École Nationale des Sciences Appliquées d'Oujda

Rapport du projet de Module sécurité mobile

Sujet :

‘Analyse d’application de portefeuilles et de trading’

Filière :

Sécurité informatique et cyber sécurité

Soutenu par :

FILALI Abdelilah

Sous la direction de :

Mr. KOULALI MOHAMMED AMIN

Année universitaire : 2022/2023

Table des Matières

Introduction Général

Les applications de trading et de gestion de portefeuille sont des outils en ligne qui permettent aux investisseurs de suivre et de gérer leurs investissements en temps réel. Elles fournissent généralement des informations en temps réel sur les marchés financiers, ainsi que des outils de gestion de portefeuille pour aider les utilisateurs à prendre des décisions d'investissement éclairées.

En raison de la nature de ces applications et de la gestion de fonds importants, la cybersécurité est une préoccupation majeure. Les pirates informatiques peuvent essayer de voler des informations sensibles, telles que les mots de passe et les détails de transaction, ou de perturber les systèmes pour influencer le marché financier. Il existe de nombreuses menaces potentielles pour la sécurité des applications de trading et de gestion de portefeuille, notamment les attaques de phishing, les logiciels malveillants, les violations de données

En effectuant une analyse de sécurité sur ces applications, les développeurs et les entreprises peuvent identifier les faiblesses de sécurité et prendre les mesures nécessaires pour protéger les utilisateurs contre ces menaces.

Parmi ces analyses de sécurité on trouve :

- **Analyse statique** consiste à examiner le code source ou le binaire d'une application de manière indépendante du contexte d'exécution. Cela peut être effectué manuellement ou en utilisant des outils automatisés. L'analyse statique vise à détecter les vulnérabilités potentielles dans le code, telles que les failles de sécurité, les erreurs de programmation et les pratiques de développement inadéquates.
- **Analyse dynamique**, en revanche, implique l'exécution de l'application dans un environnement de test et l'observation de son comportement pendant l'exécution. Cela peut être fait manuellement en utilisant des outils de test d'intrusion ou en utilisant des outils automatisés comme les outils de test d'application Web. L'analyse dynamique vise à détecter les vulnérabilités qui ne peuvent être détectées de manière statique, telles que les failles de sécurité qui ne sont exploitables qu'à des conditions spécifiques.

Chapitre 1 : Documentation

Remarque : Ce chapitre sera réservé à la documentation des notions fondamentaux de projet et les utiles utilisés.

Les outils pour effectuer l'analyse statique :

Jadx-gui:

Outil de décompilation qui permet de visualiser et de naviguer facilement dans le code source d'une application Android. Il peut être utilisé pour effectuer une analyse statique du code .

APK-TOOL :

APK Tool est un outil open source pour décompiler et recompiler des fichiers APK. Il peut décoder des ressources à la forme presque originale et les reconstruire après avoir fait quelques modifications; Il permet de déboguer le code smali étape par étape.

Ghidra :

Ghidra est un logiciel de reverse engineering , Il peut être utilisé pour décomposer du code binaire et pour analysé du code dans de nombreux langages de programmation différents, y compris le C, le C++. dont le but d'identifier les failles de sécurité dans le code et pour créer des patches de sécurité.

Les outils pour effectuer l'analyse dynamique :

bypasse ssl pinnig :

SSL pinning est une technique de sécurité utilisée pour empêcher l'interception et le man-in-the-middle attaque par des communications sécurisées par SSL/TLS. Lorsque l'application utilise SSL pinning, elle vérifie que le certificat présenté par le serveur correspond à un certificat de confiance prédéfini. Si le certificat ne correspond pas, l'application considère la connexion comme étant compromise et refuse de continuer à communiquer avec le serveur.

Il est possible de bypasser SSL pinning en utilisant des outils tels que Burp Suite, qui permettent d'intercepter les communications SSL et de fournir un certificat falsifié à l'application.

Burpsuite :

Burp Suite est un outil de sécurité informatique utilisé pour effectuer des tests d'intrusion et de sécurité des applications. L'outil fournit une interface graphique facile à utiliser qui permet de configurer et de lancer des tests, et il fournit également un grand nombre d'options de configuration avancées pour personnaliser les tests en fonction des besoins de l'utilisateur.

Frida :

Plateforme de modification de code pour les systèmes mobiles et de bureau. Et particulièrement utile pour le bypass du SSL pinning et l'analyse de sécurité des applications mobiles.

Genymotion :

Genymotion est un émulateur Android qui vous permet de tester et de développer des applications Android sur l'ordinateur, permet de simuler différents appareils et configurations Android. Est souvent utilisé par les développeurs pour tester et déboguer leur code

Chapitre 2 : Analyse statique

Méthodologie suivie :

Pour effectuer une analyse statique, on va essayer de répondre aux questions suivantes :

1. Comment l'application traite les données d'identification, comme les mots de passe et les jetons d'accès.
2. Comment l'application gère la confidentialité et la sécurité des données de l'utilisateur, comme les informations de paiement et les données sensibles.
3. Comment l'application accède aux services de réseau et aux bases de données externes, et comment elle traite les données qui y sont stockées.
4. Comment l'application est structurée et organisée, et comment elle utilise les différents fichiers et modules de code pour accomplir ses tâches.

Applications analyser :

ORANGE_MONEY :

Orange Money est le service de transfert d'argent et de paiement mobile , cette application accédez à une solution financière qui permet d´effectuer, à partir du mobile des opérations financières en toute simplicité, rapidité et autonomie.

Génération de la clé de chiffrement :

La méthode **generatePBEKey** génère une clé secrète pour le chiffrement PBE en utilisant un mot de passe, le sel et un compteur d'itérations. Le mot de passe et le sel sont fournis en entrée de la méthode sous forme de tableaux de caractères et de bytes, respectivement. Le compteur d'itérations et la taille de clé sont

également fournis en entrée. La méthode retourne un objet **SecretKey** qui peut être utilisé pour le chiffrement et le déchiffrement PBE.

