Write-Up

TAR UC CTF - SEASON 1

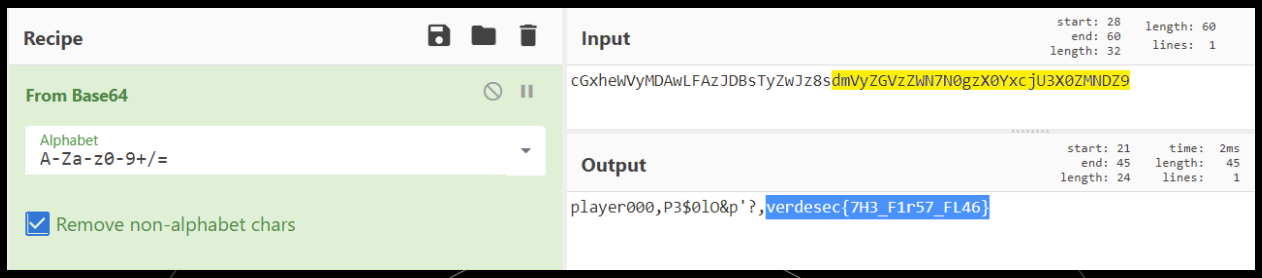
H1D3 N S33K

Table of Content

Flag #01

The flag is base-64 encoded in your password (or “secret access key”) to access the CTF Challenge Portal.

Solution? Decode it.



Flag: verdesec{7H3_F1r57_FL46}

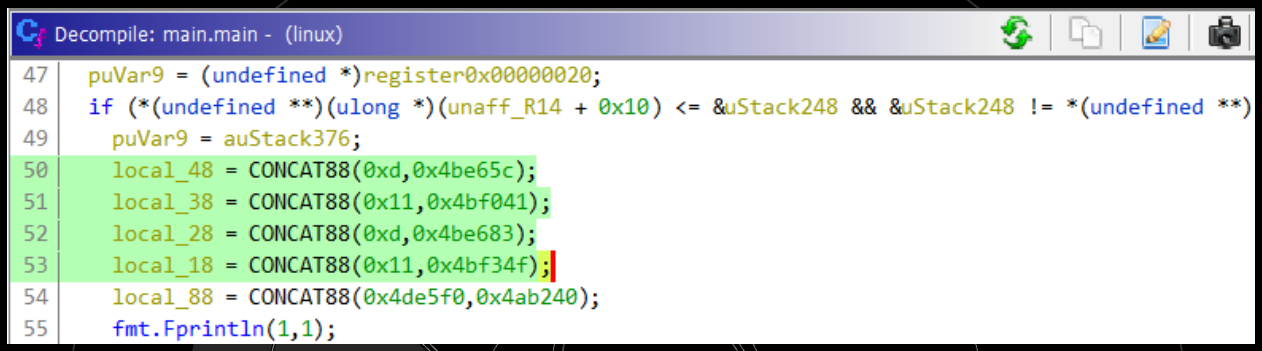
Flag #02

This flag is hidden in /linux at the CTF Challenge Portal.

Download the file with “wget” or other similar tools. The following example uses the Linux command “wget” to download the file.

```
wget https://canyouseeme.ml/linux
```

After downloading, load it into Ghidra (Version 10.0.4 is used here). Check out the main function.

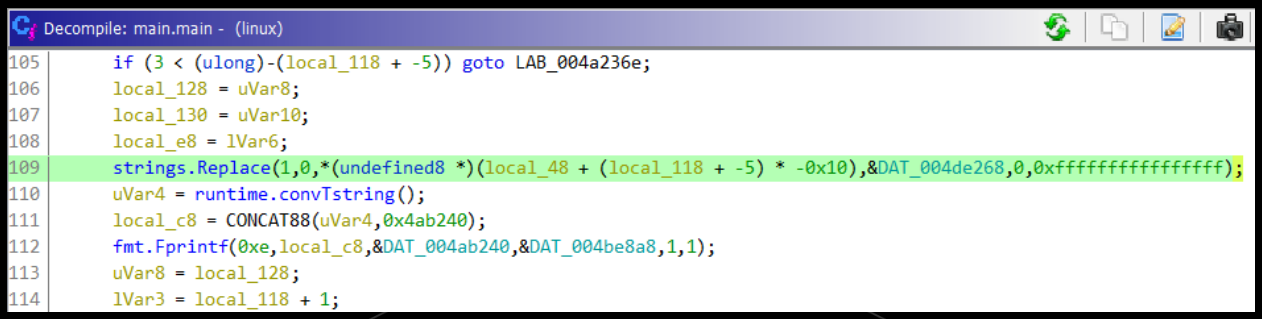


In the main function, if you double click the addresses in line 50 to 53, you will find some suspicious chunks of ASCII-printable bytes:

Variable	Address	Number of Bytes	Bytes
local_48	0x4be65c	0xd = 13	Ma4a5a7a3ara_
local_38	0x4bf041	0x11 = 17	6aUa3a5a5a1aNa6a_
local_28	0x4be683	0xd = 13	NaUaMa8a3ara_
local_18	0x4bf34f	0x11 = 17	vaearadaeasaeaca{

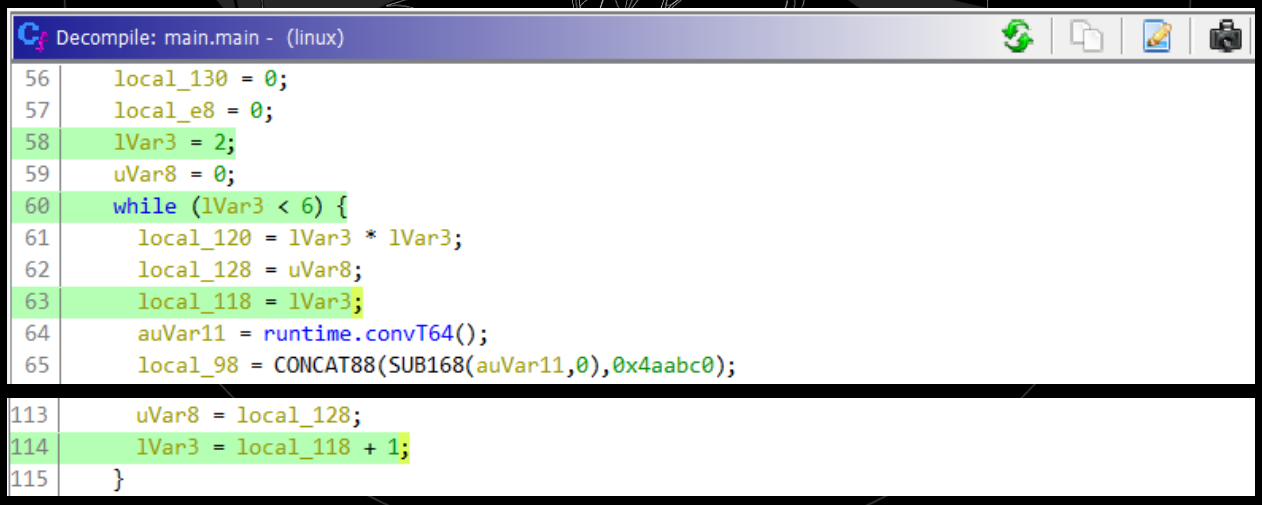
Most probably, you already noticed the pattern: “a” is repeated at the even positions of the bytes, i.e., 2nd, 4th, 6th, etc. At this point, we can firmly guess that those “a”s need to be removed in order to construct the complete flag.

Line 109 confirms our guess.



In line 109, "strings.Replace" is called with:
(1) the third argument (the entire string) set to "local_48 + (local_118 + -5) * -0x10";
(2) the forth argument (the substring to be replaced) set to "DAT_004de268", which double clicked shows its value "a";
(3) the fifth argument (the string that replaces the substring) set to "0" that indicates an empty string;
(4) the sixth argument set to "0xffffffffffffffff" (-1 in 2's complement) that indicates replacing all occurrences.

To find out what "local_118" is holding, see line 63, 58, 60 and 114.



Notice that "local_118" is assigned the value of "lVar3" in line 63, "lVar3" is assigned "2" in line 58, the while loop is set to run with the condition of "lVar3 < 6" in line 60, and "lVar3" is assigned the value of "local_118 + 1". To sum up, "local_118" starts with 2, ends with 5 and increments by 1 in the while loop, i.e., 2, 3, 4, 5.

Now, we know that in the while loop, "`local_48 + (local_118 + -5) * -0x10`" will translate to "`local_18`" in the first iteration, "`local_28`" in the second iteration, "`local_38`" in the third iteration and "`local_48`" in the final iteration.

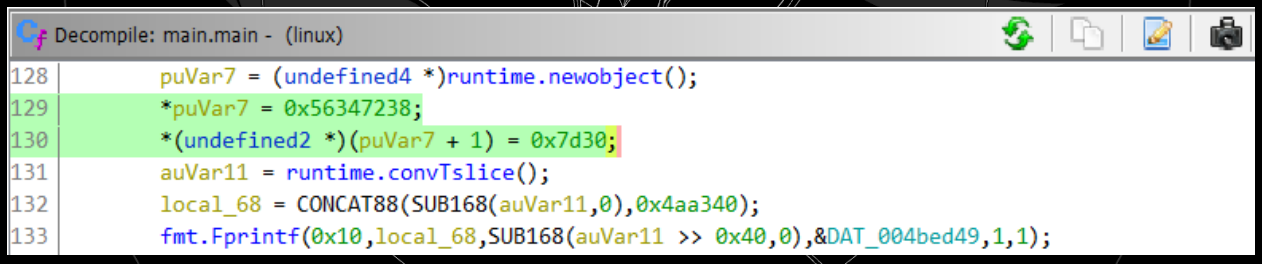
Take the first iteration as an example:

```
   local_48 + (local_118 + -5) * -0x10
 = local_48 + (2 + -5) * -0x10
 = local_48 + (-3) * -0x10
 = local_48 + 0x30
 = local_18
```

To know why "`local_48 + 0x30 = local_18`", look at line 42 to 45.

At this point, we have obtained: "`verdesec{NUM83r_6U3551N6_M4573r_`"

Now, inspect line 129 and 130, where the last part of this flag lies.



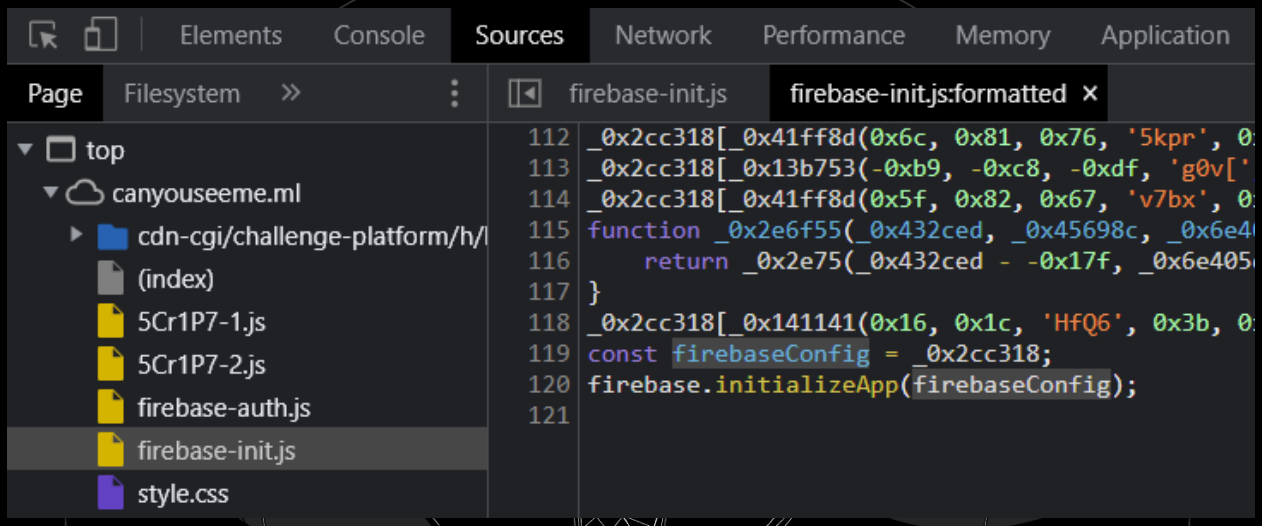
"`0x56347238`" translates to ASCII string "`V4r8`", while "`0x7d30`" translates to ASCII string "`}0`". Both of them are in little-endian format, so rearrange them to get the last part of this flag.

Flag: verdesec{NUM83r_6U3551N6_M4573r_8r4V0}

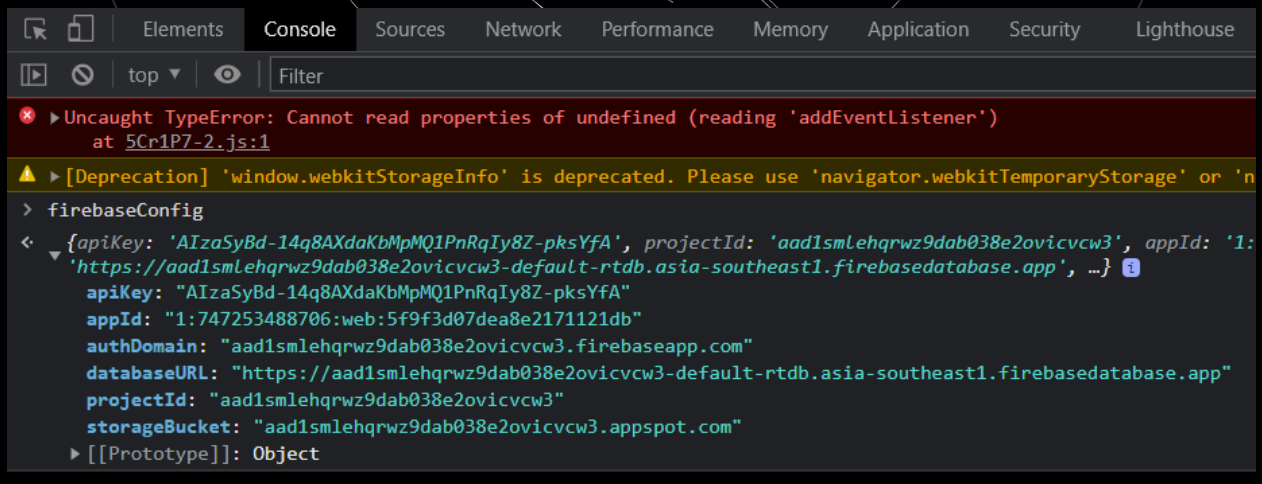
Flag #03

This flag is located at Cloud Storage for Firebase. To get started, find the Firebase configuration.

Take a look at /firebase-init.js at the CTF Challenge Portal and notice that this Firebase app is initialized with the configuration stored in the constant “firebaseConfig”. Side note, the remaining part of the script is obfuscated.



Due to the absence of anti-debugger, we can just output the constant at “Console” to get the Firebase configuration. Note that this Firebase configuration will be used in the process of obtaining this flag, as well as Flag #07, Flag #09 and Flag #10.



