



Fall 2014

CODEBREAKER CHALLENGE 2.0



Challenge Scenario

"An international terrorist organization has recently revised the operational security (OPSEC) procedures used to communicate with their members in the field.

We have recovered a program that we believe is being used to covertly send encrypted messages. On the surface, the program appears to simply check the weather forecast in a few cities, but we believe there is more to the program than meets the eye. Your mission is to figure out how to execute the covert functionality, reverse-engineer the encryption algorithm, create a decryption program, and lastly figure out a way to decipher a message that was captured from a high-value target."

The Challenge

- There are 4 different levels or "tiers" to this challenge problem
 - Tier 1: Determine how to execute the hidden functionality
 - Tier 2: Bypass an authentication check
 - Tier 3: Create a decryption program
 - Tier 4: Decrypt message from a high-value target
- Each tier gets progressively harder and builds off lower tiers

The Challenge (cont.)

- The program will provide you with directions for where to send an encrypted email after you complete Tier 3
 - Please use your *.edu address so we know you are a student
 - You will then be sent the encrypted message for Tier 4
- Solutions are due by the end of 2014

Getting Started


- Review the 'Getting Started' tips in the Codebreaker Challenge document
- Download the IDA Demo from Hex-Rays
 - https://www.hex-rays.com/products/ida/support/download_demo.html
- Try running the program with different options and observe its behavior
- Disassemble and start analyzing the binary

Reverse Engineering Tips

- Examine strings in the binary using IDA
 - Look for clues that relate to the functionality you are trying to find / reverse
 - Utilize IDA xrefs to find code that references the string(s) of interest
 - Utilize symbols (e.g., function names) to help determine what a section of code does
- Try setting debugger breakpoints to help RE code
 - Single-step after hitting a breakpoint and see how the values in registers/memory change
 - Look for the result of interesting computations. You can sometimes get the data you need from memory
- Leverage online resources, e.g., Intel manuals, RE lectures, etc. for help on reverse-engineering



Reverse Engineering Tips

- Optimizing compilers sometimes generate strange looking code for simple operations
 - In this challenge, you will encounter one such optimization during Tiers 2 – 4
 - To save you some time/frustration, we will walk through an example and explain the math behind the optimization
- 

What does this code do?

```
mov edx, 0xAC769185 // edx = 0xAC769185
mov eax, ecx        // ecx = input value
imul edx            // edx:eax = eax * edx
lea eax, [edx + ecx*0x1] // eax = edx + ecx
mov edx, eax        // edx = eax
sar edx, 0x6        // arith right shift; edx = edx >> 0x6
mov eax, ecx        // eax = ecx
sar eax, 0x1f       // eax = eax >> 0x1f (31)
mov ebx, edx        // ebx = edx
sub ebx, eax        // ebx = ebx - eax
mov eax, ebx        // eax = ebx
imul eax, eax, 0x5f // edx:eax = eax * 0x5f (95)
mov edx, ecx        // edx = ecx
sub edx, eax        // edx = edx - eax
// edx is the final result
```


Signed Division and Remainder

- The code computes: `edx = ecx % 95`
- Why multiply by `0xAC769185` and where did that number come from?
 - Division is a time consuming operation
 - When the divisor is a constant, the compiler can optimize the computation
- The basic trick is to multiply by a “magic value” ($\sim 2^{32}/d$) and extract the leftmost 32 bits of the product
- The following site computes these numbers for you: <http://www.hackersdelight.org/magic.htm>

Signed Division and Remainder

- For wordsize $W \geq 3$ and divisor d , $2 \leq d < 2^{W-1}$, we wish to find the least integer m and p such that:

$$\left\lfloor \frac{mn}{2^p} \right\rfloor = \left\lfloor \frac{n}{d} \right\rfloor \text{ for } 0 \leq n < 2^{W-1} \quad (1a)$$

$$\left\lfloor \frac{mn}{2^p} \right\rfloor + 1 = \left\lfloor \frac{n}{d} \right\rfloor \text{ for } -2^{W-1} \leq n \leq -1 \quad (1b)$$

with $0 \leq m < 2^W$ and $p \geq W$

Signed Division and Remainder

- The “magic number” M used in the multiply instruction is given by:

$$M = \begin{cases} m, & \text{if } 0 \leq m < 2^{W-1} \\ m - 2^W, & \text{if } 2^{W-1} \leq m < 2^W \end{cases}$$

- Because (1b) must hold for $n = -d$,

$$\left\lfloor \frac{-md}{2^p} \right\rfloor + 1 = -1, \text{ which implies}$$

$$\frac{md}{2^p} > 1 \quad (2)$$

Signed Division and Remainder

- Let n_c be the largest value of n such that $\text{rem}(n_c, d) = d-1$. It can be calculated from:

$$\begin{aligned} n_c &= \left\lfloor \frac{2^{W-1}}{d} \right\rfloor * d - 1 \\ &= 2^{W-1} - \text{rem}(2^{W-1}, d) - 1 \quad (3) \end{aligned}$$

- Because (1a) must hold for $n = n_c$:

$$\left\lfloor \frac{mn_c}{2^p} \right\rfloor = \left\lfloor \frac{n_c}{d} \right\rfloor = \frac{n_c - (d-1)}{d} \quad \text{or}$$
$$\frac{mn_c}{2^p} < \frac{n_c + 1}{d} \quad (4)$$

Signed Division and Remainder

- Combining (4) with (2) gives:

$$\frac{2^p}{d} < m < \frac{2^p}{d} \frac{n_c+1}{n_c} \quad (5)$$

- Because m is to be the least integer satisfying (5), it is the next integer greater than $\frac{2^p}{d}$:

$$m = \frac{2^p + d - \text{rem}(2^p, d)}{d} = \left\lfloor \frac{2^p}{|d|} \right\rfloor + 1 \quad (6)$$

$$2^p > n_c(d - \text{rem}(2^p, d)) \quad (7)$$

Signed Division and Remainder

- Algorithm to find M and shift amount s from d
 - Compute n_c using (3)
 - Solve for p by trying successively larger values, starting at W , until satisfying the inequality in (7)
 - When the smallest $p \geq W$ satisfying (7) is found, m is calculated from (6)
 - The shift amount is computed as: $s = p - W$
 - M is simply a reinterpretation of m as a signed integer

64-bit Data Types

Consider the following program:

```
int main(){
    char one           = 0x11;           // sizeof(char) == 1
    char two           = 0x22;
    int three          = 0x33333333;     // sizeof(int) == 4
    int four           = 0x44444444;
    long long five     = 0x5555555555555555; // sizeof(long long) == 8
    long long six      = 0x6666666666666666;
    printf("8b: %hu 32b: %u 64b: %llu\n", one + two, three + four, five + six);
    return 0;
}
```

64-bit Data Types – x86_64

Part 1: Move values onto the stack

```
mov  BYTE PTR [rbp-0x2],0x11
mov  BYTE PTR [rbp-0x1],0x22
mov  DWORD PTR [rbp-0xc],0x33333333
mov  DWORD PTR [rbp-0x8],0x44444444
mov  DWORD PTR [rbp-0x20],0x55555555
mov  DWORD PTR [rbp-0x1c],0x55555555
mov  DWORD PTR [rbp-0x18],0x66666666
mov  DWORD PTR [rbp-0x14],0x66666666
```


64-bit Data Types – x86_64

Part 2: Load into registers and compute

```
mov    rax,QWORD PTR [rbp-0x18]    // 0x6666666666666666 in rax
mov    rdx,QWORD PTR [rbp-0x20]    // 0x7777777777777777 in rdx
lea    rcx,[rdx+rax*1]             // rcx = rax + rdx*1
mov    eax,DWORD PTR [rbp-0x8]     // 0x44444444 in eax
mov    edx,DWORD PTR [rbp-0xc]     // 0x33333333 in edx
add    edx,eax                    // edx = edx + eax
movsx  esi,BYTE PTR [rbp-0x2]     // 0x11 in esi
movsx  eax,BYTE PTR [rbp-0x1]     // 0x22 in eax
add    esi,eax                    // esi = esi + eax
```

64-bit Data Types – x86

No 64-bit registers ☹️

```
long long five = 0x5555555555555555; // sizeof(long long) == 8  
long long six  = 0x6666666666666666;
```

Let's make it work with 32-bit ones!

