

PSG COLLEGE OF TECHNOLOGY
DEPARTMENT OF APPLIED MATHEMATICS AND
COMPUTATIONAL SCIENCES
20XC87 – MOBILE SECURITY LAB
PACKAGE REPORT

PACKAGE TITLE:
CRICKET CTF CHALLENGE

Team :

- 21PC17 – JYOTHISH K S
- 21PC19 – NANDA
PRANESH S
- 21PC25 – VARUN S

Package Type : CTF CHALLENGE

Date of submission : 03-04-2025

INTRODUCTION

Overview

CricketCTF is an Android-based Capture The Flag (CTF) application designed to test and enhance cybersecurity skills through a series of 12 cricket-themed challenges. Developed as an educational tool, it combines mobile app security concepts with engaging gameplay, requiring participants to employ techniques such as reverse engineering, dynamic analysis, SQL injection, and cryptography to uncover hidden flags. Each challenge simulates real-world vulnerabilities, offering hands-on experience in identifying and exploiting them within a controlled environment.

The app's challenges range from simple hardcoded flag extraction to advanced native library analysis, reflecting a progression of difficulty and skill requirements. This walkthrough provides a detailed guide to solving all 12 challenges, ensuring participants can follow along and understand the underlying security principles.

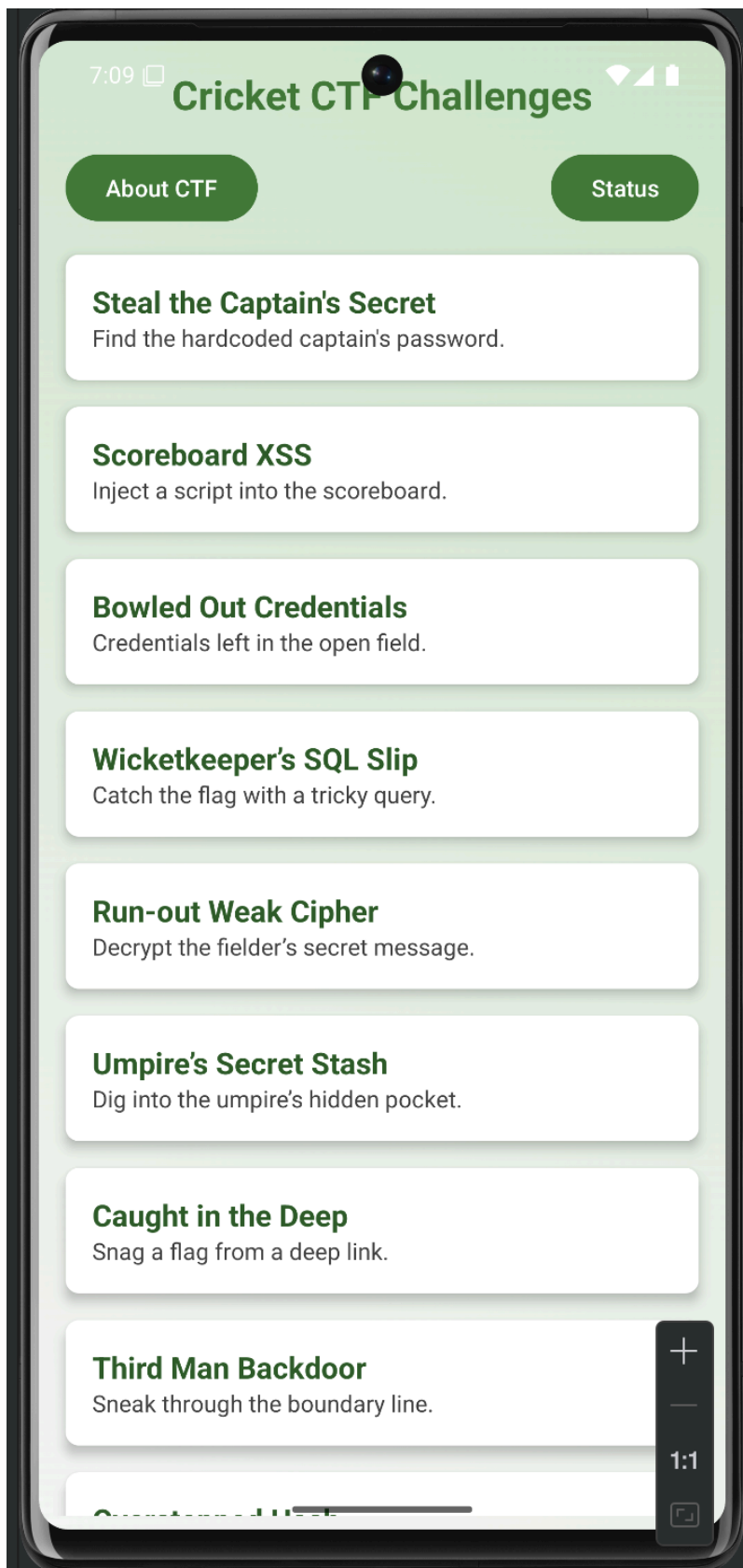
Pre-requisites

To complete this walkthrough, participants will need the following tools and setup:

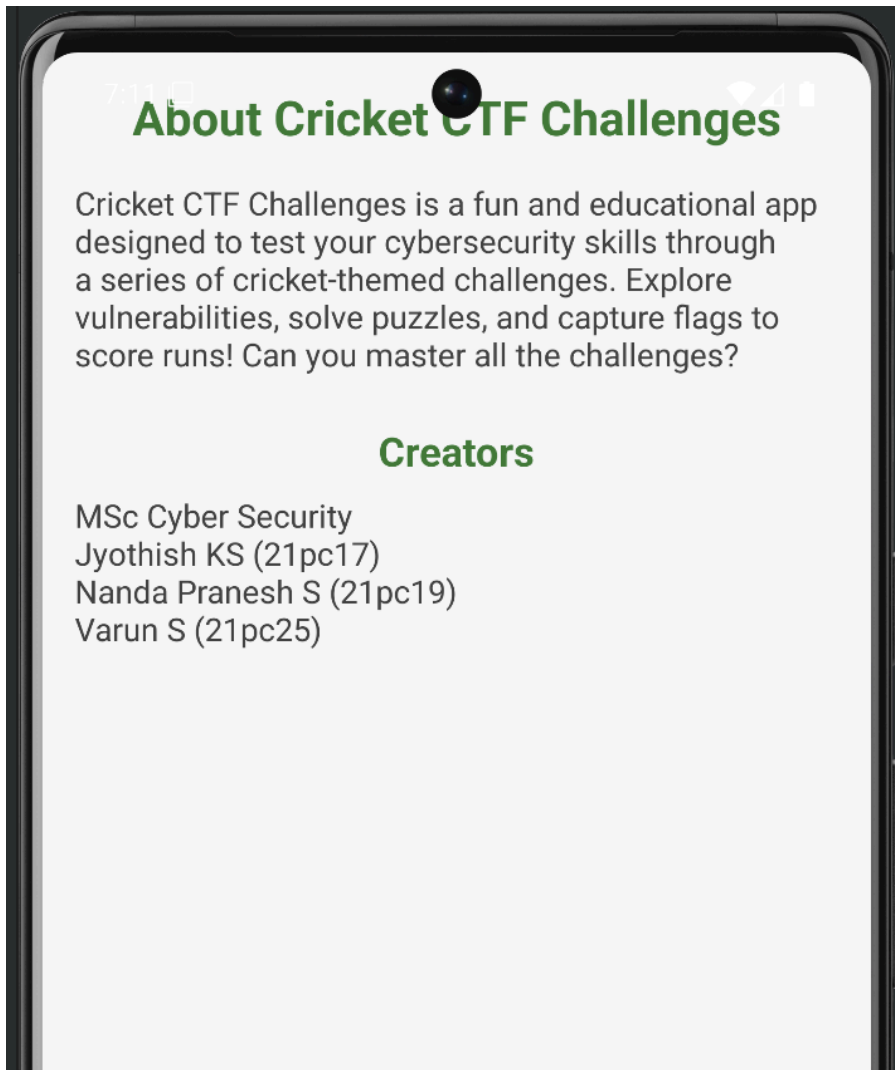
- **Android Emulator:** A virtual device (e.g., Pixel 6, API 30, armeabi-v7a) via Android Studio's AVD Manager for easy file access and root privileges.
- **ADB (Android Debug Bridge):** Installed with Android Studio or standalone, for interacting with the emulator (e.g., adb shell, adb pull).
- **Frida:** Dynamic analysis tool for hooking into app functions (install via pip install frida-tools and download Frida server for the emulator's architecture).
- **JADX-GUI:** APK decompiler to inspect Kotlin code (download from github.com/skylot/jadx).
- **Ghidra:** Reverse-engineering tool for analyzing native libraries (download from ghidra-sre.org).
- **SQLite Browser:** Tool like DB Browser for SQLite (sqlitebrowser.org) or sqlite3 to view database files.
- **CyberChef:** Web-based tool for decoding data (gchq.github.io/CyberChef/).

