Coding Ninjas Reversing WriteUp

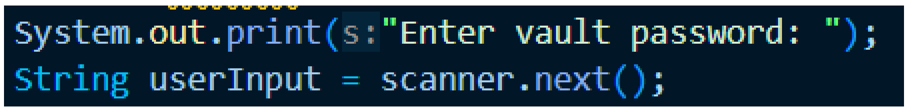
Usually in Reversing domain, what we are provided is an executable file. We have to disassemble to convert it into assembly from machine code, then decompile to convert it into languages like C/C++. For this we generally use tools like radare2 (for disassembly), jad-x (for decompilation) or Ghidra (which has both and more features like debugging, etc).
In these reversing questions, we are directly being provided with the decompiled code and only have to think about 'reversing' the logic behind the code.

CodingNinjas_rev0

At first look, it can be seen that there are two functions in this program.

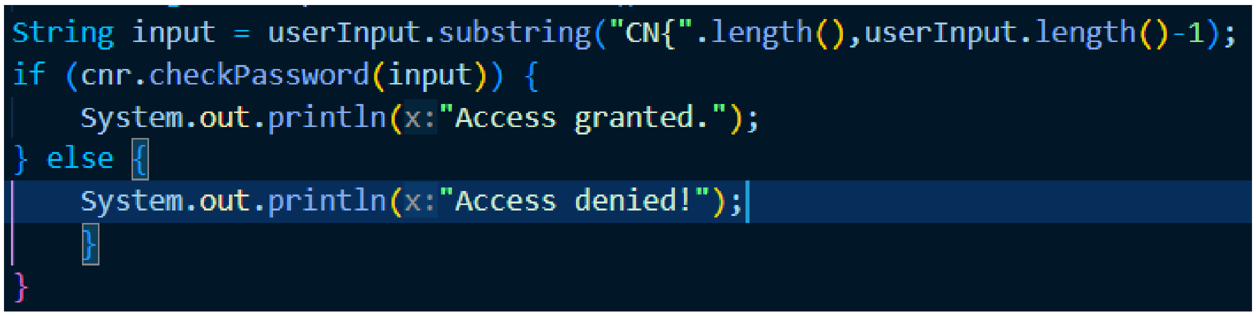
1.
```
public static void main(String args[])
```
2.
```
public boolean checkPassword(String password)
```

In Java the first function that is called by the JVM is the main function.
In the main function, we can see that



```
System.out.print(s:"Enter vault password: ");
String userInput = scanner.next();
```

Asks the user for an input and stores it in userInput.

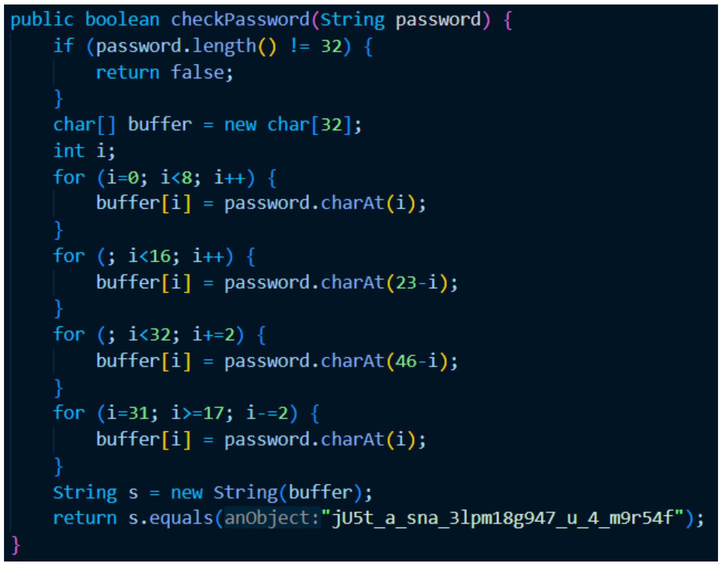


```
String input = userInput.substring("CN{".length(),userInput.length()-1);
if (cnr.checkPassword(input)) {
    System.out.println(x:"Access granted.");
} else {
    System.out.println(x:"Access denied!");
    }
}
```

It then removes the 'CN{' from the start of the input and '}' from the end of the input and stores it in a new string called 'input'.
It then proceeds to call the 'checkPassword()' function with the modified string 'input' as the parameter. If the return value of the 'checkPassword()' is True, then 'Access granted.' is printed, which is what we want. And if the return value is False, then access is denied.

