The Call

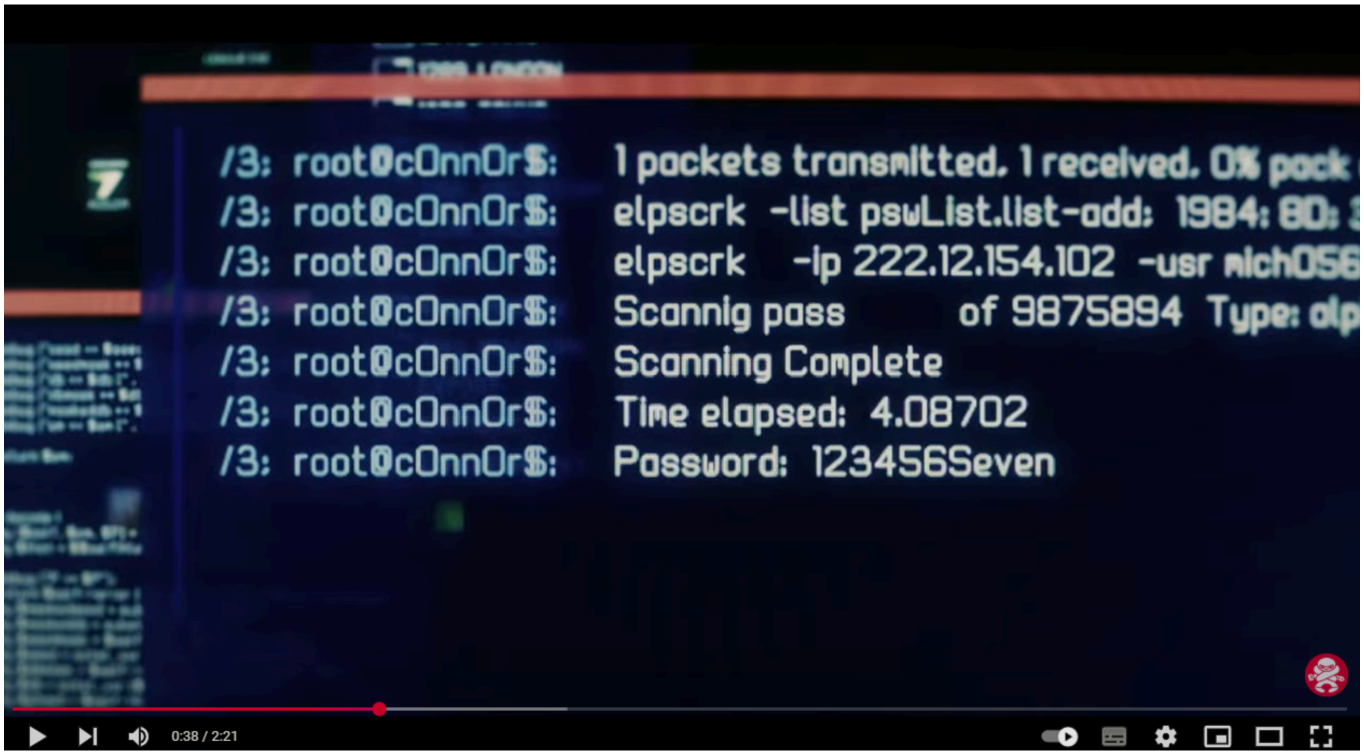
The CTF Players start with an image `cicada.jpg`

After they analyse the picture's metadata they come across a base64 encoded Comment

```
┌──(kali㉿kali)-[~]
└─$ exiftool /home/kali/Desktop/cicada.jpg
ExifTool Version Number         : 13.10
File Name                       : cicada.jpg
Directory                       : /home/kali/Desktop
File Size                       : 75 kB
File Modification Date/Time     : 2025:03:18 14:42:11-04:00
File Access Date/Time           : 2025:03:18 16:46:55-04:00
File Inode Change Date/Time     : 2025:03:18 16:46:55-04:00
File Permissions                : -rwxrw-rw-
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : inches
X Resolution                    : 150
Y Resolution                    : 150
Comment                         :
aHR0cHM6Ly95b3V0dS5iZS8zR2tOY0FldWJsRT9zaT1QY0JYMTM0dXhhaDlUN2ZEJnQ9Mzg=
Image Width                     : 1200
Image Height                    : 675
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 1
Image Size                      : 1200x675
Megapixels                      : 0.810
```

