdf**.

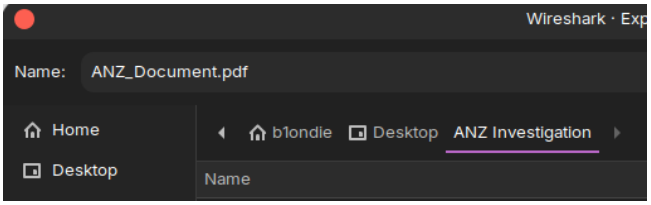

Figure 51: *Export Packet Bytes Option.*

I follow the same process for **ANZ_Document2**, where I am able to find the document in question.

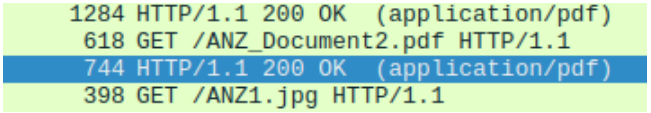

Figure 52: *Extracting ANZ_Document.pdf.*

I extract this file as well using **Export Packet Bytes**.

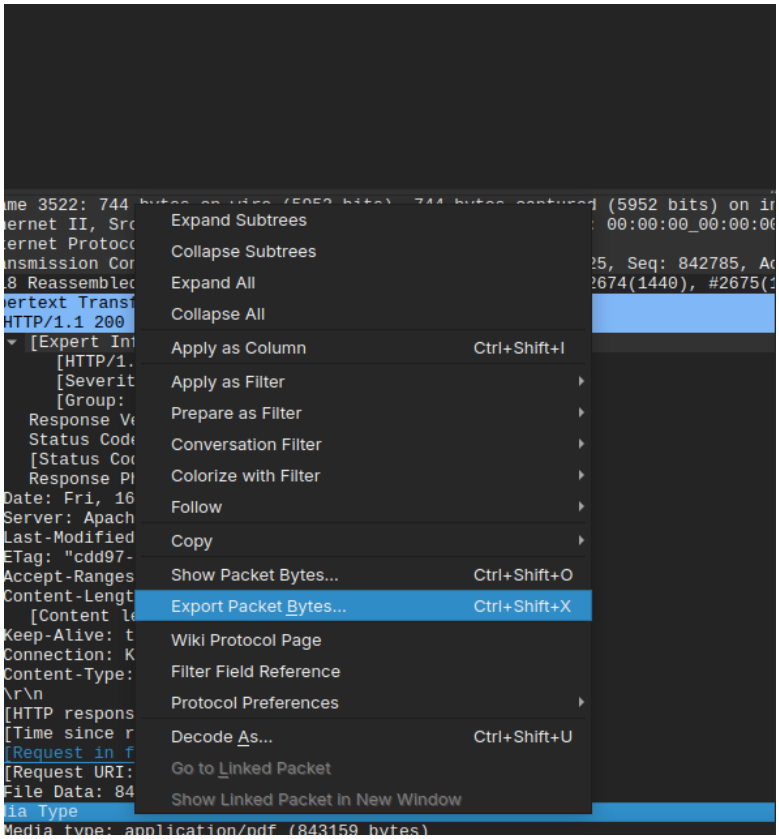

Figure 53: *Export Packet Bytes Option*

This is saved to my file system under **ANZ_Document2.pdf**.

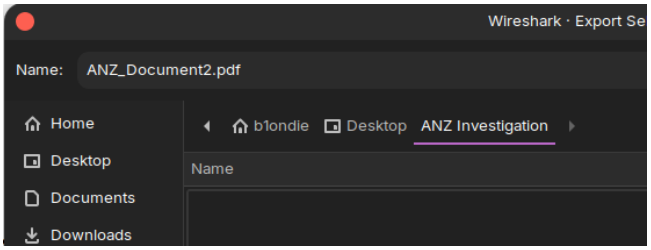

Figure 54: *Extracting PDF File.*

I followed the same process to find the **evil.pdf** file in the HTTP traffic.

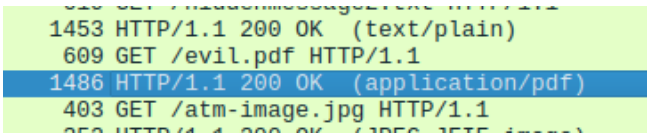

Figure 55: *PDF GET Request.*

Opening the server packet, I exported the PDF using **Export Packet Bytes**.

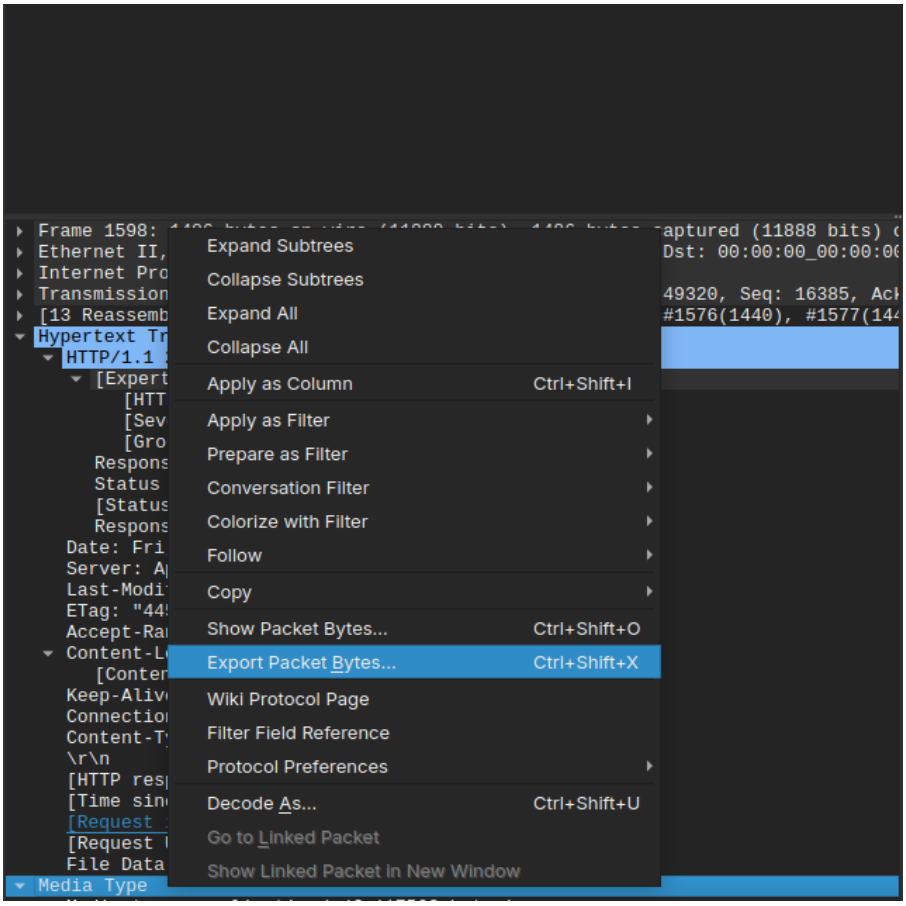

Figure 56: *Export Packet Bytes Option.*

This was subsequently saved as **evil.pdf** in my file system.

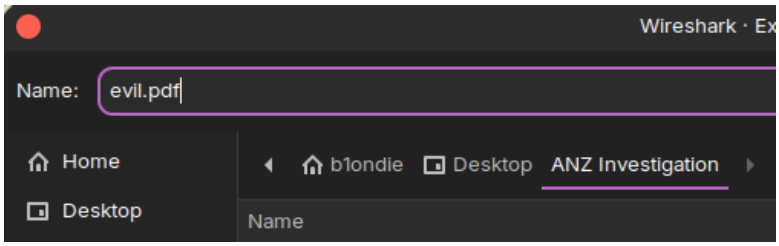

Figure 57: *Extracting File.*

I am now able to view all three files in my file system.

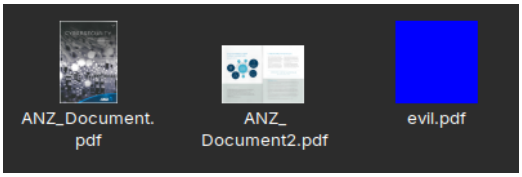

Figure 58: *All Files Extracted.*

Opening the files demonstrates their integrity, as I receive no unexpected errors.