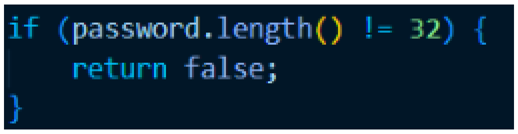
Let's look at the checkPassword(String password) function:

I should remind you now that “password” is the input that we gave to the program after removing the “CN{“ from the start and ‘}’ from the back



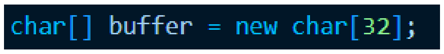
```java
public boolean checkPassword(String password) {
    if (password.length() != 32) {
        return false;
    }
    char[] buffer = new char[32];
    int i;
    for (i=0; i<8; i++) {
        buffer[i] = password.charAt(i);
    }
    for (; i<16; i++) {
        buffer[i] = password.charAt(23-i);
    }
    for (; i<32; i+=2) {
        buffer[i] = password.charAt(46-i);
    }
    for (i=31; i>=17; i-=2) {
        buffer[i] = password.charAt(i);
    }
    String s = new String(buffer);
    return s.equals(anObject:"jU5t_a_sna_3lpm18g947_u_4_m9r54f");
}
```

Now, the first if statement



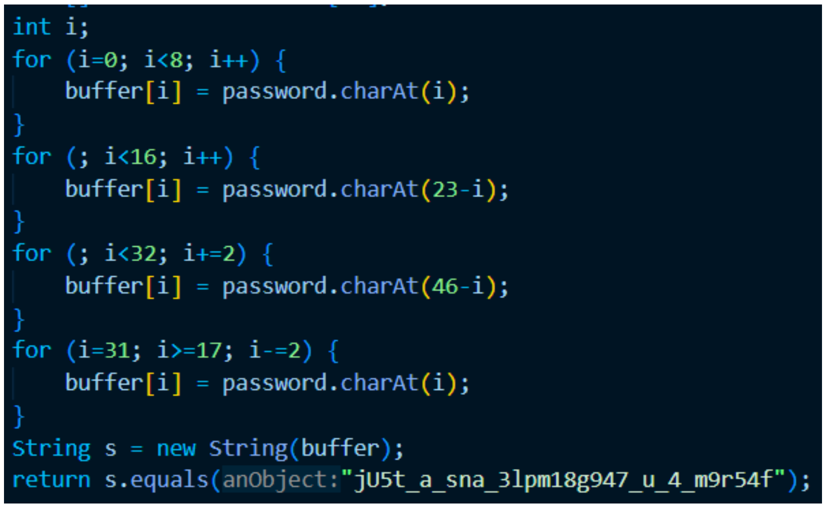
```java
if (password.length() != 32) {
    return false;
}
```

We can see that if the length of the parameter is not equal to 32, then the program will immediately return False, therefore we can understand that the flag is 32 characters long if CN{} is not considered.



```java
char[] buffer = new char[32];
```

An array buffer of characters is created with a size of 32. This buffer will be used to manipulate the password characters.



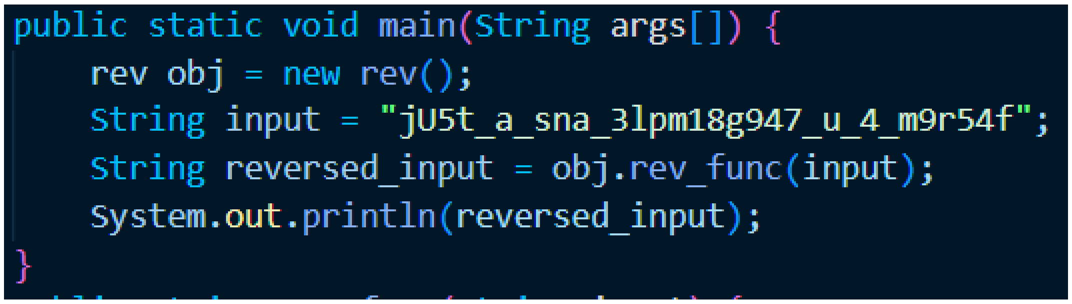
```java
int i;
for (i=0; i<8; i++) {
    buffer[i] = password.charAt(i);
}
for (; i<16; i++) {
    buffer[i] = password.charAt(23-i);
}
for (; i<32; i+=2) {
    buffer[i] = password.charAt(46-i);
}
for (i=31; i>=17; i-=2) {
    buffer[i] = password.charAt(i);
}
String s = new String(buffer);
return s.equals(anObject:"jU5t_a_sna_3lpm18g947_u_4_m9r54f");
```

Now there are four for loops which modify the input parameter which is “password” and basically rearrange it and store the jumbled version of “password” in buffer.
And the end, it gets compared to another string `"jU5t_a_sna_3lpm18g947_u_4_m9r54f"`

After analyzing the whole, we can understand that we enter the flag in the format CN{xyz}
Then it gets stripped to xyz, and then it gets jumbled to lets say yzx and gets compared to another string. If its same, then access will be granted.

Now thinking with the reverse engineer mindset, if we just reverse these steps, we should be able to find the original string which needs to be input to the program.

