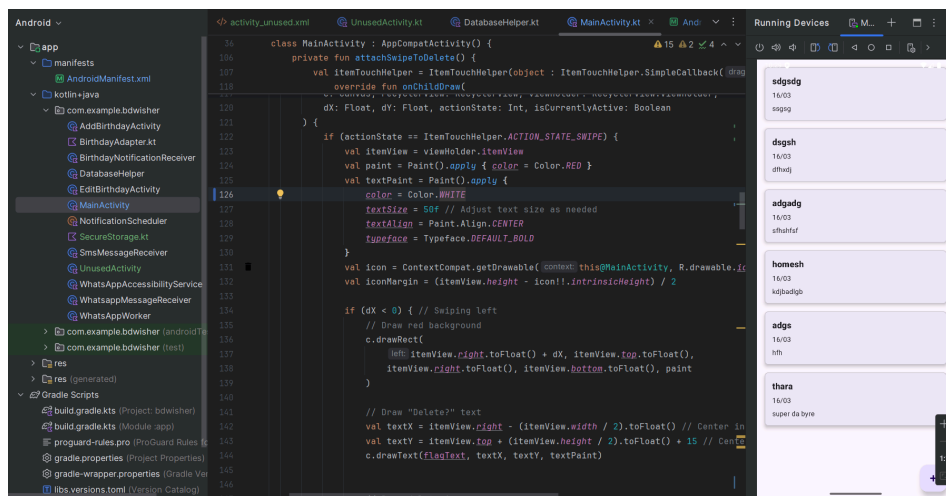


MOBILE SECURITY PACKAGE WALKTHROUGH

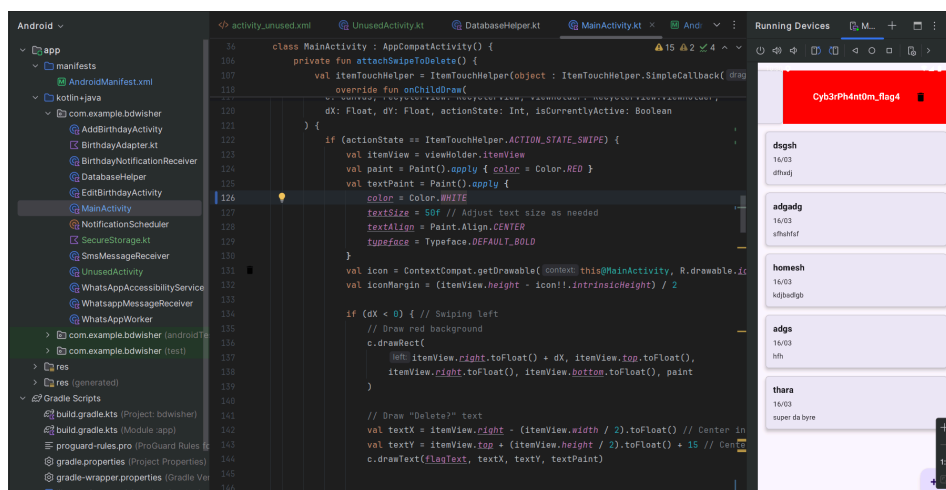
21PC14 - Hari Ganesh M
21PC22 - Preetham Bavan E
21PC32 - Sateesh Kumar RS

Flag 4:

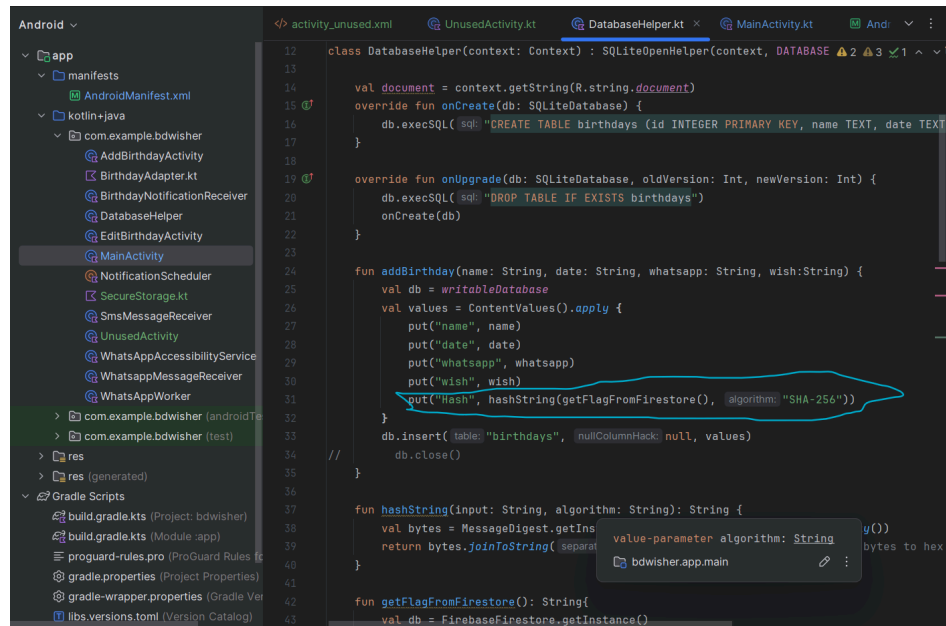


The above image shows the Home screen UI where the list of all the birthdays is shown. A swipe to left option is provided deleting a particular birthday.

