

So for this challenge we are given a Haskell executable called "content".

The rationale behind this challenge was to prove that a novice hacker (the friend in the story) can't hide his secret texts within an executable using simple cryptographic functions (such as ROT, Caesar shift on individual bytes, base64 and gzipping) and easy to decode executables.

The executable given has the following source code:

```
programming_chall > x content.hs
1  {-# LANGUAGE OverloadedStrings #-}
2  -- | A deliberately broken Base64+GZip decoder with embedded data
3  module Main where
4
5  import qualified Data.ByteString      as BS      -- strict
6  import qualified Data.ByteString.Char8 as BC      -- strict Char8
7  import qualified Data.ByteString.Lazy as BL      -- lazy for gzip
8  import      Codec.Compression.GZip (decompress)
9  import      System.Exit           (exitFailure)
10
11 -- | Top-level embedding of the Base64-encoded, gzipped blob
12 embedded :: BC.ByteString
13 embedded = BC.pack
14   |> "H4sIAHmhFmgA/+07bYwb1bwzmyYsEMgGEIcSIC4lT1A39treD28gEK/X6/Gu7fWu7U121XY22N7dm2Pd2a8u7YqGpS+iiVJlfenVHrvR1/V0khFr/lRpCA"
15
16 -- | A very simple (but buggy) Base64 decoder.
17 brokenBase64Decode :: BC.ByteString -> Maybe BS.ByteString
18 brokenBase64Decode bs = fmap BS.pack . sequence $ map decodeVal (BC.unpack bs)
19   where
20     decodeVal c
21       | 'A' <= c && c <= 'Z' = Just (fromIntegral $ fromEnum c - fromEnum 'A')
22       | 'a' <= c && c <= 'z' = Just (fromIntegral $ 26 + fromEnum c - fromEnum 'a')
23       | '0' <= c && c <= '9' = Just (fromIntegral $ 52 + fromEnum c - fromEnum '0')
24       | c == '+'              = Just 63 -- BUG: should be 62
25       | c == '/'              = Just 62 -- BUG: should be 63
26       | c == '='              = Just 0  -- BUG: padding handling is wrong
27       | otherwise             = Nothing
28
29 main :: IO ()
30 main = case brokenBase64Decode embedded of
31   Nothing    -> putStrLn "[Error] Base64 decoding failed." >> exitFailure
32   Just gzipped -> decompressBody gzipped >= BS.putStr
33
34 -- | Wrap GZip decompression by converting between strict and lazy
35 decompressBody :: BS.ByteString -> IO BS.ByteString
36 decompressBody bs =
37   -- convert strict -> lazy, decompress, back to strict
38   return $ BL.toStrict (decompress (BL.fromStrict bs))
39
```

It tries to do the actions the user will have to do (base64 and gunzipping), but the code is intentionally written wrong.

When the competitor tries to execute the Haskell binary, they get the following error message:

```
content: Codec.Compression.Zlib: compressed data stream format error (incorrect header check)
```