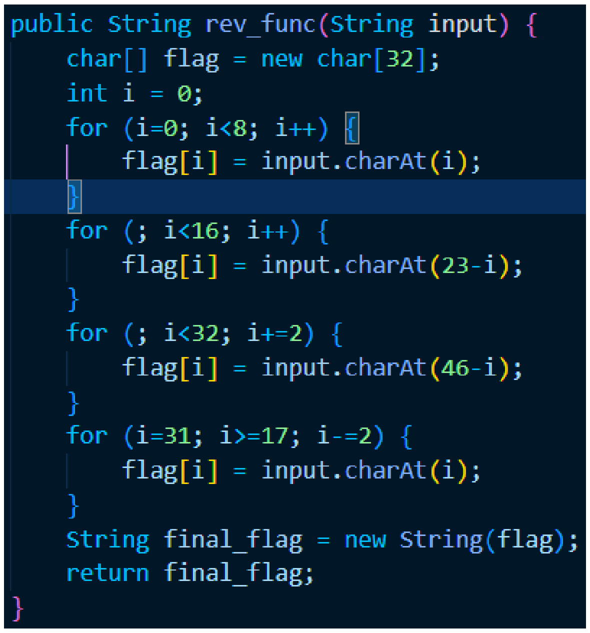
Lets create a new java program.



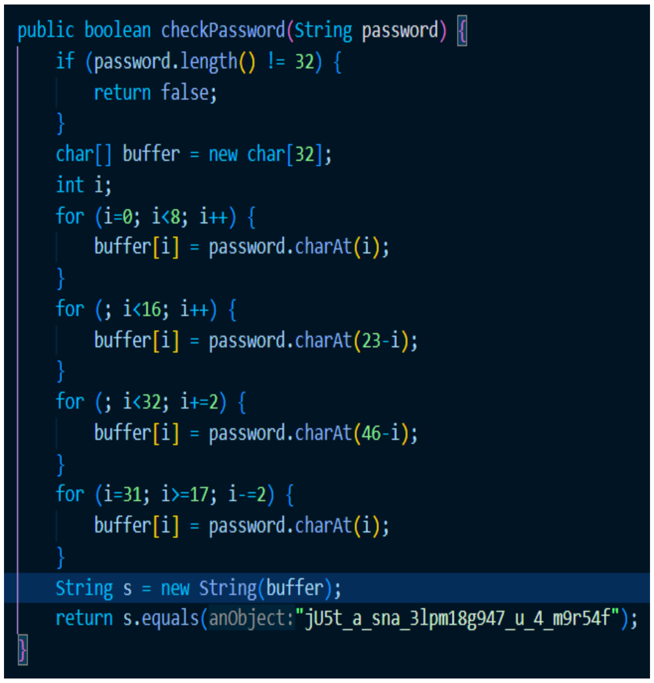
```java
public static void main(String args[]) {
    rev obj = new rev();
    String input = "jU5t_a_sna_3lpm18g947_u_4_m9r54f";
    String reversed_input = obj.rev_func(input);
    System.out.println(reversed_input);
}
```

In the main function of the new program, i am setting the input as the string that we were comparing with the modified string.
Then i am calling the reversed function and storing the returned string in “reversed_input” and simply printing it to the terminal.

The main magic happens in the “rev_func()” , lets have a look at it alongside with the original compare password function.



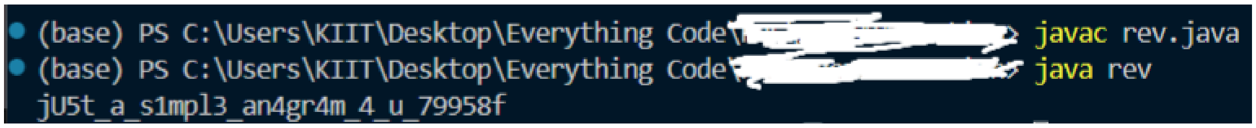
```
public String rev_func(String input) {
    char[] flag = new char[32];
    int i = 0;
    for (i=0; i<8; i++) {
        flag[i] = input.charAt(i);
    }
    for (; i<16; i++) {
        flag[i] = input.charAt(23-i);
    }
    for (; i<32; i+=2) {
        flag[i] = input.charAt(46-i);
    }
    for (i=31; i>=17; i-=2) {
        flag[i] = input.charAt(i);
    }
    String final_flag = new String(flag);
    return final_flag;
}
```



```
public boolean checkPassword(String password) {
    if (password.length() != 32) {
        return false;
    }
    char[] buffer = new char[32];
    int i;
    for (i=0; i<8; i++) {
        buffer[i] = password.charAt(i);
    }
    for (; i<16; i++) {
        buffer[i] = password.charAt(23-i);
    }
    for (; i<32; i+=2) {
        buffer[i] = password.charAt(46-i);
    }
    for (i=31; i>=17; i-=2) {
        buffer[i] = password.charAt(i);
    }
    String s = new String(buffer);
    return s.equals(anObject:"jU5t_a_sna_3lpm18g947_u_4_m9r54f");
}
```

If you look closely, theres nothing different.
We first, create a buffer array of characters of size 32 where we will store the flag.
And then apply the four for loops on it, and simply return the modified string.
Now you might be thinking - why did we not reverse the order of for loops?
Well if you look closely, each for loop modifies the string on a specific index to another specific index, hence changing the order of these for loops, would not matter.