Figure 59: *Files Available to View.*

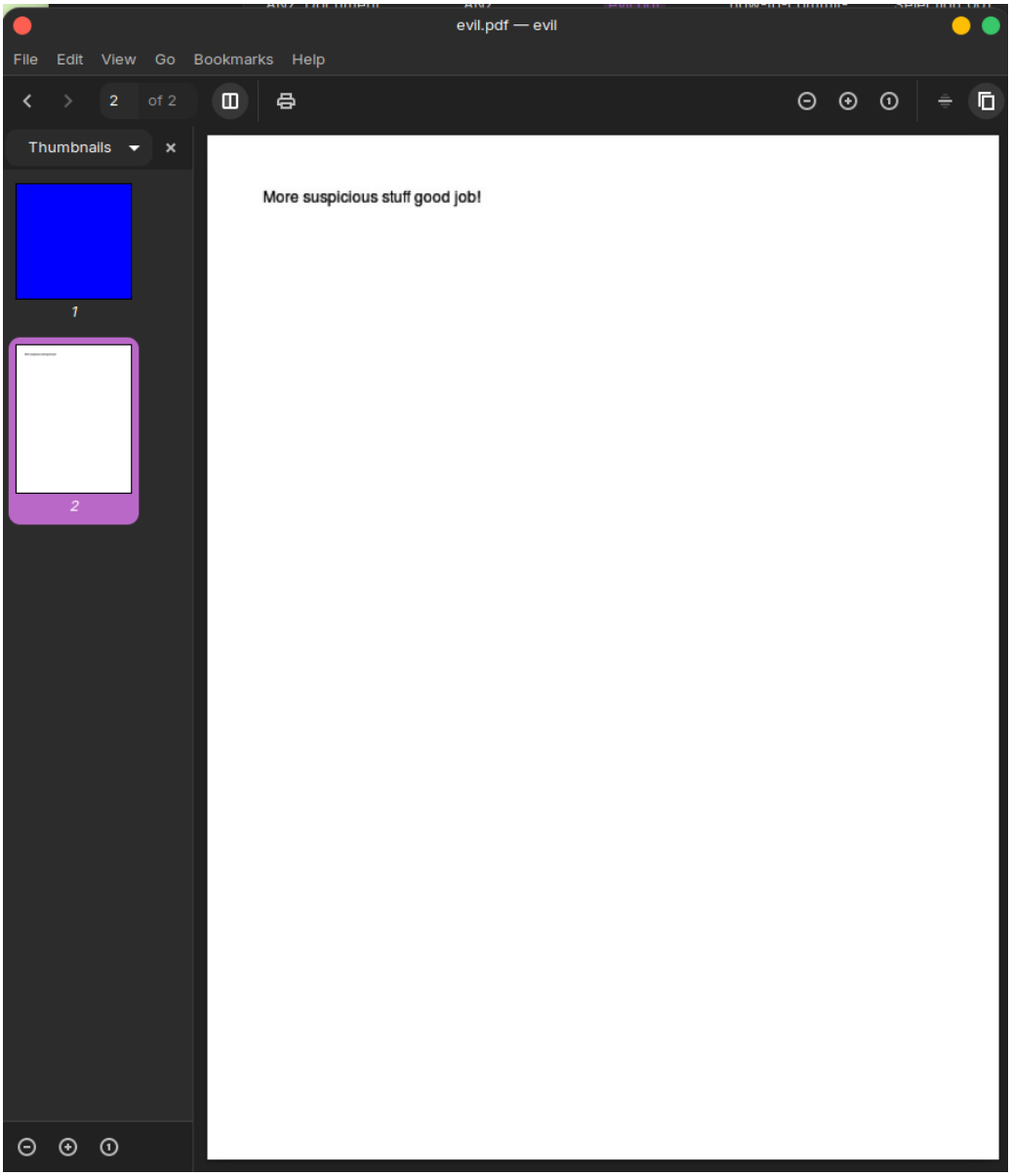

Figure 60: *Hidden Message in evil.pdf.*

ANZ_Document.pdf displays what seems to be the first page of a Cybersecurity document from ANZ.

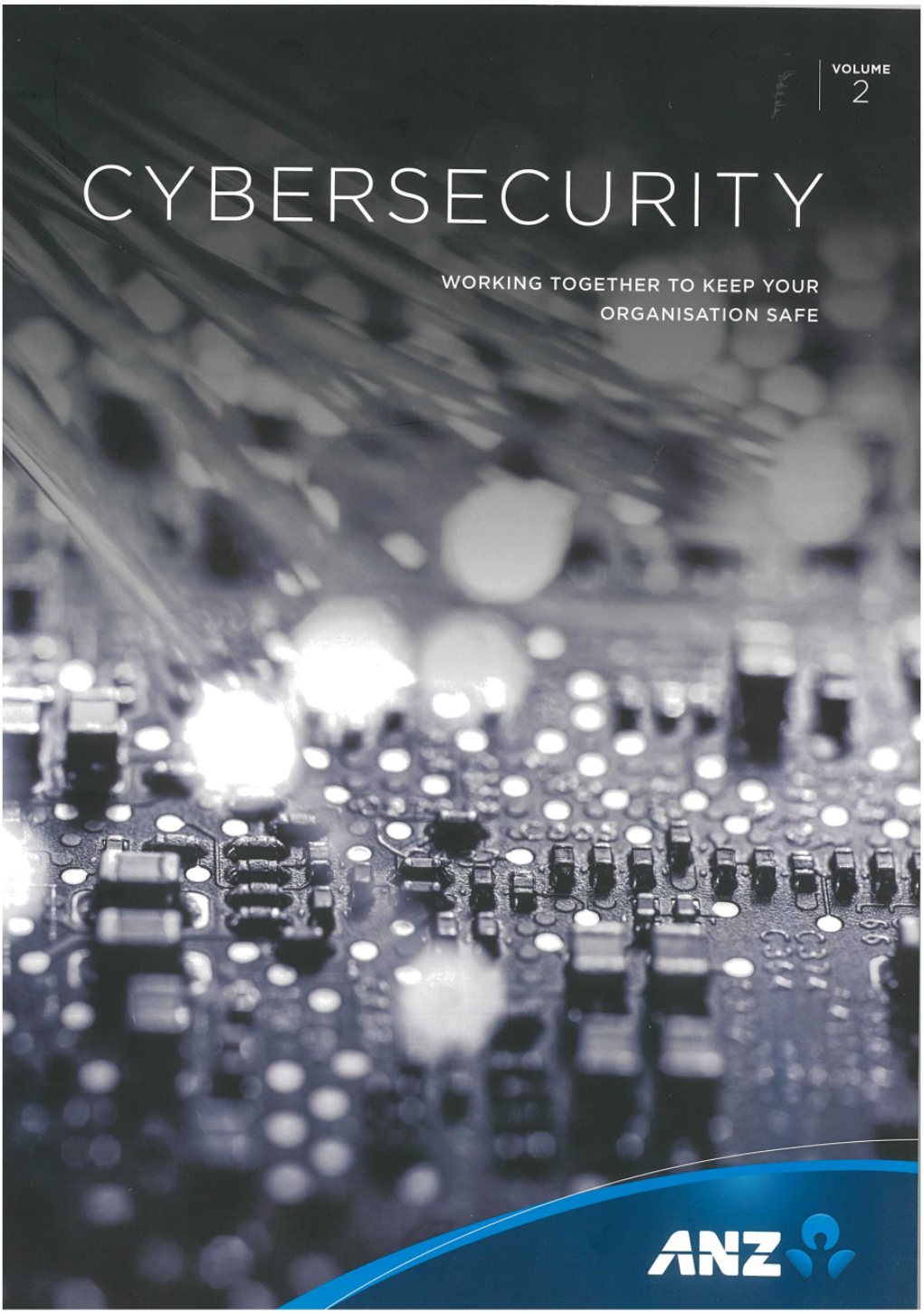

Figure 61: *Full PDF.*

ANZ_Document2.pdf appears to be another two pages from the same document, namely pages 3 and 4.

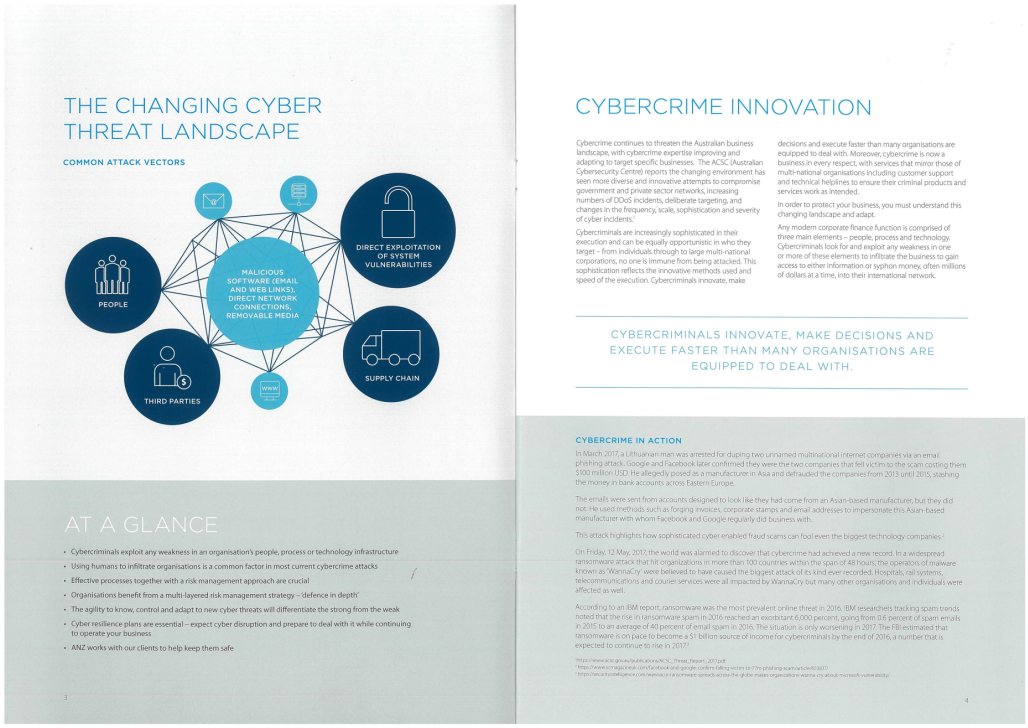

Figure 62: *Full PDF.*

evil.pdf contains two pages, the first of which is completely blue, and the second which has a hidden message: `More suspicious stuff good job!`.