64-bit Data Types – x86

Part 1: Move values onto the stack (same as x86_64)

```
mov  BYTE PTR [ebp-1],0x11
mov  BYTE PTR [ebp-2],0x22
mov  DWORD PTR [ebp-8],0x33333333
mov  DWORD PTR [ebp-12],0x44444444
mov  DWORD PTR [ebp-24],0x55555555
mov  DWORD PTR [ebp-20],0x55555555
mov  DWORD PTR [ebp-32],0x66666666
mov  DWORD PTR [ebp-28],0x66666666
```

64-bit Data Types – x86

Part 2: Load into registers and compute

```
mov    eax,DWORD PTR [ebp-32]    // 0x66666666 in eax
mov    edx,DWORD PTR [ebp-28]    // 0x66666666 in edx
add    eax,DWORD PTR [ebp-24]    // eax = eax + 0x55555555
adc    edx,DWORD PTR [ebp-20]    // edx = edx + 0x55555555 + CF
...
mov    eax,DWORD PTR [ebp-12]    // 0x44444444 in eax
add    eax,DWORD PTR [ebp-8]     // eax = eax + 0x33333333
...
movsx  edx,BYTE PTR [ebp-1]      // 0x11 in edx
movsx  eax,BYTE PTR [ebp-2]      // 0x22 in eax
lea    eax,[edx+eax]             // eax = edx + eax*
```



Questions



Technical Walkthrough

- Original Codebreaker Challenge
 - Released during Fall 2013
 - Reverse-engineering / crypt challenge
 - Program prompted for a password
 - AES key is derived from SHA256 hash of password
 - AES-encrypted blob w/ instructions for how to submit the solution stored in the executable
 - To verify the correct password was entered, the derived key is used to compute an HMAC of the encrypted blob
 - If HMAC is correct, the blob is decrypted, revealing instructions.
 - Weakness of the design is in the key-derivation function
 - Only first 2-bytes of SHA256 hash matter
 - Easy to brute force

Running the program

```
Command Prompt - CodeBreaker.exe
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>CodeBreaker.exe
Welcome to the NSA Codebreaker Challenge!

Loading.....

To win the challenge, you must be the first to decrypt the code
protected by the password. You are free to use any means at your
disposal to reverse-engineer and/or modify this binary in order to discover
the encryption key. Instructions for submitting your solution will be 'revealed'
when you are successful...good luck!

Enter Password:
```

Running the program (2)

```
CA: Command Prompt
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>
C:\challenge>CodeBreaker.exe
Welcome to the NSA Codebreaker Challenge!

Loading.....

To win the challenge, you must be the first to decrypt the code
protected by the password. You are free to use any means at your
disposal to reverse-engineer and/or modify this binary in order to discover
the encryption key. Instructions for submitting your solution will be 'revealed'
when you are successful...good luck!

Enter Password: password
ACCESS DENIED

C:\challenge>
```




Disassemble

- Disassemble the Codebreaker binary
 - If asked whether you want to use Proximity View
 - Click no
 - Use graph view

Disassemble (2)

IDA v6.6.140625

File Edit Jump Search View Options Windows Help

Load file C:\challenge\CodeBreaker.exe as

Portable executable for 80386 (PE) [pe.idw]

Processor type

MetaPC (disassemble all opcodes) [metapc] Set

Loading segment 0x00000000

Loading offset 0x00000000

Analysis

Enabled

Indicator enabled

Options

Create segments

Load resources

Rename DLL entries

Manual load

Fill segment gaps

Create imports segment

Create FLAT group

Kernel options 1

Kernel options 2

Processor options

DLL directory C:\Windows

OK Cancel Help

Output window

bytes	pages	size	description
352256	43	8192	allocating memory for b-tree
352256	43	8192	allocating memory for virt
262144	32	8192	allocating memory for name

966656			total memory allocated

Loading processor module C:\Program Files (x86)\IDA Demo 6.6\procs\pc.w32 for metapc...OK

Autoanalysis subsystem has been initialized.

Possible file format: Portable executable for 80386 (PE) (C:\Program Files (x86)\IDA Demo 6.6\loaders\pe.idw)

IDC

@:00000000 Down

Disassemble (3)

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

- sub_401000
- _main**
- sub_401310
- sub_4013B0
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0

Graph overview

```
; Attributes: bp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near

var_2D8= dword ptr -2D8h
var_2D4= dword ptr -2D4h
var_2D0= dword ptr -2D0h
var_2CC= dword ptr -2CCh
var_2C8= dword ptr -2C8h
var_2C4= dword ptr -2C4h
var_2C0= dword ptr -2C0h
var_2BC= dword ptr -2BCh
var_2B8= dword ptr -2B8h
var_2B4= dword ptr -2B4h
var_2B0= dword ptr -2B0h
var_2AC= dword ptr -2ACh
var_2A8= dword ptr -2A8h
var_2A4= dword ptr -2A4h
var_2A0= dword ptr -2A0h
var_29C= dword ptr -29Ch
var_298= dword ptr -298h
var_294= dword ptr -294h
var_290= dword ptr -290h
var_28C= dword ptr -28Ch
var_288= dword ptr -288h
var_284= dword ptr -284h
var_280= dword ptr -280h
var_27C= dword ptr -27Ch
var_278= dword ptr -278h
var_274= dword ptr -274h
var_270= dword ptr -270h
var_26C= dword ptr -26Ch
var_268= dword ptr -268h
var_264= dword ptr -264h
var_260= dword ptr -260h
var_25C= dword ptr -25Ch
var_258= dword ptr -258h
var_254= dword ptr -254h
var_250= dword ptr -250h
var_24C= dword ptr -24Ch
var_248= dword ptr -248h
var_244= dword ptr -244h
var_240= dword ptr -240h
var_23C= dword ptr -23Ch
var_238= dword ptr -238h
var_234= dword ptr -234h
var_230= dword ptr -230h
var_22C= dword ptr -22Ch
var_228= dword ptr -228h
var_224= dword ptr -224h
var_220= dword ptr -220h
var_21C= dword ptr -21Ch
var_218= dword ptr -218h
var_214= dword ptr -214h
var_210= dword ptr -210h
var_20C= dword ptr -20Ch
var_208= dword ptr -208h
var_204= dword ptr -204h
var_200= dword ptr -200h
var_1FC= dword ptr -1FCh
var_1F8= dword ptr -1F8h
var_1F4= dword ptr -1F4h
var_1F0= dword ptr -1F0h
var_1EC= dword ptr -1ECh
var_1E8= dword ptr -1E8h
var_1E4= dword ptr -1E4h
var_1E0= dword ptr -1E0h
var_1DC= dword ptr -1DCh
var_1D8= dword ptr -1D8h
var_1D4= dword ptr -1D4h
var_1D0= dword ptr -1D0h
var_1CC= dword ptr -1CCh
var_1C8= dword ptr -1C8h
var_1C4= dword ptr -1C4h
var_1C0= dword ptr -1C0h
var_1BC= dword ptr -1BCh
var_1B8= dword ptr -1B8h
var_1B4= dword ptr -1B4h
var_1B0= dword ptr -1B0h
var_1AC= dword ptr -1ACh
var_1A8= dword ptr -1A8h
var_1A4= dword ptr -1A4h
var_1A0= dword ptr -1A0h
var_19C= dword ptr -19Ch
var_198= dword ptr -198h
var_194= dword ptr -194h
var_190= dword ptr -190h
var_18C= dword ptr -18Ch
var_188= dword ptr -188h
var_184= dword ptr -184h
var_180= dword ptr -180h
var_17C= dword ptr -17Ch
var_178= dword ptr -178h
var_174= dword ptr -174h
var_170= dword ptr -170h
var_16C= dword ptr -16Ch
var_168= dword ptr -168h
var_164= dword ptr -164h
var_160= dword ptr -160h
var_15C= dword ptr -15Ch
var_158= dword ptr -158h
var_154= dword ptr -154h
var_150= dword ptr -150h
var_14C= dword ptr -14Ch
var_148= dword ptr -148h
var_144= dword ptr -144h
var_140= dword ptr -140h
var_13C= dword ptr -13Ch
var_138= dword ptr -138h
var_134= dword ptr -134h
var_130= dword ptr -130h
var_12C= dword ptr -12Ch
var_128= dword ptr -128h
var_124= dword ptr -124h
var_120= dword ptr -120h
var_11C= dword ptr -11Ch
var_118= dword ptr -118h
var_114= dword ptr -114h
var_110= dword ptr -110h
var_10C= dword ptr -10Ch
var_108= dword ptr -108h
var_104= dword ptr -104h
var_100= dword ptr -100h
var_9C= dword ptr -9Ch
var_98= dword ptr -98h
var_94= dword ptr -94h
var_90= dword ptr -90h
var_8C= dword ptr -8Ch
var_88= dword ptr -88h
var_84= dword ptr -84h
var_80= dword ptr -80h
var_7C= dword ptr -7Ch
var_78= dword ptr -78h
var_74= dword ptr -74h
var_70= dword ptr -70h
var_6C= dword ptr -6Ch
var_68= dword ptr -68h
var_64= dword ptr -64h
var_60= dword ptr -60h
var_5C= dword ptr -5Ch
var_58= dword ptr -58h
var_54= dword ptr -54h
var_50= dword ptr -50h
var_4C= dword ptr -4Ch
var_48= dword ptr -48h
var_44= dword ptr -44h
var_40= dword ptr -40h
var_3C= dword ptr -3Ch
var_38= dword ptr -38h
var_34= dword ptr -34h
var_30= dword ptr -30h
var_2C= dword ptr -2Ch
var_28= dword ptr -28h
var_24= dword ptr -24h
var_20= dword ptr -20h
var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
var_0C= dword ptr -0Ch
var_08= dword ptr -08h
var_04= dword ptr -04h
var_00= dword ptr -00h

__cdecl main @ 00401070
```

Output window

```
Compiling file C:\Program Files (x86)\IDA Demo\0-0\idc\idc100ad.idc ...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Type library 'uc6win' loaded. Applying types...
Types applied to 0 names.
Using FLIRT signature: Microsoft VisualC 2-11/net runtime
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
```

IDC

AU: idle Down Disk: 59GB

Observe Strings

- Observe the strings that show up in IDA
 - Click Views->Open Subviews->Strings
 - You should see the strings that are displayed when you run the program

Welcome to the NSA Codebreaker Challenge!