At the memory location 0x691540 we can find those Main main and what seems like a base64 string:

```

00691490 D1 49 D3 E0 44 89 D9 48 09 C7 4C 89 C8 48 D3 EB NIOaD%UH.ÇL%EHoe
006914A0 44 89 D1 48 D3 E6 44 89 D9 48 89 DA 48 D3 E8 44 D%ÑHÖæD%ÜH%ÜHÖæD
006914B0 89 D1 48 09 F0 49 D3 E1 48 F7 F7 48 89 D6 49 F7 %ÑH.8iÖáH÷÷H%Öi÷
006914C0 E0 48 89 C3 48 89 D1 48 39 D6 72 07 75 11 49 39 àH%ÀH%ÑH9Ör.u.I9
006914D0 C1 73 0C 4C 29 C0 48 19 FA 48 89 D1 48 89 C3 49 Ás.L)ÀH.úH%ÑH%Ài
006914E0 29 D9 48 19 CE 44 89 D9 48 89 F0 48 D3 E0 44 89 )ÜH.ID%ÜH%8HÖæD%
006914F0 D1 49 D3 E9 48 D3 EE 4C 09 C8 48 89 F2 48 8B 5D ÑiÖéHÖiL.ÈH%ðH<]
00691500 F8 C9 C3 0F 1F 44 00 00 4D 29 C1 48 19 FE 66 49 øEÄ..D..M)ÁH.pfi
00691510 0F 6E C1 66 48 0F 6E D6 66 0F 6C C2 0F 29 45 E0 .nÁfH.nÖf.lÄ.)Eà
00691520 E9 12 FF FF FF 00 00 00 F3 0F 1E FA 48 83 EC 08 é.yyy...ó..úHfi.
00691530 48 83 C4 08 C3 00 00 00 00 00 00 00 00 00 00 00 HfÄ.Ä.....
00691540 4D 61 69 6E 00 6D 61 69 6E 00 48 34 73 49 41 48 Main.main.H4sIAH
00691550 6D 68 46 6D 67 41 2F 2B 30 37 62 59 77 62 31 62 mhFmgA/+07bYwb1b
00691560 57 7A 6D 79 59 73 45 4D 67 47 45 6C 67 53 49 43 WzmyYsEMgGE1gSIC
00691570 34 6C 54 31 41 33 39 74 72 65 44 32 38 67 45 4B 41TlA39treD28gEK
00691580 2F 58 36 2F 47 75 37 66 57 75 37 55 31 32 31 58 /X6/Gu7fWu7U121X
00691590 59 59 32 32 4E 37 64 6D 32 50 64 32 61 38 75 37 YY22N7dm2Pd2a8u7
006915A0 59 71 47 70 53 2B 69 69 56 4A 6C 66 65 6E 56 48 YqGpS+iiVJlfenVH
006915B0 72 76 52 31 2F 56 30 6B 68 46 72 2F 6C 52 70 43 rvR1/V0khFr/lRpC
006915C0 41 39 71 51 46 4B 2B 42 41 67 55 67 6B 70 72 2F A9qQFK+BAGUgkpr/
006915D0 32 44 45 46 53 4A 67 4B 65 67 55 67 68 55 78 4A 2DEFSJgKegUghUxJ
006915E0 77 37 63 34 38 7A 65 33 63 64 6F 46 54 71 6E 7A w7c48ze3cdoFTqznz
006915F0 6E 57 2B 4D 77 39 39 35 78 37 7A 7A 33 33 7A 70 nW+Mw995x7zz33zp
00691600 31 7A 37 35 7A 37 77 31 42 30 74 4C 4F 6A 67 30 1z75z7w1B0tLOjg0
00691610 50 59 77 44 33 41 6B 64 54 50 75 38 33 30 66 6B PYwD3AkdTPu830fk
00691620 71 76 44 72 5A 59 67 4F 62 6E 62 6F 44 2F 58 64 qvDrZYgObnboD/Xd
00691630 79 64 33 43 5A 49 66 38 50 43 78 2B 4A 7A 6E 61 yd3C2If8PCx+Jzna
00691640 74 78 56 36 73 65 55 36 36 48 33 4B 79 44 62 36 txV6seU66H3KyDb6
00691650 64 38 69 46 46 44 67 6F 6C 63 4F 7A 68 4A 46 4C d8iFFDgolcOzhJFL
00691660 4A 67 6A 72 61 44 79 47 32 30 70 46 6E 38 46 4E JgJraDyG20pFn8FN
00691670 55 50 73 56 58 4F 71 4D 39 42 36 51 79 2B 33 53 UPsVX0qM9B6Qy+3S
00691680 6A 30 43 72 62 4B 45 52 56 4F 37 44 48 54 4A 2F j0CrbKERV07DHTJ/