La méthode **getDeviceSerialNumber** est utilisée pour récupérer le numéro de série de l'appareil, qui est un identifiant unique pour l'appareil. Il est utilisé en tant que sel pour générer une clé PBE dans la méthode **generateAesKeyName.**

L'utilisation du numéro de série de l'appareil comme partie du nom de clé garantit que le nom de clé est unique pour chaque appareil. Cela signifie que la valeur de la clé secrète sera également unique pour chaque appareil, ce qui contribue à augmenter la sécurité du processus de chiffrement et de déchiffrement.

```
  static SecretKey generatePBEKey(char[] cArr, byte[] bArr, String str, int i, int i2) throws
      if (i == 0) {
          i = 1000;
      }
      return SecretKeyFactory.getInstance(str, PROVIDER).generateSecret(new PBEKeySpec(cArr,
  }

  static String getDeviceSerialNumber(Context context) {
      try {
          String str = (String) Build.class.getField("SERIAL").get(null);
          return TextUtils.isEmpty(str) ? Settings.Secure.getString(context.getContentResolve
      } catch (Exception unused) {
          return Settings.Secure.getString(context.getContentResolver(), "android_id");
      }
  }
```

La méthode **generateAesKeyName** génère un nom de clé unique pour la clé secrète utilisée pour le chiffrement AES. Elle le fait en générant une clé PBE en utilisant le nom de package de l'application et le numéro de série du périphérique comme mot de passe et sel, respectivement. Le nom de clé est alors encodé en tant que chaîne et retourné

La clé PBE générée dans la méthode **generateAesKeyName** est utilisée comme nom de clé pour la clé secrète utilisée pour le chiffrement AES dans cette classe. Le nom de clé est stocké dans les préférences de l'application et est utilisé pour rechercher la valeur de la clé secrète lorsqu'elle est nécessaire pour le chiffrement ou le déchiffrement.

```
  static String generateAesKeyName(Context context) throws InvalidKeySpecException, NoSuchAlgorithmEx
      SecretKey generatePBEKey;
      char[] charArray = context.getPackageName().toCharArray();
      byte[] bytes = getDeviceSerialNumber(context).getBytes();
      try {
          generatePBEKey = generatePBEKey(charArray, bytes, PRIMARY_PBE_KEY_ALG, ITERATIONS, KEY_SIZE
      } catch (NoSuchAlgorithmException unused) {
          generatePBEKey = generatePBEKey(charArray, bytes, BACKUP_PBE_KEY_ALG, ITERATIONS, KEY_SIZE)
      }
      return encode(generatePBEKey.getEncoded());
  }
```

Data encryptions :

La classe **SecureCoderDecoder** fournit des méthodes pour chiffrer et déchiffrer des données en utilisant les algorithmes **AES (Advanced Encryption Standard) et PBE (Password-Based Encryption)**. Elle fournit également un moyen de générer et de stocker une clé secrète utilisée pour le chiffrement et le déchiffrement.

```
public SecureCoderDecoder(byte[] bArr) {
    this.sKey = bArr;
}

static byte[] decode(String str) {
    return Base64.decode(str, 3);
}

private static String encode(byte[] bArr) {
    return Base64.encodeToString(bArr, 3);
}
```

La méthode **encrypt** prend une chaîne en entrée et retourne une version chiffrée de la chaîne d'entrée en utilisant le chiffrement AES avec la clé secrète.

La méthode **decrypt** prend une chaîne chiffrée en entrée et retourne la chaîne d'origine en la déchiffrant en utilisant le déchiffrement AES avec la clé secrète.

```
public String decrypt(String str) {
    if (str == null || str.length() == 0) {
        return str;
    }
    try {
        Cipher cipher = Cipher.getInstance(AES_KEY_ALG, PROVIDER);
        cipher.init(2, new SecretKeySpec(this.sKey, AES_KEY_ALG));
        return new String(cipher.doFinal(decode(str)), "UTF-8");
    } catch (Exception unused) {
        return null;
    }
}

public String encrypt(String str) {
    if (str == null || str.length() == 0) {
        return str;
    }
    try {
        Cipher cipher = Cipher.getInstance(AES_KEY_ALG, PROVIDER);
        cipher.init(1, new SecretKeySpec(this.sKey, AES_KEY_ALG));
        return encode(cipher.doFinal(str.getBytes("UTF-8")));
    } catch (Exception unused) {
        return null;
    }
}
```

Transfert de données :

Ce code analyse et définit les valeurs pour une URL. Les valeurs qui sont définies incluent le nom d'utilisateur et le mot de passe, l'hôte, le port, les segments de chemin codés et la requête codée (s'il y en a). Les valeurs sont définies en fonction de l'URL d'entrée fournie

```
} else {
    this.encodedUsername = httpUrl.encodedUsername();
    this.encodedPassword = httpUrl.encodedPassword();
    this.host = httpUrl.host();
    this.port = httpUrl.port();
    this.encodedPathSegments.clear();
    this.encodedPathSegments.addAll(httpUrl.encodedPathSegments());
    if (indexOfFirstNonAsciiWhitespace$default == indexOfLastNonAsciiWhitespace$default
        encodedQuery(httpUrl.encodedQuery());
    }
    i = indexOfLastNonAsciiWhitespace$default;
}
```

Cette méthode semble être utilisée pour créer un objet Transaction à partir d'un objet **EthTransaction**, qui semble représenter une transaction Ethereum. L'objet Transaction est créé à l'aide des informations suivantes de l'objet **EthTransaction**

hash: the hash of the Ethereum transaction

from: the address of the account that sent the transaction

to: the address of the account that received the transaction (extracted from the input field of the transaction using the **extractToFromEthTransaction** method)

value: the value of the transaction (extracted from the input field of the transaction using the **extractValueFromEthTransaction** method)

status: the status of the transaction (passed as an argument to the method)

The Transaction object is then returned.

```
public static Transaction fromEthTransaction(EthTransaction ethTransaction, Transaction.Status status) {
    org.web3j.protocol.core.methods.response.Transaction transaction = ethTransaction.getTransaction();
    String hash = transaction.getHash();
    String from = transaction.getFrom();
    String input = ethTransaction.getTransaction().getInput();
    String extractToFromEthTransaction = extractToFromEthTransaction(input);
    String extractValueFromEthTransaction = extractValueFromEthTransaction(input);
    ethTransaction.getTransaction().getTo();
    return new Transaction(hash, from, extractToFromEthTransaction, extractValueFromEthTransaction, status);
}
```

Procédure de payement et de transaction :

La méthode **buildBuyIntent** crée un **PendingIntent** qui peut être utilisé pour démarrer une activité pour effectuer un paiement. Cette méthode prend en paramètre

Context: the context from which the **PendingIntent** will be launched

String: the package name of the app for which the payment is being made

String: the amount of the payment, in app coins

String: the product ID of the item being purchased

String: a developer payload string, which can be used to pass additional information about the purchase