Setup:

1. Install the apk on the emulator: adb install cricketCTF.apk.
2. Launch the app to access the list of challenges.



Clicking on About CTF, we get to see the below details about it.



Challenges and Solutions

Challenge 1: Steal the Captain's Secret

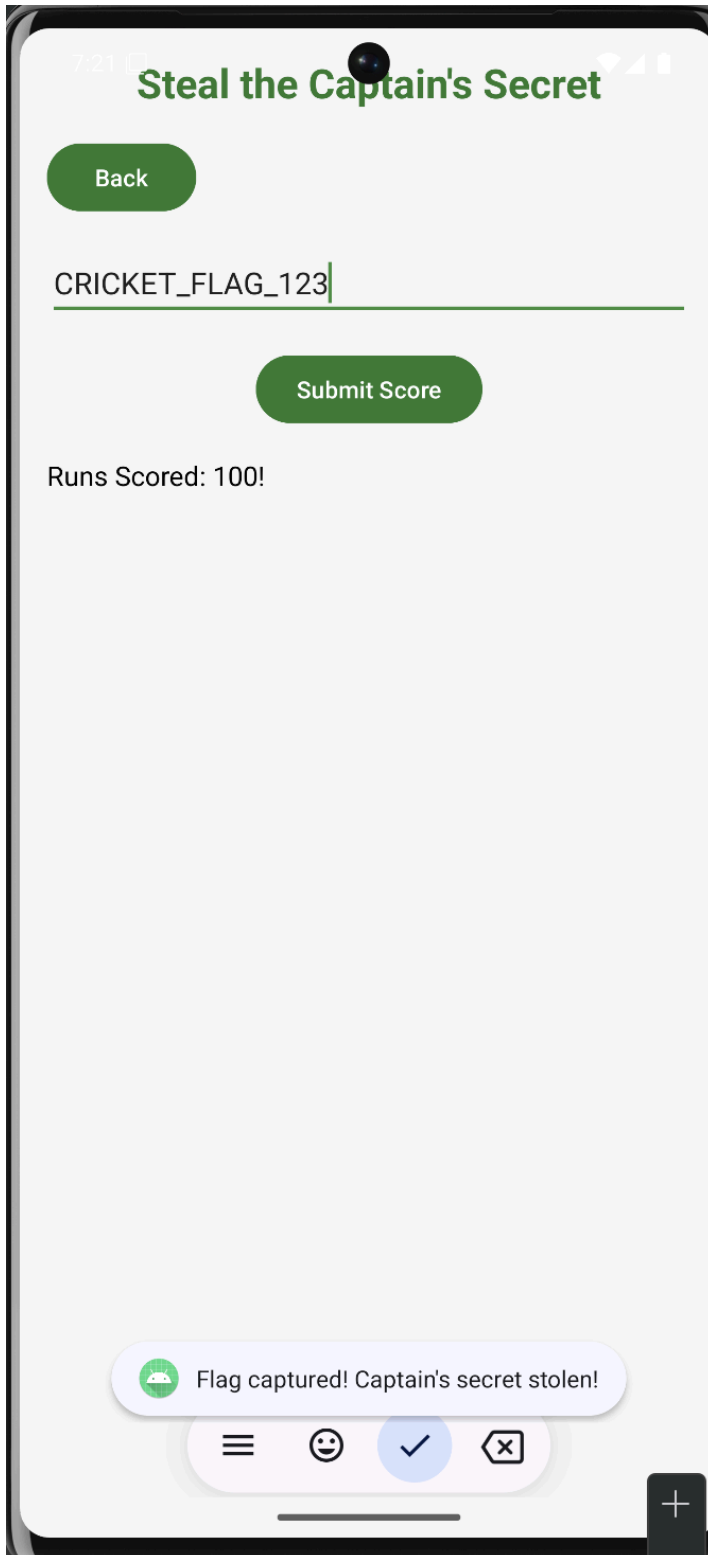
Objective: Find the hardcoded captain's password.

Solution:

Try giving random input and click submit.

```
private final String captainSecret = "CRICKET_FLAG_123";
```

Entering this flag into the challenge, we see the following completion message.



Flag: CRICKET_FLAG_123

Challenge 2: Scoreboard XSS

Objective: Inject a script into the scoreboard.

Solution: Entering a random malicious script, we can see that the input is getting displayed.