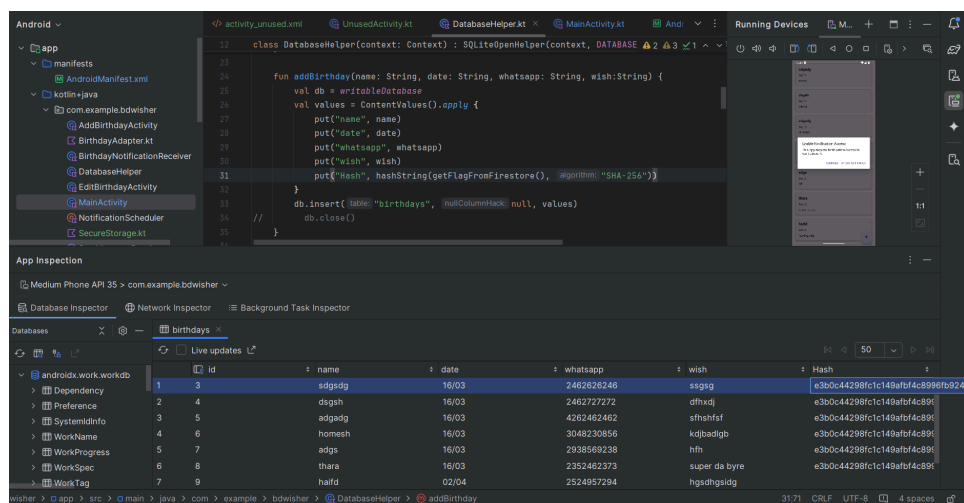
The below image shows the deleting part. Here is where Flag 4 is placed. So the CTF player should analyse the UI code to look for this flag.



Flag 1:



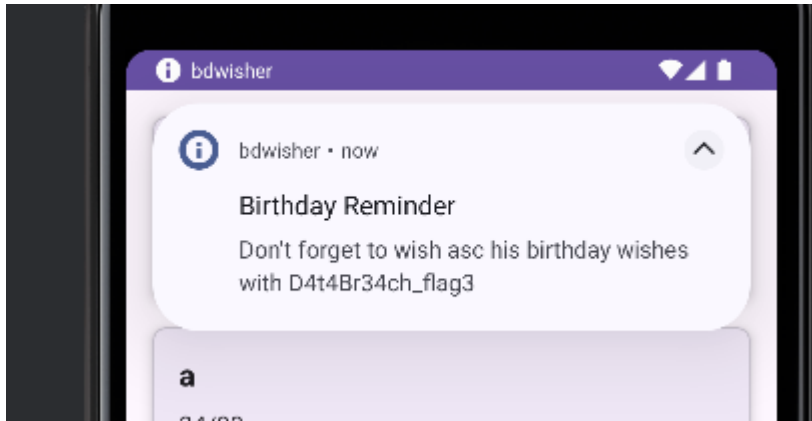
The above image shows the Hash getting stored for every record in the DataBaseHelper.kt file. This is the hash of the Flag 1 which can be viewed from the below image. We want this hash to be submitted as Flag 1.



Flag 2:

```
2025-04-02 21:57:08.482 11507-11507 NotificationScheduler com.example.bdwysher D Flag value: Sh4d0wH4ck3r_flag2
2025-04-02 21:57:08.486 11507-11507 WindowOnBackDispatcher com.example.bdwysher W sendCancelIfRunning: isInProgress=falsecallback=ImeCallba
2025-04-02 21:57:08.487 658-2003 CoreBackPreview system_server D Window{6f9c0a4 u0 com.example.bdwysher/com.example.bdwysh
2025-04-02 21:57:08.487 658-3089 CoreBackPreview system_server D Window{b66f6c0 u0 com.example.bdwysher/com.example.bdwysh
2025-04-02 21:57:08.502 658-2003 ImeTracker system_server I com.example.bdwysher:cfe573cb: onRequestHide at ORIGIN_SE
```

Flag 3:



Both the above flags can be obtained by changing the time difference returned by a function called `getTime()` in `NotificationScheduler.kt` file. This app basically sends a reminder notification for every birthday 4 hours prior to the birthdate. In order to obtain this notification which contains the flag the CTF player should change the 4 hours to the required time difference to obtain this notification.

Along with the Flag 3 in the notification you will also get the Flag 2 in the notification Logs.

