المدرسة الوطنية للعلوم التطبيقية
École Nationale des Sciences Appliquées d'Oujda
ⵜⵉⵎⵙⵙⵓⵜⵜⴰⵢⵜ ⵏ ⵜⵓⵜⵜⵔⵉⵙⵜ ⵏ ⵙⵍⴰⵄ

ECOLE NATIONAL DES SCIENCES APPLIQUEES OUJDA

SECURITE MOBILE

ANALYSE DE GESTIONNAIRES DE MOTS DE PASSE

Élève :
ESSOGNIM GILLES KAROUGBE

Enseignant :
MOHAMED AMINE KOULALI

10 août 2023

Table des matières

1 Motivation

Avec la multiplication des plates-formes à authentification Login-Mot de passe sur internet,la gestion de nos comptes se révèle être un défis de sécurité majeur.La politique un compte-un mot de passe bien qu'incontournable pour protéger les comptes rend difficile la gestion des comptes pour les utilisateurs.L'utilisateur possède plusieurs comptes et doit donc mémoriser des Mots de passe complexe pour chacun d'eux.
Face à cette difficulté,on assiste ces dernières années à un véritable boom des services de gestions de mots de passe.

Ces gestionnaires sont des applications qui permettent de garder en "toute sécurité" des identifiants.ils constituent un véritable coffre-fort pour sauvegarder des informations sensible(Carte de crédit-Compte bancaire-Mot de passe...etc) afin d'une part de protéger ces informations et d'autre part gagner du temps en pré-remplissant les interfaces de connexion aux différentes plat-formes ou réseaux sociaux.

De nombreux gestionnaires de mots de passe sont mis à la disposition des utilisateurs sur les différents stores.Cependant la question légitime que l'on doit se poser est :Comment choisir mon gestionnaire de mot de passe et être sûr qu'il gère bien mes mots de passe ? Tout au long de cet projets nous allons découvrir puis analyser un certains nombre de gestionnaires de mots de passe .Nous allons pour chaque application :

1. Vérifier si elle gère bien le Mot de passe master et les clées dérivées(KDF,salt...)
2. Analyser le processus de déchiffrement de la base de donnée
3. Résumer les conclusions et évaluer la fiabilité de dernière par rapport à son image marketing.

Tout cela nous permettra de se prononcer sur la fiabilité des différentes applications par rapport à la gestion des mots de passe mais aussi de découvrir si elles ont d'autres activitées malicieuses en background.

2 A la découverte de certains gestionnaires de mots de passe

1. **Keeper**

   Keeper est un gestionnaire qui permet d'enregistrer et de sécuriser ses mots de passe dans des coffres-forts chiffrés. Il est destiné aux particuliers ou aux entreprises qui souhaitent renforcer leur cybersécurité, pour prévenir les menaces et violations de données liées à des mots de passe ou à des fichiers sensibles.

   La plateforme propose plusieurs fonctionnalités :

   (a) **la gestion des mots de passe :** Keeper permet d'enregistrer tous les mots de passe, et propose de les remplir automatiquement (en un seul clic) à chaque connexion. Vous pouvez stocker un nombre illimité de mots de passe, mais également en générer lorsque vous serez amenés à créer un compte sécurisé par un mot de passe. Vous êtes libres de choisir le niveau de complexité souhaité pour le mot de passe généré.

   (b) **le stockage et la protection d'autres types de données :** La plateforme permet de stocker diverses informations personnelles telles que des **numéros de téléphone**,des numéros de**carte de crédit**,des **notes sécurisées**,des**mots de passe d'application** ou encore des **adresses web**, de manière totalement sécurisée. Ces informations peuvent être utilisées pour remplir plus rapidement un formulaire, par exemple. L'outil gère également les systèmes d'authentification à double facteur automatiquement.

   (c) le partage en toute sécurité : l'outil permet de partager les mots de passe ou les fichiers stockés avec des personnes de confiance.

   Keeper est accessible via une interface web, des applications de bureau (Windows, MacOS, Linux), en extension sur tous les principaux navigateurs web (Google Chrome, Mozilla Firefox, Microsoft Edge, etc.) ou encore disponible **via application mobile, sur Android et iOS**.

2. **LastPass**

   LastPass permet de stocker les mots de passe et identifiants que les internautes saisissent au quotidien pour se connecter à une session d'utilisateur sur un site internet. Afin de limiter les risques de piratage qu'entraînent des mots de passe identiques ou trop faibles, LastPass propose de centraliser des mots de passe complexes sur un seul et même support.

   Pour créer un compte sur LastPass, il est nécessaire de créer un mot de passe maître qui donnera accès à l'interface de LastPass et donc à l'ensemble des mots de passe que vous aurez stocké sur l'outil. Le gestionnaire de mot de passe LastPass respecte les recommandations(MDM) visant à préserver sa sécurité numérique en exigeant un mot de passe comprenant au **minimum 12 caractères**.

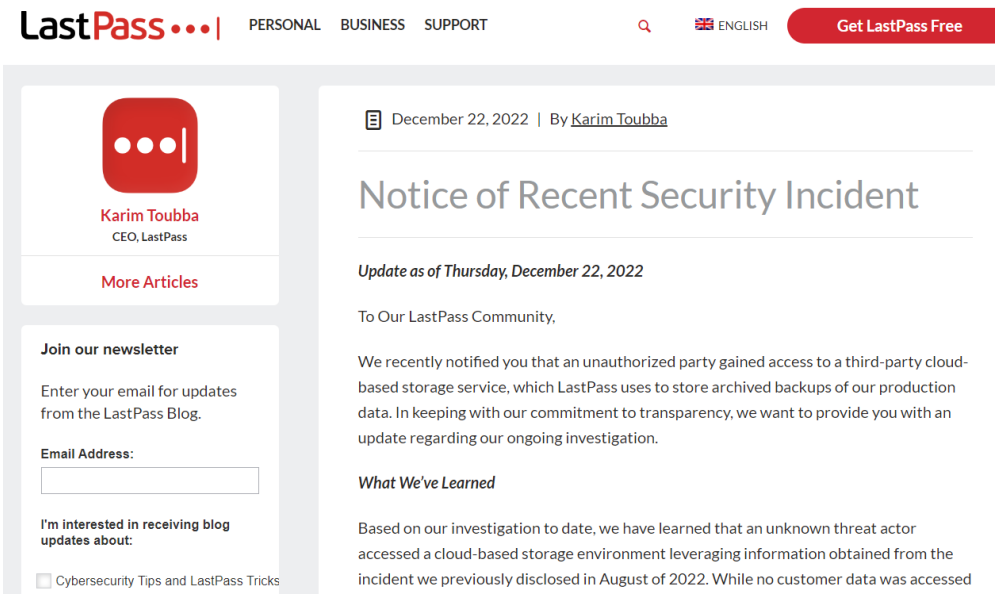


FIGURE 1 – LastPass Data breach

3. **NordPass**

   NordPass est un outil qui enregistre et stocke les mots de passe de manière ultra-sécurisée. Il donne la possibilité à ses utilisateurs d'organiser leurs mots de passe et de sécuriser leurs notes dans un coffre-fort crypté de bout en bout.

   Il permet la gestion des mots de passe de l'enregistrement à la sauvegarde au fil de la navigation ou par l'intégration d'un fichier CSV contenant des mots de passe exportés à partir d'un autre gestionnaire de mot de passe. Il est possible de les organiser dans des dossiers et de synchroniser différents appareils pour accéder à vos mots de passe où que vous soyez. Sa capacité de sauvegarde n'a pas de limitation.

   NordPass permet d'héberger d'autres éléments tels que la carte de crédit, des notes ou encore vos informations personnelles (nom, email, adresse, numéro de téléphone) afin de pouvoir remplir automatiquement les formulaires en ligne.

   Il propose de nombreuses fonctionnalités pour sécuriser ces données :

   (a) un générateur de mots de passe complexes

   (b) un scanner de violation de données pour analyser les données divulguées et les mots de passe vulnérables

   (c) un rapport sur l'état de santé des mots de passe pour identifier les mots de passe faibles, anciens et réutilisés

   (d) un accès biométrique à l'application mobile

   (e) une option d'authentification à double facteur

   (f) un cryptage de données avec l'algorithme XChaCha20

   (g) Partage sécurisé des mots de passe

   (h) Possibilité de donner un accès d'urgence à un membre de votre famille en cas de circonstances exceptionnelles

4. **Passky**

   Passky permet de gérer ses mots de passe en ligne, de façon simple, rapide et sécurisée. Les utilisateurs peuvent utiliser gratuitement **l'authentification à deux facteurs** (2FA) à la fois software et hardware.

   Sur la page principale de Passky, l'on peut modifier et copier les mots de passe enregistrés en un simple clic.On peut aussi effectuer une recherche rapide pour trouver rapidement les mots de passe souhaités. À savoir : le look de l'interface peut être personnalisé, vous pouvez choisir un thème prédéfini ou bien créer le vôtre from scratch.

   Autre point à souligner : vous pouvez importer vos mots de passe de tous les autres principaux gestionnaires de mots de passe (Dashlane, Bitwarden, NordPass,etc.). À l'inverse, il est aussi possible de passer de Passky à n'importe quel autre gestionnaire de mots de passe sans aucun problème via un export dédié.

   Ce gestionnaire de mots de passe open-source se veut plus léger que les autres présents sur le marché : Passky Server est écrit avec moins de 1 000 lignes de code, tandis que les autres gestionnaires de mots de passe sont généralement écrits avec plus de 400 000 lignes de code.

   Concernant les langages, Passky prend en charge 17 langues dont le français.

   Passky possède une extension Chrome et une application mobile (Android seulement).

   Passky est gratuit, vous devrez vous créer un compte pour l'utiliser.

