

Enumerating Local Storage

Enumerating local storage is a key part of dynamic analysis in Android applications. It's important to understand how an app handles data on the device, as this can reveal potential security risks. Android apps use several storage mechanisms—such as databases, shared preferences, and app-specific external directories—to manage and persist data. By examining these, we can uncover sensitive information that may be exposed, data stored insecurely, or possible entry points for attackers. The goal is to understand how local storage is used and how to effectively analyze it for security flaws.

In Android, local storage encompasses the various methods and locations on a device that applications use to store and access data. It is generally divided into two main categories: **internal storage** and **external storage**.

Internal Storage

Internal storage refers to the private area allocated to each application on an Android device, where it can securely store data. On non-rooted, standard Android devices, this area is inaccessible to other apps and users, ensuring high security and privacy. This makes it ideal for storing sensitive information like user preferences, app settings, and small data files specific to the app.

However, rooted devices allow users and other apps to access all file system areas, including other apps' internal storage spaces. Once the app is uninstalled, data stored in this area is deleted.

External Storage

External storage refers to storage spaces that are not exclusively tied to the app, like SD cards or shared internal partitions. This space is less secure and thus suitable for less sensitive data like media files or documents shared between apps. Since this partition is part of the device's overall internal memory but not confined to the app's private storage area, it is generally more reliable and secure than removable storage like an SD card but still less secure than **app-specific internal storage**. The **App-specific internal storage** is, even though is located in the **external storage**, it is designed to be accessible only by the app itself, offering a higher level of security for sensitive data.

Applications that store data on **external storage** must properly manage permissions to prevent unauthorized access, as accessibility depends on factors such as user-granted permissions and the presence of a physical SD card. Unlike **internal storage** and **app-specific internal storage**, data stored in this partition can remain on the device even if the app is uninstalled. In the following examples, we will examine three different cases of applications that improperly store sensitive data in both internal and external storage.

Databases

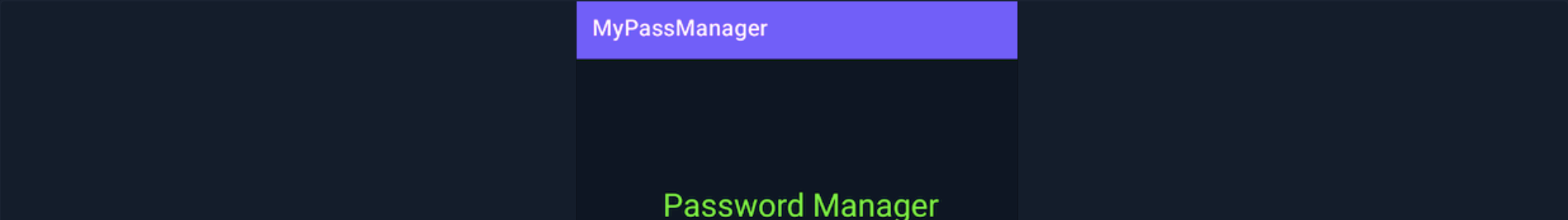
Most Android applications use databases to store structured data. In this example, we'll explore how to access and analyze these databases and identify potential misconfigurations or mishandling of sensitive data. We will primarily use an Android Virtual Device (AVD), though the process is compatible with any other Android device, physical or emulated. Let's connect to the device via ADB and install the application.