00691690 61 74 78 6C 56 61 7A 7A 4A 54 48 30 6B 53 75 56 atx1VazzJTH0kSuV
006916A0 4E 55 37 68 54 6C 52 33 79 52 47 67 49 78 32 70 NU7hTlR3yRGgIx2p
006916B0 50 6F 53 43 34 2F 74 52 2B 4C 65 79 6B 66 59 6C PoSC4/tR+Leykfy1
006916C0 61 75 53 50 6C 59 50 45 4C 35 45 4B 50 74 6B 2B auSP1YPEL5EKPtk+
006916D0 2F 6F 75 58 2B 6B 76 67 53 56 75 2F 39 75 4D 38 /ouX+kvgsVu/9uM8
006916E0 33 69 50 73 71 48 47 4F 75 62 42 44 6E 73 30 69 38 44 32 4C 31 54 74 4C 35 32 2F 58 43 51 32 68 3iPsqHG0ubBDns018D21TtL52/XCQ2h
006916F0 38 78 6A 6A 4E 33 53 63 34 4D 39 4C 6C 4C 75 54 30 6C 75 56 4A 62 33 72 50 73 48 39 67 7A 30 4F 8xjjN3Sc4M9L1LuT0luVjb3rPsH9gz00
00691700 66 53 46 4A 65 33 70 52 63 5A 41 71 54 62 77 2F 45 30 36 62 66 54 68 45 61 62 7A 33 55 65 4D 74 fSFJe3pRcZ4gTbw/E06bfThEabz3UeMt
00691710 4E 59 48 72 6E 66 52 76 4D 4A 66 2F 46 6C 64 34 2B 2F 2B 75 4C 48 39 7A 2F 31 73 34 57 44 6E 2F NYHrnfRvMJf/Fld4+/+uLH9z/1s4WDn/
00691720 35 58 75 6E 37 72 4B 37 63 51 48 71 49 33 6B 53 45 38 2B 4A 7A 67 45 4D 48 78 51 57 79 4A 34 34 5Pun7rk7cQHqI3kSE8+JzgEMHxQWYJ44
00691730 76 6A 48 6C 6C 33 37 50 53 61 2B 57 72 32 65 55 47 75 47 35 61 68 7A 35 74 47 64 70 57 49 50 vjH1137PSa+Wr2eUGuG5ahz5tGdpWIP
00691740 57 73 78 30 2F 71 58 6F 2F 75 61 6B 4E 58 32 35 52 50 74 46 2B 58 76 67 75 75 39 5A 72 51 4C Wsx0/qXo/uaKNX2SRPtF+Pvguuu9ZrQdL
00691750 57 6D 61 35 79 6D 71 79 57 70 77 67 6B 43 39 46 4A 57 30 48 52 52 31 59 57 79 4B 42 4E 4B 64 6C Wma5ymqyWpwgkC9FJW0HRR1YWyKBKNd1
00691760 6B 55 38 6E 4A 46 4C 4D 6B 4E 69 61 75 71 63 6B 58 50 63 34 53 4C 64 4E 30 41 46 34 35 47 68 6F kU8nJFLMkN1auqckXPC4SLdN0AF45Gho
00691770 4F 43 31 2B 56 31 39 62 66 75 66 58 32 63 45 45 6E 46 68 4A 79 6B 53 67 56 5A 39 79 55 31 46 51 OC1+V19bfufX2cEEfHJykSgV28Y01FQ
00691780 75 57 6C 49 71 55 45 6A 4D 6C 43 59 6F 73 6C 4A 55 4B 72 55 71 57 64 64 6C 4E 48 71 6E 41 33 uW1IqEjM1CYos1JUKrUQWdd1NHqnA3
00691790 71 4D 2F 44 71 4D 4F 32 4B 2F 54 71 37 66 4D 68 35 72 4F 2B 52 72 53 61 38 2B 53 47 6D 2F 2B 34 qM/DqM02K/Tq7Fmh5r0+RrSa8+Sgm/+4
00691800 2B 66 67 69 6B 37 75 58 47 61 6A 2B 4D 53 2B 2F 50 51 56 6A 71 66 4D 66 54 39 6C 4E 46 34 31 +fgik7uXGaj+MS+/PQVjqfMfT91N5FJ1
00691810 71 6B 59 37 71 62 70 56 4D 6A 6E 79 4D 56 30 2F 36 37 4D 75 74 7A 33 4A 73 57 75 6E 57 38 6E 62 qkY7qbpVMjnyMW0/67Mutz3J3sWunW8nb
00691820 66 51 72 63 2F 66 52 51 76 39 47 67 76 39 6B 6F 57 4F 64 52 4D 34 54 65 6D 45 46 35 38 52 41 69 fQrc/fRQv9Gv9kQW0DRM4TemE58RA1