Loading.....

To win the challenge, you must be the first to decrypt the code protected by the password. You are free to use any means at your disposal to reverse-engineer and/or modify this binary in order to discover the encryption key. Instructions for submitting your solution will be 'revealed' when you are successful...good luck!

Enter Password:

Observe Strings (2)

The screenshot shows the IDA Pro interface with the 'View' menu open. The 'Strings' option is highlighted, with a keyboard shortcut of Shift+F12. The main window displays assembly code with a highlighted line: `char **argv, const char **envp)`. The 'Output window' at the bottom shows the following text:

```
Compiling file C:\Program Files (x86)\IDA Demo 6.0\idc\idc60ad.idc ...  
Executing function 'OnLoad'...  
IDA is analysing the input file...  
You may start to explore the input file right now.  
Type library 'ucówin' loaded. Applying types...  
Types applied to 0 names.  
Using FLIRT signature: Microsoft VisualC 2-11/net runtime  
Propagating type information...  
Function argument information has been propagated  
The initial autoanalysis has been finished.
```

At the bottom left, there is a button labeled 'IDC' and a text prompt 'Open strings window'.

Observe Strings (3)

The screenshot shows the IDA Pro interface with the Strings window open. The Strings window displays a list of strings with their addresses, lengths, and types. The strings are sorted by address. The output window at the bottom shows the progress of the initial autoanalysis.

Address	Length	Type	String
.rdata:004117F7	00000005	C	\a\b\t\n\v
.rdata:00411810	0000005F	C	\!#\$%&'()*+,-./0123456789;<=>?@ABCDEFGHIJKLMN... PQRSTUVWXYZ[\]^_`ABCDEFGHIJK...
.rdata:004118F6	00000005	C	\a\b\t\n\v
.rdata:0041190F	0000005F	C	\!#\$%&'()*+,-./0123456789;<=>?@ABCDEFGHIJKLMN... PQRSTUVWXYZ[\]^_`abcdefghijklmnop...
.rdata:00411980	00000036	C	Welcome to the NSA Codebreaker Challenge!\n\nLoading...
.rdata:004119B8	00000005	C	...\n
.rdata:004119C8	00000041	C	To win the challenge, you must be the first to decrypt the code\n
.rdata:00411A10	00000042	C	protected by the password. You are free to use any means at your\n
.rdata:00411A58	0000004D	C	disposal to reverse-engineer and/or modify this binary in order to discover\n
.rdata:00411AA8	00000052	C	the encryption key. Instructions for submitting your solution will be 'revealed'\n
.rdata:00411AFC	00000027	C	when you are successful...good luck!\n\n
.rdata:00411B24	00000011	C	Enter Password:
.rdata:00411B38	00000009	C	%32[^\n\r]
.rdata:00411B44	0000000F	C	ACCESS DENIED\n
.rdata:00411C62	00000006	C	&\ \x1B.
.rdata:00411CE2	00000005	C	AQ\b7
.rdata:00411E88	0000000D	C	=j&&LZ66IA??-
.rdata:00411EB8	00000005	C	S11b?
.rdata:00411EC8	00000005	C	e##F^
.rdata:00411F08	00000005	C	t,X.
.rdata:00411F0F	00000005	C	4-\x1B\x1B6
.rdata:00411F24	00000005	C	M;:va
.rdata:00411F2F	00000006	C	}()R>

Output window:

```
Compiling file C:\Program Files (x86)\IDA\demo\0-0\idc\onload.idc ...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Type library 'ucówin' loaded. Applying types...
Types applied to 0 names.
Using FLIRT signature: Microsoft VisualC 2-11/net runtime
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
```

ACCESS DENIED

- Double click on the “ACCESS DENIED\n” string
 - This takes you to the data section of the binary where the string is stored
- To the right of the string are cross references to this address (show up as DATA XREF in IDA)
- Press ctrl-x to pull up a cross-references window; you will see two different references

ACCESS DENIED (2)

The screenshot shows the IDA Pro interface with the following details:

- File Name:** C:\challenge\CodeBreaker.exe
- Functions window:** Lists various functions including `sub_401000`, `_main`, `sub_401310`, `sub_4013B0`, `sub_401450`, `sub_4014E0`, `sub_401620`, `sub_401670`, `sub_401D30`, `sub_401DC0`, `sub_401FE0`, `sub_4023A0`, `sub_4028F0`, `sub_4029E0`, `sub_403000`, `sub_403060`, `sub_403140`, `sub_404160`, `sub_4041F0`, `sub_4043B0`, `sub_404420`, `sub_4044A0`, `sub_406510`, `sub_4065E0`, `__security_check_cookie(x)`, `_vscanf`, `_scanf`, `sub_4067FB`, and `__initstdio`.
- Assembly Code:**

```
.rdata:00411AFc aWhenYouAreSucc db 'when you are successful...good luck!',0Ah  
.rdata:00411AFc ; DATA XREF: sub_401000+54f0  
.rdata:00411AFc db 0Ah,0  
.rdata:00411B23 align 4  
.rdata:00411B24 ; char aEnterPassword[]  
.rdata:00411B24 aEnterPassword db 'Enter Password: ',0 ; DATA XREF: _main+7Efo  
.rdata:00411B35 align 4  
.rdata:00411B38 ; char a32[]  
.rdata:00411B38 a32 db '%32[^\',0Ah ; DATA XREF: _main+90fo  
.rdata:00411B38 db 0Dh,']',0  
.rdata:00411B41 align 4  
.rdata:00411B44 ; char aAccessDenied[]  
.rdata:00411B44 aAccessDenied db 'ACCESS DENIED',0Ah,0 ; DATA XREF: _main+B4fo  
.rdata:00411B44 ; _main:loc_4012D5fo  
.rdata:00411B53 align 4  
.rdata:00411B54 ; char aS[]  
.rdata:00411B54 aS db '%s',0Ah,0 ; DATA XREF: _main+242fo  
.rdata:00411B58 dword_411B58 dd 0D728AE22h ; DATA XREF: sub_4044A0+18Ftr  
.rdata:00411B5C dword_411B5C dd 428A2F98h ; DATA XREF: sub_4044A0+1961r  
.rdata:00411B60 dword_411B60 dd 23EF65CDh ; DATA XREF: sub_4044A0+37D1r  
.rdata:00411B64 dword_411B64 dd 71374491h ; DATA XREF: sub_4044A0+3841r  
.rdata:00411B68 dword_411B68 dd 0EC4D3B2Fh ; DATA XREF: sub_4044A0+57D1r  
.rdata:00411B6C dword_411B6C dd 0B5C0FBCFh ; DATA XREF: sub_4044A0+5841r  
.rdata:00411B70 dword_411B70 dd 8189DBBCh ; DATA XREF: sub_4044A0+7801r  
.rdata:00411B74 dword_411B74 dd 0E9B5DBA5h ; DATA XREF: sub_4044A0+7871r  
.rdata:00411B78 dword_411B78 dd 0F348B538h ; DATA XREF: sub_4044A0+9801r  
.rdata:00411B7C dword_411B7C dd 3956C25Bh ; DATA XREF: sub_4044A0+9871r  
.rdata:00411B80 dword_411B80 dd 0B605D019h ; DATA XREF: sub_4044A0+B831r  
.rdata:00411B84 dword_411B84 dd 59F111F1h ; DATA XREF: sub_4044A0+B8A1r  
.rdata:00411B88 dword_411B88 dd 0AF194F9Bh ; DATA XREF: sub_4044A0+D8C1r  
.rdata:00411B8C dword_411B8C dd 923F82A4h ; DATA XREF: sub_4044A0+D931r  
.rdata:00411B90 dword_411B90 dd 0DA6D8118h ; DATA XREF: sub_4044A0+F981r  
.rdata:00411B94 dword_411B94 dd 0AB1C5ED5h ; DATA XREF: sub_4044A0+F9F1r  
.rdata:00411B98 dword_411B98 dd 0A3030242h ; DATA XREF: sub_4044A0+11A71r  
.rdata:00411B9C dword_411B9C dd 0D807AA98h ; DATA XREF: sub_4044A0+11AE1r
```
- Status Bar:** Line 1 of 235, 00010144 00411B44: .rdata:aAccessDenied
- Output window:** The initial autoanalysis has been finished.
- Buttons:** IDC
- System Info:** AU: idle, Down, Disk: 59GB

ACCESS DENIED (3)