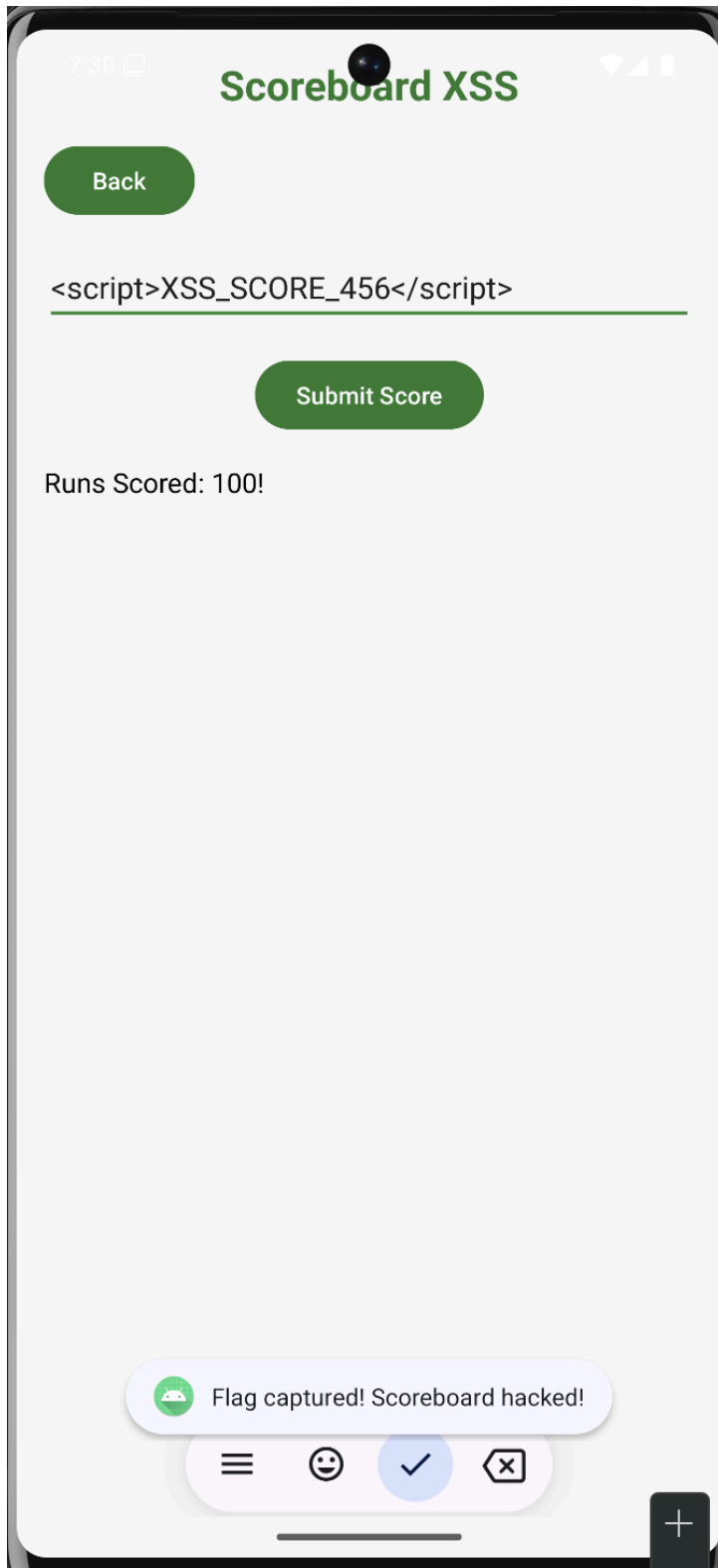


Let's open this in JADX and try to analyze. In the search option, let's put xss and see if any pops up.

```
private final String xssFlag = "XSS_SCORE_456";
```

Thus, we can see the flag here when we use the search option. Now, since we have to inject something into the input box, let's curate our input string in the following way.

```
<script>XSS_SCORE_456</script>
```



Flag: XSS_SCORE_456

CHALLENGE STATUS:

7:31

Challenge Status

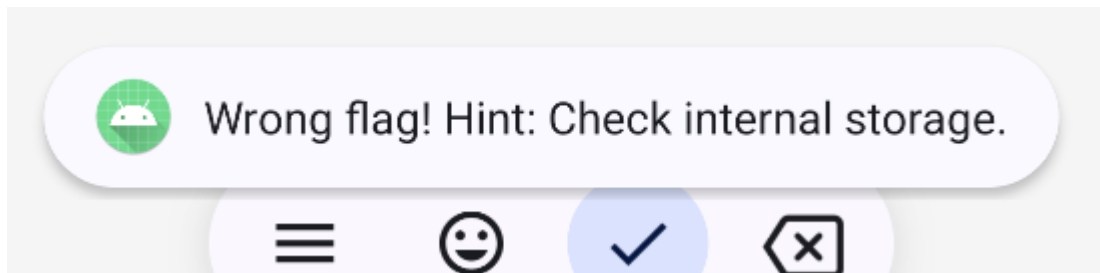
Back

Steal the Captain's Secret	Completed
Scoreboard XSS	Completed
Bowled Out Credentials	Not Completed
Wicketkeeper's SQL Slip	Not Completed
Run-out Weak Cipher	Not Completed
Umpire's Secret Stash	Not Completed
Caught in the Deep	Not Completed
Third Man Backdoor	Not Completed
Overstepped Hash	Not Completed
Frida Boundary Break	Not Completed
Logcat Leak	Not Completed
Ghidra Reverse Wicket	Not Completed

Challenge 3: Bowled Out Credentials

Objective: Find credentials in internal storage.

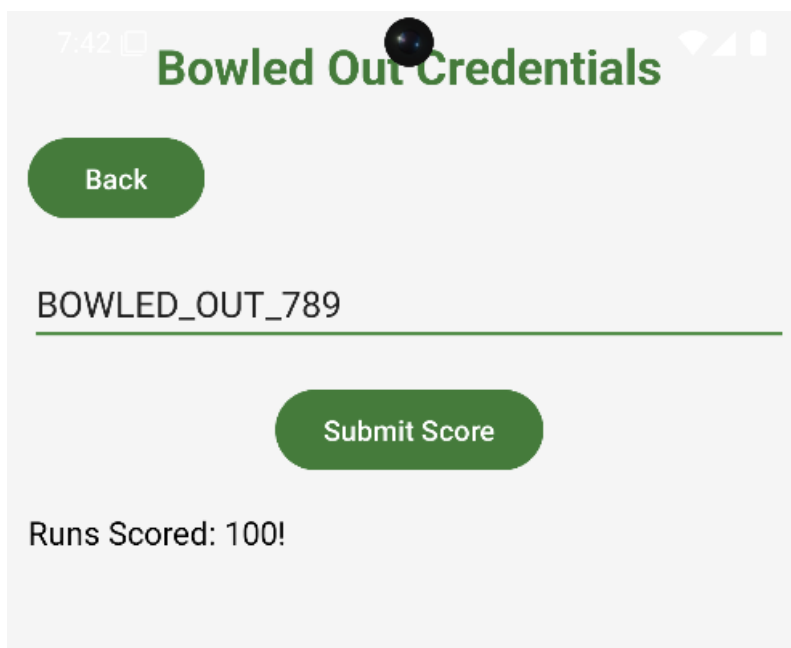
Solution: Giving random input gives the following hint. It asks us to check internal storage.



So, let's use ADB to access the app's files:

```
varunselvaraj@Varuns-MacBook-Pro CricketCTF % adb shell
emu64a:/ # cd /data/data/com.example.cricketctf
emu64a:/data/data/com.example.cricketctf # ls
cache  code_cache  databases  files  shared_prefs
emu64a:/data/data/com.example.cricketctf # cd files
emu64a:/data/data/com.example.cricketctf/files # ls
profileInstalled  scoreboard.txt
emu64a:/data/data/com.example.cricketctf/files # cat scoreboard.txt
Player: Admin, Secret: BOWLED_OUT_789emu64a:/data/data/com.example.cricketctf/files #
```

We navigate into the files of the app and find the flag within the scoreboard.txt file.