Frida Script:

```
JS code.js x Release Notes: 1.97.2
C: > Users > preet > Desktop > PACKAGES > MOB_SEC > RaptorAndroidCTF > app > JS code.js > ...
1  Java.perform(()=> {
2      console.log("[*] Hooking NotificationScheduler...");
3
4
5      var NotificationScheduler = Java.use("com.example.bdwysher.NotificationScheduler");
6
7      NotificationScheduler.getTime.implementation = function() {
8          return 123 * 60 * 1000;
9      };
10 }
```

Flag 5:

```
private fun loadBirthdays() {
    val birthdays = dbHelper.getAllBirthdays().toMutableList()
    birthdayAdapter = BirthdayAdapter(birthdays) { birthday ->
        val intent = Intent(packageContext: this, EditBirthdayActivity::class.java)
        intent.putExtra(name: "BIRTHDAY_ID", birthday.id)
        startActivity(intent)
    }
    recyclerView.adapter = birthdayAdapter

    // Schedule all birthday reminders
    scheduleBirthdayReminders()

    // Start service for each birthday
    for (birthday in birthdays) {
        Log.d(tag: "Info PASSED", msg: "${birthday.date} ${birthday.id} ${birthday.name}")
    }

    if (birthdays.size == 7) {
        getFlagFromFirestore(context: this, index: 5) { flag ->
            storeFlagInSharedPreferences(flag)
        }
    }
}
```

In the loadBirthdays function, we can see that if birthdays.size==7 getFlagFromFirestore is called and the flag is passed to storeFlagInSharedPreferences.

```
public fun storeFlagInSharedPreferences(flag: String) {
    val sharedPreferences = getSharedPreferences(name: "Flags", Context.MODE_PRIVATE)
    with(sharedPreferences.edit()) {
        putString("flag_5", flag)
        apply()
    }
    Log.d(tag: "SharedPref", msg: "Flag stored: $flag")
}
```

In the storeFlagInSharedPreferences function we can see that the flag is stored in the shared preference named 'Flags'.

After creating 7 birthday items we can check if the shared preference named 'Flags' is created.

▼ com.example.bdwisher	drwx-----	2025-03-24 20:13	4 KB
> cache	drwxrws--x	2025-03-24 20:13	4 KB
> code_cache	drwxrws--x	2025-04-02 22:07	4 KB
> databases	drwxrwx--x	2025-03-24 20:13	4 KB
> files	drwxrwx--x	2025-04-02 21:37	4 KB
> no_backup	drwxrwx--x	2025-03-24 20:13	4 KB
▼ shared_prefs	drwxrwx--x	2025-04-02 22:05	4 KB
</> BirthdayPrefs.xml	-rw-rw----	2025-04-02 21:57	201 B
</> com.google.android.gms.appid.xml	-rw-rw----	2025-04-02 20:59	364 B
</> com.google.android.gms.measurement.prefs.xml	-rw-rw----	2025-04-02 22:05	846 B
</> com.google.firebase.messaging.xml	-rw-rw----	2025-03-24 20:13	137 B
</> FirebaseHeartBeatW0RFRkFVTFRd+MT04MDU	-rw-rw----	2025-04-02 21:32	720 B
</> Flags.xml	-rw-rw----	2025-04-02 21:25	120 B
> com.example.datedifference	drwx-----	2025-01-18 00:50	4 KB

Using device explorer we can check the directory '/data/data/package-name/shared_prefs' For the shared preference named 'Flags'. We can see that it has been created.

```
Flags.xml x  surpriseProvider.kt  UnusedActivity.kt  WhatsAppAccessibilityS

<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="flag_5">R00tK!tX_flag5</string>
</map>
```

The contents of shared preference 'Flags' contains the 5th flag which is **R00tK!tX_flag5**.

Flag6:

```
<provider
  android:name=".SurpriseProvider"
  android:authorities="com.example.bdwisher.provider"
  android:exported="true"
  android:enabled="true" />
```

Looking at the manifest, we can see that there is a provider named 'SupriseProvider'. Exported is set to true, which means that the provider can be called from outside.

```

class SurpriseProvider : ContentProvider() {

    companion object {
        const val AUTHORITY = "com.example.bdwisher.provider"
        const val PATH_FLAG = "surprise"

        val CONTENT_URI: Uri = Uri.parse(uriString: "content://$AUTHORITY/$PATH_FLAG")

        const val CONTENT_TYPE = "vnd.android.cursor.dir/vnd.com.example.bdwisher.surprise"
        const val COLUMN_FLAG = "flag"
    }
}

```

Looking at the code we find the SurpriseProvider's Uri:
content://com.example.bdwisher.provider/surprise.

```

override fun query(
    uri: Uri,
    projection: Array<String>?,
    selection: String?,
    selectionArgs: Array<String>?,
    sortOrder: String?
): Cursor? {
    if (uriMatcher.match(uri) != FLAG_CODE) {
        throw IllegalArgumentException("Unknown URI: $uri")
    }

    val cursor = MatrixCursor(arrayOf(COLUMN_FLAG))

    if (selection == SECRET_KEY) {
        cursor.addRow(arrayOf(fetchedFlag))
    } else {
        cursor.addRow(arrayOf("Access denied. Incorrect key provided."))
    }

    return cursor
}

```

Looking at the query function of SurpriseProvider we find that the flag will be displayed if the query is equal to the SECRET_KEY.

```

private const val SECRET_KEY = "TODAY_IS_MY_BIRTHDAY"

```

The SECRET_KEY is found in the same file.

