Kennedy Anukam

CS 453

Professor Doherty

April 30, 2021

Health Tap Application (V7.27.2-Android)

# V1: Architecture, Design and Threat Modeling Requirements

|  |  |  |
| --- | --- | --- |
| **1.1** | MSTG-ARCH-1 | All app components are identified and known to be needed. |

**I did not see any app components that seemed unneeded. The application passed this standard.**

**1.2** MSTG-ARCH-2 Security controls are never enforced only on the client side, but on the respective remote endpoints.

**The application utilized cookies for authentication when requests were made to the serverside. The application passed this standard.**

**1.3** MSTG-ARCH-3 A high-level architecture for the mobile app and all connected remote services has been defined and security has been addressed in that architecture.

**I do not have access to the high-level architecture of the app. I was not able to test this standard.**

**1.4** MSTG-ARCH-4 Data considered sensitive in the context of the mobile app is clearly identified.

**Sensitive data was not identified within the application. The application failed this standard.**

**1.5** MSTG-ARCH-5 All app components are defined in terms of the business functions and/or security functions they provide.

**I did not find definitions of the app components within the application. The application failed this standard.**

**1.6** MSTG-ARCH-6 A threat model for the mobile app and the associated remote services has been produced that identifies potential threats and countermeasures.

**I did not have access to the threat model that the Health Tap developers potentially produced. I was not able to test this standard.**

**1.7** MSTG-ARCH-7 All security controls have a centralized implementation.

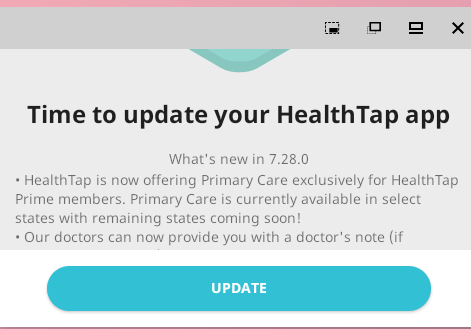
**I did not have access to their security controls used on the server side. The implementation was not known to me. I was not able to test this standard.**

**1.8** MSTG-ARCH-8 There is an explicit policy for how cryptographic keys (if any) are managed, and the lifecycle of cryptographic keys is enforced. Ideally, follow a key management standard such as NIST SP 800-57.

**I did not have access to the cryptographic key policy. I was not able to test this standard.**

**1.9** MSTG-ARCH-9 A mechanism for enforcing updates of the mobile app exists.

**I tested this standard by trying to log into the application with a version of the Health Tap application that is not the most latest. I was prompted with a message to update the application and was not allowed to continue:**

****

**The application passed this standard.**

**1.10** MSTG-ARCH-10 Security is addressed within all parts of the software development lifecycle.

**I was not able to test this standard as the software development lifecycle used for the app was not known to me.**

**1.11** MSTG-ARCH-11 A responsible disclosure policy is in place and effectively applied.

**I was not able to find a responsible disclosure policy for the application. The application failed this standard.**

**1.12** MSTG-ARCH-12 The app should comply with privacy laws and regulations.

**The app complies with privacy laws and regulations according to the privacy policy. The application passed this standard.**

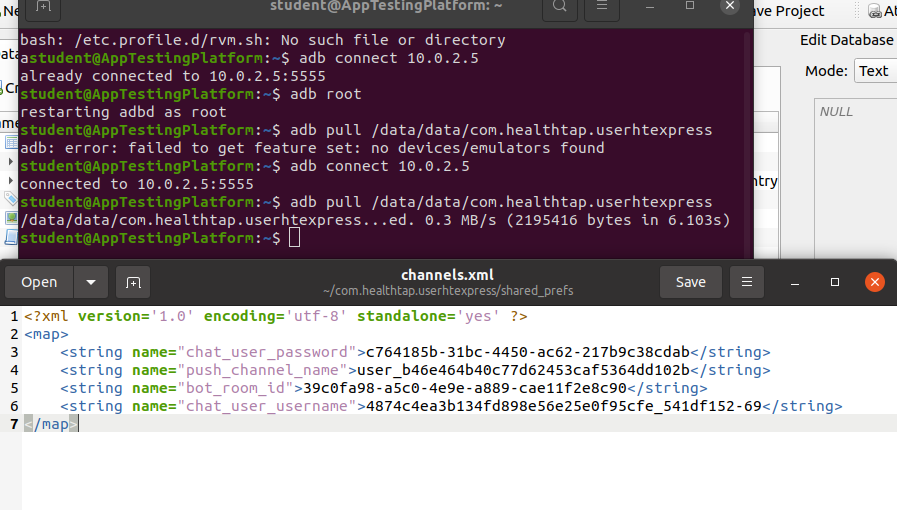
V2: Data Storage and Privacy Requirements

**2.1** MSTG-STORAGE-1 System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys.

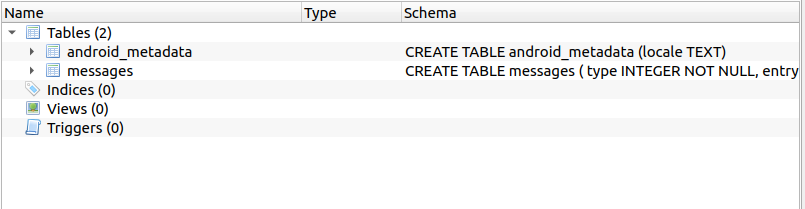
**2.2** MSTG-STORAGE-2 No sensitive data should be stored outside of the app container or system credential storage facilities.

**To test this standard, I first looked at the files in /data/data/<package-name>/.**

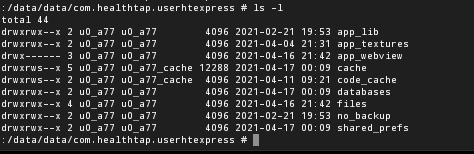
**I found a sensitive shared preference; the chat user password is stored:**

****

**The only sqlite db was a google app measurement database which did not contain any sensitive information:**

****

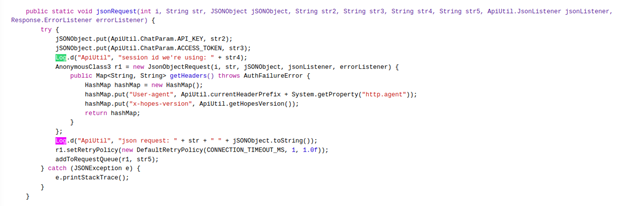
**I then checked the permissions of the files in the /data/data/<package-name>:**

****

**They adhere to the guidelines specified in the OWASP Guide. Overall, the application fails these standards due to the preference file containing potentially sensitive information.**

**2.3** MSTG-STORAGE-3 No sensitive data is written to application logs.

**I tested this standard by searching the applications source code for keywords involving log. MobSF made this simple as it showed all of the code instances where logging was performed. I found a spot in the code where the session is logged for JSON requests:**