Flag: BOWLED_OUT_789

Challenge 4: Wicketkeeper's SQL Slip

Objective: Exploit an SQL injection vulnerability.

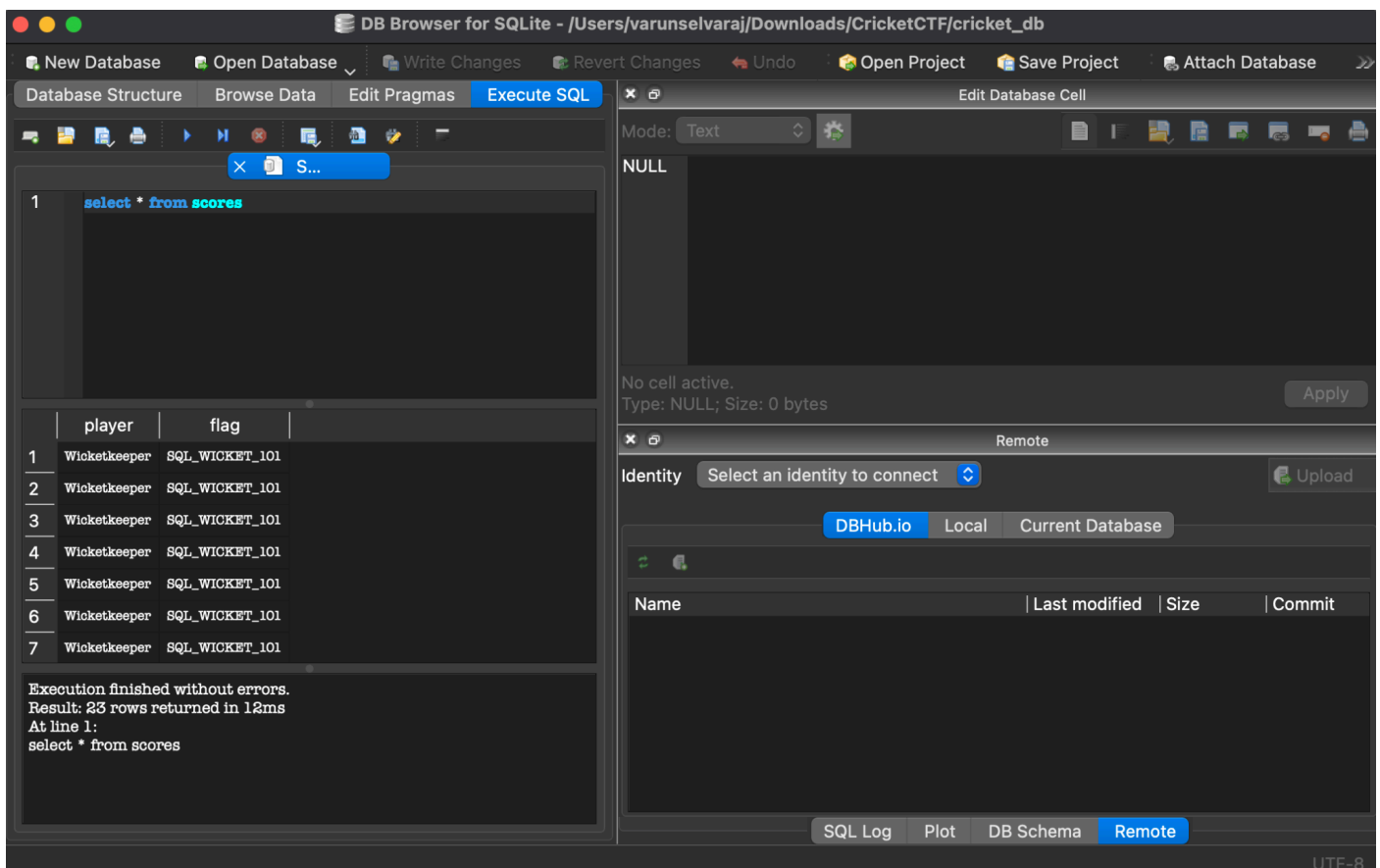
Solution: Since this challenge is based on SQL querying, there should be a table we are querying from in the database. Let's find that table using adb.

```
emu64a:/data/data/com.example.cricketctf/files # cd /data/data/com.example.cricketctf/
emu64a:/data/data/com.example.cricketctf # ls
cache  code_cache  databases  files  shared_prefs
emu64a:/data/data/com.example.cricketctf # cd databases
emu64a:/data/data/com.example.cricketctf/databases # ls
cricket_db  cricket_db-journal
```

We can see the cricket_db inside the databases, let's pull that db.

```
varunselvaraj@Varuns-MacBook-Pro CricketCTF % adb pull /data/data/com.example.cricketctf/databases/cricket_db ./cricket_db
/data/data/com.example.cricketctf/databases/cricket_db: 1 file pulled, 0 skipped. 3.3 MB/s (16384 bytes in 0.005s)
varunselvaraj@Varuns-MacBook-Pro CricketCTF %
```

Now, we open the table in SQLite Browser and query to find out the details present.



The screenshot shows the DB Browser for SQLite application. The 'Execute SQL' tab is active, displaying the query `select * from scores`. The results are shown in a table with two columns: 'player' and 'flag'. There are 7 rows of data, all showing 'Wicketkeeper' as the player and 'SQL_WICKET_101' as the flag. The status bar at the bottom indicates 'Execution finished without errors. Result: 23 rows returned in 12ms. At line 1: select * from scores'.

	player	flag
1	Wicketkeeper	SQL_WICKET_101
2	Wicketkeeper	SQL_WICKET_101
3	Wicketkeeper	SQL_WICKET_101
4	Wicketkeeper	SQL_WICKET_101
5	Wicketkeeper	SQL_WICKET_101
6	Wicketkeeper	SQL_WICKET_101
7	Wicketkeeper	SQL_WICKET_101

We can see the flag in the query result.

Flag: SQL_WICKET_101

Challenge 5: Run-out Weak Cipher

Objective: Decrypt a Caesar cipher message.