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name

- sub_401000
- _main
- sub_401310
- sub_4013B0
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0
- sub_403000
- sub_403060
- sub_403140
- sub_404160
- sub_4041F0
- sub_4043B0
- sub_404420
- sub_4044A0
- sub_406510
- sub_4065E0
- __security_check_cookie(x)
- _vscanf
- _scanf
- sub_4067FB
- __initstdio

```
.rdata:00411AFc aWhenYouAreSucc db 'when you are successful...good luck!',0Ah
.rdata:00411AFc ; DATA XREF: sub_401000+54fo
.rdata:00411AFc db 0Ah,0
.rdata:00411B23 align 4
.rdata:00411B24 ; char aEnterPassword[]
.rdata:00411B24 aEnterPassword db 'Enter Password: ' 0 ; DATA XREF: _main+7Efo
.rdata:00411B35 align 4
.rdata:00411B38 ; char a32[]
.rdata:00411B38 a32 db '%32[^\',0Ah ; DATA XREF: _main+90fo
.rdata:00411B38 db 0Dh,']',0
.rdata:00411B41 align 4
.rdata:00411B44 ; char aAccessDenied[]
.rdata:00411B44 aAccessDenied db 'ACCESS DENIED',0Ah,0 ; DATA XREF: _main+B4fo
.rdata:00411B44 ; _main:loc_4012D5fo
.rdata:00411B53 align 4
.rdata:00411B54 ; char aS[]
.rdata:00411B54 aS db '%s',0Ah,0 ; DATA XREF: _main+242fo
.rdata:00411B58 dword_411B58 dd 0D728AE22h ; DATA XREF: sub_4044A0+18Ftr
.rdata:00411B5C dword_411B5C dd 428A2F98h ; DATA XREF: sub_4044A0+196tr
.rdata:00411B60 dword_411B60 dd 23EF65CDh ; DATA XREF: sub_4044A0+37Dtr
.rdata:00411B64 dword_411B64 dd 71374491h ; DATA XREF: sub_4044A0+384tr
.rdata:00411B68 dword_411B68 dd 0EC4D3B2Fh ; DATA XREF: sub_4044A0+57Dtr
.rdata:00411B6C dword_411B6C dd 0B5C0FBCFh ; DATA XREF: sub_4044A0+584tr
.rdata:00411B70 dword_411B70 dd 8189DBBCh ; DATA XREF: sub_4044A0+780tr
.rdata:00411B74 dword_411B74 dd 0E9B5DBA5h ; DATA XREF: sub_4044A0+787tr
.rdata:00411B78 dword_411B78 dd 0F348B538h ; DATA XREF: sub_4044A0+980tr
.rdata:00411B7C dword_411B7C dd 3956C25Bh ; DATA XREF: sub_4044A0+987tr
.rdata:00411B80 dword_411B80 dd 0B605D019h ; DATA XREF: sub_4044A0+B83tr
.rdata:00411B84 dword_411B84 dd 59F111F1h ; DATA XREF: sub_4044A0+B8Atr
.rdata:00411B88 dword_411B88 dd 0AF194F9Bh ; DATA XREF: sub_4044A0+D8Ctr
.rdata:00411B8C dword_411B8C dd 923F82A4h ; DATA XREF: sub_4044A0+D93tr
.rdata:00411B90 dword_411B90 dd 0DA6D8118h ; DATA XREF: sub_4044A0+F98tr
.rdata:00411B94 dword_411B94 dd 0AB1C5ED5h ; DATA XREF: sub_4044A0+F9Ftr
.rdata:00411B98 dword_411B98 dd 0A3030242h ; DATA XREF: sub_4044A0+11A7tr
.rdata:00411B9C dword_411B9C dd 0D807AA98h ; DATA XREF: sub_4044A0+11AEtr
```

Line 1 of 235

Output window

The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 59GB

Referenced at
_main+7E

Double-click Reference

- You should now be looking at disassembled x86 code
 - We just leveraged the fact that in order to print the ACCESS DENIED message to the screen, the code had to reference the address in the data section of the program where the string was stored.
- Using xrefs in IDA is a quick and easy way to find interesting code sections

Double-click Reference (2)

The screenshot shows the IDA Pro interface with the following components:

- Functions window:** Lists various subroutines such as `sub_401000`, `_main`, `sub_401310`, etc.
- Assembly view:** Displays the following assembly code:

```
mov [esp+2D8h+var_4], eax
xor eax, eax
push ebx
push esi
xor ebx, ebx
push 0EFh ; size_t
mov [esp+2E4h+var_13B], eax
mov [esp+2E4h+var_137], eax
mov [esp+2E4h+var_133], eax
mov [esp+2E4h+var_12F], eax
mov [esp+2E4h+var_12B], eax
mov [esp+2E4h+var_127], eax
mov [esp+2E4h+var_123], eax
mov [esp+2E4h+var_11F], eax
lea eax, [esp+2E4h+var_F7]
push ebx ; int
push eax ; void *
mov [esp+2ECh+var_13C], bl
mov [esp+2ECh+var_F8], bl
call _memset
call sub_401000
push offset aEnterPassword ; "Enter Password: "
call _printf
lea ecx, [esp+2F0h+var_13C]
push ecx
push offset a32 ; "%32[^\n\r]"
call _scanf
lea eax, [esp+2F8h+var_13C]
add esp, 18h
lea edx, [eax+1]
```
- Graph overview:** Shows a control flow graph with a highlighted block.
- Output window:** Contains the message: "The initial autoanalysis has been finished." and an IDC button.
- Status bar:** Shows "AU: idle", "Down", and "Disk: 59GB".

A double-click reference is shown as a blue arrow pointing from the instruction `lea edx, [eax+1]` to a call window for `loc_401117:`.

Explore Code Block

- Explore the code block preceding the access denied message (note: you can double-click call statements to visit the function body)
- You will see routines to:
 - print the initial welcome screen,
 - prompt the user for the password and
 - compute the password length
- The branch for “ACCESS DENIED” is taken when the strlen of the password is 0.

Explore Code Block (2)

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

- sub_401000
- _main
- sub_401310
- sub_4013B0
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0
- sub_403000
- sub_403060
- sub_403140
- sub_404160
- sub_4041F0
- sub_4043B0
- sub_404420
- sub_4044A0

```
mov [esp+2D8h+var_4], eax
xor  eax, eax
push ebx
push esi
xor  ebx, ebx
push 0EFh ; size_t
mov  [esp+2E4h+var_13B], eax
mov  [esp+2E4h+var_137], eax
mov  [esp+2E4h+var_133], eax
mov  [esp+2E4h+var_12F], eax
mov  [esp+2E4h+var_12B], eax
mov  [esp+2E4h+var_127], eax
mov  [esp+2E4h+var_123], eax
mov  [esp+2E4h+var_11F], eax
lea  eax, [esp+2E4h+var_F7]
push ebx ; int
push eax ; void *
mov  [esp+2ECh+var_13C], b1
mov  [esp+2ECh+var_F8], b1
call _memset
call sub_401000
push offset aEnterPassword ; "Enter Password: "
call _printf
lea  ecx, [esp+2F0h+var_13C]
push ecx
push offset a32 ; "%32[^\n\r]"
call _scanf
lea  eax, [esp+2F8h+var_13C]
add  esp, 18h
lea  edx, [eax+1]
```

Line 1 of 235

Graph overview

100.00% (9,717) (961,306) 00000551 00401151: _main:loc_401151

Output window

The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 59GB

Explore Code Block (3)

Initializes 33 byte buffer to zero.

```
mov     [esp+2E4h+var_4], eax
xor     eax, eax
push   esi
xor     ebx, ebx
push   [esp+2E4h+var_13B], eax
mov     [esp+2E4h+var_137], eax
mov     [esp+2E4h+var_133], eax
mov     [esp+2E4h+var_12F], eax
mov     [esp+2E4h+var_12B], eax
mov     [esp+2E4h+var_127], eax
mov     [esp+2E4h+var_123], eax
mov     [esp+2E4h+var_11F], eax
lea    eax, [esp+2E4h+var_11B]
push   ebx ; int
push   [esp+2E4h+var_10F]
mov     [esp+2ECh+var_13C], bl
mov     [esp+2E8h+var_137], eax
call   _memset
call   sub_401000
push   offset aEnterPassword ; "Enter Password: "
call   _printf
lea    ecx, [esp+2F0h+var_13C]
push   ecx
push   offset a32 ; "%32[^\n\r]"
call   _scanf
lea    eax, [esp+2F8h+var_13C]
add    esp, 18h
lea    edx, [eax+1]
```

100.00% (9,717) (961,306) 00000551 00401151: _main:loc_401151

The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 59GB

Explore Code Block (4)

The screenshot shows the IDA Pro interface with the following components:

- Functions window:** Lists various subroutines including `sub_401000`, `_main`, and others.
- Assembly view:** Displays assembly instructions for `_main:loc_401151`. The instructions are:

```
mov [esp+2D8h+var_4], eax
xor eax, eax
push ebx
push esi
xor ebx, ebx
push 0EFh ; size_t
mov [esp+2E4h+var_13B], eax
mov [esp+2E4h+var_139], eax
mov [esp+2E4h+var_12F], eax
mov [esp+2E4h+var_12B], eax
mov [esp+2E4h+var_127], eax
mov [esp+2E4h+var_123], eax
mov [esp+2E4h+var_11F], eax
lea eax, [esp+2E4h+var_F7]
push ebx ; int
push eax ; void *
mov [esp+2ECh+var_136], ebx
mov [esp+2ECh+var_F8], bl
call _memset ; _memset(var_F7, 0, 0xEF)
push offset aEnterPassword ; "Enter Password: "
call _printf
lea ecx, [esp+2F0h+var_13C]
push ecx
push offset a32 ; "%32[^\n\r]"
call _scanf
lea eax, [esp+2F8h+var_13C]
add esp, 18h
lea edx, [eax+1]
```
- Annotations:** Red circles highlight the `push 0EFh`, `push eax`, and `call _memset` instructions. A red text annotation `_memset(var_F7, 0, 0xEF)` is placed next to the `call` instruction.
- Graph overview:** Shows a control flow graph with a single block.
- Output window:** Displays the message "The initial autoanalysis has been finished."
- Status bar:** Shows "AU: idle", "Down", and "Disk: 59GB".