The base64 decodes to the following youtube url:
`aHR0cHM6Ly95b3V0dS5iZS8zR2tOY0FldWJsRT9zaT1QY0JYMTM0dXhhaDlUN2ZEJnQ9Mzg=` ->
https://youtu.be/3GkNcAeublE?si=PcBX134uxah9T7fD&t=38

Playing the trailer at the already selected second will show a password:



```
/3: root@cOnnOr$:   1 packets transmitted. 1 received. 0% pack
/3: root@cOnnOr$:   elpscrk  -list pswList.list-add:  1984: 8D:
/3: root@cOnnOr$:   elpscrk   -ip 222.12.154.102  -usr mich056
/3: root@cOnnOr$:   Scannig pass        of 9875894  Type: alp
/3: root@cOnnOr$:   Scanning Complete
/3: root@cOnnOr$:   Time elapsed:  4.08702
/3: root@cOnnOr$:   Password:  123456Seven
```

Using the specified password to extract the embeded `message.txt`

```
┌──(kali㉿kali)-[~/Desktop]
└─$ steghide extract -sf cicada.jpg -p "123456Seven"
wrote extracted data to "message.txt".
```

and the content of the `message.txt` is:

Not everything is as it seems. Seek the unseen where the wise gather.

This is just a rabbit hole but the player is really close to the right path, a quick look using `binwalk` shows us there is Embedded Encrypted Data inside the file.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ binwalk cicada.jpg

DECIMAL       HEXADECIMAL     DESCRIPTION
------------------------------------------------------------------------------
--
0             0x0             JPEG image data, JFIF standard 1.01
74665         0x123A9         OpenSSL encryption, salted, salt:
0xE702B166FBAFDD75
```

Using `binwalk`, we previously identified an OpenSSL-encrypted payload starting at **offset 74665**. To extract this encrypted data, we use the `dd` command:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ dd if=cicada.jpg bs=1 skip=74665 of=extracted_clue.bin
64+0 records in
64+0 records out
64 bytes copied, 0.000502619 s, 127 kB/s
```

We do a quick check to see what kind of file `extracted_clue.bin` is:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ file extracted_clue.bin
extracted_clue.bin: openssl enc'd data with salted password
```

Since `binwalk` identified OpenSSL encryption, we attempt to decrypt the extracted file. The salt was displayed in the `binwalk` output, meaning it follows OpenSSL's **salted** key derivation scheme. Using OpenSSL, we decrypt it reusing the previous passphrase:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl enc -aes-256-cbc -d -salt -in extracted_clue.bin -out
decrypted_message.txt -k "123456Seven"
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
bad decrypt
4037A044D07F0000:error:1C800064:Provider routines:ossl_cipher_unpadblock:bad
decrypt:../providers/implementations/ciphers/ciphercommon_block.c:107:
```

Seems like we have to use `-pbkdf2`

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl enc -aes-256-cbc -d -salt -pbkdf2 -in extracted_clue.bin -out
decrypted_message.txt -k "123456Seven"