Write a script to list all files in Cloud Storage and print their content using the Firebase configuration obtained earlier. Shown below is a fully functional ECMAScript module (.mjs) written to be executed using Node.js with the npm package “firebase@9” installed in prior.

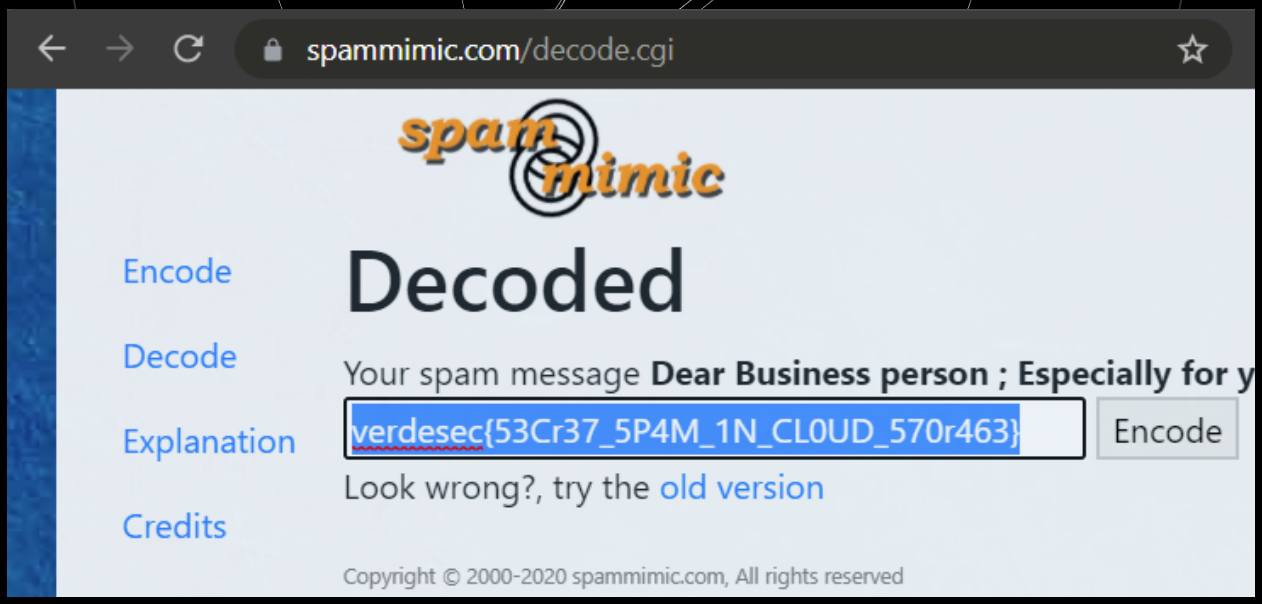
```
import { initializeApp } from "firebase/app";
import { getStorage, ref, listAll, getDownloadURL } from "firebase/storage";
import { get } from "https";

const firebaseConfig = {
    apiKey: "AIzaSyBd-14q8AXdaKbMpMQ1PnRqIy8Z-pksYfA",
    appId: "1:747253488706:web:5f9f3d07dea8e2171121db",
    authDomain: "aad1smlehqrwz9dab038e2ovicvcw3.firebaseapp.com",
    databaseURL:
"https://aad1smlehqrwz9dab038e2ovicvcw3-default-rtdb.asia-southeast1.firebasedatabase.
app",
    projectId: "aad1smlehqrwz9dab038e2ovicvcw3",
    storageBucket: "aad1smlehqrwz9dab038e2ovicvcw3.appspot.com"
};
listAll(ref(getStorage(initializeApp(firebaseConfig)))).then((res) => {
    res.items.forEach((itemRef) => {
        console.log(itemRef.fullPath + "\n==============");
        getDownloadURL(itemRef).then((url) => {
            get(url, (resp) => {
                let data = "";
                resp.on("data", (chunk) => data += chunk);
                resp.on("end", () => console.log(data));
            });
        })
    });
});
```

Run the script and you will get something similar to the output below.

```
5p4MM1m1C.txt
==============
Dear Business person ; Especially for you - this cutting-edge
information . This is a one time mailing there is no
need to request removal if you won't want any more
. This mail is being sent in compliance with Senate
bill 2516 , Title 3 , Section 301 . THIS IS NOT MULTI-LEVEL
MARKETING . Why work for somebody else when you can
become rich as few as 14 DAYS ! Have you ever noticed
the baby boomers are more demanding than their parents
plus society seems to be moving faster and faster !
Well, now is your chance to capitalize on this . WE
will help YOU sell more and SELL MORE . You can begin
at absolutely no cost to you ! But don't believe us
! Prof Anderson of South Carolina tried us and says
"Now I'm rich, Rich, RICH" ! We are a BBB member in
good standing . Don't delay - order today . Sign up
a friend and you get half off ! God Bless . Dear Friend
, This letter was specially selected to be sent to
you ! If you no longer wish to receive our publications
simply reply with a Subject: of "REMOVE" and you will
immediately be removed from our club ! This mail is
being sent in compliance with Senate bill 1622 ; Title
2 ; Section 307 ! THIS IS NOT MULTI-LEVEL MARKETING
! Why work for somebody else when you can become rich
as few as 19 weeks ! Have you ever noticed more people
```

```
than ever are surfing the web and people are much more
likely to BUY with a credit card than cash ! Well,
now is your chance to capitalize on this . WE will
help YOU turn your business into an E-BUSINESS & increase
customer response by 160% ! You can begin at absolutely
no cost to you ! But don't believe us . Ms Jones of
Washington tried us and says "Now I'm rich, Rich, RICH"
! We are a BBB member in good standing ! So make yourself
rich now by ordering immediately . Sign up a friend
and your friend will be rich too . Warmest regards
. Dear Internet user , Your email address has been
submitted to us indicating your interest in our publication
. If you are not interested in our publications and
wish to be removed from our lists, simply do NOT respond
and ignore this mail . This mail is being sent in compliance
with Senate bill 2516 , Title 9 , Section 309 . This
is not multi-level marketing . Why work for somebody
else when you can become rich in 28 WEEKS ! Have you
ever noticed more people than ever are surfing the
web plus nobody is getting any younger ! Well, now
is your chance to capitalize on this ! We will help
you turn your business into an E-BUSINESS and process
your orders within seconds ! You can begin at absolutely
no cost to you ! But don't believe us ! Ms Anderson
who resides in Washington tried us and says "Now I'm
rich many more things are possible" ! We assure you
that we operate within all applicable laws ! DO NOT
DELAY - order today . Sign up a friend and you'll get
a discount of 50% ! Warmest regards !
```

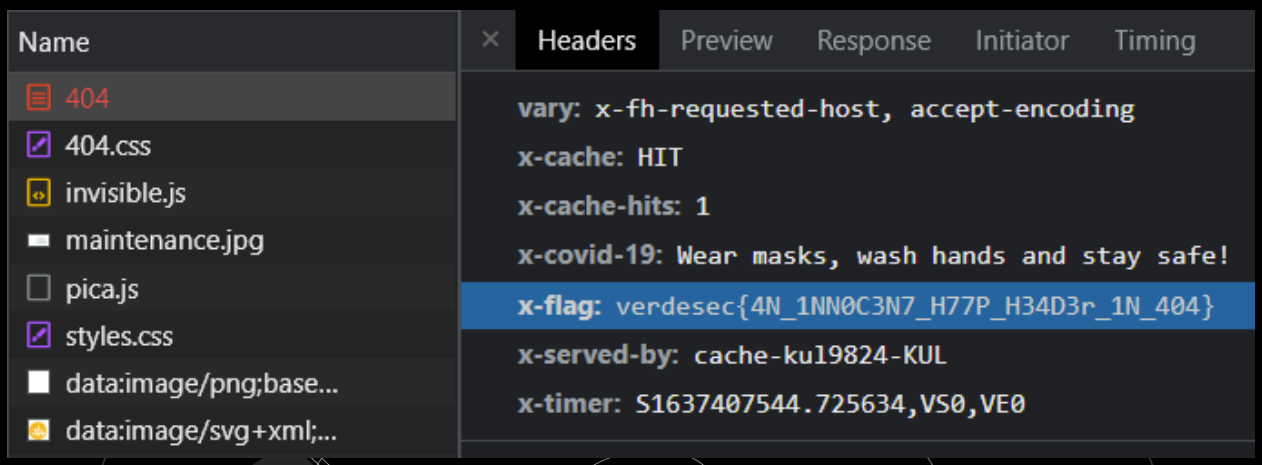
At first, it appears to be like a spam message, but the filename hints at "spammimic". Search online for this keyword and it will eventually lead you to https://www.spammimic.com/decode.shtml.



Flag: verdesec{53Cr37_5P4M_1N_CL0UD_570r463}

Flag #04

Visit any non-existent web page at the CTF Challenge Portal, e.g., /404, to get this flag from the HTTP response header “x-flag”.

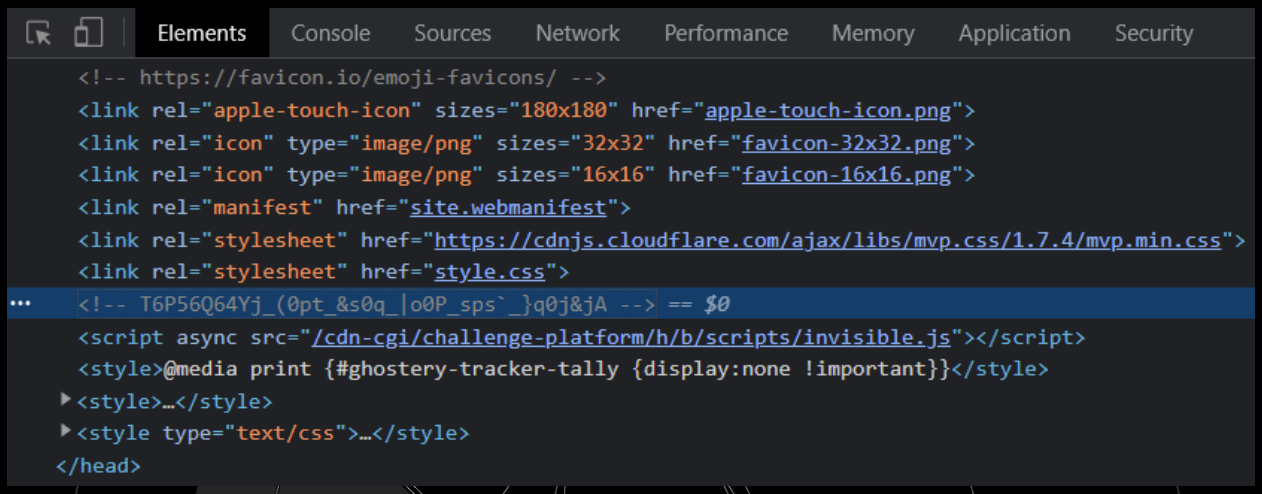


Flag: verdesec{4N_1NN0C3N7_H77P_H34D3r_1N_404}

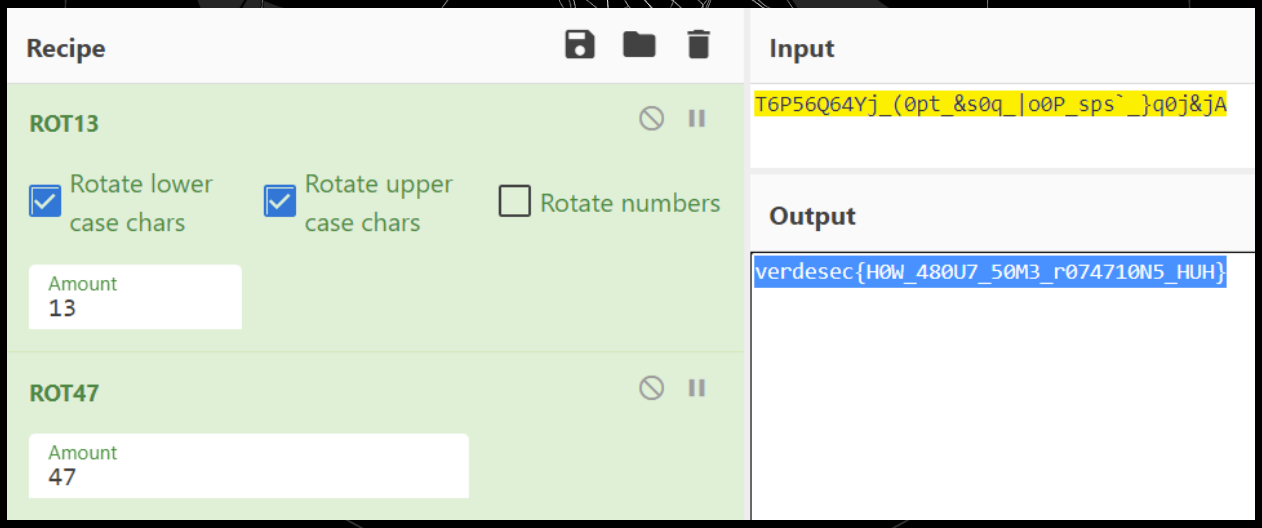
Flag #05

A suspicious HTML comment is hidden in the head section of the CTF Challenge Portal landing page.

<!-- T6P56Q64Yj_(0pt_&s0q_|o0P_sps`_}q0j&jA -->



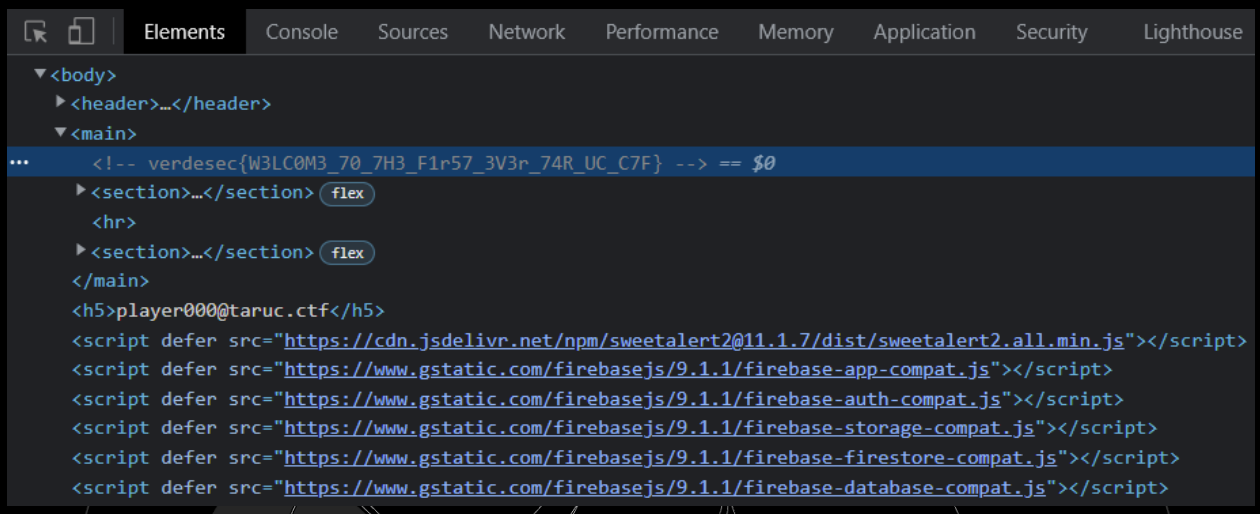
Applying ROT13, followed by ROT47 reveals the flag.