```

We can find the same string at the memory location 0xa91540 in Binary Ninja (or any dissassembler):

```

.rodata (PROGBITS) section started {0xa91540-0xad2808}
00a91540 char const data_a91540[0x5] = "Main", 0
00a91545 char const data_a91545[0x5] = "main", 0

00a9154a c1SU_str:
00a9154a                                     48 34 73 49 41 48-6d 68 46 6d 67 41 2f 2b-30 37 62 59 77 62 31 62                                     H4sIAHmhFmgA/+07bYwb1b
00a91560 57 7a 6d 79 59 73 45 4d-67 47 45 6c 67 53 49 43-34 6c 54 31 41 33 39 74-72 65 44 32 38 67 45 4b WzmyYsEMgGE1gSIC41T1A39treD28gEK
00a91580 2f 58 36 2f 47 75 37 66-57 75 37 55 31 32 31 58-59 59 32 32 4e 37 64 6d-32 50 64 32 61 38 75 37 /X6/Gu7fWu7U121XY22N7dm2Pd2a8u7
00a915a0 59 71 47 70 53 2b 69 69-56 4a 6c 66 65 6e 56 48-72 76 52 31 2f 56 30 6b-68 46 72 2f 6c 52 70 43 YqGpS+iiVJlfenVHrvR1/V0khFr/lRpC
00a915c0 41 39 71 51 46 4b 2b 42-41 67 55 67 6b 70 72 2f-32 44 45 46 53 4a 67 4b-65 67 55 67 68 55 78 4a A9qQFK+BAGUgkpr/2DEFSJgKegUghUxJ
00a915e0 77 37 63 34 38 7a 65 33-63 64 6f 46 54 71 6e 7a-6e 57 2b 4d 77 39 39 35-78 37 7a 7a 33 33 7a 70 w7c48ze3cdoFTqznznW+Mw995x7zz33zp
00a91600 31 7a 37 35 7a 37 77 31-42 30 74 4c 4f 6a 67 30-50 59 77 44 33 41 6b 64-54 50 75 38 33 30 66 6b 1z75z7w1B0tLOjg0PYwD3AkdTPu830fk
00a91620 71 76 44 72 5a 59 67 4f-62 6e 62 6f 44 2f 58 64-79 64 33 43 5a 49 66 38-50 43 78 2b 4a 7a 6e 61 qvDrZYgObnboD/Xdyd3C2If8PCx+Jzna
00a91640 74 78 56 36 73 65 55 36-36 48 33 4b 79 44 62 36-64 38 69 46 46 44 67 6f-6c 63 4f 7a 68 4a 46 4c txV6seU66H3KyDb6d8iFFDgolcOzhJFL
00a91660 4a 67 6a 72 61 44 79 47-32 30 70 46 6e 38 46 4e-55 50 73 56 58 4f 71 4d-39 42 36 51 79 2b 33 53 JgJraDyG20pFn8FNUPsVX0qM9B6Qy+3S
00a91680 6a 30 43 72 62 4b 45 52-56 4f 37 44 48 54 4a 2f-61 74 78 6c 56 61 7a 7a-4a 54 48 30 6b 53 75 56 j0CrbKERV07DHTJ/atx1VazzJTH0kSuV
00a916a0 4e 55 37 68 54 6c 52 33-79 52 47 67 49 78 32 70-50 6f 53 43 34 2f 74 52-2b 4c 65 79 6b 66 59 6c NU7hTlR3yRGgIx2pPoSC4/tR+Leykfy1
00a916c0 61 75 53 50 6c 59 50 45-4c 35 45 4b 50 74 6b 2b-2f 6f 75 58 2b 6b 76 67-53 56 75 2f 39 75 4d 38 auSP1YPEL5EKPtk+/ouX+kvgsVu/9uM8
