

Altering Method Values

In the previous section, we explored how to hook the return value of a Java method using Frida. Now, we'll take it a step further by not only intercepting the return value but also modifying it before passing it back to the application. This technique is particularly useful for manipulating application behavior at runtime and observing how the app reacts to unexpected or edge-case inputs.

By altering return values dynamically, we can simulate conditions that might not occur during normal use, which can help uncover hidden vulnerabilities or logic flaws. This method enhances both the depth and effectiveness of a security assessment, while also contributing to the development of more resilient applications.

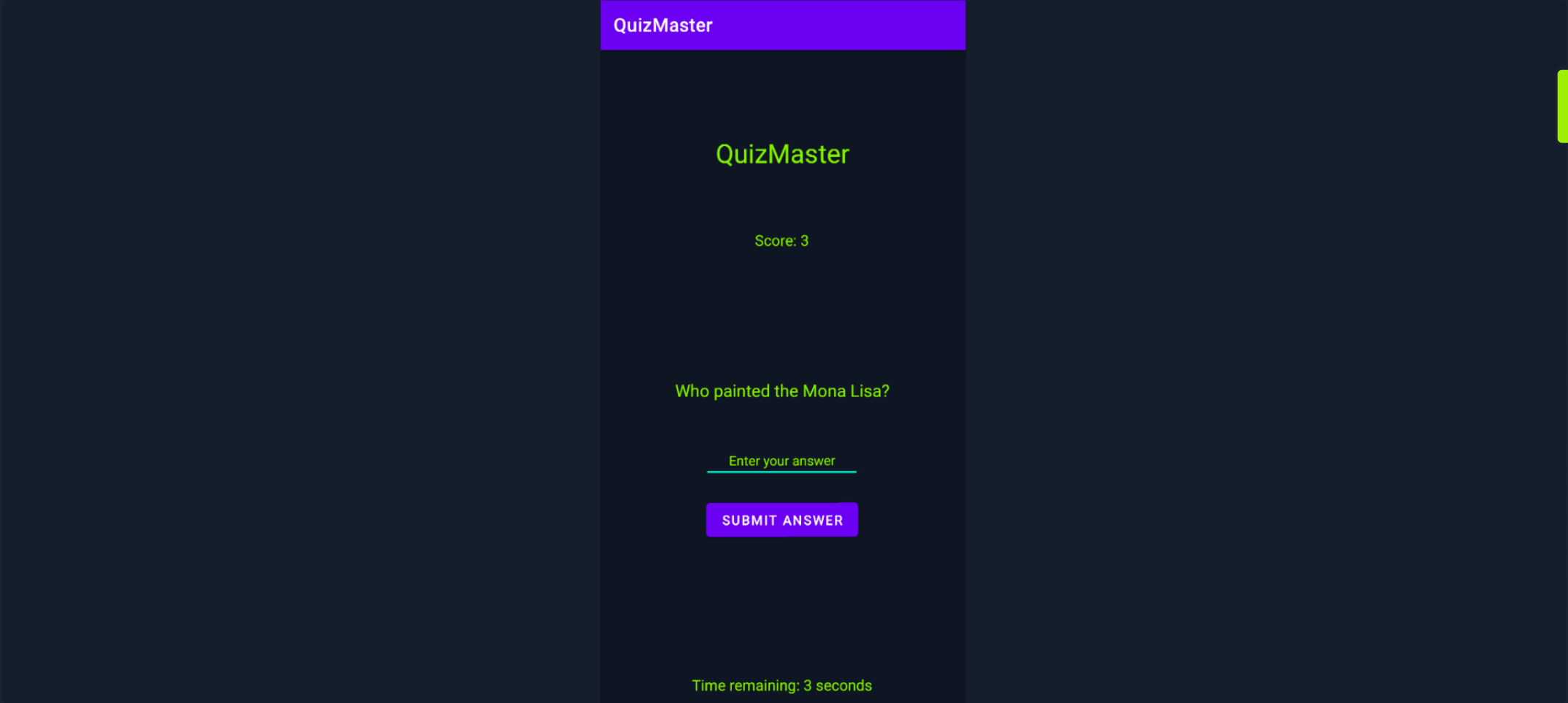
In this example, we'll use an Android Virtual Device (AVD), though the same process applies to physical or other emulated devices. Let's begin by connecting to the device via ADB and installing the application.

