

# PicoCTF

## JAVA SCRIPT KIDDIE

### HURRY UP! WAIT!

The only thing in the description of this CTF is the download link to a file.

When you use the 'file' command to determine the type of file it is, we find that it is an ELF type.

```
rigved@Rigved: ~/Downloads
rigved@Rigved: $ vln picoCTF
rigved@Rigved: $ /home/rigved/Downloads
bash: /home/rigved/Downloads: Is a directory
rigved@Rigved: $ cd /home/rigved/Downloads
rigved@Rigved: ~/Downloads $ ls -l
total 242736
-rwxr-xr-x 1 rigved rigved 727 Jan 1 1970 app.py
-rw-rw-r-- 1 rigved rigved 81564 Feb 13 12:01 'Crash course-Time Table.pdf'
-rwxr-xr-x 1 rigved rigved 405 Jan 1 1970 Dockerfile
-rw-rw-r-- 1 rigved rigved 5516 Feb 14 23:14 'main(1).wav'
-rw-rw-r-- 1 rigved rigved 5516 Feb 14 17:55 main.wav
-rw-rw-r-- 1 rigved rigved 139680406 Feb 12 11:51 microsoft-edge-stable_110.0.1587.41-1_amd64.deb
-rw-rw-r-- 1 rigved rigved 589 Feb 9 18:29 new_caesar.py
-rw-rw-r-- 1 rigved rigved 10240 Feb 13 16:48 notepad.tar
-rw-rw-r-- 1 rigved rigved 197171 Feb 14 23:29 scrambled1.png
-rw-rw-r-- 1 rigved rigved 197173 Feb 14 23:29 scrambled2.png
-rw-rw-r-- 1 rigved rigved 57507 Feb 13 12:08 'Screenshot 2023-02-13 at 12-08-19 Factory Login.png'
-rw-rw-r-- 1 rigved rigved 1558808 Feb 13 17:55 shark1.pcapng
-rw-rw-r-- 1 rigved rigved 1984519 Feb 11 21:10 source
drwxr-xr-x 2 rigved rigved 4096 Jan 1 1970 static
-rw-rw-r-- 1 rigved rigved 18416 Feb 19 14:53 svchost.exe
drwxr-xr-x 3 rigved rigved 4096 Jan 1 1970 templates
-rw-rw-r-- 1 rigved rigved 81954 Feb 13 11:48 'TG_Form Rigved Waradpande.docx'
-rw-rw-r-- 1 rigved rigved 104624068 Feb 7 20:48 tor-browser-linux64-12.0.2_ALL.tar.xz
rigved@Rigved: ~/Downloads $ file svchost.exe
svchost.exe: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=c083b0f6ecae1517082fb6ed0cd9e3f295ec2cc, str
ipped
rigved@Rigved: ~/Downloads $
```

ELF here, stands for Executable and Linkable Format of a file, which is usually the output of a compiler.

LSB here stands for Least Significant Byte, which implies that the file is Little-Endian. This is determined by the argument 'file' from the sixth byte of the header:

>5 byte 1 LSB (source in references)

## References:

For ELF header:

<https://github.com/file/file/blob/4264364d4a46d632ceb095e8cef56339f592931d/magic/Magdir/elf#L305>

[https://en.wikipedia.org/wiki/Executable\\_and\\_Linkable\\_Format#File\\_header](https://en.wikipedia.org/wiki/Executable_and_Linkable_Format#File_header)

For ELF:

<https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/#what-is-an-elf-file>