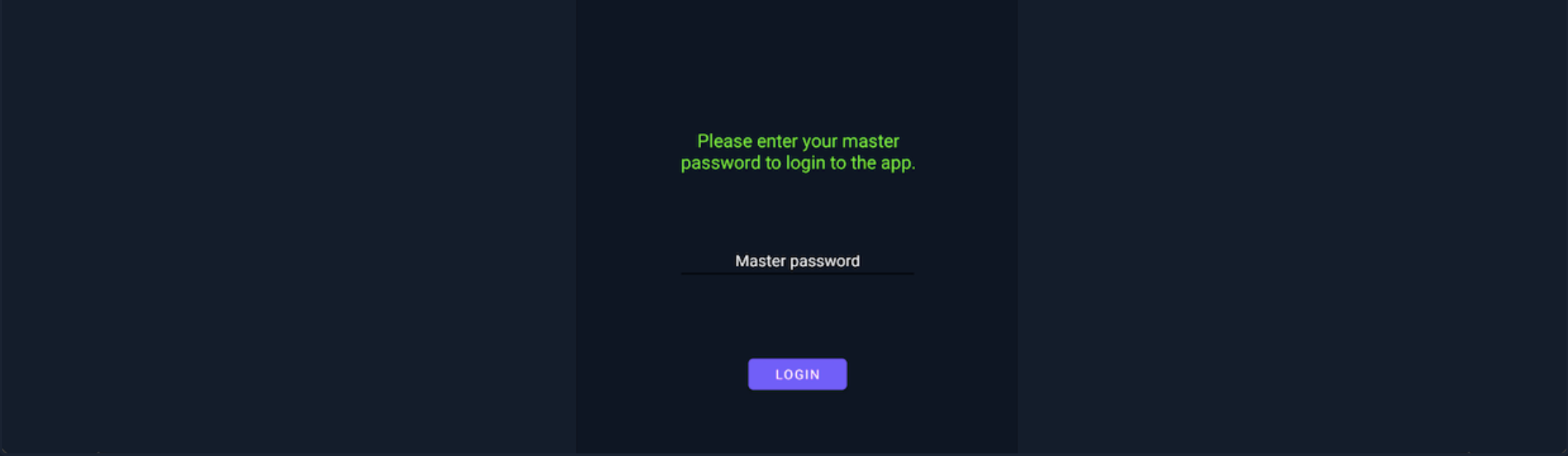
Enumerating Local Storage

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

Performing Streamed Install
Success
```

We can now run the application on the device.





Here, the password manager application asks the user for the master password to log in. Let's investigate the application's local storage and see if any information is stored there. We can get a shell as the user `root` using the commands below.

Enumerating Local Storage

```
r11k@htb[/htb]$ adb root
r11k@htb[/htb]$ adb shell

root:/#
```

We can now execute commands interactively in the remote device as the user `root`. In order to search the application's local storage, we need root access to the device, and we also need to know the application's package name. Since we already have root access, let's issue the following command to get the package name while the app is running on the device.

Android Shell

Enumerating Local Storage

```
root:/# dumpsys activity activities | grep VisibleActivityProcess

VisibleActivityProcess:[ ProcessRecord{4b8a399 2263:com.hackthebox.mypassmanager/u0a111}]
```

The above command can be an efficient way to identify apps that are actively running with a UI. Since this is the only app with a UI running on the device, the results reveal the application's package name `com.hackthebox.mypassmanager`. We can list the content of the application's local storage by executing the following command.

Android Shell

Enumerating Local Storage

```
root:/# ls -l /data/data/com.hackthebox.mypassmanager

total 16
drwxrws--x 2 u0_a111 u0_a111_cache 4096 2024-01-10 18:51 cache
drwxrws--x 2 u0_a111 u0_a111_cache 4096 2024-01-10 18:51 code_cache
drwxrwx--x 2 u0_a111 u0_a111      4096 2024-01-10 20:34 databases
drwxrwx--x 2 u0_a111 u0_a111      4096 2024-01-10 19:10 files
```

Along with other directories, the directory `databases` is also listed. Reading the content of this directory reveals the following files.

Android Shell

Enumerating Local Storage

```
root:/# ls -l /data/data/com.hackthebox.mypassmanager/databases/

total 16
-rw-rw---- 1 u0_a111 u0_a111 1/70/ 2024-01-10 20:34 SecureVault.db
```

```
-rw-rw---- 1 u0_a111 u0_a111 16384 2024-01-10 20:34 SecureVault.db
-rw-rw---- 1 u0_a111 u0_a111    0 2024-01-10 20:34 SecureVault.db-journal
```

We can read the content of `.db` files using the pre-installed tool `sqlite3`.

Android Shell

Enumerating Local Storage

```
root:/# sqlite3 /data/data/com.hackthebox.mypassmanager/databases/SecureVault.db

SQLite version 3.32.2 2021-07-12 15:00:17
Enter ".help" for usage hints.
sqlite>
```

Now, we are able to execute SQLite queries in the database `SecureVault.db`. Listing the tables of this database reveals the following.

Android Shell

Enumerating Local Storage

