Département Informatique

Sécurité des Applications Nomades

Modification d'un projet existant par rétro-engeneering

Zo RABARIJAONA
Jérémy MOROSI
Willy FRANÇOIS
Raya DJADLI

Année : 2013-2014

Table des matières

1 Introduction **2**

2 Répartition du travail **2**

3 Difficultés rencontrées **2**
3.1 Signatures 2
3.2 Code Natif 3
3.3 Un code obscur 3
3.3.1 Les variables 3
3.3.2 Les méthodes 3
3.3.3 Les classes 4

4 Conclusion **4**

1 Introduction

Dans le cadre de nos études, nous avions à réaliser un projet dans le domaine de la sécurité des applications nomades. Nous avons donc décidé de prendre une application existante afin de la modifier.

L'application que nous avions désigné était à la base **Castle Defense** (*com.hz.game.cd*) de **Elite Games**, mais, suite à un problème que nous développerons dans la suite du rapport, nous avons dû changer d'application et avons finalement choisi **Checkers** (*com.xxogli.xadroid.checkers*) de **User inc.**.

Cette application nous a demandé un gros travail d'analyse, mais nous avons fini par y apporter quelques modifications plus ou moins importantes. Nous avons commencé par retirer les publicités qui s'affichaient en bas de l'écran. Un bouton "Exit" permettant de quitter proprement l'application a été ajouté ainsi que le fait de faire commencer les noirs avec uniquement des dames. Un choix des positions des différents pions et dames a été ajouté lors de la validation d'une nouvelle partie. Et enfin, le fait de pouvoir inverser pions et dames quand on change de couleur.

2 Répartition du travail

Dans le cadre de ce projet, nous avons eu une grande part d'analyse du code afin de pouvoir le comprendre et enfin y apporter des modifications. Nous avons donc chacun apporté une pierre à l'édifice en désobscursissant le code java décompilé.

Zo s'est chargé de retirer la publicité et de désobscurcir une bonne partie du code Java. Willy s'est chargé du bouton "Exit", de faire commencer les noirs avec des dames, a réactivé le popup de démarrage et a désobscurci une bonne partie du code Java. Raya a désobscurci un peu de code et s'est occupée de l'inversion des pions et dames à changement de couleur. Jérémy a désobscurci du code Java et a ajouté le choix des positions des pions et dames à la validation d'une nouvelle partie.

3 Difficultés rencontrées

3.1 Signatures

À la base, nous devions modifier l'application **Castle Defense**. Le problème qui s'est rapidement posé est que l'application ne se lançait pas après signature avec une de nos clés. Ceci s'est expliqué par le fait que l'application possédait une constante permettant de vérifier qu'elle était bien signée par la clé du développeur, empêchant ainsi toute recompilation par une tierce personne.

Lors d'une séance de travaux pratiques, nous avons pris connaissance d'un moyen de passer outre cette vérification en utilisant une preuve de réalisation (proof of concept [1]) afin de faire passer notre application modifiée pour celle d'origine et ainsi pouvoir l'exécuter sans problème.

Par la suite, cette faille de sécurité a été corrigée sur la plupart des appareils mobiles, rendant impossible toute modification sur **Castle Defense**. D'un autre coté, le

PoC n'était pas nécessaire sur l'application **Checkers**, nous permettant donc de signer simplement l'application avec la clé de debug d'Android.

3.2 Code Natif

En décompilant les `.apk`, nous nous sommes rendu compte que, pour de nombreux jeux sur Android, la majorité du code est contenu dans une librairie écrite en `C/C++`, et que le code Java ne sert qu'à gérer la publicité et à faire le lien entre Android et les fonctions de la librairie native.

Bien qu'il soit possible de désassembler ces librairies, d'avoir les noms des classes et fonctions en clair, et de modifier les instructions en faisant des modifications hexadécimales, nous avons préféré chercher un jeu qui n'utilise pas de librairie car il est bien plus compliqué de modifier de l'assembleur.

3.3 Un code obscur

Après avoir effectué la décompilation de l'apk vers du code en hexadécimal, et ce dernier vers du code java, nous étions face à un code presque illisible car les variables, les noms des méthodes et les noms des classes étaient constitués avec des lettres de l'alphabet. Ce qui ne nous a pas beaucoup aidé pour la compréhension du code. Seule la classe principale *Checkers* est inchangée. Il nous a fallu donc étudier minutieusement chaque variable, méthode, classe et leurs relations pour pouvoir décrypter les noms.

3.3.1 Les variables

```
...
  private int q;
  private int r;
  private int S[];
  private int T[];
...
```

3.3.2 Les méthodes

```
...
  private final int a(boolean f){...}
  private final int a(boolean l, int a,int b){...}
  private final int a(boolean g1, int a1, int a2){...}
  private final int a(boolean j, int a){...}
...
```

3.3.3 Les classes

```
public class Checkers extends Activity {
private b view;
...
}
public class b extends View {...} //classe CheckersView

public class a extends {...} //classe de la ControleurView
```

4 Conclusion

Ce projet avait un coté ludique et éducatif. En effet, il nous a permis d'être très curieux en nous faisant fouiller dans les différentes méthodes et classes afin de comprendre le fonctionnement de l'application et de décrypter le code obscurci. Nous avons aussi pu apprendre le langage `smali` au cours de ce projet grâce aux différentes modifications apportées et aux traductions de méthodes de `smali` à `Java`.

Finalement, nous avons découvert que la sécurité faisait partie intégrante des applications d'aujourd'hui. Elle peut passer par un simple obscurcissement de code ou par l'utilisation de libraries écrites en `C/C++` natif, sans oublier une protection solide contre la signature par une personne mal intentionnée.

Références

[1] pof. Quick & dirty poc for android bug 8219321 discovered by blueboxsec. https://gist.github.com/poliva/36b0795ab79ad6f14fd8, 2013.