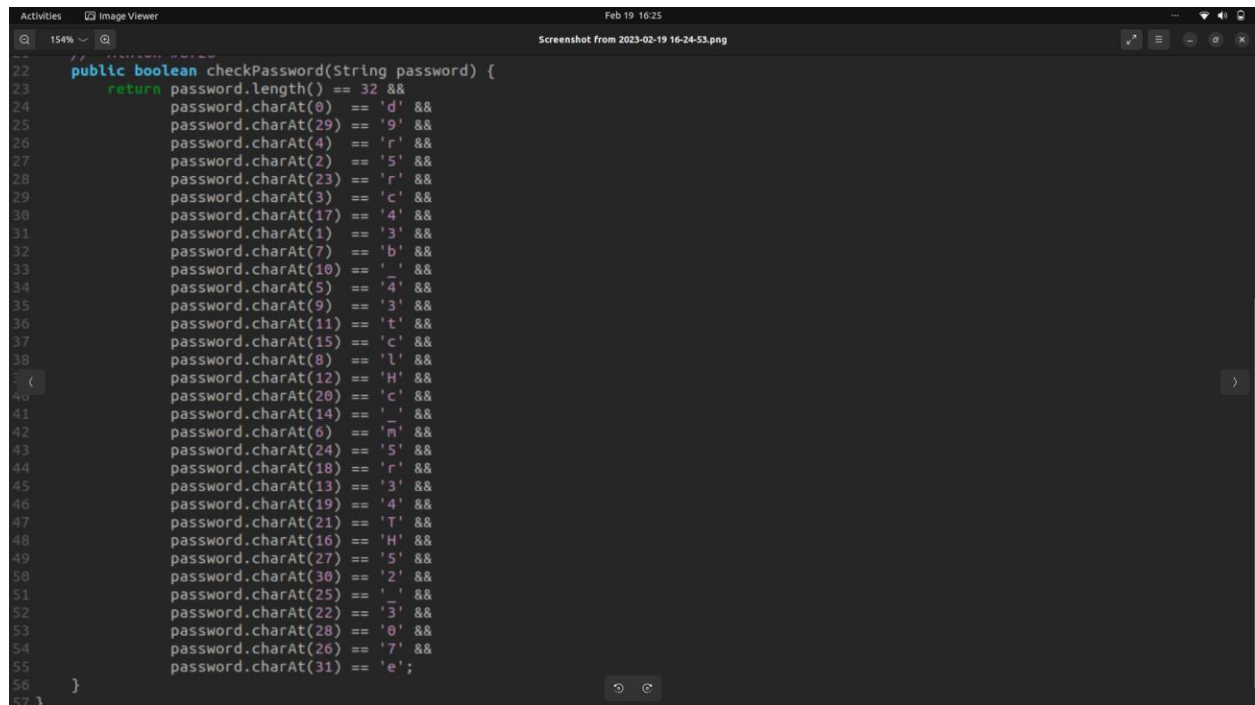
VAULT DOOR-1

```
Activities Text Editor Feb 19 16:23
VaultDoor1.java
~/Downloads
Save

1 import java.util.*;
2
3 class VaultDoor1 {
4     public static void main(String args[]) {
5         VaultDoor1 vaultDoor = new VaultDoor1();
6         Scanner scanner = new Scanner(System.in);
7         System.out.print("Enter vault password: ");
8         String userInput = scanner.next();
9         String input = userInput.substring("picoCTF".length(), userInput.length()-1);
10        if (vaultDoor.checkPassword(input)) {
11            System.out.println("Access granted.");
12        } else {
13            System.out.println("Access denied!");
14        }
15    }
16
17    // I came up with a more secure way to check the password without putting
18    // the password itself in the source code. I think this is going to be
19    // UNHACKABLE!! I hope Dr. Evil agrees...
20    //
21    // -Minion #0720
22    public boolean checkPassword(String password) {
23        return password.length() == 32 &&
24            password.charAt(0) == 'd' &&
25            password.charAt(29) == '9' &&
26            password.charAt(4) == 'r' &&
27            password.charAt(2) == '5' &&
28            password.charAt(23) == 'r' &&
29            password.charAt(3) == 'c' &&
30            password.charAt(17) == '4' &&
31            password.charAt(1) == '3' &&
32            password.charAt(7) == 'b' &&
33            password.charAt(10) == '-' &&
34            password.charAt(5) == 'q' &&
35            password.charAt(9) == '3' &&
36            password.charAt(11) == 't' &&
37            password.charAt(15) == 'c' &&
38            password.charAt(6) == 'l' &&
39            password.charAt(12) == 'H' &&
40            password.charAt(20) == 'c' &&
41            password.charAt(14) == '-' &&
42            password.charAt(6) == 'n' &&
43            password.charAt(24) == '5' &&
44            password.charAt(18) == 'r' &&
45            password.charAt(13) == '3' &&
46            password.charAt(19) == '4' &&
47            password.charAt(21) == '7' &&
48            password.charAt(16) == 'H' &&
49            password.charAt(27) == '5' &&
50            password.charAt(30) == '2' &&
51            password.charAt(25) == 'i' &&
52            password.charAt(22) == '3' &&
53            password.charAt(28) == '0' &&
54            password.charAt(26) == '7' &&
```

As can be seen upon downloading the source code, this program makes use of the `charAt` function, which returns the index position of any specified character in an array.