Flag: verdesec{H0W_480U7_50M3_r074710N5_HUH}

Flag #06

This is obvious, just another HTML comment at the CTF Challenge Portal landing page, in plaintext.



Flag: verdesec{W3LC0M3_70_7H3_F1r57_3V3r_74R_UC_C7F}

Flag #07

This flag is also located at Cloud Storage for Firebase, but it is hidden in metadata this time.

Using the Firebase configuration obtained earlier (details at Flag #03), write a script to list all files in Cloud Storage and print their metadata. Shown below is a fully functional ECMAScript module (.mjs) written to be executed using Node.js with the npm package “firebase@9” installed in prior.

```
import { initializeApp } from "firebase/app";
import { getStorage, ref, listAll, getMetadata } from "firebase/storage";

const firebaseConfig = {
    apiKey: "AIzaSyBd-14q8AXdaKbMpMQ1PnRqIy8Z-pksYfA",
    appId: "1:747253488706:web:5f9f3d07dea8e2171121db",
    authDomain: "aad1smlehqrwz9dab038e2ovicvcw3.firebaseapp.com",
    databaseURL:
"https://aad1smlehqrwz9dab038e2ovicvcw3-default-rtdb.asia-southeast1.firebasedatabase.
app",
    projectId: "aad1smlehqrwz9dab038e2ovicvcw3",
    storageBucket: "aad1smlehqrwz9dab038e2ovicvcw3.appspot.com"
};
listAll(ref(getStorage(initializeApp(firebaseConfig)))).then((res) => {
    res.items.forEach((itemRef) => {
        console.log(itemRef.fullPath + "\n===============");
        getMetadata(itemRef).then((metadata) => {
            console.log(metadata);
        });
    });
});
```

You can find the flag in “customMetadata”.

```
5p4MM1m1C.txt
===============
{
  type: 'file',
  bucket: 'aad1smlehqrwz9dab038e2ovicvcw3.appspot.com',
  generation: '1636816408820332',
  metageneration: '2',
  fullPath: '5p4MM1m1C.txt',
  name: '5p4MM1m1C.txt',
  size: 2940,
  timeCreated: '2021-11-13T15:13:28.821Z',
  updated: '2021-11-13T15:15:24.341Z',
  md5Hash: 'AHwELQEDinuJxcS2qS+0Iw==',
  cacheControl: undefined,
  contentDisposition: "inline; filename*=utf-8''5p4MM1m1C.txt",
  contentEncoding: 'identity',
  contentLanguage: undefined,
  contentType: 'text/plain',
  customMetadata: { flag: 'verdesec{F1L3_M374D474_0N_CL0UD_570r463}' }
}
```

Flag: verdesec{F1L3_M374D474_0N_CL0UD_570r463}

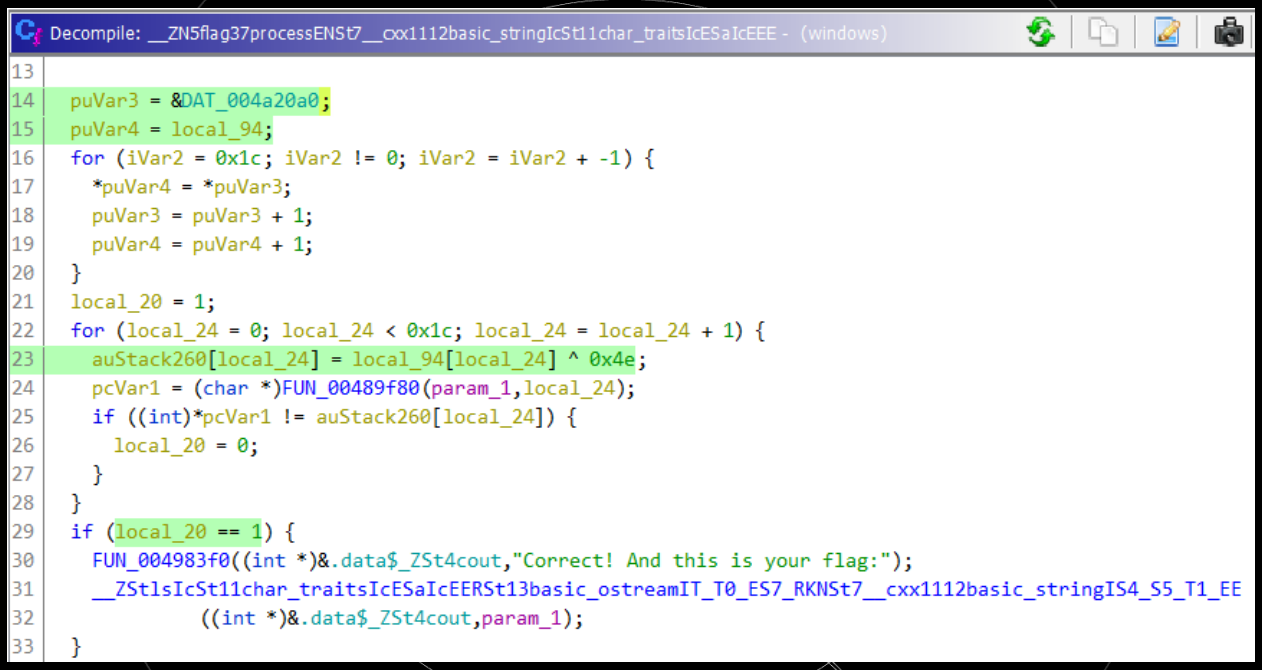
Flag #08

Same with Flag #11 and Flag #17, this flag can be extracted from /windows at the CTF Challenge Portal.

Download the file with “wget” or other similar tools. The following example uses the Linux command “wget” to download the file.

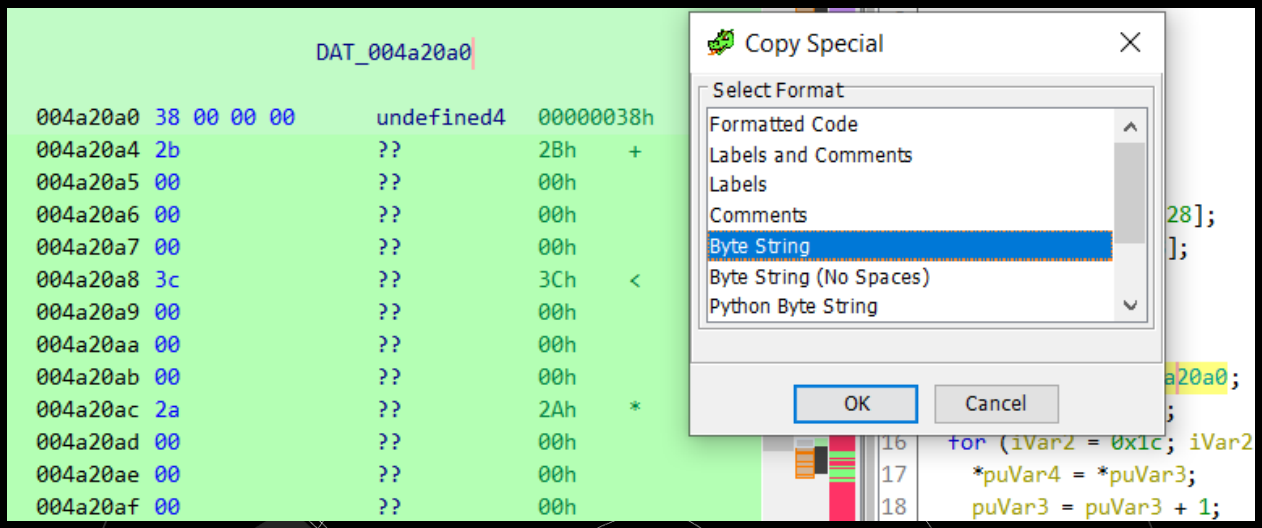
```
wget https://canyouseeme.ml/windows
```

After downloading, load it into Ghidra (Version 10.0.4 is used here). Check out the decompiled code of the function below that is being called in the main function for “Flag 3”.

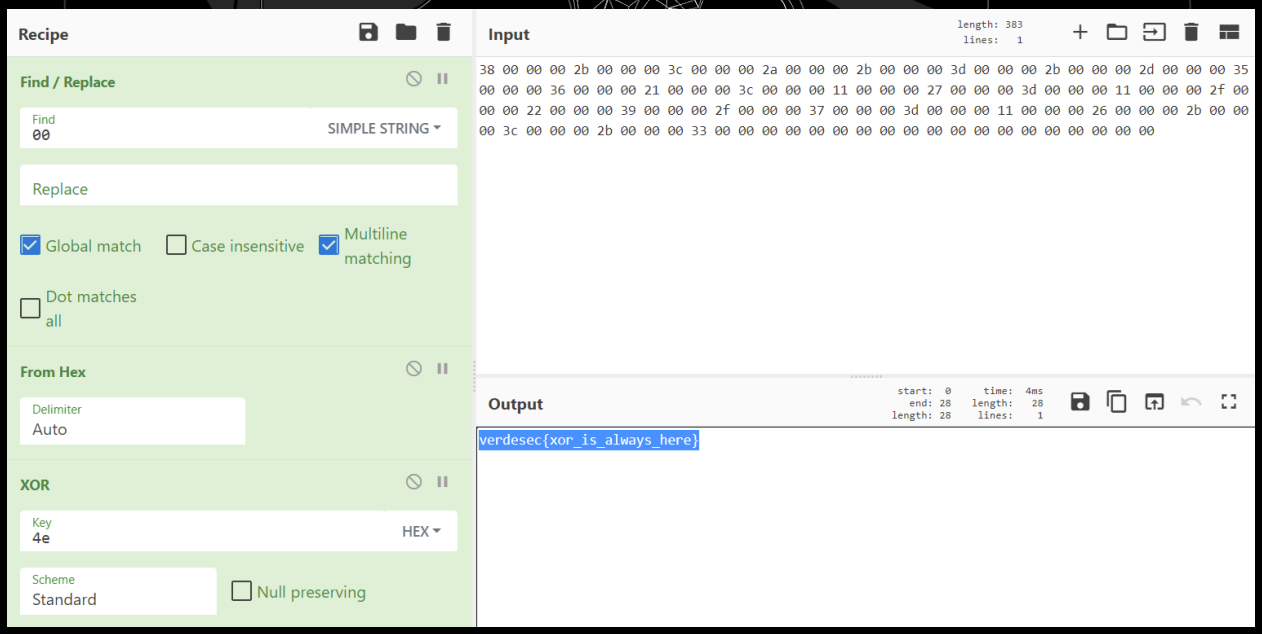


From the decompiled code, we know that “`local_20 == 1`” (line 29) will print out the flag. In order to do so, we need our input to match with “`local_94[local_24] ^ 0x4e`” (line 23) in which “`local_24`” is the index of the loop. As for “`local_94`”, we can notice from line 14 to 20 that it is essentially the value stored at “`DAT_004a20a0`”. Meanwhile, the symbol “`^`” indicates XOR. Now, we are confident to say that the value stored at “`DAT_004a20a0`” is the result of an XOR operation of the flag and the byte “`0x4e`” (in hex).

Select the entire chunk of “DAT_004a20a0” by highlighting them, right click, select “Copy Special...” and then select “Byte String”, in order to copy the values as a byte string.



Paste it into CyberChef and apply the following recipe to get the flag. The key “4e” (in hex) obtained earlier is used in another XOR operation here for the decryption of the byte string to the flag, as XOR operations are symmetrical (reverse XOR = XOR).



Flag: verdesec{xor_is_always_here}

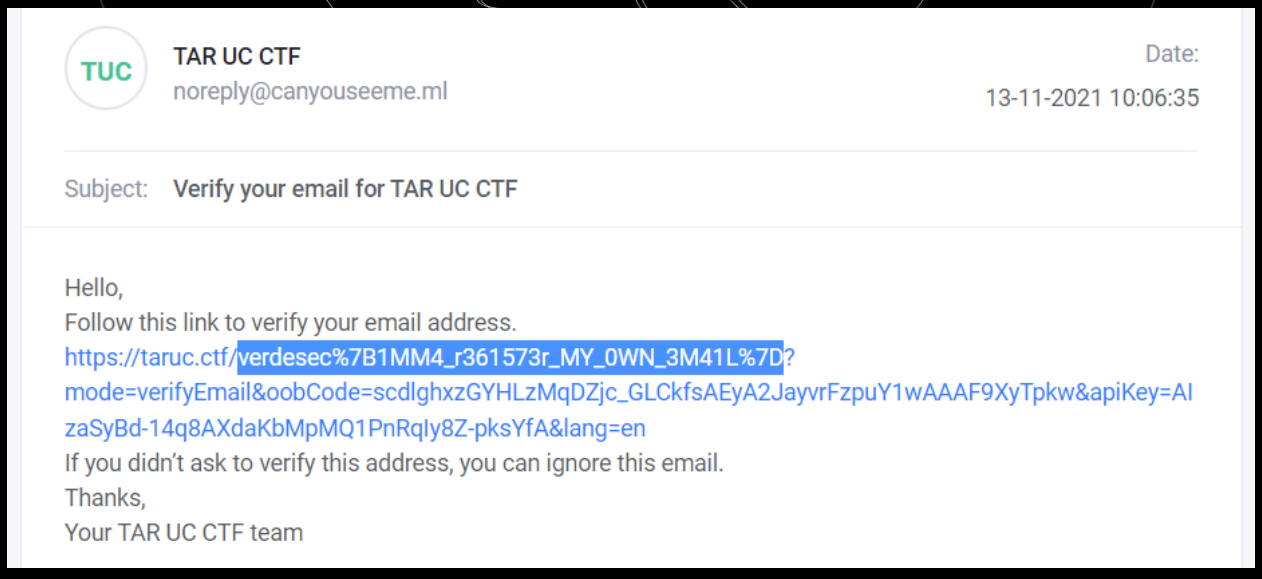
Flag #09

Firebase Authentication is where this flag hides.

Using the Firebase configuration obtained earlier (details at Flag #03), write a script to create a new user account at Firebase Authentication and trigger it to send a verification email. Shown below is a fully functional ECMAScript module (.mjs) written to be executed using Node.js with the npm package "firebase@9" installed in prior. Remember to replace "*<your-email>*" with your email.

```
import { initializeApp } from "firebase/app";
import { getAuth, createUserWithEmailAndPassword, sendEmailVerification } from
"firebase/auth";

const firebaseConfig = {
    apiKey: "AIzaSyBd-14q8AXdaKbMpMQ1PnRqIy8Z-pksYfA",
    appId: "1:747253488706:web:5f9f3d07dea8e2171121db",
    authDomain: "aad1smlehqrwz9dab038e2ovicvcw3.firebaseapp.com",
    databaseURL:
"https://aad1smlehqrwz9dab038e2ovicvcw3-default-rtdb.asia-southeast1.firebasedatabase.
app",
    projectId: "aad1smlehqrwz9dab038e2ovicvcw3",
    storageBucket: "aad1smlehqrwz9dab038e2ovicvcw3.appspot.com"
};
const auth = getAuth(initializeApp(firebaseConfig));
createUserWithEmailAndPassword(auth, "<your-email>", "123456").then((userCredential)
=> {
    sendEmailVerification(userCredential.user);
});
```

You should receive a similar email shortly after running the script. Note that the flag included in the email is URL-encoded, decode it.