****

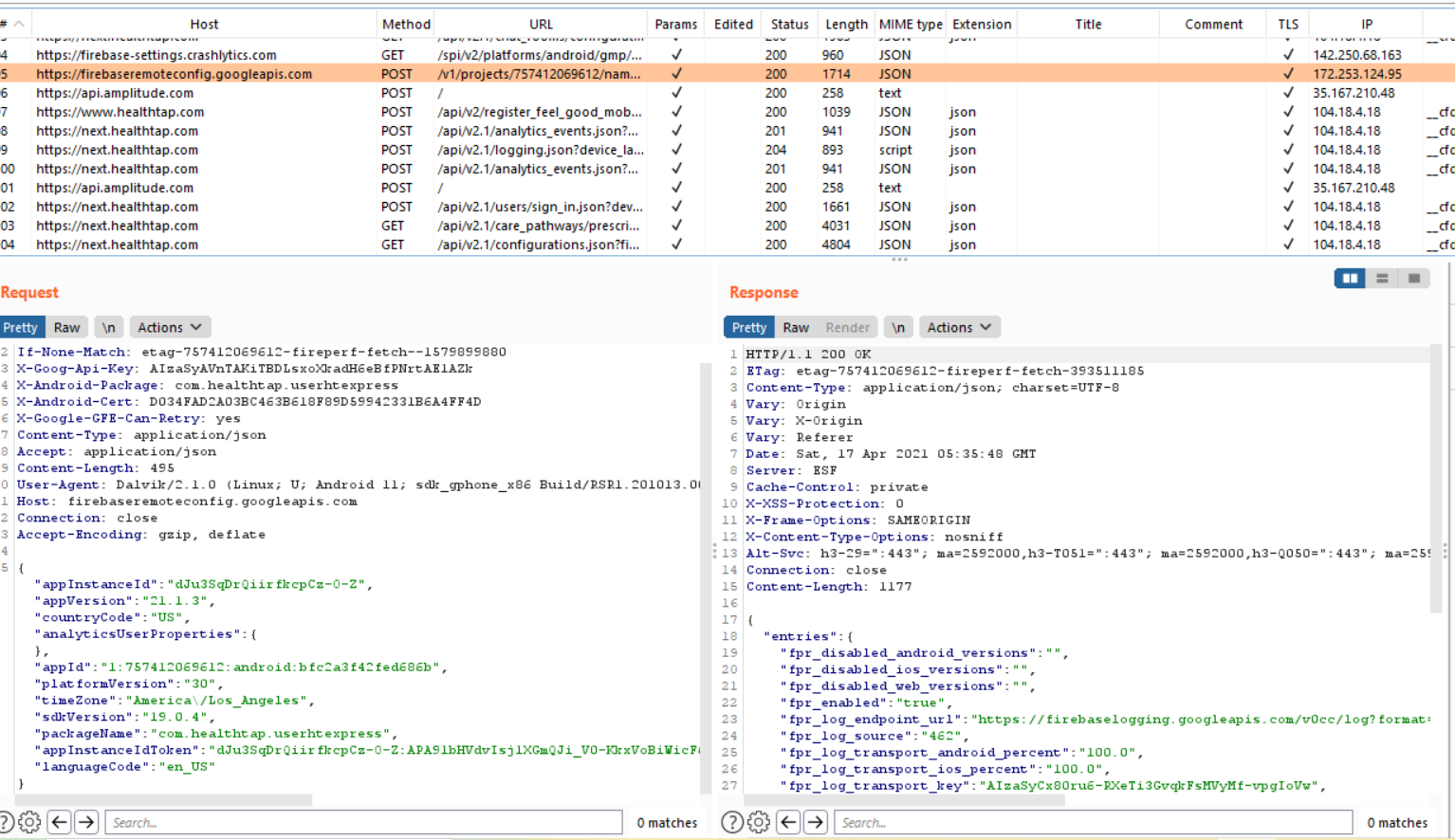
**I performed dynamic testing as well by looking in the /data/data/<package-name> for application logs. This was done after utilizing many of the apps functions. For some reason, I could not find the application logs in the directory:**

****

**Nonetheless, the application fails this standard as the session ID and JSON Request are logged for requests.**

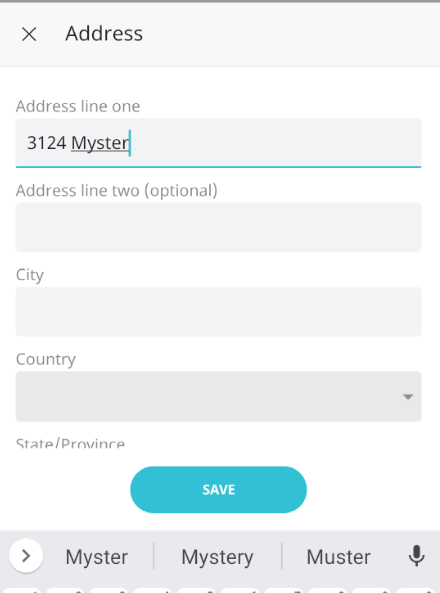
MSTG-STORAGE-4 No sensitive data is shared with third parties unless it is a necessary part of the architecture.

**I tested this standard by utilizing Burpsuite to see if the application sends sensitive data to third parties.  I performed various operations in the app and third parties are infrequently contacted and sensitive data is not sent. 1 of the two third parties contacted was Google Firebase and nothing sensitive was sent:**

**It is a necessary part of the architecture. The application passes this standard.**

**2.5** MSTG-STORAGE-5 The keyboard cache is disabled on text inputs that process sensitive data.

**I tested this standard by performing dynamic analysis on the app. I clicked on input fields that take in sensitive data and checked if strings were suggested. When I was entering in a fake address, strings were being suggested:**

****

**The application failed this standard as a person's address is sensitive and the keyboard cache is not disabled.**

**2.6** MSTG-STORAGE-6 No sensitive data is exposed via IPC mechanisms.

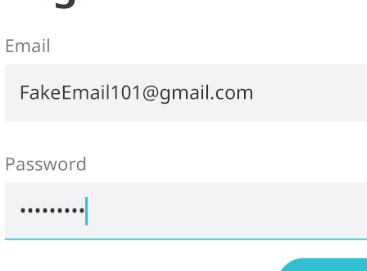
**I tested this standard by looking for exported activities, receivers, content providers. I found two services exported and one receiver exported in the Manifest:**

****

**They are all related to Google and do not seem like an issue. I did not test them with Drozer as they are not com.healthtap related. The application passes the standard.**

**2.7** MSTG-STORAGE-7 No sensitive data, such as passwords or pins, is exposed through the user interface.

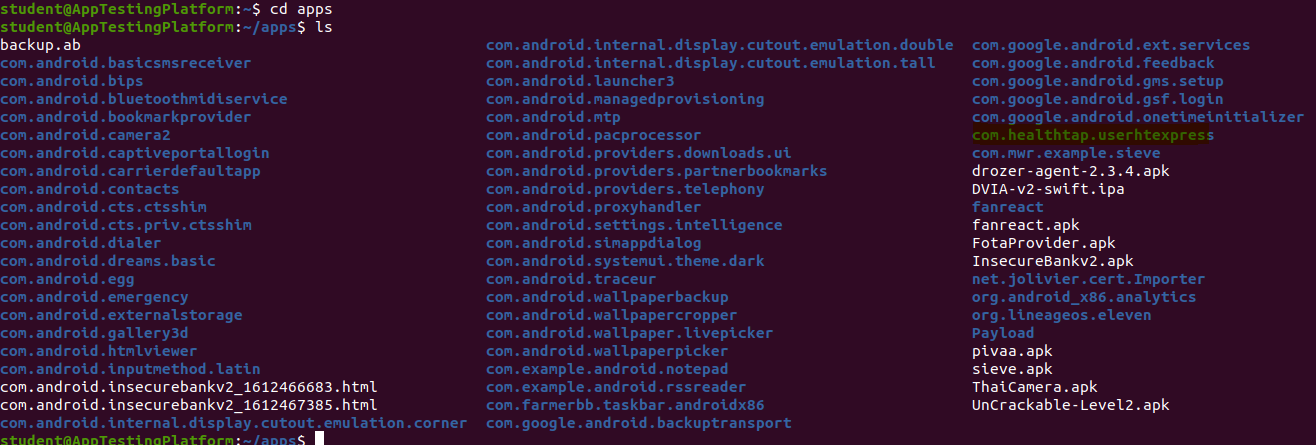
**The application masks sensitive user input (passwords):**