┌──(kali㉿kali)-[~/Desktop]
└─$ cat decrypted_message.txt
https://pastebin.com/raw/nzPenCYz
```

After which we get a pastebin link

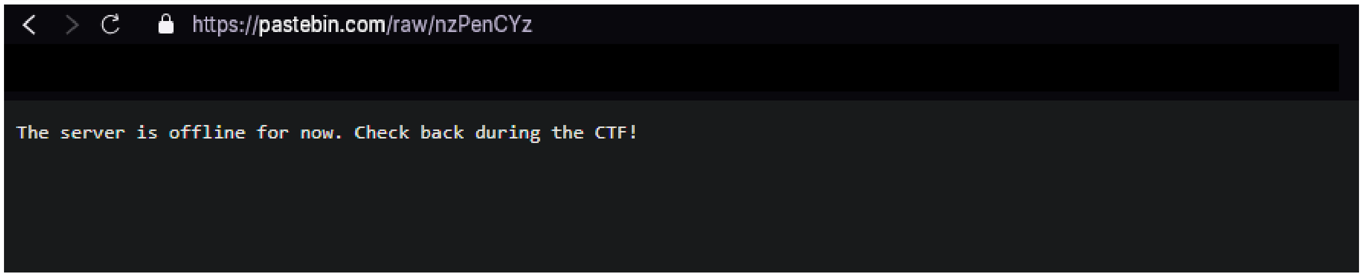
If the player doesn't know it's encrypted using `AES` he could run a `hexdump` or `xxd` on `extracted_clue.bin` which helps confirm the encryption type:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ xxd extracted_clue.bin | head
00000000: 5361 6c74 6564 e702 b166 fbaf dd75 3a9f  Salted...õf...u:.
00000010: 9f1c 2e3a 48cf 75b8 273c a1ff 248a 6bdf  ...:H.u.'<..$.k.
```

The first **6 bytes** read `"Salted"`, a signature **used by OpenSSL for AES encryption**.

Since OpenSSL defaults to **AES-256-CBC** when using `enc` commands, and `binwalk` found an OpenSSL signature, the most likely cipher is **AES-CBC**

Let's see what's on the pastebin website:
`https://pastebin.com/raw/nzPenCYz`

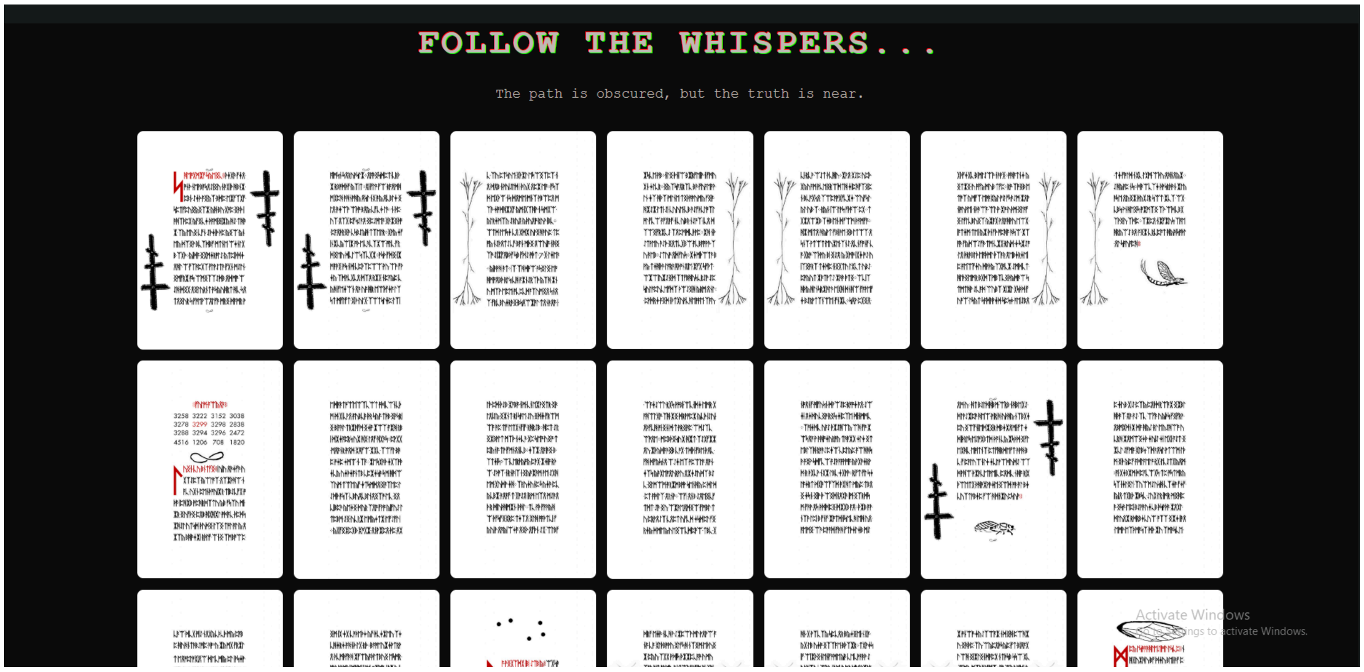


https://pastebin.com/raw/nzPenCYz

```
The server is offline for now. Check back during the CTF!
```