```
sqlite> .tables

android_metadata  credentials
```

Let's try to read the content of the table `credentials`.

Android Shell

Enumerating Local Storage

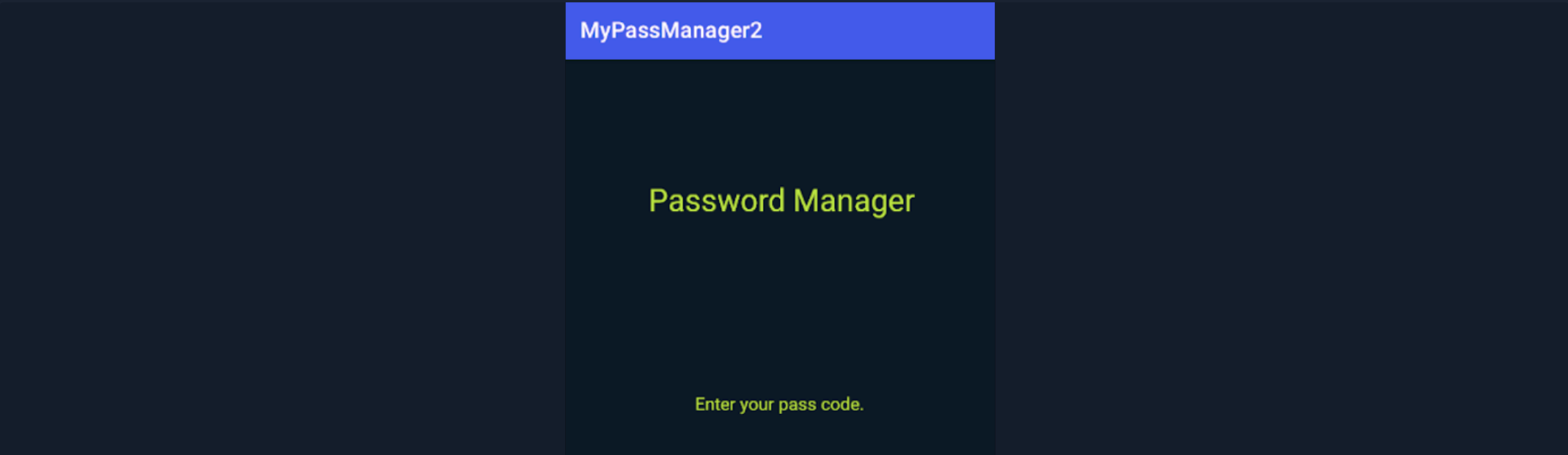
```
sqlite> select * from credentials;

1|bob@evilcorp.com|password123
2|john2@evilcorp.com|my_evil_pass!@#
3|max@evilcorp.com|4_Str0ng_P@sSw0rd!
```

The table contains the user's credentials for various apps in a non-encrypted format. Using this methodology, we bypassed the use of the master password and got access to the credentials directly. Information stored in local databases can often be encrypted. This means that pen-testers often have to combine static analysis to read the source code and find any potential encryption keys, along with reading the content of the local databases.

Shared Preferences

`Shared Preferences` is another common storage method used for storing key-value pairs. This example will demonstrate how to enumerate shared preferences and discuss the implications of insecure storage practices. The following application is the same password manager app, but this time, a feature allows the user to log in using a passcode instead of a master password.



Pass code

LOGIN

Assuming we are still connected to the device and can execute shell commands over ADB as the root user, let's list the local storage content of the application **MyPassManager2**. First, we need to find the app's package name, so we execute the following while the app is running.

Android Shell

Enumerating Local Storage

```
root:/# dumpsys activity activities | grep VisibleActivityProcess

VisibleActivityProcess:[ ProcessRecord{7ed3bad 3619:com.hackthebox.mypassmanager2/u0a114}]
```

Then, we execute the following to list the app's local storage content.

Android Shell

Enumerating Local Storage