Let's execute this program.

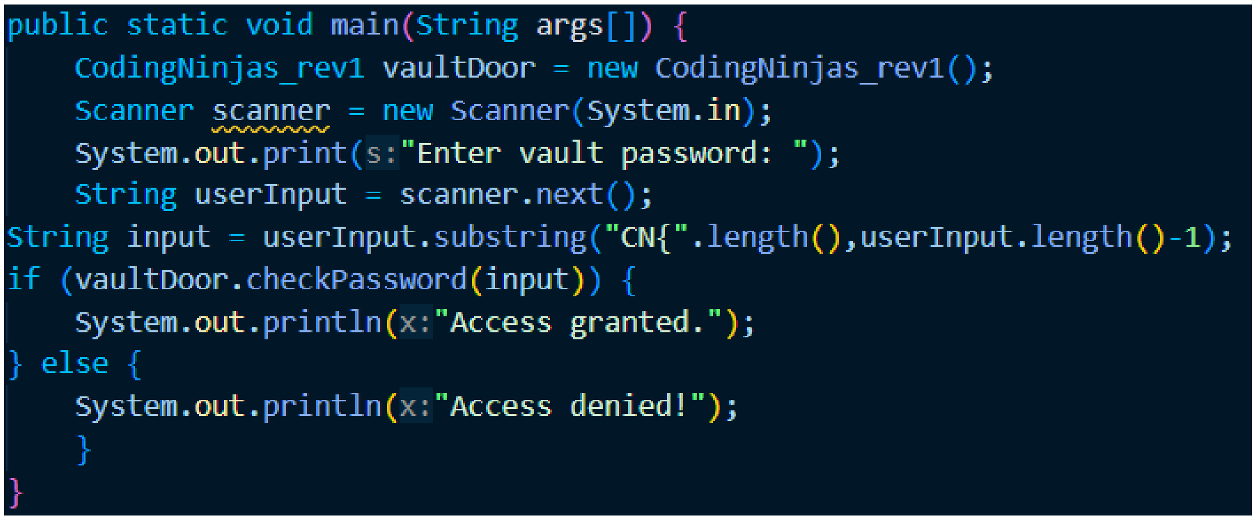


```
(base) PS C:\Users\KIIT\Desktop\Everything Code\[redacted]> javac rev.java
(base) PS C:\Users\KIIT\Desktop\Everything Code\[redacted]> java rev
jU5t_a_s1mpl3_an4gr4m_4_u_79958f
```

The output is "jU5t_a_s1mpl3_an4gr4m_4_u_79958f"
Hence the flag is CN{jU5t_a_s1mpl3_an4gr4m_4_u_79958f}

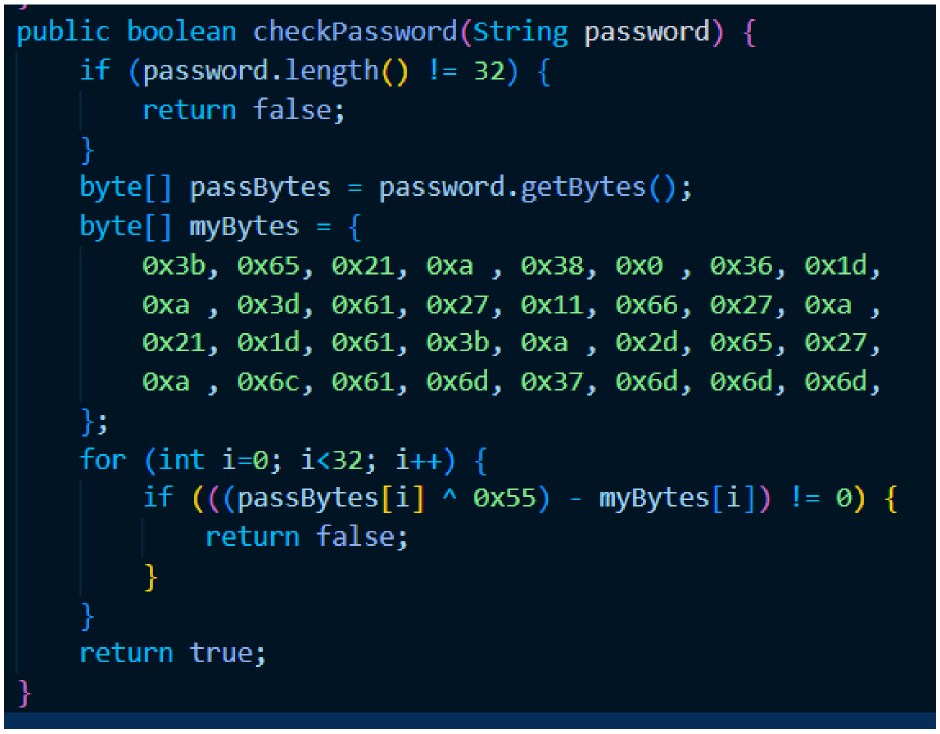
CodingNinjas_rev1

Similar to last program, this program also has 2 functions, a main function and a checkpassword function.



```
public static void main(String args[]) {
    CodingNinjas_rev1 vaultDoor = new CodingNinjas_rev1();
    Scanner scanner = new Scanner(System.in);
    System.out.print(s:"Enter vault password: ");
    String userInput = scanner.next();
String input = userInput.substring("CN{".length(),userInput.length()-1);
if (vaultDoor.checkPassword(input)) {
    System.out.println(x:"Access granted.");
} else {
    System.out.println(x:"Access denied!");
    }
}
```