String: the public key of the app, used for verifying the purchase

String: the package name of the app that will handle the payment

boolean: a flag indicating whether the payment should be made on the Ropsten test network (true) or the main Ethereum network (false)

```
public static PendingIntent buildBuyIntent(final Context context, final String str, final String str2, final String str3, final String str4, final String str5, fina
    AppCoinsAddressProxySdk createAddressProxySdk = new AppCoinsAddressProxyBuilder().createAddressProxySdk();
    final int i = z ? 3 : 1;
    return (PendingIntent) Single.m1800a(createAddressProxySdk.getAppCoinsAddress(i).m1791b(C9681a.m1857b()), createAddressProxySdk.getIabAddress(i).m1791b(C9681a.m
        @Override // p339q.p340a.p342c0.BiFunction
        /* renamed from: a */
        public final Object mo2197a(Object obj, Object obj2) {
            return GenericPaymentIntentBuilder.m27420a(context, i, str, str2, str3, str4, str5, str6, (String) obj, (String) obj2);
        }
    }).m1793b();
}

public static PendingIntent buildPaymentIntent(Context context, int i, String str, String str2, String str3, String str4, String str5, String str6, String str7, Str
    BigDecimal multiply = new BigDecimal(str2).multiply(BigDecimal.TEN.pow(18));
    Intent intent = new Intent("android.intent.action.VIEW");
    intent.setData(Uri.parse(buildUriString(str3, str4, multiply, str5, str, i, str6, str7, str8)));
    return PendingIntent.getActivity(context, 0, intent, 134217728);
}

private static String buildUriData(String str, String str2, String str3, String str4) throws UnsupportedEncodingException {
    return "0x" + Hex.m20248a(new Gson().m1977a(new TransactionData(str3, str2, str, str4)).getBytes("UTF-8")).toLowerCase();
}

private static String buildUriString(String str, String str2, BigDecimal bigDecimal, String str3, String str4, int i, String str5, String str6, String str7) {
    StringBuilder sb = new StringBuilder(4);
    try {
        new Formatter(sb).format("ethereum:%s@%d/buy?uint256=%s&address=%s&data=%s&iabContractAddress=%s", str, Integer.valueOf(i), bigDecimal.toString(), str3, bui
        return sb.toString();
    } catch (UnsupportedEncodingException e) {
        throw new RuntimeException("UTF-8 not supported!", e);
    }
}
```

La méthode crée d'abord une instance d'**AppCoinsAddressProxySdk** en utilisant la classe **AppCoinsAddressProxyBuilder**. Elle récupère ensuite l'adresse **AppCoins** et l'adresse IAB pour le réseau spécifié (soit Ropsten, soit le réseau principal) en utilisant les méthodes **getAppCoinsAddress** et **getIabAddress** de l'instance **d'AppCoinsAddress**

La méthode **buildPaymentIntent** est appelée une fois que les adresses **AppCoins** et IAB ont été récupérées, avec les paramètres suivants :

Context: the context from which the **PendingIntent** will be launched

int: the network ID (either 3 for Ropsten or 1 for the main network)

String: the package name of the app for which the payment is being made

String: the amount of the payment, in app coins

String: the AppCoins address

String: **the IAB address**

String: the product ID of the item being purchased

String: a developer payload string, which can be used to pass additional information about the purchase

String: the public key of the app, used for verifying the purchase

Une fois l'activité de paiement terminée, l'application appelante recevra le résultat via une méthode de rappel telle que **onActivityResult**, qui peut être utilisée pour mettre à jour l'interface utilisateur de l'application ou effectuer toute autre tâche nécessaire

IAB signifie In-App Billing. Les adresses IAB sont des adresses Ethereum qui correspondent à des contrats intelligents qui gèrent les achats intégrés dans une application ou un jeu particulier. Ces contrats intelligents sont chargés de vérifier et de compléter le processus de paiement, ainsi que d'enregistrer la transaction sur la blockchain Ethereum.

Dans le contexte de la méthode buildPaymentIntent, l'adresse IAB est incluse dans la chaîne URI utilisée pour démarrer l'activité de paiement. L'activité de paiement utilisera cette adresse pour communiquer avec le contrat intelligent IAB et terminer le processus de paiement.

Validation de Certificat

La méthode vérifie d'abord si le niveau de l'API Android est 17 ou supérieur et si f11336b est non null si est vrai, elle récupère les certificats du serveur de la **HttpsURLConnection** et vérifie leur validité à l'aide de la méthode **checkServerTrusted**. Si la chaîne de certificats n'est pas valide, elle lève une **SSLException**.(Indique une sorte d'erreur détectée par un sous-système SSL)

Si le niveau d'API Android est inférieur à 17 ou si f11336b est nul, la méthode récupère les certificats du serveur et vérifie leur validité à l'aide de la méthode de vérification du X509TrustManager par défaut. Si la chaîne de certificats n'est pas valide, elle lève une **SSLException**.

```
public static void m21383a(HttpsURLConnection httpsURLConnection) throws SSLException {
    List<X509Certificate> checkServerTrusted;
    if (Build.VERSION.SDK_INT >= 17 && f11336b != null) {
        String str = "";
        try {
            Certificate[] serverCertificates = httpsURLConnection.getServerCertificates();
            checkServerTrusted = f11336b.checkServerTrusted((X509Certificate[]) Arrays.copyOf(serverCertificates, serve
        } catch (NoSuchAlgorithmException e) {
            C5217d1.m21519a("SslPinningValidator", "Error in validating pinning: ", e);
        } catch (CertificateException e2) {
            C5217d1.m21519a("SslPinningValidator", "Error in getting certificate: ", e2);
        }
        if (checkServerTrusted != null) {
            MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
            for (X509Certificate x509Certificate : checkServerTrusted) {
                byte[] encoded = x509Certificate.getPublicKey().getEncoded();
                messageDigest.update(encoded, 0, encoded.length);
                String encodeToString = Base64.encodeToString(messageDigest.digest(), 2);
                if (f11335a.contains(encodeToString)) {
                    C5217d1.m21520a("SslPinningValidator", "Found matched pin: ".concat(String.valueOf(encodeToString))
                    return;
                }
                str = str + "    sha256/" + encodeToString + ": " + x509Certificate.getSubjectDN().toString() + "\n";
            }
```

IQ option :

IQ Option est une plate-forme multi-actifs qui offre à la fois des investissements en actions, en crypto-devises, ainsi que des actifs de trading

Génération de la clé de chiffrement :

CredentialsModel est une classe qui stocke divers types d'informations d'identification, telles qu'un jeton de porteur ou un jeton et un secret d'API, qui sont utilisées pour authentifier un utilisateur ou un appareil. Il stocke également le centre de données auquel l'utilisateur ou l'appareil est associé.