```
root:/# ls -l /data/data/com.hackthebox.mypassmanager2/

total 16
drwxrws--x 2 u0_a114 u0_a114_cache 4096 2024-01-11 12:36 cache
drwxrws--x 2 u0_a114 u0_a114_cache 4096 2024-01-11 12:36 code_cache
drwxrwx--x 2 u0_a114 u0_a114      4096 2024-01-11 12:40 files
drwxrwx--x 2 u0_a114 u0_a114      4096 2024-01-11 12:40 shared_prefs
```

The output reveals the directory **shared_prefs**. As we mentioned, this is used from the app to store small collections of key-value pairs. Listing the content of this directory reveals the file **app_properties.xml**.

Android Shell

Enumerating Local Storage

```
root:/# ls -l /data/data/com.hackthebox.mypassmanager2/shared_prefs/

total 4
-rw-rw---- 1 u0_a114 u0_a114 120 2024-01-11 12:40 app_properties.xml
```

We can read the content of this file by using the **cat** command.

Android Shell

Enumerating Local Storage

```
root:/# cat /data/data/com.hackthebox.mypassmanager2/shared_prefs/app_properties.xml

<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <boolean name="pinLockStatus" value="true" />
</map>
```

This XML file contains the boolean key-value pair **<boolean name="pinLockStatus" value="true" />**. Let's try to change the value to **false** and see if we can bypass the passcode authentication step. We can change the value to **false** using the **vi** editor, which is pre-installed on the device.

Android Shell

Enumerating Local Storage

```
root:/# vi /data/data/com.hackthebox.mypassmanager2/shared_prefs/app_properties.xml
```

Once we change it, it should appear like this.

Android Shell

Enumerating Local Storage

```
root:/# cat /data/data/com.hackthebox.mypassmanager2/shared_prefs/app_properties.xml