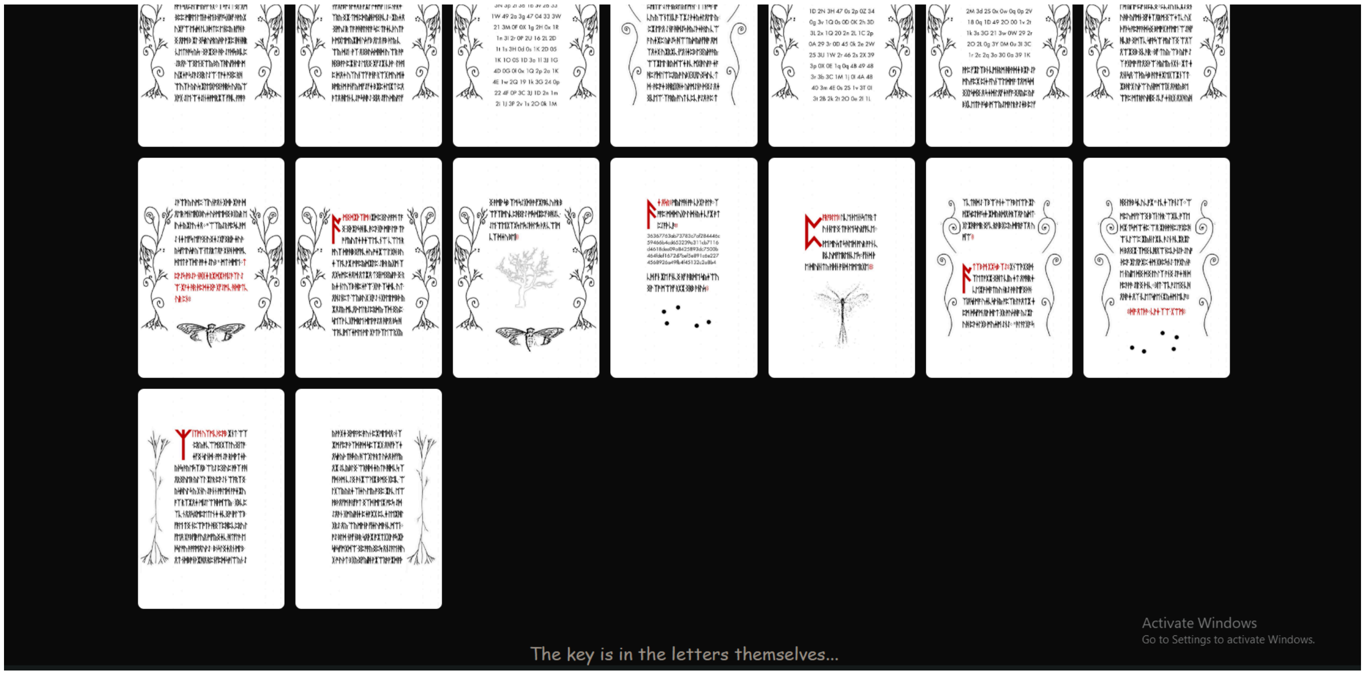
When the CTF will be available I will edit the pastebin to point to the specified website url.