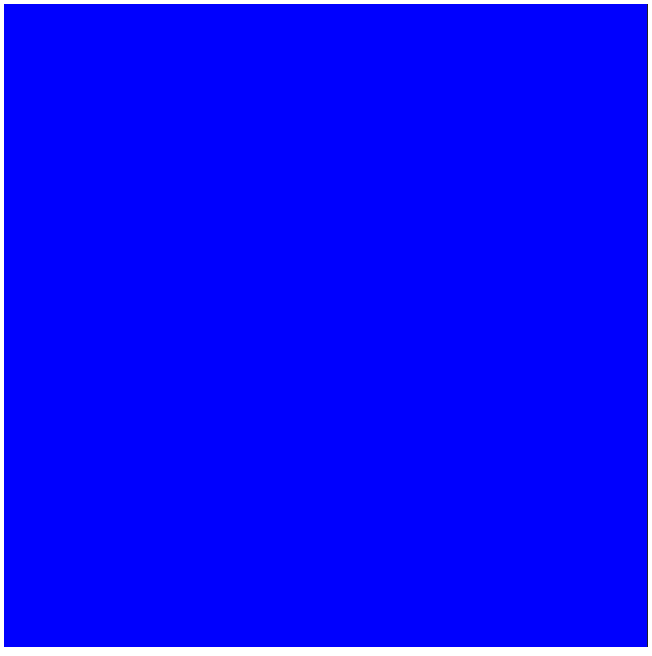

Figure 63: *First Page of evil.pdf.*

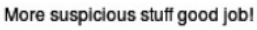

Figure 64: *Second Page of PDF.*

6 Section 5: Examine the Contents Within hiddenmessage2.txt.

6.1 Tools Used

- **Wireshark**
- **Linux Mint Text Editor**: a tool that can edit and modify text files

This part of the investigation explores analysing file signatures. To start, I navigated to the **hiddenmessage2.txt** file resource sent from the server.

Figure 65: *.txt GET Request.*

From here, I extract the file using **Export Packet Bytes**.

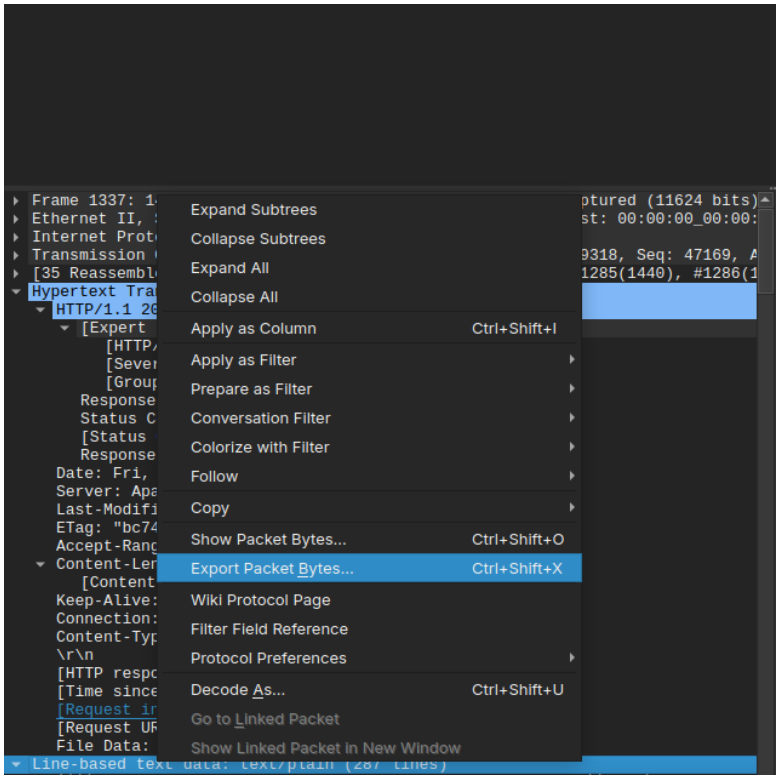

Figure 66: *Export Packet Bytes Option.*

Before saving this to my file system under the name **hiddenmessage.txt**.

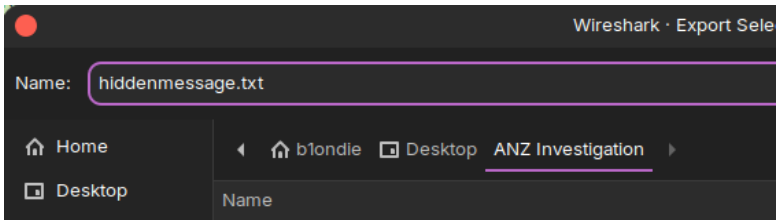

Figure 67: *Extracting .txt File.*

Opening this file in my text editor, I noticed that the values looked very similar to what you would see in an image file.

Figure 68: *Viewing File in Text Editor.*

Investigating further, I can see that the file is, in fact, an image file, specifically a JPEG image due to the presence of the file signature **FF D8 FF E0**.

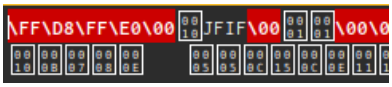

Figure 69: *JPG File Signature.*

Due to this, I chose to extract the file but instead of saving it as a **.txt** file, I named it **hiddenmessage.jpg**.

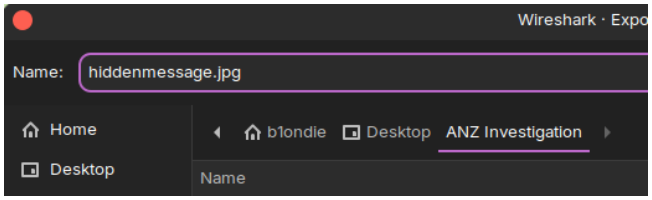

Figure 70: *Saving File as JPG.*

Once saved, I can now see the image in my file system, and it has successfully rendered as an image file.

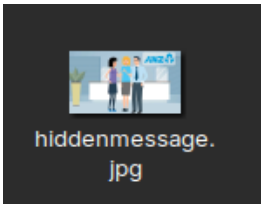

Figure 71: *Full Image in File System.*

Opening the image displays an ANZ graphic with three individuals. This demonstrates the importance of

validating file signatures, and ensuring files are what they claim they are. Otherwise, a user could be tricked into opening a file that is disguised as one file, but is actually another file type.

Figure 72: *File Operable.*

7 Section 6: Examine atm-image.jpg.

7.1 Tools Used

- **Wireshark**
- **Okteta**: a linux hex editor used for editing and modifying hex values in files

In this section, I explore file carving and examining file values for hidden images. To begin, I navigated to the **atm-image.jpg** file in **pcap** file.

```
403 GET /atm-image.jpg HTTP/1.1
352 HTTP/1.1 200 OK  (JPEG JFIF image)
617 GET /ANZ_Document.pdf HTTP/1.1
```

Figure 73: *JPG GET Request.*

Examining the packet, I notice that there is one section noting the start of an image.

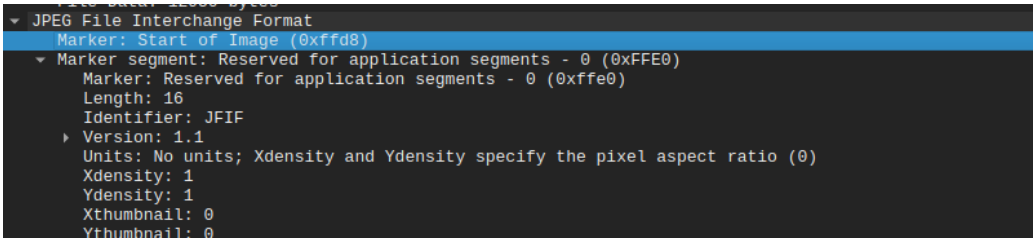

Figure 74: *Packet Details.*

Examining the rest of the packet displays another image, denoted by the presence of **Marker: Start of Image (0xffd8**, which I know is the file signature for a JPEG file.

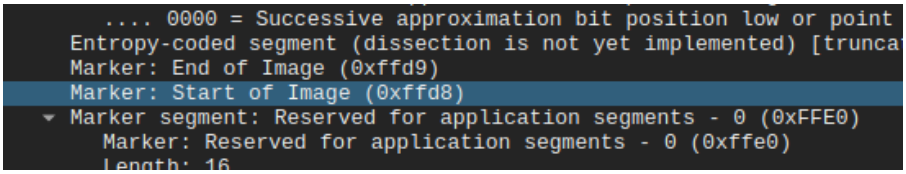

Figure 75: *Packet Details.*

I analysed the hex values further by right-clicking the packet, and selecting **Follow > HTTP Stream**.

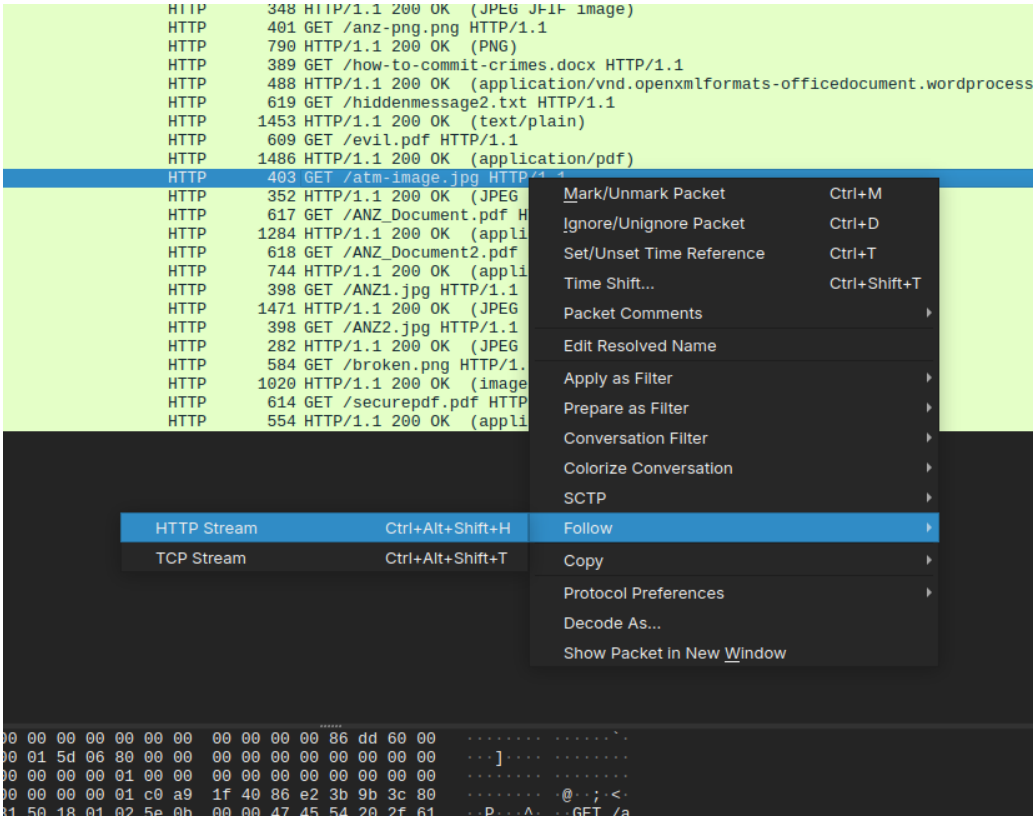

Figure 76: *Following HTTP Stream.*

In this view, I was able to see two mentions of **FFD8**, and **FFD9** which demonstrate the start and end of two image files.

