Reverse Engineering

vault-door-training

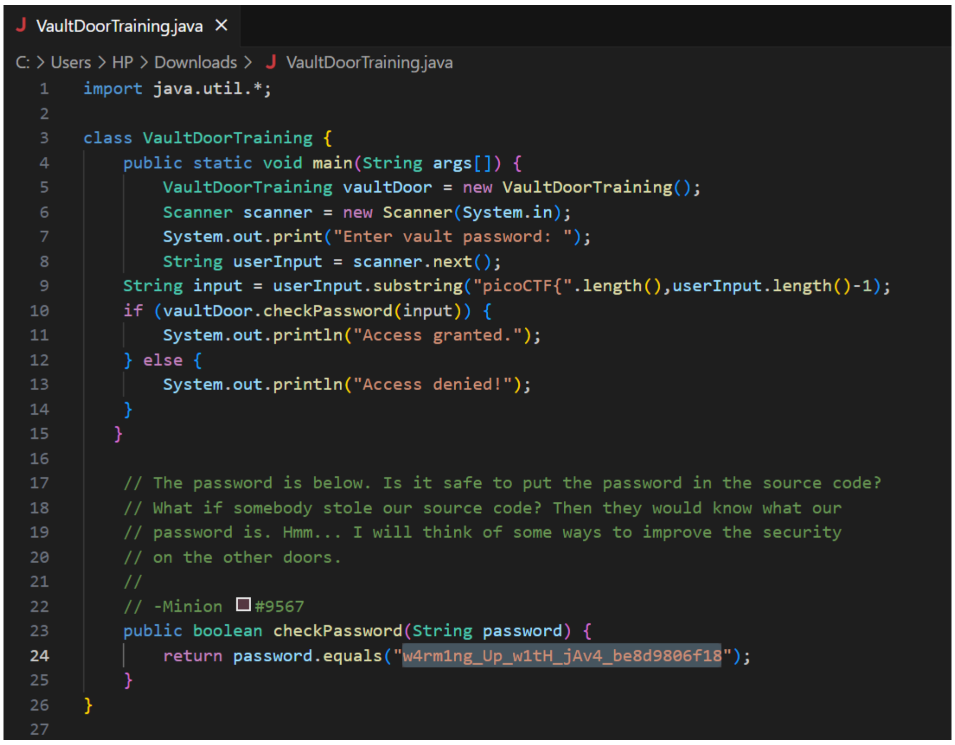
Author: Mark E. Haase

Description

Your mission is to enter Dr. Evil's laboratory and retrieve the blueprints for his Doomsday Project. The laboratory is protected by a series of locked vault doors. Each door is controlled by a computer and requires a password to open. Unfortunately, our undercover agents have not been able to obtain the secret passwords for the vault doors, but one of our junior agents obtained the source code for each vault's computer! You will need to read the source code for each level to figure out what the password is for that vault door. As a warmup, we have created a replica vault in our training facility. The source code for the training vault is here: VaultDoorTraining.java

Hints

The password is revealed in the program's source code.



```java
import java.util.*;

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

    // The password is below. Is it safe to put the password in the source code?
    // What if somebody stole our source code? Then they would know what our
    // password is. Hmm... I will think of some ways to improve the security
    // on the other doors.
    //
    // -Minion #9567
    public boolean checkPassword(String password) {
        return password.equals("w4rm1ng_Up_w1tH_jAv4_be8d9806f18");
    }
}

```

so the flag is picoCTF{w4rm1ng_Up_w1tH_jAv4_be8d9806f18}

vault-door-5

Author: Mark E. Haase

Description

In the last challenge, you mastered octal (base 8), decimal (base 10), and hexadecimal (base 16) numbers, but this vault door uses a different change of base as well as URL encoding! The source code for this vault is here: VaultDoor5.java

debug info: [u:880571 e: p: c:77 i:184]

Hints

You may find an encoder/decoder tool helpful, such as https://encoding.tools/

Read the wikipedia articles on URL encoding and base 64 encoding to understand how they work and what the results look like.

```
import java.net.URLDecoder;
import java.util.*;

class VaultDoor5 {
    public static void main(String args[]) {
        VaultDoor5 vaultDoor = new VaultDoor5();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
    String input = userInput.substring("picoCTF{".length(),userInput.length()-1);
    if (vaultDoor.checkPassword(input)) {
        System.out.println("Access granted.");
    } else {
        System.out.println("Access denied!");
        }
    }

    // Minion #7781 used base 8 and base 16, but this is base 64, which is
    // like... eight times stronger, right? Riiigghtt? Well that's what my twin
    // brother Minion #2415 says, anyway.
    //
    // -Minion #2414
    public String base64Encode(byte[] input) {
        return Base64.getEncoder().encodeToString(input);
    }

    // URL encoding is meant for web pages, so any double agent spies who steal
    // our source code will think this is a web site or something, defintely not
    // vault door! Oh wait, should I have not said that in a source code
    // comment?
    //
    // -Minion #2415
    public String urlEncode(byte[] input) {
        StringBuffer buf = new StringBuffer();
        for (int i=0; i<input.length; i++) {
            buf.append(String.format("%%%2x", input[i]));
        }
        return buf.toString();
    }

    public boolean checkPassword(String password) {
        String urlEncoded = urlEncode(password.getBytes());
        String base64Encoded = base64Encode(urlEncoded.getBytes());
        String expected = "JTYzJTMwJTZlJTc2JTMzJTcyJTc0JTMxJTZlJTY3JTVm"
                        + "JTY2JTcyJTMwJTZkJTVmJTYyJTYxJTM1JTY1JTVmJTM2"
                        + "JTM0JTVmJTMwJTYyJTM5JTM1JTM3JTYzJTM0JTY2";
        return base64Encoded.equals(expected);
    }
}
```