Now we can construct the command as we know everything that is required.

Command= adb shell content query --uri content://com.example.bdwisher.provider/surprise --where "TODAY_IS_MY_BIRTHDAY"

```
PS C:\Users\preet\Desktop\PACKAGES\MOB_SEC\RaptorAndroidCTF> adb shell content query --uri content://com.example.bdwisher.provider/surprise --where "TODAY_IS_MY_BIRTHDAY"
Row: 0 flag=3xpL01tH4ck3r_flag6
PS C:\Users\preet\Desktop\PACKAGES\MOB_SEC\RaptorAndroidCTF>
```

We get the flag6 which is **3xpL01tH4ck3r_flag6**.

Flag 7 :

From the AndroidManifest file, we came to see there is an activity named UnusedActivity by the name that says it is not used anymore and it is actually exported as true by which we can open the activity even from outside the application.

```
<activity
    android:name=".UnusedActivity"
    android:exported="true" />
<activity
```

```

class UnusedActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_unused)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }
        getFlagFromFirestore(context: this) { flag ->
            showFlag(flag)
        }
    }

    private fun showFlag(flag: String):String {
        AlertDialog.Builder(context: this)
            .setTitle("Captured Flag")
            .setMessage("The 1st Half of the Flag is : "+flag)
            .setPositiveButton(text: "OK") { _, _ -> finish() }
            .show()
        return flag;
    }

    private fun getFlagFromFirestore(context: Context, callback: (String) -> Unit) {
        val db = FirebaseFirestore.getInstance()
        val documentId = context.getString(R.string.document)

        db.collection(collectionPath: "flags").document(documentId)
            .get()
            .addOnSuccessListener { document ->
                if (document.exists()) {
                    val flag = document.getString(field: "flag8") ?: "N/A"
                    Log.d(tag: "HiddenActivity", msg: "Fetched flag: $flag")
                    callback(flag)
                } else {
                    Log.e(tag: "HiddenActivity", msg: "Document not found!")
                    callback("N/A")
                }
            }
            .addOnFailureListener { exception ->
                Log.e(tag: "HiddenActivity", msg: "Error fetching flag: ${exception.message}")
                callback("N/A")
            }
    }
}

```

From the above code, we came to see that the onCreate of the function calls the showFlag method which shows the first part of the flag in the Alert Dialog.


```

Java.perform(function () {
    console.log("[*] Waiting for the application to be initialized...");
    var ActivityThread = Java.use("android.app.ActivityThread");
    var app = ActivityThread.currentApplication();
    while (app == null) {
        app = ActivityThread.currentApplication();
    }
    var context = app.getApplicationContext();
    console.log("[+] Application context obtained!");
    try {
        console.log("[*] Attempting to start UnusedActivity...");
        var Intent = Java.use("android.content.Intent");
        var intent = Intent.$new();
        intent.setClassName("com.example.bdwysher", "com.example.bdwysher.UnusedActivity");
        intent.setFlags(0x10000000);
        context.startActivity(intent);
        console.log("[+] UnusedActivity triggered successfully!");
    } catch (e) {
        console.error("[-] Error triggering activity: " + e);
    }
});

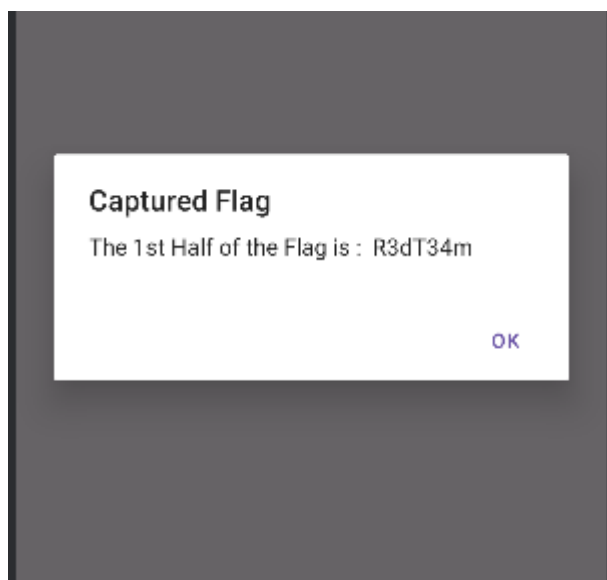
```

Using the above frida script or using the below adb command to start that activity results in the first part of the flag.

```

PS D:\bdwysher> adb shell am start -n com.example.bdwysher/.UnusedActivity
Starting: Intent { cmp=com.example.bdwysher/.UnusedActivity }

```



For the Second part of the flag :

There will be a class called Secret Storage which is not included in the actual business logic of the app, it will be known only after reverse engineering the application.

```

override fun onCreate(savedInstanceState: Bundle?) {

    recyclerView.layoutManager = LinearLayoutManager( context: this)
    loadBirthdays()

    addButton.setOnClickListener {
        startActivity(Intent( packageContext: this, AddBirthdayActivity::class.java))
    }

    getFlagFromFirestore( context: this, index: 3) { flag ->
        flagText = flag
        recyclerView.invalidate()
    }

    attachSwipeToDelete()
    SecretStorage.initializeFlag( context: this)
    obfuscatedUrl();
}