00a916e0 33 69 50 73 71 48 47 4f-75 62 42 44 6e 73 30 69-38 44 32 4c 31 54 74 4c-35 32 2f 58 43 51 32 68 3iPsqHG0ubBDns018D21TtL52/XCQ2h
00a91700 38 78 6a 6a 4e 33 53 63-34 4d 39 4c 6c 4c 75 54-30 6c 75 56 4a 62 33 72-50 73 48 39 67 7a 30 4F 8xjjN3Sc4M9L1LuT0luVjb3rPsH9gz00
00a91720 66 53 46 4a 65 33 70 52-63 5a 41 71 54 62 77 2f-45 30 36 62 66 54 68 45-61 62 7a 33 55 65 4d 74 fSFJe3pRcZ4gTbw/E06bfThEabz3UeMt
00a91740 4e 59 48 72 6e 66 52 76-4d 4a 66 2f 46 6c 64 34-2b 2f 2b 75 4c 48 39 7a-2f 31 73 34 57 44 6e 2F NYHrnfRvMJf/Fld4+/+uLH9z/1s4WDn/
00a91760 35 58 75 6e 37 72 4b 37-63 51 48 71 49 33 6b 53-45 38 2b 4a 7a 67 45 4d-48 78 51 57 79 4a 34 34 5Pun7rk7cQHqI3kSE8+JzgEMHxQWYJ44
00a91780 76 6a 48 6c 6c 33 37-50 53 61 2b 57 72 32 65-55 47 75 47 35 61 68 7a-35 74 47 64 70 57 49 50 vjH1137PSa+Wr2eUGuG5ahz5tGdpWIP
00a917a0 57 73 78 30 2f 71 58 6f-2f 75 61 6b 4e 58 32 35-52 50 74 46 2b 58 76 67-75 75 39 5a 72 51 4C Wsx0/qXo/uaKNX2SRPtF+Pvguuu9ZrQdL
00a917c0 57 6d 61 35 79 6d 71 79-57 70 77 67 6b 43 39 46-4a 57 30 48 52 52 31 59-57 79 4b 42 4e 4b 64 6C Wma5ymqyWpwgkC9FJW0HRR1YWyKBKNd1
00a917e0 6b 55 38 6e 4a 46 4c 4d-6b 4e 69 61 75 71 63 6b 58-50 63 34 53 4c 64 4e-30 41 46 34 35 47 68 6F kU8nJFLMkN1auqckXPC4SLdN0AF45Gho
00a91800 4f 43 31 2b 56 31 39 62-66 75 66 58 32 63 45 45 6e 46 68 4a 79 6b 53 67-56 5a 39 79 55 31 46 51 OC1+V19bfufX2cEEfHJykSgV28Y01FQ
00a91820 75 57 6c 49 71 55 45 6a-4d 6c 43 59 6f 73 6c 4a 55 4b 72 55 71 57 64-64 6c 4e 48 71 6e 41 33 uW1IqEjM1CYos1JUKrUQWdd1NHqnA3
00a91840 71 4d 2f 44 71 4d 4f 32-4b 2f 54 71 37 66 4d 68 35 72 4f 2b 52 72 53 61 38 2b 53 47 6d 2f 2b 34 qM/DqM02K/Tq7Fmh5r0+RrSa8+Sgm/+4
00a91860 2b 66 67 69 6b 37 75 58-47 61 6a 2b 4d 53 2b 2f 50 51 56 6a 71 66 4d 66 54 39 6c 4e 46 34 31 +fgik7uXGaj+MS+/PQVjqfMfT91N5FJ1
00a91880 71 6b 59 37 71 62 70 56-4d 6a 6e 79 4d 56 30 2f 36 37 4d 75 74 7a 33 4a 73 57 75 6e 57 38 6e 62 qkY7qbpVMjnyMW0/67Mutz3J3sWunW8nb
00a918a0 66 51 72 63 2f 66 52 51-76 39 47 67 76 39 6b 6f 57 4f 64 52 4d 34 54 65 6d 45 46 35 38 52 41 69 fQrc/fRQv9Gv9kQW0DRM4TemE58RA1