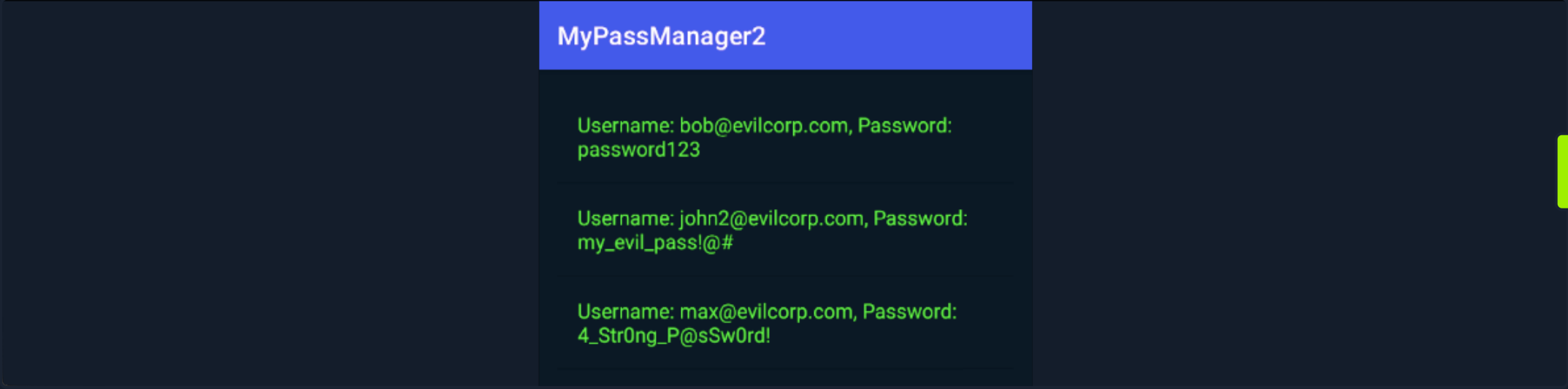
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <boolean name="pinLockStatus" value="false" />
</map>
```

Finally, we stop the application using the command below and tap it to start again.

Android Shell

Enumerating Local Storage

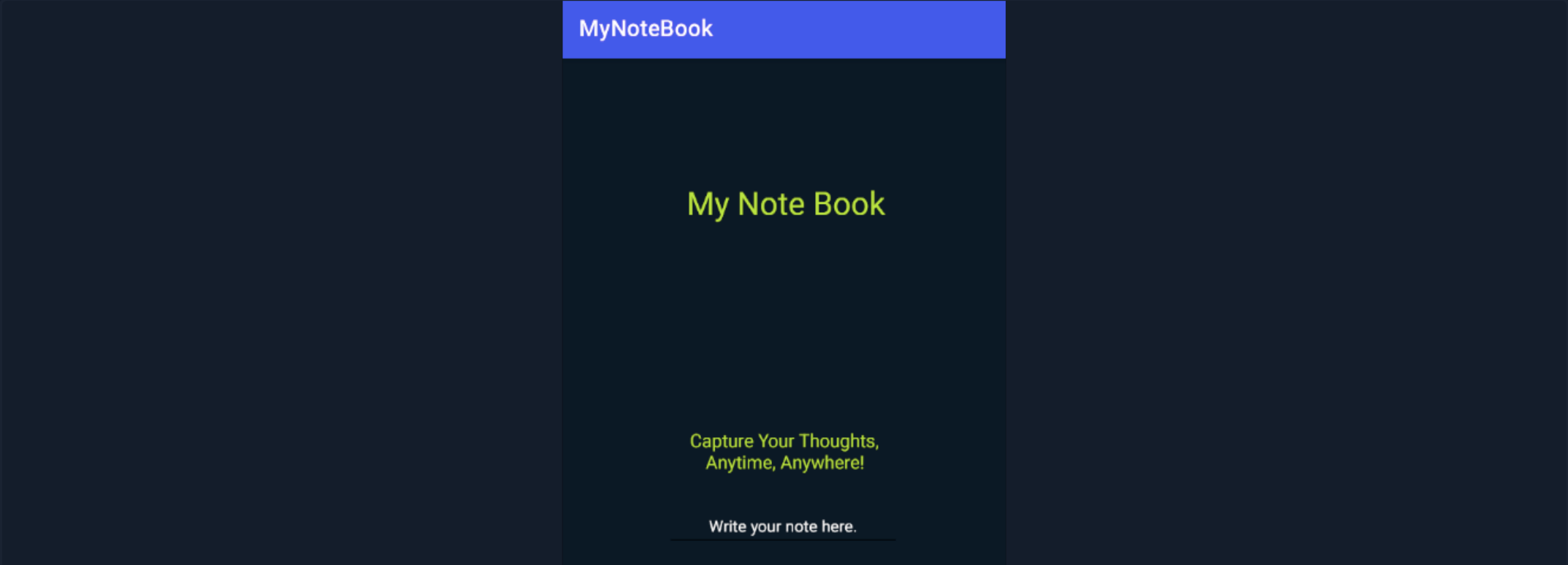
```
root:/# am force-stop com.hackthebox.mypassmanager2
```

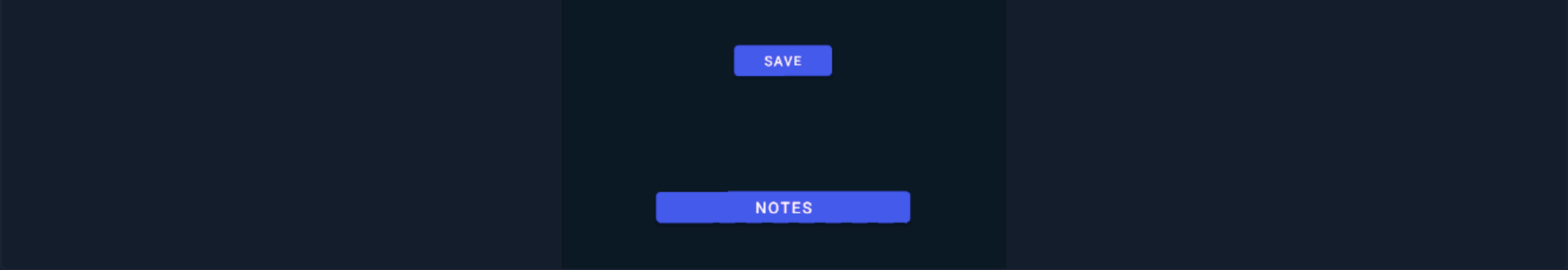


The passcode authentication step has been successfully bypassed. We can see the user's saved passwords for other apps.

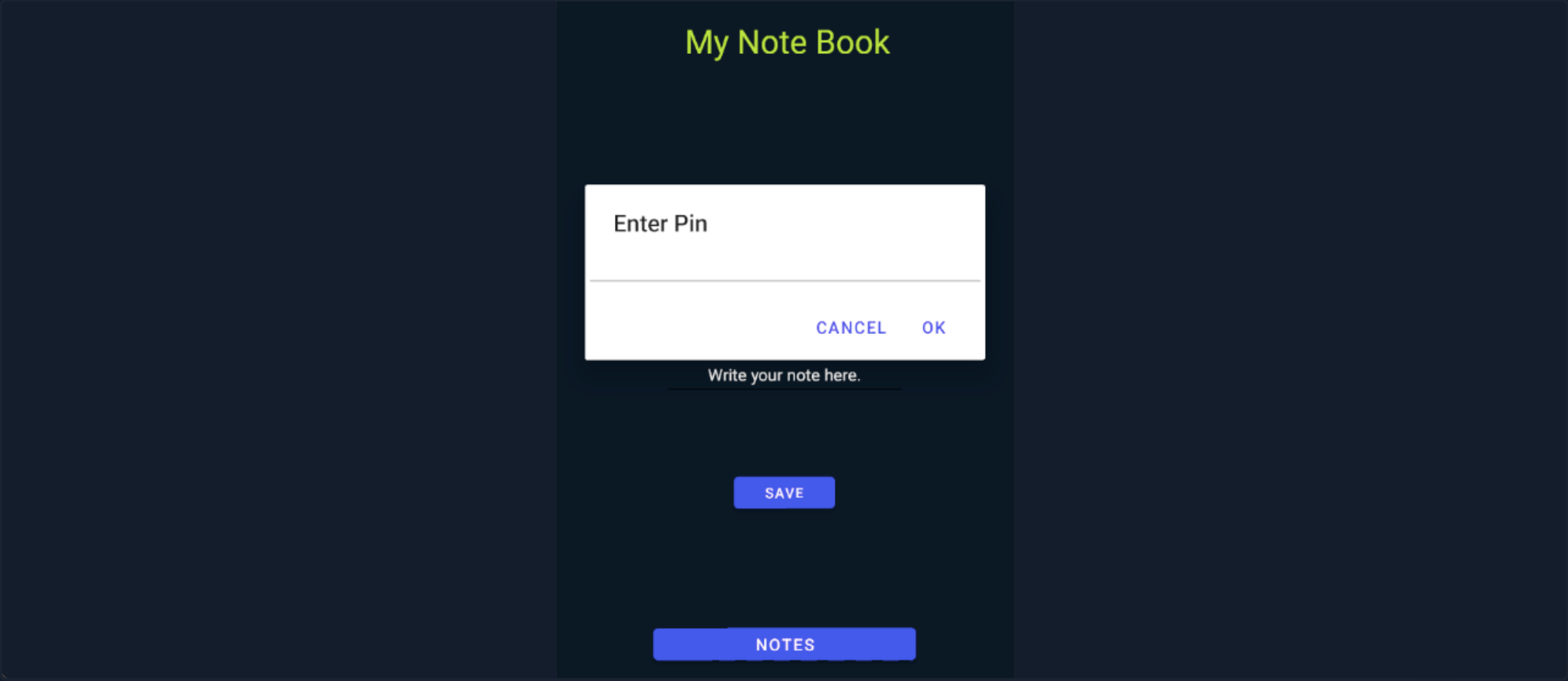
App-Specific External Storage

Many apps use external storage to save larger datasets or share data between applications. In this example, we'll explore the potential security risks associated with external storage and demonstrate how to inspect and analyze the contents of these directories. The following is a note-taking app that allows users to save notes on their devices.