****

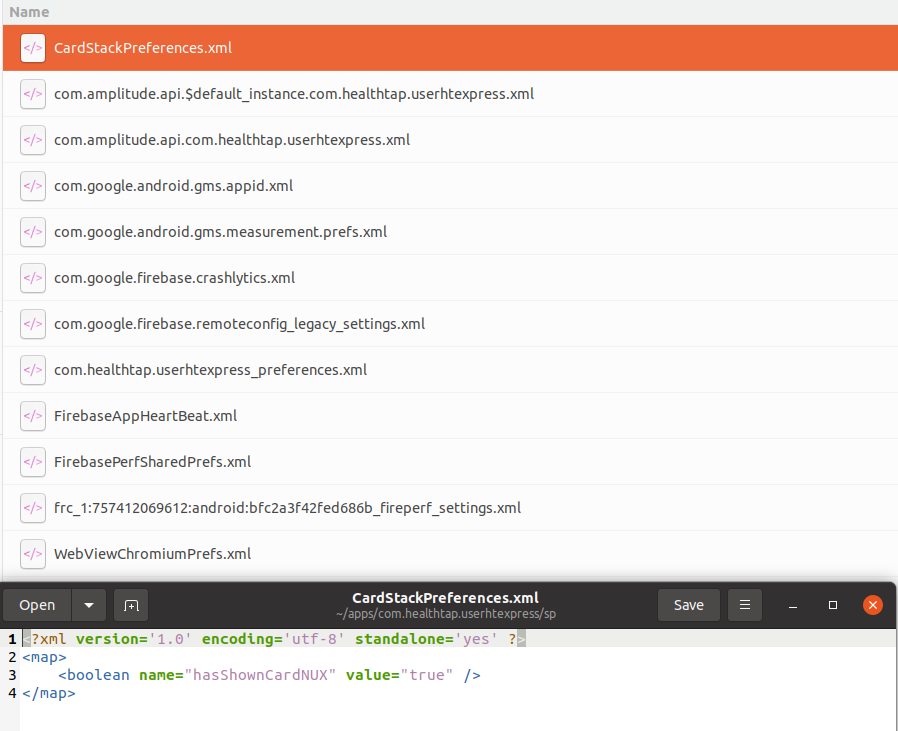
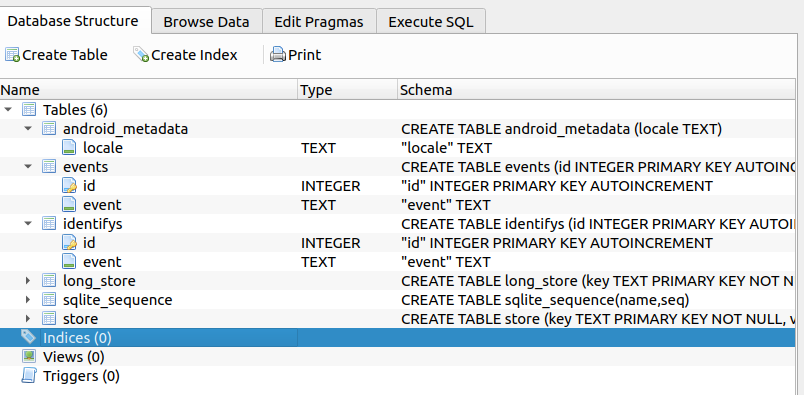
**The application passes the standard.**

**2.8** MSTG-STORAGE-8 No sensitive data is included in backups generated by the mobile operating system.

**I tested this standard by obtaining a backup of the application and testing it for sensitive information:**

****

**I did not find any sensitive data in the db or sp files:**

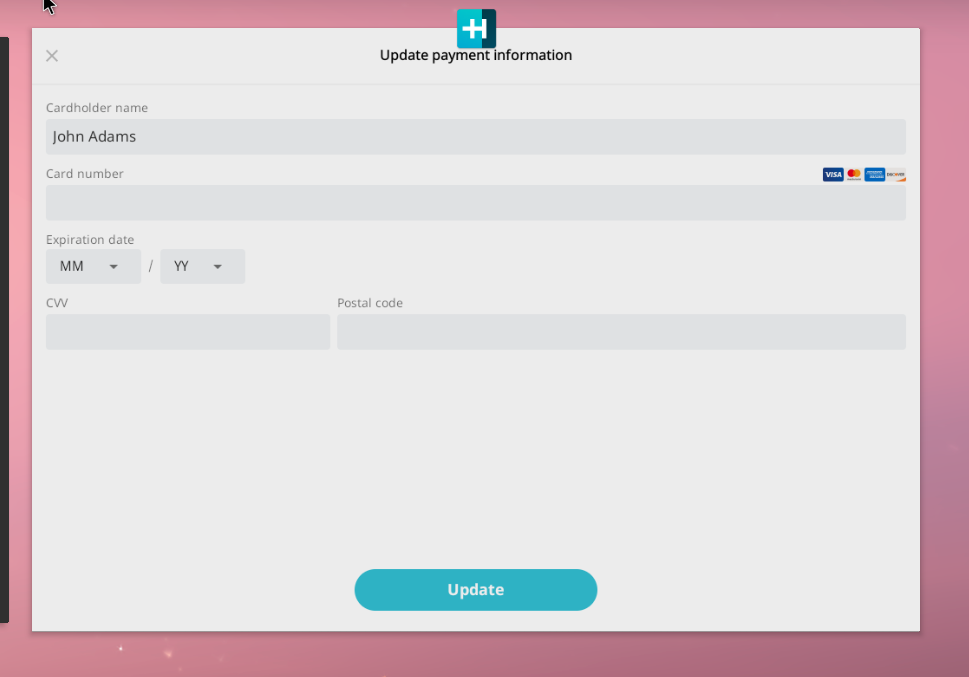
****

**The application passed the standard.**

**2.9** MSTG-STORAGE-9

**The app removes sensitive data from views when moved to the background.**

**I performed dynamic testing for this standard. I first navigated to a view on the application that shows sensitive information. I then hit the home button. I next clicked the app switcher button to see if the snapshot was empty:**

****

**The app failed the standard as sensitive information was still present in the view.**

**2.10** MSTG-STORAGE-10 The app does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use.

**Sensitive data is copied to the clipboard. In one file, the appointment registration code is copied to the clipboard. The application failed the standard.**

**2.11** MSTG-STORAGE-11 The app enforces a minimum device-access-security policy, such as requiring the user to set a device passcode.

**The application did not enforce a minimum device-access-security policy. This is something that is concerning for a telehealth application.**

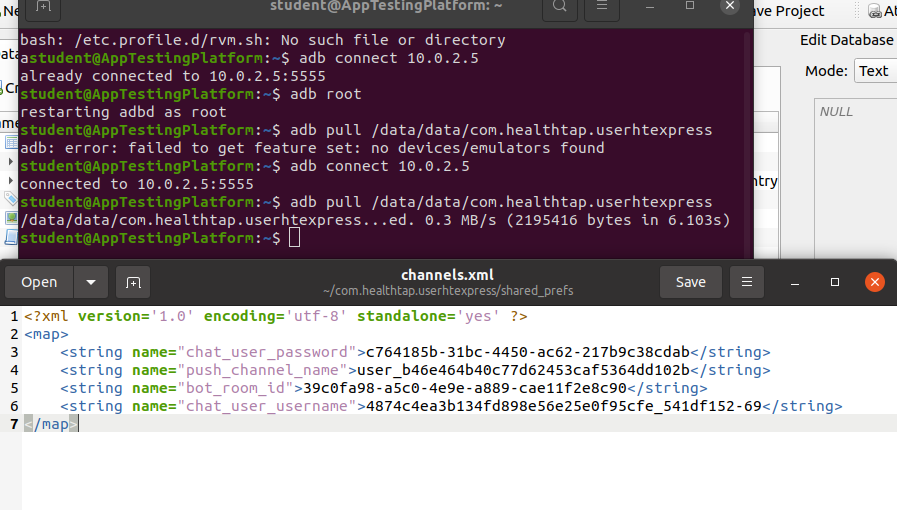
**2.12** MSTG-STORAGE-12 The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.

**The privacy policy educates the user about the types of personally identifiable information processed. The application passed the standard.**

**2.13** MSTG-STORAGE-13 No sensitive data should be stored locally on the mobile device. Instead, data should be retrieved from a remote endpoint when needed and only be kept in memory.