Understand the logic

The checkPassword() function works like this:

```
public boolean checkPassword(String password) {
    String urlEncoded = urlEncode(password.getBytes());      // Step 1: URL-encode
    String base64Encoded = base64Encode(urlEncoded.getBytes()); // Step 2: Base64-encode
    String expected = "...";  // The final encoded string
    return base64Encoded.equals(expected);
}
```

So:

password → URL-encoded → Base64-encoded → compared to expected

Our job is to **reverse this process**.

Expected value

From code:

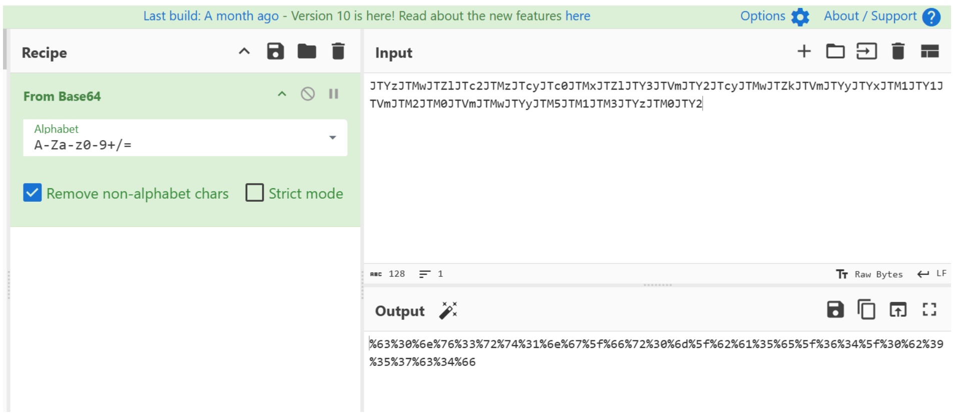
```
JTYzJTMwJTZlJTc2JTMzJTcyJTc0JTMxJTZlJTY3JTVm
JTY2JTcyJTMwJTZkJTVmJTYyJTYxJTM1JTY1JTVmJTM2
JTM0JTVmJTMwJTYyJTM5JTM1JTM3JTYzJTM0JTY2
```

Concatenate:

JTYzJTMwJTZlJTc2JTMzJTcyJTc0JTMxJTZlJTY3JTVmJTY2JTcyJTMwJTZkJTVmJTYyJTYxJTM1JTY1JTVmJTM2JT

Base64 decode

Decoding that base64 string gives us:


Last build: A month ago - Version 10 is here! Read about the new features here
Options
About / Support
Recipe
Input
Output
Raw Bytes
LF

From Base64

Alphabet: A-Za-z0-9+/=

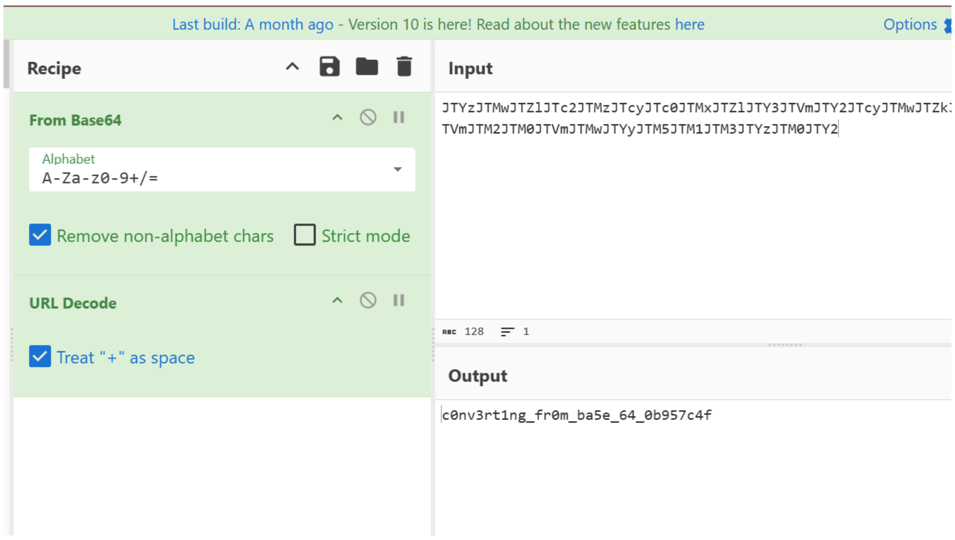
☑ Remove non-alphabet chars ☐ Strict mode

```
JTYzJTMwJTZlJTc2JTMzJTcyJTc0JTMxJTZlJTY3JTVmJTY2JTcyJTMwJTZkJTVmJTYyJTYxJTM1JTY1J
TVmJTM2JTM0JTVmJTMwJTYyJTM5JTM1JTM3JTYzJTM0JTY2
```