```

This shift is due to the fact that the dissassembler starts at 0x00400000

We can see that after the base64 code we have some error catchers for base64 and that we have important some gzip functions which hint to the next steps in the process:

The base64 text when put into CyberChef:

we see that a gzip file is detected, as expected

[illegible]

And we see that an ELF file is hidden inside:

In the disassembler we find 2 sections relevant for us: the main and what seems like a base64 encoded string:

```
Linear ▾ High Level IL ▾

int32_t main(int32_t argc, char** argv, char** envp)

0000119f      puts(str: "Initializing subsystem...")
000011bd      printf(format: "Loaded transport block at address_ ", transport_block)
000011e3      printf(format: "Transport block size: %zu bytes\n", strlen(transport_block))
000011f2      puts(str: "Applying configuration profile:")
0000120d      printf(format: "  - Alpha Shift Parameter: %d\n", 0xd)
00001228      printf(format: "  - Radix Mode Setting:      %d\n", 0x40)
00001237      puts(str: "Configuration applied.")
00001246      puts(str: "Performing preliminary validation_ ")
00001269      printf(format: "Executing critical check... (Poi_ ", 0)
00001274      int32_t var_14
00001274      __builtin_strncpy(dest: &var_14, src: "\x7fELF", n: 4)
0000128b      printf(format: "Validation successful! Result: %_", 0x464c457f)
0000129a      puts(str: "Proceeding with main operation... ")
000012a5      return 0

.text (PROGBITS) section ended {0x10a0-0x12a6}

000012a6      00 00

.fini (PROGBITS) section started {0x12a8-0x12b5}

000012a8      int64_t _fini() __pure {...}

.fini (PROGBITS) section ended {0x12a8-0x12b5}

.rodata (PROGBITS) section started {0x2000-0x243a}
00002000      uint32_t _IO_stdin_used = 0x20001

00002004      00 00 00 00-2b 53 6b 4e 4e 43 6c 2b-62 52 5a 4e 2f 32 2f 48-2b 49 6c 7a 4e 66 4e 32      ...+SkNNC1+bRZN/2/H+IlzNfN2
00002020      2f 7a 73 47 43 2f 6a 46-73 77 42 56 62 5a 63 70-61 61 41 66 37 74 79 45-66 49 56 74 77 4d 32 4c      /zsGC/jFswBVbZcpaaAf7tyEfIVtwM2L
00002040      41 42 4c 65 4e 62 41 35-77 50 41 48 33 49 4c 4a-71 61 68 4f 33 4a 71 6d-2f 73 6a 51 6e 36 56 77      ABLeNba5wPAH3ILJqah03Jqm/sjQn6Vw
00002060      6c 62 51 7a 53 79 6e 74-53 50 30 48 6f 51 6a 74-62 50 6f 7a 4d 76 73 66-34 61 54 44 4c 4a 48 7a      lbQzSyntSP0HoQjtbPozMvsf4aTDLJHz
00002080      69 37 63 7a 7a 4d 38 77-39 54 6e 49 4c 35 6b 61-61 6c 72 32 37 67 35 63-43 74 75 6e 4a 55 5a 52      i7czzM8w9TnIL5kaalr27g5cCtunJUZR