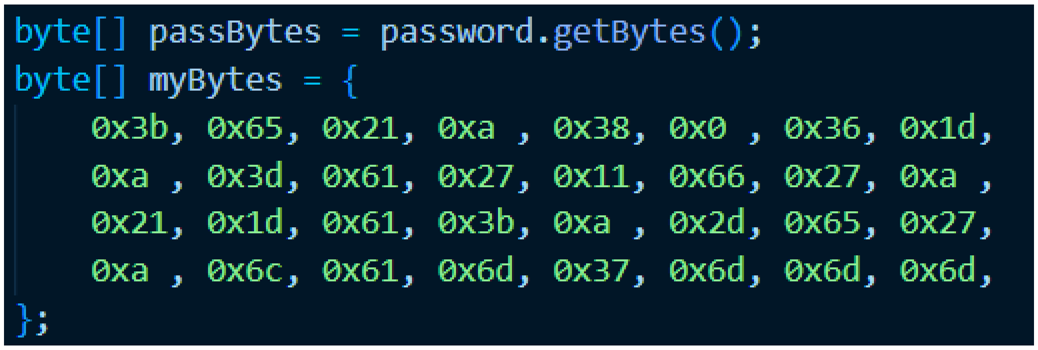
And similar to last function, its exactly the same,
Takes the input into userInput, strips CN{} from it and passes it through checkPassword() function.
So lets have a look at the checkPassword() function.



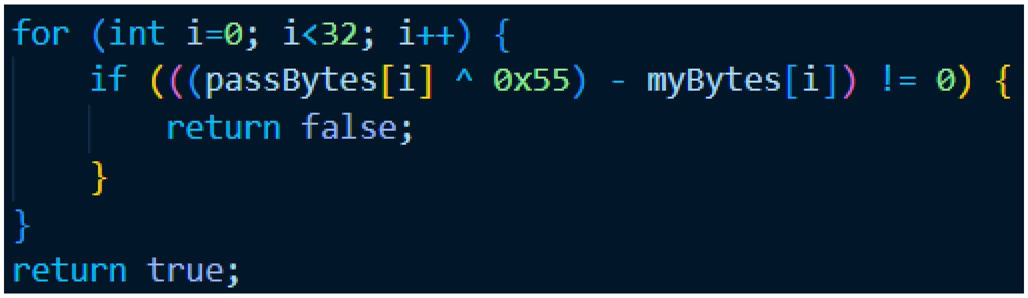
```
public boolean checkPassword(String password) {
    if (password.length() != 32) {
        return false;
    }
    byte[] passBytes = password.getBytes();
    byte[] myBytes = {
        0x3b, 0x65, 0x21, 0xa , 0x38, 0x0 , 0x36, 0x1d,
        0xa , 0x3d, 0x61, 0x27, 0x11, 0x66, 0x27, 0xa ,
        0x21, 0x1d, 0x61, 0x3b, 0xa , 0x2d, 0x65, 0x27,
        0xa , 0x6c, 0x61, 0x6d, 0x37, 0x6d, 0x6d, 0x6d,
    };
    for (int i=0; i<32; i++) {
        if (((passBytes[i] ^ 0x55) - myBytes[i]) != 0) {
            return false;
        }
    }
    return true;
}
```

Similar to last one it checks if the input to the function is of size 32 or not.

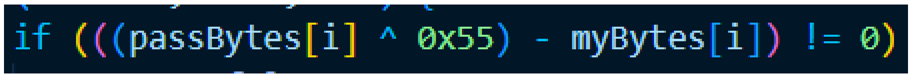
Lets have a look at the main part now,


```
byte[] passBytes = password.getBytes();
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,
};
```

Here, there are two byte arrays, one is the input to the function converted to bytearray using .getbytes function() and the other one which is myBytes which is declared explicitly.


```
for (int i=0; i<32; i++) {
    if (((passBytes[i] ^ 0x55) - myBytes[i]) != 0) {
        return false;
    }
}
return true;
```