Flag: verdesec{1MM4_r361573r_MY_0WN_3M41L}

Flag #10

Write a script to access all data stored in this Firebase Realtime Database using the Firebase configuration obtained earlier (details at Flag #03). Shown below is a fully functional ECMAScript module (.mjs) written to be executed using Node.js with the npm package “firebase@9” installed in prior.

```
import { initializeApp } from "firebase/app";
import { getDatabase, ref, get } from "firebase/database";

const firebaseConfig = {
  apiKey: "AIzaSyBd-14q8AXdaKbMpMQ1PnRqIy8Z-pksYfA",
  appId: "1:747253488706:web:5f9f3d07dea8e2171121db",
  authDomain: "aad1smlehqrwz9dab038e2ovicvcw3.firebaseapp.com",
  databaseURL:
"https://aad1smlehqrwz9dab038e2ovicvcw3-default-rtdb.asia-southeast1.firebasedatabase.
app",
  projectId: "aad1smlehqrwz9dab038e2ovicvcw3",
  storageBucket: "aad1smlehqrwz9dab038e2ovicvcw3.appspot.com"
};
get(ref(getDatabase(initializeApp(firebaseConfig)))).then((snapshot) => {
  console.log(snapshot.val());
});
```

Run the script and you can spot the flag in the output.

```
{
  flag: 'verdesec{53CUr17Y_rUL35_84D_Pr4C71C3}',
  welcome: {
    subtitle: 'Hello, World!',
    title: 'Can You See Me?'
  }
}
```

Flag: verdesec{53CUr17Y_rUL35_84D_Pr4C71C3}

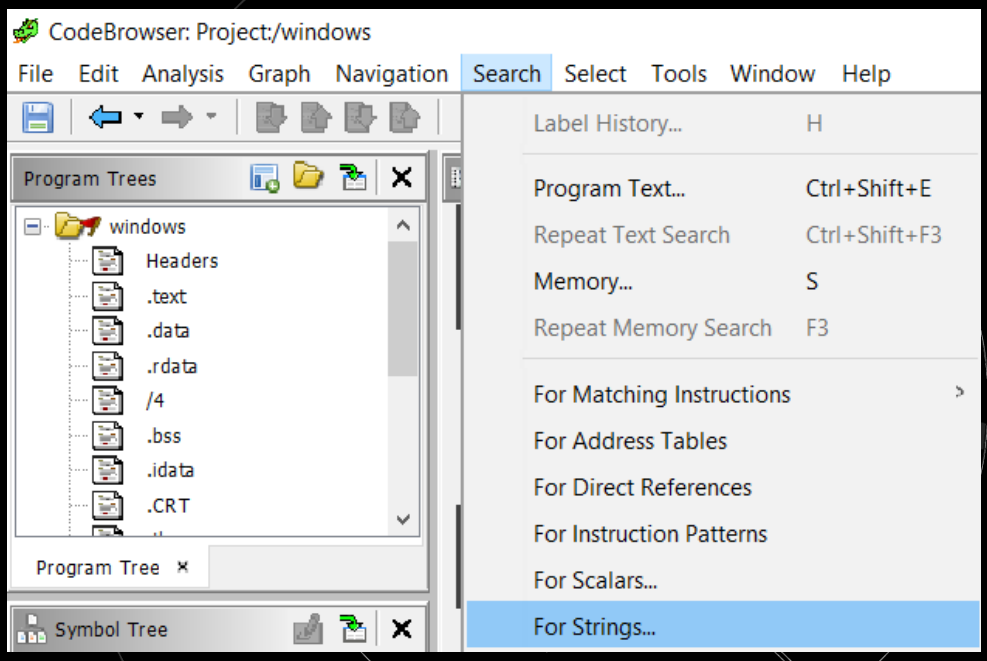
Flag #11

This flag can be found in /windows at the CTF Challenge Portal. Note that this method is also applicable for capturing Flag #17.

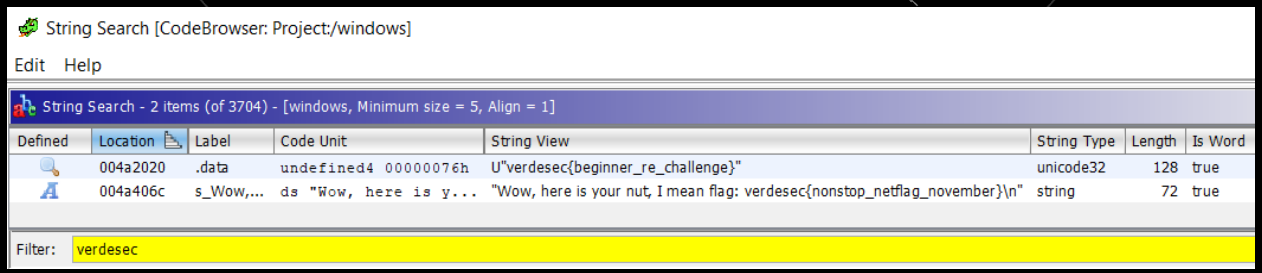
Download the file with “wget” or other similar tools. The following example uses the Linux command “wget” to download the file.

```
wget https://canyouseeme.ml/windows
```

After downloading, load it into Ghidra (Version 10.0.4 is used here), and search for strings.



Filter the result using any appropriate keyword, e.g., “verdesec”, and you’ve got the flag.



Flag: verdesec{beginner_re_challenge}

Flag #12

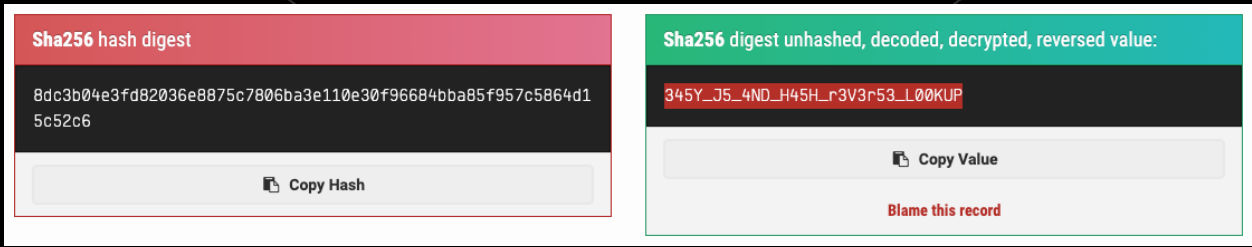
This flag is located in /5Cr1P7-2.js at the CTF Challenge Portal.

The two highlighted strings shown below are originally encoded in JSFuck. Decode them and you will get “SHA-256” and a SHA-256 digest.

```
document.getElementsByTagName("button")[1].addEventListener("click", (e) => {
  e.preventDefault();
  const i = document.getElementsByTagName("input")[0].value;
  (async (m) =>
    Array.from(
      new Uint8Array(
        await crypto.subtle.digest("SHA-256", new TextEncoder().encode(m))
      )
    )
      .map((b) => b.toString(16).padStart(2, "0"))
      .join(""))(i).then((d) =>
    Swal.fire(
      "",
      ...(d ===
      "8dc3b04e3fd82036e8875c7806ba3e110e30f96684bba85f957c5864d15c52c6"
        ? [
            `Bravo! Here goes your flag: <p><code>verdesec{${i}}</code></p>`,
            "success",
          ]
        : ["Nope, this is not what I'm looking for...", "error"])
    )
  );
});

```

8dc3b04e3fd82036e8875c7806ba3e110e30f96684bba85f957c5864d15c52c6

This digest can be “reversed” at https://md5hashing.net/hash/sha256.



Flag: verdesec{345Y_J5_4ND_H45H_r3V3r53_L00KUP}

Flag #13

This flag is hidden in the EXIF metadata of /apple-touch-icon.png at the CTF Challenge Portal.

A base-64 string is spotted at the “Comment” entry.

```
└─$ exiftool apple-touch-icon.png
ExifTool Version Number         : 12.34
File Name                       : apple-touch-icon.png
Directory                       : .
File Size                       : 10 KiB
File Modification Date/Time     : 2021:09:27 08:35:22+08:00
File Access Date/Time           : 2021:11:22 22:10:10+08:00
File Inode Change Date/Time     : 2021:10:27 00:24:19+08:00
File Permissions                : -rwxrwxrwx
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 180
Image Height                    : 180
Bit Depth                       : 8
Color Type                      : RGB with Alpha
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
SRGB Rendering                  : Perceptual
Author                          : zhixuan
Comment                         : dmVyZGVzZWN7ODQ1MzY0XzNOQzBEM0RfMU5fRDQ3NF84MFU3X0Q0NzR9
Image Size                      : 180x180
Megapixels                      : 0.032
```

dmVyZGVzZWN7ODQ1MzY0XzNOQzBEM0RfMU5fRDQ3NF84MFU3X0Q0NzR9

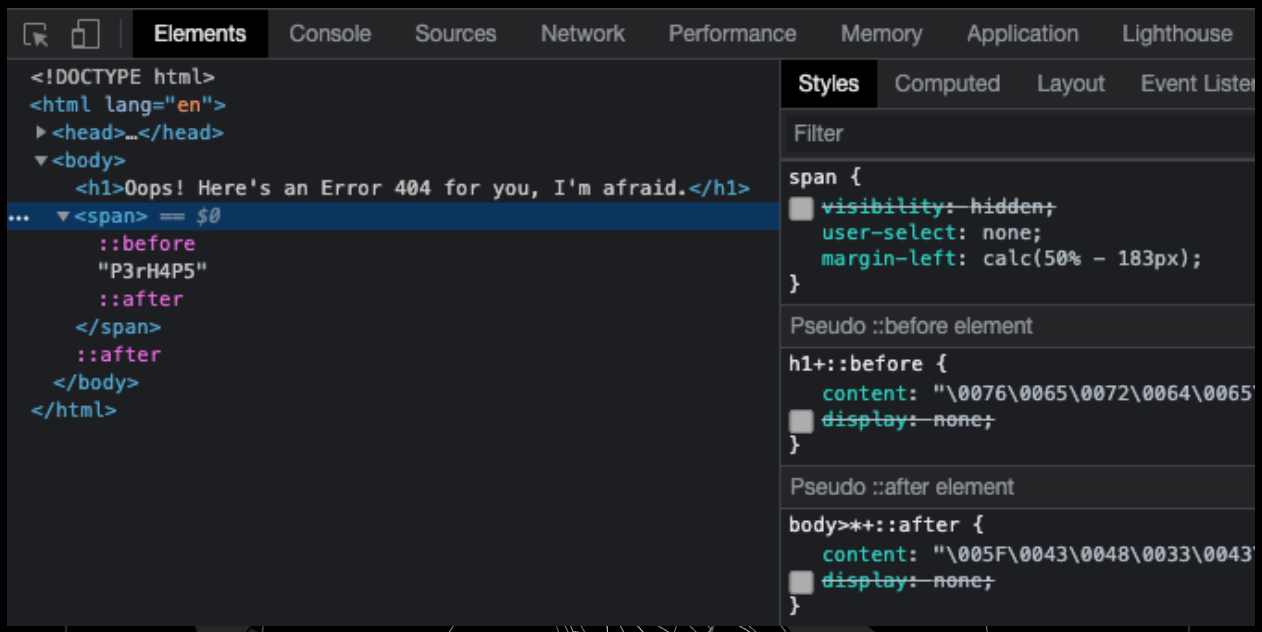
Decode it and the flag is captured.

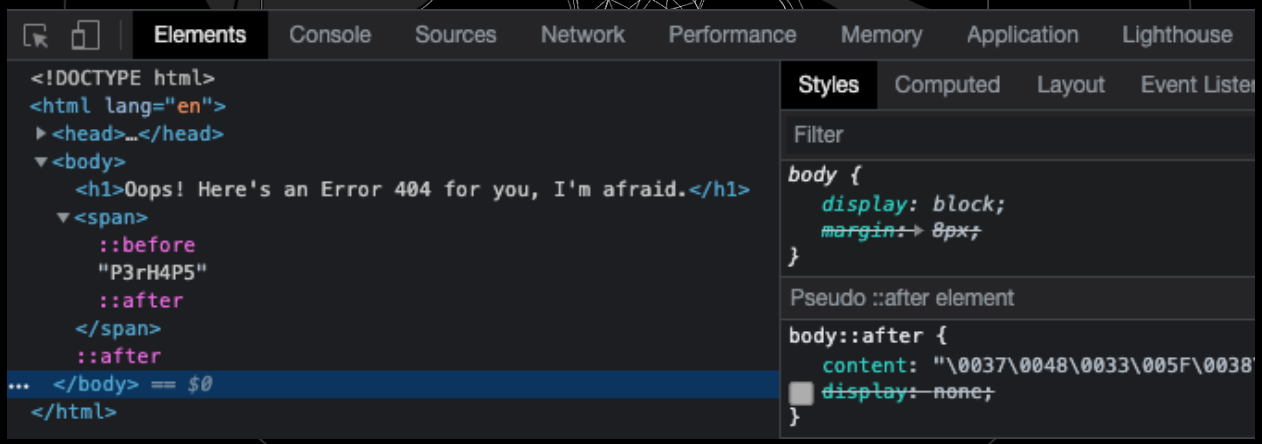
```
└─$ echo dmVyZGVzZWN7ODQ1MzY0XzNOQzBEM0RfMU5fRDQ3NF84MFU3X0Q0NzR9 | base64 -d
verdesec{845364_3NC0D3D_1N_D474_80U7_D474}
```

Flag: verdesec{845364_3NC0D3D_1N_D474_80U7_D474}

Flag #14

Disable all “visibility: hidden;” and “display: none;” in the CSS of and <body> HTML elements to get the flag.

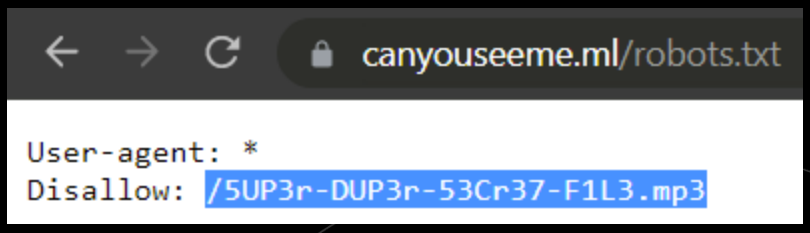




Flag: verdesec{P3rH4P5_CH3CK_0U7_7H3_84CK6r0UND}

Flag #15

Visit /robots.txt to get a clue on which file to look for.