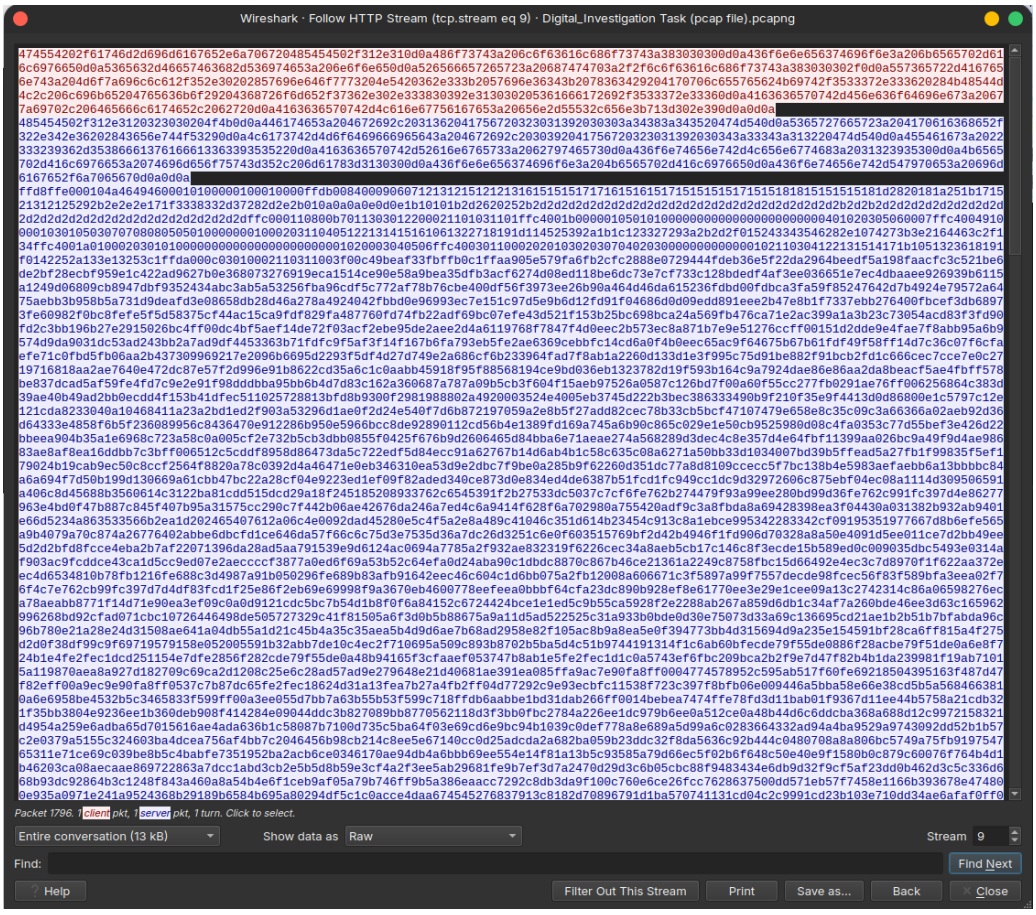

Figure 77: *Analysing Hex Values.*

I proceeded to extract the file and all of its contents using the **Export Packet Bytes** option.

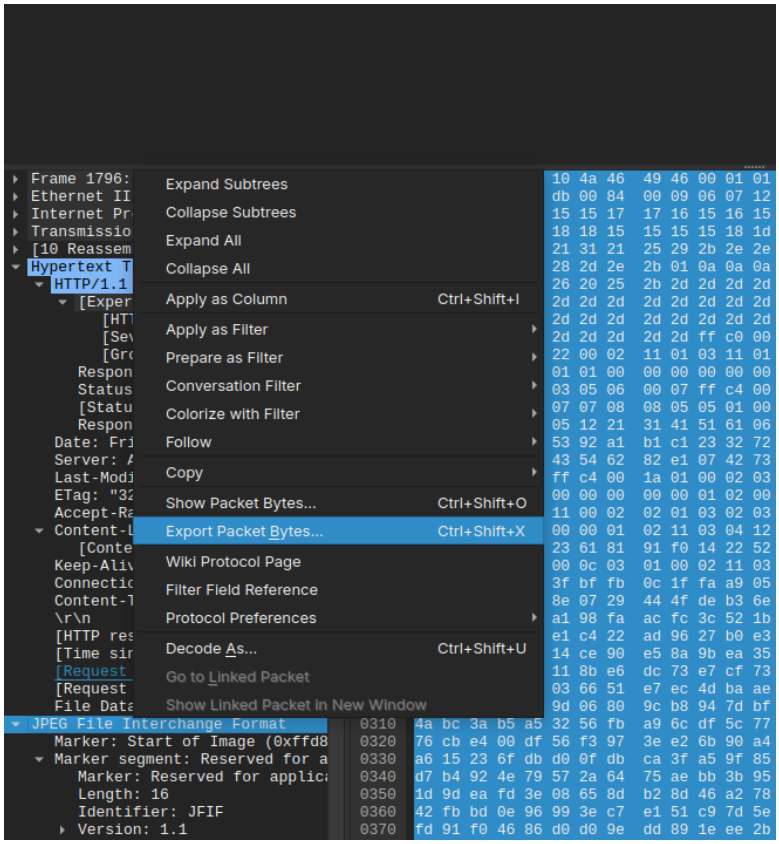

Figure 78: *Export Packet Bytes Options.*

Opening Okteta, I clicked **File > Open** to open the existing file in the program. This will allow me to carve out the files, if needed.

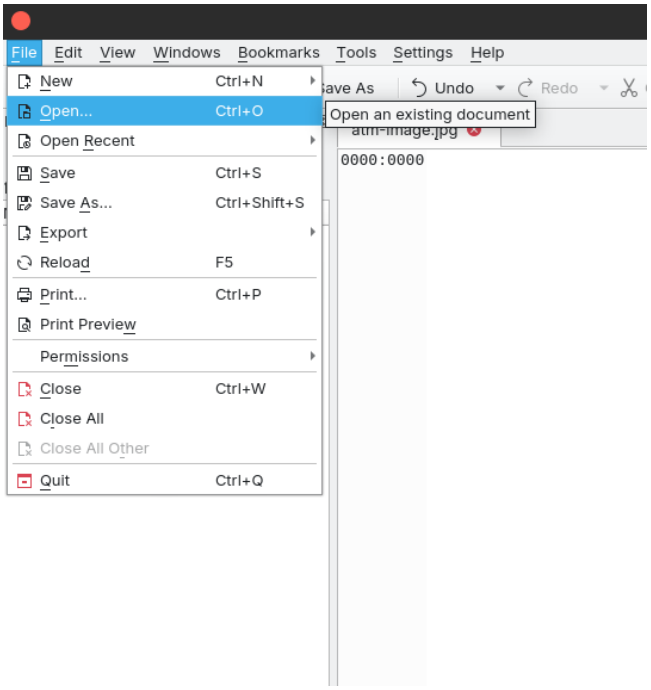

Figure 79: *Opening File in Okteta.*

I ensure I select the correct file, navigating to **atm-image.jpg** and selecting it.

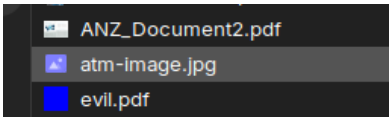

Figure 80: *Selecting Given File.*

Within Okteta, I use the shortcut **Ctrl + F** to bring up the search window. Within this window, I chose to search for the end of the JPEG image, **FFD9**.

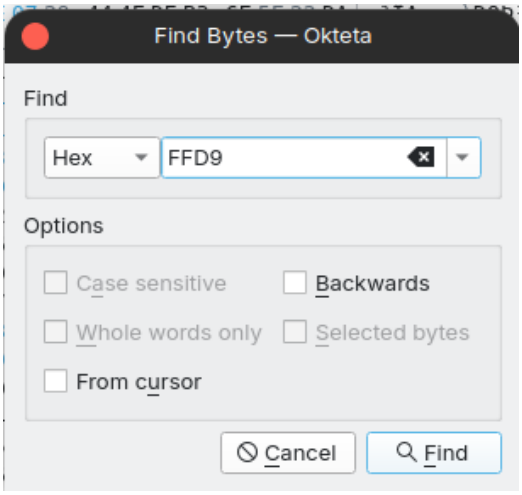

Figure 81: *Seearching for JPG File Signature.*

I was able to find the point at which the first image ends, and the second image starts, denoted by the **DDF9** from Image 1, and **DDF8** from Image 2.

Figure 82: *Signature Found in File.*

I proceed with selecting all of the bytes of the second image, ensuring I encapsulate the final **DDF9** as well.