Altering Method Values

```
r11k@htb[/htb]$ adb connect
r11k@htb[/htb]$ adb install myapp.apk

Performing Streamed Install
Success
```

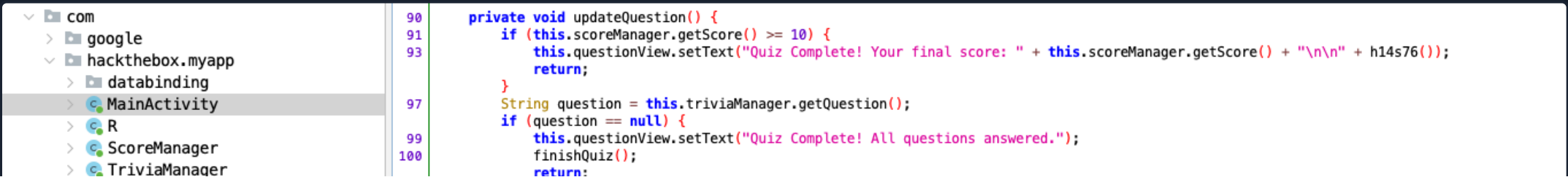
Running the application, we see that it is a quiz game that presents the user with questions to answer within a short time limit of 10 seconds.



When a question is answered correctly, the score increases by 1 point. If the timer expires, the message "Time's up! Please answer faster next time." is displayed. Given the limited time available to respond, users might attempt to find alternative methods of collecting points. Let's test the app to identify unintended ways of gaining points. We will begin by using JADX to examine the application's source code.

Altering Method Values

```
r11k@htb[/htb]$ jadx-gui myapp.apk
```



```
> kotlin
> kotlinox.coroutines
> org
> Resources
102         this.questionView.setText(question);
103         this.answerInput.setText("");
    }
```

The snippet above reveals the method `updateQuestion()`.

Code: `java`

```
this.scoreManager.getScore() >= 10
```

This method checks whether the score is greater than `10`. If so, the message `Quiz Complete! Your final score:` is displayed, followed by the score retrieved from `this.scoreManager.getScore()` and the result of the method `h14s76()`. In other words, this message appears only when the score is sufficiently high.

While it's possible to trigger this condition by significantly increasing the score using application patching techniques, a deeper look at the code reveals another method of interest: `isAppSignatureValid()`.

```
com
├── google
├── hackthebox.myapplication
│   ├── databinding
│   ├── MainActivity
│   ├── R
│   ├── ScoreManager
│   └── TriviaManager
├── kotlin
├── kotlinox.coroutines
├── org
└── Resources
    ├── APK signature
    └── Summary
138 private boolean isAppSignatureValid() {
    Signature[] signatureArr;
    try {
        for (Signature signature : getPackageManager().getPackageInfo(getPackageName(), 64).signatures) {
            MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
            messageDigest.update(signature.toByteArray());
            if ("6f5ab54275107245a26dcca21e7c3828ed404234e20dc416b3d0e07d370c7a1b".equalsIgnoreCase(bytesToHex(messageDigest.digest()))) {
                return true;
            }
        }
    } catch (PackageManager.NameNotFoundException | NoSuchAlgorithmException unused) {
    }
    return false;
}
158 private static String bytesToHex(byte[] bytes) {
159     StringBuilder sb = new StringBuilder();
    for (byte b : bytes) {
        sb.append(String.format("%02x", Byte.valueOf(b)));
    }
    return sb.toString();
}
161
163 }
```

The method in question verifies the application's certificate signature. This form of anti-patching is commonly implemented in Android apps to detect tampering. While it can often be bypassed by patching the app, developers frequently add multiple, distributed checks throughout the codebase. Instead of removing all of them, we can use dynamic code instrumentation with Frida to override the method's return value at runtime.

Assuming the score is returned by the `getScore()` method, and knowing the application's package name, we can now create a JavaScript script to modify the score dynamically. Create a file named `snippets.js` and add the following content.

Code: `js`

```
// Use Frida's Java.perform method to safely interact with Java classes
Java.perform(function () {
    // Obtain a reference to the 'ScoreManager' class from the target app
    var ScoreManager = Java.use('com.hackthebox.myapplication.ScoreManager');

    // Override the implementation of the 'getScore' method in the ScoreManager class
    ScoreManager.getScore.implementation = function () {
        return 12; // Always return 12, effectively forcing the score to always be greater than 10
    };
});
```

We see that `getScore()` is a method of the `ScoreManager` class:

Code: `java`

```
this.scoreManager.getScore() >= 10
```

Therefore, we need to hook into the correct class in our Frida script. This means we should instantiate the class as:

Code: `java`

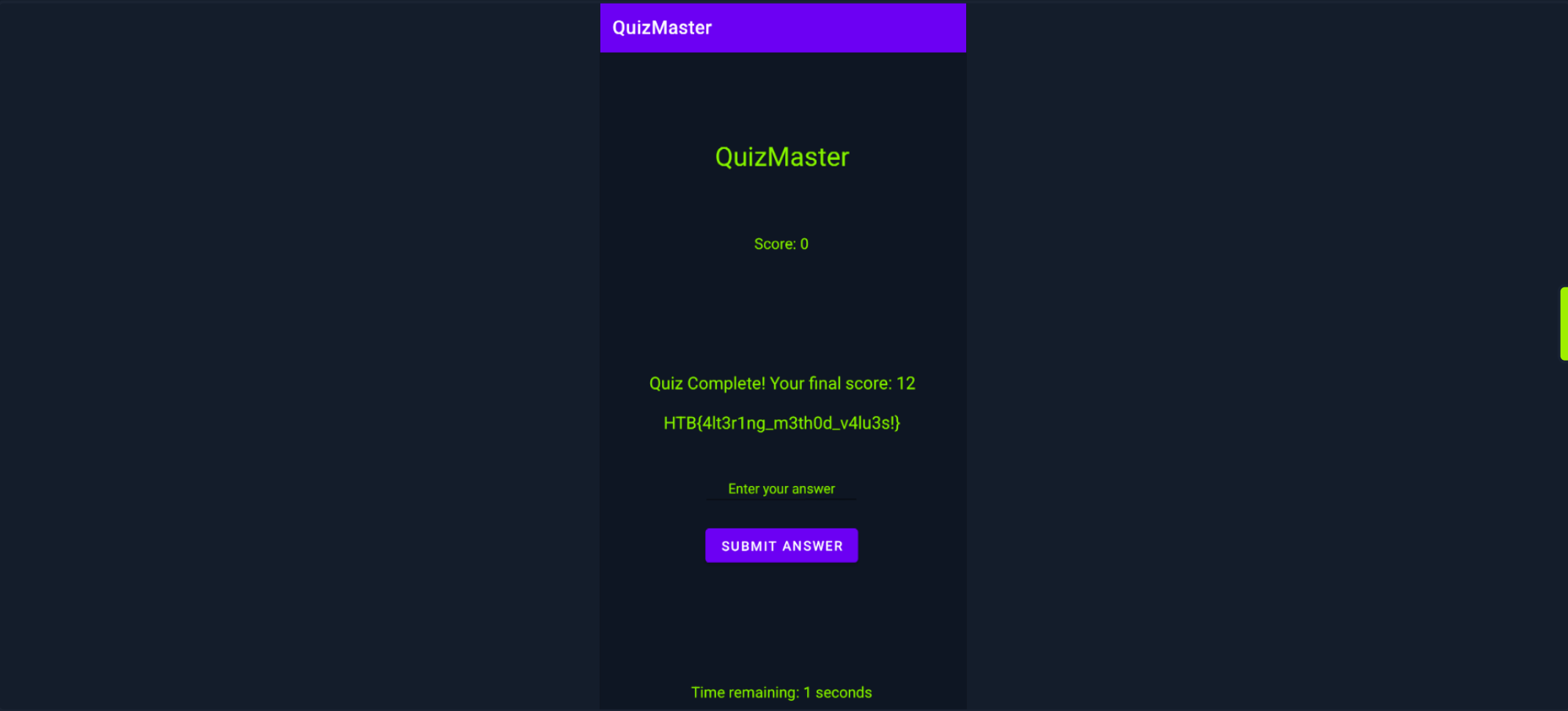
```
Java.use('com.hackthebox.myapp.ScoreManager');
```

Once we have the correct class reference, we can override the `getScore()` method to return a value of 12, which satisfies the condition for displaying the success message. Assuming the Frida server is already installed and running on the device, we can inject our script into the application using the following command:

Altering Method Values

```
r1k@htb[/htb]$ frida -U -l snippet.js -f com.hackthebox.myapp

----
/ _ |   Frida 16.1.11 - A world-class dynamic instrumentation toolkit
| (| |
> _ |   Commands:
/_/ |_|   help       -> Displays the help system
. . . .   object?    -> Display information about 'object'
. . . .   exit/quit  -> Exit
. . . .
. . . .   More info at https://frida.re/docs/home/
. . . .
. . . .   Connected to Android Emulator 5554 (id=emulator-5554)
Spawned `com.hackthebox.myapp`. Resuming main thread!
[Android Emulator 5554::com.hackthebox.myapp ]->
```



Static analysis techniques, such as inspecting the app's source code with JADX, are one way to identify the method and class names needed to craft a script that alters values. Another option is to gather this information at runtime through code instrumentation. If the objective is simple—for example, assigning a high score in the quiz game app—it's best to start by enumerating the methods of all custom classes in the application, while excluding those from external libraries. To do this, create a file named `list_methods.js` and add the following JavaScript snippet. Each line is explained through comments placed directly above the corresponding code.