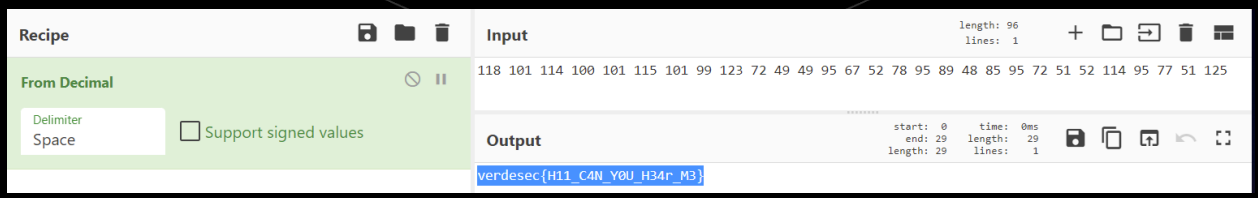


Download and play /5UP3r-DUP3r-53Cr37-F1L3.mp3. Apart from the noise, some dial tones can be heard. DTMF should be popping into your head.

Decode it and you will get a bunch of numbers, separated by intervals.

118 101 114 100 101 115 101 99 123 72 49 49 95 67 52 78 95 89 48 85 95 72 51 52 114 95 77 51 125

Paste them in CyberChef and you've got the flag.



Flag: verdesec{H11_C4N_Y0U_H34r_M3}

Flag #16

This flag is hidden in /linux at the CTF Challenge Portal.

Download the file with “wget” or other similar tools. The following example uses the Linux command “wget” to download the file.

```
wget https://canyouseeme.ml/linux
```

After downloading, load it into Hex-Rays’ IDA (Freeware, Version 7.6 is used here). Check out the main function.

```
.text:00000000004A2169 loc_4A2169:                             ; CODE XREF: main_main+1E6↑j
.text:00000000004A2169                 movups  [rsp+178h+var_D8], xmm15
.text:00000000004A2172                 lea     rdx, unk_4AB240
.text:00000000004A2179                 mov     qword ptr [rsp+178h+var_D8], rdx
.text:00000000004A2181                 lea     r8, off_4DE610  ; "\nHere's for you: "
.text:00000000004A2188                 mov     qword ptr [rsp+178h+var_D8+8], r8
.text:00000000004A2190                 mov     rbx, cs:os_Stdout
.text:00000000004A2197                 lea     rax, go_itab__os_File_io_Writer
.text:00000000004A219E                 lea     rcx, [rsp+178h+var_D8]
.text:00000000004A21A6                 mov     edi, 1
.text:00000000004A21AB                 mov     rsi, rdi
.text:00000000004A21AE                 call    fmt_Fprint
```

Under “loc_4A4169”, notice that the string “\nHere’s for you: ” is printed. With this, we can safely assume what’s going to be printed next would be the flag. So, continue the inspection.

```
.text:00000000004A21AE                 call    fmt_Fprint
.text:00000000004A21B3                 lea     rdx, [rsp+178h+var_107]
.text:00000000004A21B8                 movups  xmmword ptr [rdx], xmm15
.text:00000000004A21BC                 lea     rdx, [rsp+178h+var_F8]
.text:00000000004A21C4                 movups  xmmword ptr [rdx], xmm15
.text:00000000004A21C8                 mov     rdx, 7E717E6D7966687Eh
.text:00000000004A21D2                 mov     [rsp+178h+var_107], rdx
.text:00000000004A21D7                 mov     rdx, 2922524C2D243B73h
.text:00000000004A21E1                 mov     [rsp+79h], rdx
.text:00000000004A21E6                 mov     rdx, 4258413B4A4B3E45h
.text:00000000004A21F0                 mov     [rsp+178h+var_F7], rdx
.text:00000000004A21F8                 mov     [rsp+178h+var_EF], 44204158h
.text:00000000004A2203                 mov     [rsp+178h+var_EB], 493Bh
.text:00000000004A220D                 mov     [rsp+178h+var_E9], 69h ; 'i'
.text:00000000004A2215                 mov     rdx, [rsp+178h+var_E8]
.text:00000000004A221D                 mov     r8, [rsp+178h+var_130]
.text:00000000004A2222                 xor     eax, eax
.text:00000000004A2224                 jmp     loc_4A22AB
```

After calling the print function, three quadwords (“7E717E6D7966687Eh”, “2922524C2D243B73h” and “4258413B4A4B3E45h”), one doubleword (“44204158h”), one word (“493Bh”) and one byte (“69h”) are stored sequentially in the stack. Trying to view them in ASCII shows scrambled text, so, clearly, some processing needs to be done.

After copying "var_E8" to "rdx", copying "var_130" to "r8" and resetting "eax" to 0, jump to "loc_4A22AB".

```
.text:00000000004A22AB loc_4A22AB:                    ; CODE XREF: main_main+524↑j
.text:00000000004A22AB                cmp     rax, 1Fh
.text:00000000004A22AF                jge     short loc_4A22D8
.text:00000000004A22B1                movzx   esi, byte ptr [rsp+rax+178h+var_107]
.text:00000000004A22B6                test    r8, r8
.text:00000000004A22B9                jz      loc_4A2369
.text:00000000004A22BF                mov     rcx, rax
.text:00000000004A22C2                mov     rbx, rdx
.text:00000000004A22C5                cqo
.text:00000000004A22C7                idiv    r8
.text:00000000004A22CA                cmp     rdx, r8
.text:00000000004A22CD                jb      loc_4A2229
.text:00000000004A22D3                jmp     loc_4A235E
```

From here, we know that "var_107" (recall that it is where the first quadword is stored) with "rax" acting as the index starting from 0 is loaded into "esi". Note that the content of "rax" (the index for "var_107") is copied to "rcx" and that of "rdx" ("var_E8") is copied to "rbx". After performing "cqo" that converts the quadword in "rax" to an octword in "rdx:rax", "idiv r8" is performed, where "rdx:rax" (now storing the index for "var_107") is divided by "r8" ("var_130") with the quotient stored to "rax" and the remainder stored to "rdx". After the division is performed successfully, jump to "loc_4A2229".

```
.text:00000000004A2229 loc_4A2229:                    ; CODE XREF: main_main+5CD↓j
.text:00000000004A2229                mov     [rsp+178h+var_110], rcx
.text:00000000004A222E                movzx   ecx, byte ptr [rbx+rdx]
.text:00000000004A2232                xor     esi, ecx
.text:00000000004A2234                movzx   ecx, sil
.text:00000000004A2238                xor     eax, eax
.text:00000000004A223A                mov     rbx, rcx
.text:00000000004A223D                nop     dword ptr [rax]
.text:00000000004A2240                call    runtime_intstring
.text:00000000004A2245                call    runtime_convTstring
.text:00000000004A224A                movups  [rsp+178h+var_58], xmm15
.text:00000000004A2253                lea     rcx, unk_4AB240
.text:00000000004A225A                mov     qword ptr [rsp+178h+var_58], rcx
.text:00000000004A2262                mov     qword ptr [rsp+178h+var_58+8], rax
.text:00000000004A226A                mov     rbx, cs:os_Stdout
.text:00000000004A2271                lea     rax, go_itab__os_File_io_Writer
.text:00000000004A2278                mov     edi, 1
.text:00000000004A227D                mov     rsi, rdi
.text:00000000004A2280                lea     rcx, [rsp+178h+var_58]
.text:00000000004A2288                call    fmt_Fprint
```

This section reveals the processing that should be done to the scrambled text starting at "var_107". From what we have discovered earlier, "movzx ecx, byte ptr [rbx+rdx]" is actually something like "ecx = var_E8[(index for var_107) % var_130]". Continue looking at "xor esi, ecx", we now know that "var_107" should be XOR-ed with "ecx", i.e., "flag = var_107 ^ var_E8[(index for var_107) % var_130]".

At this point, we can already safely guess that “var_130” is set to the length of “var_E8”. After some searching, we can spot the usage of “var_130” and “var_E8” under “loc_4A1ED5”.

```
.text:00000000004A206D                  mov     rdx, [rsp+178h+var_E8]
.text:00000000004A2075                  jmp     short loc_4A20A4
```

After “var_E8” is copied to “rdx”, jump to “loc_4A20A4”.

```
.text:00000000004A20A4 loc_4A20A4:                      ; CODE XREF: main_main+375↑j
.text:00000000004A20A4                  mov     [rdx+rcx], al
.text:00000000004A20A7                  jmp     short loc_4A20ED
```

Here, the value of “rdx” (“var_E8”) plus “rcx” (probably acting as the index for “var_E8”) is set to “al”. Now we need to find out what will be stored inside “al”. Go back until before jumping to “loc_4A20A4”.

```
.text:00000000004A204C                  mov     rdx, [rsp+178h+var_120]
.text:00000000004A2051                  add     rdx, 4
.text:00000000004A2055                  cmp     rax, rdx
.text:00000000004A2058                  jnz     short loc_4A20A9
.text:00000000004A205A                  mov     rcx, [rsp+178h+var_130]
.text:00000000004A205F                  lea     rsi, [rcx+1]
.text:00000000004A2063                  mov     rdi, [rsp+178h+var_128]
.text:00000000004A2068                  cmp     rdi, rsi
.text:00000000004A206B                  jb      short loc_4A2077
.text:00000000004A206D                  mov     rdx, [rsp+178h+var_E8]
.text:00000000004A2075                  jmp     short loc_4A20A4
```

A few notable instructions are spotted here. First, “var_120” is copied to “rdx”. Second, “rdx” is added with “4”. Third, “rdx” should now be equal to “rax” in order to skip “jnz short loc_4A20A9”. Now we know that “al” (the least significant byte of “rax”) should be set to “var_120 + 4”. Fourth, “var_130” (possibly the length of “var_E8”) is copied to “rcx” (possibly acting as the index for “var_E8”). Fifth, “rsi” is set to “rcx” plus “1”. So, “rsi = var_130 + 1”.

Go to the top of “loc_4A1ED5” to find what value is held by “var_120”.

```
.text:00000000004A1ED5 loc_4A1ED5:                      ; CODE XREF: main_main+11A↑j
.text:00000000004A1ED5                  mov     [rsp+178h+var_130], rcx
.text:00000000004A1EDA                  mov     [rsp+178h+var_E8], rbx
.text:00000000004A1EE2                  cmp     rax, 5
.text:00000000004A1EE6                  jg      loc_4A2169
.text:00000000004A1EEC                  mov     [rsp+178h+var_118], rax
.text:00000000004A1EF1                  mov     [rsp+178h+var_128], rdx
.text:00000000004A1EF6                  imul    rax, rax
.text:00000000004A1EFA                  mov     [rsp+178h+var_120], rax
.text:00000000004A1EFF                  nop
```

After multiplying “rax” by itself, it is stored into “var_120”, so “var_120 = rax * rax”. Before this, note that “rax” should be less than or equal to “5”, and that “rax” is copied to “var_118”.

Now, we look at the assembly instructions before jumping to "loc_4A1ED5" in order to find out what value does "rax" start with.

```
.text:00000000004A1E0F          mov     eax, 2
.text:00000000004A1E14          xor     ecx, ecx
.text:00000000004A1E16          xor     edx, edx
.text:00000000004A1E18          xor     ebx, ebx
.text:00000000004A1E1A          jmp     loc_4A1ED5
```

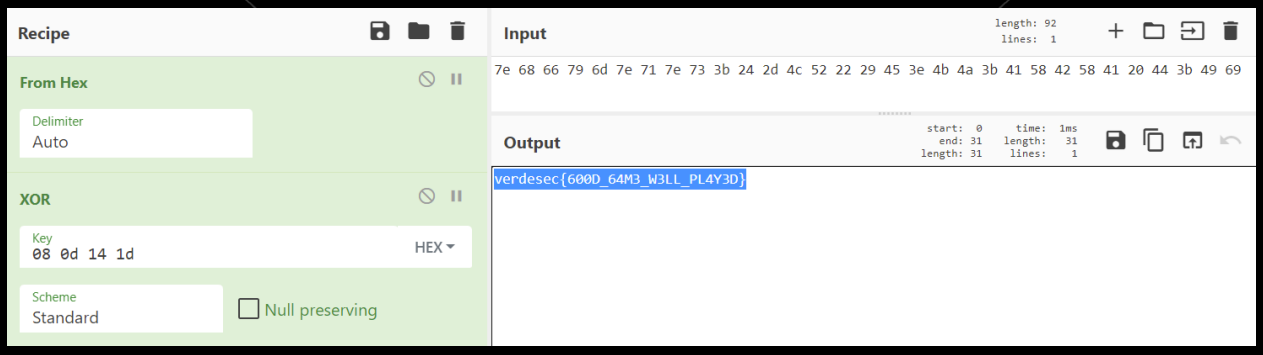
From here, we can observe that "eax" (the lower half of "rax") is set to begin with "2". At this point, we can deduce that the loop starts with "rax" being "2" and ends with "rax" being "5".

Head over to "loc_4A1E1F" to see what is run at the end of each loop.

```
.text:00000000004A1E1F loc_4A1E1F:                     ; CODE XREF: main_main+404↓j
.text:00000000004A1E1F          mov     [rsp+178h+var_128], rdi
.text:00000000004A1E24          mov     [rsp+178h+var_130], rsi
.text:00000000004A1E29          mov     [rsp+178h+var_E8], rdx

.text:00000000004A1EBA          mov     rdx, [rsp+178h+var_118]
.text:00000000004A1EBF          lea     rax, [rdx+1]
.text:00000000004A1EC3          mov     rcx, [rsp+178h+var_130]
.text:00000000004A1EC8          mov     rdx, [rsp+178h+var_128]
.text:00000000004A1ECD          mov     rbx, [rsp+178h+var_E8]
```

"rsi" that was set to "var_130 + 1" earlier is now copied to "var_130". This means that "var_130" increments by 1, which perfectly matches our guess that it is the length for "var_E8". At the same time, we can also confirm that the "rcx" mentioned earlier is in fact the index for "var_E8" as it is the value of "var_130" before this increment. Next, we can see that "var_118" (with its value copied from "rax" earlier) is copied to "rdx", and "rax" is updated with "rdx" plus "1". So, "rax = var_118 + 1". This helps us confirm that "rax" is incremented by 1 in each iteration.

With "key[j] = i * i + 4" where "2 <= i <= 5" and "0 <= j <= 3", the key would be "8", "13", "20" and "29". Note that the scrambled text is in little-endian format. Rearrange them before XOR-ing at CyberChef.