La classe **CredentialsModel** fournit également une méthode pour générer une nouvelle clé de session. La classe **SessionKey** a une méthode pour vérifier si elle est valide, qui est déterminée par le fait que la clé secrète et l'IV ne sont pas tous les deux nuls

```
/* loaded from: classes3.dex */
public class CredentialsModel implements StaticModel {
    private String authorization;
    private JumioDataCenter dataCenter;
    private SessionKey sessionKey = new SessionKey();
    private boolean usesBearerToken = false;

    /* loaded from: classes3.dex */
    public class SessionKey implements Serializable {
        private SecretKey key = null;
        private byte[] vector = null;

        public SessionKey() {
        }

        public SecretKey getKey() {
            return this.key;
        }

        public IvParameterSpec getVector() {
            return new IvParameterSpec(this.vector);
        }

        public boolean isSessionKeyValid() {
            return (this.key == null || this.vector == null) ? false : true;
        }
    }
```

La classe **SessionKey** est une classe imbriquée qui stocke une clé secrète **AES et un vecteur d'initialisation (IV)** qui sont utilisés pour chiffrer et déchiffrer les données :

```
    public void generateSessionKey(SessionKey sessionKey) {
        try {
            sessionKey.key = KeyGenerator.getInstance("AES").generateKey();
            sessionKey.vector = new byte[16];
            new SecureRandom().nextBytes(sessionKey.vector);
        } catch (NoSuchAlgorithmException e) {
            Log.printStackTrace(e);
        }
    }
```

Data encryptions :

AesEaxJce est une classe qui fournit un moyen de chiffrer et de déchiffrer des données à l'aide d'AES en mode EAX

EAX (Authenticated Encryption with Associated Data) est un mode de fonctionnement pour les chiffrements par blocs tels que AES. Il permet le chiffrement et l'authentification des données en une seule opération. Cela signifie qu'en plus de crypter les données, il génère également un code d'authentification de message (MAC) qui peut être utilisé pour vérifier l'authenticité et l'intégrité des données cryptées.

La classe AesEaxJce prend une clé secrète et un IV (vecteur d'initialisation) en entrée et les utilise pour initialiser une instance de la classe. Il fournit ensuite des méthodes pour chiffrer et déchiffrer les données, ainsi que des méthodes pour calculer et vérifier le MAC pour les données. Les méthodes de chiffrement et de déchiffrement utilisent la classe Cipher de JCE pour effectuer le chiffrement et le déchiffrement réels, et les méthodes de calcul et de vérification MAC utilisent la propre implémentation de la classe.

```
/* compiled from: AesEaxJce.java */
/* renamed from: g5.c$b */
/* loaded from: classes2.dex */
public class C7119b extends ThreadLocal<Cipher> {
    @Override // java.lang.ThreadLocal
    public final Cipher initialValue() {
        try {
            return EngineFactory.f21014e.m8983a("AES/CTR/NOPADDING");
        } catch (GeneralSecurityException e) {
            throw new IllegalStateException(e);
        }
    }
}

public AesEaxJce(byte[] bArr, int i) {
    if (i != 12 && i != 16) {
        throw new IllegalArgumentException("IV size should be either 12 or 16 bytes");
    }
    this.f20968d = i;
    Validators.m8963a(bArr.length);
    SecretKeySpec secretKeySpec = new SecretKeySpec(bArr, "AES");
    this.f20967c = secretKeySpec;
    Cipher cipher = f20963e.get();
    cipher.init(1, secretKeySpec);
    byte[] m9001c = m9001c(cipher.doFinal(new byte[16]));
    this.f20965a = m9001c;
    this.f20966b = m9001c(m9001c);
}
```

Validation de Certificat :

TLSSocketFactory est une implémentation de l'interface **SSLSocketFactory**. Elle est utilisée pour créer des sockets SSL/TLS (canal de communication sécurisé entre deux machines). Cette classe possède deux constructeurs :

Un constructeur par défaut qui crée une instance de **TLSSocketFactory** en utilisant un **SSLContext** par défaut.

Un constructeur qui prend un objet **InputStream** en argument. Ce constructeur crée une instance de **TLSSocketFactory** en chargeant les certificats à partir de l'**InputStream** donné et en créant un **SSLContext** personnalisé à l'aide de ces certificats

X509Certificate est un type de certificat utilisé dans les systèmes d'infrastructure à clé publique (PKI) pour vérifier qu'une clé publique appartient à l'identité de l'utilisateur, de l'appareil ou du service contenue dans le certificat. Il s'agit d'un certificat numérique qui utilise la norme internationale X509 largement acceptée pour authentifier l'identité du propriétaire du certificat. Dans ce code, une **CertificateFactory** est utilisée pour générer une collection d'objets X509Certificate à partir d'un **InputStream**. Les certificats sont ensuite ajoutés à un objet KeyStore, qui est une base de données de certificats et de clés privées associées. Le KeyStore est utilisé pour initialiser une **TrustManagerFactory**, qui est responsable de la gestion du matériel de confiance utilisé pour déterminer si une connexion doit être établie avec une entité distante. La **TrustManagerFactory** est ensuite utilisée pour initialiser un **SSLContext**, qui est utilisé pour créer une **SSLSocketFactory** qui établit des connexions sécurisées.

```
public TLSSocketFactory(InputStream inputStream) {
    try {
        try {
            KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
            keyStore.load(null, null);
            for (Certificate certificate : CertificateFactory.getInstance("X.509").generateCertificates(inputStream)) {
                if (certificate instanceof X509Certificate) {
                    keyStore.setCertificateEntry(((X509Certificate) certificate).getSubjectDN().getName(), certificate);
                }
            }
            TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
            trustManagerFactory.init(keyStore);
            SSLContext sSLContext = SSLContext.getInstance("TLS");
            sSLContext.init(null, trustManagerFactory.getTrustManagers(), null);
            this.f4005a = sSLContext.getSocketFactory();
            try {
                inputStream.close();
            } catch (IOException | NullPointerException unused) {
            }
        } catch (Throwable th2) {
            try {
                inputStream.close();
            } catch (IOException | NullPointerException unused2) {
            }
            throw th2;
        }
    } catch (Exception e) {
        throw new SSLException(e.getMessage());
    }
}


public void checkCertificate(X509Certificate x509Certificate) {
    try {
        try {
            x509Certificate.checkValidity();
        } catch (CertificateNotYetValidException e) {
            Log.m10660w("JumioTrustManager", "SSL Certificate is not yet valid", e);
        }
    } catch (Exception e2) {
        throw new CertificateException("SSL Certificate match error", e2);
    }
}
```