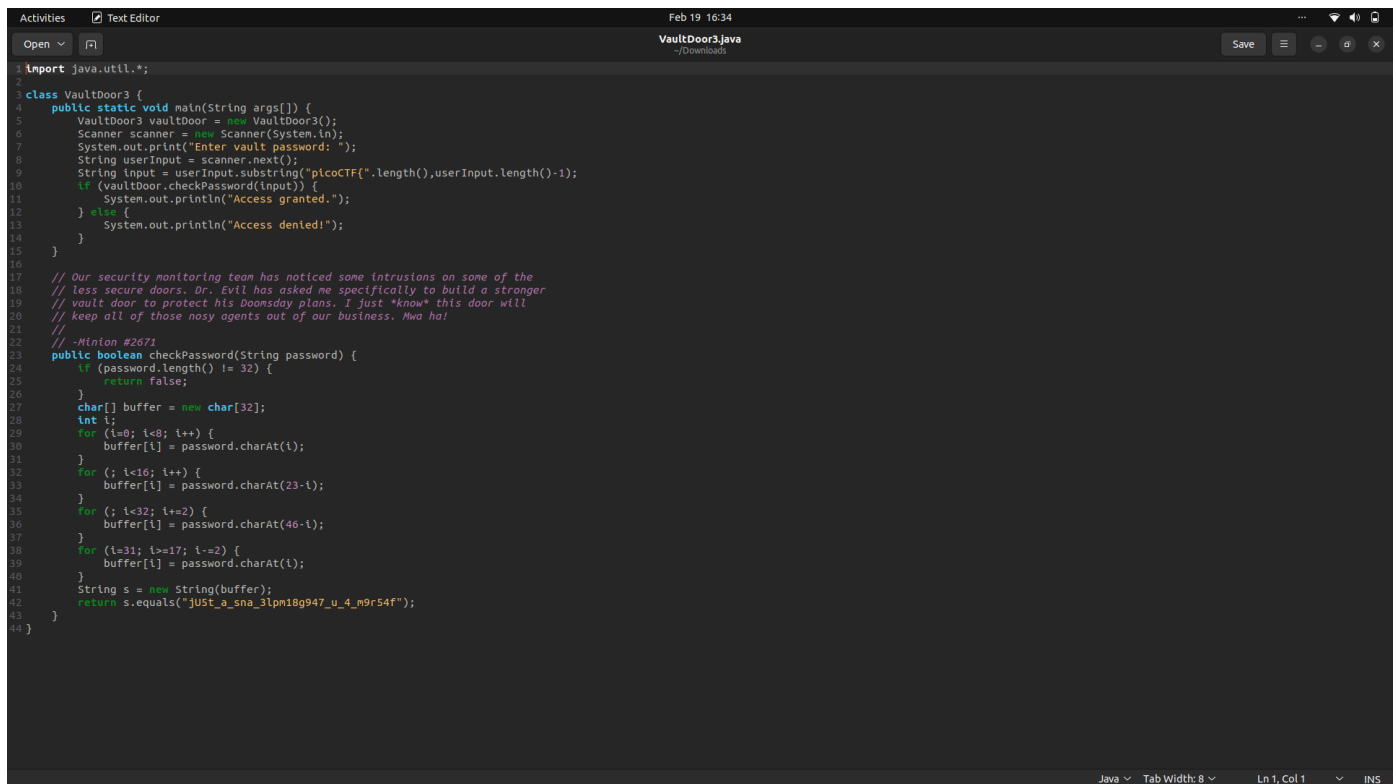
Thus, if we simply arrange the array from the indices given, we get the flag:



```
22 public boolean checkPassword(String password) {
23     return password.length() == 32 &&
24         password.charAt(0) == 'd' &&
25         password.charAt(29) == '9' &&
26         password.charAt(4) == 'r' &&
27         password.charAt(2) == '5' &&
28         password.charAt(23) == 'r' &&
29         password.charAt(3) == 'c' &&
30         password.charAt(17) == '4' &&
31         password.charAt(1) == '3' &&
32         password.charAt(7) == 'b' &&
33         password.charAt(10) == ' ' &&
34         password.charAt(5) == '4' &&
35         password.charAt(9) == '3' &&
36         password.charAt(11) == 't' &&
37         password.charAt(15) == 'c' &&
38         password.charAt(8) == 'l' &&
39         password.charAt(12) == 'H' &&
40         password.charAt(20) == 'c' &&
41         password.charAt(14) == ' ' &&
42         password.charAt(6) == 'm' &&
43         password.charAt(24) == '5' &&
44         password.charAt(18) == 'r' &&
45         password.charAt(13) == '3' &&
46         password.charAt(19) == '4' &&
47         password.charAt(21) == 'T' &&
48         password.charAt(16) == 'H' &&
49         password.charAt(27) == '5' &&
50         password.charAt(30) == '2' &&
51         password.charAt(25) == ' ' &&
52         password.charAt(22) == '3' &&
53         password.charAt(28) == '0' &&
54         password.charAt(26) == '7' &&
55         password.charAt(31) == 'e';
56     }
57 }
```