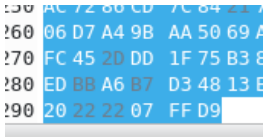

Figure 83: *Selecting the Hex Values of the Second Image.*

I right-click in the window and select **Copy**.

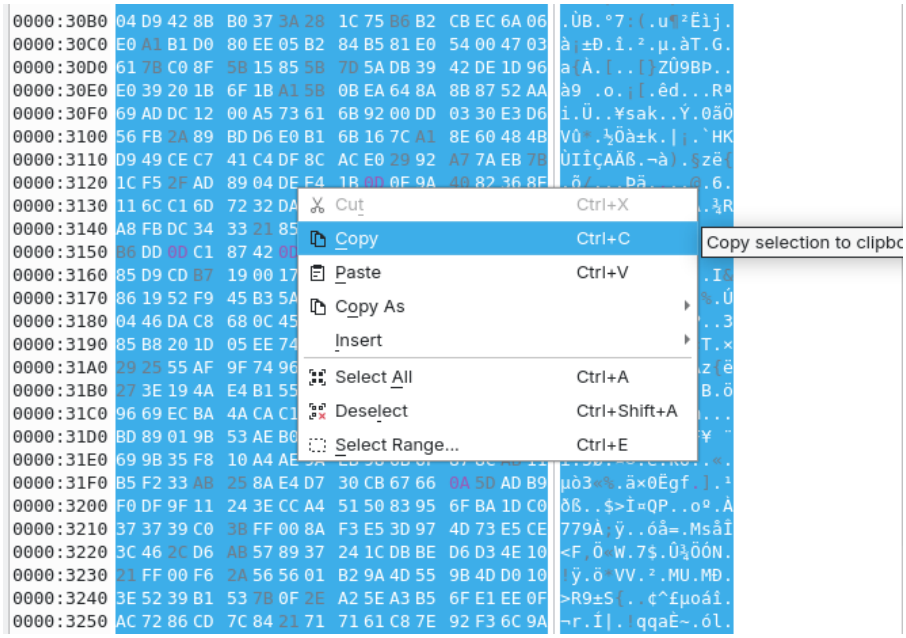

Figure 84: *Copying Selection to Clipboard.*

Using the shortcut **Ctrl + N**, I open a new file. In this file, I paste the hex values that I just copied from the previous file.

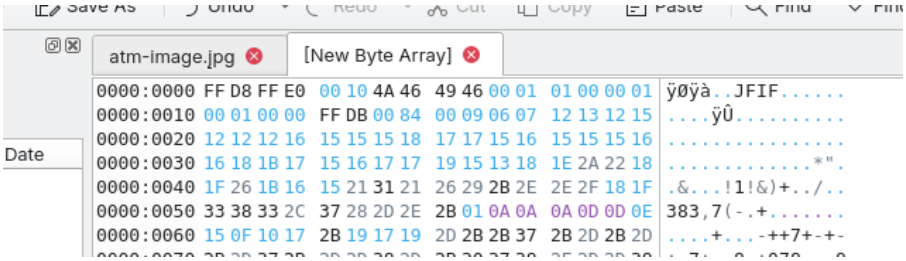

Figure 85: *Pasting Clipboard to New File.*

Once pasted, I saved this file by clicking **File > Save As**.

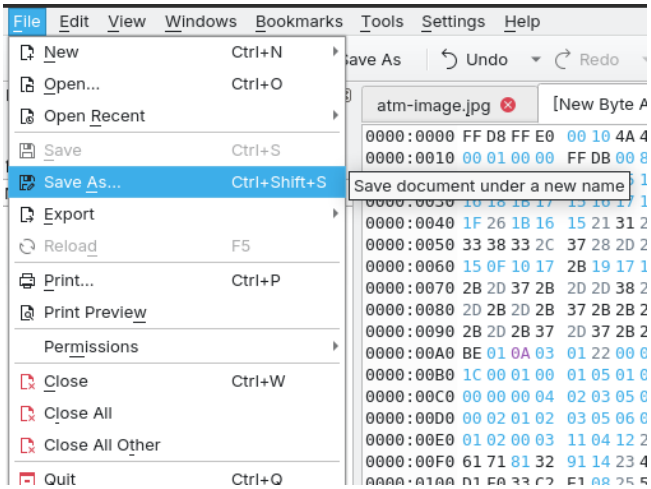

Figure 86: *Saving File.*

This file was saved in my file system under **atm-image2.jpg**.

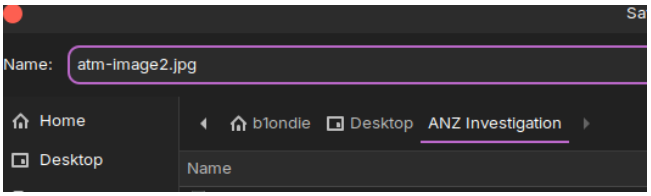

Figure 87: *Renaming for Consistency.*

I repeated the same process for the first image, by selecting between **FFD8** and **DDF9** and clicking **Copy**.

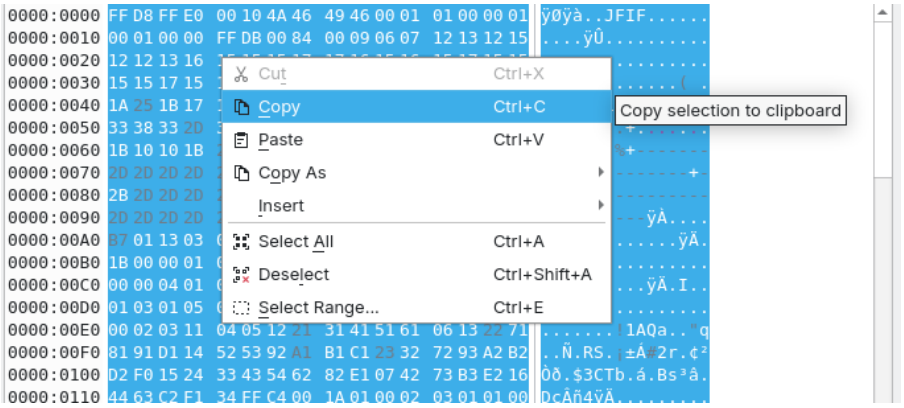

Figure 88: *Copying the Hex Values of First Image.*

This file was also saved.

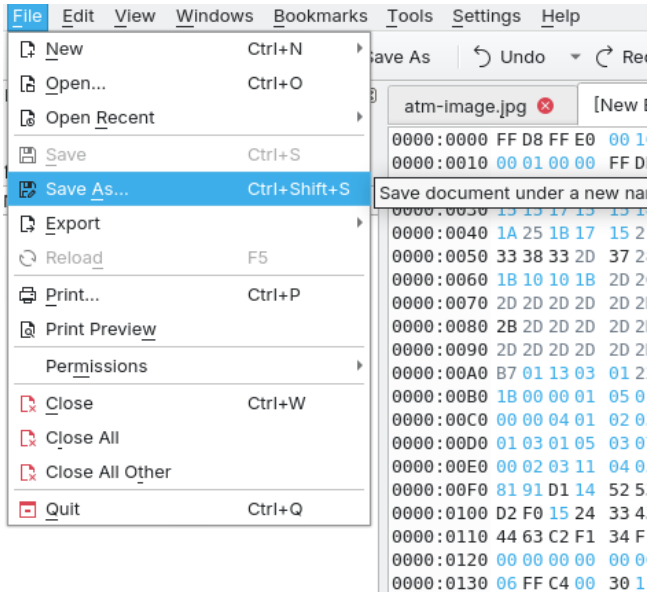

Figure 89: *Saving to a New File.*

In similar fashion to the first image, this image was saved under **atm-image.jpg**.

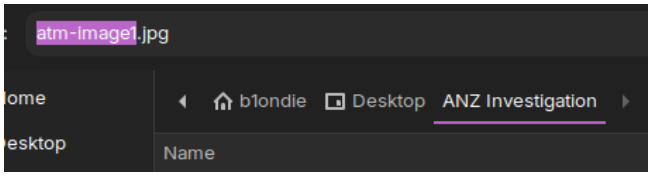

Figure 90: *Renaming for Consistency.*

With both files now in my file system, I can see that both images are operable and display no error, demonstrating their successful extraction.

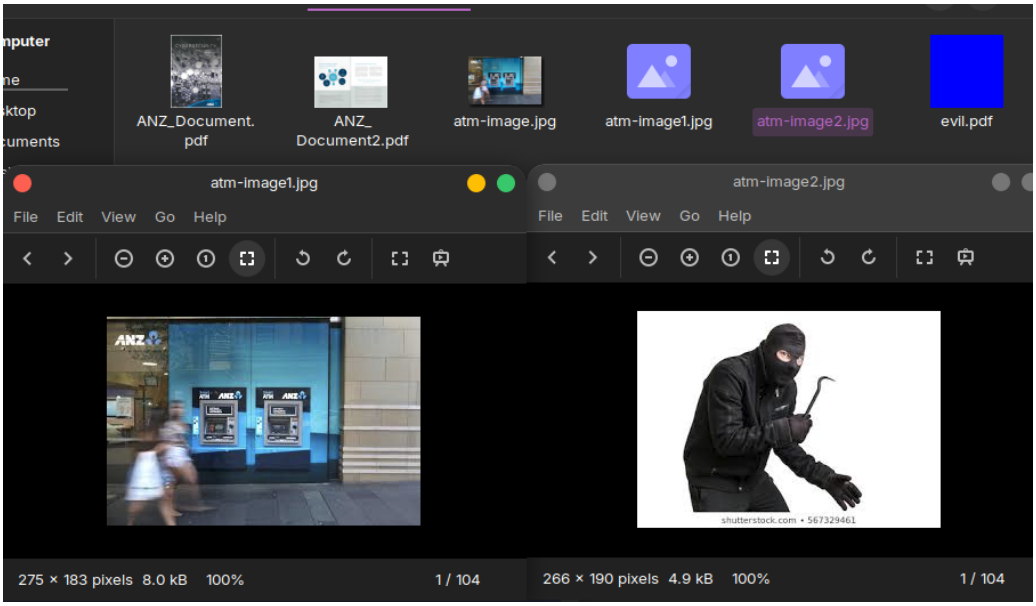

Figure 91: *Two Files Available to View in File System.*

atm-image1.jpg displays two individuals walking in front of an ANZ ATM.

