# Assignment - 2

**Ans 4.**

a) The program simulates OTP generation and verification using hashing. The two files/programs, namely ***otpGenerator.py*** and ***otpVerifier.py*** handle the generation and verification while the ***main.py*** is the driver program that uses those files to simulate OTP generation and verification. Below are the complete implementation details:

**otpGenerator.py-**

1. **getHash_H(T):**
    a. This function is implemented to derive a Hash, H, from input Time String. The time string format is YYYY-MM-DD HH:MM:SS. The sha256 function of hashlib library has been used here.
2. **getF(H) (function F):**
    a. This function is the F function given in the question.
    It derives a 4-digit numeric OTP from the hash string H obtained using the getHash_H() function.
    b. The function divides the 64-character hexadecimal hash into 8-character chunks, converts each chunk into an integer, and then XORs all these integers together to combine their information. This ensures that every part of the hash influences the final result.
    c. After folding, the function takes the result modulo 10,000 to map it into the 4-digit numeric range (0000–9999). Finally, it formats and returns the OTP as a 4-digit string (adding leading zeros if needed).
3. **getOTP():**
    a. It first obtains the system date and time using datetime.now(). Then it converts the date-time to a string.
    b. Then it calls the getHash_H function with the string date-time as input parameter, thus generating a hash value and storing it in a hash.txt file.
    c. getF() is then called with hash value as input parameter, and it returns a 4-digit OTP, which is then displayed to the user.

**otpVerifier.py-**

1. **verifyOTP():**
    a. Checks if hash.txt exists, if it does then it reads the hash value.
    b. Then it calls the getF function(same as otpGenerator.py) with this read hash value to get expected OTP.
    c. It then asks the user to input the generated OTP.

d. The expected OTP is then compared with the input OTP from the user. If the OTPs match then the verification message is displayed followed by the deletion of the hash.txt file storing the hash value.

**main.py-**

1. This program controls the flow of the complete generator and verifier mechanism.
2. It uses a CLI interface, first executing the OTP generation mechanism followed by verification mechanism.

**GitHub Link:** https://github.com/Abhinav0821/UsS_A2

b) Following are the issues specific to the OTP generation method mentioned in the question(not focussing on implementation of F):

**1. Predictable / low-entropy seed:**

- The system derives H from the system timestamp alone, which is easy to guess.
- If an attacker knows (or can estimate) when the OTP was generated — even to within a few minutes, they can enumerate plausible timestamps (second-by-second), compute the corresponding hashes, run the same F on each, and recover the OTP offline.
- This can be solved using a hidden/secret factor that the attacker cannot guess, like a server based attribute, or add a random noise in the hash.

**2. Small numeric space:**

- A 4-digit code offers only 10,000 possible values.
- An attacker can automate brute force attempts and try the full code space in affordable time duration.
- Enforce strict attempt limits (e.g., 3 tries), apply progressive delays and account lockouts(e.g., After certain tries, next attempt only after 24 hrs), or increase OTP length while keeping in mind usability.

**3. Plaintext and local storage of H:**

- Storing the raw hash on disk (e.g., hash.txt) lets any process or user with read access obtain H.
- A local or privileged process can read the file, run F(H) locally, and immediately obtain the OTP. Backups or synced copies of the file create additional leakage vectors.

- Encrypt the hash file with authenticated encryption and keep the key separate or avoid disk storage entirely (keep H in secure memory or a protected database).

**4. Metadata leakage attacks:**

- File timestamps, logs, or backup metadata can reveal when H was generated.
- Knowing exact or approximate generation time reduces the timestamp search space, making offline guessing far easier.
- Avoid exposing generation times in accessible files or logs, restrict backup, and prefer not storing metadata.

**5. Replay and race-condition issues:**

- If the stored hash is not atomically consumed or deleted after verification, the same OTP may be used multiple times.
- Two verification attempts in parallel might both read the stored H before it is deleted, allowing duplicate acceptance. Such OTP, if identified, could be used before the server marks it as used and deletes it.
- Use atomic operations or a transactional storing/deletion of data to mark and consume OTPs.