Explore Code Block (5)

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

- sub_401000
- _main
- sub_401310
- sub_4013B0
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0
- sub_403000
- sub_403060
- sub_403140
- sub_404160
- sub_4041F0
- sub_4043B0
- sub_404420
- sub_4044A0

```
mov [esp+2D8h+var_4], eax
xor  eax, eax
push ebx
push esi
xor  ebx, ebx
push 0EFh ; size_t
mov  [esp+2E4h+var_13B], eax
mov  [esp+2E4h+var_137], eax
mov  [esp+2E4h+var_133], eax
mov  [esp+2E4h+var_12F], eax
mov  [esp+2E4h+var_12B], eax
mov  [esp+2E4h+var_127], eax
mov  [esp+2E4h+var_123], eax
mov  [esp+2E4h+var_11F], eax
lea  eax, [esp+2E4h+var_F7]
push ebx ; int
push eax ; void *
mov  [esp+2ECh+var_13C], b1
mov  [esp+2ECh+var_F8], b1
call _memset
call sub_401000
push offset _EnterPassword ; "Enter Password: "
call _printf
lea  ecx, [esp+2F0h+var_13C]
push ecx
push offset a32 ; "%32[^\n\r]"
call _scanf
lea  eax, [esp+2F8h+var_13C]
add  esp, 18h
lea  edx, [eax+1]
```

Prints the introductory text

loc_401117:

100.00% (9,717) (961,306) 00000551 00401151: _main:loc_401151

Output window

The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 59GB

Explore Code Block (6)

The screenshot shows the IDA Pro interface with the following assembly code block highlighted by a red oval:

```
mov [esp+2D8h+var_4], eax
xor eax, eax
push ebx
push esi
xor ebx, ebx
push 0EFh ; size_t
mov [esp+2E4h+var_13B], eax
mov [esp+2E4h+var_137], eax
mov [esp+2E4h+var_133], eax
mov [esp+2E4h+var_12F], eax
mov [esp+2E4h+var_12B], eax
mov [esp+2E4h+var_127], eax
mov [esp+2E4h+var_123], eax
mov [esp+2E4h+var_11F], eax
lea eax, [esp+2E4h+var_F7]
push ebx ; int
push eax ; void *
mov [esp+2ECh+var_13C], bl
mov [esp+2ECh+var_138], bl
call _memset
call sub_401000
push offset aEnterPassword ; "Enter Password: "
call _printf
lea ecx, [esp+2F0h+var_13C]
push ecx
push offset a32 ; "%32[^\n\r]"
call _scanf
lea eax, [esp+2F8h+var_13C]
add esp, 18h
mov edx, [eax+1]
```

Prints "Enter Password: " and stores the characters typed into the 33 byte buffer.

The assembly code below the oval shows the function's return and cleanup:

```
loc_401117:
mov eax, [esp+2F8h+var_13C]
add esp, 18h
mov edx, [eax+1]
```

The output window at the bottom displays the message: "The initial autoanalysis has been finished."

Explore Code Block (7)

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

- sub_401000
- sub_401310
- sub_4013B0
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0
- sub_403000
- sub_403060
- sub_403140
- sub_404160
- sub_4041F0
- sub_4043B0
- sub_404420
- sub_4044A0

```
mov [esp+2D8h+var_4], eax
xor eax, eax
push ebx
push esi
xor ebx, ebx
push 0EFh ; size_t
mov [esp+2E4h+var_13B], eax
mov [esp+2E4h+var_137], eax
mov [esp+2E4h+var_133], eax
mov [esp+2E4h+var_12F], eax
mov [esp+2E4h+var_12B], eax
mov [esp+2E4h+var_127], eax
mov [esp+2E4h+var_123], eax
mov [esp+2E4h+var_11F], eax
lea eax, [esp+2E4h+var_F7]
push ebx ; int
push eax ; void *
mov [esp+2ECh+var_13C], b1
mov [esp+2ECh+var_F8], b1
call _memset
call sub_401000
push offset aEnterPassword ; "Enter Password: "
call _printf
lea ecx, [esp+2F0h+var_13C]
push ecx
push offset a32 ; "%32[^\n\r]"
call _scanf
lea eax, [esp+2F8h+var_13C]
add esp, 18h
lea edx, [eax+1]
```

eax = pointer to password
ebx = 0
edx = eax + 1

loc_401117:

100.00% (9,717) (961,306) 00000551 00401151: _main:loc_401151

Output window

The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 59GB

Explore Code Block (8)

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

- sub_401000
- _main
- sub_401310
- sub_4013B0
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0
- sub_403000
- sub_403060
- sub_403140
- sub_404160
- sub_4041F0
- sub_4043B0
- sub_404420
- sub_4044A0

loc_401117:

```
mov    cl, [eax]
inc    eax
cmp    cl, bl
jnz   short loc_401117
```

loc_401151:

```
sub    eax, edx
cmp    eax, ebx
jnz   short loc_401151
```

```
push offset aAccessDenied ; "ACCESS DENIED\n"
call  sub_4067FB
add   eax, 40h
ush   eax           ; FILE *
call  _fprintf
add   esp, 8
mov   esi, 0FFFFFFFh
mov   ebx
mov   ecx, [esp+2D8h+var_4]
mov   ecx, esp
call  @_security_check_cookie@4 ; __security_check_cookie(x)
mov   esp, ebp
pop   ebp
ret
```

```
loc_401151:           ; size_t
push  eax
lea   edx, [esp+2E4h+var_13C]
push  edx           ; void *
lea   esi, [esp+2E8h+var_2D8]
mov   [esp+2E8h+var_2C8], 6A09E667h
mov   [esp+2E8h+var_2C8], 0BB67AE85h
mov   [esp+2E8h+var_2C4], 3C6EF372h
mov   [esp+2E8h+var_2C0], 0A54FF53Ah
mov   [esp+2E8h+var_2BC], 510E527Fh
mov   [esp+2E8h+var_2B8], 9B05688Ch
mov   [esp+2E8h+var_2B4], 1F83D9A8h
mov   [esp+2E8h+var_2B0], 5BE0CD19h
mov   [esp+2E8h+var_2D8], ebx
mov   [esp+2E8h+var_2D4], ebx
```

100.00% (9,1167) (937,486) 000004C6 004010C6: _main+56

Output window

The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 59GB

Explore Code Block (9)

The screenshot shows the IDA Pro interface with the following components:

- Functions window:** Lists functions from `sub_401000` to `sub_404440`.
- Graph overview:** Shows a control flow graph with a loop.
- Assembly view:** Displays assembly code for `loc_401117`, `loc_401151`, and a function starting with `offset aAccessDenied`.

Code Block 9 (loc_401117):

```
loc_401117:
mov    cl, [eax]
inc    eax
cmp    cl, b1
jnz   short loc_401117
```

Code Block 10 (loc_401151):

```
loc_401151:                ; size_t
push  eax
lea  edx, [esp+2E4h+var_13C]
push  edx                  ; void *
lea  esi, [esp+2E8h+var_2D8]
mov   [esp+2E8h+var_2C0], 6A09E667h
mov   [esp+2E8h+var_2C8], 0BB67AE85h
mov   [esp+2E8h+var_2C4], 3C6EF372h
mov   [esp+2E8h+var_2C0], 0A54FF53Ah
mov   [esp+2E8h+var_2BC], 510E527Fh
mov   [esp+2E8h+var_2B8], 9B05688Ch
mov   [esp+2E8h+var_2B4], 1F83D9A8h
mov   [esp+2E8h+var_2B0], 5BE0CD19h
mov   [esp+2E8h+var_2D8], ebx
mov   [esp+2E8h+var_2D4], ebx
```

Code Block 11 (Function):

```
push  offset aAccessDenied ; "ACCESS DENIED\n"
call  sub_4067FB
add   eax, 40h
ush   eax                ; FILE *
call  _fprintf
add   esp, 8
mov   eax, 0FFFFFFFFh
mov   esi, eax
mov   ebx, eax
mov   ecx, [esp+2D8h+var_4]
xor   ecx, esp
call  @_security_check_cookie@4 ; __security_check_cookie(x)
mov   esp, ebp
pop   ebp
ret
```

Annotation: A red circle highlights the loop in `loc_401117`. A red arrow points from the text "Compute the password length" to this loop.