Code: js

```
// Execute the following code within the context of the Java VM
Java.perform(function () {
  // Specify the base package name of the app you're interested in
  var targetPackageName = 'com.hackthebox.myapp';

  // Enumerate all classes currently loaded by the Java VM
  Java.enumerateLoadedClasses({
    // Function called for each class found
    onMatch: function(className) {
      // Check if the class belongs to our target package
      if (className.startsWith(targetPackageName)) {
```

```

    try {
        // Use the class name to get a reference to its Java Class object
        var clazz = Java.use(className);
        // Log the class name being processed
        console.log('\n[*] Enumerating methods of class: ' + className);

        // Get all methods declared by the class
        var methods = clazz.class.getDeclaredMethods();
        // Iterate over each method and log its signature
        methods.forEach(function(method) {
            console.log(method.toString());
        });

        // Clean up the reference to the Java Class object
        clazz.$dispose();
    } catch (err) {
        // Log any errors that occur during processing
        console.error('Error enumerating methods of ' + className + ': ' + err.message);
    }
},
// Function called once all classes have been processed
onComplete: function() {
    // Log a message indicating completion of the enumeration process
    console.log('[*] Class enumeration complete');
}
});
});

```

Next, run the following command and wait for the app to start.

Altering Method Values

```

rl1k@htb[/htb]$ frida -U -l list_methods.js -f com.hackthebox.myapp

----
/_ _ |  Frida 16.1.11 - A world-class dynamic instrumentation toolkit
|(_|_|
> _ _ |  Commands:
/_/ |_|    help      -> Displays the help system
. . . .    object?   -> Display information about 'object'
. . . .    exit/quit -> Exit
. . . .
. . . .    More info at https://frida.re/docs/home/
. . . .
. . . .    Connected to Android Emulator 5554 (id=emulator-5554)
Spawned `com.hackthebox.myapp`. Resuming main thread!
[Android Emulator 5554::com.hackthebox.myapp ]->

[*] Enumerating methods of class: com.hackthebox.myapp.MainActivity
<SNIP
private void com.hackthebox.myapp.MainActivity.checkAnswer()
private void com.hackthebox.myapp.MainActivity.finishQuiz()
private boolean com.hackthebox.myapp.MainActivity.isAppSignatureValid()
private void com.hackthebox.myapp.MainActivity.startTimer()
private void com.hackthebox.myapp.MainActivity.updateQuestion()
private void com.hackthebox.myapp.MainActivity.updateScoreView()
public native java.lang.String com.hackthebox.myapp.MainActivity.h14s76()
protected void com.hackthebox.myapp.MainActivity.onCreate(android.os.Bundle)


[*] Enumerating methods of class: com.hackthebox.myapp.TriviaManager
private void com.hackthebox.myapp.TriviaManager.populateQuestions()
public boolean com.hackthebox.myapp.TriviaManager.areQuestionsExhausted()
public boolean com.hackthebox.myapp.TriviaManager.checkAnswer(java.lang.String,java.lang.String)
public java.lang.String com.hackthebox.myapp.TriviaManager.getQuestion()
void com.hackthebox.myapp.TriviaManager.resetQuestions()

<SNIP>
[*] Enumerating methods of class: com.hackthebox.myapp.ScoreManager
public int com.hackthebox.myapp.ScoreManager.getScore()

```

```
public int com.hackthebox.myapp.ScoreManager.resetScore()
public void com.hackthebox.myapp.ScoreManager.updateScore(boolean)
```

The script successfully lists the three Java classes we previously identified using JADX, along with their methods. Based on its name, `getScore()` in the `ScoreManager` class appears to be the method responsible for returning the score. Our next step is to write a Frida script to modify this method's return value and assign a higher score. While dynamic enumeration offers a faster way to discover classes and methods, combining it with static analysis provides a more comprehensive and effective testing strategy.



Connect to Pwnbox
Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

UK

36ms

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

☐

Enable step-by-step solutions for all questions ⓘ ✨

Questions

Answer the question(s) below to complete this Section and earn cubes!

+ 5 📦 What message is displayed upon completing the game?

Submit your answer here...

+10 Streak pts

Submit

alter_method_value.zip

Table of Contents

Enumerating and Exploiting Installed Apps

Introduction
Enumerating Local Storage
Exported Activities
Insecure Logging
Pending Intents
Exploiting WebViews
Insecure Library Load Through Deep Linking

Dynamic Code Instrumentation

Hooking Java Methods
Altering Method Values
Hooking Native Methods
Bypassing Detection Mechanisms
Authentication Token Manipulation

Intercepting HTTP/HTTPS Requests

Intercepting API Calls
IDOR Attack
SSL/TLS Certificate Pinning Bypass

Skills Assessments

Skills Assessment

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left