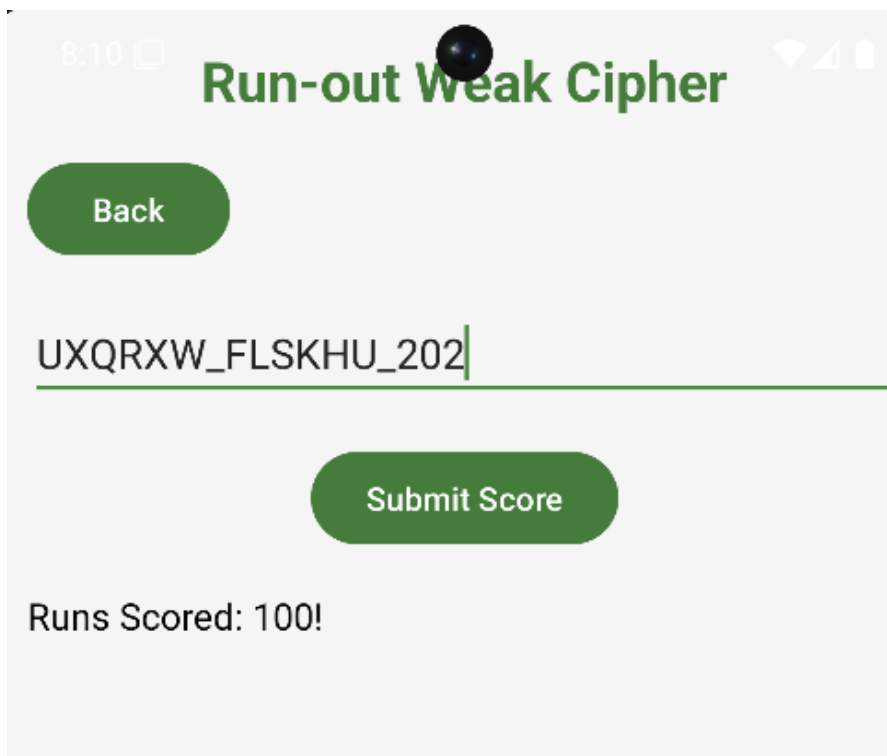
Solution: If we search for runout within the apk, we can find the flag.

```
private final String runoutCipherFlag = "RUNOUT_CIPHER_202";
```

But, if we put the flag into the challenge, it will not work. Checking further into the code, we find the following function.

```
if (challengeName.equals("Run-out Weak Cipher")) {  
    String decrypted = caesarDecrypt(input, 3);  
    if (Intrinsics.areEqual(decrypted, this.runoutCipherFlag)) {  
        showSuccess(R.string.run_out_success, challengeName);  
        return;  
    } else {  
        showError(R.string.run_out_error);  
        return;  
    }  
}  
break;
```

Here, we can see that Caesar cipher is used and we have to encrypt the found flag and then give it as input. So, we have to shift all the characters by 3 and then give them as the flag.



Flag: RUNOUT_CIPHER_202 (UXQRXW_FLSKHU_202)

Challenge 6: Umpire's Secret Stash

Objective: Find a flag in SharedPreferences.

Solution: Giving random input, it prompts us to check the SharedPreferences.



Wrong flag! Hint: Check SharedPreferences.

So let's check it using adb.

```
emu64a:/ # cd /data/data/com.example.cricketctf/
emu64a:/data/data/com.example.cricketctf # ls
cache  code_cache  databases  files  shared_prefs
emu64a:/data/data/com.example.cricketctf # cd shared_prefs
emu64a:/data/data/com.example.cricketctf/shared_prefs # ls
challenge_status.xml  umpire_prefs.xml
emu64a:/data/data/com.example.cricketctf/shared_prefs # cat umpire_prefs.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="umpire">UMPIRE_STASH_303</string>
</map>
```

Thus, we have found the flag within the xml file.

Flag: UMPIRE_STASH_303

Challenge 7: Caught in the Deep

Objective: Retrieve a flag via a deep link.

Solution: Look at the hint for this challenge.



Wrong flag! Hint: Try a deep link like 'cricketctf://catch'.

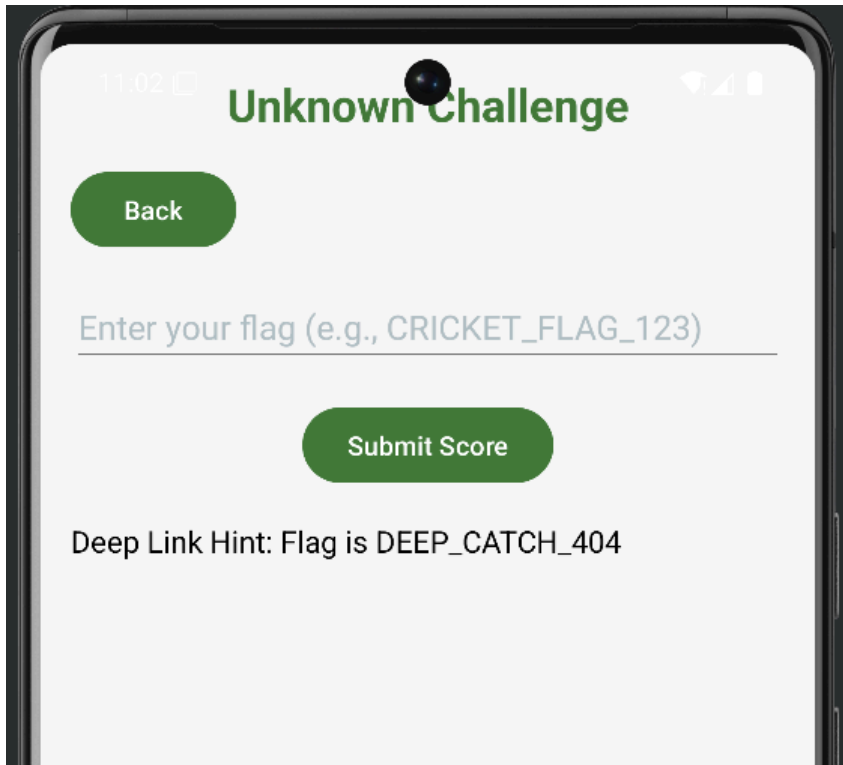


Now, let's perform this by creating a new intent.

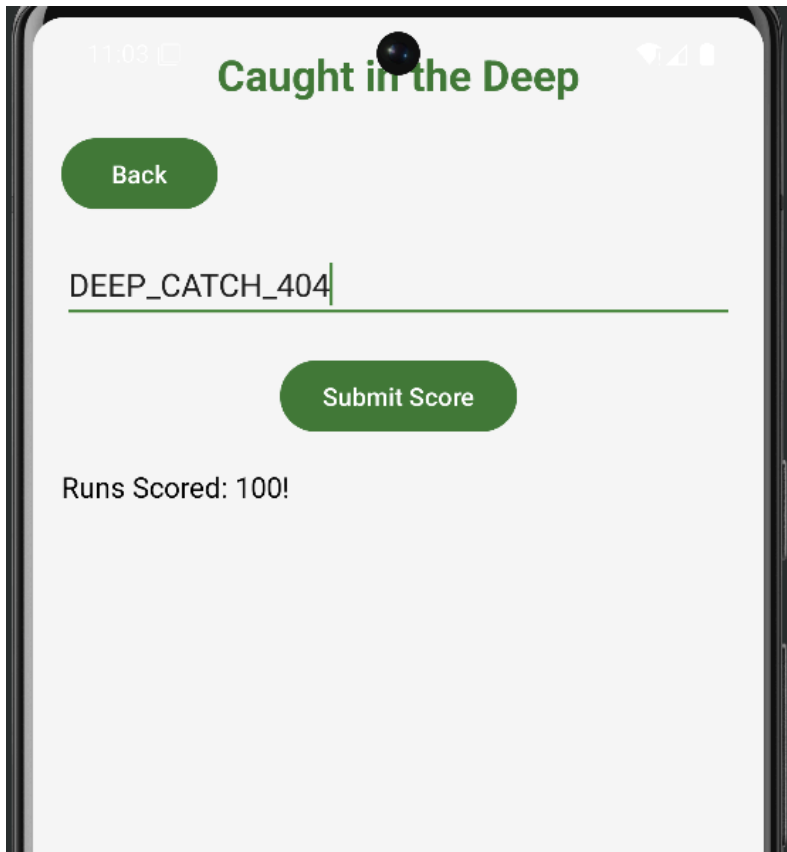
```
varunselvaraj@Varuns-MacBook-Pro CricketCTF % adb shell am start -W -a android.intent.action.VIEW -d "cricketctf://catch" com.example.cricketctf
Starting: Intent { act=android.intent.action.VIEW dat=cricketctf://catch/... pkg=com.example.cricketctf }
Status: ok
LaunchState: WARM
Activity: com.example.cricketctf/.ChallengeActivity
TotalTime: 55
WaitTime: 59
Complete
```

We can see that the intent has been created successfully.