Explore Code Block (10)

The screenshot shows the IDA Pro interface with the following assembly code blocks:

```
loc_401117:
mov    cl, [eax]
inc    eax
cmp    cl, bl
jnz    short loc_401117

sub    eax, edx
cmp    eax, ebx
jnz    short loc_401151

push  offset aAccessDenied ; "ACCESS DENIED\n"
call  sub_4067FB
add   eax, 40h
ush   eax           ; FILE *
call  _fprintf
add   esp, 8
eax, 0FFFFFFFFh

loc_401151:
; size_t
push  eax
lea   edx, [esp+2E4h+var_13C]
push  edx           ; void *
lea   esi, [esp+2E8h+var_2D8]
mov   [esp+2E8h+var_2C8], 6A09E667h
mov   [esp+2E8h+var_2C8], 0BB67AE85h
mov   [esp+2E8h+var_2C4], 3C6EF372h
mov   [esp+2E8h+var_2C0], 0A54FF53Ah
mov   [esp+2E8h+var_2BC], 510E527Fh
mov   [esp+2E8h+var_2B8], 9B05688Ch
mov   [esp+2E8h+var_2B4], 1F83D9A8h
mov   [esp+2E8h+var_2B0], 5BE0CD19h
mov   [esp+2E8h+var_2D8], ebx
mov   [esp+2E8h+var_2D4], ebx
```

Annotations in the image include:

- A red circle around the `push offset aAccessDenied ; "ACCESS DENIED\n"` instruction.
- A red circle around the `sub eax, edx` and `cmp eax, ebx` instructions, with a red arrow pointing to the `jnz short loc_401151` instruction.
- A text box on the right stating: "Print 'ACCESS DENIED\n' if the password has a length of 0".

The output window at the bottom displays the message: "The initial autoanalysis has been finished."

Secure Hash Algorithm

- Hash values can be used to verify the integrity of the data without providing any means to derive the original data. The algorithm is irreversible (SHA-2 is implemented in some widely used security applications and protocols: TLS, SSL, PGP, SSH, S/MIME, Bitcoin and IPsec)
- The first call computes the SHA256 hash of the password
 - You can figure out that it is a SHA256 by Googling for some of the large constants that you will find in the code (0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a, 0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19)

Secure Hash Algorithm (2)

The screenshot displays the IDA Pro interface with the following components:

- Functions window:** Lists various subroutines including `sub_401000`, `_main`, `sub_401310`, `sub_4013B0`, `sub_401450`, `sub_4014E0`, `sub_401620`, `sub_401670`, `sub_401D30`, `sub_401DC0`, `sub_401FE0`, `sub_4023A0`, `sub_4028F0`, `sub_4029E0`, `sub_403000`, `sub_403060`, `sub_403140`, `sub_404160`, and `sub_4041F0`.
- Assembly code (loc_401151):**

```
loc_401151:          ; size_t
push    eax
lea     edx, [esp+2E4h+var_13C]
push    edx          ; void *
lea     esi, [esp+2E8h+var_2D8]
mov     [esp+2E8h+var_2CC], 6A09E667h
mov     [esp+2E8h+var_2C8], 0BB67AE85h
mov     [esp+2E8h+var_2C4], 3C6EF372h
mov     [esp+2E8h+var_2C0], 0A54FF53Ah
mov     [esp+2E8h+var_2BC], 510E527Fh
mov     [esp+2E8h+var_2B8], 9B05688Ch
mov     [esp+2E8h+var_2B4], 1F83D9ABh
mov     [esp+2E8h+var_2B0], 5BE0CD19h
mov     [esp+2E8h+var_2D8], ebx
mov     [esp+2E8h+var_2D4], ebx
mov     [esp+2E8h+var_2D0], ebx
call    sub_401D30
lea     eax, [esp+2E8h+var_118]
push    eax
mov     ebx, esi
call    sub_401DC0
mov     c1, byte ptr [esp+2ECh+var_118+1]
xor     c1, byte_416EB0
mov     d1, byte ptr [esp+2ECh+var_118]
xor     d1, byte_416EB1
add     esp, 0Ch
xor     eax, eax
mov     [esp+2E0h+var_14C], c1
mov     [esp+2E0h+var_148], d1
lea     ecx, [eax+7]
mov     edi, edi
```
- Output window:**

```
The initial autoanalysis has been finished.
Command "MakeUnknown" failed
401D30: can't rename byte as 'sub_401D30' because the name has a reserved prefix.
401DC0: can't rename byte as 'sub_sdf' because the name has a reserved prefix.
```

Secure Hash Algorithm (3)

loc_401151: ; size_t
push eax
lea edx, [esp+2E4h+var_13C]
push edx ; void *
lea esi, [esp+2E8h+var_2D8]
mov [esp+2E8h+var_2CC], 6671067h
mov [esp+2E8h+var_2C8], 8BB67AE85h
mov [esp+2E8h+var_2C4], 3C6EF372h
mov [esp+2E8h+var_2C0], 0A54FF53Ah
mov [esp+2E8h+var_2BC], 510E527Fh
mov [esp+2E8h+var_2B8], 9B05688Ch
mov [esp+2E8h+var_2B4], 1F83D9ABh
mov [esp+2E8h+var_2B0], 5BE0CD19h
mov [esp+2E8h+var_2D8], ebx
mov [esp+2E8h+var_2D4], ebx
mov [esp+2E8h+var_2D0], ebx
call sub_401D30
lea eax, [esp+2E8h+var_118]
push eax
mov ebx, esi
call sub_401DC0
mov c1, byte ptr [esp+2ECh+var_118+1]
xor c1, byte_416EB0
mov d1, byte ptr [esp+2ECh+var_118]
xor d1, byte_416EB1
add esp, 0Ch
xor eax, eax
mov [esp+2E0h+var_14C], c1
mov [esp+2E0h+var_14B], d1
lea ecx, [eax+7]
mov edi, edi

Putting pointer to password buffer and password length on the stack

100.00% (490,1456) | (756,110) | 00000552 00401152: _main+E2

Output window
The initial autoanalysis has been finished.
Command "MakeUnknown" failed
401D30: can't rename byte as 'sub_401D30' because the name has a reserved prefix.
401DC0: can't rename byte as 'sub_sdf' because the name has a reserved prefix.
IDC
Add a breakpoint to the current address

Secure Hash Algorithm (4)

The screenshot displays the IDA Pro interface for a file named 'CodeBreaker.exe'. The assembly view shows the following code for function `loc_401151`:

```
loc_401151:          ; size_t
push    eax
lea    edx, [esp+2E4h+var_13C]
push    edx          ; word *
lea    esi, [esp+2E8h+var_2D8]
mov    [esp+2E8h+var_2C], 6A09E667h
mov    [esp+2E8h+var_28], 8BB67AE85h
mov    [esp+2E8h+var_24], 3C6EF372h
mov    [esp+2E8h+var_20], 0A54FF53Ah
mov    [esp+2E8h+var_1C], 510E527Fh
mov    [esp+2E8h+var_18], 9B05688Ch
mov    [esp+2E8h+var_14], 1F83D9ABh
mov    [esp+2E8h+var_10], 5BE0CD19h
mov    [esp+2E8h+var_0C], ebx
mov    [esp+2E8h+var_08], ebx
mov    [esp+2E8h+var_04], ebx
call   sub_401030
lea    eax, [esp+2E8h+var_118]
push    eax
mov    ebx, esi
call   sub_401DC0
mov    c1, byte ptr [esp+2ECh+var_118+1]
xor    c1, byte_416EB0
mov    d1, byte ptr [esp+2ECh+var_118]
xor    d1, byte_416EB1
add    esp, 0Ch
xor    eax, eax
mov    [esp+2E0h+var_14C], c1
mov    [esp+2E0h+var_148], d1
lea    ecx, [eax+7]
mov    edi, edi
```

A red circle highlights the constants: `6A09E667h`, `8BB67AE85h`, `3C6EF372h`, `0A54FF53Ah`, `510E527Fh`, `9B05688Ch`, `1F83D9ABh`, and `5BE0CD19h`.

Output window messages:

```
The initial autoanalysis has been finished.
Command "MakeUnknown" failed
401030: can't rename byte as 'sub_401030' because the name has a reserved prefix.
401DC0: can't rename byte as 'sub_sdf' because the name has a reserved prefix.
```

Constants associated with
SHA256 hashing

Secure Hash Algorithm (5)

The screenshot shows the IDA Pro interface with the following components:

- Functions window:** Lists functions including `sub_401000`, `_main`, and `sub_401D30`.
- Graph overview:** Shows a control flow graph with nodes for `sub_401D30` and `loc_401D44`.
- Assembly view:**

```
; Attributes: bp-based frame
; int __cdecl sub_401D30(void *, size_t)
sub_401D30 proc near

arg_0= dword ptr 8
arg_4= dword ptr 0Ch

push    ebp
mov     ebp, esp
push    ebx
push    edi
mov     edi, [ebp+arg_4]
lea    ebx, [esi+2Ch]
test   edi, edi
jz     short loc_401D86

jmp     short loc_401D44

loc_401D44:
mov     eax, 40h
sub     eax, [esi]
cmp     edi, eax
```
- Output window:** Contains the message: "Propagating type information... Function argument information has been propagated. The initial autoanalysis has been finished."
- Status bar:** Shows "AU: idle", "Down", and "Disk: 59GB".