5. 1password
6. Empass
7. Dashlane
8. Keepass2Android

3 Méthodologie et outils d'analyse

3.1 Lab

3.2 Analyse Statique

— **Jadx-gui**

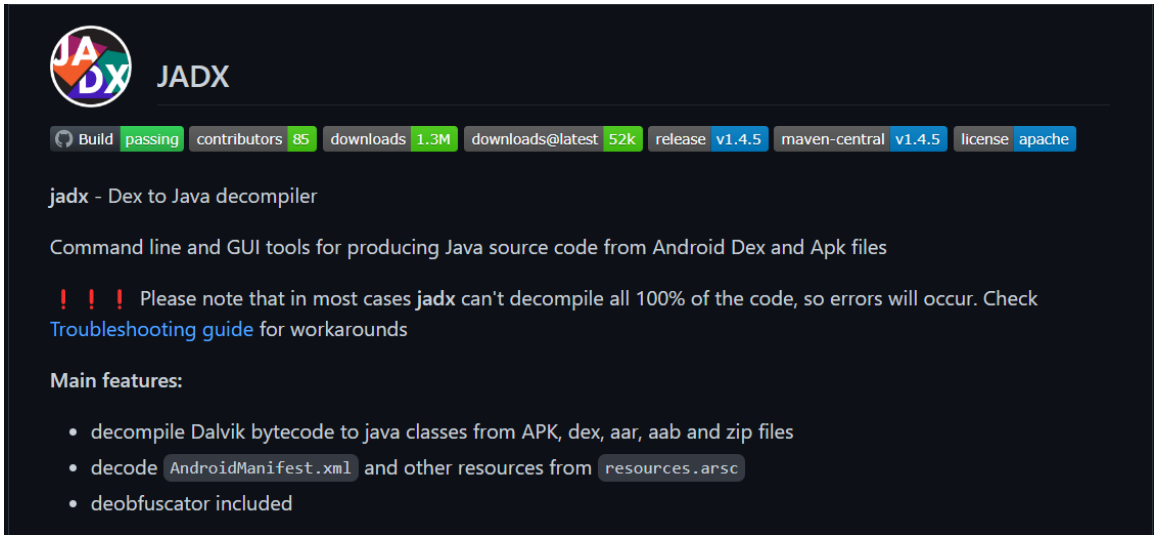


FIGURE 2 – dexToJavaDecompiler

— **Mobsf**

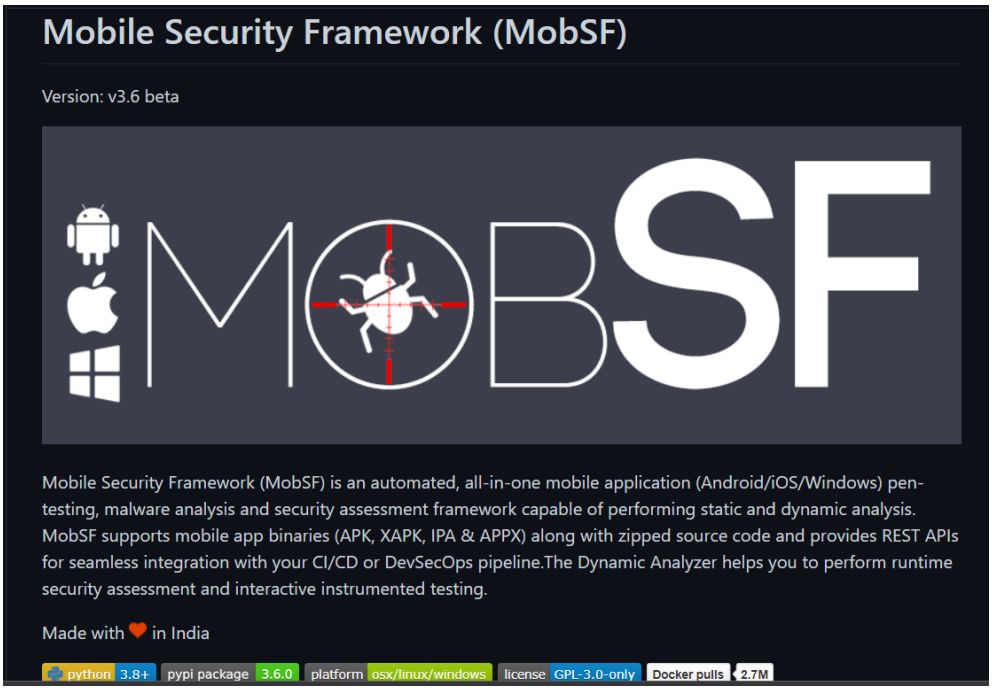


FIGURE 3 – MobsF

— **Quark**

FIGURE 4 – QuarkEngine

— **Ghidra**

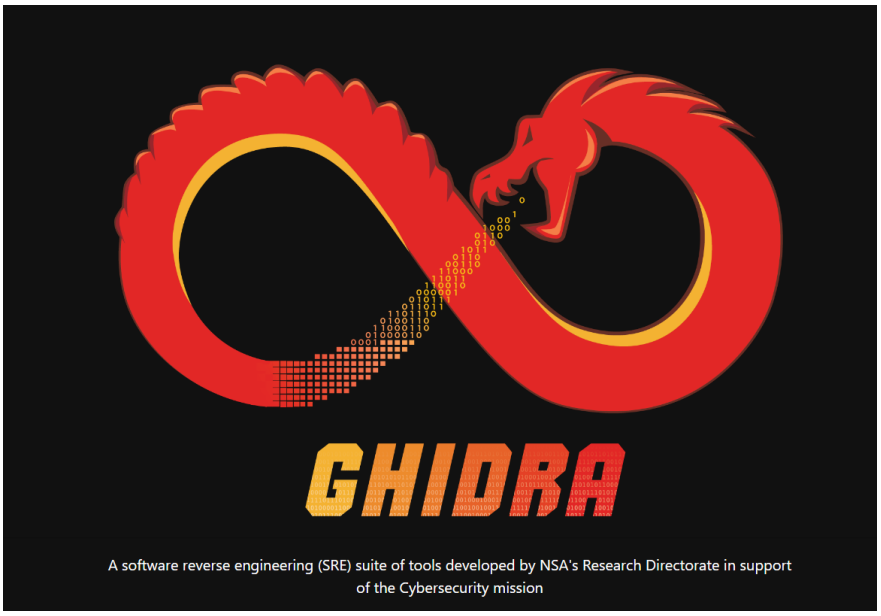


FIGURE 5 – ghidra

— **Apktool**

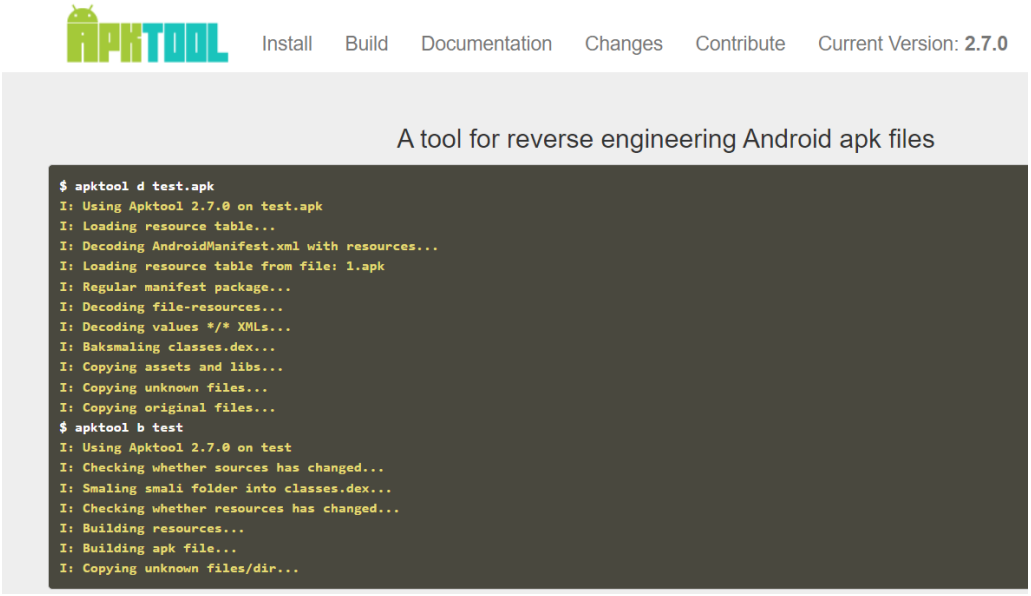


FIGURE 6 – Apktool

3.3 Analyse Dynamique

— **Burpsuite proxy**

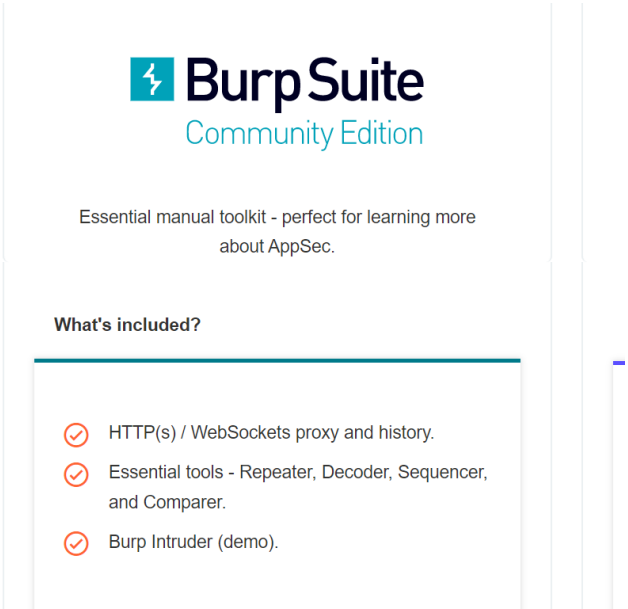


FIGURE 7 – Burpsuite

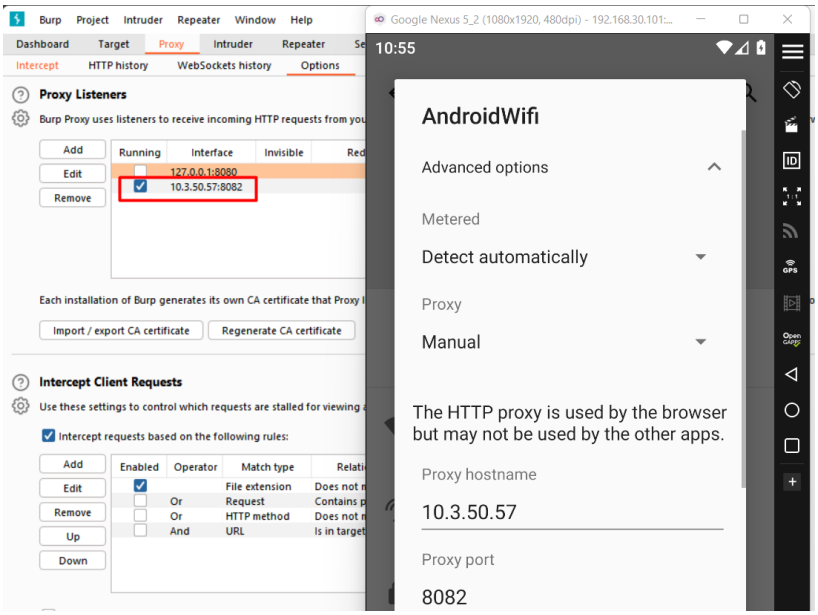


FIGURE 8 – ProxySetUp

On installe ensuite le certificat de Portswigger afin de pouvoir proxifier les requettes HTTPS