Figure 92: *Full Image 1.*

atm-image2.jpg shows what appears to be a robber with a crowbar in all black.

Figure 93: *Full Image 2.*

8 Section 7: Examine broken.jpg.

8.1 Tools Used

- **Wireshark**
- **Okteta**
- **CyberChef**: an cryptographic tool used for encoding and decoding text

In this section, I searched for the **broken.png** file in the HTTP traffic.

```
    282 HTTP/1.1 200 OK  (JPEG JFIF image
    584 GET /broken.png HTTP/1.1
   1020 HTTP/1.1 200 OK  (image/png)
    614 GET /securepdf.pdf HTTP/1.1
```

Figure 94: *PNG GET Request.*

Once found, I extracted the file and saved it to my file system.

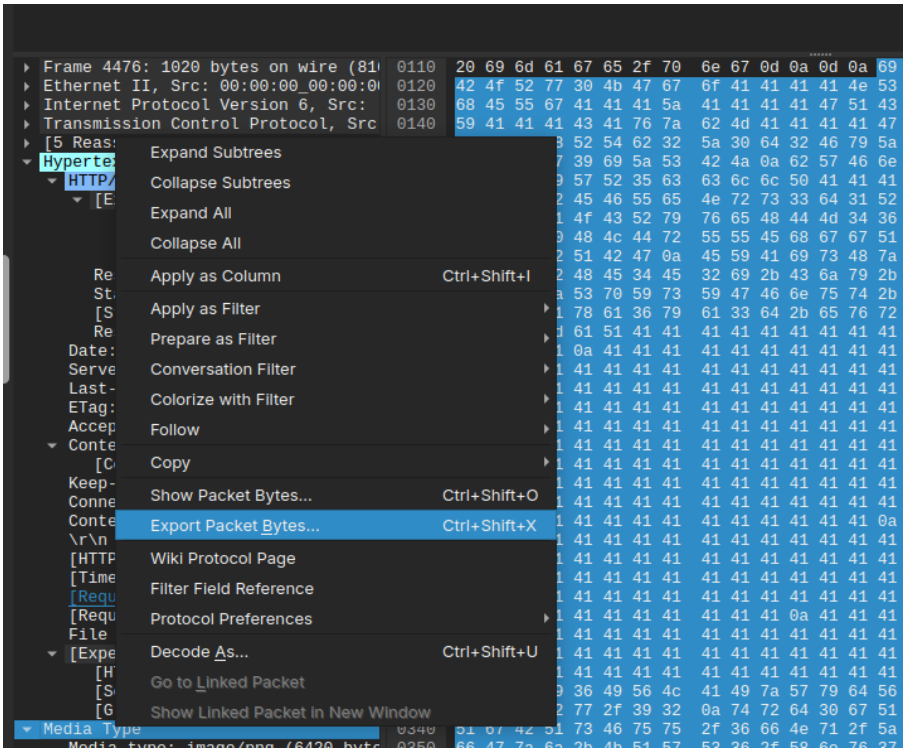

Figure 95: *Export Packet Bytes Options.*

Opening the file presents an error which is unusual, and cause for additional investigation in Wireshark.

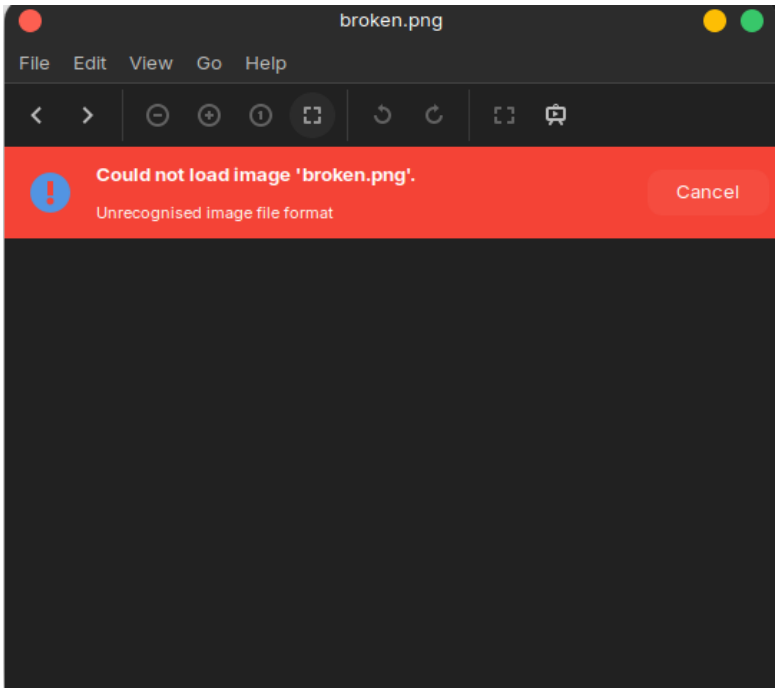

Figure 96: *File Error.*

After analysing the ASCII panel, I noticed the presence of **==g** which is an indicator of a Base 64 encoded string.

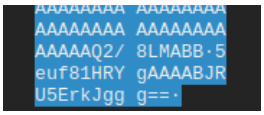

Figure 97: *Base64 Encoding.*

Upon this find, I opened **broken.png** in Okteta.

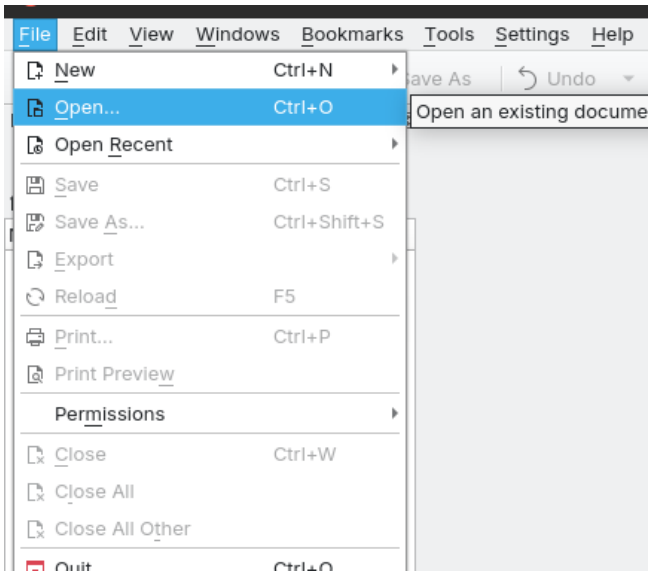

Figure 98: *Opening File in Okteta.*

I selected all of the ASCII text, and ensured I exported it via **Copy > Characters** to ensure it was in the appropriate formatting.

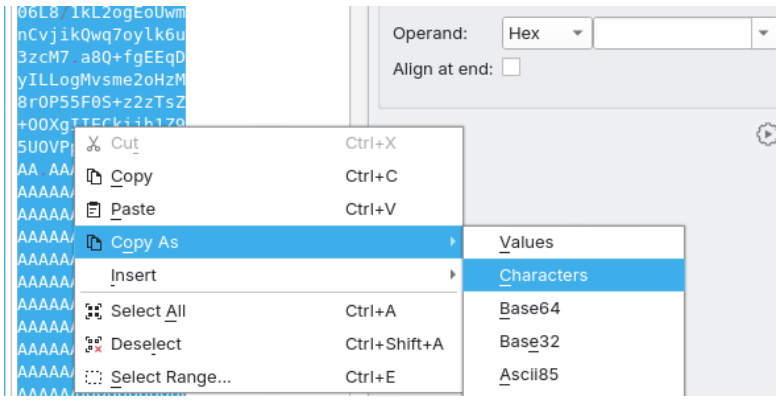

Figure 99: *Copying ASCII Values.*

I begin decoding the ASCII text, by dragging **From Base64** into the CyberChef UI, with all of the default

setting applied.

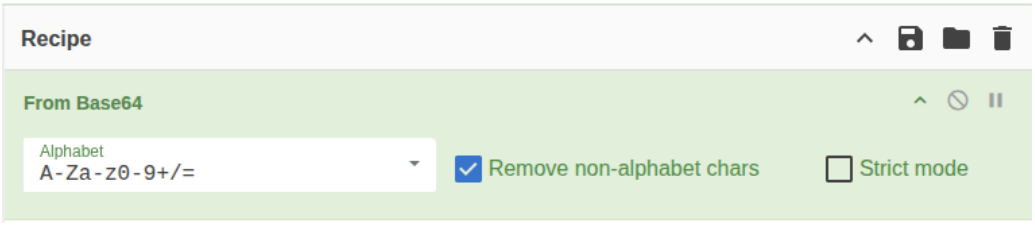

Figure 100: *Using CyberChef to Decode Base64.*

After baking this, I could now see the decoded text in thee bottom right panel.

Figure 101: *New Output.*

To extract the file, I clicked in the **Save output to file** option.

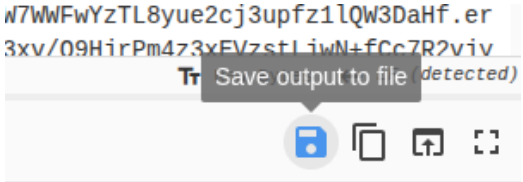

Figure 102: *Saving Outut from CyberChef.*

I named the file **broken.png** to replace the existing version.

