TP Android Security

LAB 1

Mohamed BOUCHENGUOUR
Hamadi DAGHAR
MASTER 2 CYBERSECURITE
POLYTECH NICE SOPHIA

Nous avons tout d'abord cherché le nom du package en explorant le code source de l'application. Ce dernier va nous servir au moment où nous allons lancer le script Frida. Ce dernier se trouve dans la classe **R** sous le nom de **owasp.mstg.uncrackable1**.

Ensuite, nous essayons de comprendre l'application. Nous voyons d'abord qu'il y a une détection du mode root et du mode Debug :

```java
@Override // android.app.Activity
protected void onCreate(Bundle bundle) {
    if (c.a() || c.b() || c.c()) {
        a("Root detected!");
    }
    if (b.a(getApplicationContext())) {
        a("App is debuggable!");
    }
    super.onCreate(bundle);
    setContentView(R.layout.activity_main);
}
```

- **c.a()** : Recherche le binaire su dans le PATH.
- **c.b()** : Vérifie si le firmware est signé avec des test-keys.
- **c.c()** : Recherche des fichiers root spécifiques dans le système.
- **b.a(Context context)** : Vérifie si l'application est en mode débogage.

Une autre méthode utilise l'algorithme **AES** en mode **ECB** avec un padding **PKCS7** pour déchiffrer un texte chiffré.

```java
package sg.vantagepoint.a;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

/* loaded from: classes.dex */
public class a {
    public static byte[] a(byte[] bArr, byte[] bArr2) {
        SecretKeySpec secretKeySpec = new SecretKeySpec(bArr, "AES/ECB/PKCS7Padding");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(2, secretKeySpec);
        return cipher.doFinal(bArr2);
    }
}
```

La méthode utilise l'algorithme **AES** en mode **ECB** avec le padding **PKCS7Padding** pour déchiffrer les données. Une clé brute (**bArr**) est transformée en une clé utilisable via **SecretKeySpec**. Le **Cipher** est ensuite initialisé en mode déchiffrement (**init(2)**) avec cette clé. Enfin, la méthode **doFinal** applique le déchiffrement sur les données chiffrées (**bArr2**) et retourne les données déchiffrées.

Ces deux éléments sont intégrés dans la partie principale de l'application, qui s'exécute comme suit :

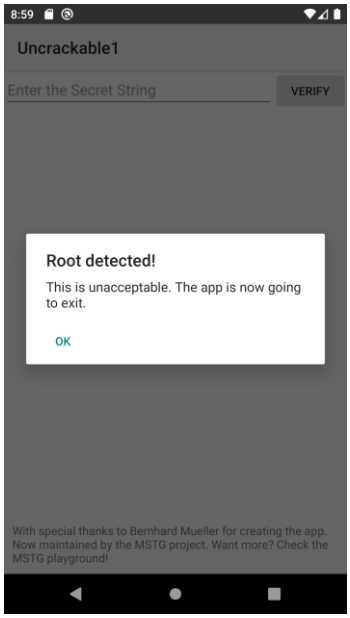
Vérifications au démarrage (**onCreate**) :

- Détecte si l'appareil est rooté (**c.a()**, **c.b()**, **c.c()**).
- Vérifie si l'app est en mode debug (**b.a()**).
- En cas de détection, affiche une alerte et ferme l'application.

Vérification du secret **(verify) :**

- Récupère l'entrée utilisateur depuis le champ texte.
- Déchiffre la valeur stockée à l'aide de **a.a()**.
- Compare la valeur déchiffrée avec celle saisie.
- Affiche "Success!" si correct, sinon "Nope...".

```
public static boolean a(String str) {
    byte[] bArr;
    byte[] bArr2 = new byte[0];
    try {
        bArr = sg.vantagepoint.a.a.a(b("8d127684cbc37c17616d806cf50473cc"), Base64.decode("5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=", 0));
    } catch (Exception e) {
        Log.d("CodeCheck", "AES error:" + e.getMessage());
        bArr = bArr2;
    }
    return str.equals(new String(bArr));
}
```