picoCTF{d35cr4mb13\_tH3\_cH4r4cT'3r5\_75092e}

VAULT-DOOR-3



```
1 import java.util.*;
2
3 class VaultDoor3 {
4     public static void main(String args[]) {
5         VaultDoor3 vaultDoor = new VaultDoor3();
6         Scanner scanner = new Scanner(System.in);
7         System.out.print("Enter vault password: ");
8         String userInput = scanner.next();
9         String input = userInput.substring("picoCTF".length(),userInput.length()-1);
10        if (vaultDoor.checkPassword(input)) {
11            System.out.println("Access granted.");
12        } else {
13            System.out.println("Access denied!");
14        }
15    }
16
17    // Our security monitoring team has noticed some intrusions on some of the
18    // less secure doors. Dr. Evil has asked me specifically to build a stranger
19    // vault door to protect his Doomsday plans. I just *know* this door will
20    // keep all of those nosy agents out of our business. Mwa ha!
21    //
22    // -Minton #2671
23    public boolean checkPassword(String password) {
24        if (password.length() != 32) {
25            return false;
26        }
27        char[] buffer = new char[32];
28        int i;
29        for (i=0; i<8; i++) {
30            buffer[i] = password.charAt(i);
31        }
32        for (; i<16; i++) {
33            buffer[i] = password.charAt(23-i);
34        }
35        for (; i<32; i+=2) {
36            buffer[i] = password.charAt(46-i);
37        }
38        for (i=31; i>=17; i-=2) {
39            buffer[i] = password.charAt(i);
40        }
41        String s = new String(buffer);
42        return s.equals("jU5t_a_s1mpl3_an4gr4m_4_u_c79a21");
43    }
44 }
```

As can be observed, the program compares the value of the input string with the characters at particular indices of the buffer string. Simply grouping all the required characters from the buffer string in the right order gives us the flag:

picoCTF{jU5t\_a\_s1mpl3\_an4gr4m\_4\_u\_c79a21}

## VAULT-DOOR 4

All we need to do is convert bytes to strings of base 10, and then convert it further to ASCII characters. This can be done using the function `Byte.toString()` for the first one and `Integer.parseInt()` for the second conversion.



While both are forms of encoding an array of characters, base64 converts binary to a corresponding number in base 64, while URL encoding uses only the characters from ASCII which are permitted in an URL to encode it.

https://www.base64decode.org/

Decode and Encode

Decode

Decode from Base64 format

Copy and paste your data then press the decode button.

Try to paste your data then press the decode button.

TRY TO PASTE YOUR DATA HERE

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8

Source character set.

Decode each line separately (useful for when you have multiple entries).

Use mode OFF

Decodes in real time as you type or paste (supports only the UTF-8 character set).

DECODE

Decodes your data into the area below.

NE2N30d6rN7PnJ2N72N749J32N6dN4E7N9J6d6N72N70N6dN4Pn6N2N732N6N5N4N6N39N34N5N6N7N3N1N2N32N6N2N6N9N34

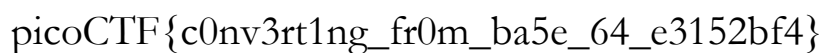
Decode files from Base64 format

Select a file to upload and process, then you can download the decoded result.

Other tools

URL Decode

URL Encode



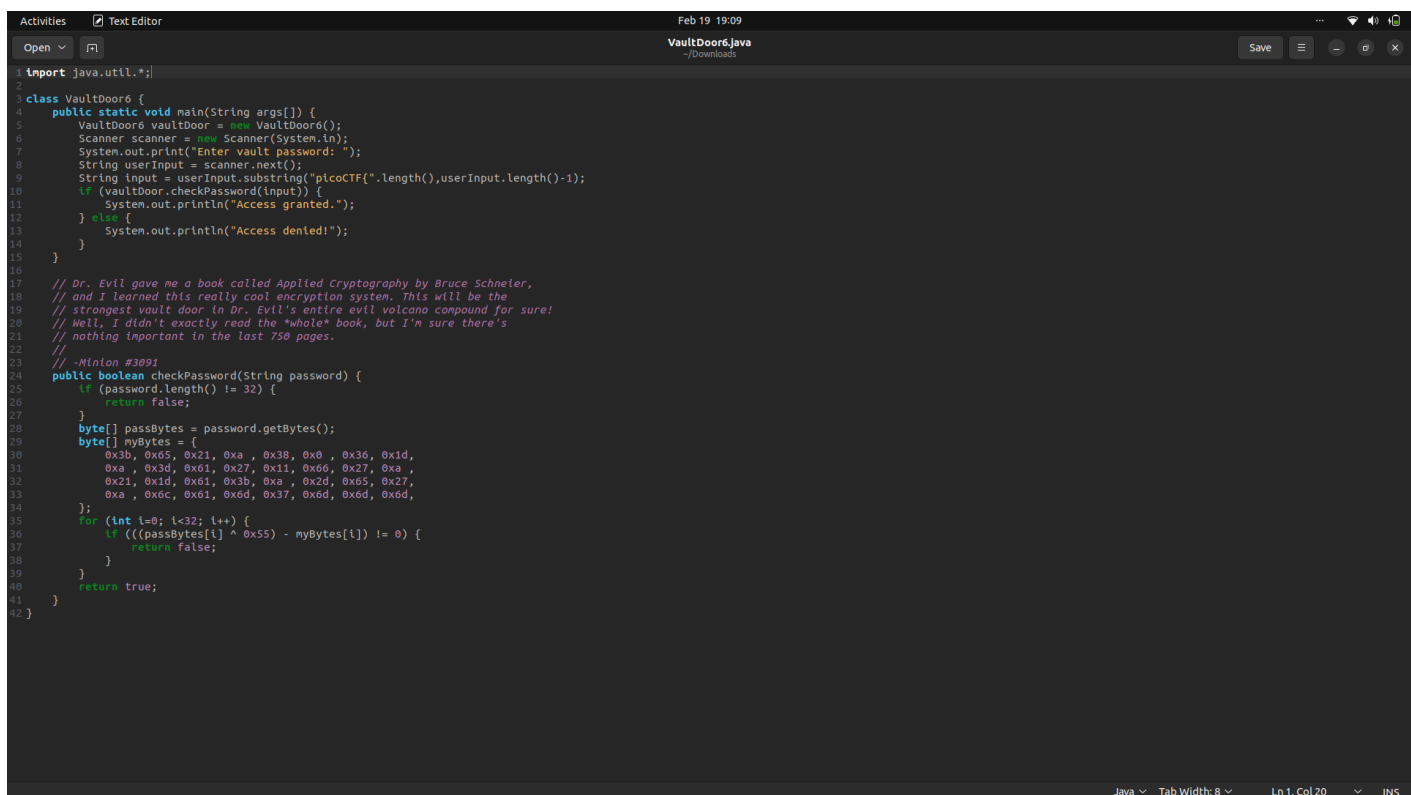
Base64:

<https://en.wikipedia.org/wiki/Base64>

URL Encoding:

[https://en.wikipedia.org/wiki/URL\\_encoding](https://en.wikipedia.org/wiki/URL_encoding)

## VAULT-DOOR 6



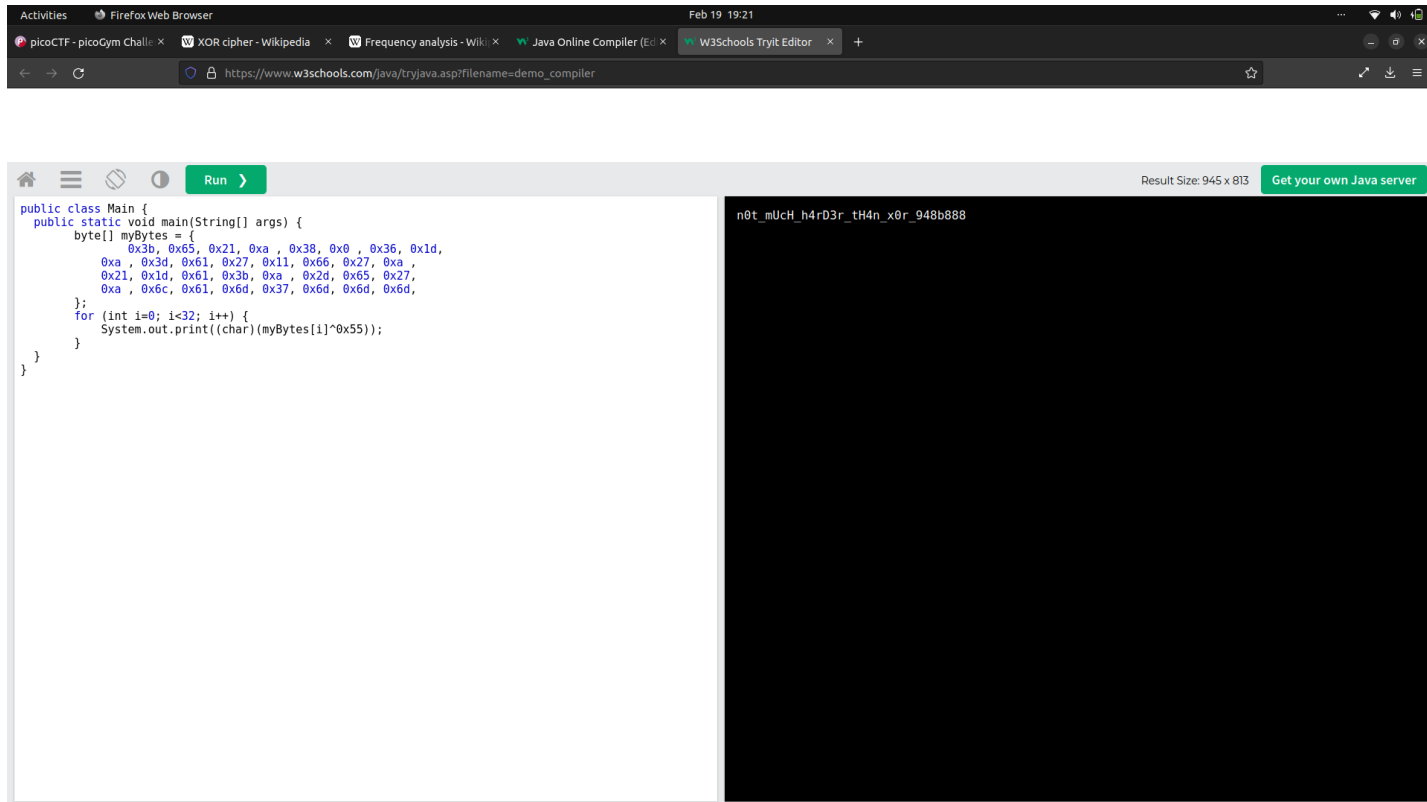
```
1 import java.util.*;
2
3 class VaultDoor6 {
4     public static void main(String args[]) {
5         VaultDoor6 vaultDoor = new VaultDoor6();
6         Scanner scanner = new Scanner(System.in);
7         System.out.print("Enter vault password: ");
8         String userInput = scanner.next();
9         String input = userInput.substring("picoCTF".length(), userInput.length()-1);
10        if (vaultDoor.checkPassword(input)) {
11            System.out.println("Access granted.");
12        } else {
13            System.out.println("Access denied!");
14        }
15    }
16
17    // Dr. Evil gave me a book called Applied Cryptography by Bruce Schneier,
18    // and I learned this really cool encryption system. This will be the
19    // strongest vault door in Dr. Evil's entire evil volcano compound for sure!
20    // Well, I didn't exactly read the "whole" book, but I'm sure there's
21    // nothing important in the last 750 pages.
22    //
23    // -Minion #3091
24    public boolean checkPassword(String password) {
25        if (password.length() != 32) {
26            return false;
27        }
28        byte[] passBytes = password.getBytes();
29        byte[] myBytes = {
30            0x3b, 0x65, 0x21, 0xa , 0x38, 0x0 , 0x36, 0x1d,
31            0xa , 0x3d, 0x61, 0x27, 0x11, 0x66, 0x27, 0xa ,
32            0x21, 0x1d, 0x61, 0x3b, 0xa , 0x2d, 0x65, 0x27,
33            0xa , 0x6c, 0x61, 0x6d, 0x37, 0x6d, 0x6d, 0x6d,
34        };
35        for (int i=0; i<32; i++) {
36            if (((passBytes[i] ^ 0x55) - myBytes[i]) != 0) {
37                return false;
38            }
39        }
40        return true;
41    }
42 }
```