Following up we have a Flask simple website that lands us to the homepage



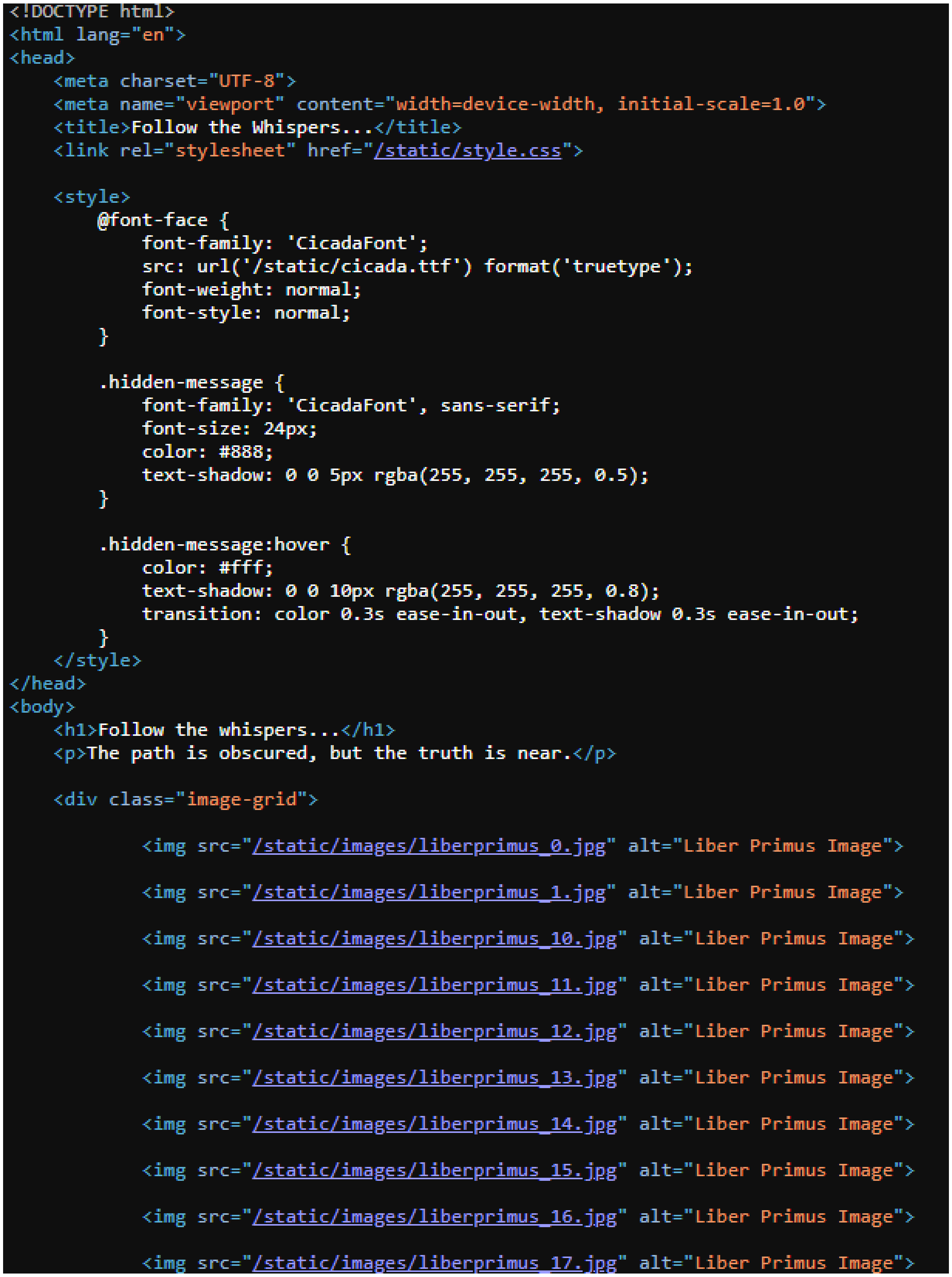
FOLLOW THE WHISPERS...

The path is obscured, but the truth is near.