Flag: verdesec{600D_64M3_W3LL_PL4Y3D}

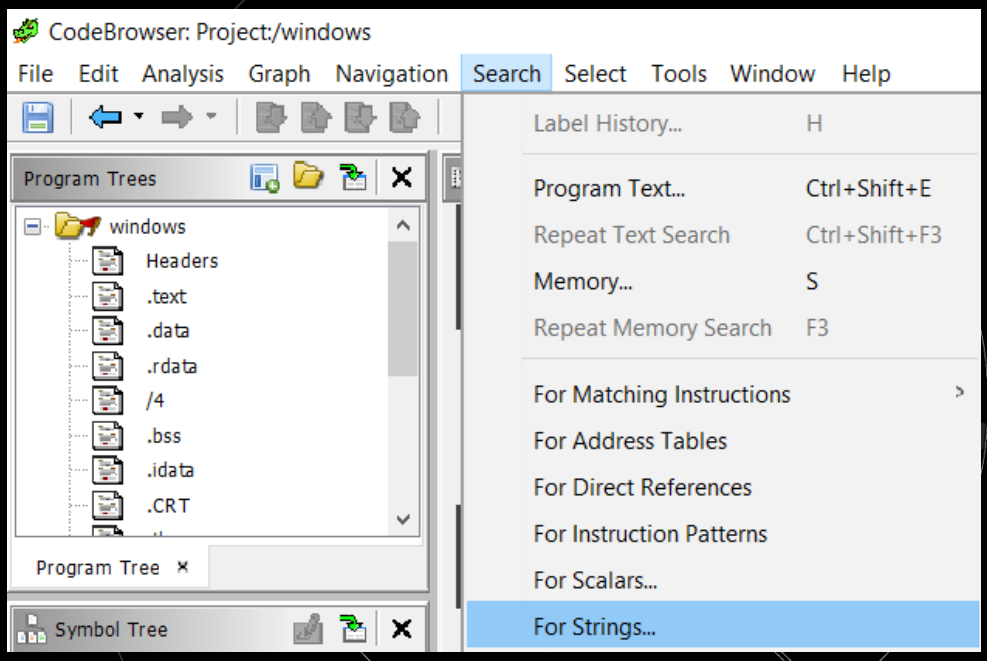
Flag #17

This flag can be found in /windows at the CTF Challenge Portal using the exactly same method described in Flag #11.

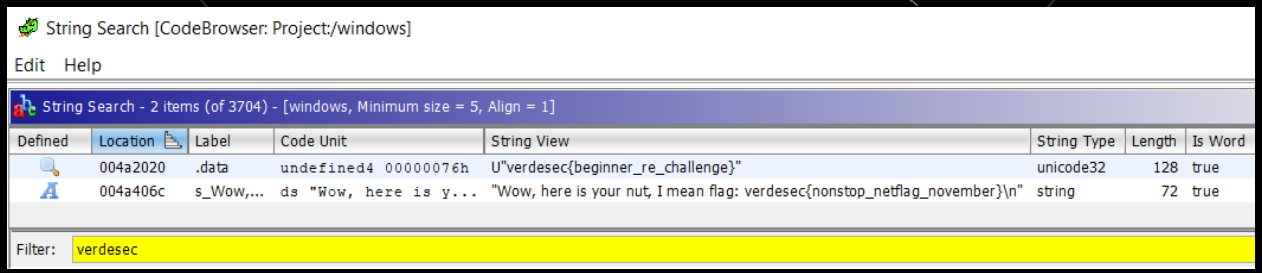
Download the file with “wget” or other similar tools. The following example uses the Linux command “wget” to download the file.

```
wget https://canyouseeme.ml/windows
```

After downloading, load it into Ghidra (Version 10.0.4 is used here), and search for strings.



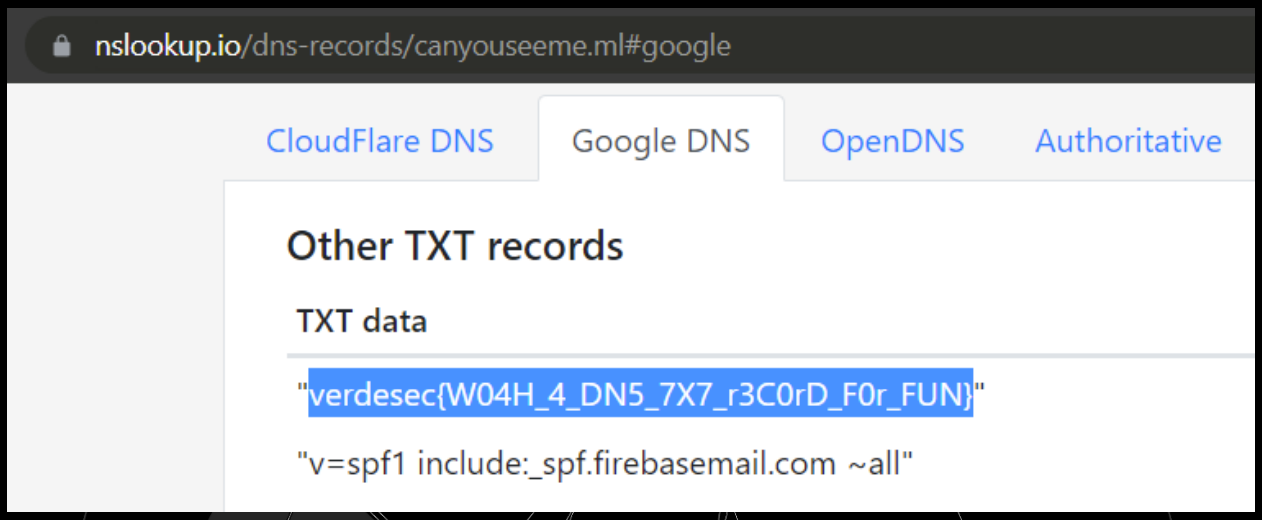
Filter the result using any appropriate keyword, e.g., “verdesec”, and you’ve got the flag.



Flag: verdesec{nonstop_netflag_november}

Flag #18

Look up the DNS records of the CTF Challenge Portal and you will find it in one of the TXT records.

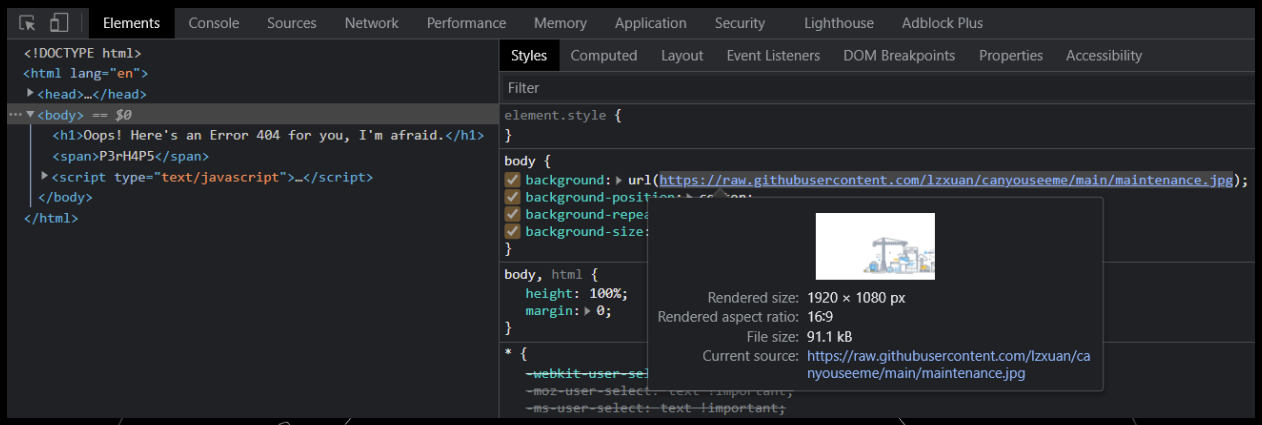



```
PS C:\Users\XUAN> nslookup -type=txt canyouseeme.ml
Server:  one.one.one.one
Address:  1.1.1.1

Non-authoritative answer:
canyouseeme.ml  text =

        "verdesec{W04H_4_DN5_7X7_r3C0rD_F0r_FUN}"
```

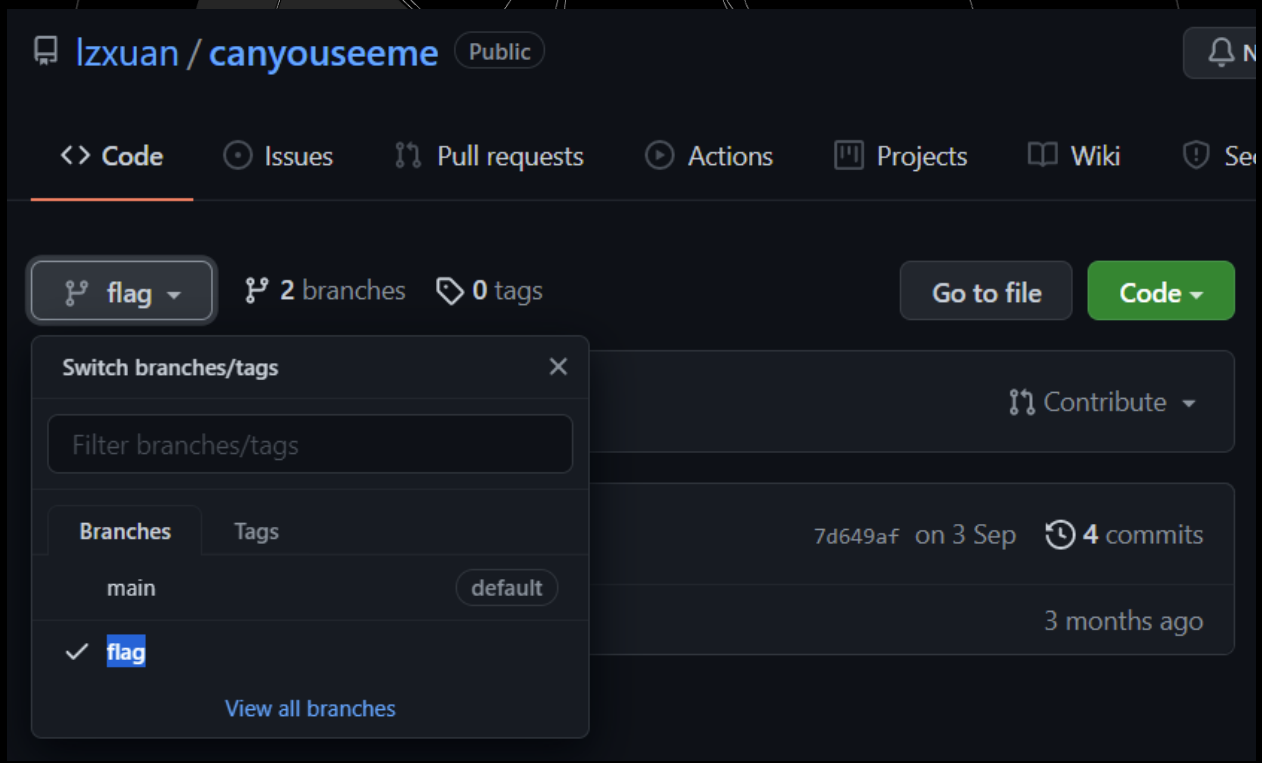
Flag: verdesec{W04H_4_DN5_7X7_r3C0rD_F0r_FUN}

Flag #19

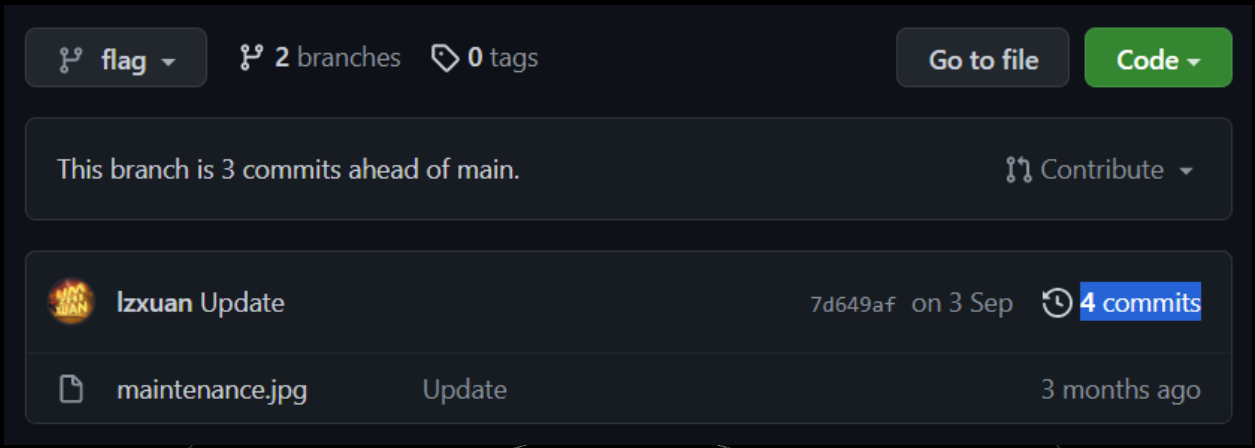
Visit any non-existent web page at the CTF Challenge Portal and you will find the background image to be coming from a GitHub repository.



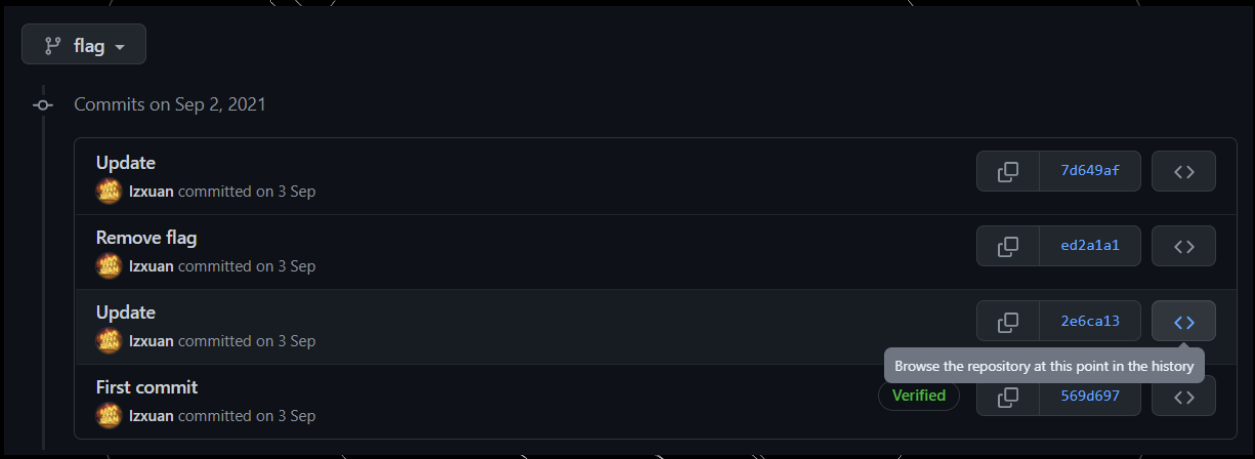
Go to the GitHub repository https://github.com/lzxuan/canyouseeme. While the main branch looks normal, take a look at the branch “flag”.



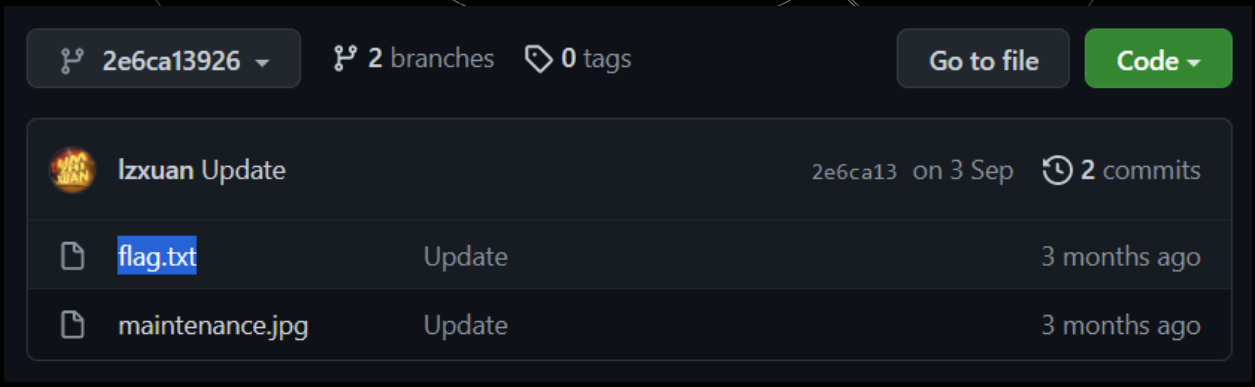
Notice there are 4 commits, check them out.