As can be observed, we first need to convert bytes to string and for convenience, we'll be using the same program we used in a previous CTF, altered slightly such that we will XOR the output of the string first before printing. We could alternatively also use an online tool to convert the string which has not been decoded by XOR.

We know that each byte is equal to each byte in the string XOR 0x55 as can be seen in the line from the code:



“if (((passBytes[i] ^ 0x55) - myBytes[i]) != 0)”



```
public class Main {
    public static void main(String[] args) {
        byte[] myBytes = {
            0x3b, 0x65, 0x21, 0xa , 0x38, 0x0 , 0x36, 0x1d,
            0xa , 0x3d, 0x61, 0x27, 0x11, 0x66, 0x27, 0xa ,
            0x21, 0x1d, 0x61, 0x3b, 0xa , 0x2d, 0x65, 0x27,
            0xa , 0x6c, 0x61, 0x6d, 0x37, 0x6d, 0x6d, 0x6d,
        };
        for (int i=0; i<32; i++) {
            System.out.print((char)(myBytes[i]*0x55));
        }
    }
}
```

n0t\_mUcH\_h4rD3r\_tH4n\_x0r\_948b888

The flag is:

picoCTF{n0t\_mUcH\_h4rD3r\_tH4n\_x0r\_948b888}

Resources and References:

XOR: [https://en.wikipedia.org/wiki/XOR\\_cipher](https://en.wikipedia.org/wiki/XOR_cipher)

Online Java Editor and Compiler:

[https://www.w3schools.com/java/tryjava.asp?filename=demo\\_compiler](https://www.w3schools.com/java/tryjava.asp?filename=demo_compiler)

VAULT-DOOR 7

```
Activities Text Editor Feb 19 19:26
VaultDoor7.java
~/Downloads
Save

14 // System.out.println("Access granted.");
15 } else {
16     System.out.println("Access denied!");
17 }
18 }
19
20 // Each character can be represented as a byte value using its
21 // ASCII encoding. Each byte contains 8 bits, and an int contains
22 // 32 bits, so we can "pack" 4 bytes into a single int. Here's an
23 // example: If the hex string is "010b", then those can be
24 // represented as the bytes (0x0b, 0x21, 0x61, 0x62). When those
25 // bytes are represented as binary, they are:
26 //
27 // 0x0b: 00110000
28 // 0x21: 00110001
29 // 0x61: 01100001
30 // 0x62: 01100010
31 //
32 // If we put those 4 binary numbers end to end, we end up with 32
33 // bits that can be interpreted as an int.
34 //
35 // 00110000001100010110000101100010 -> 008542562
36 //
37 // Since 4 chars can be represented as 1 int, the 32 character password can
38 // be represented as an array of 8 ints.
39 //
40 // - Minion #7816
41 public int[] passwordToArray(String hex) {
42     int[] x = new int[8];
43     byte[] hexBytes = hex.getBytes();
44     for (int i=0; i<8; i++) {
45         x[i] = hexBytes[i*4] << 24
46             | hexBytes[i*4+1] << 16
47             | hexBytes[i*4+2] << 8
48             | hexBytes[i*4+3];
49     }
50     return x;
51 }
52
53 public boolean checkPassword(String password) {
54     if (password.length() != 32) {
55         return false;
56     }
57     int[] x = passwordToArray(password);
58     return x[0] == 1096770097
59         && x[1] == 1952395366
60         && x[2] == 1680270708
61         && x[3] == 1601398833
62         && x[4] == 1716808014
63         && x[5] == 1734304867
64         && x[6] == 942695730
65         && x[7] == 942740212;
66 }
67 }
```