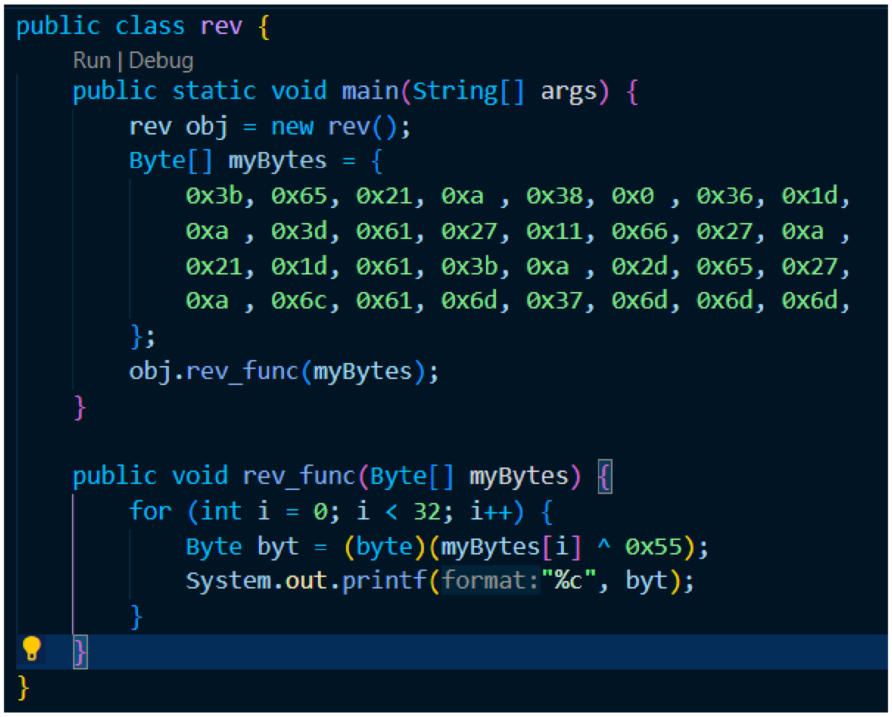
Now, the main thing is happening in this for loop,


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

For item byte in passBytes array, its being XOR'ed with 0x55 and it it is not equal to the item of mybytes array the function returns False, if all are same, then True is returned.

Hence we can understand that, whatever we give as the input to the program. Its being converted to bytes, and then each character's bytes are XOR'ed with 0x55, and those should be equal to the items present in myBytes array.

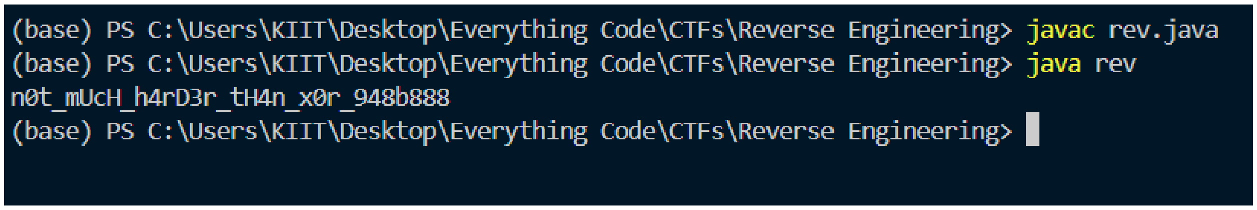
Let's reverse this program.



```
public class rev {
    Run | Debug
    public static void main(String[] args) {
        rev obj = new rev();
        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,
        };
        obj.rev_func(myBytes);
    }

    public void rev_func(Byte[] myBytes) {
        for (int i = 0; i < 32; i++) {
            Byte byt = (byte)(myBytes[i] ^ 0x55);
            System.out.printf(format:"%c", byt);
        }
    }
}
```

We start with declaring the myBytes array.
Now, what we are doing is reversing the XOR operation which was earlier being performed on each character byte of our input. Luckily, inverse of XOR is XOR itself, hence if we XOR each item in myBytes array with 0x55 and print it as a character, we should find our flag.
Running this program, we get



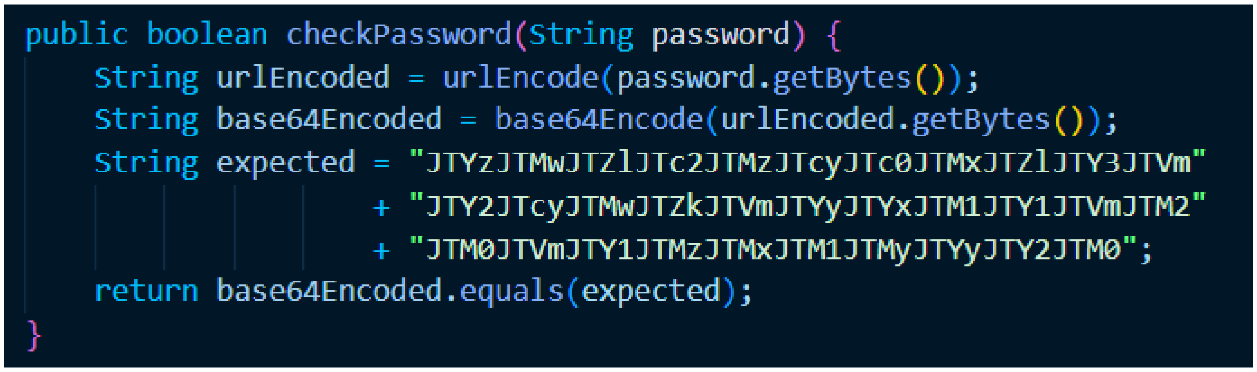
```
(base) PS C:\Users\KIIT\Desktop\Everything Code\CTFs\Reverse Engineering> javac rev.java
(base) PS C:\Users\KIIT\Desktop\Everything Code\CTFs\Reverse Engineering> java rev
n0t_mUcH_h4rD3r_tH4n_x0r_948b888
(base) PS C:\Users\KIIT\Desktop\Everything Code\CTFs\Reverse Engineering>
```