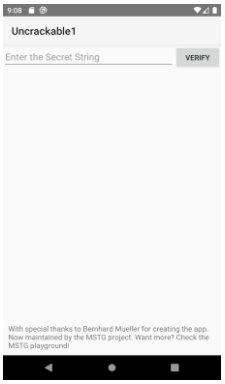


Lors du premier lancement de l'application, nous sommes bloqués en raison des permissions root. Cela est dû au fait qu'Android Studio configure la simulation du téléphone avec des permissions root. Le blocage se produit au niveau de la fonction **onCreate**, plus précisément avec les méthodes **a**, **b**, et **c** abordées précédemment. Pour résoudre ce problème, nous allons forcer le retour de false pour ces trois méthodes, ainsi que pour la détection du mode Debug en réalisant le code suivant pour chaque méthode :

```
RootDetection.a.implementation = function () {
    console.log("RootDetection.a() intercepted!");
    return false; // Forcer false
};
```

Nous lançons l'application avec le script Frida, éliminant ainsi les blocages liés à la détection du mode root ou Debug.

```
RootDetection.a() intercepted!
RootDetection.b() intercepted!
RootDetection.c() intercepted!
DebugDetection.a() intercepted!
```

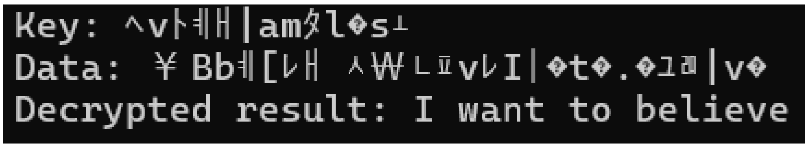


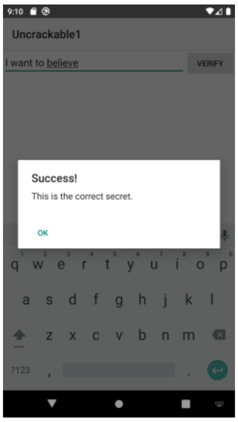
Nous devons désormais saisir la phrase secrète. Lors de cette étape, l'application compare la phrase saisie avec une valeur stockée et chiffrée. Pour effectuer la comparaison, elle déchiffre cette valeur à l'aide de la méthode **a.a()**. Nous allons donc écrire un script pour intercepter le retour de cette méthode, afin de récupérer et afficher la valeur déchiffrée.

```javascript
AES.a.overload("[B", "[B").implementation = function (key, data) {
    console.log("Key: " + bytesToString(key));
    console.log("Data: " + bytesToString(data));

    var result = this.a(key, data); // Appeler l'implémentation originale

    console.log("Decrypted result: " + bytesToString(result));
    return result;
};

function bytesToString(byteArray) {
    var result = "";
    for (var i = 0; i < byteArray.length; i++) {
        result += String.fromCharCode(byteArray[i]);
    }
    return result;
}

console.log("Hook applied to AES decryption.");
```

La valeur retournée étant en bytes, il suffit de la convertir en String pour afficher directement le texte en clair. Nous relançons l'application avec le script Frida, puis cliquons sur **Verify** afin de déclencher la fonction et récupérer son retour.

Nous avons maintenant accès à la phrase secrète : il ne reste plus qu'à la saisir.





La phrase secrète peut être obtenue sans utiliser Frida. Le code contient la clé, le texte chiffré, et la méthode de chiffrement. En intégrant ces éléments dans un programme Python, il est possible de la déchiffrer directement :

```python
from Crypto.Cipher import AES  # Importer AES pour le déchiffrement
import base64  # Importer base64 pour décoder le texte chiffré

hex_key = "8d127684cbc37c17616d806cf50473cc"  # Clé AES en hexadécimal
base64_ciphertext = "5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc="  # Texte chiffré encodé en Base64
key = bytes.fromhex(hex_key)  # Convertir la clé hexadécimale en bytes
ciphertext = base64.b64decode(base64_ciphertext)  # Décoder le texte chiffré de Base64 en bytes
cipher = AES.new(key, AES.MODE_ECB)  # Initialiser le déchiffreur AES en mode ECB
plaintext = cipher.decrypt(ciphertext)  # Déchiffrer le texte chiffré
print("Texte déchiffré :", plaintext.decode('utf-8', errors='ignore'))  # Afficher le texte déchiffré en UTF-8
```

STDIN

This is demo plain text
Mayuri_Shinde_62

Output:

```
Texte déchiffré : I want to believe
```