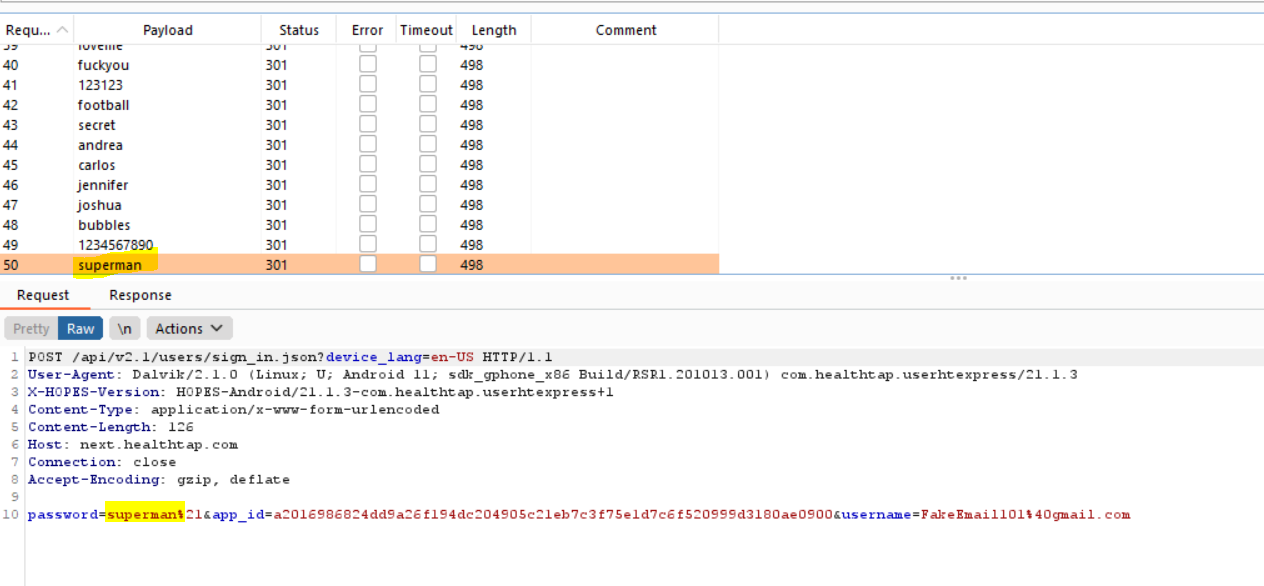
**2.14** MSTG-STORAGE-14 If sensitive data is still required to be stored locally, it should be encrypted using a key derived from hardware backed storage which requires authentication.

**The application failed these standards. The chat user password is stored in the shared preferences:**



**2.15** MSTG-STORAGE-15 The app’s local storage should be wiped after an excessive number of failed authentication attempts.

My apps local storage was not cleared when performing 100 failed logins with the rockyou.txt file. The application failed the standard.



# V3: Cryptography Requirements

**3.1** MSTG-CRYPTO-1 The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.

**I was not able to test this standard. I did not have access to the server-side code.**

**3.2** MSTG-CRYPTO-2 The app uses proven implementations of cryptographic primitives.

**I was not able to test this standard. I did not have access to the server-side code.**

**3.3** MSTG-CRYPTO-3 The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.

**I was not able to test this standard. I did not have access to the server-side code.**

**3.4** MSTG-CRYPTO-4 The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.

**The application utilizes instances of MD4 and MD5 which are known as a weak hashing algorithm:**

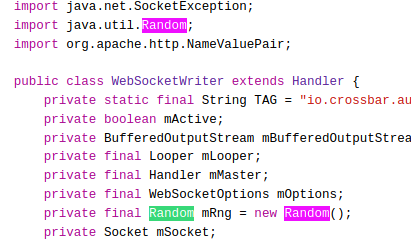
**  The application failed this standard.**

**3.5** MSTG-CRYPTO-5 The app doesn't re-use the same cryptographic key for multiple purposes.

**I was not able to test this standard. I did not have access to the server-side code.**

**3.6** MSTG-CRYPTO-6 All random values are generated using a sufficiently secure random number generator.

**The app uses insecure random number generators:**



**The application failed this standard.**

# V4: Authentication and Session Management Requirements

**4.1** MSTG-AUTH-1 If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.

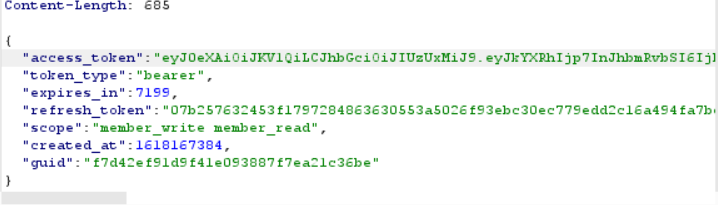
**The app does not utilize remote services. The application passed this standard.**

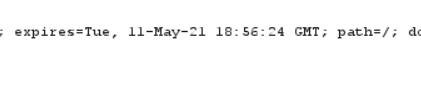
**4.2** MSTG-AUTH-2 If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user's credentials.

**Stateless session management is used. The application passed this standard.**

**4.3** MSTG-AUTH-3 If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.

**Access tokens and refresh tokens are used, expiration dates are used for a duration of a month:**

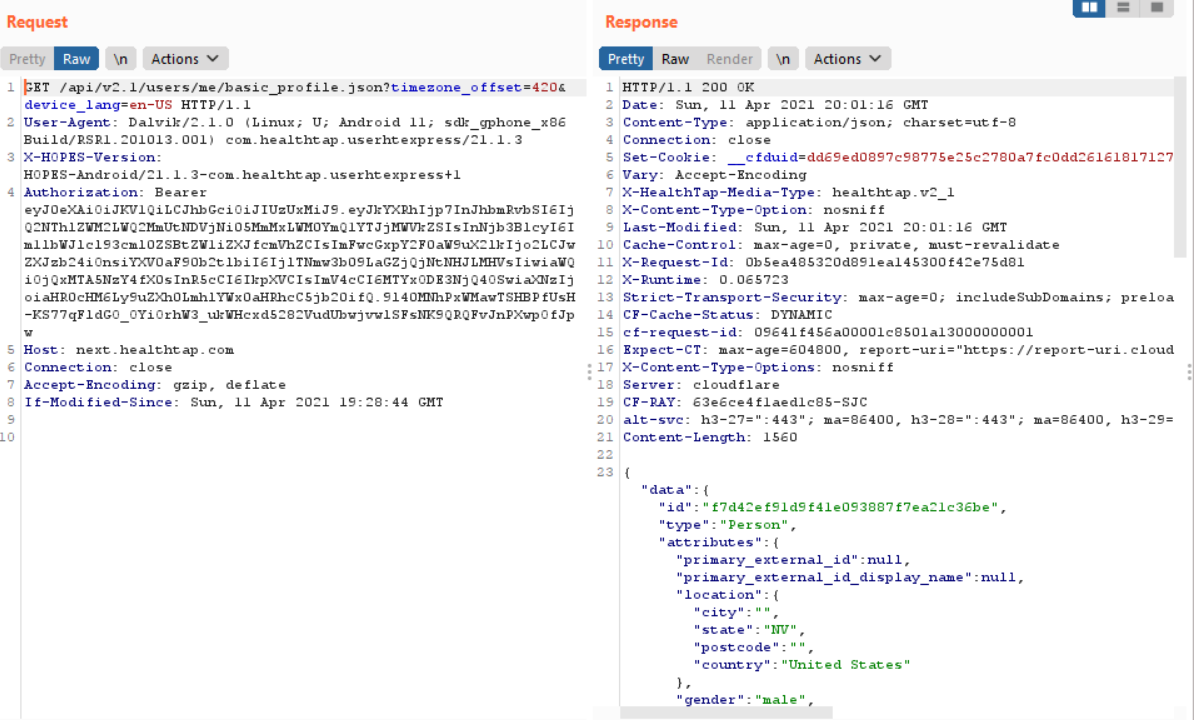




**I am assuming that the token was created with a secure algorithm. The application passed the standard.**

**4.4** MSTG-AUTH-4 The remote endpoint terminates the existing session when the user logs out.

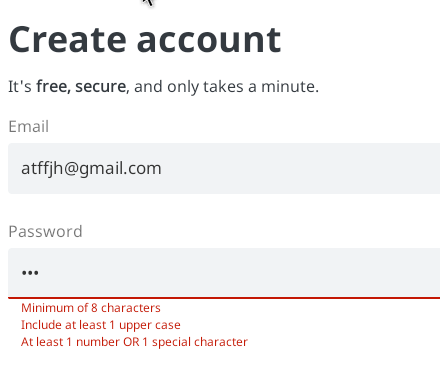
**I tested this standard by first logging into the application on the emulator. I accessed a resource that requires authentication then logged out. I then tried to access the same resource when logged out:**

****

**This indicated that logging out does not terminate the session. The application failed the standard.**

**4.5** MSTG-AUTH-5 A password policy exists and is enforced at the remote endpoint.

**For this standard, I performed dynamic testing by utilizing the app on the Android emulator. Through navigating to the Forgot Password screen, I found the following password policy:**