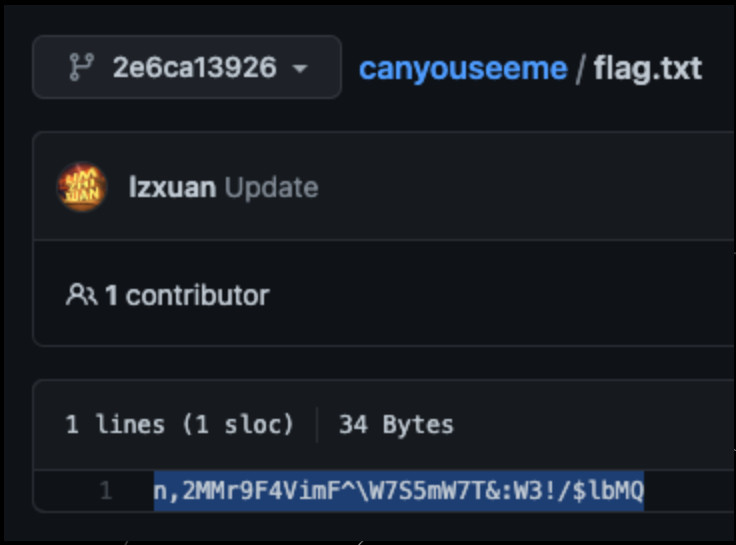
A commit is spotted with the message of “Remove flag”, so browse the files at the one before it (2e6ca13926) to look for the removed flag.



flag.txt is found, open it.

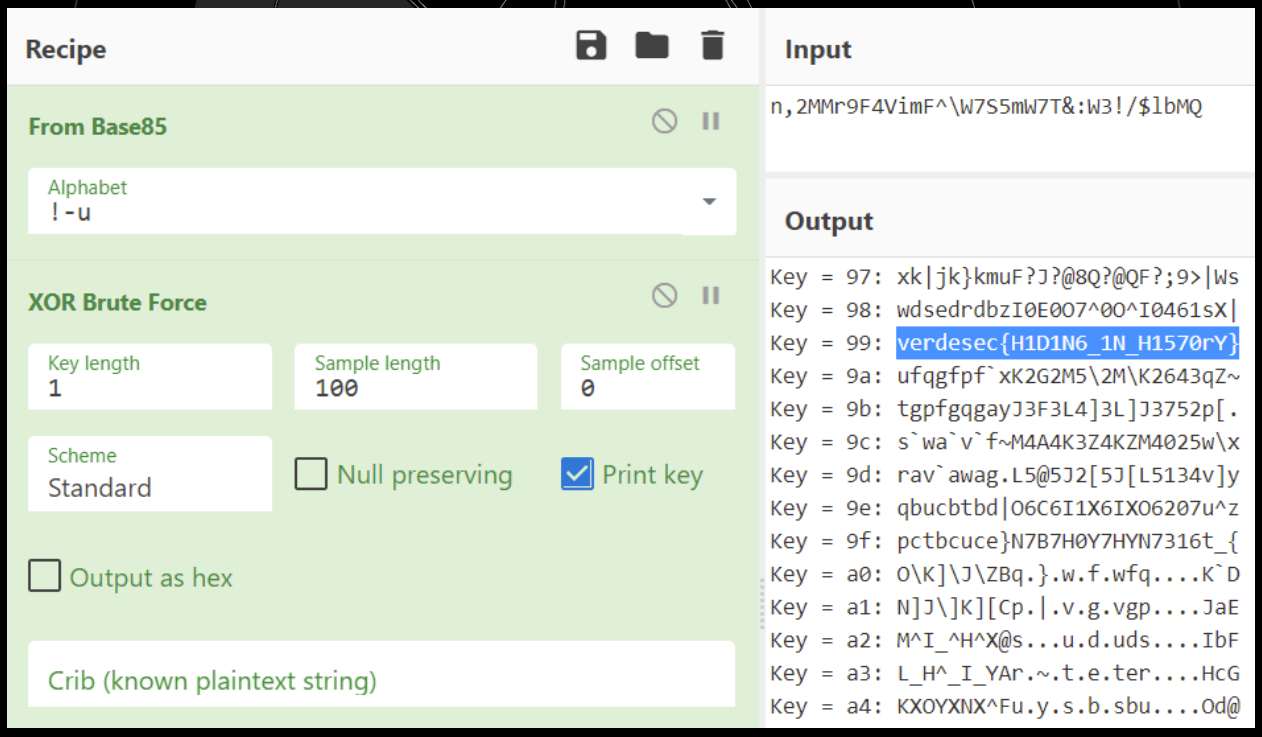


flag.txt contains a seemingly random string.



n,2MMr9F4VimF^\W7S5mW7T&:W3!/$lbMQ

Decode it with the following recipe at CyberChef to get the flag.



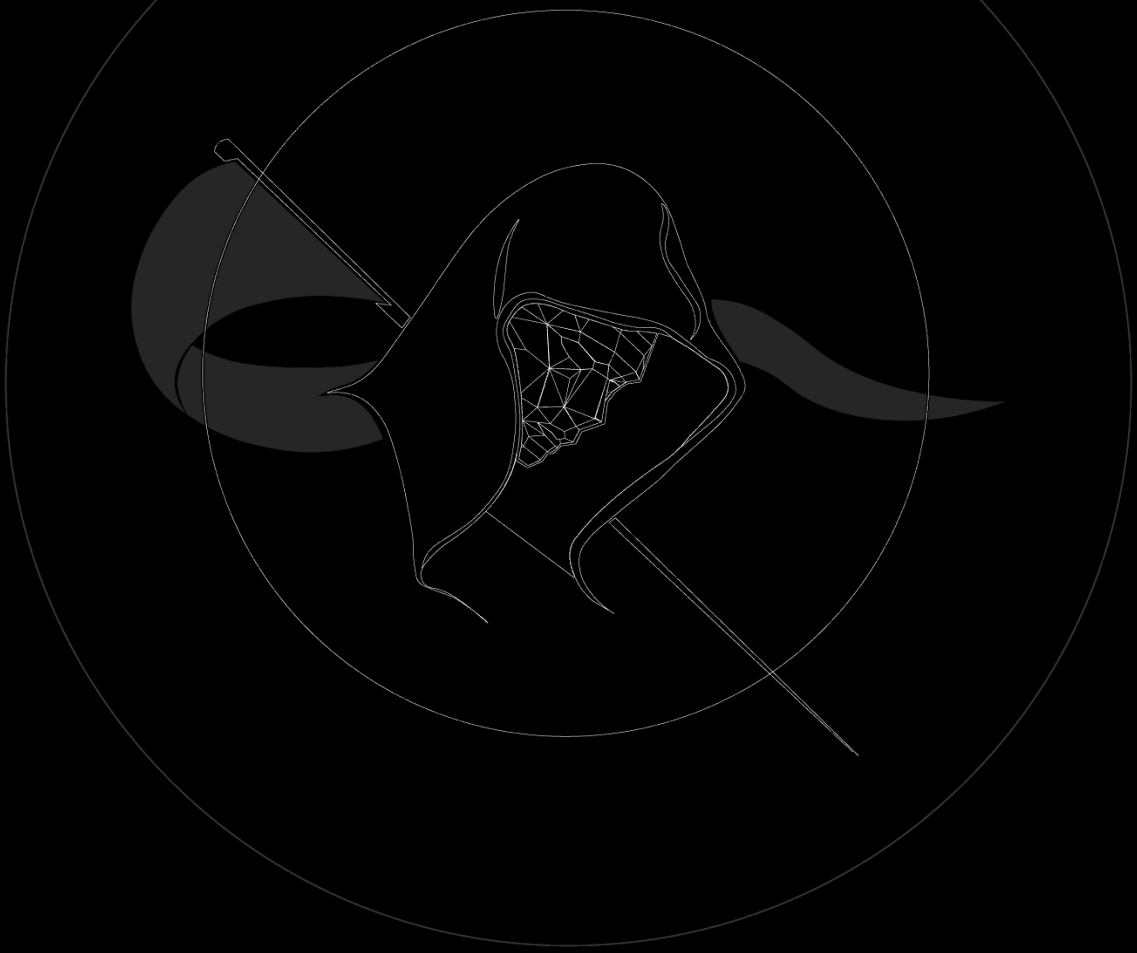
Flag: verdesec{H1D1N6_1N_H1570rY}

Flag #20

strings the file /favicon-16x16.png at the CTF Challenge Portal and (optionally) grep the flag format to get this flag.

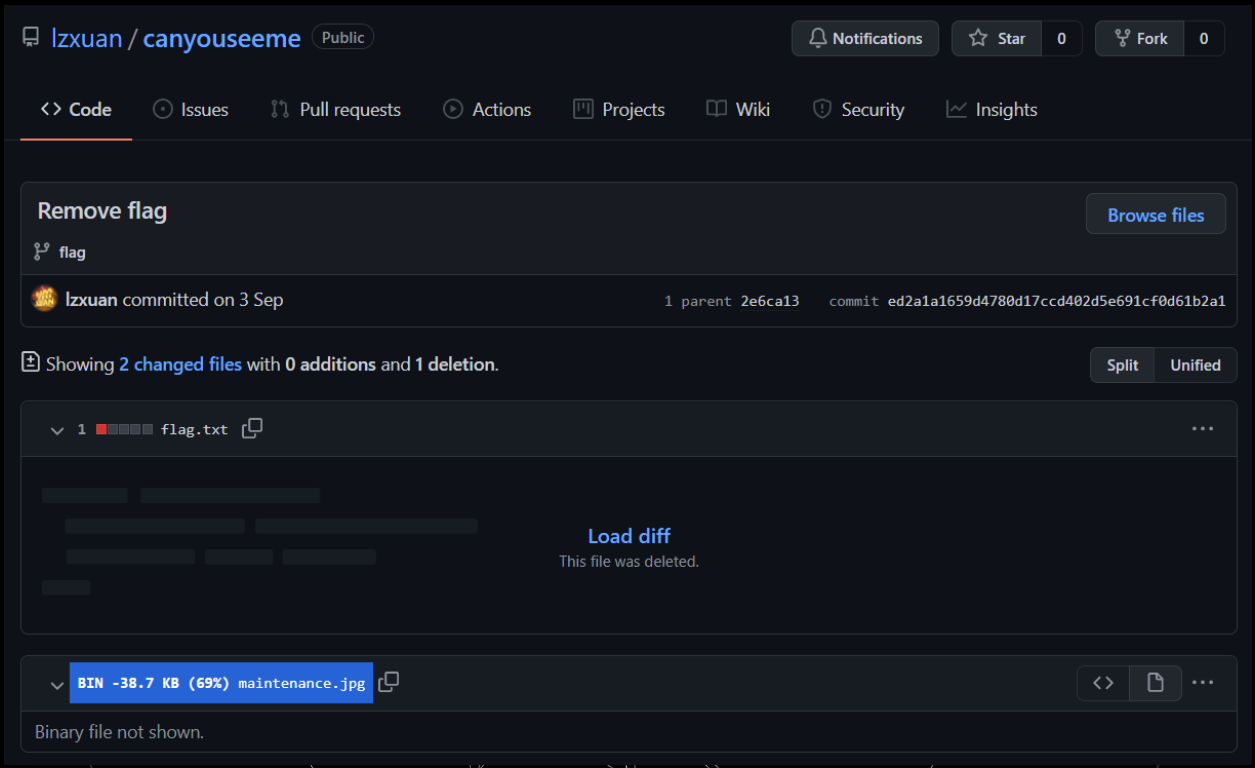
```
└─$ strings favicon-16x16.png | grep verdesec
verdesec{L00K_0U7_7r41L1N6_D474_4F73r_13ND}
```

Flag: verdesec{L00K_0U7_7r41L1N6_D474_4F73r_13ND}

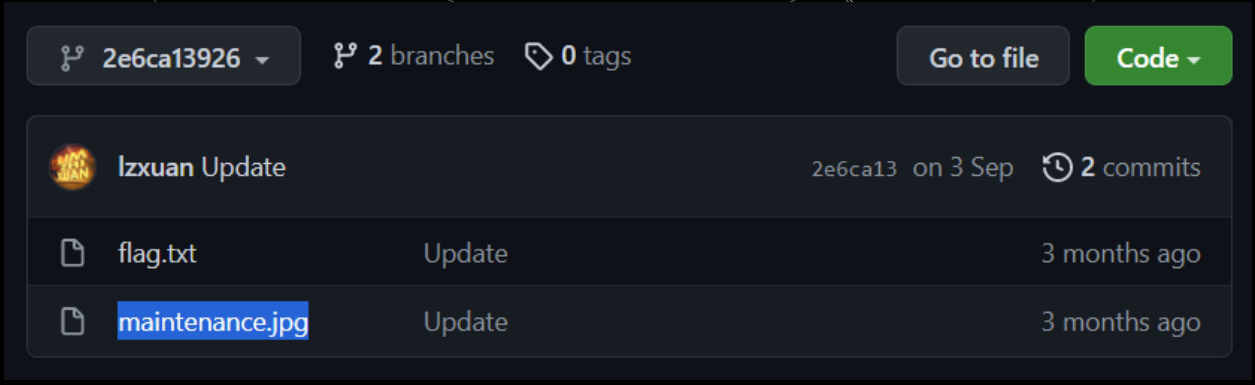


Flag #21

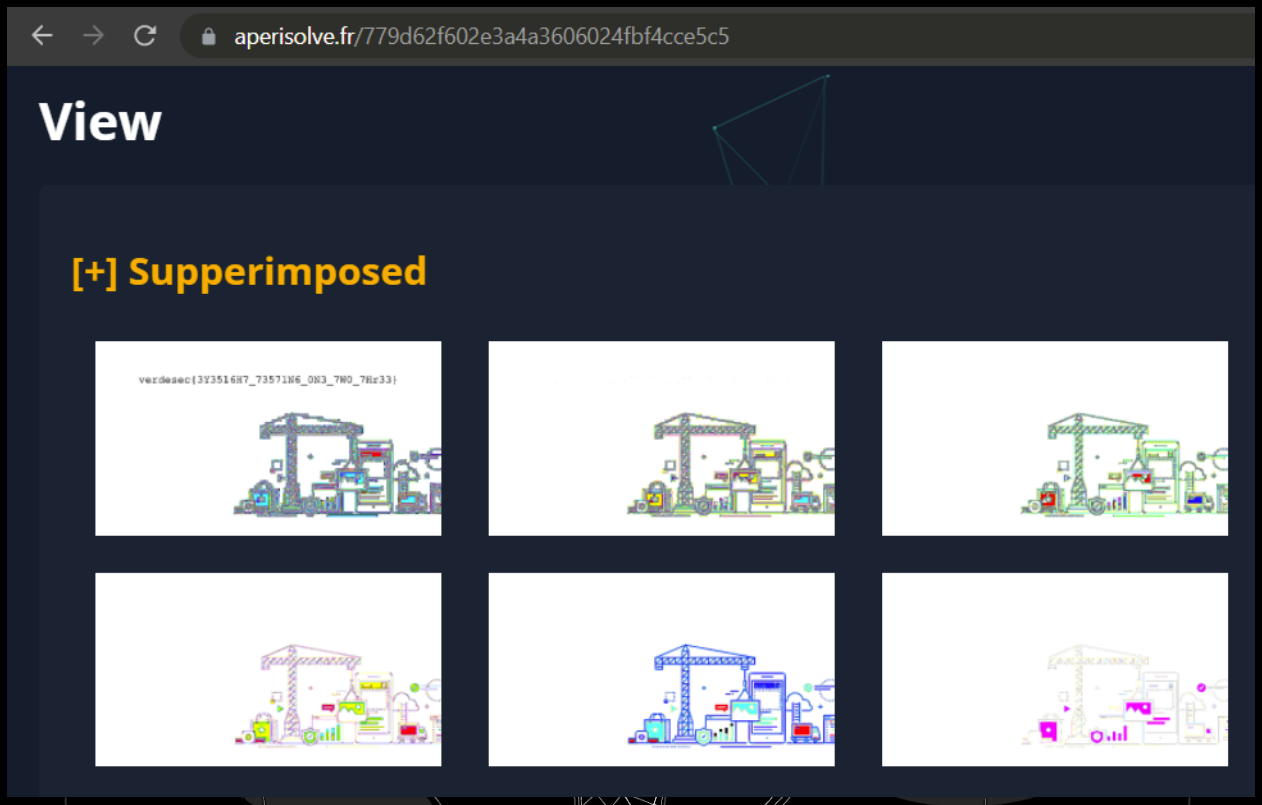
Following the first few steps at Flag #19, if you click the commit with the message of "Remove flag" to see its details, you should be able to notice that maintenance.jpg is also modified (apart from flag.txt being removed).