128 1

```
%63%30%6e%76%33%72%74%31%6e%67%5f%66%72%30%6d%5f%62%61%35%65%5f%36%34%5f%30%62%39
%35%37%63%34%66
```

now using url decode to decode this
%63%30%6e%76%33%72%74%31%6e%67%5f%66%72%30%6d%5f%62%61%35%65%5f%36%34%5f%30%62%


Last build: A month ago - Version 10 is here! Read about the new features here
Options
Recipe
Input
Output

From Base64

Alphabet: A-Za-z0-9+/=

☑ Remove non-alphabet chars ☐ Strict mode

URL Decode

☑ Treat "+" as space

```
JTYzJTMwJTZlJTc2JTMzJTcyJTc0JTMxJTZlJTY3JTVmJTY2JTcyJTMwJTZk
TVmJTM2JTM0JTVmJTMwJTYyJTM5JTM1JTM3JTYzJTM0JTY2
```

128 1

```
c0nv3rt1ng_fr0m_ba5e_64_0b957c4f
```

so the flag is

picoCTF{c0nv3rt1ng_fr0m_ba5e_64_0b957c4f}

vault-door-4

Author: Mark E. Haase

Description

This vault uses ASCII encoding for the password. The source code for this vault is here: VaultDoor4.java

```
import java.util.*;

class VaultDoor4 {
    public static void main(String args[]) {
        VaultDoor4 vaultDoor = new VaultDoor4();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
    String input = userInput.substring("picoCTF{".length(),userInput.length()-1);
    if (vaultDoor.checkPassword(input)) {
        System.out.println("Access granted.");
    } else {
        System.out.println("Access denied!");
        }
    }

    // I made myself dizzy converting all of these numbers into different bases,
    // so I just *know* that this vault will be impenetrable. This will make Dr.
    // Evil like me better than all of the other minions--especially Minion
    // #5620--I just know it!
    //
    //  .:::.   .:::.
    // :::::::.:::::::
    // :::::::::::::::
    // ':::::::::::::'
    //   ':::::::::'
    //     ':::::'
    //       ':'
    // -Minion #7781
    public boolean checkPassword(String password) {
        byte[] passBytes = password.getBytes();
        byte[] myBytes = {
            106 , 85  , 53  , 116 , 95  , 52  , 95  , 98  ,
            0x55, 0x6e, 0x43, 0x68, 0x5f, 0x30, 0x66, 0x5f,
            0142, 0131, 0164, 063 , 0163, 0137, 0146, 064 ,
            'a' , '8' , 'c' , 'd' , '8' , 'f' , '7' , 'e' ,
        };
        for (int i=0; i<32; i++) {
            if (passBytes[i] != myBytes[i]) {
                return false;
            }
        }
        return true;
    }
}
```

Step 1: Array values

```
byte[] myBytes = {
   106 , 85  , 53  , 116 , 95  , 52  , 95  , 98  ,
   0x55, 0x6e, 0x43, 0x68, 0x5f, 0x30, 0x66, 0x5f,
   0142, 0131, 0164, 063 , 0163, 0137, 0146, 064 ,
   'a' , '8' , 'c' , 'd' , '8' , 'f' , '7' , 'e' ,
};
```

- Decimal: `106, 85, 53, 116 ...`
- Hex: `0x55, 0x6e ...`
- Octal: `0142, 0131 ...`
- Direct chars: `'a','8','c','d'...`

Convert each to ASCII

- 106 → `j`
- 85 → `U`
- 53 → `5`
- 116 → `t`
- 95 → `_`
- 52 → `4`
- 95 → `_`
- 98 → `b`
- 0x55 = 85 → `U`
- 0x6e = 110 → `n`
- 0x43 = 67 → `C`
- 0x68 = 104 → `h`
- 0x5f = 95 → `_`
- 0x30 = 48 → `0`
- 0x66 = 102 → `f`
- 0x5f = 95 → `_`
- 0142 (octal) = 98 → `b`
- 0131 (octal) = 89 → `Y`
- 0164 (octal) = 116 → `t`
- 063 (octal) = 51 → `3`
- 0163 (octal) = 115 → `s`
- 0137 (octal) = 95 → `_`
- 0146 (octal) = 102 → `f`
- 064 (octal) = 52 → `4`
- `'a'` → `a`
- `'8'` → `8`
- `'c'` → `c`
- `'d'` → `d`
- `'8'` → `8`
- `'f'` → `f`

- `'7'` → `7`
- `'e'` → `e`

Combine

```
jU5t_4_bUnCh_0f_bYt3s_f4a8cd8f7e
```

so the flag is

picoCTF{jU5t_4_bUnCh_0f_bYt3s_f4a8cd8f7e}