****

**The password policy does not adhere to the following guidelines:**

**Password Length:**

**1. Minimum password length (10 characters) should be enforced.**

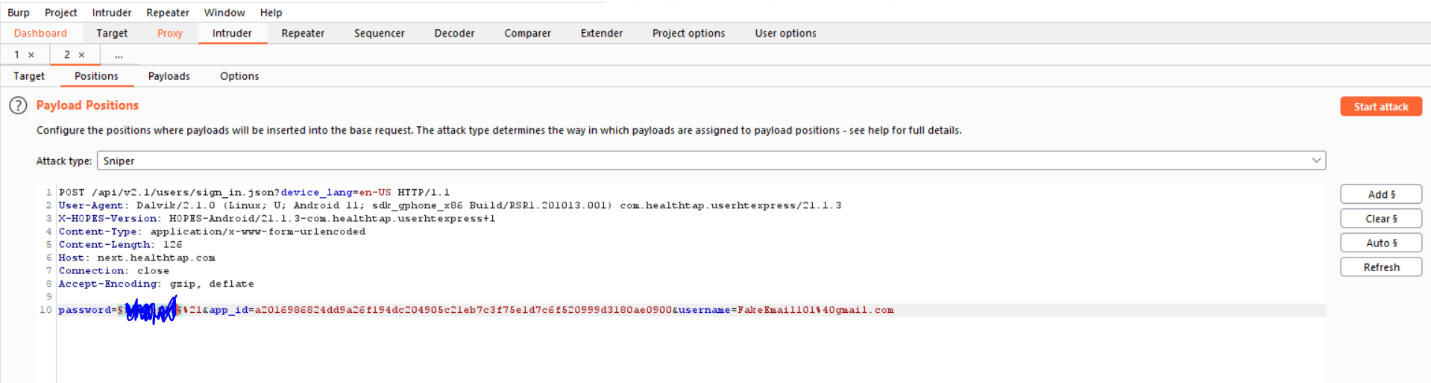
**Password Complexity:**

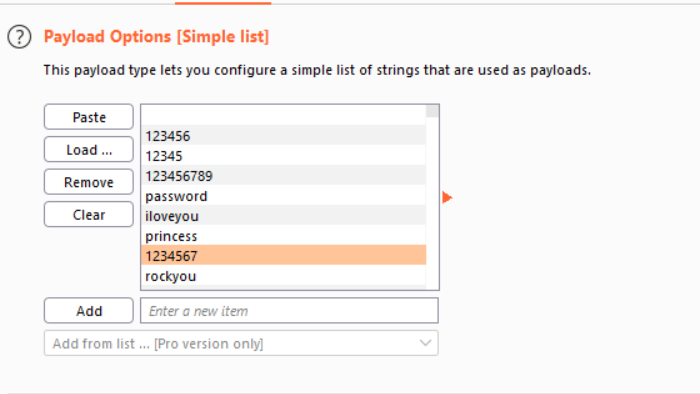
**1. at least one lowercase character (a-z)**

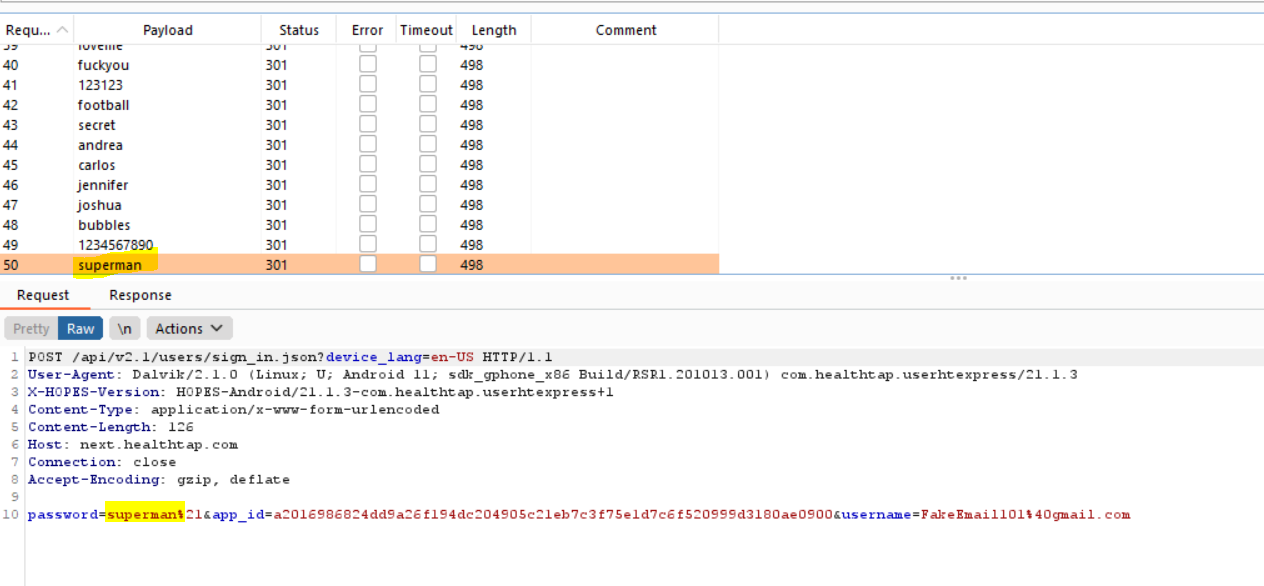
**The application failed the standard.**

**4.6** MSTG-AUTH-6 The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.

**I tested the standard through utilizing the Burp Intruder tool. I set up burp suite to first establish the packet format and endpoint for login requests:**