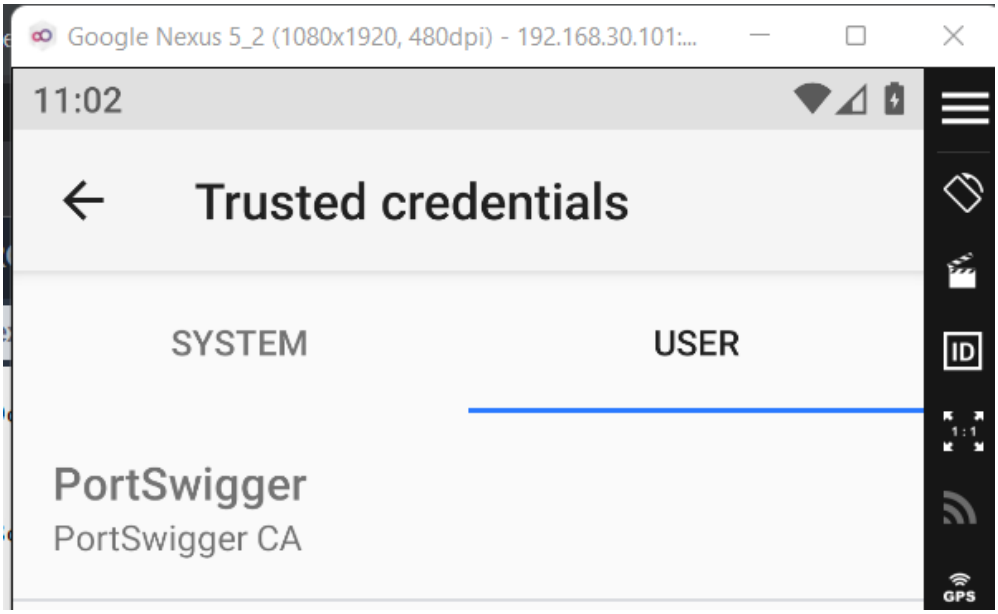


FIGURE 9 – ProxySetUp

— **Frida**

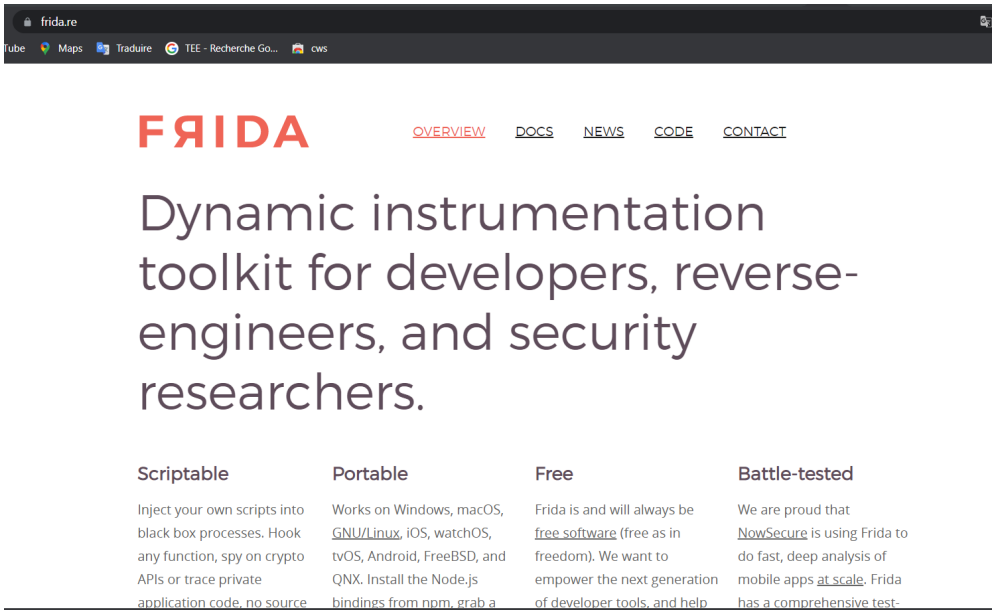


FIGURE 10 – frida.re

— **Objection**

FIGURE 11 – objection

SSL pinning Bypass

On va utiliser fridra pour injecter du javaScript afin de forcer les applications dans lesquelles est implémenté le SSLpinning à communiquer avec notre proxyWeb burpSuite.

1. Frida server
   On push le serveur frida sur l'émulateur android puis on l'exécute.

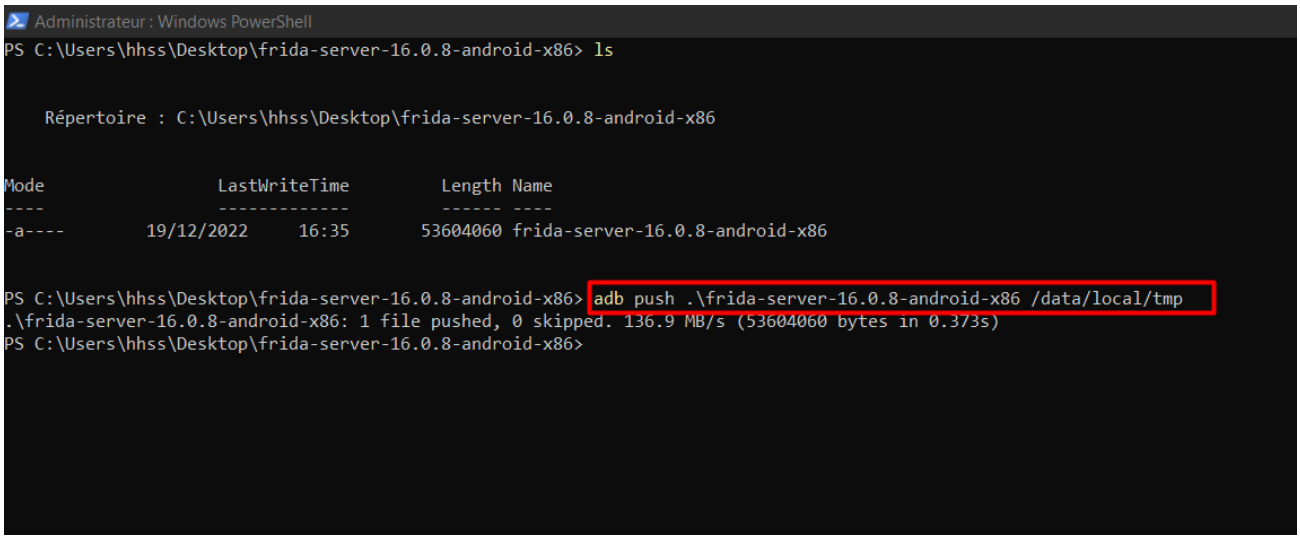


FIGURE 12 – FridaServeur

2. Bypass.js

FIGURE 13 – Bypass.js

3. Frida command maintenant on exécute la commande pour injecter le code javaScript.
   frida -U -f targetApp -l bypasScript

-U, –usb connect to USB device

-f TARGET, –file TARGET

-l SCRIPT, –load SCRIPT

4 Keeper

4.1 Analyse Statique

```
<uses-sdk android:minSdkVersion="23" android:targetSdkVersion="31"/>
<supports-screens android:anyDensity="true" android:smallScreens="true" android:normalScreens="true" andrc
<uses-feature android:name="android.hardware.wifi" android:required="false"/>
<uses-feature android:name="android.hardware.telephony" android:required="false"/>
<uses-feature android:name="android.hardware.screen.portrait" android:required="false"/>
<uses-feature android:name="android.hardware.touchscreen"/>
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.USE_BIOMETRIC"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="com.android.vending.BILLING"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.USE_FINGERPRINT"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-feature android:name="android.hardware.camera" android:required="false"/>
<uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
<uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE"/>
```

FIGURE 14 – keeper.Permissions

```
1  public final class MasterKeys {
2      private static final String ANDROID_KEYSTORE = "AndroidKeyStore";
3      static final String KEYSTORE_PATH_URI = "android-keystore://";
4      private static final int KEY_SIZE = 256;
5      static final String MASTER_KEY_ALIAS = "
           _androidx_security_master_key_";
```

Il s'agit d'une classe Java appelée MasterKeys qui contient un ensemble de constantes et une méthode statique. La classe semble utilisée pour générer et gérer des clés cryptographiques dans le système d'exploitation Android.La classe définit une variable privée statique finale ANDROID_KEYSTORE qui contient la chaîne "AndroidKeyStore", qui est probablement le nom de la clé de stockage où les clés sont stockées.

La classe définit également une variable statique finale KEYSTORE_PATH_URI qui contient la chaîne "android-keystore ://", qui est probablement le chemin ou l'URI de la clé de stockage.

La classe définit également une variable privée statique finale KEY_SIZE qui contient la valeur 256, qui est probablement la taille (en bits) des clés qui sont générées.

La classe définit également une variable statique finale MASTER_KEY _ALIAS qui contient la chaîne "androidx_security_master_key", qui est probablement le nom ou l'alias de la clé maîtresse qui est générée.

```
6      private static KeyGenParameterSpec
           createAES256GCMKeyGenParameterSpec(String str) {
```

```
7           return new KeyGenParameterSpec.Builder(str,3).setBlockModes(
                CodePackage.GCM).setEncryptionPaddings("NoPadding").
                setKeySize(256).build();
8       }
```

Ce code crée une instance de la classe KeyGenParameterSpec en utilisant le patron de conception Builder. La KeyGenParameterSpec est utilisée pour la génération de clés dans le système Android Keystore. Les paramètres spécifiques qui sont définis dans cet exemple sont :

l'alias de clé (un identificateur de chaîne pour la clé) défini sur la valeur du paramètre d'entrée "str" l'objectif de la clé défini sur "ENCRYPT_DECRYPT" (3) :

1. le mode de bloc défini sur "GCM"
2. le remplissage de chiffrement défini sur "NoPadding"
3. la taille de la clé définie sur 256 bits

Cette KeyGenParameterSpec est utilisée pour créer une clé AES-256 en mode GCM sans remplissage.

```
9   public static final KeyGenParameterSpec AES256_GCM_SPEC =
        createAES256GCMKeyGenParameterSpec(MASTER_KEY_ALIAS);
```

Cette ligne de code définit une constante nommée "AES256_GCM_SPEC" qui contient la valeur retournée par la méthode "createAES256GCMKeyGenParameterSpec" avec un paramètre "MASTER_KEY_ALIAS". Il crée une spécification de clé AES256 GCM avec l'alias de clé défini sur la valeur de la constante "MASTER_KEY_ALIAS". La constante AES256_GCM_SPEC peut être utilisée par la suite en tant que paramètre pour la génération de clés dans le système Android Keystore.

```
10  private static void generateKey(KeyGenParameterSpec
        keyGenParameterSpec) throws GeneralSecurityException {
11          KeyGenerator keyGenerator = KeyGenerator.getInstance("AES",
                ANDROID_KEYSTORE);
12          keyGenerator.init(keyGenParameterSpec);
13          keyGenerator.generateKey();
14      }
```

Ce code définit une méthode nommée "generateKey" qui prend une KeyGenParameterSpec en tant que paramètre d'entrée. À l'intérieur de la méthode :

Il instancie un objet KeyGenerator en appelant la méthode getInstance() de la classe KeyGenerator, en passant l'algorithme "AES" et "ANDROID_KEYSTORE" en arguments. Le ANDROID_KEYSTOREE est un fournisseur pour le système Android Keystore.

Il initialise l'objet KeyGenerator à l'aide de la keyGenParameterSpec d'entrée en appelant la méthode init() sur l'objet KeyGenerator.

Il génère une nouvelle clé en appelant la méthode generateKey() sur l'objet KeyGenerator.