Following are the issues with implementation of function F i.e. getF(H) specifically:

**1. Still limited by 4-digit output size:**

- No mapping, in this scope, could guarantee protection against brute force attacks.
- Use longer OTP or enforce rate limits

**2. Deterministic and trivially computable from H:**

- If the attacker obtains the raw H, computing the OTP is trivial (same as with any deterministic F).
- Protect stored hash with encryption

**3. Modulo Bias:**

- f % 10000 depends slightly on the distribution of f. Even with 32-bit words it's negligible but not 100% uniformly distributed.
- A common practice of utilising truncated 31-bit integers and then mod.

**4. Statistical analysis:**

- Repeated generation and observation of many OTPs allows an attacker to identify biases in digit-extraction (frequency analysis), then prioritize guesses accordingly.

**Ans 1.**

a)

b) i. I downloaded the Tor browser and used it to visit three onion links which are as follows:

- DuckDuckGo - https://duckduckgogg42xjoc72x3sjasowoarfbgcmvfimaftt6twagswzczad.onion/
- The Washington Post - https://vfnmxpa6fo4jdpyq3yneqhglluweax2uclvxkytfpmpkp5rsl75ir5qd.onion/
- The Guardian - http://xp44cagis447k3lpb4wwhcqukix6cgqokbuys24vmxmbzmaq2qjvc2yd.onion/

ii. The browsing experience across the three services demonstrates clear examples of security-usability trade-offs inherent in Tor network.

iii. All three sites took significantly more time to load than their counterparts on other popular browsers. This delay is the most immediate usability issue. It is a direct consequence of the multi-hop routing and layered encryption, which is the cost of anonymity.

iv. Unlike standard site links(like www.duckduckgo.com), the .onion links are extremely difficult to find or predict. As they are cryptographically generated and cannot be guessed or easily remembered. I had to use third party directories to find these links and then copy-paste them, which significantly degrades the user's Mental Model.

v. Out of the 3 sites visited, DuckDuckGo loaded fastest while the other two took much more time. This can be attributed to DuckDuckGo being a simple text-based interface while others had complex structure and took time to render.

vi. The Tor Browser also provides an option to view the Tor circuit created for an onion link being visited. Following are the Tor circuits for the 3 links I visited,

respectively:

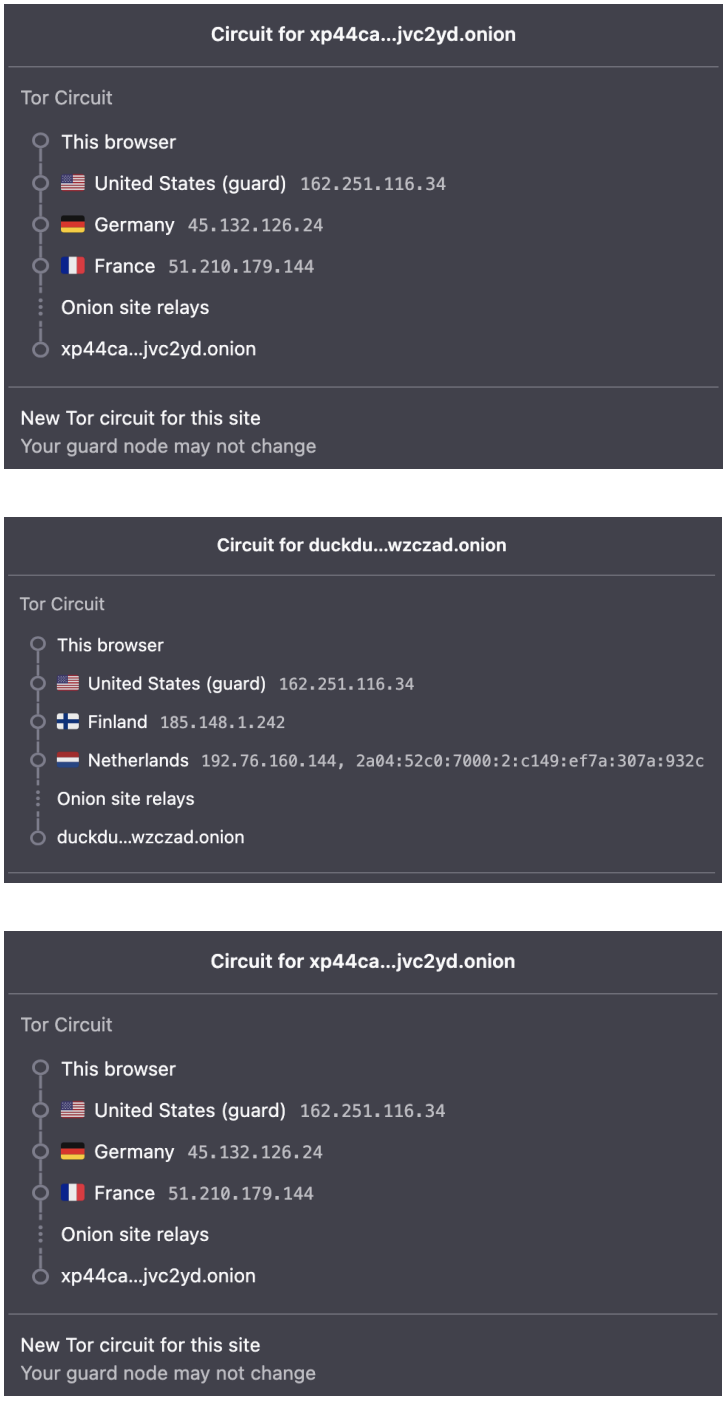


Fig: Tor Circuits formed for the 3 Links Visited

vii. The provided Tor circuits for all three services confirm the highest level of connection security. Providing an option to view Tor circuit helps keep the user informed about what exactly has been done to provide security, but one has to have knowledge about the security concepts to understand these mechanisms

and thus even though there is information, it misses the usability aspect of inclusivity.

viii. Both the Washington Post and Guardian websites maintained their recognizable layout, but appeared sort of simplified and without ads.

c) Based on my interaction with Tor Browser, I would say that it's less usable in comparison to other popular browsers like Firefox, Google Chrome, etc.

The main reason for less usability is the latency. Tor, due to its architecture and security handling mechanisms is very slow. It favours security and anonymity over the ease of usage, resulting in:

- Inefficient Performance: Pages are loaded slowly due to the 3-hop circuit required for Channel anonymity.
- Limited Functionality: To safeguard against Browser Fingerprint attacks, it often disables JavaScript, causing improper rendering of sites.
- Breaks Mental Model: The need to understand circuits, security levels, and .onion links makes the browser hard to use for novice users or users with no technical knowledge.

Following are the advantages and disadvantages of using Tor:

| **Advantages** | **Disadvantages.** |
|---|---|
| Anonymization: It provides robust sender anonymity by masking the user's actual IP address. | Less Usability: Aspects like complex to understand and slow speed make it less usable. |
| Prevents Tracking: The default settings in Tor automatically deal with third party cookies and many fingerprinting vectors. | High Latency: Slow speed in almost every task makes it inefficient and impacts the User Experience. |
| Prevent External Surveillance: Avoids traffic monitoring by local networks or ISPs, by using just a partial view of the network. | Mistrust: Some services block traffic from the exit nodes of Tor circuits, resulting in access denials. |

d) i. Based on the fact that Tor prioritizes anonymity and security most, I think the onion links appear to be random URLs because they are cryptographically generated, unlike the usual links used in other browsers.

ii. In Tor, the URLs are not domain names, rather they are hash of the service's public cryptographic key. The URL itself serves as a trust anchor.

iii. This setup also provides self authentication as once the user connects to an address, he/she is guaranteed to be connecting to the service that holds the corresponding private key.

iv. This is an anonymity mechanism that prevents identity spoofing, providing trust and integrity without relying on a centralized authority.

v. The identity of the service is verified at the moment of connection and since the traffic never leaves the Tor circuit, the Exit Nodes are bypassed. This protects against attacks from malicious Exit Nodes.

vi. There is no hint of the service name or location or hosting details, thus anonymizing service.

e) i. The URL naming convention used in Tor, is a main reason it lacks usability features, increasing the likelihood of user failure.

ii. The randomized, long strings make it difficult for users to memorize them. This is a key component in other browsers as remembering links makes searching efficient.

iii. This convention conflicts with the existing, much simpler Mental Model of the users. Regular URLs map to Designer Model, allowing users to guess common sub-pages, whereas the onion links defy any patter recognition, creating a distorted User Model.

iv. Due to uneasiness and confusion, Tor users are vulnerable to Social Engineering attacks. Attackers can trick a user into clicking a link that is off by one character.

v. The complex structure of onion links makes it difficult to share links confidently. The user cannot quickly verify the address's correctness, leading to frustration, making it a poor design.

vi. Due to its structure, the unusual onion links contribute to a sense of confusion and discomfort, especially for the novice users. This makes the browsing experience less welcoming leading to lower adoption rates.