**I used the rockyou.txt file for the attack:**

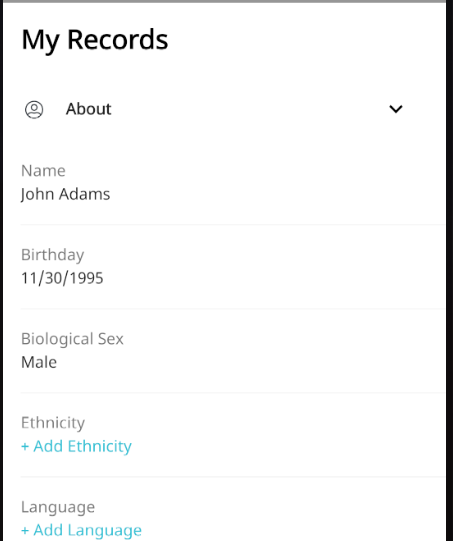
****

****

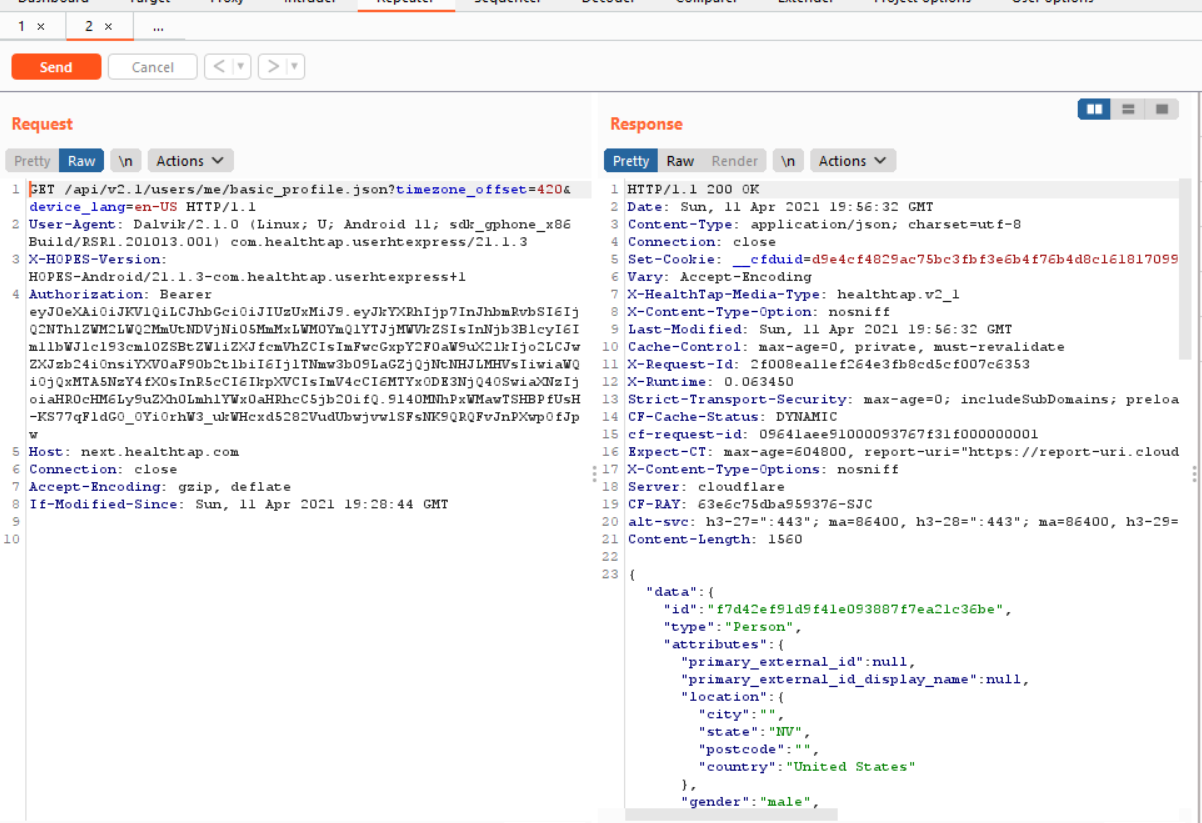
**The server’s responses did not indicate that there is a mechanism in place to protect against the submission of credentials an excessive amount of times. I let 100 words in the list run and the status code was the same for all of them. This leads me to believe that a rate limiter is not in place for login. The application failed the standard.**

**4.7** MSTG-AUTH-7 Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.

**I tested this standard by first logging into the application. I then accessed a resource which requires authentication:**

****

**I waited intervals of 5 minutes reaching up to 30 minutes. I accessed the data proxying my requests through burpsuite as an interception proxy. After 30 minutes, data could still be accessed:**

****

**The app failed this standard.**

**4.8** MSTG-AUTH-8 Biometric authentication, if any, is not event-bound (i.e. using an API that simply returns "true" or "false"). Instead, it is based on unlocking the keychain/keystore.

**The app passed the standard as it does not use any biometric authentication.**

**4.9** MSTG-AUTH-9 A second factor of authentication exists at the remote endpoint and the 2FA requirement is consistently enforced.

**The app does not use 2FA, the app failed the standard.**

**4.10** MSTG-AUTH-10 Sensitive transactions require step-up authentication.

**The app does not utilize step-up authentication even with sensitive transactions. The app failed the standard.**

**4.11** MSTG-AUTH-11 The app informs the user of all sensitive activities with their account. Users are able to view a list of devices, view contextual information (IP address, location, etc.), and to block specific devices.

**The app does not inform the user of any sensitive transactions.** **The app failed the standard.**

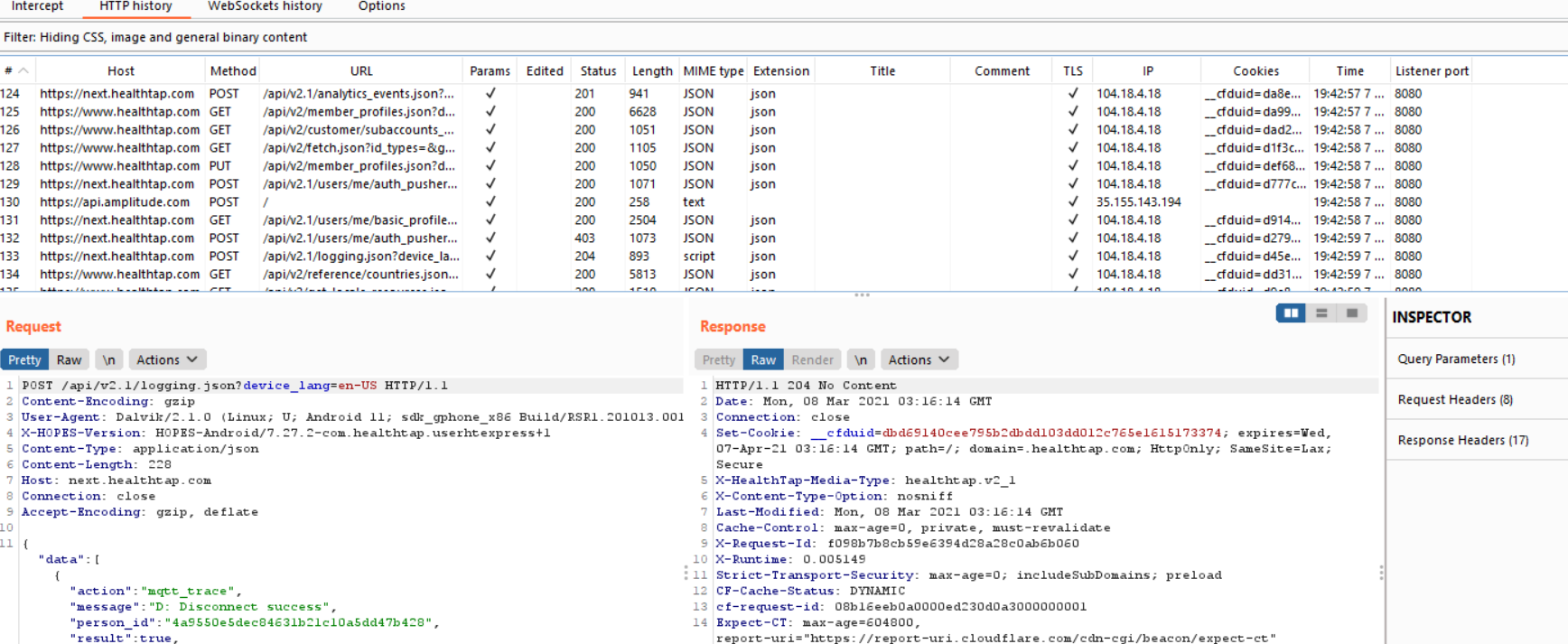
**4.12** MSTG-AUTH-12

Authorization models should be defined and enforced at the remote endpoint.

**I do not have access to the server side code. I was not able to test this.**

**5.1** MSTG-NETWORK-1 Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.

**Data is encrypted on the network using TLS. The app passes this standard.**



**5.2** MSTG-NETWORK-2 The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.

**The application has insecure TLS certificate domain:**

****

The application failed this standard.

**5.3** MSTG-NETWORK-3 The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.

**5.4** MSTG-NETWORK-4 The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA.

**Something unusual that I noticed about my app is the tag <certificates src="user" /> was already in the res/xml/network\_security.cfg file. This represents that the app trusts all user added certificates. The app failed these standards.**