La clé générée sera stockée dans le système Android Keystore et peut être accédée par son alias qui est passé dans l'objet keyGenParameterSpec.Bref ce code génère une clé AES.

```
15 static void validate(KeyGenParameterSpec keyGenParameterSpec) {
16         if (keyGenParameterSpec.getKeySize() != 256) {
17             throw new IllegalArgumentException("invalid key size,
                   want 256 bits got " + keyGenParameterSpec.getKeySize()
                    + " bits");
18         } else if (!Arrays.equals(keyGenParameterSpec.getBlockModes()
               , new String[]{CodePackage.GCM})) {
19             throw new IllegalArgumentException("invalid block mode,
                   want GCM got " + Arrays.toString(keyGenParameterSpec.
                   getBlockModes()));
20         } else if (keyGenParameterSpec.getPurposes() != 3) {
21             throw new IllegalArgumentException("invalid purposes mode
                   , want PURPOSE_ENCRYPT | PURPOSE_DECRYPT got " +
                   keyGenParameterSpec.getPurposes());
22         } else if (!Arrays.equals(keyGenParameterSpec.
               getEncryptionPaddings(), new String[]{"NoPadding"})) {
23             throw new IllegalArgumentException("invalid padding mode,
                    want NoPadding got " + Arrays.toString(
                   keyGenParameterSpec.getEncryptionPaddings()));
24         } else if (keyGenParameterSpec.isUserAuthenticationRequired()
                && keyGenParameterSpec.
               getUserAuthenticationValidityDurationSeconds() < 1) {
25             throw new IllegalArgumentException("per-operation
                   authentication is not supported (
                   UserAuthenticationValidityDurationSeconds must be >0)"
                   );
26         }
27     }
```

Ce code définit une méthode nommée "validate" qui prend une KeyGenParameterSpec en tant que paramètre d'entrée. A l'intérieur de la méthode, il vérifie les propriétés spécifiées de la KeyGenParameterSpec contre un ensemble de valeurs attendues, et s'il y en a qui ne correspondent pas, il lève une exception IllegalArgumentException avec un message décrivant le non-accord.

Les propriétés spécifiques qui sont vérifiées sont :

1. la taille de clé est de 256 bits,
2. le mode de bloc est GCM,
3. le but de la clé est ENCRYPT_DECRYPT,
4. le remplissage de chiffrement est NoPadding
5. l'authentification de l'utilisateur n'est pas requise
6. la durée de validité de l'authentification de l'utilisateur est supérieure à 1.

La méthode valide les paramètres de la clé générée donnée contre les propriétés attendues. S'il y a un quelconque propriété qui n'est pas respectée, il lance une exception.

```
28  private static boolean keyExists(String str) throws
        GeneralSecurityException, IOException {
29          KeyStore keyStore = KeyStore.getInstance(ANDROID_KEYSTORE);
```

```
30          keyStore.load(null);
31          return keyStore.containsAlias(str);
32      }
```

Ce code définit une méthode nommée "keyExists" qui prend une chaîne en tant que paramètre d'entrée et retourne une valeur booléenne. À l'intérieur de la méthode :

1. Il instancie un objet KeyStore en appelant la méthode getInstance() de la classe KeyStore, en passant "ANDROID_KEYSTORE" en argument. Le ANDROID_KEYSTORE est un fournisseur pour le système Android Keystore.
2. Il charge la Keystore en passant null à la méthode load() sur l'objet KeyStore.
3. Il retourne le résultat de l'appel de la méthode containsAlias() sur l'objet KeyStore avec la chaîne d'entrée en argument. Cette méthode vérifie si la Keystore contient une entrée sous l'alias spécifié.

Cette méthode vérifie si la clé avec l'alias donné existe déjà dans la Keystore Android ou non. Si la Keystore contient une entrée sous l'alias spécifié, elle retournera true, sinon elle retournera false.

```
33
34      public static String getOrCreate(KeyGenParameterSpec
            keyGenParameterSpec) throws GeneralSecurityException,
            IOException {
35          validate(keyGenParameterSpec);
36          if (!keyExists(keyGenParameterSpec.getKeystoreAlias())) {
37              generateKey(keyGenParameterSpec);
38          }
39          return keyGenParameterSpec.getKeystoreAlias();
40      }
```

Ce code définit une méthode nommée "getOrCreate" qui prend une KeyGenParameterSpec en tant que paramètre d'entrée et retourne une valeur de chaîne. A l'intérieur de la méthode :

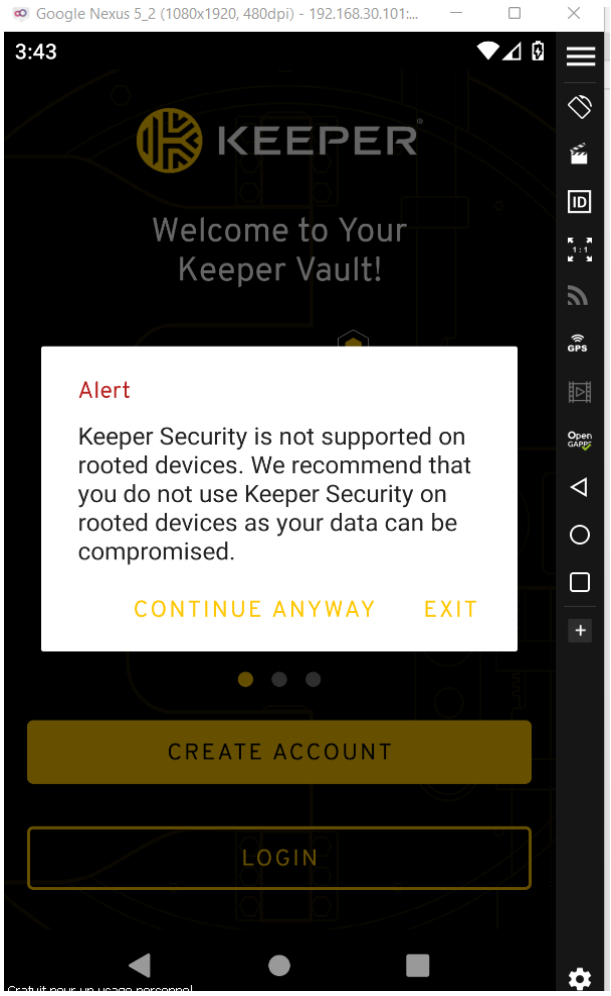
1. Il valide la KeyGenParameterSpec d'entrée en appelant la méthode validate() sur celle-ci.
2. Il vérifie si la clé existe déjà dans le système Android Keystore en appelant la méthode keyExists() avec l'alias de la clé de la KeyGenParameterSpec d'entrée en tant qu'argument.
3. Si la clé n'existe pas dans le système Android Keystore, il génère la clé en appelant la méthode generateKey() sur la KeyGenParameterSpec d'entrée.
4. Il retourne l'alias de clé de la KeyGenParameterSpec d'entrée.

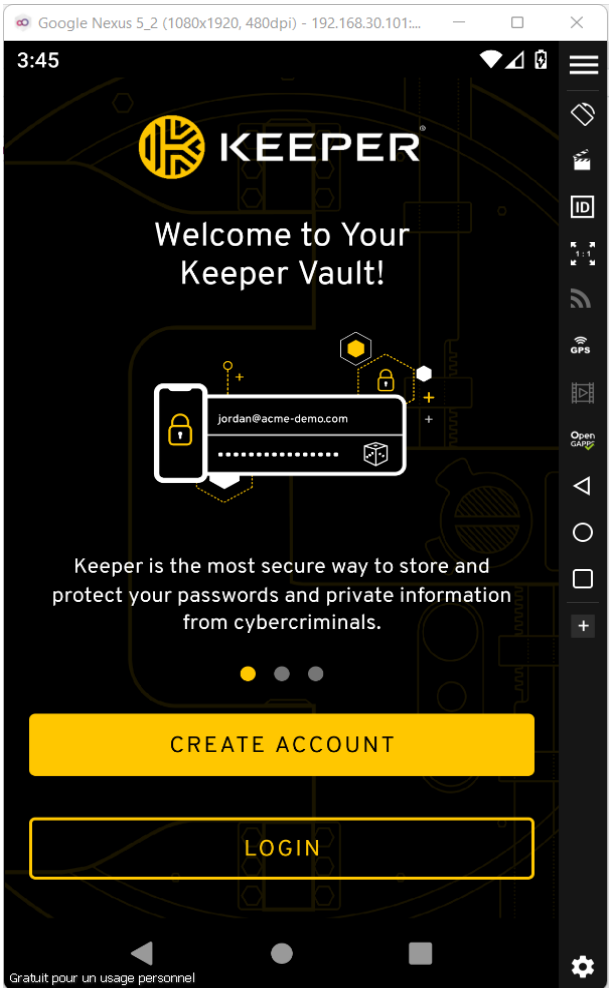
Cette méthode valide d'abord les paramètres de la clé générée. Si la clé avec l'alias donné n'existe pas dans le système de clés, elle va générer la clé en utilisant les paramètres de la clé générée, sinon elle va retourner

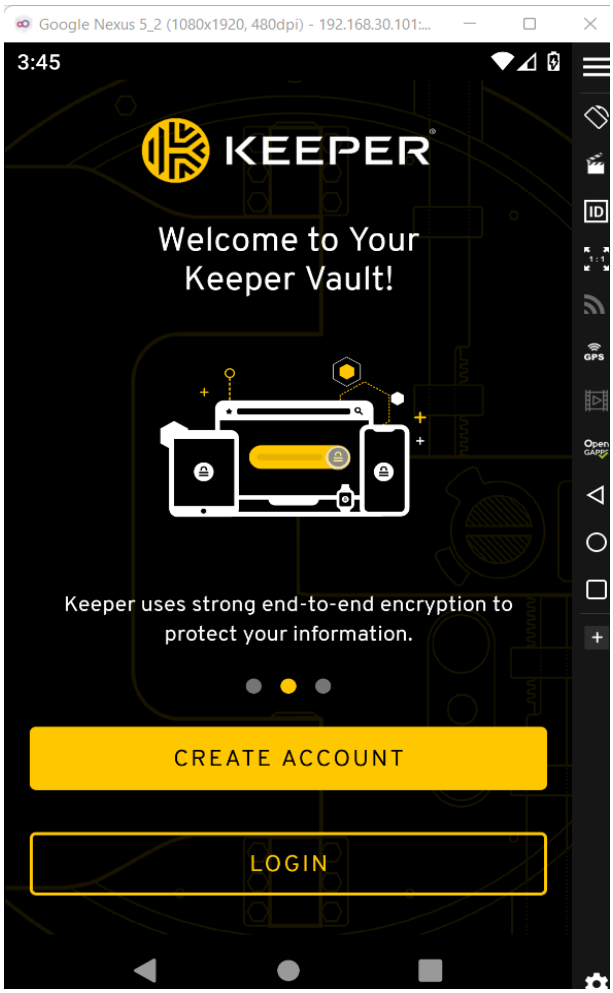
```java
package com.callpod.android_apps.keeper.encryption;

/* loaded from: classes3.dex */
public interface Pbkdf2 {
    byte[] generateKey(String str, byte[] bArr, int i, int i2);
}
```