Transmission et la sécurisation de données :

La classe fournit plusieurs méthodes pour créer des sockets SSL/TLS et configurer leur comportement. Certaines des méthodes importantes sont :

createSocket : Cette méthode est utilisée pour créer un nouveau socket qui se connecte à l'hôte distant spécifié sur le port spécifié.

```
@Override // javax.net.SocketFactory
public final Socket createSocket(InetAddress inetAddress, int i) {
    Socket createSocket = this.f4005a.createSocket(inetAddress, i);
    m16148a(createSocket);
    return createSocket;
}
```

createSocket : Cette méthode est utilisée pour créer un nouveau socket qui se connecte à l'hôte distant spécifié sur le port spécifié et le lie à l'adresse et au port locaux spécifiés.

```
@Override // javax.net.SocketFactory
public final Socket createSocket(InetAddress inetAddress, int i, InetAddress inetAddress2, int i2) {
    Socket createSocket = this.f4005a.createSocket(inetAddress, i, inetAddress2, i2);
    m16148a(createSocket);
    return createSocket;
}
```

getSupportedCipherSuites : Cette méthode renvoie les noms des suites de chiffrement qui sont prises en charge par cette **SSLSocketFactory**.

```
@Override // javax.net.ssl.SSLSocketFactory
public final String[] getDefaultCipherSuites() {
    return this.f4005a.getDefaultCipherSuites();
}
```

Dat pour établir la sécurité sur un réseau. La classe **OkHttpClient** utilise de nombreuses méthodes pour mettre en place et gérer la communication de données entre un client et un serveur, Elle dispose de différents champs pour configurer le client tels que :

connectionSpecs pour configurer les types de connexions que le client doit établir

proxy pour spécifier un proxy à utiliser.

sslSocketFactoryOrNull pour fournir une SSLSocketFactory personnalisée pour créer des sockets sécurisés.

authentificateur fournissant des justificatifs d'identité pour s'authentifier auprès de serveurs.

retryOnConnectionFailure pour spécifier si le client doit réessayer les demandes lorsqu'une connexion échoue .

Dispatcher pour contrôler la simultanéité des requêtes du client a flow communication.

```java
public class OkHttpClient implements Cloneable, Call.Factory, WebSocket.Factory {
    private final Authenticator authenticator;
    private final Cache cache;
    private final int callTimeoutMillis;
    private final CertificateChainCleaner certificateChainCleaner;
    private final CertificatePinner certificatePinner;
    private final int connectTimeoutMillis;
    private final ConnectionPool connectionPool;
    private final List<ConnectionSpec> connectionSpecs;
    private final CookieJar cookieJar;
    private final Dispatcher dispatcher;
    private final Dns dns;
    private final EventListener.Factory eventListenerFactory;
    private final boolean followRedirects;
    private final boolean followSslRedirects;
    private final HostnameVerifier hostnameVerifier;
    private final List<Interceptor> interceptors;
    private final long minWebSocketMessageToCompress;
    private final List<Interceptor> networkInterceptors;
    private final int pingIntervalMillis;
    private final List<Protocol> protocols;
    private final Proxy proxy;
    private final Authenticator proxyAuthenticator;
    private final ProxySelector proxySelector;
    private final int readTimeoutMillis;
    private final boolean retryOnConnectionFailure;
    private final RouteDatabase routeDatabase;
    private final SocketFactory socketFactory;
    private final SSLSocketFactory sslSocketFactoryOrNull;
    private final int writeTimeoutMillis;
    private final X509TrustManager x509TrustManager;
```

newSSLContext() : crée un nouvel objet **SSLContext**, qui est utilisé pour transmettre en toute sécurité des données sur le réseau.

newSslSocketFactory() : crée un nouvel objet **SSLSocketFactory**, qui est utilisé pour créer des objets **SSLSocket** pour une communication sécurisée.

```java
    public SSLContext newSSLContext() {
        SSLContext sSLContext = SSLContext.getInstance("TLS");
        C11018h.m1570g(sSLContext, "SSLContext.getInstance(\"TLS\")");
        return sSLContext;
    }

    public SSLSocketFactory newSslSocketFactory(X509TrustManager x509TrustManager) {
        C11018h.m1569h(x509TrustManager, "trustManager");
        try {
            SSLContext newSSLContext = newSSLContext();
            newSSLContext.init(null, new TrustManager[]{x509TrustManager}, null);
            SSLSocketFactory socketFactory = newSSLContext.getSocketFactory();
            C11018h.m1570g(socketFactory, "newSSLContext().apply {\n…ll)\n      }.socketFactory");
            return socketFactory;
        } catch (GeneralSecurityException e) {
            throw new AssertionError("No System TLS: " + e, e);
        }
    }
```

Chapitre 3: Analyse dynamique

Lab installation:

On va analyser comment l'application traite les données pendant l'exécution et comment envoi et intercepte les requetés, pour cela il faut effectuer une MITM attaque a l'aide du méthode **bypass ssl pinning**, et les outils **Burpsuite**, **Frida,** et **Emulateur Android (j'ai** choisi **Genymotion)**

Les étapes pour installer et configurer LAB sont bien détaillées dans se site :

https://medium.com/@brewmon15/ssl-pinning-and-how-to-bypass-it-on-android-platform-by-frida-feat-nox-emulator-for-mac-user-e7a38d360d0c

Points analysés :

Problèmes de cryptage complet/partiel :