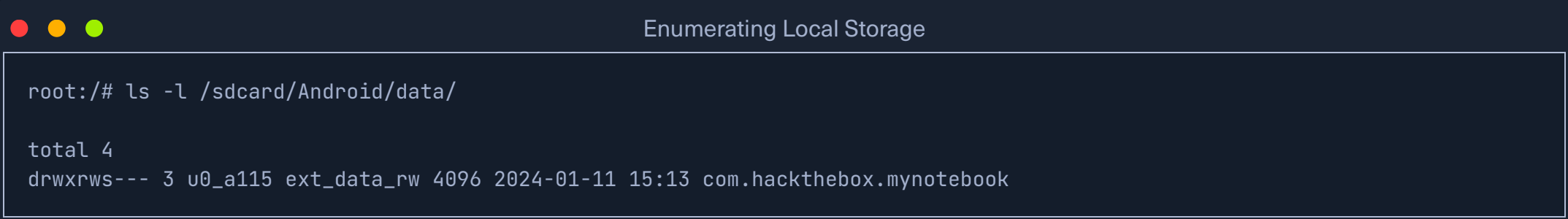


Tapping the **NOTES** button prompts the user for a PIN.



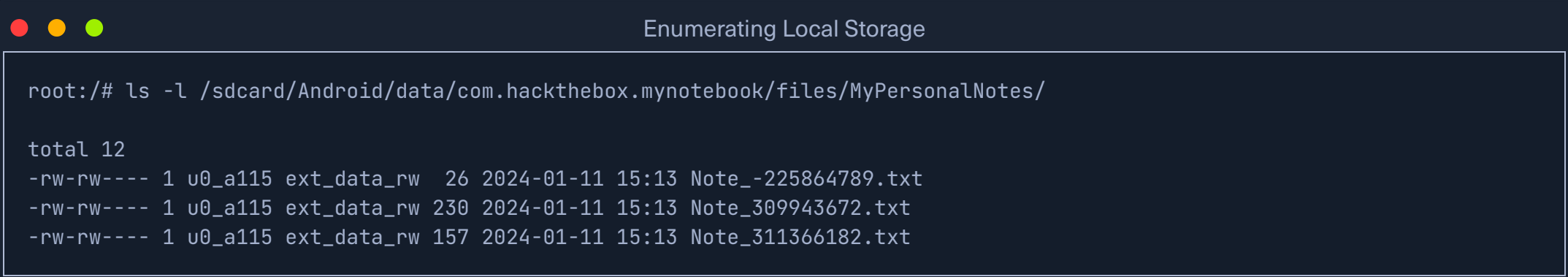
Let's enumerate the device's storage to see if we can find where the app stores its notes. Enumeration of the directory `/sdcard/Android/data/`, which is the directory that hosts applications' specific files, reveals the directory `com.hackthebox.mynotebook`.

Android Shell



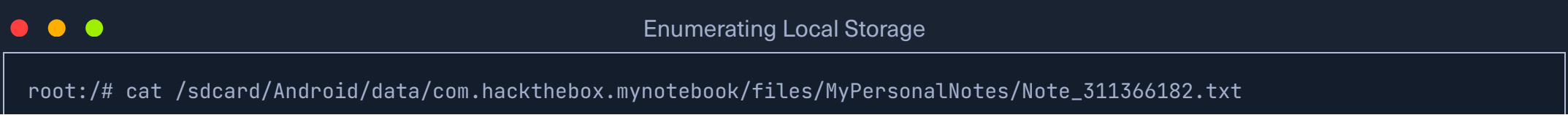
Listing its content will show the directories `files/MyPersonalNotes` that contain the following files.

Android Shell



We can read the content of these files using the command `cat`.

Android Shell




Grocery List:

- Milk
- Eggs
- Bread (whole grain)
- Apples
- Chicken breasts
- Green tea
- Almonds
- Greek yogurt

Remember to check for gluten-free options!

These `.txt` files contain the saved notes that users cannot see directly from the app without knowing the pin. This highlights the importance of securing external storage, as unprotected files can be accessed easily on rooted devices or through unauthorized apps.



Connect to Pwnbox

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

Pwnbox Location

UK

40ms

▼

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left



Waiting to start...

☐

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

Questions

Cheat Sheet

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

+ 5 📦

What is the password of the user "ethan@evilcorp.com" ?

Submit your answer here...

+10 Streak pts

Submit

enum_local_storage_1.zip

+ 5

What is the password of the user "emma@evilcorp.com" ?

Submit your answer here...

+10 Streak pts

Submit

enum_local_storage_2.zip

+ 5

What is the content of the file "Note_-225864789.txt" ?

Submit your answer here...

+10 Streak pts

Submit

enum_local_storage_3.zip

Previous

Next

Cheat Sheet

Go to Questions

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