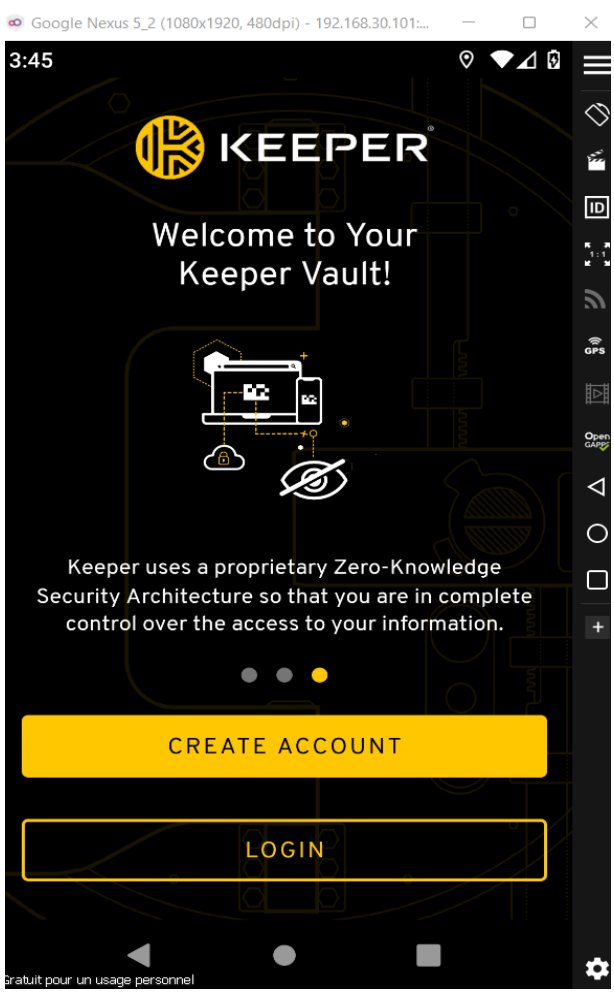
FIGURE 15 – pbkdf2

4.2 Analyse Dynamique










```
(MobSec) PS C:\Users\hhss\Documents\MobSec> frida -U -f com.callpod.android_apps.keeper -l C:\Users\hhss\Desktop\bypas
     ____
    / _  |   Frida 16.0.8 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Nexus 5 (id=192.168.30.101:5555)
Spawned `com.callpod.android_apps.keeper`. Resuming main thread!
[Nexus 5::com.callpod.android_apps.keeper ]-> Bypassing SSL Pinning
Bypassing SSL Pinning
[Nexus 5::com.callpod.android_apps.keeper ]->
```

FIGURE 16 – SSL pinning Bypass

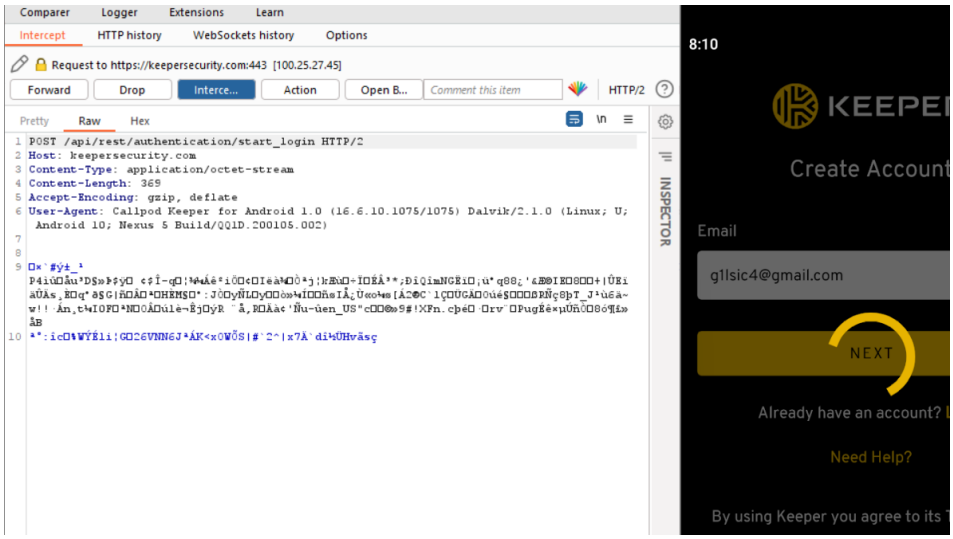


FIGURE 17 – SSL pinning Bypass

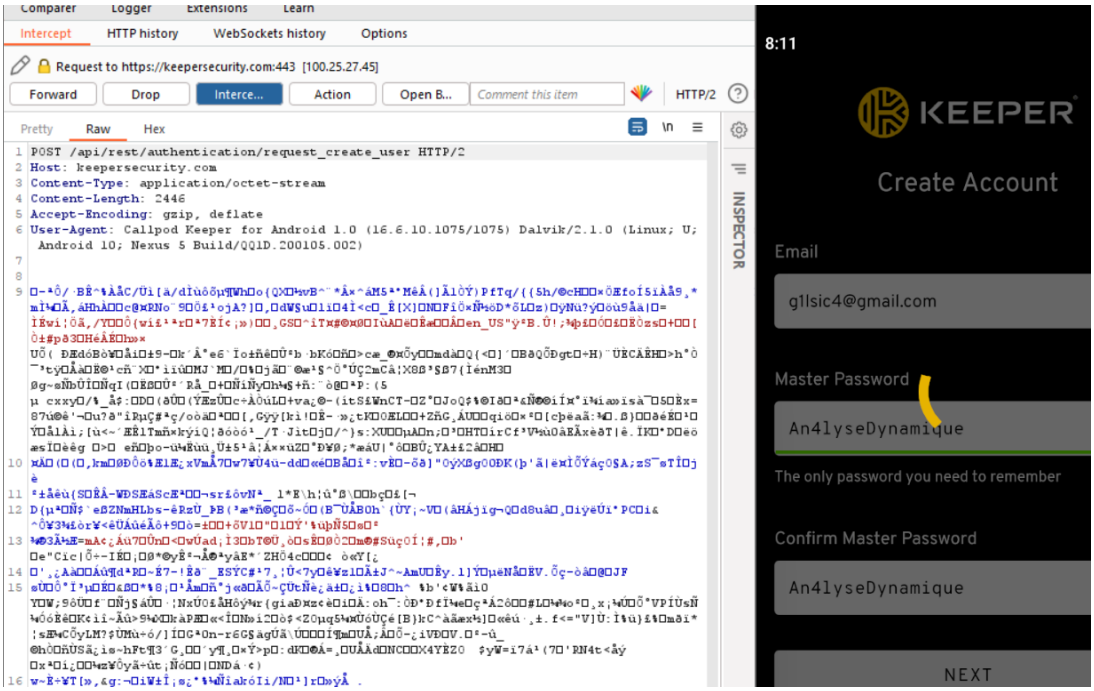


FIGURE 18 – SSL pinning Bypass

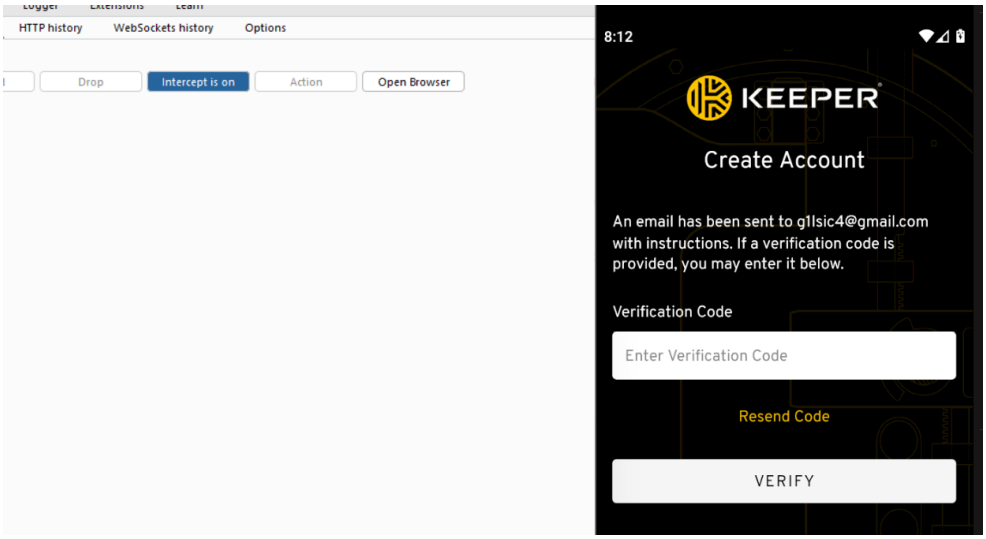


FIGURE 19 – SSL pinning Bypass

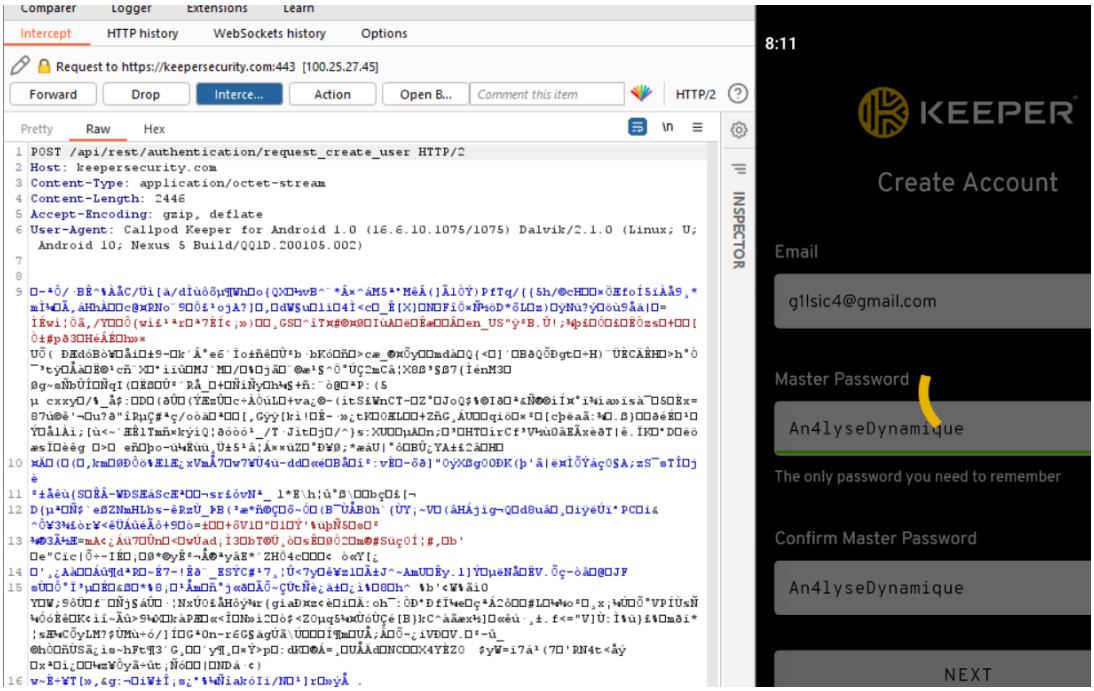


FIGURE 20 – SSL pinning Bypass

4.3 Conclusion

5 LastPass

5.1 Analyse statique-1

Les permissions utiliser par l'application sont :