AES Key Derivation Routine

- AES is a symmetric-key algorithm; same key is used to encrypt and decrypt the data.
- The AES key is being derived from the hash of the password
- Lets take a closer look at what is being done to the password hash

AES Key Derivation Routine (2)

The screenshot shows the IDA Pro interface with the assembly window displaying the following code:

```
loc_401151:          ; size_t
push    eax
lea    edx, [esp+2E4h+var_13C]
push    edx          ; void *
lea    esi, [esp+2E8h+var_2D8]
mov    [esp+2E8h+var_2CC], 6A09E667h
mov    [esp+2E8h+var_2C8], 8BB67AE85h
mov    [esp+2E8h+var_2C4], 3C6EF372h
mov    [esp+2E8h+var_2C0], 0A54FF53Ah
mov    [esp+2E8h+var_2BC], 510E527Fh
mov    [esp+2E8h+var_2B8], 9B05688Ch
mov    [esp+2E8h+var_2B4], 1F8D9ABh
mov    [esp+2E8h+var_2B0], 5BE0CD19h
mov    [esp+2E8h+var_2D8], ebx
mov    [esp+2E8h+var_2D4], ebx
mov    [esp+2E8h+var_2D0], ebx
call   sub_401030
lea    eax, [esp+2E8h+var_118]
push    eax
mov    ebx, esi
call   sub_401DC0
mov    c1, byte ptr [esp+2ECh+var_118+1]
xor    c1, byte_416EB0
mov    d1, byte ptr [esp+2ECh+var_118]
xor    d1, byte_416EB1
add    esp, 0Ch
xor    eax, eax
mov    [esp+2E0h+var_14C], c1
mov    [esp+2E0h+var_14B], d1
lea    ecx, [eax+7]
mov    edi, edi
```

Annotations on the right side of the image:

- `var_118[1] xor byte_416EB0`
- `var_118[0] xor byte_416EB1`
- both values pushed onto the stack starting at `var_14B`
- `eax = 0`
- `ecx = 7`

The assembly window also shows the following output window messages:

```
the initial autoanalysis has been finished.
Command "MakeUnknown" failed
401D30: can't rename byte as 'sub_401D30' because the name has a reserved prefix.
401DC0: can't rename byte as 'sub_sdf' because the name has a reserved prefix.
```

AES Key Derivation Routine (3)

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name

- sub_401000
- _main
- sub_401310
- sub_4013B0
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0
- sub_403000
- sub_403060
- sub_403140
- sub_404160
- sub_4041F0
- sub_404300

Line 1 of 235

Graph overview

```
call sub_401DC0
mov c1, byte ptr [esp+2ECh+var_118+1]
xor c1, byte_416EB0
mov d1, byte ptr [esp+2ECh+var_118]
xor d1, byte_416EB1
add esp, 0Ch
xor eax, eax
mov [esp+2E0h+var_14C], c1
mov [esp+2E0h+var_14B], d1
lea ecx, [eax+7]
mov edi, edi
```

```
loc_4011F0:
movzx edx, [esp+eax+2E0h+var_14B]
mov [esp+eax+2E0h+var_14A], d1
movzx edx, [esp+eax+2E0h+var_14C]
mov [esp+eax+2E0h+var_149], d1
movzx edx, byte_416EB2[eax]
xor [esp+eax+2E0h+var_14A], d1
movzx edx, byte_416EB3[eax]
xor [esp+eax+2E0h+var_149], d1
add eax, 2
dec ecx
jnz short loc_4011F0
```

```
lea edx, [esp+2E0h+var_14C]
lea eax, [esp+2E0h+var_268]
```

100.00% (364,1783) (922,216) 00000615 00401215: _main+1A5

Output window

Using PEAPI signature: microsoft:visualc:2.71:msvc_runtime

Propagating type information...

Function argument information has been propagated

The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 59GB

AES Key Derivation Routine (4)

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name

- sub_401000
- _main
- sub_401310
- sub_4013B0
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0
- sub_403000
- sub_403060
- sub_403140
- sub_404160
- sub_4041F0
- sub_404200

Line 1 of 235

Graph overview

```
call sub_401DC0
mov cl, byte ptr [esp+2ECh+var_118+1]
xor cl, byte_416EB0
mov d1, byte ptr [esp+2ECh+var_118]
xor d1, byte_416EB1
add esp, 0Ch
xor eax, eax
mov [esp+2E0h+var_14C], cl
mov [esp+2E0h+var_148], d1
lea ecx, [eax+7]
edi, edi

loc_4011F0:
movzx edx, [esp+eax+2E0h+var_14B]
mov [esp+eax+2E0h+var_14A], d1
movzx edx, [esp+eax+2E0h+var_14C]
mov [esp+eax+2E0h+var_149], d1
movzx edx, byte_416EB2[eax]
xor [esp+eax+2E0h+var_14A], d1
movzx edx, byte_416EB3[eax]
xor [esp+eax+2E0h+var_149], d1
add eax, 2
dec ecx
jnz short loc_4011F0

lea edx, [esp+2E0h+var_14C]
lea eax, [esp+2E0h+var_268]
```

For 7 iterations, computes:

$$k[i + 2] = k[i] \text{ xor } \text{byte}[i + 2]$$
$$k[i + 3] = k[i + 1] \text{ xor } \text{byte}[i + 3]$$

where $k = \text{var_118}$ and $\text{byte} = \text{byte_416EB0}$

Output window

Using PEAPI signature: microsoft:visualc:6.0.9596:2:1176c:func:kw

Propagating type information...

Function argument information has been propagated

The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 59GB

AES Key Derivation

- The 16-byte key is derived as follows:
- H_0, H_1, \dots, H_{31} represents the 32-bytes of the SHA256 hash
- K_0, K_1, \dots, K_{15} are the 16-bytes of key
- R_0, R_1, \dots, R_{15} are the 16-bytes of fixed random data that is in the program

(0x23,0x8e,0x60,0xe1,0xd2,0x96,0x38,0xc7,0xa5,0xc0,0x22,0x74,0x4f,0x31,0x5b,0xcd)

AES Key Derivation

- $K_0 = H_1 \text{ xor } R_0$
- $K_1 = H_0 \text{ xor } R_1$
- $K_2 = K_1 \text{ xor } R_2$
- $K_3 = K_0 \text{ xor } R_3$
- $K_4 = K_3 \text{ xor } R_4$
- $K_5 = K_2 \text{ xor } R_5$
- :
- $K_{15} = K_{12} \text{ xor } R_{15}$
- Looking at the above equations, do you notice any problems?
 - How many bytes of the password hash are ultimately used in the derivation of the key?
 - Given this, how many possible AES keys could be derived from this function?
 - Is that something that you can efficiently compute and brute-force?

Keyed-Hash Message Authentication Code

- Following the code you will see that the AES key is passed to a function that computes the Keyed-Hash Message Authentication Code (HMAC) of the cipher text stored in the program.
- If the computed HMAC-SHA256 matches the expected HMAC

(0xbc,0xb0,0x78,0x41,0xaa,0xdd,0x06,0x68,0xdd,0xa4,0x74,0x06,0xd7,0x1a,0x2b,0xd2,0x9b,0xae,0xdb,0xd,0xf9,0xce,0xf4,0xa7,0xf9,0x51,0x6c,0x99,0xfc,0xb6,0x95,0xf8)

- Then, the AES key is used to decrypt the cipher text
- Else, "ACCESS DENIED" message is displayed

Keyed-Hash Message Authentication Code (2)

The screenshot displays the IDA Pro interface for the file `C:\challenge\CodeBreaker.exe`. The main window shows assembly code with a control flow graph. The assembly code is as follows:

```
lea    edx, [esp+2E0h+var_14C]
lea    eax, [esp+2E0h+var_268]
call   sub_401310
lea    eax, [esp+2E0h+var_268]
call   sub_401450
lea    eax, [esp+2E0h+var_118]
push   eax
lea    eax, [esp+2E4h+var_268]
call   sub_4014E0
add    esp, 4
mov    eax, 20h
xor    ecx, ecx
jmp    short loc_401270

loc_401270:
mov    edx, [esp+ecx+2E0h+var_118]
cmp    edx, dword_416FB0[ecx]
jnz    short loc_4012D5

sub    eax, 4
add    ecx, 4
cmp    eax, 4
jnb   short loc_401270

loc_4012D5:
; "ACCESS DENIED\n"
push  offset aAccessDenied
call  sub_4067FB
add   eax, 40h
push  eax
; FILE *
```

The control flow graph shows a loop structure. The `loc_401270` block branches to `loc_4012D5` if the comparison fails (`jnz`). `loc_4012D5` contains a call to `sub_4067FB` and a `push` instruction. The `loc_401270` block also branches to `loc_4012D5` if the comparison is not below (`jnb`).