**5.5** MSTG-NETWORK-5 The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery.

**The app relies on email only for account recovery. The app failed this standard.**

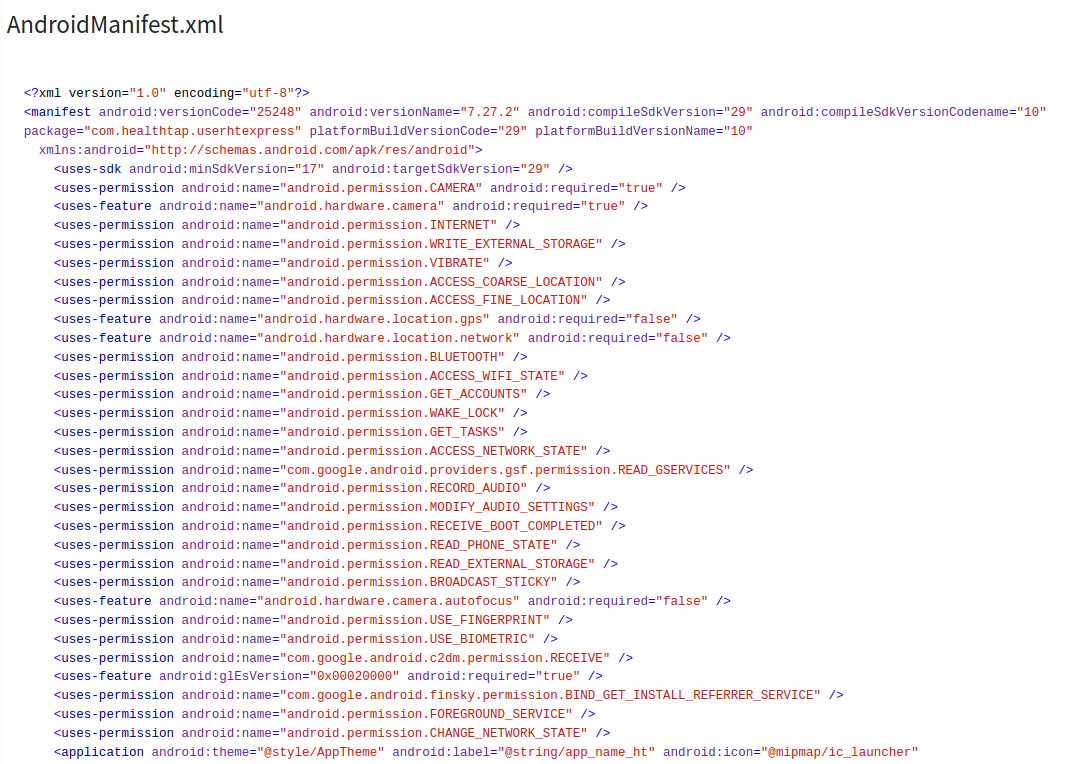
**5.6** MSTG-NETWORK-6 The app only depends on up-to-date connectivity and security libraries.

**I was not able to test this standard.**

# V6: Platform Interaction Requirement

**6.1** MSTG-PLATFORM-1The app only requests the minimum set of permissions necessary.

**I completed this portion by examining the ANDROIDMANIFEST.XML file. Through utilizing MobSF, I inspected the manifest. None of the permissions seem out of the ordinary for a tele-health application:**

****

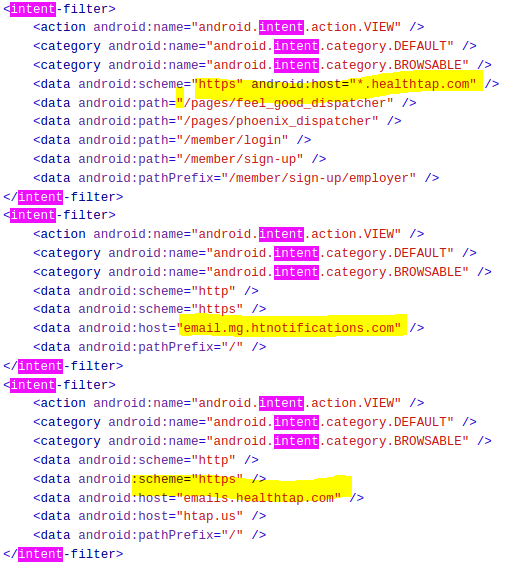
**The application passed this standard.**

**6.2** MSTG-PLATFORM-2 All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.

**I was not able to test this standard.**

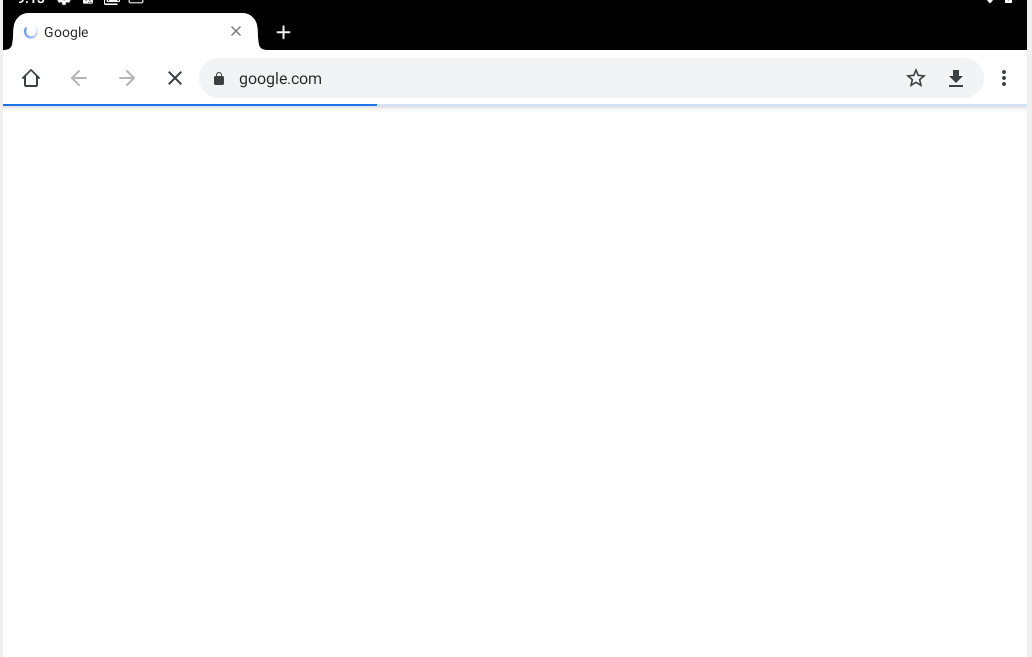
**6.3** MSTG-PLATFORM-3 The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.

**I tested this standard by examining the ANDROIDMANIFEST.XML file. I looked for activities that had url related intents. I found one activity that had 3 URL related intents:**

****

**I utilized Drozer to see if it was possible to go to other sites:**

**The command caused google.com to be visited in the emulator:**

****

**The application failed this standard.**

**6.4** MSTG-PLATFORM-4 The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected.

**I tested this standard by looking for exported activities, receivers, content providers. I found two services exported and one receiver exported in the Manifest:**

****

**They are all related to Google and do not seem like an issue. I did not test them with Drozer as they are not com.healthtap related. The application passed this standard.**

**6.5** MSTG-PLATFORM-5 JavaScript is disabled in WebViews unless explicitly required.

**I was not able to test this standard.**

**6.6** MSTG-PLATFORM-6 WebViews are configured to allow only the minimum set of protocol handlers required (ideally, only https is supported). Potentially dangerous handlers, such as file, tel and app-id, are disabled.

**Outside the scope of this course.**

**6.7** MSTG-PLATFORM-7 If native methods of the app are exposed to a WebView, verify that the WebView only renders JavaScript contained within the app package.

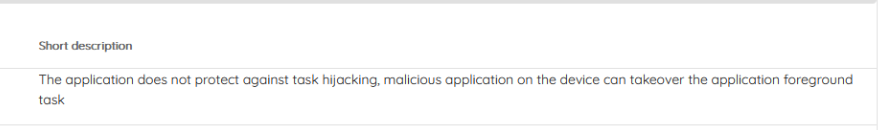
**Outside the scope of this course.**

**6.8** MSTG-PLATFORM-8 Object deserialization, if any, is implemented using safe serialization APIs.

**Outside the scope of this course.**

**6.9** MSTG-PLATFORM-9 The app protects itself against screen overlay attacks. (Android only)

**The application is vulnerable to task hijacking:**



**The application failed this standard.**

**6.10** MSTG-PLATFORM-10 A WebView's cache, storage, and loaded resources (JavaScript, etc.) should be cleared before the WebView is destroyed.

**I was not able to test this standard.**

**6.11** MSTG-PLATFORM-11 Verify that the app prevents usage of custom third-party keyboards whenever sensitive data is entered (iOS only).

**The application is an Android app. I was not able to test this standard.**

# V7: Code Quality and Build Setting Requirements

**7.1** MSTG-CODE-1 The app is signed and provisioned with a valid certificate, of which the private key is properly protected.

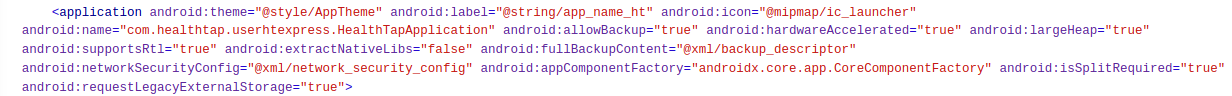
**I tested this standard through looking at MobSF. The application is signed:**

****

**The application passed this standard.**

**7.2** MSTG-CODE-2 The app has been built in release mode, with settings appropriate for a release build (e.g. non-debuggable).