f) i. Tor's current architecture and mechanism is security centered but lacks in terms of usability. We need to achieve a balance between security and usability, thus we must try to achieve the most appropriate trade off. Following are the import areas to work to achieve this trade off:
    - Problematic URL memorability - The browser should provide users with an option to assign aliases to the complex onion links. This will significantly improve recognition and memorization while still not compromising security.
    - Low Speed - Instead of building circuits when a user is trying to access a service, it should build circuits beforehand, when it's idle. This would reduce the latency as a pre-made circuit could be used almost instantaneously when a user clicks a link.
    - Limited Functionality - Instead of just denying access, the browser should inform the user that the page couldn't be loaded due to JavaScript being disabled and it should also provide the option of temporarily enabling it.
    - Built-in authenticity markers - The browser could show a unique icon or color for trusted bookmarks, so users can instantly see the site is real and not a phishing page.

**Ans 2.**

For this question I downloaded the Uncrackable-Level1.apk file(https://mas.owasp.org/crackmes/), which is a classic challenge binary designed by OWASP. I used JADX tool to reverse engineer and convert the application's DEX back into readable Java source code to trace control flow, find the anti-reversing mechanisms and finally obtain the encrypted secret key.

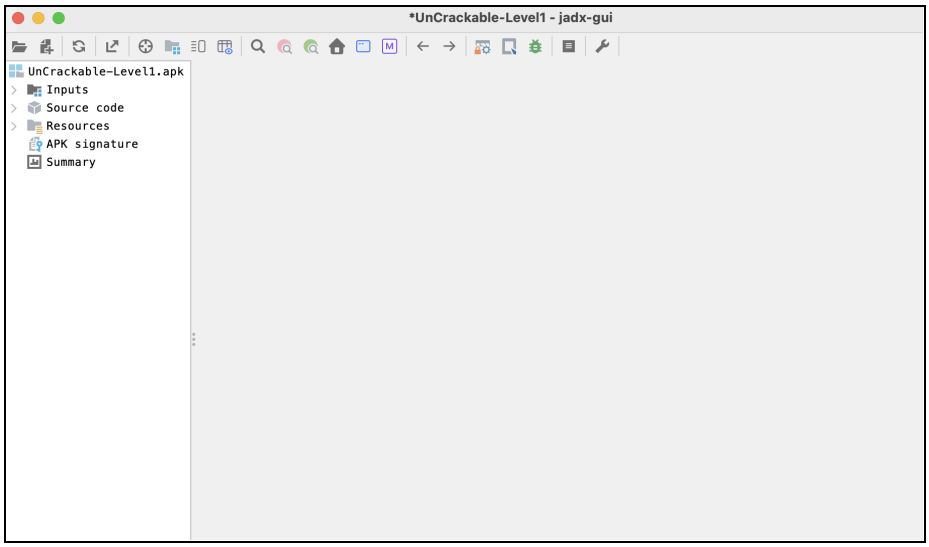


Fig: JADX GUI Interface after loading the APK file

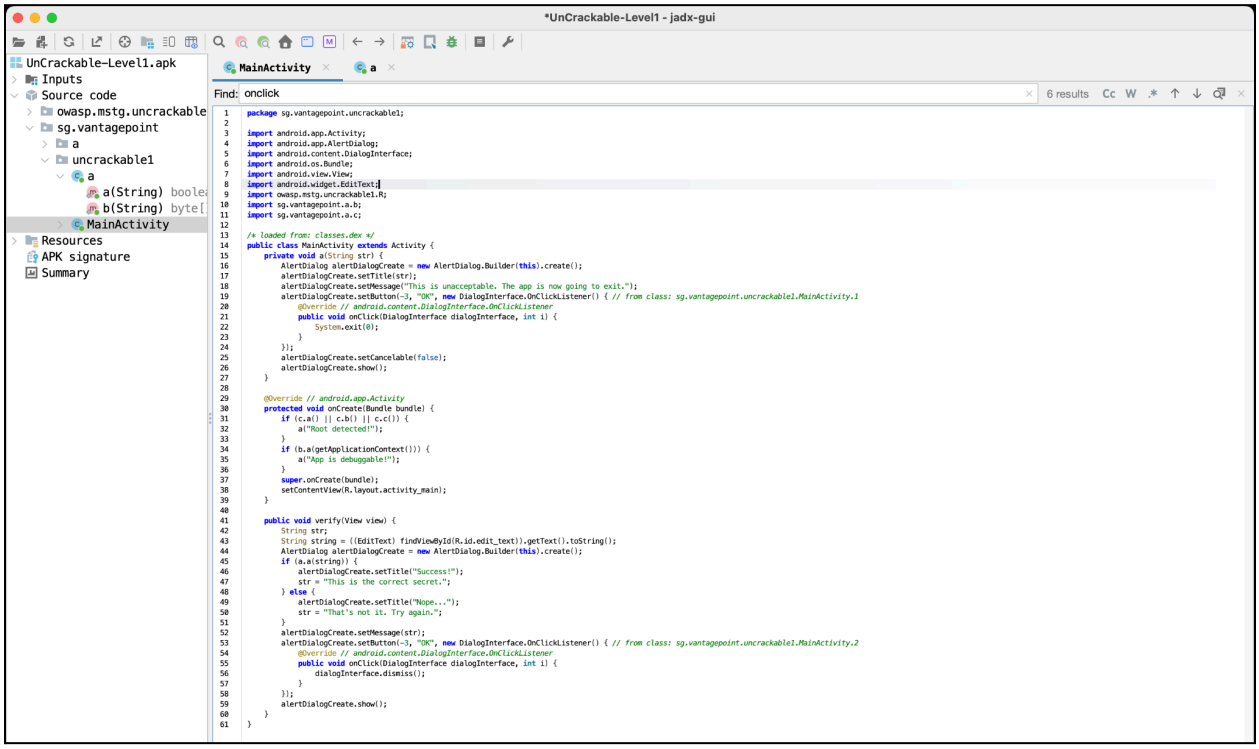


Fig: The source code of sg.vantagepoint.MainActivity

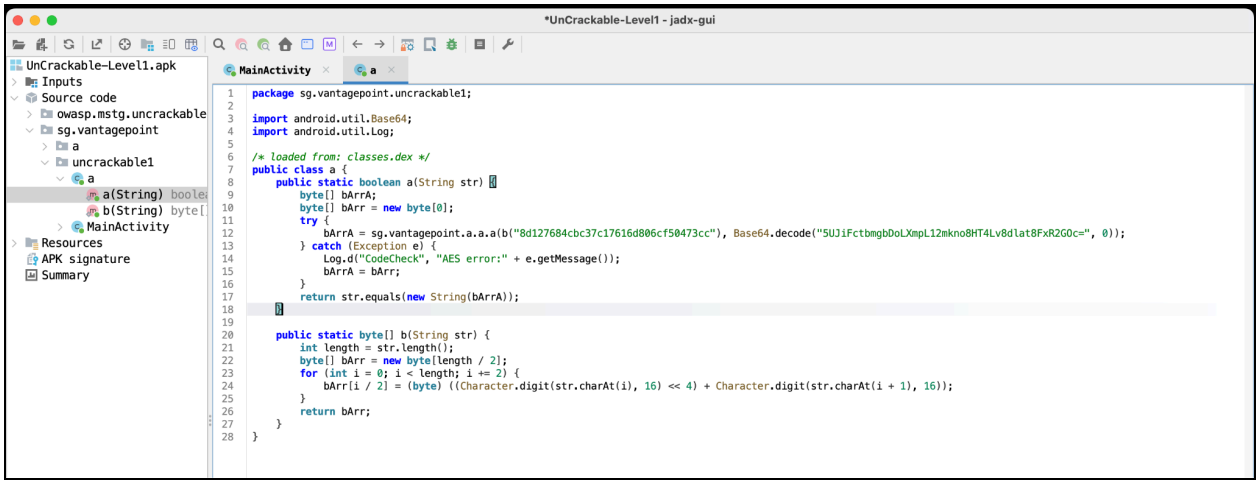


Fig: Code of sg.vantagepoint.a

a) The used executable performs following system calls which are Android Framework API calls:

   i) System.exit(0) - This syscall immediately terminates the process. It is also used as anti-tampering mechanism. In the given executable, it was called in MainActivity.a(). It is used in this file as an anti-reversing measure.

   ii) Log.d(...) - This syscall is a logging system call and it writes system data to the system's logcat stream. In the current system, it is called when decryption fails.

   iii) AlertDialog.create() - This syscall is responsible for creating and displaying modal pop-up windows. It has been used in the MainActivity.verify() to provide the final user feedback and in MainActivity.a() for the anti-tampering warning.

   iv) Base64.decode(...) - This syscall instructs the system utility to convert Base64-encoded strings to raw bytes. It is used in sg.vantagepoint.uncrackable1.a.a() to prepare the ciphertext for decryption.

b) In order to get the “right” output, the input is a hardcoded secret string that is masked, cryptographically via an AES encryption algorithm.