```

```

class SecretStorage {
    companion object {
        private var cachedFlag: String? = null // Store flag for quick access

        fun initializeFlag(context: Context) {
            val db = FirebaseFirestore.getInstance()
            val documentId = context.getString(R.string.document)

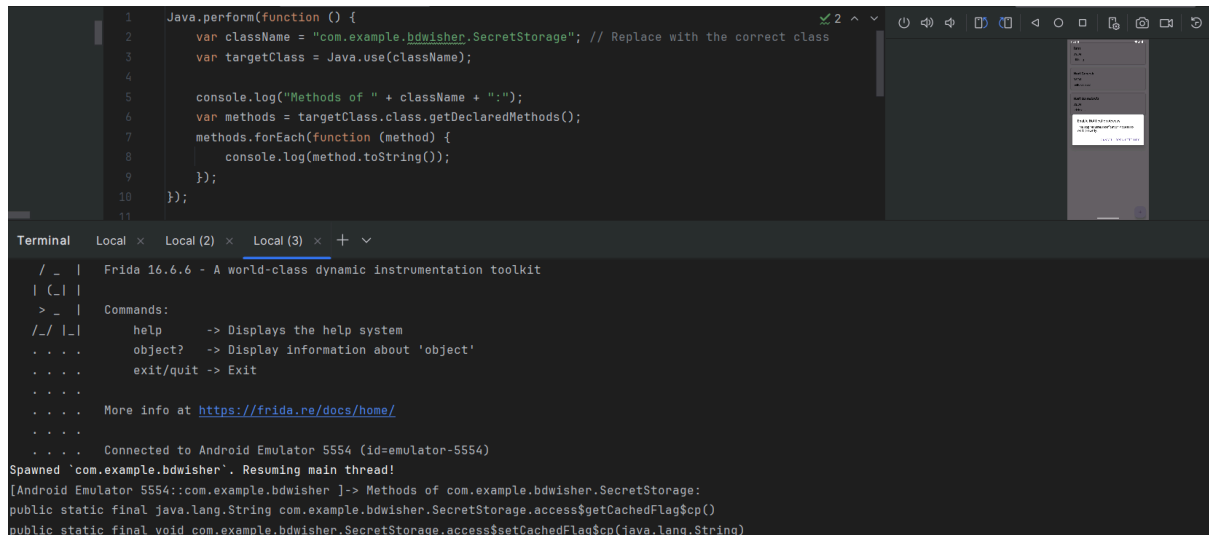
            db.collection( collectionPath: "flags").document(documentId)
                .get()
                .addOnSuccessListener { document ->
                    if (document.exists()) {
                        val flag = document.getString( field: "flag8_second") ?: "N/A"
                        cachedFlag = flag
                        Log.d( tag: "Firestore", msg: "Flag cached successfully: $flag")
                    } else {
                        Log.e( tag: "Firestore", msg: "Document does not exist!")
                    }
                }
                .addOnFailureListener { exception ->
                    Log.e( tag: "Firestore", msg: "Error fetching flag: ${exception.message}")
                }
        }

        fun revealFlag(): String {
            return cachedFlag ?: "FLAG_NOT_READY"
        }
    }
}

```

From the above we know that there is a function called `revealFlag` which returns the flag from the firestore.