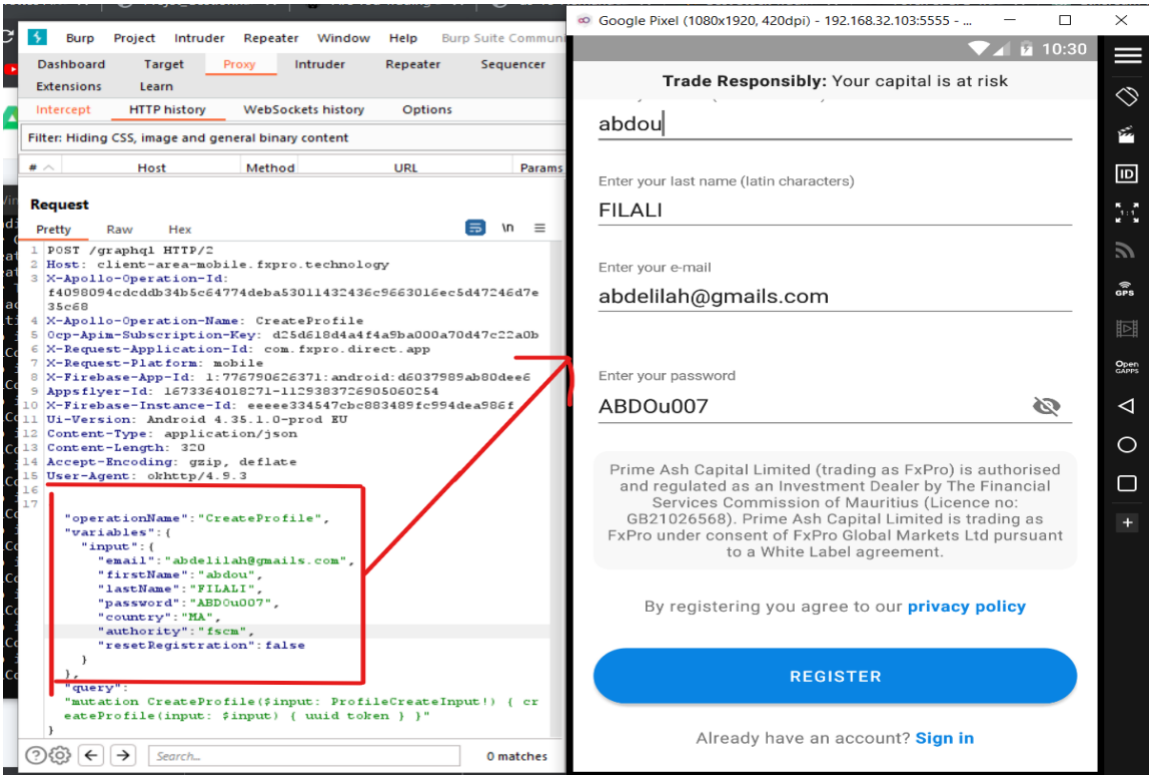
La plupart des applications chiffrent toutes les données mais laissent quelques requêtes non cryptées :

- ✓ Http channel protocol
- ✓ logging username

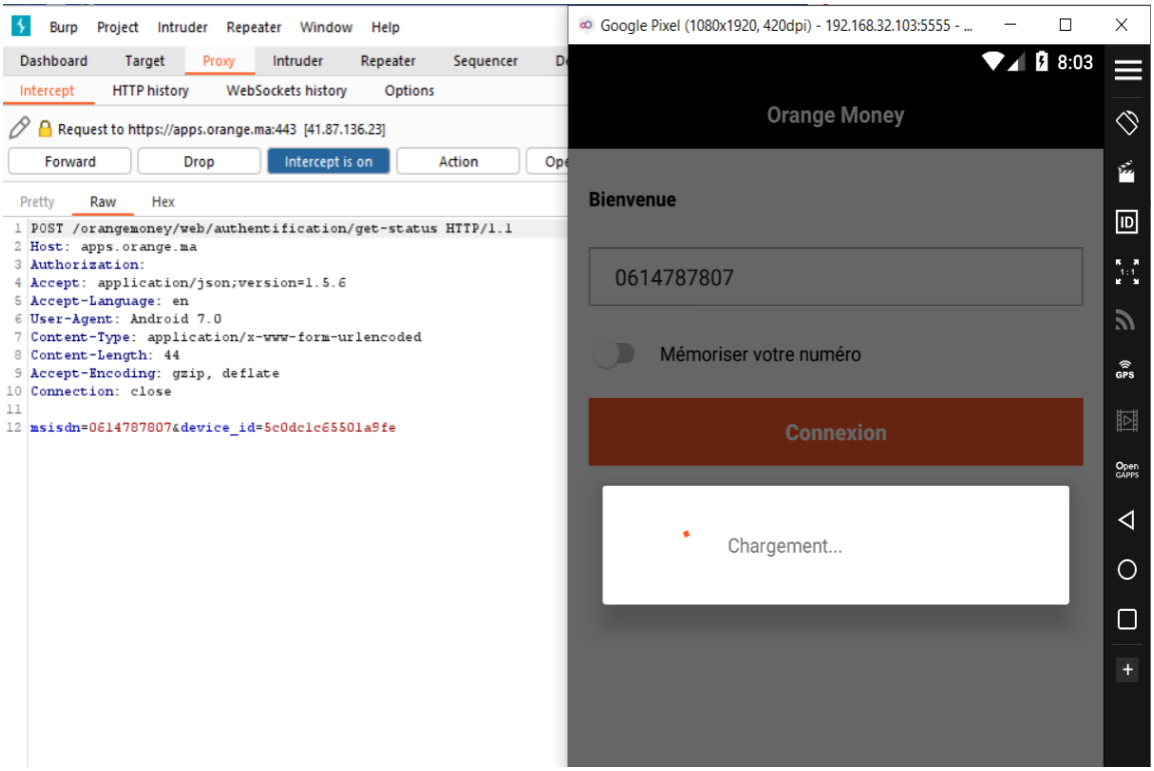
Example application:

FXpro :