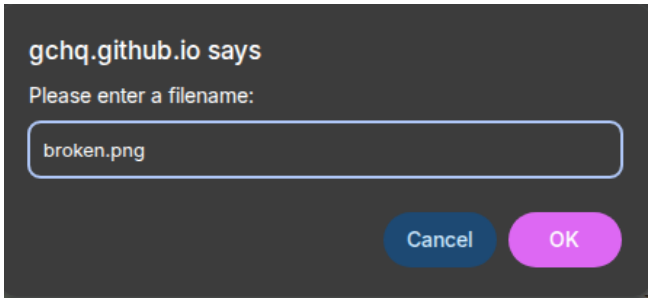

Figure 103: *Inputing File Name.*

Inspecting my file system, I can now see the file that had previously been Base 64 encoded.

Figure 104: *Image Available to View in File System.*

broken.png displays the blue ANZ logo.

Figure 105: *Full Image.*

9 Section 8: Examine securepdf.pdf.

9.1 Tools Used

- **Wireshark**
- **Okteta**

I began this section by finding the **securepdf.pdf** resource sent by the server to the user.

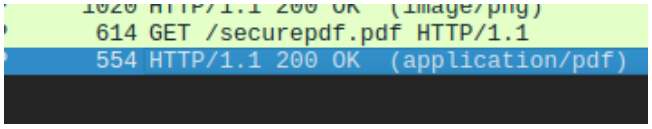

Figure 106: *PDF GET Request.*

I analysed the packet, and noticed some interesting information hidden within the **HTTP segment data (480 bytes)** section of the packet.

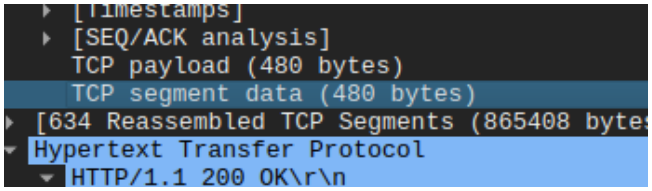

Figure 107: *Packet Details.*

In this segment, it is clear that there is a hidden message in the ASCII portion, which states `Password is "secure"`, possibly hinting at a password in the **securepdf.pdf** file.

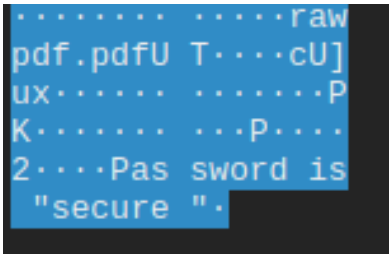

Figure 108: *Hidden Message in Packet Details.*

I extracted the file of interest using the **Export Packet Bytes** option in Wireshark.

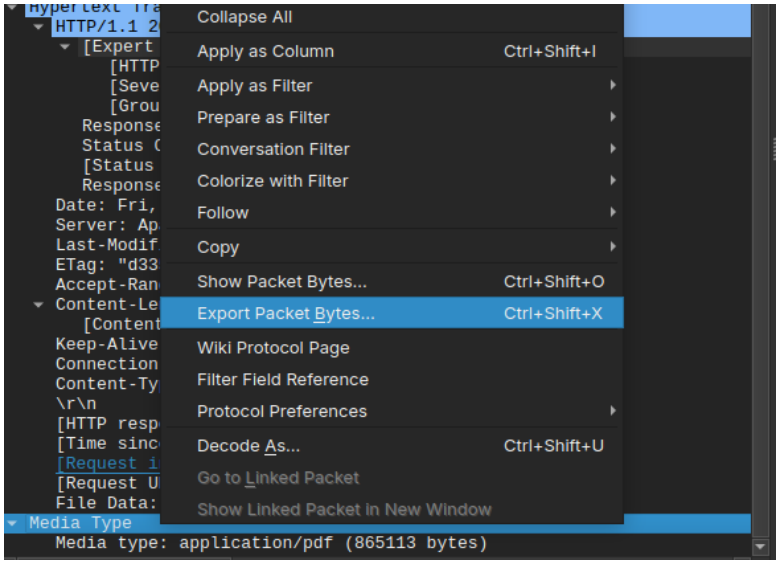

Figure 109: *Export Packet Bytes Option.*

I name the file **securepdf.pdf** to maintain accuracy to the **pcap** file.

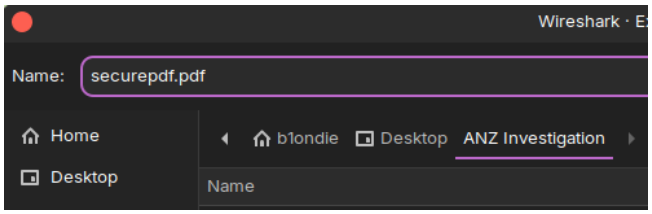

Figure 110: *Extracting securepdf.pdf.*

I am now able to see this file in my file system.

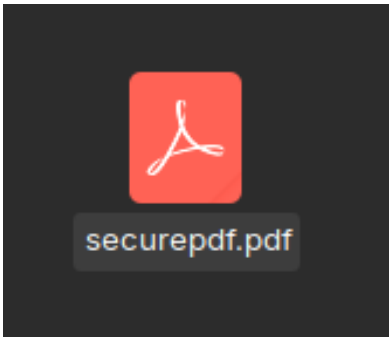

Figure 111: *New File in File System.*

Opening the file displays an error message that indicates this file is, in fact, a ZIP file.

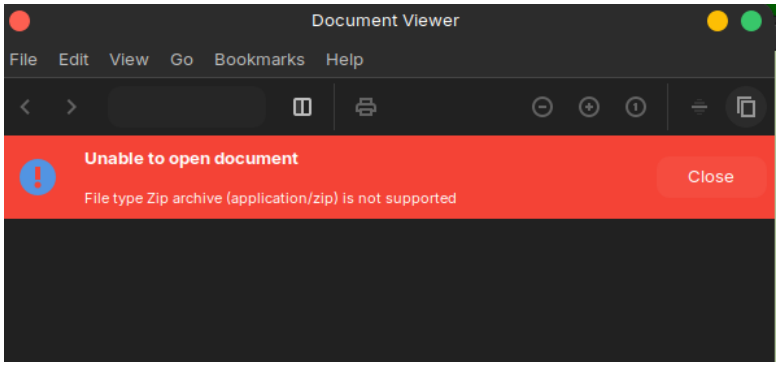

Figure 112: *File Error.*

In an effort to validate this, I proceeded to open the file in Okteta to analyse the file signatures further.

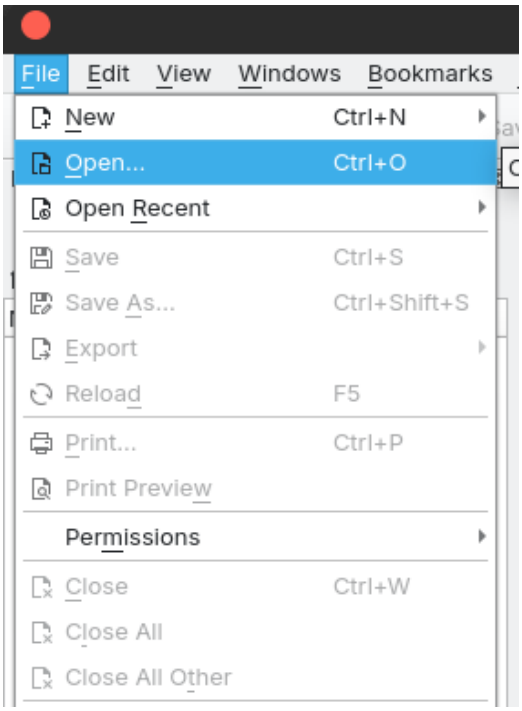

Figure 113: *Opening File in Okteta.*

I ensured I was opening the correct file.

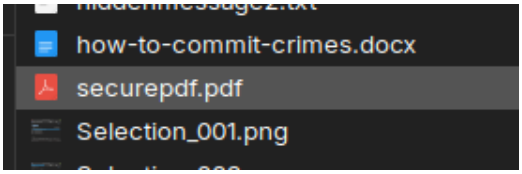

Figure 114: *Selecting Correct File.*

Examining the file signatures, it is clear that this is a ZIP file, denoted by the presence of **504B0304** in the

header.

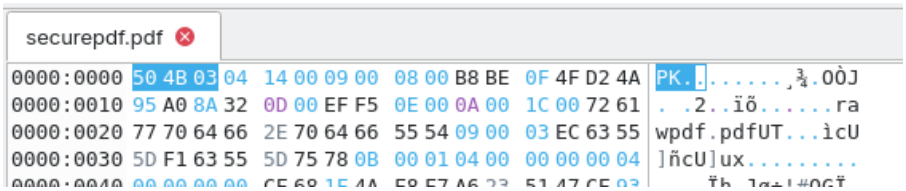

Figure 115: *Analysing File Signature.*

To examine the contents, I renamed **securepdf.pdf** to **securepdf.zip**.

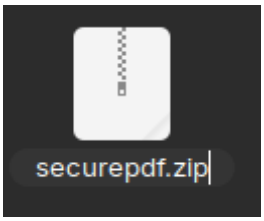

Figure 116: *Renaming File to a .zip File.*

I then extracted the file to the same directory.