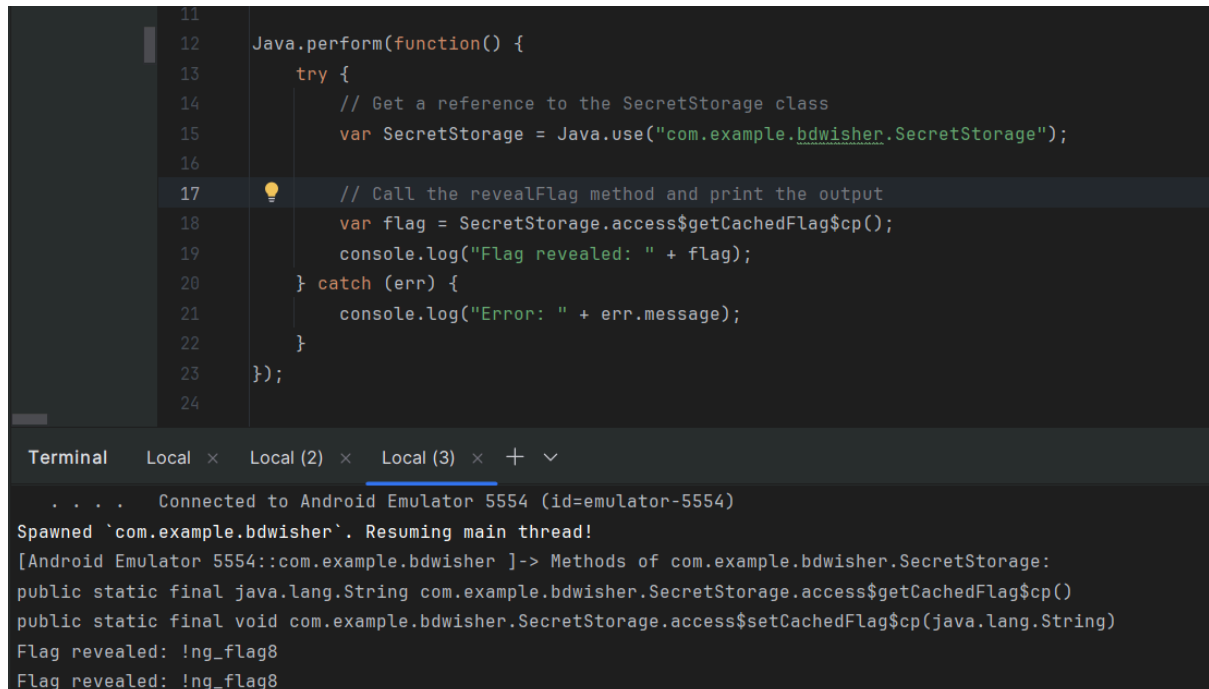
Using the frida script we need to trigger the `revealFlag` method to print the flag, for that we initially retrieve all the functions names from the `SecretStorage` class.



```
1  Java.perform(function () {
2      var className = "com.example.bdwisher.SecretStorage"; // Replace with the correct class
3      var targetClass = Java.use(className);
4
5      console.log("Methods of " + className + ":");
6      var methods = targetClass.class.getDeclaredMethods();
7      methods.forEach(function (method) {
8          console.log(method.toString());
9      });
10 });
11
```

```
Terminal Local x Local (2) x Local (3) x + v
/ _ | Frida 16.6.6 - 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.bdwisher'. Resuming main thread!
[Android Emulator 5554::com.example.bdwisher ]-> Methods of com.example.bdwisher.SecretStorage:
public static final java.lang.String com.example.bdwisher.SecretStorage.access$getCachedFlag$cp()
public static final void com.example.bdwisher.SecretStorage.access$setCachedFlag$cp(java.lang.String)
```

As there is a function called `access$getCachedFlag$cp`, so by calling that function reveals the second part of the flag which is : “ !ng_flag8 ”



```
11
12  Java.perform(function() {
13      try {
14          // Get a reference to the SecretStorage class
15          var SecretStorage = Java.use("com.example.bdwisher.SecretStorage");
16
17          // Call the revealFlag method and print the output
18          var flag = SecretStorage.access$getCachedFlag$cp();
19          console.log("Flag revealed: " + flag);
20      } catch (err) {
21          console.log("Error: " + err.message);
22      }
23  });
24
```

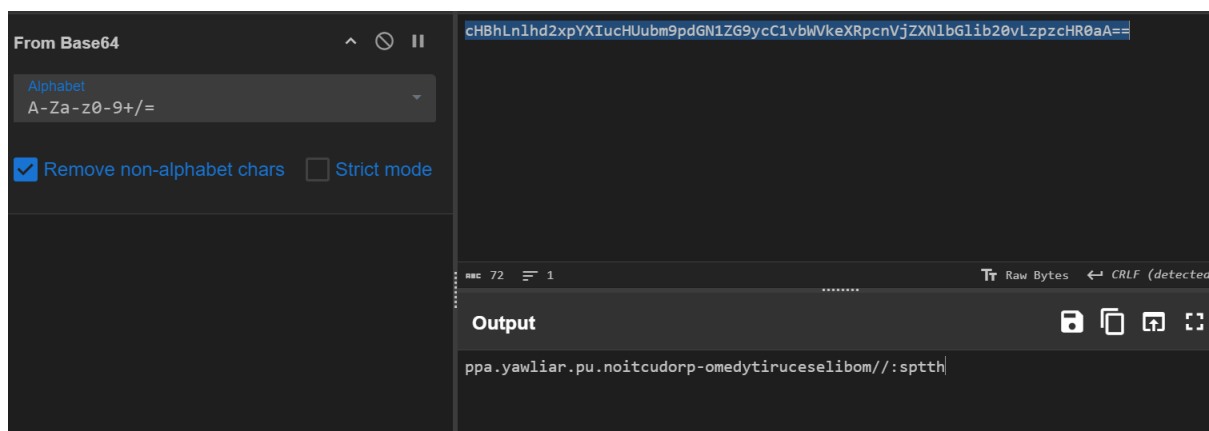
```
Terminal Local x Local (2) x Local (3) x + v
. . . | Connected to Android Emulator 5554 (id=emulator-5554)
Spawned 'com.example.bdwisher'. Resuming main thread!
[Android Emulator 5554::com.example.bdwisher ]-> Methods of com.example.bdwisher.SecretStorage:
public static final java.lang.String com.example.bdwisher.SecretStorage.access$getCachedFlag$cp()
public static final void com.example.bdwisher.SecretStorage.access$setCachedFlag$cp(java.lang.String)
Flag revealed: !ng_flag8
Flag revealed: !ng_flag8
```