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="com.android.browser.permission.WRITE_HISTORY_BOOKMARKS"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.NFC"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.BIND_ACCESSIBILITY_SERVICE"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="com.android.vending.BILLING"/>
<uses-permission android:name="org.onepf.openiab.permission.BILLING"/>
<uses-permission android:name="com.sec.android.iap.permission.BILLING"/>
<uses-permission android:name="android.permission.USE_BIOMETRIC"/>
<uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE"/>
<uses-permission android:name="android.permission.MANAGE_ACCOUNTS"/>
<uses-permission android:name="android.permission.USE_CREDENTIALS"/>
<permission android:name="com.lastpass.lpandroid.permission.AUTOFILL_AUTH" android:protectionLevel="signature"/>
<uses-permission android:name="com.lastpass.lpandroid.permission.AUTOFILL_AUTH"/>
<uses-permission android:name="com.lastpass.authenticator.permission.CLOUD_SYNC_LPA"/>
<permission android:name="com.lastpass.lpandroid.permission.CLOUD_SYNC_LPM" android:protectionLevel="signature"/>
<uses-permission android:name="com.lastpass.lpandroid.permission.CLOUD_SYNC_LPM"/>
<uses-permission android:name="com.lastpass.lpandroid.permission.C2D_MESSAGE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="nu.tommie.inbrowser.PERMISSION_READ_URL"/>
<uses-permission android:name="com.samsung.android.providers.context.permission.WRITE_USE_APP_FEATURE_SURVEY"/>
<permission android:name="com.lastpass.lpandroid.permission.QUICK_SETTINGS_TILE" android:protectionLevel="signature"/>
<uses-permission android:name="com.lastpass.lpandroid.permission.QUICK_SETTINGS_TILE"/>
<permission android:name="com.lastpass.lpandroid.permission.ADFS_LOGIN" android:protectionLevel="signature"/>
<uses-permission android:name="com.lastpass.lpandroid.permission.ADFS_LOGIN"/>
```

FIGURE 21 – LastPassPermissions

```
41 public class Pbkdf2JniWrapper {
42     private boolean f11524a;
43
44     public Pbkdf2JniWrapper() {
45         try {
46             System.loadLibrary("lastpass_pbkdf2");
47             this.f11524a = true;
48         } catch (UnsatisfiedLinkError e10) {
49             x0.x(e10);
50             e10.printStackTrace(); }
51     }
52     private final native byte[] pbkdf2HmacSha256Impl(byte[] bArr,
           byte[] bArr2, int i10, int i11);
53
54     public final boolean a() {
55         return this.f11524a;
56     }
57     public final synchronized byte[] b(byte[] bArr, byte[] bArr2, int
           i10, int i11) {
58         p.g(bArr, "password");
59         p.g(bArr2, "salt");
```

```
60          return pbkdf2HmacSha256Impl(bArr, bArr2, i10, i11);
61      }
62  }
```

Ceci est une classe Java nommée Pbkdf2JniWrapper, qui est une enveloppe pour la fonction de dérivation de clé PBKDF2 implémentée dans une bibliothèque JNI (Java Native Interface) appelée "lastpass_pbkdf2". La classe possède un constructeur qui essaie de charger la bibliothèque JNI et un champ booléen f11524a qui est défini sur **true si la bibliothèque est chargée avec succès**, et **false sinon**.
La classe possède une méthode unique **b(byte[] bArr, byte[] bArr2, int i10, int i11)**, qui est une méthode synchronisée et prend en entrée 4 paramètres, un mot de passe et un sel sous forme de bytes, ainsi que deux valeurs int, qui est implémentée nativement dans la bibliothèque JNI chargée et retourne la clé dérivée sous forme de tableau de bytes. Il a également une méthode d'aide a() qui retourne le booléen indiquant si la bibliothèque JNI a été chargée correctement.

```
63  public final void AFKeystoreWrapper(String str) {
64          AFLogger.values("Creating a new key with alias: ".concat(
               String.valueOf(str)));
65          try {
66              Calendar calendar = Calendar.getInstance();
67              Calendar calendar2 = Calendar.getInstance();
68              calendar2.add(1, 5);
69              AlgorithmParameterSpec algorithmParameterSpec = null;
70              synchronized (this.AFInAppEventParameterName) {
71                  if (!this.valueOf.containsAlias(str)) {
72                      int i10 = Build.VERSION.SDK_INT;
73                      if (i10 >= 23) {
74                          algorithmParameterSpec = new
                               KeyGenParameterSpec.Builder(str, 3).
                               setCertificateSubject(new X500Principal("
                               CN=AndroidSDK, O=AppsFlyer")).
                               setCertificateSerialNumber(BigInteger.ONE)
                               .setCertificateNotBefore(calendar.getTime
                               ()).setCertificateNotAfter(calendar2.
                               getTime()).build();
75                      } else if (i10 >= 18 && !z.valueOf()) {
76                          algorithmParameterSpec = new
                               KeyPairGeneratorSpec.Builder(this.
                               AFKeystoreWrapper).setAlias(str).
                               setSubject(new X500Principal("CN=
                               AndroidSDK, O=AppsFlyer")).setSerialNumber
                               (BigInteger.ONE).setStartDate(calendar.
                               getTime()).setEndDate(calendar2.getTime())
                               .build();
77                      }
78                      KeyPairGenerator keyPairGenerator =
```

```
                        KeyPairGenerator.getInstance("RSA", "
                            AndroidKeyStore");
79                      keyPairGenerator.initialize(
                            algorithmParameterSpec);
80                      keyPairGenerator.generateKeyPair();
81                  } else {
82                      AFLogger.values("Alias already exists: ".concat(
                            String.valueOf(str)));
83                  }
84              }
85          } catch (Throwable th2) {
86              StringBuilder sb2 = new StringBuilder("Exception ");
87              sb2.append(th2.getMessage());
88              sb2.append(" occurred");
89              AFLogger.valueOf(sb2.toString(), th2);
90          }
91      }
```

Ceci est une méthode nommée AFKeystoreWrapper qui prend en paramètre une chaîne unique str. Il crée une nouvelle clé avec str comme alias dans le Keystore Android, un système de stockage sécurisé pour les clés et les certificats spécifiques à l'application.

La méthode commence par enregistrer le message "Création d'une nouvelle clé avec l'alias : str" en utilisant la classe AFLogger, puis elle tente de créer une nouvelle clé à l'intérieur d'un bloc try. Il initialise les instances de Calendar pour le début et la fin de la validité de la clé, puis définit une instance d'AlgorithmParameterSpec comme étant nulle. Il utilise la synchronisation sur l'objet AFInAppEventParameterName et vérifie si le Keystore n'a pas déjà une entrée avec l'alias donné, puis il vérifie la version d'android, si c'est supérieur à 23, il crée une instance de KeyGenParameterSpec, sinon s'il est supérieur à 18 et quelques autres conditions sont vérifiées, il crée une KeyPairGeneratorSpec. Ensuite il initialise un KeyPairGenerator avec RSA et AndroidKeyStore et génère la paire de clé avec les paramètres d'algorithme définis ci-dessus. Si l'alias existe déjà, il enregistre le message "L'alias existe déjà : str". Si une exception est survenue, il enregistre le message d'exception et l'exception.

```
92  abstract class e0 {
93      private static final Method f32506a = q1.a("java.security.
            KeyStore$PasswordProtection", "getProtectionAlgorithm", new
            Class[0]);
94
95      public static Key a(KeyStore keyStore, String str, KeyStore.
            ProtectionParameter protectionParameter) {
96          if (protectionParameter != null) {
97              if (protectionParameter instanceof KeyStore.
                    PasswordProtection) {
98                  KeyStore.PasswordProtection passwordProtection = (
                        KeyStore.PasswordProtection) protectionParameter;
99                  Method method = f32506a;
```