**I tested this standard through inspecting the Android manifest. The word debuggale was not in the application element which represents the app is not debuggable as that is the default if it is not specified:**

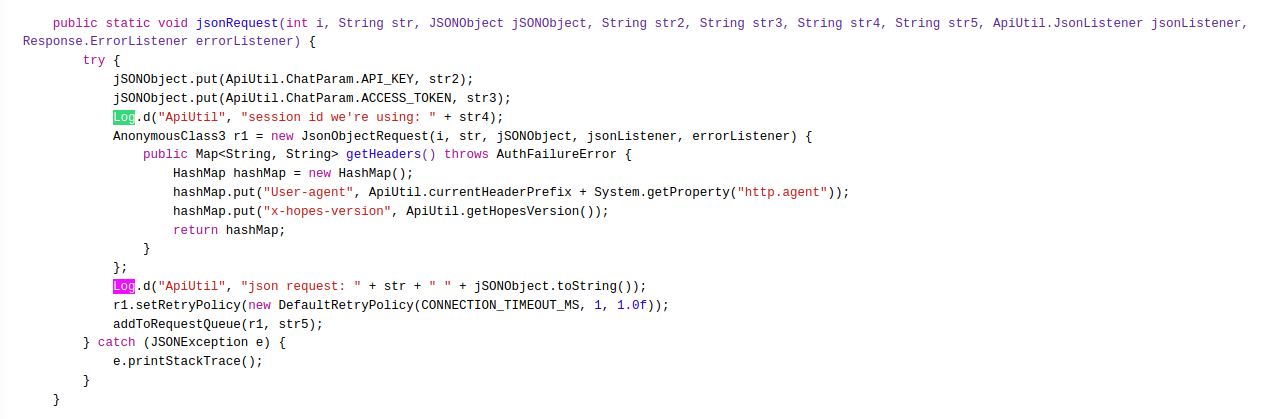
**  The application passed this standard.**

**7.3** MSTG-CODE-3 Debugging symbols have been removed from native binaries.

**Outside the scope of this course.**

**7.4** MSTG-CODE-4 Debugging code and developer assistance code (e.g. test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages.

**JSON requests are logged:**



The application failed this standard.

**7.5** MSTG-CODE-5 All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.

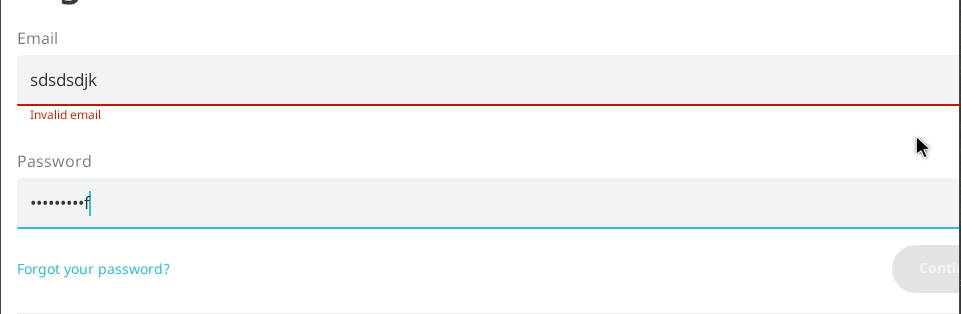
**Outside the scope of this course.**

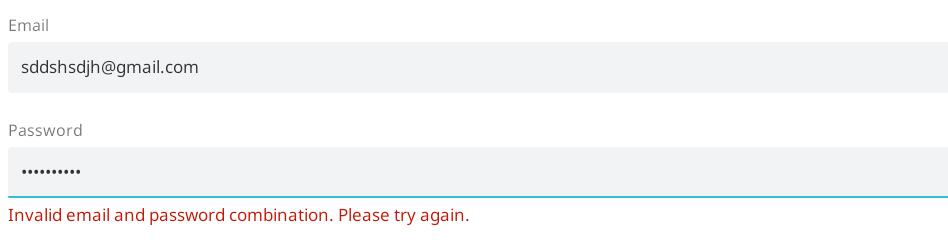
**7.6** MSTG-CODE-6 The app catches and handles possible exceptions.

**7.7** MSTG-CODE-7 Error handling logic in security controls denies access by default.

**The application passed these standards.**

**Invalid email:**



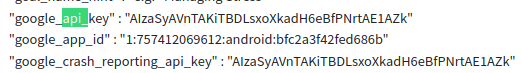


**7.8** MSTG-CODE-8 In unmanaged code, memory is allocated, freed and used securely.

**Outside the scope of this course.**

**7.9** MSTG-CODE-9 Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated.

**Some keys obfuscated:**





**MobSF was able to detect the non-obfuscated keys. The app failed this standard.**

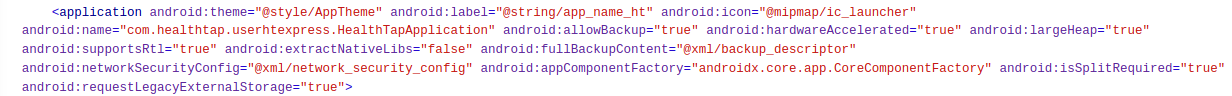
# V8: Resilience Requirements

**8.1** MSTG-RESILIENCE-1 The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.

**I did not find any code to detect for jailbroken devices. The application failed this standard.**

**8.2** MSTG-RESILIENCE-2 The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.

**I tested this standard through inspecting the Android manifest. The word debuggale was not in the application element which represents the app is not debuggable as that is the default if it is not specified:**

****

**The application passed this standard.**

**8.3** MSTG-RESILIENCE-3 The app detects, and responds to, tampering with executable files and critical data within its own sandbox.

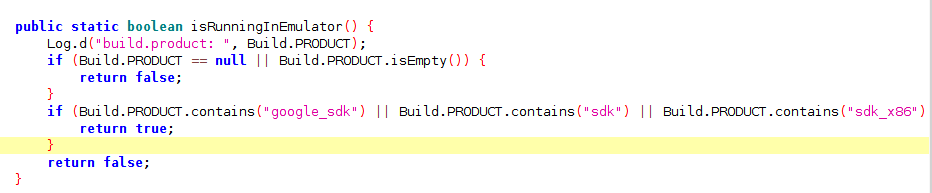
**I was not able to test this standard.**

**8.4** MSTG-RESILIENCE-4 The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device.

**The app did nothing when Drozer was running on the Android. The application failed the standard.**

**8.5** MSTG-RESILIENCE-5 The app detects, and responds to, being run in an emulator.

**The first string I was looking for was "BUILD" to check if the application had any code to detect running in an emulator. I found the following:**



**The application passed the standard.**

**8.6** MSTG-RESILIENCE-6 The app detects, and responds to, tampering the code and data in its own memory space.

**I was not able to test this standard.**

**8.8** MSTG-RESILIENCE-8 The detection mechanisms trigger responses of different types, including delayed and stealthy responses.

**I was not able to test this standard.**

**8.9** MSTG-RESILIENCE-9 Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis.

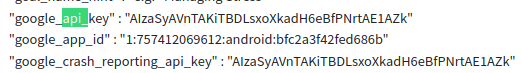
**8.12** MSTG-RESILIENCE-12

If the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are preferred over obfuscation whenever possible.

**Jadx showed the application code was not obfuscated.**

****

**Some keys were obfuscated:**

****

****

**The application failed this standard.**

**8.10** MSTG-RESILIENCE-10 The app implements a 'device binding' functionality using a device fingerprint derived from multiple properties unique to the device.

**I was not able to test this standard.**

**8.11** MSTG-RESILIENCE-11 All executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data.

**A Google API key was found through MobSF, the application failed the standard:**

****

**8.13** MSTG-RESILIENCE-13 As a defense in depth, next to having solid hardening of the communicating parties, application level payload encryption can be applied to further impede eavesdropping.

**I did not see application level payload encrypted, the application failed this standard:**