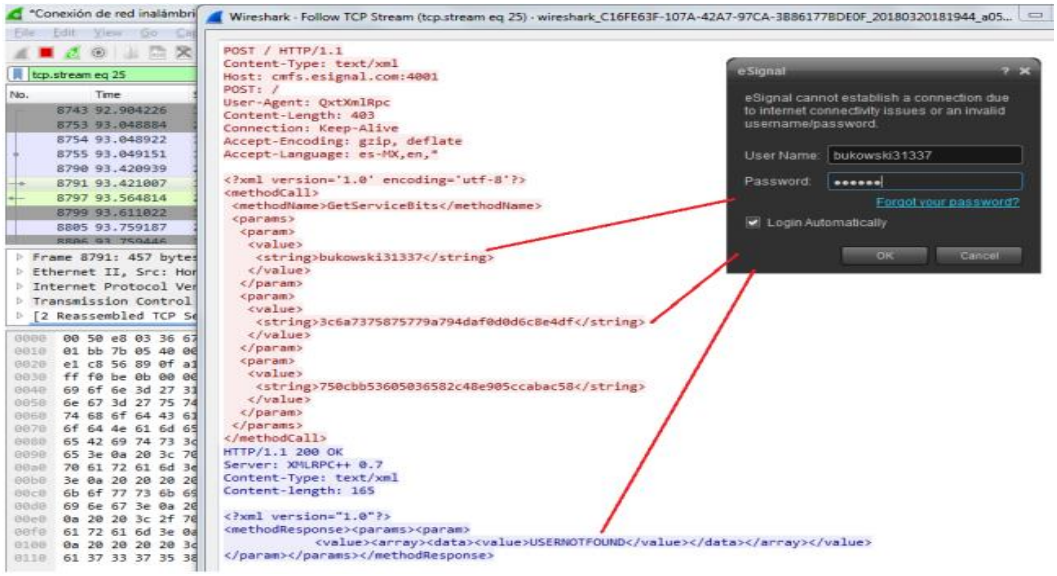
Le champ d'email et mot de passe n'est pas chiffre



ORANGE-money : elle se base pour authentification sur numéro de téléphone



E-SIGNAL :

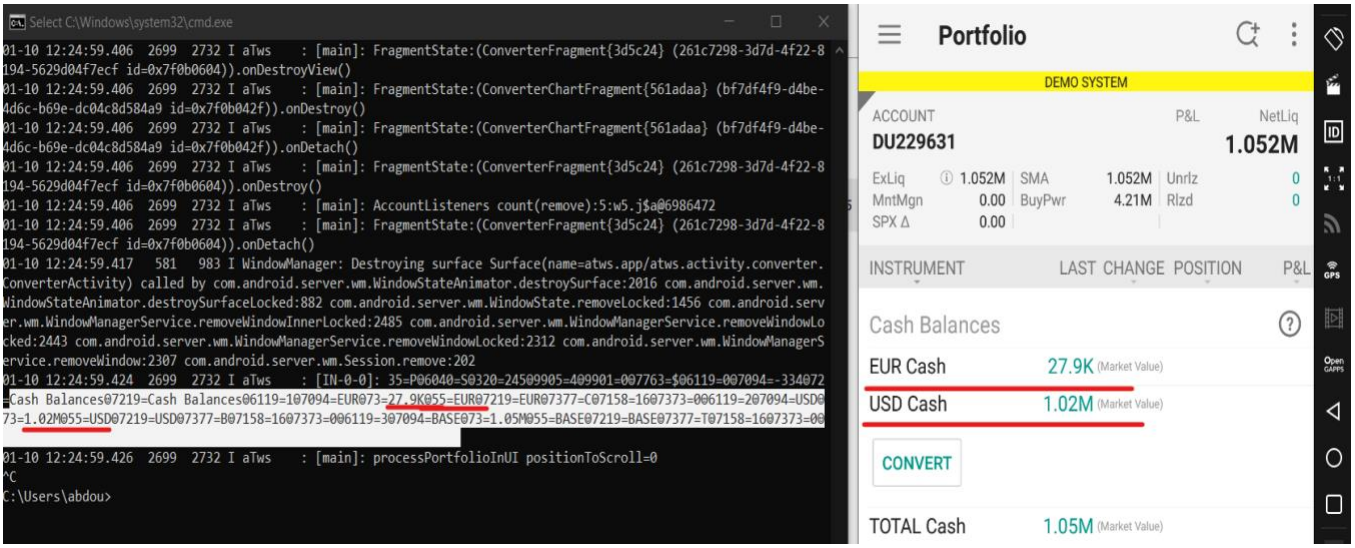


LOGS file:

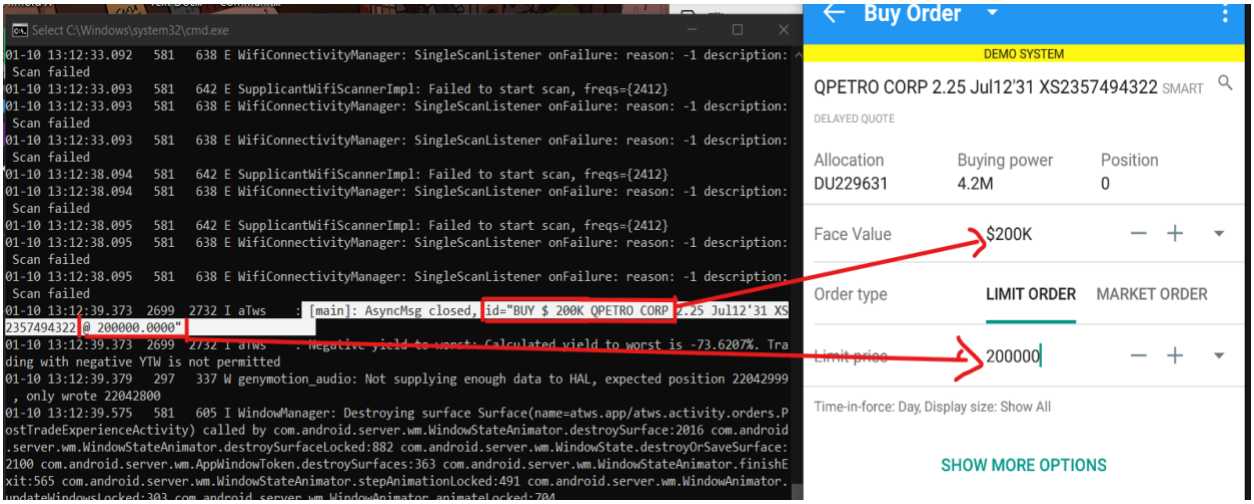
Les applications envoient des données sensibles aux log file et les stockent sans chiffrement

InteractiveBrokers :