```
100                 if (method == null || q1.a(passwordProtection, method
                        ) == null) {
101                     return keyStore.getKey(str, passwordProtection.
                            getPassword());
102                 }
103                 throw new KeyStoreException("unsupported password
                        protection algorithm");
104             }
105             throw new UnsupportedOperationException();
106         }
107         throw new UnrecoverableKeyException("requested key requires a
                password");
108     }
109 }
```

Ceci est une **classe abstraite nommée e0** qui contient un champ statique privé f32506a qui contient une instance de la méthode. Ce champ est initialisé dans un bloc statique avec le résultat de l'appel de la méthode q1.a avec trois (03) paramètres :

1. le premier paramètre est une chaîne représentant la classe "java.security.KeyStore$PasswordProte
2. le deuxième paramètre est une chaîne "getProtectionAlgorithm"
3. le troisième paramètre est un tableau vide de classes

La classe a également une méthode statique publique **a(KeyStore keyStore, String str, KeyStore.ProtectionParameter protectionParameter)** qui prend en entrée trois paramètres, un KeyStore, une chaîne, et un KeyStore.ProtectionParameter. Il vérifie d'abord si le ProtectionParameter n'est pas null et si le ProtectionParameter est une instance de KeyStore.PasswordProtection, il crée alors un KeyStore.PasswordProtection et extrait l'algorithme de protection en vérifiant avec le champ privé f32506a, s'il n'est pas null ou si l'algorithme n'est pas null, il retourne la clé du keystore en utilisant la méthode getKey en passant la chaîne et le mot de passe du ProtectionParameter, sinon il lance une exception KeyStore. Si le ProtectionParameter n'est pas une instance de KeyStore.PasswordProtection, il lance une UnsupportedOperationException. Si le ProtectionParameter est null, il lance une UnrecoverableKeyException.

5.2 Analyse statique-2

— Dans ghidra nous allons essayer d'analyser la bibliothèque Native "liblastpass_pbkdf2.so" afin de vérifier l'implémentation de la Key Derivation Function
— on remarque que le linkage est dynamique et la methode native chargée est : Java_com_lastpass_lpandroid_domain_encryption_Pbkdf2JniWrapper_pbkdf2HmacSha256I

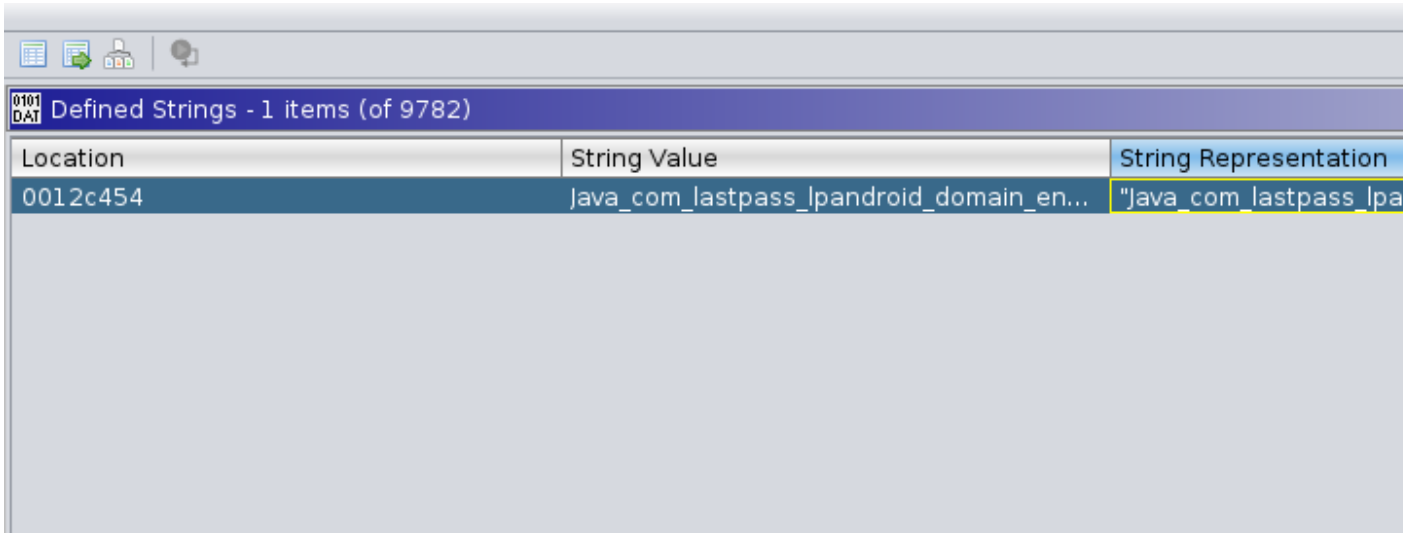


FIGURE 22 – definedStrings

FIGURE 23 – Java_com_lastpass_lpandroid_domain_encryption_Pbkdf2JniWrapper_pbkdf2HmacSha256Impl

Il s'agit d'une fonction C, qui semble être une implémentation de la fonction de dérivation de clé PBKDF2. La fonction prend plusieurs paramètres :

1. param_1 : un pointeur vers une sorte de structure de données
2. param_2 : un mot de passe ou une phrase secrète
3. param_3 : une valeur de sel
4. param_4 : le nombre d'itérations
5. param_5 : la longueur souhaitée de la clé dérivée
6. param_6 : un autre paramètre

La fonction utilise la bibliothèque OpenSSL pour effectuer le calcul PBKDF2, plus précisément elle utilise la fonction PKCS5_PBKDF2_HMAC pour effectuer le calcul. La fonction de hachage utilisée est EVP_sha256.

La fonction effectue ensuite des opérations de gestion de la mémoire en allouant et libérant de la mémoire à l'aide des fonctions appelées cRea_8, *(code **)(*param_1 + 0x5c0), *(code **)(*param_1 + 0x558), *(code **)(*param_1 + 0x600) et *(code **)(*param_1 + 0x680).

Il vérifie également un guardien de pile avec __stack_chk_fail(), c'est une partie d'un mécanisme pour détecter les erreurs de dépassement de pile en fournissant une valeur de gardien canari.

cette fonction est utilisée par notre App LastPass pour Android pour dériver des clés cryptographiques à partir de phrases secrètes fournies par l'utilisateur(MasterPassword). Le fait qu'elle utilise la bibliothèque OpenSSL est un inconvénient par rapport aux bibliothèques spécifiques aux plateformes car elle n'utilise pas les propriétés de sécurité de la plate-forme.

Il est important de noter que **PBKDF2** n'est plus considéré comme une fonction de dérivation de clé sûre depuis les environs de **2018** car elle n'est pas considérée comme **très coûteuse pour les puissances de calcul modernes**.Comme cette méthode est utilisée pour dériver des clés pour protéger des données sensibles, il est recommandé d'utiliser des alternatives plus sûres telles qu'**Argon2** ou **scrypt**.

5.3 Analyse Dynamique

1. SSL pinning Bypass

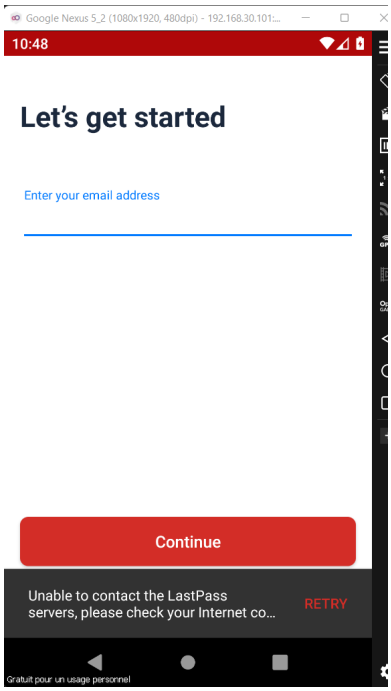


FIGURE 24 – SSLPinningEnabled

On remarque que le SSL pinning a été implémenté dans notre application et donc refuse automatiquement toutes communications avec tous serveurs autres que ceux prévu .ce qui empêche par défaut les attaques MITM.

```
   -  Files                com.android.documentsui
   -  Gallery              com.android.gallery3d
   -  Keeper               com.callpod.android_apps.keeper
   -  LastPass             com.lastpass.lpandroid
   -  Messaging            com.android.messaging
   -  NordPass             com.nordpass.android.app.password.manager
   -  Search               com.android.quicksearchbox
   -  Superuser            com.genymotion.superuser
   -  WebView Shell        org.chromium.webview_shell