000020a0      6c 7a 33 68 62 49 38 47-56 5a 42 65 51 72 4c 49-35 4a 75 51 4b 4c 68 62-66 7a 6f 32 54 47 76 38      lz3hbI8GVZBeQrLI5JuQKLhbfzo2TGv8
000020c0      48 48 6a 46 2b 45 39 4a-48 4a 4a 62 53 4d 67 4d-65 58 33 63 73 79 72 64-63 4a 32 6c 7a 31 33 47      HHjF+E9JHJbSMgMeX3csyrdcJ2Lz13G
000020e0      79 76 65 39 4b 5a 7a 76-68 54 67 71 59 2b 39 56-53 4e 53 4e 53 4e 53 4e 53 4e 53 2f 78 32      yve9KZzvhtGqY+9VSNsNSNSNSNSNS/x2
00002100      32 6a 38 55 61 57 2b 73-4b 79 43 51 66 33 43 64-2b 48 65 6f 72 69 66 52-6a 44 2f 30 69 35 70 66      2j8UaW+sKyCQf3Cd+HeorifRjD/0i5pf
00002120      69 43 33 43 79 36 66 31-68 4a 52 54 6d 42 64 6e-31 35 57 35 70 66 6a 38-68 6b 44 4f 44 4f 44 4f      iC3Cy6f1hJRTmBdn15W5pfj8hkD0D0D0
00002140      44 4f 44 4f 44 4f 44 4f-39 6c 74 31 6d 33 37 36-73 2f 49 62 38 47 5a 73-62 77 5a 71 69 63 55 73      D0D0D0D091t1m376s/Ib8GZsbwZqicUs
00002160      62 73 45 78 56 39 45 62-39 77 43 2b 59 6d 77 5a-70 76 5a 6b 43 31 34 43-34 79 6e 77 6a 73 65 31      bsExV9Eb9wC+YmwZpvZkC14C4ynwjse1
00002180      2f 59 55 68 43 48 70 56-4d 6c 2b 65 2f 6c 54 31-48 53 6f 2f 47 34 6b 6a-31 37 71 69 55 50 69 68      /YUhChpVml+e/1T1HSo/G4kj17qiUPih
000021a0      4d 41 69 41 38 38 76 58-59 31 51 30 79 62 33 63-50 50 73 45 31 37 75 6f-39 2b 7a 73 69 69 33 4b      MAiA88vXY1Q0yb3cPPsE17uo9+zsii3K
000021c0      62 64 61 2f 71 56 2f 6e-4f 37 4d 32 37 73 34 55-2b 48 42 55 51 4b 38 35-53 50 62 57 73 38 39 42      bda/qV/n07M27s4U+HBuQK85SPbWs89B
000021e0      66 4a 70 76 76 71 66 7a-68 6a 78 43 32 62 7a 34-4d 51 6d 6e 76 74 4c 4c-4e 76 69 65 38 4f 56 54      fJpvvqfzhjxc2bz4MQmnvtLLNvie80VT
00002200      65 2b 46 69 55 48 72 41-37 51 75 45 70 54 45 65-5a 48 75 6e 57 43 75 4c-4e 2b 54 55 59 55 30 75      e+FiuHrA7QuEpTEeZHunWcULn+TUYU0u
00002220      4c 4e 51 47 47 61 72 31-75 37 7a 66 49 5a 4a 54-43 59 5a 6b 6b 33 44 52-39 5a 37 73 69 72 45 67      LNQGGArlu7zfIZJTCYZkk3DR9Z7sirEg
00002240      66 31 53 6a 65 34 53 47-37 31 6a 30 30 46 6b 31-59 52 73 77 33 67 4e 65-2b 4f 32 32 55 74 57 6a      f1Sje4SG71j00Fk1YRsw3gNe+022UtWj
00002260      68 6c 63 48 4d 71 72 65-35 6d 45 70 6d 6b 69 50-6e 62 37 41 55 71 41 66-69 61 34 53 38 55 33 30      hlcHMqre5mEpmkiPnb7AUqAfia4S8U30
00002280      36 74 66 4c 33 31 54 69-72 30 70 53 47 35 58 51-31 44 79 35 76 73 67 43-72 5a 2f 65 66 50 4b 51      6tfL31Tir0pSG5XQ1Dy5vsgCrZ/efPKQ
000022a0      63 4e 56 4d 4e 4e 4e 3d-00 00 00 00      cNVMNNN=...
```