Similarly, browse the files at the commit before it (2e6ca13926) to download the original maintenance.jpg.



Upload it to https://aperisolve.fr/ or open it using StegSolve and search for the view that spells out the flag.



Flag: verdesec{3Y3516H7_73571N6_0N3_7W0_7Hr33}

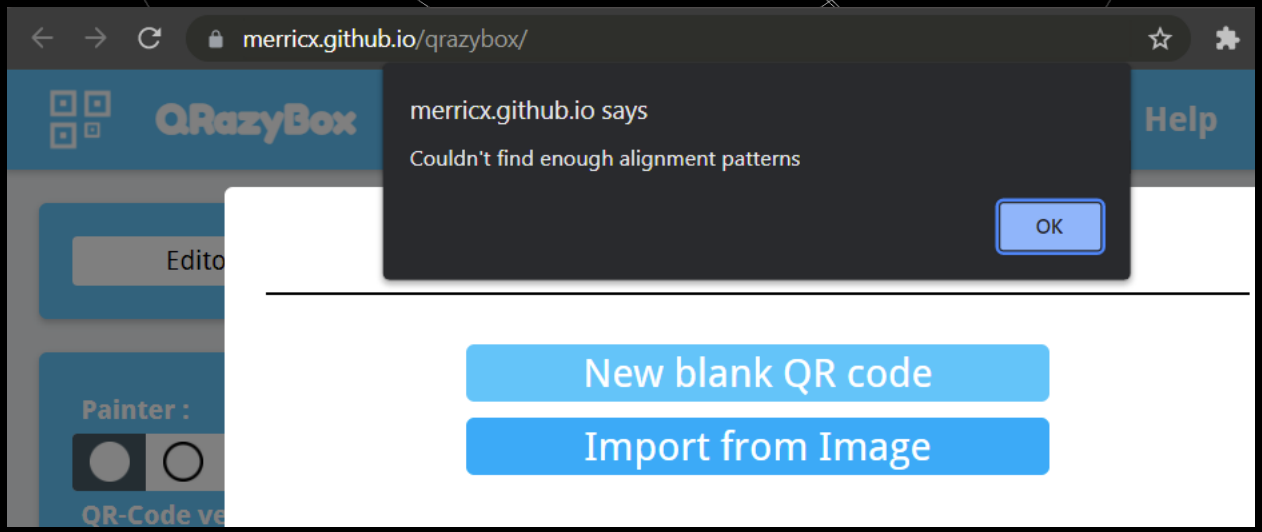
Flag #22

Download /android-chrome-192x192.png at the CTF Challenge Portal and inspect it with "binwalk". Notice that within the file, there is another PNG image with the dimension of 200x200 starting from 0x2E11. Extract it by using the option "--dd='.*'".

```
└─$ binwalk --dd='.*' android-chrome-192x192.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 192 x 192, 8-bit/color RGBA, non-interlaced
11793         0x2E11          PNG image, 200 x 200, 8-bit/color RGBA, non-interlaced
11887         0x2E6F          Zlib compressed data, compressed
```

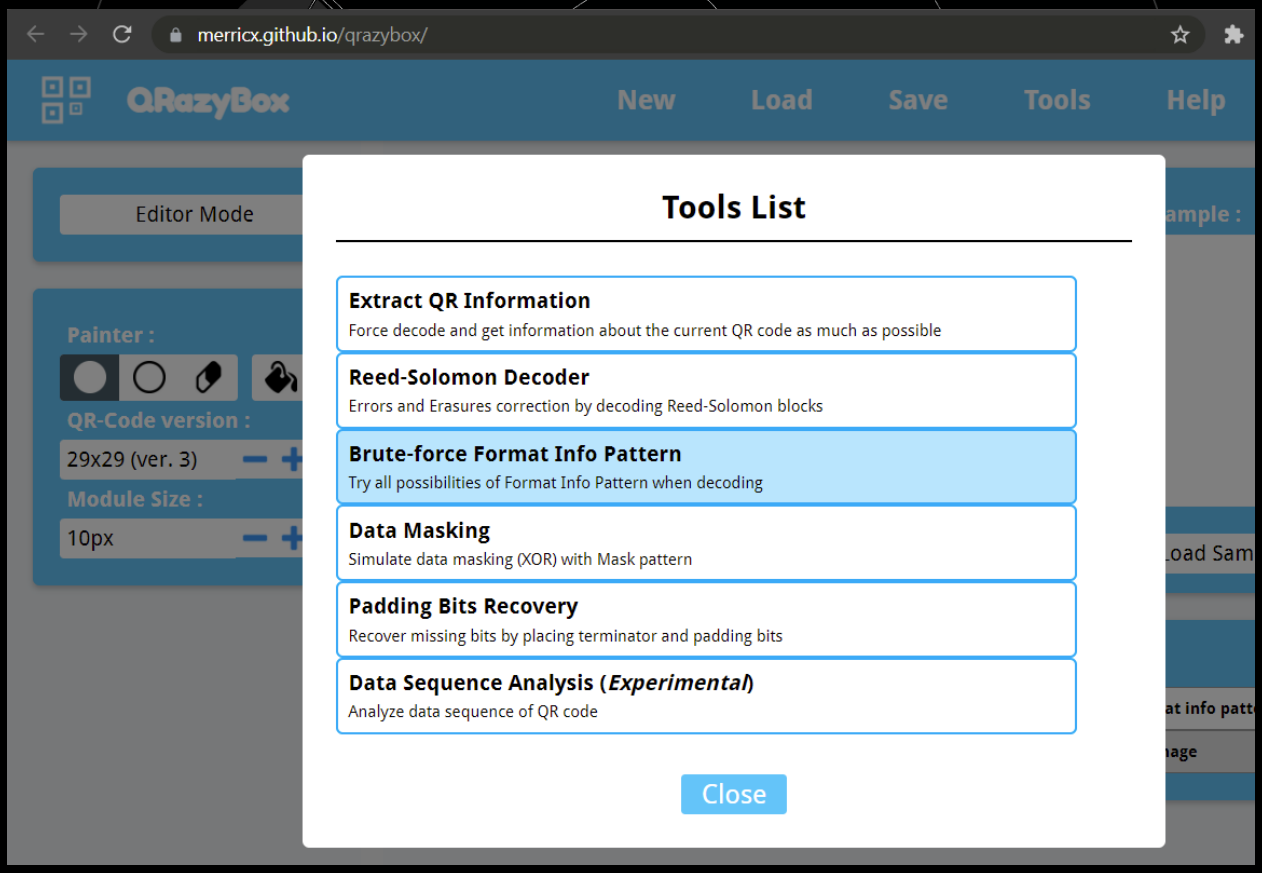
Open the extracted image "2E11" (append ".png" to the filename if necessary) and you will realize that it is an incomplete QR code.

Trying to recover it by uploading it to https://merricx.github.io/qrazybox/ yields the error below.

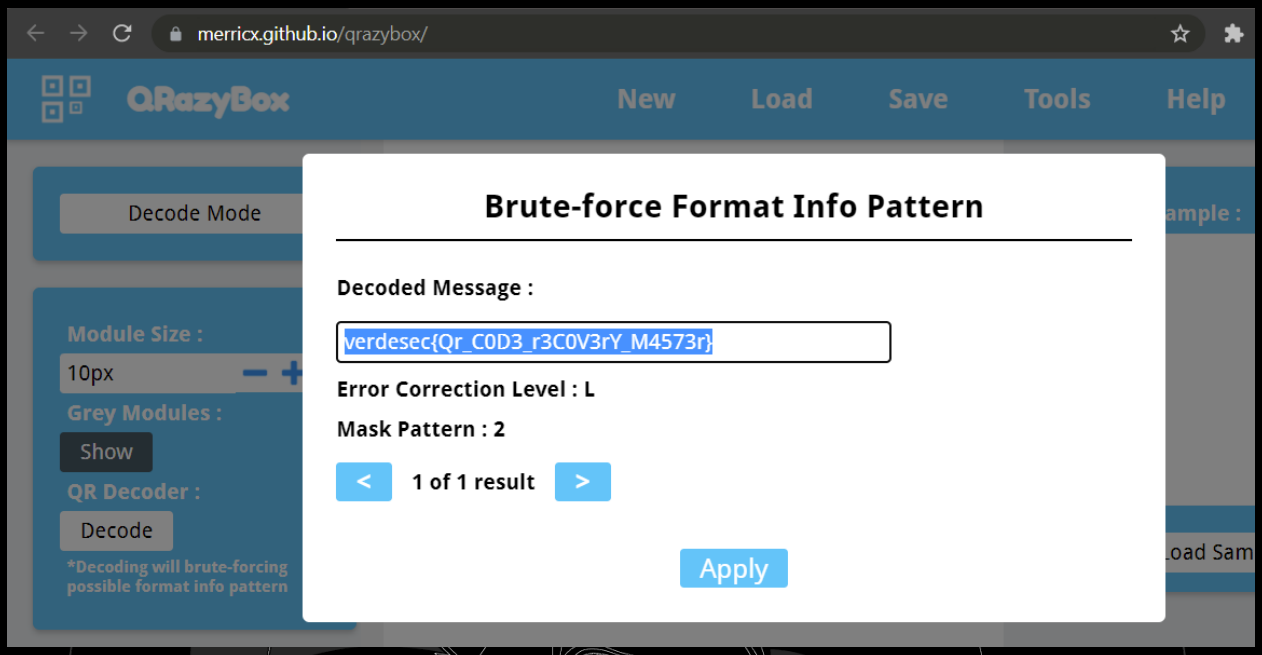


Add the missing alignment pattern (the green dot) to the QR code.

It can now be loaded successfully. Next, select the option “Brute-force Format Info Pattern” from the tab “Tools” and then close the popup. After that, click the button “Editor Mode” at the top left corner to switch to the “Decode Mode”.



In the “Decode Mode”, click the button “Decode” under the label “QR Decoder” to start the recovery process. The decoded message appears almost instantly, and it turns out to be the flag.

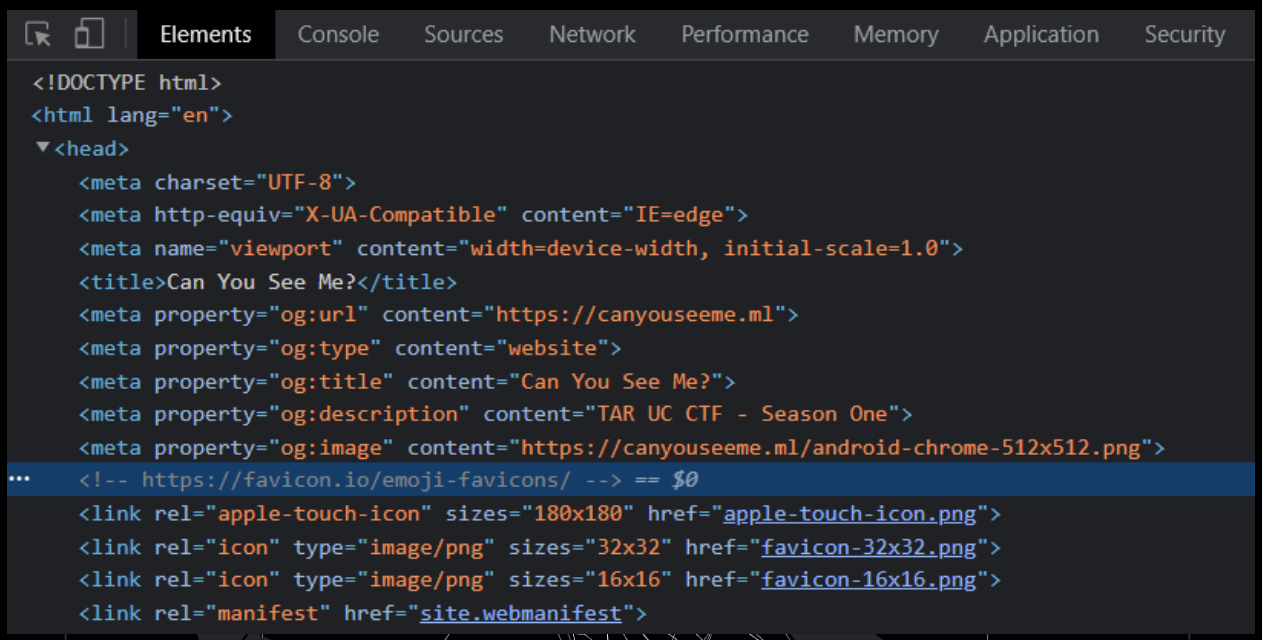


The complete QR code is shown below for reference.

Flag: verdesec{Qr_C0D3_r3C0V3rY_M4573r}

Flag #23

The HTML comment “https://favicon.io/emoji-favicons/” at the CTF Challenge Portal landing page hints at the origin of the favicons.



A closer look at the favicons’ origin website uncovers favicon.ico, which is not included in the HTML source code.

Download /favicon.ico and open it to look for the flag residing at the layer with the resolution of 96 x 96.

Flag: verdesec{1M463_H1D1N6_1N_F4V1C0N1C0}