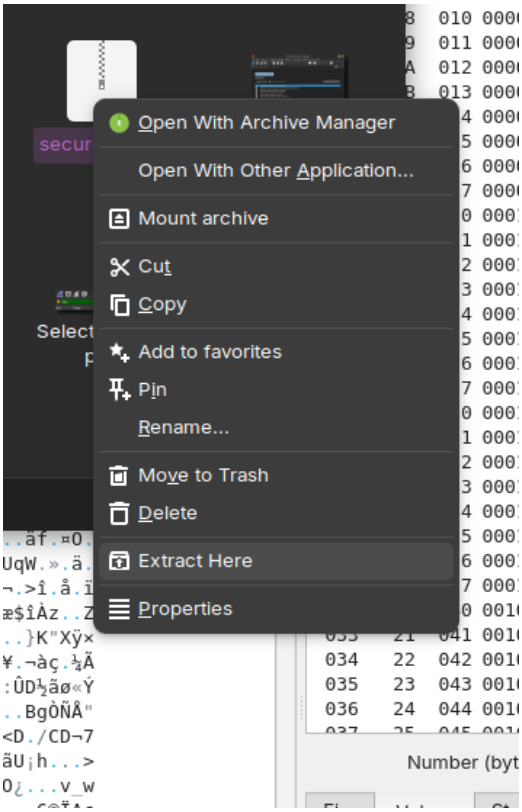

Figure 117: *Extracting the File to Current Directory.*

When extracting, I am asked for a password, whereby I type the password provided to me within the ASCII of the file, **secure**.

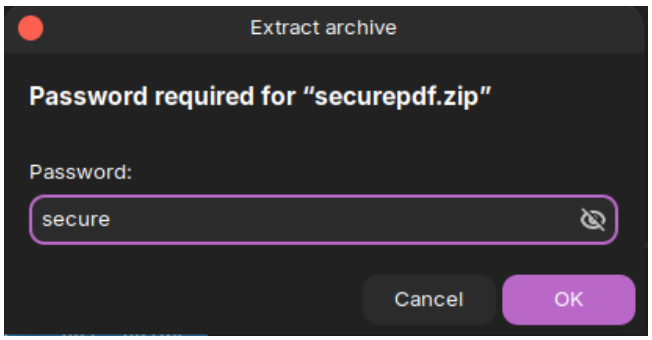

Figure 118: *Inputting Password as Given in Packet Details.*

This is successful, as I can now view an actual PDF file titled **rawpdf.pdf**.

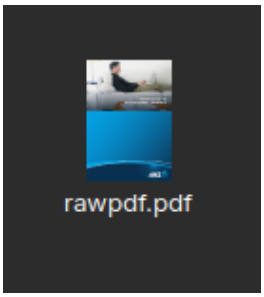

Figure 119: *New PDF Available to View.*

Opening this file within my image viewer, I can observe a PDF file with many items in the contents. Clicking these contents returns nothing, however implies that at some point there may have been more to this PDF file than I can currently see.

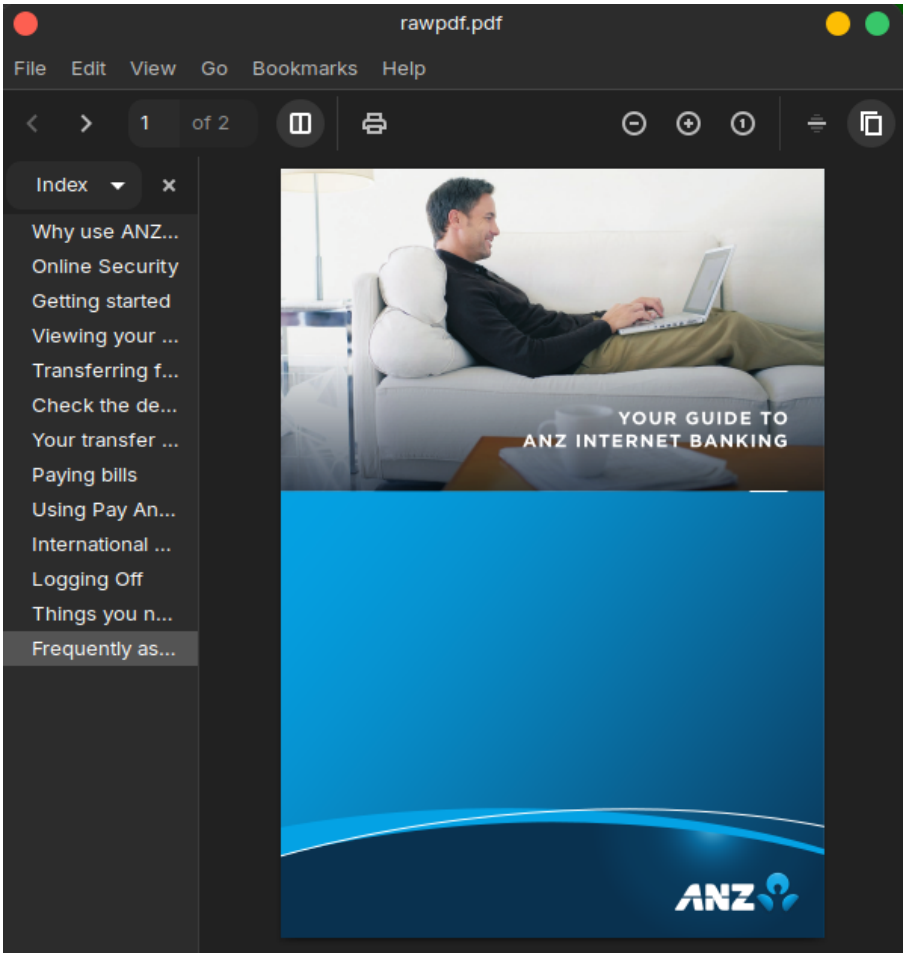

Figure 120: *File Operable.*

Below I can see the first page of the PDF as a title page.

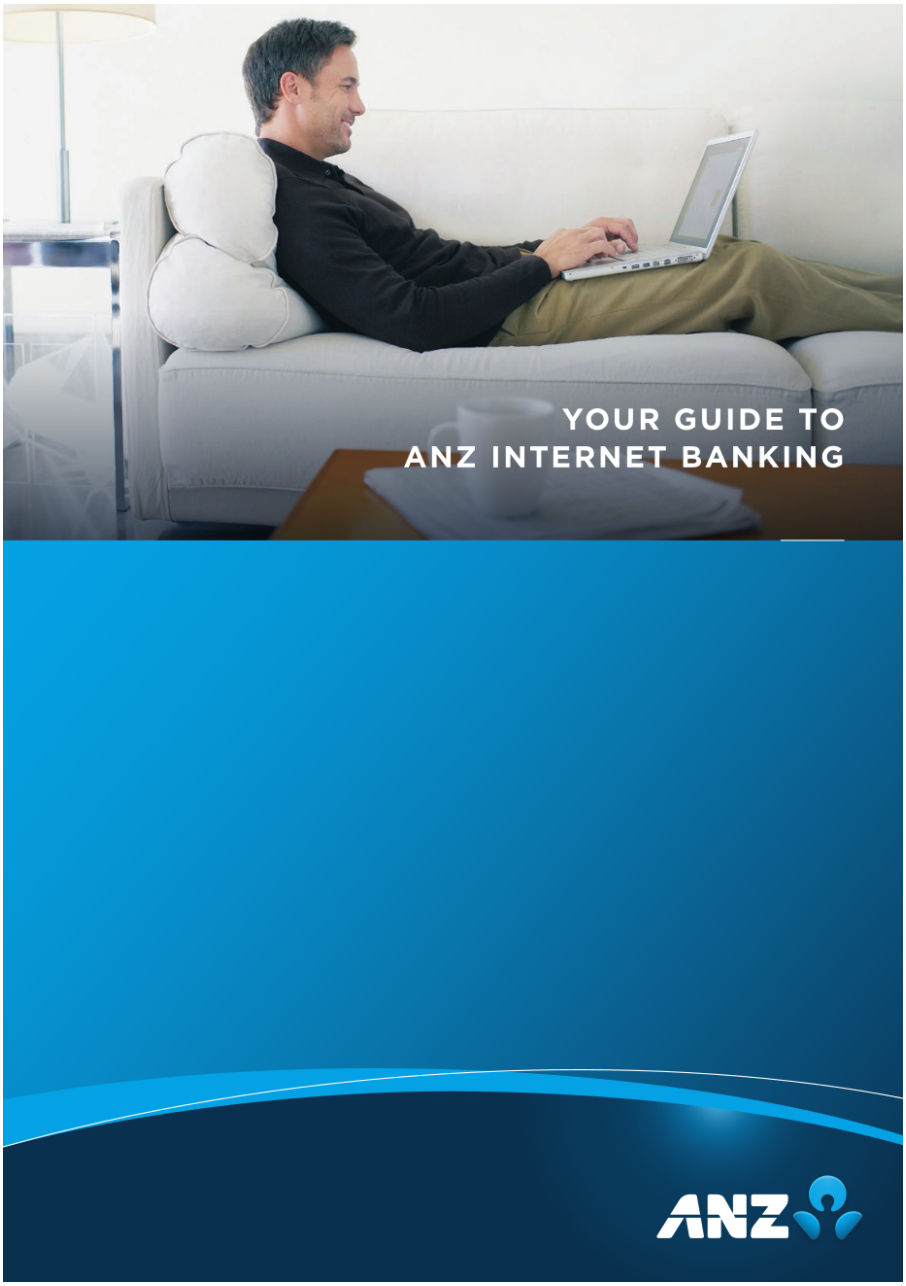

Figure 121: *Full PDF Page 1.*

The second page in this PDF is the Table of Contents page that outlines all of the individual sub-headings in this PDF, although I am unable to view the rest of this file.

TABLE OF CONTENTS

Figure 122: *Full PDF Page 2.*

References

[1] G. Kessler, “Gck’s file signatures table,” 2025. [Online]. Available: https://filesig.search.org/