The comment helps us understand what needs to be done in this challenge.

We first need to convert given integers to binary which is then to be broken down into four 8-bits long sequences. Each such sequence represents a byte which represents an integer which represents an ASCII character. This string of ASCII characters is our flag.

## Decimal to Binary converter

From: Decimal To: Binary

Enter decimal number: 1096770097 10

Binary number (31 digits): 100001010111101100100010001 2

Binary signed 2's complement (32 digits): 01000001010111101100100010001 2

Hex number (8 digits): 415F6231 16

☐ Digit grouping

Little endian

Address	0	1	2	3
Data	31	62	5F	41

Big endian

Address	0	1	2	3
Data	41	5F	62	31

### NUMBER CONVERSION

- [ASCII,Hex,Binary,Decimal converter](#)
- [ASCII text to binary converter](#)
- [ASCII text to hex converter](#)
- [Base converter](#)
- [Binary converter](#)
- [Binary to ASCII text converter](#)
- [Binary to decimal converter](#)
- [Binary to hex converter](#)
- [Date to roman numerals converter](#)

Dec	Hex	Binary	HTML	Char	Description
39	27	00100111	&#39;	'	Single quote
40	28	00101000	&#40;	(	Left parenthesis
41	29	00101001	&#41;	)	Right parenthesis
42	2A	00101010	&#42;	*	Asterisk
43	2B	00101011	&#43;	+	Plus
44	2C	00101100	&#44;	,	Comma
45	2D	00101101	&#45;	-	Minus
46	2E	00101110	&#46;	.	Period
47	2F	00101111	&#47;	/	Slash
48	30	00110000	&#48;	0	Zero
49	31	00110001	&#49;	1	One
50	32	00110010	&#50;	2	Two
51	33	00110011	&#51;	3	Three
52	34	00110100	&#52;	4	Four
53	35	00110101	&#53;	5	Five
54	36	00110110	&#54;	6	Six
55	37	00110111	&#55;	7	Seven
56	38	00111000	&#56;	8	Eight
57	39	00111001	&#57;	9	Nine
58	3A	00111010	&#58;	:	Colon
59	3B	00111011	&#59;	;	Semicolon
60	3C	00111100	&#60;	<	Less than
61	3D	00111101	&#61;	=	Equality sign
62	3E	00111110	&#62;	>	Greater than
63	3F	00111111	&#63;	?	Question mark
64	40	01000000	&#64;	@	At sign
65	41	01000001	&#65;	A	Capital A
66	42	01000010	&#66;	B	Capital B
67	43	01000011	&#67;	C	Capital C
68	44	01000100	&#68;	D	Capital D
69	45	01000101	&#69;	E	Capital E
70	46	01000110	&#70;	F	Capital F
71	47	01000111	&#71;	G	Capital G
72	48	01001000	&#72;	H	Capital H
73	49	01001001	&#73;	I	Capital I
74	4A	01001010	&#74;	J	Capital J
75	4B	01001011	&#75;	K	Capital K

Dec	Hex	Binary	HTML	Char	Description
39	27	00100111	&#39;	'	Single quote
40	28	00101000	&#40;	(	Left parenthesis
41	29	00101001	&#41;	)	Right parenthesis
42	2A	00101010	&#42;	*	Asterisk
43	2B	00101011	&#43;	+	Plus
44	2C	00101100	&#44;	,	Comma
45	2D	00101101	&#45;	-	Minus
46	2E	00101110	&#46;	.	Period
47	2F	00101111	&#47;	/	Slash
48	30	00110000	&#48;	0	Zero
49	31	00110001	&#49;	1	One
50	32	00110010	&#50;	2	Two
51	33	00110011	&#51;	3	Three
52	34	00110100	&#52;	4	Four
53	35	00110101	&#53;	5	Five
54	36	00110110	&#54;	6	Six
55	37	00110111	&#55;	7	Seven
56	38	00111000	&#56;	8	Eight
57	39	00111001	&#57;	9	Nine
58	3A	00111010	&#58;	:	Colon
59	3B	00111011	&#59;	;	Semicolon
60	3C	00111100	&#60;	<	Less than
61	3D	00111101	&#61;	=	Equality sign
62	3E	00111110	&#62;	>	Greater than
63	3F	00111111	&#63;	?	Question mark
64	40	01000000	&#64;	@	At sign
65	41	01000001	&#65;	A	Capital A
66	42	01000010	&#66;	B	Capital B
67	43	01000011	&#67;	C	Capital C
68	44	01000100	&#68;	D	Capital D
69	45	01000101	&#69;	E	Capital E
70	46	01000110	&#70;	F	Capital F
71	47	01000111	&#71;	G	Capital G
72	48	01001000	&#72;	H	Capital H
73	49	01001001	&#73;	I	Capital I
74	4A	01001010	&#74;	J	Capital J
75	4B	01001011	&#75;	K	Capital K