   Following are the steps used to get the correct input string:
   i. The AES key was obtained using the b() function given and the input is a hexadecimal string, “8d127684cbc37c17616d806cf50473cc". The output is a 16-byte raw key(AES-128), K.

(K = 0x8d127684cbc37c17616d806cf50473cc)

ii. Then to get the ciphertext, Base64.decode() function was used with input as the Base64 string= “5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=”. The output is a 32-byte raw ciphertext(C).

(C = 0xe5426215cb5b9a06c3a0b9e6a4bd769a49e8f074f82efc1d965b7c171451d863).

iii. I then used the source code’s sg.vantagepoint.a.a.a}(Key K, Ciphertext C) function with the already obtained AES key and ciphertext as the input to get the plaintext secret key, I.

I = “I’m a good reverse engineer”

```python
import base64
import binascii
from Crypto.Cipher import AES # type: ignore
from Crypto.Util.Padding import unpad # type: ignore

HEX_KEY_STRING = "8d127684cbc37c17616d806cf50473cc"
B64_CIPHERTEXT_STRING = "5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc="

key_bytes = binascii.unhexlify(HEX_KEY_STRING)

print(f"Step i (Key K): Length = {len(key_bytes)} bytes")
print(f"Key K (Hex): {HEX_KEY_STRING}")
print("-" * 30)


ciphertext_bytes = base64.b64decode(B64_CIPHERTEXT_STRING)
print(f"Step ii (Ciphertext C): Length = {len(ciphertext_bytes)} bytes")
print(f"Ciphertext C (Base64): {B64_CIPHERTEXT_STRING}")
print("-" * 30)



IV = b'\x00' * 16
MODE = AES.MODE_CBC

try:
    cipher = AES.new(key_bytes, MODE, IV)

    decrypted_bytes = cipher.decrypt(ciphertext_bytes)

    plaintext_bytes = unpad(decrypted_bytes, AES.block_size)
    plaintext_string = plaintext_bytes.decode('utf-8')

    print(f"Step iii (Plaintext I): Decryption Successful.")
    print(f"Correct Input: \"{plaintext_string}\"")

except ValueError as e:
    print(f"Decryption Error: {e}")
    print("Check Key, IV, and Padding assumptions.")
```

Fig: Code run for obtaining the Correct Input

iv. Additionally, I also used apktool and then analyzed the AndroidManifest.xml. This analysis confirmed the architecture and security constraints which in turn helped me get the correct input.

```
abhinavkashyap@Abhinavs-MacBook-Air-2 ~ % cd Downloads
abhinavkashyap@Abhinavs-MacBook-Air-2 Downloads % apktool d UnCrackable-Level1.apk
I: Using Apktool 2.12.1 on UnCrackable-Level1.apk with 8 threads
I: Baksmaling classes.dex...
I: Loading resource table...
I: Decoding file-resources...
I: Loading resource table from file: /Users/abhinavkashyap/Library/apktool/framework/1.apk
I: Decoding values */* XMLs...
I: Decoding AndroidManifest.xml with resources...
I: Copying original files...
I: Copying unknown files...
abhinavkashyap@Abhinavs-MacBook-Air-2 Downloads %
```

Fig: Using apktool

```
abhinavkashyap@Abhinavs-MacBook-Air-2 Downloads % cd UnCrackable-Level1
abhinavkashyap@Abhinavs-MacBook-Air-2 UnCrackable-Level1 % ls
AndroidManifest.xml     original                smali
apktool.yml             res
abhinavkashyap@Abhinavs-MacBook-Air-2 UnCrackable-Level1 % cat AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="owasp.mstg.uncrackable1">
    <application android:allowBackup="true" android:icon="@mipmap/ic_launcher" android:label="@string/app_name"
 android:theme="@style/AppTheme">
        <activity android:label="@string/app_name" android:name="sg.vantagepoint.uncrackable1.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
abhinavkashyap@Abhinavs-MacBook-Air-2 UnCrackable-Level1 %
```