So, after this, our UI will be redirected to a different intent, which will display the following flag.



Putting the flag in the challenge,

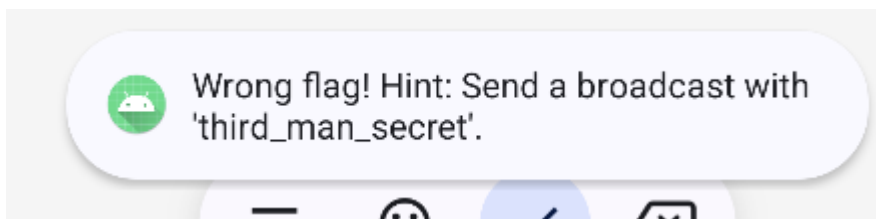


Flag: DEEP_CATCH_404

Challenge 8: Third Man Backdoor

Objective: Exploit a broadcast receiver.

Solution: Look at the hint for this challenge.



Which means that the app has a `BackdoorReceiver` that listens for a broadcast with the action `com.example.cricketctf.THIRD_MAN_SECRET`. When the broadcast is received, it displays the flag in a Toast message.

```
varunselvaraj@Varuns-MacBook-Pro CricketCTF % adb shell am broadcast -a com.example.cricketctf.THIRD_MAN_SECRET -n com.example.cricketctf/.BackdoorReceiver
Broadcasting: Intent { act=com.example.cricketctf.THIRD_MAN_SECRET flg=0x400000 cmp=com.example.cricketctf/.BackdoorReceiver }
Broadcast completed: result=0
varunselvaraj@Varuns-MacBook-Pro CricketCTF %
```

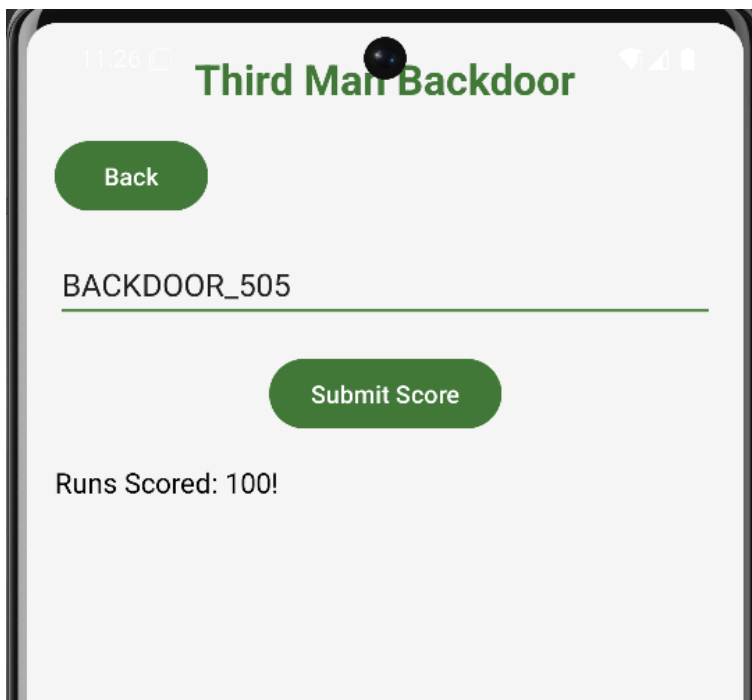
Now, checking the logcat, we can see the flag.

```

I Broadcasting: Intent { act=com.example.cricketctf.THIRD_MAN_SECRET flg=0x400000 cmp=co
I Enqueued broadcast Intent { act=com.example.cricketctf.THIRD_MAN_SECRET flg=0x400000 c
D Backdoor Hint: Flag is BACKDOOR_505
D Challenge 8: Third Man Backdoor
D Challenge 8: Third Man Backdoor
D Binding challenge at position 7: Third Man Backdoor
I Broadcasting: Intent { act=com.example.cricketctf.THIRD_MAN_SECRET flg=0x400000 cmp=co
I Enqueued broadcast Intent { act=com.example.cricketctf.THIRD_MAN_SECRET flg=0x400000 c
D Backdoor Hint: Flag is BACKDOOR_505

```

Putting it in the challenge,

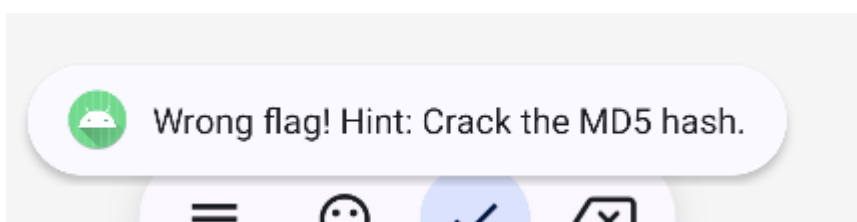


Flag: BACKDOOR_505

Challenge 9: Overstepped Hash

Objective: Crack an MD5 hash check.

Solution: Finding the hint from the challenge,



We can see that it hints something related to the MD5 hash. Let's take a closer look at the function present within the apk file in JADX-GUI.

```

if (challengeName.equals("Overstepped Hash")) {
    String hashedInput = md5Hash(input);
    if (Intrinsics.areEqual(hashedInput, md5Hash(this.hashOverFlag))) {
        showSuccess(R.string.hash_over_success, challengeName);
        return;
    } else {
        showError(R.string.hash_over_error);
        return;
    }
}
break;

```

From this, we can understand that our input is MD5 hashed and then compared with the MD5 hash of the actual flag stored and then succeeds if the flag is correct or rejects if the flag is wrong.