We can see the printf s in the main functions:

```
printf(format: "  - Alpha Shift Parameter: %d\n", 0xd)
```

```
printf(format: "  - Radix Mode Setting:      %d\n", 0x40)
```

the 0xd is 13 in base 10 and 0x40 is 64 in base 10

We can see those values when we execute the file:

```

Initializing subsystem...
Loaded transport block at address: 0x562db72ba008
Transport block size: 672 bytes
Applying configuration profile:
  - Alpha Shift Parameter: 13
  - Radix Mode Setting:    64
Configuration applied.
Performing preliminary validation...
Executing critical check... (Pointer: (nil))
Segmentation fault (core dumped)

```

This is the hint for us to use ROT13 and base64:

But it still seems that we get some gibberish

This is where we use the given hint:

I remember that my friend told me last week: "If you're ever stuck remember to move 37 steps to the left".

Which refers to a Caesar shifting to the left for each byte:

Using a simple python script we can do that

```

import sys

def rotate_left(b, bits):
    bits %= 8
    return ((b << bits) & 0xFF) | (b >> (8 - bits))

def main():
    if len(sys.argv) != 3:
        print(f"Usage: {sys.argv[0]} <input> <output>")

```

```
if __name__ == '__main__':  
    main()
```

[illegible][illegible]

We get the last executable file that when disassembled gives us this:

```

00401000 void _start() __noreturn

00401000 void* rsi = &secret_encoded
00401000
0040101a for (int64_t i = 0x2f; i != 0; i -= 1)
0040101c     *rsi ^= xor_key
0040101e     rsi += 1
0040101e
0040102b *0 = 1
00401038 syscall(sys_exit {0x3c}, status: 0)
00401038 noreturn

.text (PROGBITS) section ended {0x401000-0x40103a}
-----
.data (PROGBITS) section started {0x402000-0x402030}
00402000 char xor_key = -0x56

00402001 secret_encoded:
00402001 ff fc fe d1 f3 9a df-f5 9e d8 99 f5 9d c2 99-f5 e7 9e 9f 9d 99 d8 f5-9a cc f5 f8 99 dc 99 d8
00402020 9f 99 f5 99 c4 cd 9b c4-99 99 d8 9b c4 cd d7 aa
.data (PROGBITS) section ended {0x402000-0x402030}

```

which is simple enough to do

The xor key is shown here as -0x56 which in binary 2's complement gives -01010110 which is

10101010 which evaluates to 0xaa

Therefore with this simple Python script (or with out friend GPT) we can get the flag.

```

xor_key = 0xAA
encoded_bytes = bytes.fromhex(
    "ff fc fe d1 f3 9a df f5 9e d8 99 f5 9d c2 99 f5 e7 9e 9f 9d 99 d8 f5 9a"
    "cc f5 f8 99 dc 99 d8 9f 99 f5 99 c4 cd 9b c4 99 99 d8 9b c4 cd d7 aa"
    .replace(" ", "") # Remove spaces for easier parsing
)
decoded_bytes = bytearray()
for byte in encoded_bytes:
    decoded_bytes.append(byte ^ xor_key)
try:
    flag = decoded_bytes.decode('ascii')
    print(f"Decoded Flag (ASCII): {flag}")
except UnicodeDecodeError:
    print("Failed to decode as ASCII. Decoded bytes (hex):")
    print(decoded_bytes.hex())

```