Fig: Analysis of the Generated AndroidManifest.xml

v. The value of android : debuggable was observed to be false which implied that static analysis was required. The value of Main Entry point is sg.vantagepoint.uncrackable1.MainActivity, which confirms that MainActivity is the entry point of the application.

vi. The Permissions, found in apktool and AndroidManifest.xml analysis, are android.permission.INTERNET, and android.permission.ACCESS_NETWORK_STATE.

vii. Ultimately, the analysis of AndroidManifest.xml acts as a confirmation of the method we used for reverse engineering.

c) The working of the executable can be explained in following four step security and validation sequence.

i. When the application starts, the app first checks for anomalies via calls to classes b and c. If any of these calls return true, then the application displays a warning dialog and immediately calls System.exit(0).

ii. When the application is running, the user enters text and presses the “Verify” button, which in turn triggers the verify(View view) method. This method then uses findViewById(R.id.edit_text).getText().toString() to get the user’s input.

iii. The user input, a string, is passed to the validation method sg.vantagepoint.uncrackable1.a.a(String). This method then decodes a 16-byte static key, via hex-decoding. It then decodes a static ciphertext, via Base64-decoding, followed by usage of sg.vantagepoint.a.a.a, a cryptography function to decrypt the ciphertext. It then performs a direct string comparison between the user’s input and the decrypted secret string.

iv. The verify method then uses an AlertDialog call to provide feedback, either “Success” or “Nope…”.

d) i. For the task of reverse engineering and analysis, my preference would depend on my ultimate goal. If my goal is to do high level analysis, in case of finding logical flaws in code or if reverse engineering some simple, hardcoded apk files, then I would prefer output from Hex-Rays or Ghidra's decompiler. This is decompilation.

ii. Whereas, if my goal is to do a low-level, granular analysis to find flaws at system level(like Buffer Overflow) or if the file to be reverse engineered is complex and the secret is hidden from the High level code, then I would prefer disassembly.

iii. The outputs produced by Hex-Rays/Ghidra Decompiler are structured code(similar to C or Java), which makes it easy for me to follow high-level logic and reduces the cognitive load significantly.

iv. The other option is the output from disassembly which outputs low-levle process instructions(similar to assembly language). This requires expertise in topics like OS, ISA, mnemonics, etc. This makes it a choice of poor usability.

v. The output produced by Hex-Rays or Ghidra are much more aligned with a User’s as well as a Designer’s Mental Model. It retains human produced and understandable labels like variable names, class names, API calls, etc.

vi. Whereas the other form of output closely aligns with System Model.

vii. In cases similar to this question, outputs from Hex-Rays or Ghidra make the process much more efficient as they instantly show the hardcoded secrets like “8d12…”. Whereas the other option is much slower.

viii. In some cases the outputs from Hex-Rays or Ghidra are imperfect. Compilers can often mislead the outputs whereas the other option represents the exact code the machine executed making it most accurate.

ix. In conclusion, I would say that decompiled output or output from Hex-Rays or Ghidra is the default choice based on the balance speed, and readability. Disassembled outptuts are necessary when the decompiled outputs can’t be trusted or there is a need of more granular, low-level control.

**Ans 3.**

To analyze the current authentication system at IIIT Delhi, following are the major systems used as reference:

i. The ERP system - https://iiitd.nurecampus.com/

ii. IIIT Delhi Mail system using Gmail

iii. Fee Payment Portal - https://payments.iiitd.edu.in/pg/studentsfee/

iv. IIIT Delhi’s Library Portal - https://library.iiitd.edu.in/

v. CG/SG Registration Portal - http://sgcw.iiitd.edu.in/student/home/

vi. IIIT Delhi’s Wifi

Issues with current Authentication System:

- The current system, in many cases, exhibits a conflict between a strict Designer Model and the User Model, raising several usability issues.
- High Cognitive Load: The initial assignment of a random, complex 8-character password ensures security but is opposed to the user's need for memorability and recognition.

- Due to complex credentials, students adopt insecure coping techniques like writing the password down or relying on password managers. This makes the user's physical environment a vulnerability and a target for attackers.

- As mentioned that we are discussing 6 different systems, the students are required to manage logins for all these, along with the FortiClient VPN client. While some of the credentials might be reused, in order to maintain high security standards, students often use inconsistent passwords thus increasing their cognitive load.