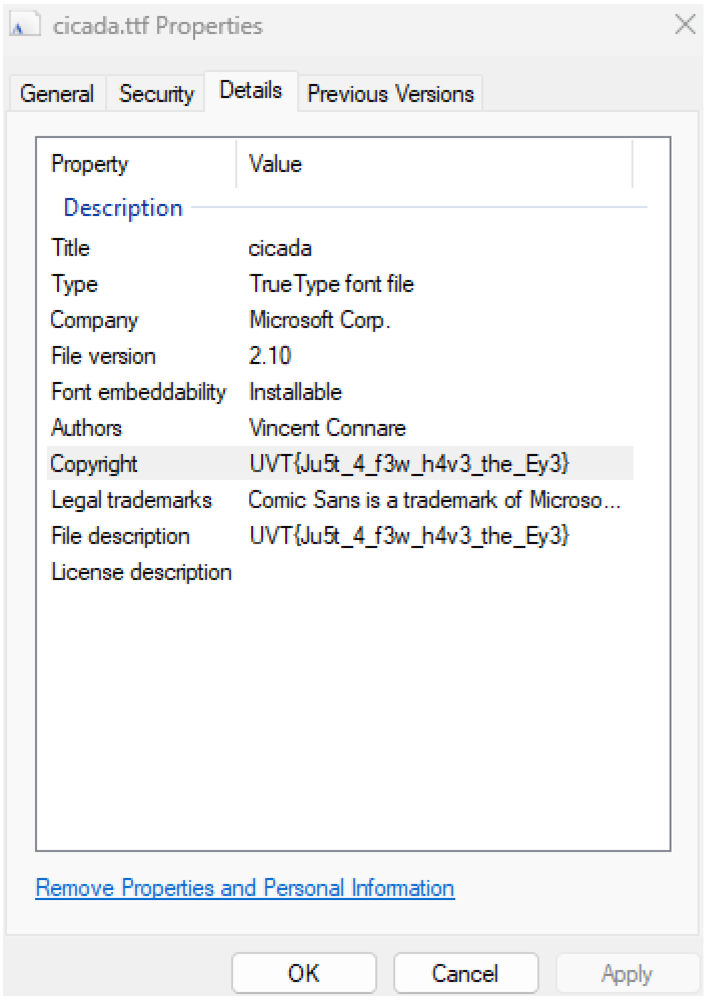
The key is in the letters themselves...

A list of the `Liber Primus` pages followed by some cryptic sentences. The pages are another rabbit hole and if we inspect the page source



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Follow the Whispers...</title>
    <link rel="stylesheet" href="/static/style.css">

    <style>
        @font-face {
            font-family: 'CicadaFont';
            src: url('/static/cicada.ttf') format('truetype');
            font-weight: normal;
            font-style: normal;
        }

        .hidden-message {
            font-family: 'CicadaFont', sans-serif;
            font-size: 24px;
            color: #888;
            text-shadow: 0 0 5px rgba(255, 255, 255, 0.5);
        }

        .hidden-message:hover {
            color: #fff;
            text-shadow: 0 0 10px rgba(255, 255, 255, 0.8);
            transition: color 0.3s ease-in-out, text-shadow 0.3s ease-in-out;
        }
    </style>
</head>
<body>
    <h1>Follow the whispers...</h1>
    <p>The path is obscured, but the truth is near.</p>

    <div class="image-grid">

            <img src="/static/images/liberprimus_0.jpg" alt="Liber Primus Image">

            <img src="/static/images/liberprimus_1.jpg" alt="Liber Primus Image">

            <img src="/static/images/liberprimus_10.jpg" alt="Liber Primus Image">

            <img src="/static/images/liberprimus_11.jpg" alt="Liber Primus Image">

            <img src="/static/images/liberprimus_12.jpg" alt="Liber Primus Image">

            <img src="/static/images/liberprimus_13.jpg" alt="Liber Primus Image">

            <img src="/static/images/liberprimus_14.jpg" alt="Liber Primus Image">

            <img src="/static/images/liberprimus_15.jpg" alt="Liber Primus Image">

            <img src="/static/images/liberprimus_16.jpg" alt="Liber Primus Image">

            <img src="/static/images/liberprimus_17.jpg" alt="Liber Primus Image">
```

The player will notice a weird font named `cicada.ttf`, if we download it and inspect it



cicada.ttf Properties

Property	Value
Description	
Title	cicada
Type	TrueType font file
Company	Microsoft Corp.
File version	2.10
Font embeddability	Installable
Authors	Vincent Connare
Copyright	UVT{Ju5t_4_f3w_h4v3_the_Ey3}
Legal trademarks	Comic Sans is a trademark of Microso...
File description	UVT{Ju5t_4_f3w_h4v3_the_Ey3}
License description	

We will get the flag: `UVT{Ju5t_4_f3w_h4v3_the_Ey3}`

``