(MobSec) C:\Users\hhss\Documents\MobSec>frida -U -f com.lastpass.lpandroid -l C:\Users\hhss\Desktop\bypass.js
     ____
    / _  |   Frida 16.0.8 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Nexus 5 (id=192.168.30.101:5555)
Spawned `com.lastpass.lpandroid`. Resuming main thread!
[Nexus 5::com.lastpass.lpandroid ]->
```

FIGURE 25 – FridaInject

2. Analyse des requettes HTTP dans BurpSuite

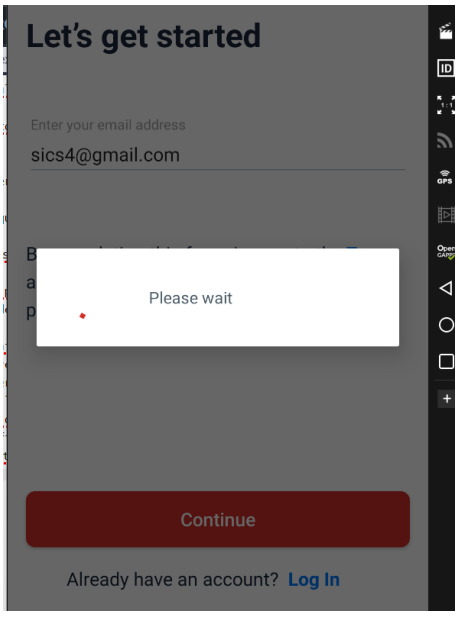


FIGURE 26 – account

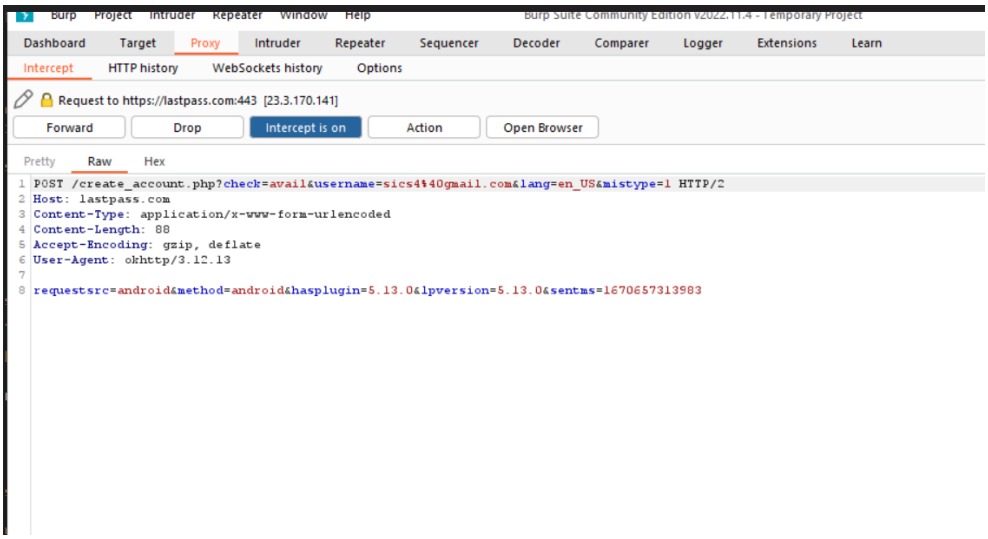


FIGURE 27 – accountRequest

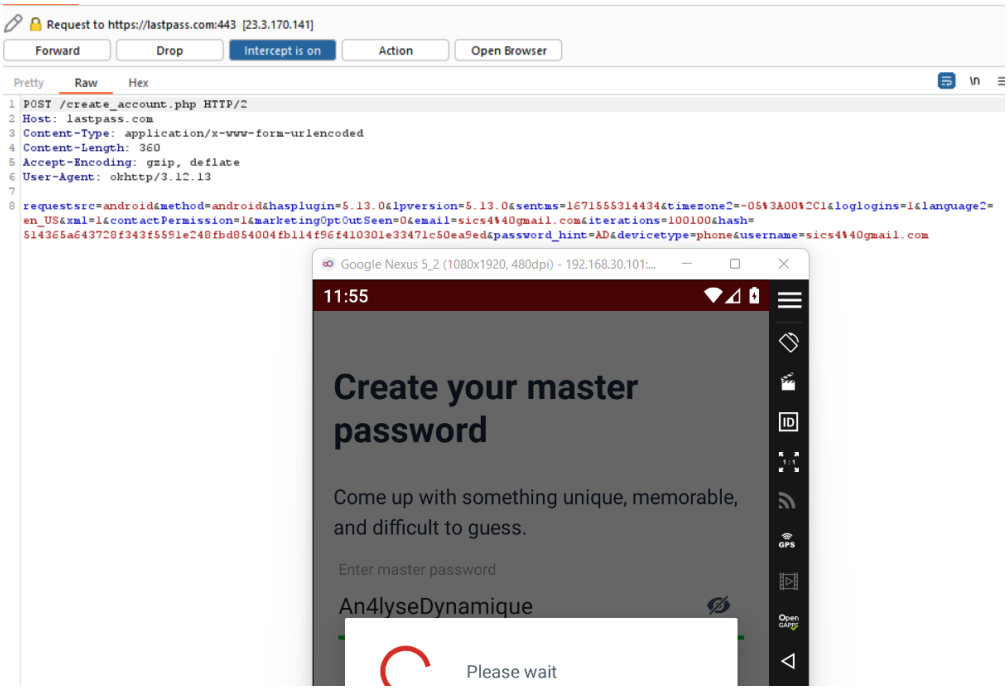


FIGURE 28 – MasterPassword

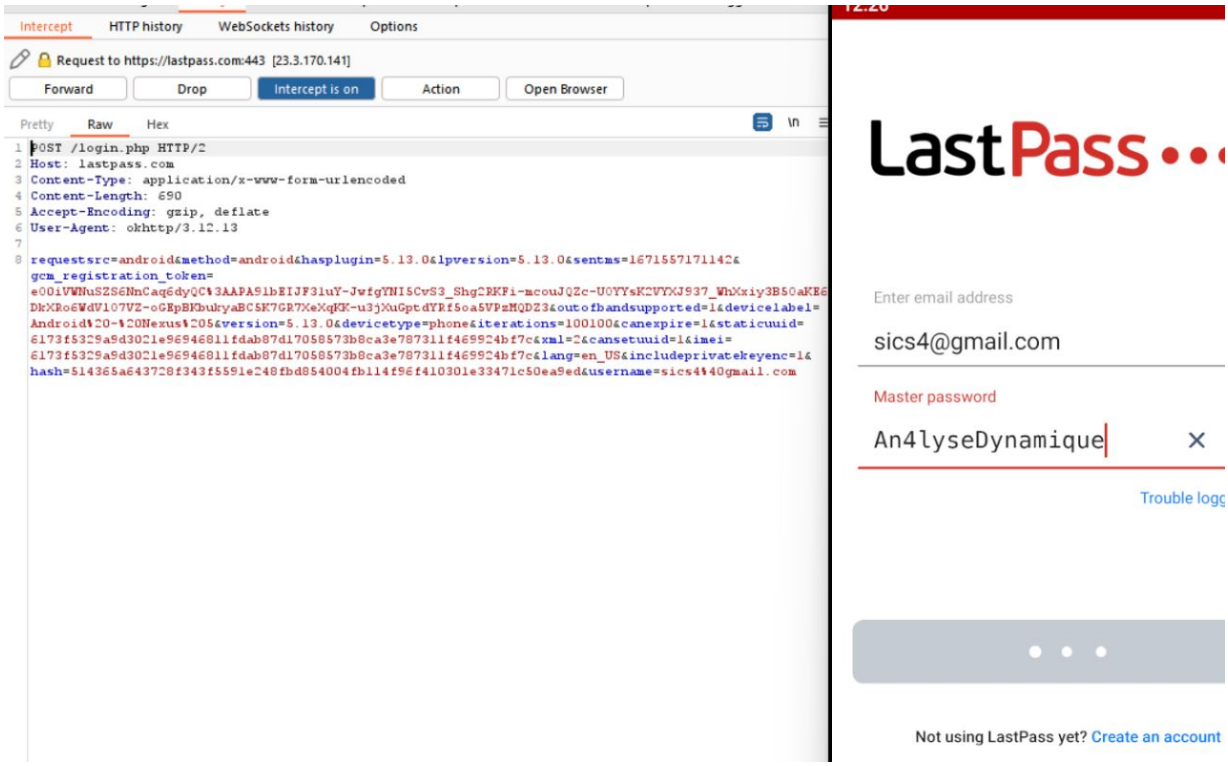


FIGURE 29 – Login

— gcm registration token=xxxxxxxxxxxxxxxxxxxxxxxxx
— outofbandsupported=1
— devicelabel=Android-Nexus5version=5.13.0 | devicetype=phone
— iterations=100100
— canexpire=1 | staticuuid=yyyyyyyyyyyyyy
— cansetuuid=1 | imei=yyyyyyyyyyyyyy
— ncludeprivatekeyenc=1
— hash= | username=sics4@gmail.com

ceci est une requette HTTP effectuée aux serveurs de lastpass utilisant la méthode POST. La demande est envoyée à l'URL "https ://lastpass.com/login.php" et contient divers en-têtes tels que "Content-Type" et "User-Agent", ainsi que plusieurs paramètres dans le corps de la demande.La demande est pour se connecter à notre un compte lastpass avec un nom d'utilisateur spécifique encodé et un hachage spécifique.

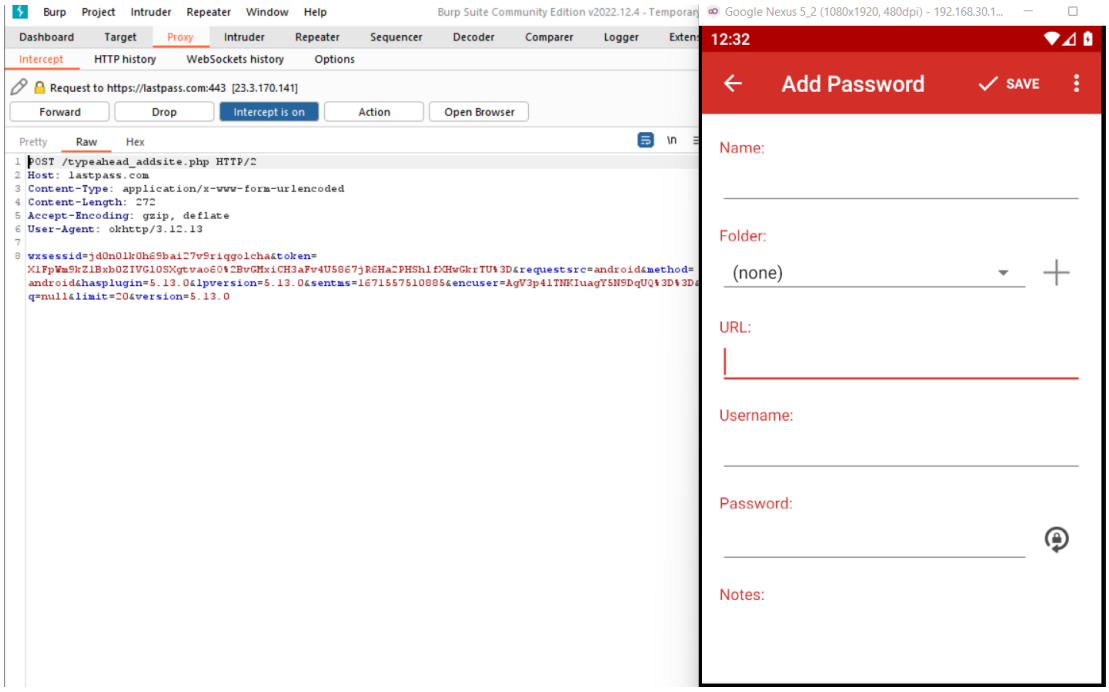


FIGURE 30 – AddPassword

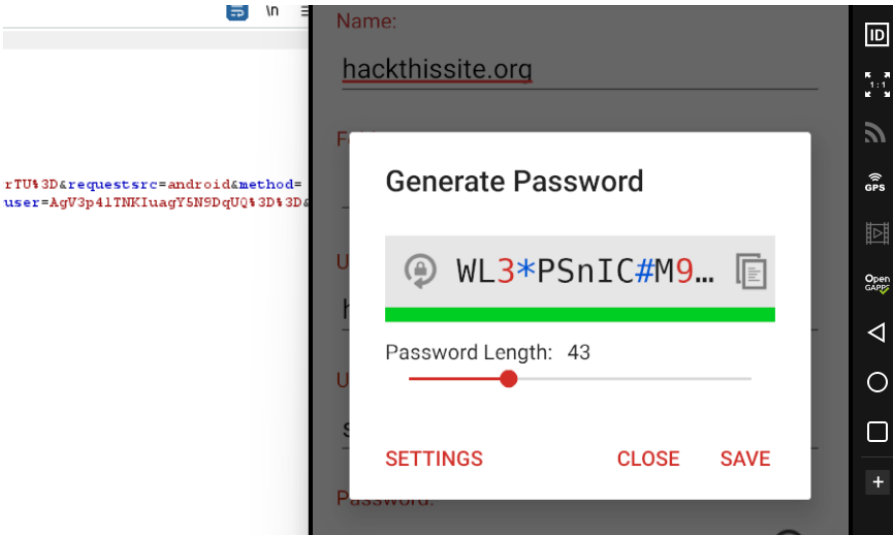


FIGURE 31 – generatePassword

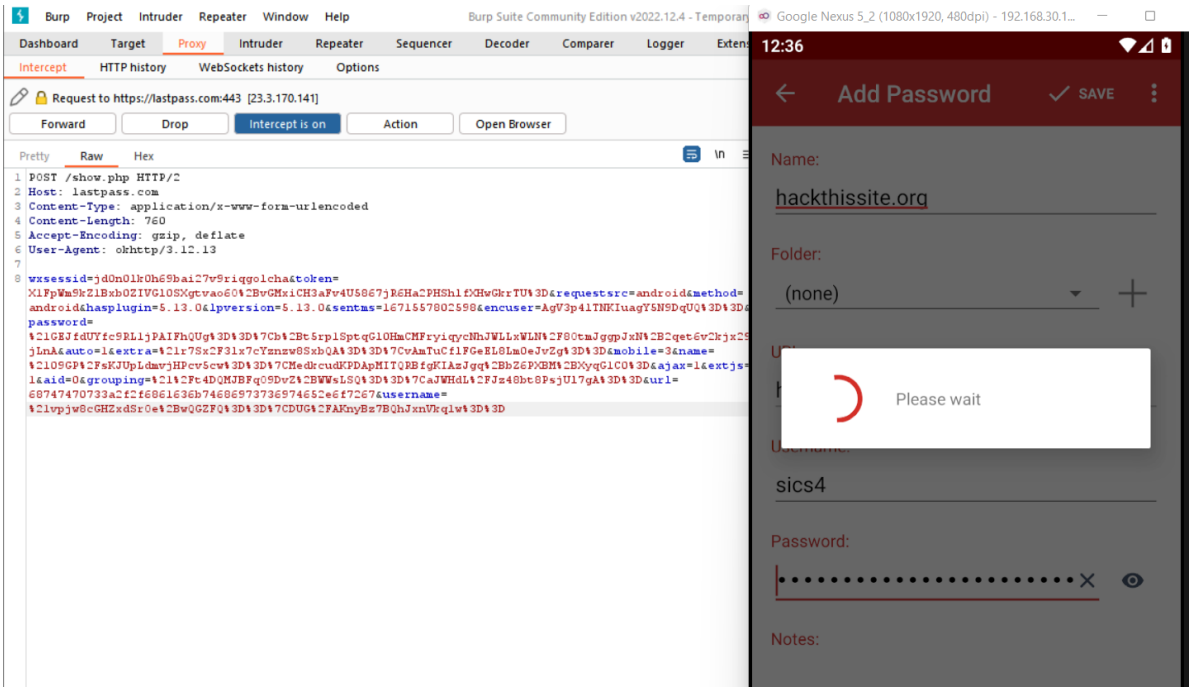


FIGURE 32 – savePassword

5.4 Conclusions

6 NordPass

6.1 Analyse Statique

6.2 Analuse Dynamique

6.3 Conclusion

7 Passky

7.1 Analyse Statique

7.2 Analyse Dynamique

7.3 Conclusion