Android Security

LAB2

Mohamed BOUCHENGUOUR
Hamadi DAGHAR

MASTER 2 CYBERSECURITE
2024-2025

POLYTECH
NICE SOPHIA

Dans ce **lab2**, nous voulons afficher le **secret**, en modifiant directement le code source de l'application. En ouvrant l'application avec **jadx**, on analyse le code et on voit que le **secret** est au bout d'un moment dans une variable dans la fonction **a.a** du package **uncrackable1**. Cette fonction est appelée lors du clic sur **verify**, et vérifie si la chaîne saisie (**str**) est égale à **bArr** (le secret déchiffré). L'objectif est d'ajouter une instruction pour afficher le contenu de **bArr** avant que la fonction ne retourne le résultat.

```
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));
}
```

PRINT DE bArr (System.*out*.println(new String(bArr));

Dans **Android Studio**, nous créons une méthode qui affiche le contenu d'un tableau de **bytes** (**bArr**) en le convertissant en chaîne grâce à new String(bArr). Cette méthode nous permettra ensuite de récupérer le **code Smali** correspondant, que nous pourrons insérer manuellement dans l'application.

```
public class Dummy {
    no usages
    public static void printSecret(byte[] bArr) {
        System.out.println(new String(bArr));
    }
}
```

On génère l'**APK**, le désassemblons en **Smali**, puis récupérons le code de la fonction **printSecret** de la classe **Dummy**.

```
.method public static printSecret([B)V
    .locals 2
    .param p0, "bArr"    # [B

    .line 5
    sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;

    new-instance v1, Ljava/lang/String;

    invoke-direct {v1, p0}, Ljava/lang/String;-><init>([B)V

    invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V

    .line 6
    return-void
.end method
```

Ensuite, nous désassemblons l'**APK** de **app1**, récupérons la fonction **a** de la classe **a** du package **uncrackable1**, et ciblons le code après le bloc **try-catch**.

```
    :goto_0
    new-instance v1, Ljava/lang/String;

    invoke-direct {v1, v0}, Ljava/lang/String;-><init>([B)V

    invoke-virtual {p0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result p0

    return p0
.end method
```

Ici, nous remarquons que les deux codes ont ces deux lignes en commun :
new-instance v1, Ljava/lang/String;
invoke-direct {v1, v0}, Ljava/lang/String;-><init>([B)V

Ces deux lignes correspondent à new String(bArr) réalisé par les applications. Lors du **merge** du code **Smali** de **Dummy** dans celui de app1, il ne sera pas nécessaire de réinsérer ces lignes, car elles sont déjà présentes et réalisées par la méthode existante. La chaîne déchiffrée **new String(bArr)** sera donc stockée dans **v1**.

Il nous reste alors deux lignes spécifiques au code de Dummy :

sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
Cette ligne crée une instance pour la sortie standard **System.out** et l'assigne à **v0**.

invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
Cette ligne utilise l'instance de **System.out** (contenue dans **v0**) pour afficher la chaîne stockée dans **v1**.

Ces deux lignes seront insérées avant le **invoke-virtual {p0, v1}** qui effectue la comparaison entre les chaînes. Avant cela, nous vérifions que les noms de variables sont corrects et cohérents avec le code existant.

- **v0 :** Après avoir stocké le tableau de bytes déchiffré, **v0** n'est plus utilisé dans le reste de la méthode. Il peut donc être réassigné à **System.out** sans conflit.
- **v1 :** Contient déjà la **chaîne déchiffrée** (new String(bArr)).

Une fois ces vérifications effectuées, nous insérons les deux lignes issues de Dummy à l'emplacement identifié (avant le **invoke-virtual {p0, v1}**). Le code final de la méthode a après insertion sera le suivant :

```
    :goto_0
    new-instance v1, Ljava/lang/String;

    invoke-direct {v1, v0}, Ljava/lang/String;-><init>([B)V

    sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;

    invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V

    invoke-virtual {p0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result p0

    return p0
.end method
```

On compile l'**APK** et on vérifie sur **jadx** si l'**insertion** s'est réalisée correctement.

```
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;
    }
    String str2 = new String(bArr);
    System.out.println(str2);
    return str.equals(str2);
}
```

Ici, nous voyons que l'insertion a été correctement réalisée. La méthode stocke désormais **new String(bArr)** dans **str2**, l'affiche via **System.out.println(str2)**, puis effectue la comparaison avec **str.equals(str2)**.

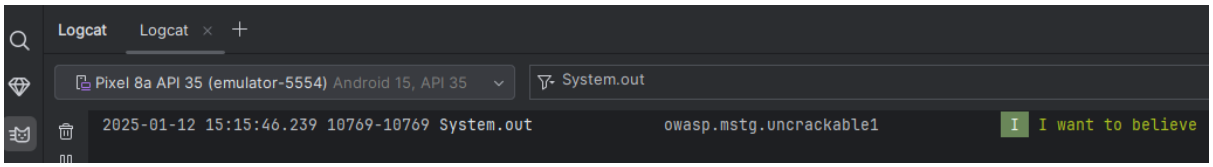
Maintenant que la ligne a été insérée, nous **signons** l'**APK** en générant une **clé de signature**, puis en l'utilisant pour **signer** l'**APK modifié**.

Nous installons **l'APK** sur **l'émulateur Android** et vérifions que l'installation est réussie.

```
PS C:\Users\Momol\AppData\Local\Android\Sdk\platform-tools> ./adb install -r "C:\Users\Momol\Documents\GitHub\Master2-Cybersecurite\Securite android\lab2\lab2\app1\dist\app1.apk"
Performing Streamed Install
Success
```

Ensuite, nous lançons l'application depuis l'interface de l'émulateur et cliquons sur **verify** pour déclencher la fonction **a.a**, ce qui permet d'afficher le secret grâce au **print** ajouté.

Pour accéder au **print**, nous utilisons **Logcat** dans Android Studio. En filtrant par **System.out**, nous pouvons voir le message imprimé par le code ajouté. Le secret, **"I want to believe"**, s'affiche dans la console, confirmant que la modification **Smali** fonctionne correctement et permet de récupérer le secret.



Dans ce **Lab2**, contrairement au **Lab1**, nous utilisons une **machine non rootée** sur l'émulateur. Cela est possible car nous modifions directement l'application via le code **Smali**, sans utiliser d'outils comme **Frida**. Ainsi, la **restriction liée au root** de l'application n'est pas un obstacle ici et n'a pas besoin d'être traitée.