The output window at the bottom shows the following messages:

```
Using PE-1 signature: Microsoft Windows [Version 6.0.6002.1.81473.0]
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
```

Keyed-Hash Message Authentication Code (3)

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name

- sub_401000
- _main
- sub_401310
- sub_401380
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0
- sub_403000
- sub_403060
- sub_403140
- sub_404160
- sub_4041F0
- sub_404300

Line 1 of 235

Graph overview

```
lea     edx, [esp+2E0h+var_14C]
lea     eax, [esp+2E0h+var_268]
call    sub_401310
lea     eax, [esp+2E0h+var_268]
call    sub_401450
lea     eax, [esp+2E0h+var_118]
push   eax
lea     eax, [esp+2E4h+var_268]
call    sub_4014E0
add     esp, 4
mov     eax, 20h
xor     ecx, ecx
jmp     short loc_401270
```

loc_401270:

```
mov     edx, [esp+ecx+2E0h+var_118]
cmp     edx, dword_416FB0[ecx]
jnz    short loc_4012D5
```

loc_4012D5: ; "ACCESS DENIED\n"

```
push   offset aAccessDenied
call   sub_4067FB
add     eax, 40h
push   eax
```

100.00% (340,2218) (976,403) 00000615 00401215: _main+1A5

Output window

Using PEAPI signature: Microsoft Visual Studio 2.0 (x64) Function
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 59GB

Compares var_118[32] with byte_416FB0[32] and prints "ACCESS DENIED\n" if they are not

Keyed-Hash Message Authentication Code (4)

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

- sub_401000
- _main
- sub_401310
- sub_401380
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0
- sub_403000
- sub_403060
- sub_403140
- sub_404160
- sub_4041F0

Line 1 of 235

Graph overview

```
loc_401270:
mov     edx, [esp+ecx+2E0h+var_118]
cmp     edx, dword_416FB0[ecx]
jnz     short loc_4012D5

sub     eax, 4
add     ecx, 4
cmp     eax, 4
jnb     short loc_401270

loc_4012D5:
; "ACCESS DENIED\n"
push   offset aAccessDenied
call   sub_4067FB
add    eax, 40h
push   eax
; FILE *
call   _fprintf
mov    ecx, [esp+2E8h+var_4]
add    esp, 8
pop    esi
pop    ebx
xor    ecx, esp
mov    eax, 0FFFFFFDh
call   @_security_check_cookie@4 ; __security_check_cookie(x)
mov    esp, ebp
pop    ebp
retn
_main endp
```

100.00% (196,2454) (36,82) 00000655:00401255: _main+1E5

Output window

Using PEAPI signature for Microsoft Visual C++ 2.0 linked function
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.

AES Decryption

IDA - C:\challenge\CodeBreaker.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

- sub_401000
- _main
- sub_401310
- sub_4013B0
- sub_401450
- sub_4014E0
- sub_401620
- sub_401670
- sub_401D30
- sub_401DC0
- sub_401FE0
- sub_4023A0
- sub_4028F0
- sub_4029E0
- sub_403000
- sub_403060
- sub_403140
- sub_404160
- sub_4041F0
- sub_404300

Line 1 of 235

Graph overview

```
lea    eax, [esp+2E0h+var_14C]
lea    esi, [esp+2E0h+var_268]
call   sub_401FE0
lea    eax, [esp+2E0h+var_F8]
push   eax
mov    ecx, esi
push   ecx
call   sub_4028F0
lea    edx, [esp+2E0h+var_F8]
push   edx
push   offset aS          ; "%s\n"
call   _printf
add    esp, 10h
xor    eax, eax
pop    esi
pop    ebx
mov    ecx, [esp+2D8h+var_4]
xor    ecx, esp
call   @_security_check_cookie@4 ; __security_check_cookie(x)
mov    esp, ebp
pop    ebp
retn
```

100.00% (196,2919) (68,6) 00000655 00401255: _main+1E5

Output window

Using PEAPI signature: microsoft.windows.common-core-1.0.0-2-1130c012

Propagating type information...

Function argument information has been propagated

The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 59GB

Brute Force Attack

- Leverage the fact that you know the expected HMAC and you know how to derive putative key
 - (0xbc,0xb0,0x78,0x41,0xaa,0xd0,0x06,0x68,0xdd,0xa4,0x74,0x06,0xd7,0x1a,0x2b,0xd2,0x9b,0xae,0xdb,0xd0,0xf9,0xce,0xf4,0xa7,0xf9,0x51,0x6c,0x99,0xfc,0xb6,0x95,0xf8)
- Write a simple program to brute force the key and decrypt the ciphertext
 - The CyaSSL library is easy to use and contains all the functions that you need (it was used to write the challenge problem)

Brute Force Code

```
printf("Searching for a solution...\n");
for (i=0; i < (1<<16); i++)
{
    memcpy(hashGuess, (uint8_t*)&i, 2);
    // Derive the AES key to decrypt secret code
    retCode = deriveAESKey(hashGuess, (uint8_t*)AESKey);
    // Compute HMAC of ciphertext
    computeHMAC(ciphertext, sizeof(ciphertext), AESKey, AES_KEY_SIZE, HMACdigest);
    // Compare computed HMAC with expected value
    if (memcmp(HMACdigest, HMAC_cipher, SHA256_DIGEST_SIZE) == 0)
    {
        fprintf(stderr, "FOUND SOLUTION!\n");
        fprintf(stderr, "SHA256 Hash of Password -> Byte 1: 0x%x\tByte 2: 0x%x\n", hashGuess[0], hashGuess[1]);
        fprintf(stderr, "AES Key: ");
        printHexString(AESKey, AES_KEY_SIZE);
        printf("\n");
        // Decrypt the challenge ciphertext
        decrypt(AESKey, plain, sizeof(plain));
        printf("%s\n", plain);
    }
}
```

Ciphertext

- `uint8_t ciphertext[] = {0x34,0x3b,0x71,0x62,0xba,0x9f,0x55,0xb8,0xd9,0x88,0x71,0x98,0x24,0x41,0x3e,0x4d,0xa4,0xbd,0x1a,0xe1,0x2c,0xae,0x2c,0xff,0x1d,0x4e,0x1a,0x9e,0x94,0x8a,0x4d,0x07,0x71,0x5d,0xc6,0x1f,0xef,0x91,0x09,0x67,0xda,0xfa,0x37,0x96,0x11,0xe1,0x67,0xd6,0x3e,0xa1,0x5e,0x58,0x0b,0x81,0xdd,0xb2,0xaf,0x5d,0xde,0xde,0x9d,0x82,0xb3,0x72,0x36,0x86,0xa6,0x72,0xea,0x3e,0x5a,0xa0,0x21,0x4e,0x94,0xbf,0x51,0x12,0xbe,0xfc,0xb6,0x07,0x3d,0x51,0x36,0xcf,0x76,0x93,0xab,0xc6,0x6c,0x7b,0x5f,0xc8,0x16,0xa2,0x11,0xc0,0xe6,0x87,0x9e,0xab,0x40,0x56,0xa4,0xb7,0xa5,0x20,0x44,0xbf,0xb0,0xb7,0x5b,0x43,0x4a,0x02,0x19,0x09,0x1d,0xb2,0x30,0xbb,0x15,0xce,0x1c,0x97,0xd8,0x77,0xbc,0x42,0x87,0x14,0x93,0x85,0xd2,0x0a,0x7d,0xc4,0x44,0x0e,0x82,0x35,0x3b,0xc4,0x40,0x78,0x7a,0x59,0xa1,0x59,0x18,0x09,0x22,0x17,0x68,0xc0,0xfc,0x7a,0x5f,0x67,0x5b,0x2a,0xb3,0xfc,0x53,0xbc,0xe0,0x92,0xff,0x0d,0x84,0x74,0x31,0x1f,0xf5,0x16,0x4f,0x17,0x50,0x8d,0x95,0x51,0x06,0xf7,0xbc,0xda,0x15,0x05,0x76,0xb5,0x10,0x78,0xa4,0xa1,0xf1,0x45,0xf1,0x6e,0x78,0x2c,0x3a,0x01,0x4e,0x82,0x68,0x4f,0xe8,0x12,0x69,0xdd,0x00,0x77,0x17,0xec,0x95,0x76,0xbc,0x8c,0x43,0xc5,0x99,0x53,0xba,0x86,0x4c,0x6b,0x46,0x4a,0x35,0x82,0xe1,0x10,0xee,0x2a,0x73,0x4d,0x80,0x55,0xdc,0xbc};`

Brute Force Solution

```
Searching for a solution...  
FOUND SOLUTION!  
SHA256 Hash of Password -> Byte 1: 0x12 Byte 2: 0x2  
AES Key: 0x21 0x9c 0xfc 0xc0 0x12 0x6a 0x52 0xd5 0x70 0x92 0xb0 0x4 0x4b 0x81 0x  
da 0x86  
  
Congratulations! You have successfully decrypted the challenge cipher.  
  
Please send an email (use your *.edu address) with a writeup of your solution an  
d the string bd461a21f932a4130a6f670d to the following address: Senior_Project@n  
sa.gov
```

Plaintext

- Congratulations! You have successfully decrypted the challenge cipher.
- Please send an email (use your *.edu address) with a writeup of your solution and the string `bd461a21f932a4130a6f670d` to the following address: _____



Questions