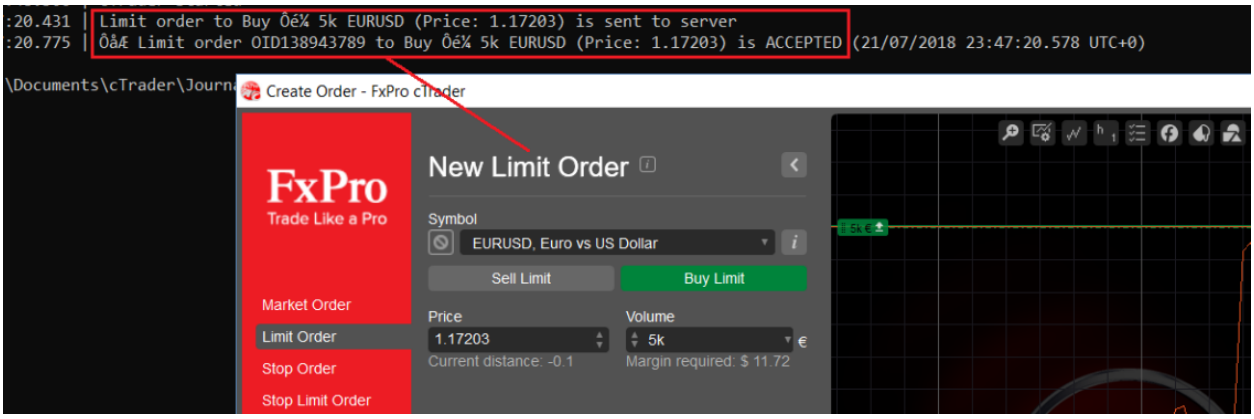
Balance de compte n’est pas chiffrée



Historique d’achat :



FX pro :



Session remains active after logout :

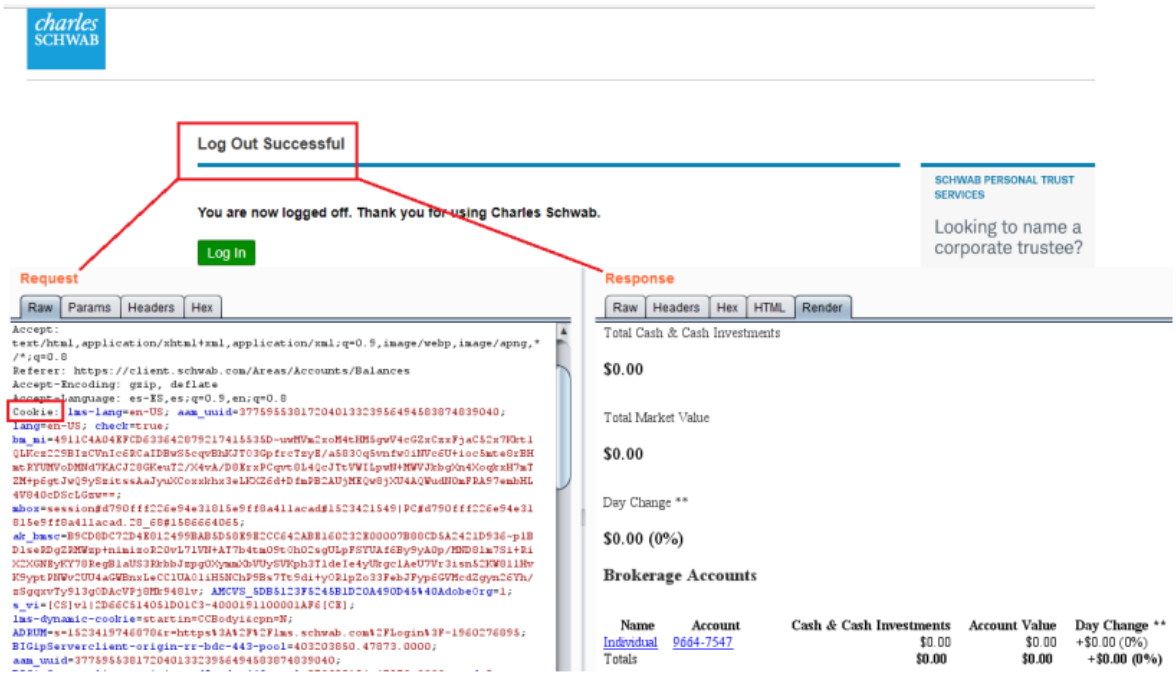
Une erreur fréquente dans la fin de session est que le jeton de session côté client est défini sur une nouvelle valeur tandis que l'état côté serveur reste actif et peut être réutilisé en redéfinissant le cookie de session sur la valeur précédente. Parfois, seul un message de confirmation est affiché à l'utilisateur sans effectuer aucune autre action.

Pour vérifie cela :

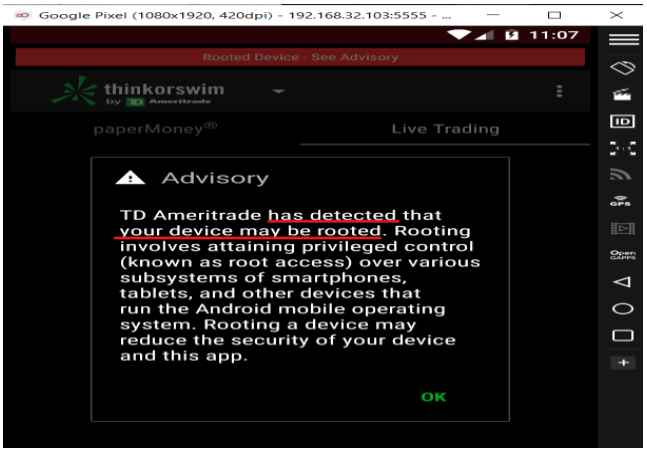
Cliqué sur le bouton "déconnexion"

Envoyé une demande précédemment capturée avec burpsuite +1 heure plus tard

On obtient une réponse du serveur même si on est déconcerté



Detection d'appareil rooté :



Peu d'applications qui ont détecte que ma machine est rooté **:TD Ameritrade app**

Communication non sécurisée :

Un utilisateur non expérimenté qui ne connaît pas la signification de "SSL" ne l'activera pas sur login page, et toutes les données sensibles seront envoyées et reçues en texte clair, sans cryptage.

Conclusion

En conclusion, l'ingénierie inverse et l'analyse de l'application de trading ont révélé des informations sur son fonctionnement et ont identifié des domaines potentiels d'amélioration. Les performances, la sécurité et les fonctionnalités globales de l'application peuvent être améliorées en fonction des résultats de l'analyse. Des tests et des recherches supplémentaires devraient être menés pour confirmer les faiblesses identifiées et évaluer l'efficacité des modifications proposées.

Fin .