The flag is: picoCTF{A\_b1t\_of\_b1t\_sh1fTiNg\_dc80e28124}

Resources:

Decimal to Binary:

<https://www.rapidtables.com/convert/number/decimal-to-binary.html>

ASCII table: <https://www.rapidtables.com/code/text/ascii-table.html>

VAULT-DOOR 8

When we open the source code, we are greeted with a code without any spacing. After spending a lot of time beautifying it, we get a code like the following:

```
Activities Text Editor Feb 19 20:06
VaultDoor8.java
~/Downloads

1 import java.util.*;
2 import javax.crypto.Cipher;
3 import javax.crypto.spec.SecretKeySpec;
4 import java.security.*;
5 class VaultDoor8 { public static void main(String args[]) {
6     Scanner b = new Scanner(System.in);
7     System.out.println("Enter vault password: ");
8     String c = b.next();
9     String f = c.substring(8,c.length()-1);
10    VaultDoor8 a = new VaultDoor8();
11    if (a.checkPassword(f))
12    {
13        System.out.println("Access granted.");
14    }
15    else
16    {
17        System.out.println("Access denied!");
18    }
19 }
20 }
21 }
22 }
23 }
24 }
25 public char[] scramble(String password)
26 /* scramble a password by transposing pairs of bits. */
27 char[] a = password.toCharArray();
28 for (int b=a.length; b-->0; b++)
29 {
30     char c = a[b];
31     c = switchBits(c,1,2);
32     c = switchBits(c,0,3);
33     /* c = switchBits(c,14,3);
34     c = switchBits(c, 2, 0); */
35     c = switchBits(c,5,6);
36     c = switchBits(c,4,7);
37     c = switchBits(c,0,1);
38     /* d = switchBits(a, 4, 5);
39     e = switchBits(e, 5, 0); */
40     c = switchBits(c,3,4);
41     c = switchBits(c,2,5);
42     c = switchBits(c,6,7);
43     a[b] = c; }
44 return a;
45 }
46 public char switchBits(char c, int p1, int p2) {
47     /* Move the bit in position p1 to position p2, and move the bit
48     that was in position p2 to position p1. Precondition: p1 < p2 */
49     char mask1 = (char)(1 << p1);
50     char mask2 = (char)(1 << p2);
51     /* char mask1 = (char)(1 << p1 << p2); mask1++; mask1--; */
52     char b11 = (char)(c & mask1);
53     char b12 = (char)(c & mask2);
54     /* System.out.println("b11: " + Integer.toBinaryString(b11));
55     System.out.println("b12: " + Integer.toBinaryString(b12)); */
56     char rest = (char)(c & ~(mask1 | mask2));
57     char shift = (char)(p2 - p1);
58     char result = (char)((b11 << shift) | (b12 >> shift) | rest);
59     return result;
60 }
61 public boolean checkPassword(String password) {char[] scrambled = scramble(password);
```

Here we see a number of bit shifts. Lets start off by representing each character as what they are: 8 bits. Before the character is put through the scramble function, its indices are like this:

Index
0
1
2
3
4
5
6
7

Pretty obvious. Lets now go through the bit switches. The first instance of the **switchBits** function switches bit 1 and 2. The character now look like this:

Index	Original Index
0	0
1	2
2	1
3	3
4	4

5	5
6	6
7	7

We then switch bits 0 with 3, 5 with 6, and 4 with 7. Now every single bit has been switched once. The character now looks like this:

Index	Original Index
0	3
1	2
2	1
3	0
4	7
5	6
6	5
7	4

Once every bit has been switched once, we have to remember to switch the bits according to their current index and not their original index. So, when we continue to switch bits, we have to switch them based on their index. We continue and switch 0 with 1, 3 with 4, 2 with 5, and 6 with 7. The character now looks like this:

Index	Original Index
0	2
1	3
2	7
3	0
4	6
5	1
6	4
7	5

NOTE: The above explanation has been taken from the internet simply because this is too long to write and the original author has explained it nicely.

Now all we have to do is reverse this. We switch each index in the character with its original index as according to the above table to get the

original character, switching 0 with 2, 1 with 3, 2 with 7, etc. and putting it into a char array which we then convert to a String and get our flag.

The flag is: picoCTF{