Hence the flag is CN{n0t_mUcH_h4rD3r_tH4n_x0r_948b888}

CodingNinjas_rev2

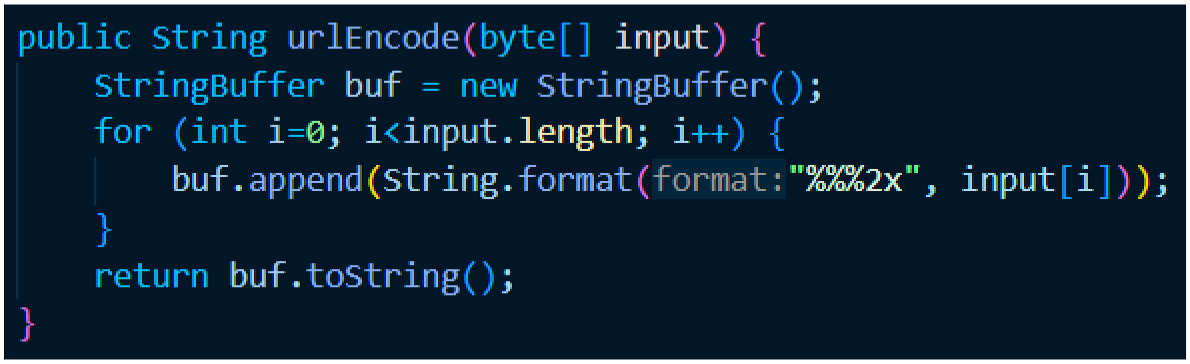
Similar to last two programs, this one also has a main function with exact similar functionality. However there are multiple other functions as well, which are base64Encode(), urlEncode() and checkPassword().

Since, in the main function, checkPassword is being called with the input parameter, lets take a look at it.



```java
public boolean checkPassword(String password) {
    String urlEncoded = urlEncode(password.getBytes());
    String base64Encoded = base64Encode(urlEncoded.getBytes());
    String expected = "JTYzJTMwJTZlJTc2JTMzJTcyJTc0JTMxJTZlJTY3JTVm"
                    + "JTY2JTcyJTMwJTZkJTVmJTYyJTYxJTM1JTY1JTVmJTM2"
                    + "JTM0JTVmJTY1JTMzJTMxJTM1JTMyJTYyJTY2JTM0";
    return base64Encoded.equals(expected);
}
```

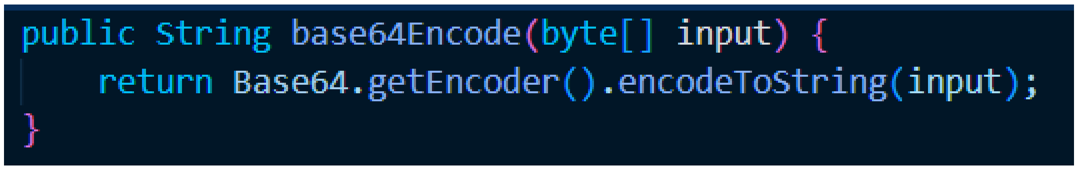
In the checkPassword() function, we can see that there are multiple functions being applied to the "password". Which are:
urlEncoded():



```java
public String urlEncode(byte[] input) {
    StringBuffer buf = new StringBuffer();
    for (int i=0; i<input.length; i++) {
        buf.append(String.format(format:"%%%2x", input[i]));
    }
    return buf.toString();
}
```

   a. What this function does is this encodes the the parameter passed into url encoding. Which is suggested by the name of the function itself.