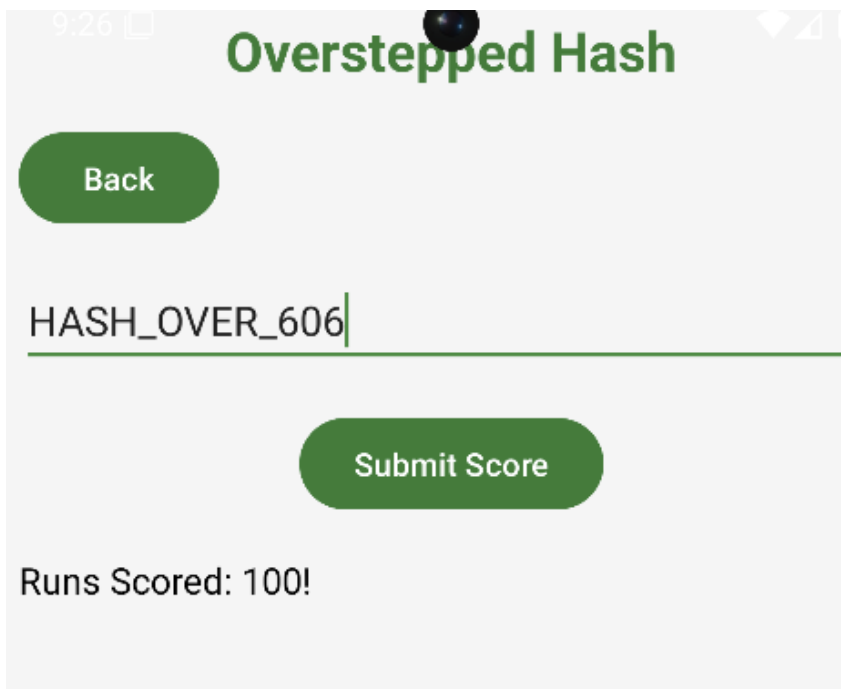
Searching through the apk file in JADX, we can find

```

private final String hashOverFlag = "HASH_OVER_606";

```

Putting this flag in the challenge will complete it.

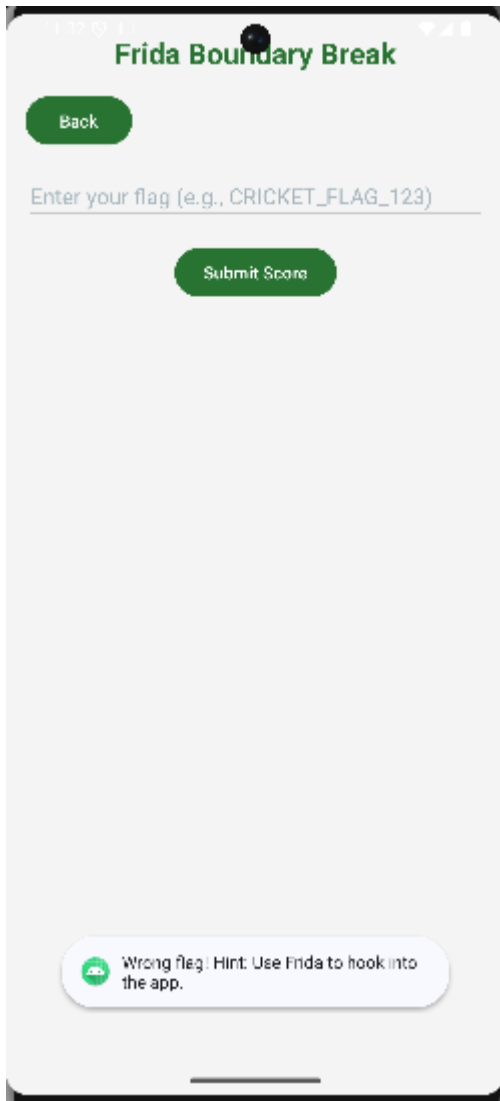


Flag: HASH_OVER_606

Challenge 10 : Frida Boundary Break

Objective : Break the boundary with Frida

Solution: Upon entering a random string, we can find the hint,



Upon uploading the apk to jadx GUI, we can find the function responsible for the flag:

```
private final String getFridaFlag() {  
    return System.currentTimeMillis() % ((long) 2) == 0 ? "FRIDA_BREAK_707" : "WRONG_FLAG";  
}
```

As per the function, we have the flag, but we need to satisfy the condition. This can be done by hooking a javascript function that bypasses the condition and forces the flag to be correct. The below is the javascript function that will be hooked through frida:

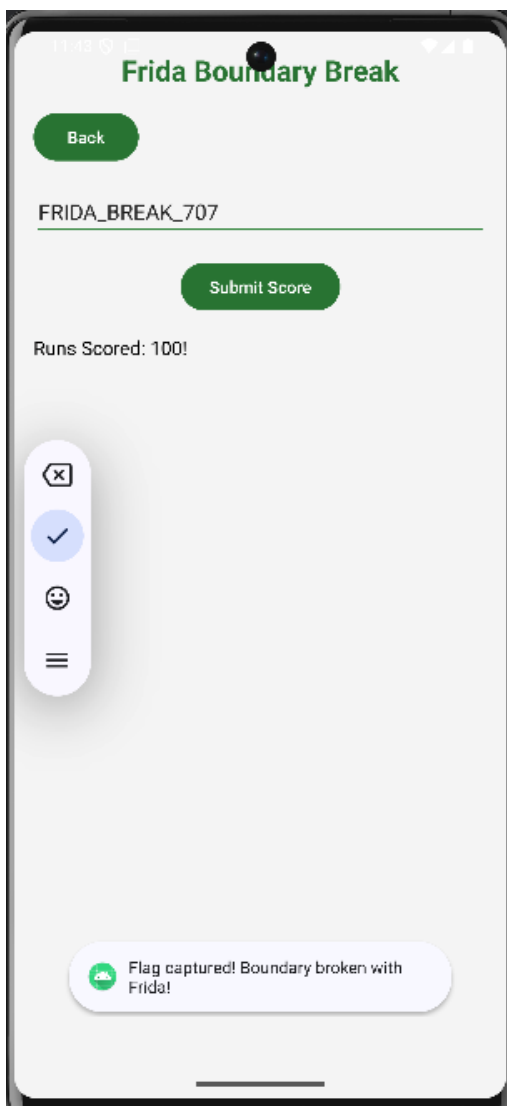
```
hook.js  
File Edit View  
  
Java.perform(function () {  
    var ChallengeActivity = Java.use("com.example.cricketctf.ChallengeActivity");  
    ChallengeActivity.getFridaFlag.implementation = function () {  
        console.log("Hooked getFridaFlag!");  
        return "FRIDA_BREAK_707"; // Force the method to return the flag  
    };  
});
```

Now we will run the script with the help of frida, as shown below:

```
PS C:\Users\csuser\Downloads\CricketCTF 2\CricketCTF> frida -U -f com.example.cricketctf -l hook.js

----
/ _ |   Frida 16.5.7 - 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.example.cricketctf`. Resuming main thread!
[Android Emulator 5554::com.example.cricketctf ]->
```

And now when we run the flag, we can see that we have passed:

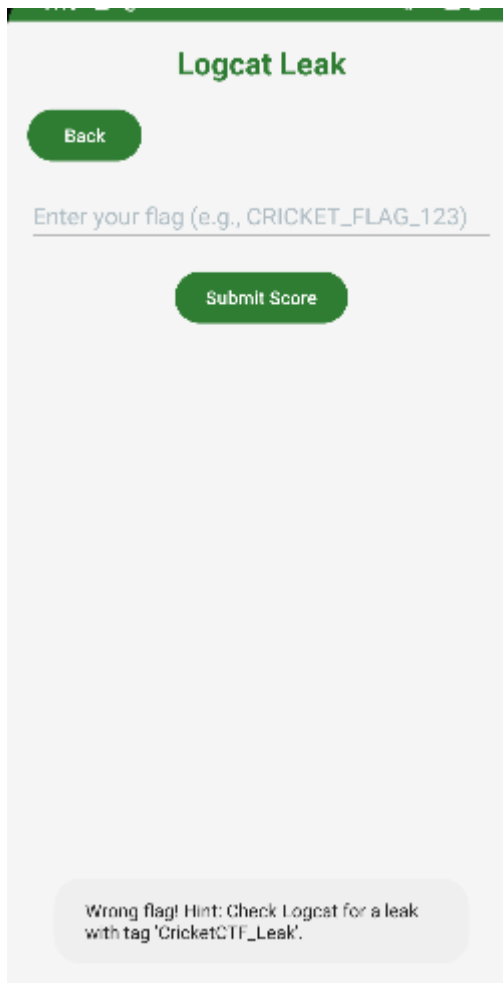


Flag: FRIDA_BREAK_707

Challenge 11: Logcat Leak

Objective: Find the flag leaking in the logs.

Solution: Upon entering a random string, we can find the hint, asking us to check logcat for leak with tag for leak with the tag 'CricketCTF_Leak'



Logcat Leak

Back

Enter your flag (e.g., CRICKET_FLAG_123)

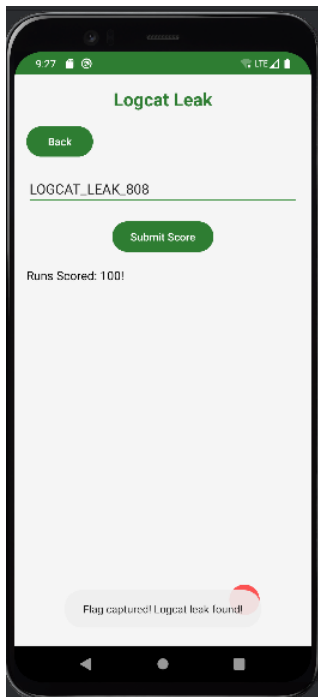
Submit Score

Wrong flag! Hint: Check Logcat for a leak with tag 'CricketCTF_Leak'.

So when we look for flag in logcat with the tag given in the hint, we can see the following:

```
04-03 09:12:44.135 12408 12408 D CricketCTF_Leak: Sensitive data leaked: LOGCAT_LEAK_808
```

And here, we got the flag



Flag: LOGCAT_LEAK_808

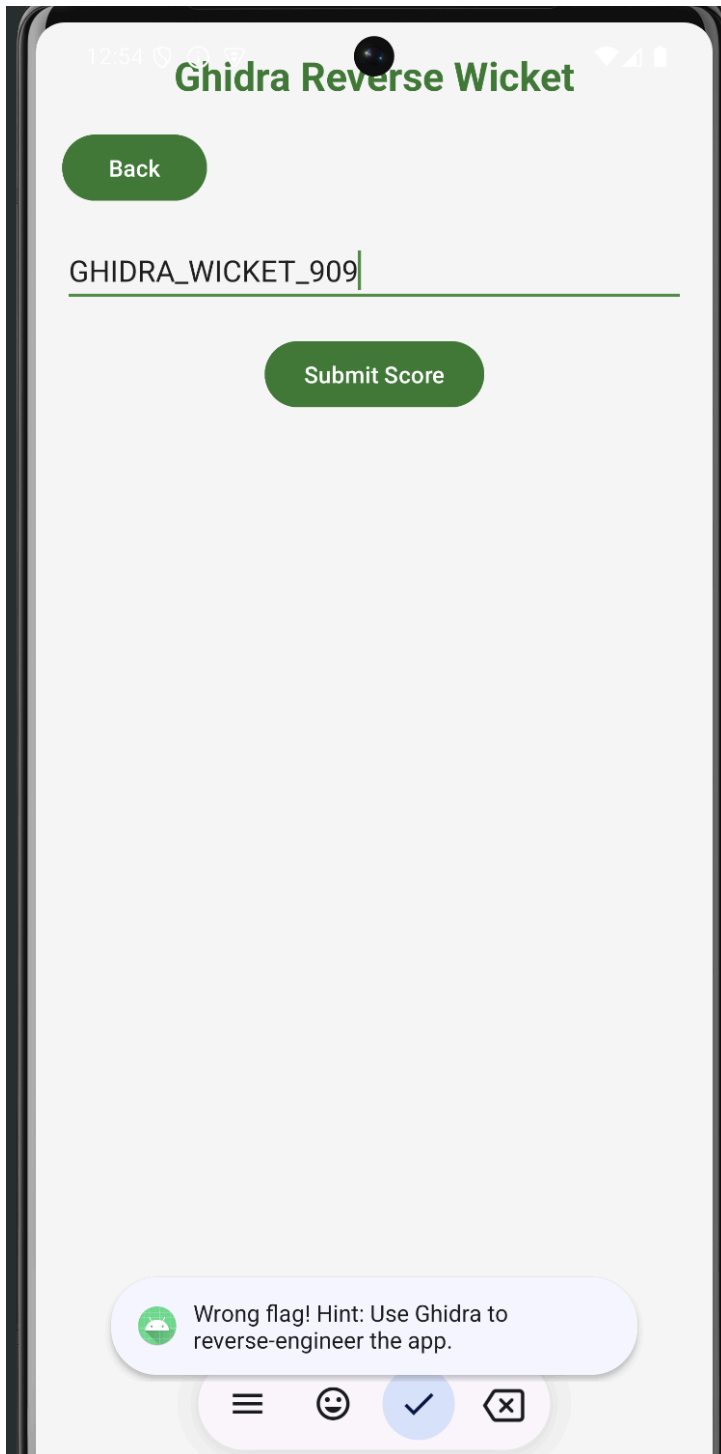
Challenge 12: Ghidra Reverse Wicket (Tough to solve)

Objective: Reverse Engineering using Ghidra and Frida

Solution: Using JADX to decompile the apk, we find a function `getGhidraFlag()` which returns true if the flag is of length 15.

```
private fun getGhidraFlag(): String {  
    val hiddenFlag = decodeFlag(ghidraWicketFlagPart1, ghidraWicketFlagPart2)  
    return if (hiddenFlag.length == 15) hiddenFlag else "WRONG_FLAG"  
}
```

If we try to decode the hidden flag, it comes to GHIDRA_WICKET_909. But putting this in the challenge won't solve it since the length of the flag is 17.



You can solve this challenge by modifying the condition of the function using Ghidra and Frida and then recompile the apk and then give the flag to solve the challenge.

1:06



Ghidra Reverse Wicket

Back

GHIDRA_WICKET_909

Submit Score

Runs Scored: 100!

This challenge is tough as it might be difficult to configure Ghidra on your system.

Flag: GHIDRA_WICKET_909