- The addition of the CW/SG portal's network constraint (must use VPN) and the requirement of a separate VPN client (FortiClient) also adds additional burden on the user. Not only does this mechanism increase the cognitive load but also makes it difficult to navigate for a not so tech-friendly user.

- The default, forced reuse of the ERP password for the highly sensitive VPN/WiFi access, creates a critical flaw: a single compromise leaks credentials to the entire campus network infrastructure.

- Moreover, the control to change password for VPN lies in the hands of the Admin only. This eliminates the user's ability to mitigate a known security incident (e.g., if their password is leaked), leading to a feeling of learned helplessness and a fundamental violation of user control over their own security.

- I also faced some frustrating issues while using ERP's login portal. On detailed inspection of the console I found critical console errors like (CORS, 500 server errors) and reCAPTCHA failures. These failures slow down the login process, introducing ambiguity and disappointment.

Suggestions to Improve the System

- Mandate Universal Single Sign-On(SSO) -
    - Necessitating the use of the College Gmail/Google ID as the only login method for all the services. Integrate the VPN system to use Google ID as well, e.g. via RADIUS.
    - This improves usability by using one login, one identity mechanism, while simultaneously increasing security by enforcing Google's built in 2FA for all access points.

- Update the Password Policy (For non-SSO Systems)-
    - There would definitely be systems that are not covered by SSO, so replace the 8-character password requirement with a focus on passphrase length(minimum 16 characters). Example: I_Lo_cats_&_dogs.

    - The passphrases are easy to remember in comparison to random complex passwords, guarantee security as they are hard to crack. These are based on preferences generally which aren't documented.

- Remove all Knowledge Based Security Questions:
    - All the recovery and reset mechanisms must leverage the existing 2FA mechanism provided by the Google SSO.
    - This prevents Data Mining and Social Engineering-based attacks.
- Resolve all identified backend (500 errors, CORS) and frontend (reCAPTCHA) issues.
- Minimize the reliance on the FortiClient VPN for services like CW/SG. This can be done using alternatives like using authenticated public facing domains or more flawlessly integrated client-less VPN options.
- Services that are related should be grouped together and provided through a single interface. For example: the ERP/NureCampus could be integrated with the payment portal, CW/SG portal and Library portal. This would reduce the cognitive load.

## *Bonus Question: Decompilation vs. Disassembly: Analysis and Use Case

**a. Analysis**

i. When doing reverse engineering, the choice between decompilation and disassembly totally depends on the motive i.e. if the goal is to get a high level overview of the application logic then use decompilation or else if the goal is to analyze at a granular level i.e. the low level security implementations, disassembly is the way ahead.

ii. After decompilation, the output is in the form of High level language like Java, C++, etc. whereas in the case of disassembly the output is in the form of low level processor instructions like the Assembly Langauge.

iii. Decompilation focuses on the structure and components of the output high level code like classes, variables, imports, etc. On the other hand, disassembly analyzes low-level execution components like processor registers, heap, stack, and syscalls.

iv. Decompilation has more usability than decompilation since it's much more readable, and is also the fastest way to understand application flow and identify hardcoded

secrets. Decompilation requires the user to have in-depth knowledge of core system topics like OS, ISA, mnemonics, etc.

v. Decompilation is executed with focus on finding vulnerabilities like logical flaws(e.g. Hardcoded secrets, weak password checks,etc). Whereas, disassembly focusses on finding memory flaws like Buffer Overflows, instruction-level anti-debugging checks, etc.

**b. Use Case**

i. Question 2 illustrates the utility of Decompilation perfectly.

- In Q2, we quickly identified the variables, classes, and function calls.
- We used static analysis to find the secret.
- Using disassembly in this case would have taken longer time as we would have had to manually track memory and registers to reconstruct the hardcoded strings and API calls.

ii. Disassembly is necessary when the application logic is masked from the high-level language. In cases where an app has checks like anti-debugging mechanism, decompilation may miss subtle, low level checks.

iii. For hardened apps that hide secrets and have flaws(e.g.: Buffer Overflow) in native C/C++ libraries, decompilation would produce incorrect or complex output. In such cases disassembly is the most accurate way.

iv. So it can be concluded for initial, high level checks for simple code flaws, decompilation is superior as it offers better usability and speed. For in depth vulnerability analysis or overcoming strong low-level security controls, disassembly is the better choice.