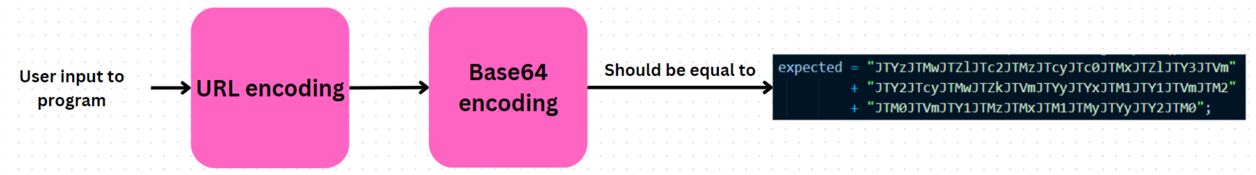
2. base64Encoded():



```java
public String base64Encode(byte[] input) {
    return Base64.getEncoder().encodeToString(input);
}
```

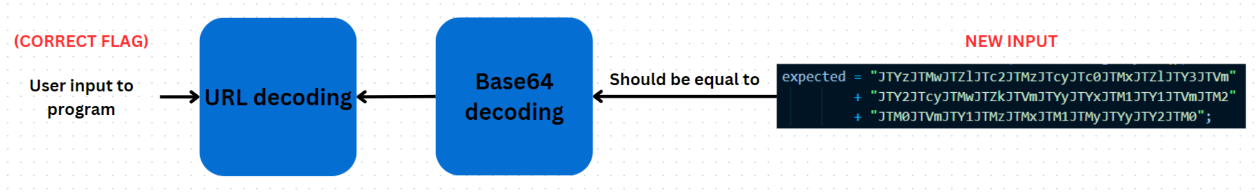
   a. Similarly this function encodes the parameter passed into Base64, again evident by the name of the function.

Then there is a expected String which is then matched with what has been produced by the previous two functions being performed one after another. If they match, checkPassword() returns true, else it returns False.



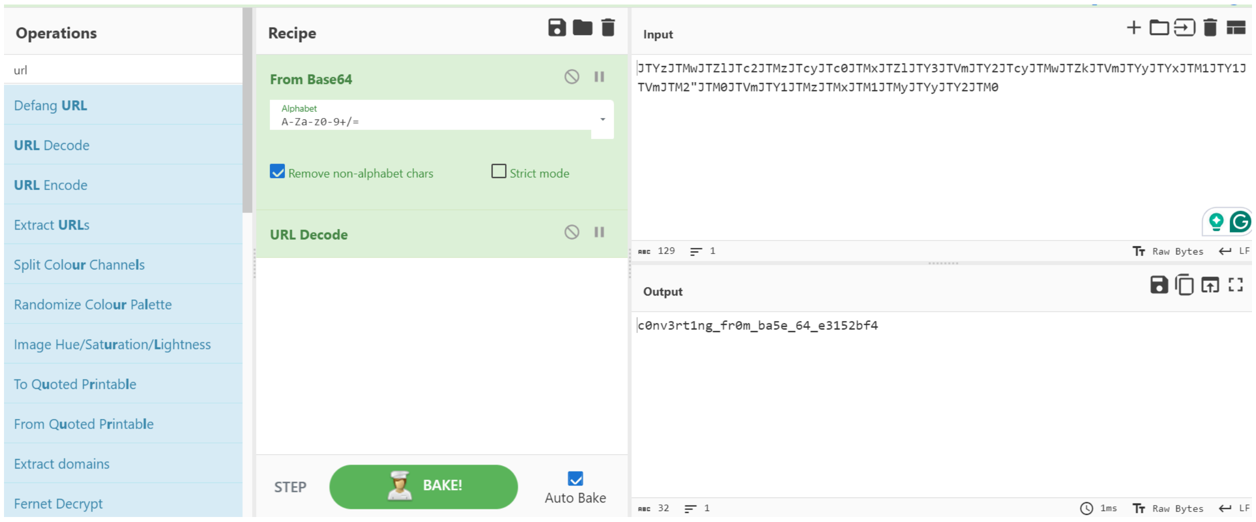
```
expected = "JTYzJTMwJTZlJTc2JTMzJTcyJTc0JTMxJTZlJTY3JTVm"
         + "JTY2JTcyJTMwJTZkJTVmJTYyJTYxJTM1JTY1JTVmJTM2"
         + "JTM0JTVmJTY1JTMzJTMxJTM1JTMyJTYyJTY2JTM0";
```

If we Reverse the functioning of this program, it would look like this:



```
expected = "JTYzJTMwJTZlJTc2JTMzJTcyJTc0JTMxJTZlJTY3JTVm"
         + "JTY2JTcyJTMwJTZkJTVmJTYyJTYxJTM1JTY1JTVmJTM2"
         + "JTM0JTVmJTY1JTMzJTMxJTM1JTMyJTYyJTY2JTM0";
```

Lets follow these steps on cyberchef.



Recipe

- From Base64
  - Alphabet A-Za-z0-9+/=
  - [✓] Remove non-alphabet chars
  - [ ] Strict mode
- URL Decode

Input

```
JTYzJTMwJTZlJTc2JTMzJTcyJTc0JTMxJTZlJTY3JTVmJTY2JTcyJTMwJTZkJTVmJTYyJTYxJTM1JTY1J
TVmJTM2"JTM0JTVmJTY1JTMzJTMxJTM1JTMyJTYyJTY2JTM0
```

Output

```
c0nv3rt1ng_fr0m_ba5e_64_e3152bf4
```

Hence the correct Flag is CN{c0nv3rt1ng_fr0m_ba5e_64_e3152bf4}