Flag 8 :

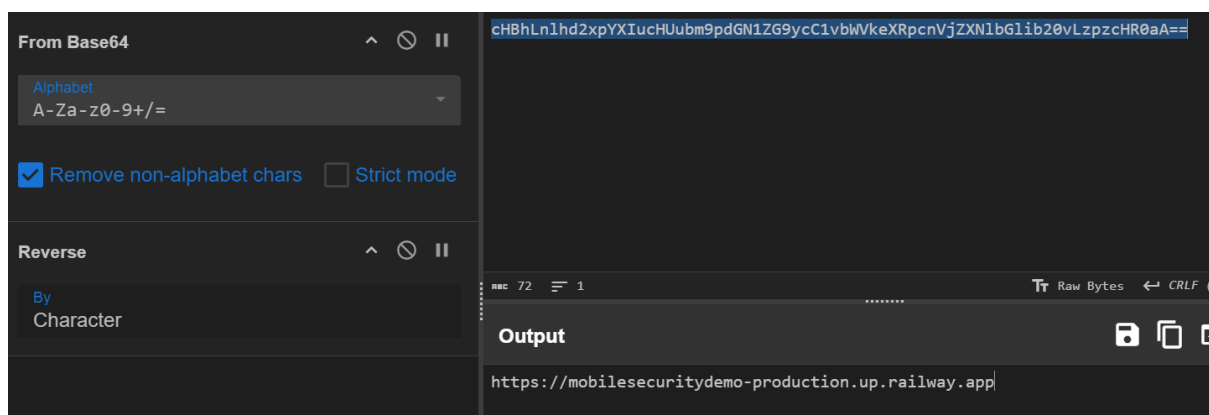
There will be a function called `obfuscatedUrl` in the Main Activity, which logs the random string in the Logcat. From the '=' suffix of the string, we may know that it can be a base64 encoded string.

```
fun obfuscatedUrl() {  
    val url = "CnBwYS55YXdsaWFyLnB1Lm5vaXRjdWRvcnAtb21lZHL0aXJ1Y2VzZWxpYm9tLy86c3B0dGg=";  
    Log.d("tag: \"The Flag Url is \",url);  
}
```

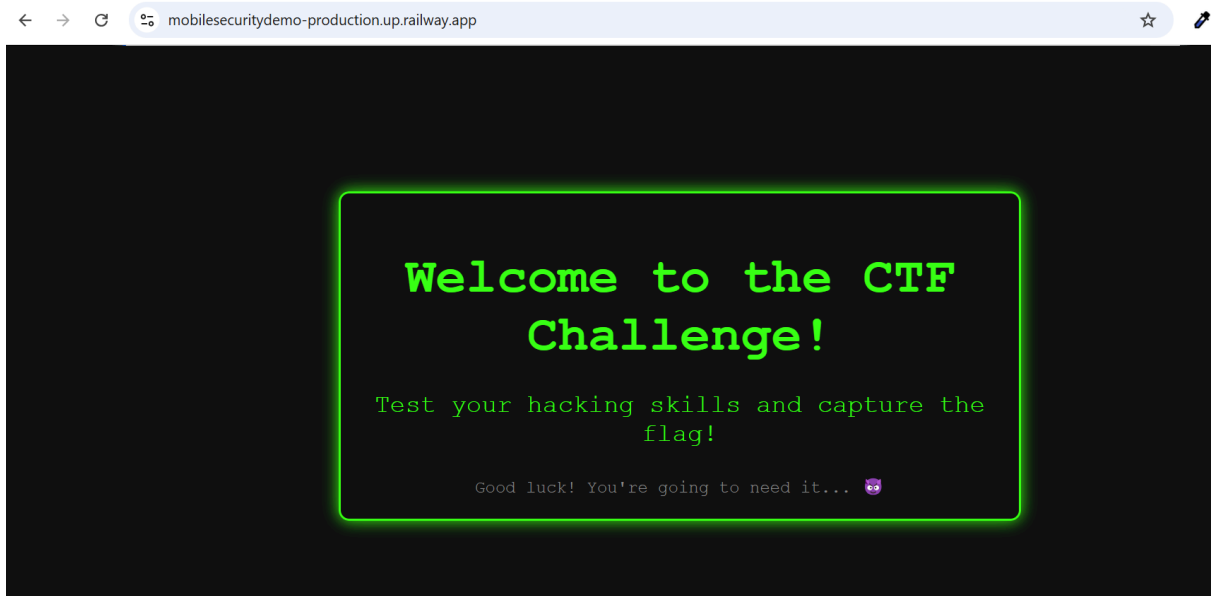
After decoding the string it results in : `ppa.yawliar.pu.noitcudorp-omedytiruceselibom//:sptth`



After reversing the obtained string we get the url of a web page.



As we get the web url, opening it in the browser results in the below web page.



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CTF Challenge - Welcome</title>
    <style>
  </head>
  <body>
    <div class="container">
      <h1>Welcome to the CTF Challenge!</h1>
      <p>Test your hacking skills and capture the flag!</p>
      <div class="footer">Good luck! You're going to need it... </div>
    </div>
  </body>
</html>
```

Once inspecting the webpage source html we didn't get any flag.

When we try to look into the response header of the url, we get the flag in header attribute called X-Flag attribute.

✕ Headers		Preview	Response	Initiator	Timing
▼ General					
Request URL:				https://mobilesecuritydemo-production.up.railway.app/	
Request Method:				GET	
Status Code:				● 200 OK	
Remote Address:				66.33.22.2:443	
Referrer Policy:				strict-origin-when-cross-origin	
▼ Response Headers					
Content-Length:				1518	
Content-Type:				text/html; charset=utf-8	
Date:				Wed, 02 Apr 2025 17:28:01 GMT	
Server:				railway-edge	
X-Flag:				R3dT34m!ng_flag8	

Based on your second document, **MobSecPackageWriteup.pdf**, I've compiled a structured **question-based challenge-solving document** designed to guide CTF participants through each flag challenge using a problem-solving approach.



MobSec CTF Challenge — Question-Based Solver Guide

Created by: Hari Ganesh M (21PC14), Preetham Bavan E (21PC22),
Satheesh Kumar RS (21PC32)

Overview

This document is intended for solvers of the *Mobile Security Package CTF*. Each flag challenge is presented with investigative questions designed to stimulate reverse engineering, code analysis, and exploitation skills.



Flag 1 – Hash in DB Helper

Questions to Guide You:

1. Where is the data being inserted or stored in the app?
 2. Can you locate any hash or cryptographic functions within DataBaseHelper.kt?
 3. Does the hash relate to a flag directly or a verification step?
 4. What part of the database code provides the final hashed value?
 5. Is the hash used for comparison or logging anywhere?
-



Flag 2 & 3 – Notification Trigger via getTime()

Questions to Guide You:

1. What is the role of the `getTime()` function in `NotificationScheduler.kt`?
2. How does the app determine the timing for birthday notifications?
3. What would happen if you modified the returned time via runtime tools like Frida?
4. Can you alter the scheduled time difference to trigger immediate flag notifications?
5. What are the outputs in logs and notification pop-ups after modification?

 *Note:* Flag 2 appears in the **logs**, Flag 3 appears in the **notification content**.



Flag 4 – Swipe to Delete UI Logic

Questions to Guide You:


1. What UI action leads to the deletion of a birthday item?
2. Is there any code or callback triggered on swipe left?
3. Where in the layout or adapter is the flag value tied to a delete action?
4. Does the flag appear as part of a toast, log, or background task when deleting?



Flag 5 – Shared Preference Check after Adding Items

Questions to Guide You:

1. What function loads birthday data in the app?
2. What happens when the list contains exactly 7 items?
3. Does any function like `getFlagFromFirestore()` get triggered at this point?
4. Where is the shared preference 'Flags' located on the device?
5. Can you extract and decode data from the XML of the shared preference?

 *Tip:* Use Android Device File Explorer to access
`/data/data/<package>/shared_prefs/Flags.xml`



Flag 6 – Exploiting Exported Content Provider

Questions to Guide You:

1. Is there any provider component defined in `AndroidManifest.xml`?
2. What is the exported status of the `SurpriseProvider`?
3. What is the correct URI and query expected by the provider?
4. What is the `SECRET_KEY` value and how is it verified?
5. Can you construct the right adb shell content query to fetch the flag?

 Example format:

```
adb shell content query --uri content://com.example.bdwisher.provider/surprise --where  
"TODAY_IS_MY_BIRTHDAY"
```



Flag 7 – Exploiting Unused Exported Activity

Questions to Guide You:

1. Is there an activity marked `exported=true` that is not used in navigation?
 2. What does the `onCreate` method of `UnusedActivity` reveal?
 3. Is the flag partially shown via `AlertDialog` or other UI?
 4. Is there a `SecretStorage` class not linked to visible functionality?
 5. What function within `SecretStorage` contains or reveals the flag?
 6. Can Frida be used to hook and call `access$getCachedFlag$cp`?
-



Flag 8 – Obfuscated URL and Web Header Flag

Questions to Guide You:

1. Where is the function `obfuscatedUrl()` defined?
2. What kind of encoding (e.g., Base64) is used in the logged string?
3. After decoding, is the string reversed or scrambled further?
4. What happens when you visit the resulting URL in a browser?
5. Is the flag present in HTML source or response headers?
6. Can tools like `curl` or browser DevTools help inspect X-Flag headers?

✓ Example using `curl`:

```
curl -I <url>
```

✓ Tools You'll Need:

- **Frida** (runtime hooking)
- **ADB** (debug bridge and command execution)

- **Jadx / apktool** (decompiling APK)
- **Device File Explorer** (for shared preferences)
- **Base64 tools / Python** (decoding strings)
- **Browser Developer Tools** (web header inspection)

Would you like me to export this guide as a **PDF** or **Word document**?