Sécurité des applications Nomades
Vérificateur de bytecode

L'objectif de ce projet est de concevoir un vérificateur de bytecode Dalvik à l'aide de l'outil `Androguard`. La liste des instructions à traiter est celle donnée par la documentation

https://source.android.com/devices/tech/dalvik/dalvik-bytecode.

Description de l'outil L'outil permettra de réaliser trois types d'analyses.

1. vérification de bytecode simple : cette analyse reprend l'état abstrait vu en cours (types des registres). Il s'agit de vérifier qu'à tout moment les instructions exécutées sont cohérentes avec le type des valeurs contenues dans les registres. N'oubliez pas d'inclure la vérification de la validité des numéros de registres utilisés par les instructions.

2. bonne initialisation des objets : cette analyse permet de vérifier qu'il n'y a aucun accès (lecture/écriture d'un champ, appel de méthode) à un objet alloué mais non initialisé.

3. extraction des communications : cette analyse permet d'extraire à partir d'une liste de classes à analyser (liste donnée dans un fichier) et des informations du paquet de déterminer

   - les permissions demandées par l'application
   - les permission déclarées par l'application
   - les composants de l'application pouvant répondre à une sollicitation interne ou externe

- les demandes de communication réalisée au travers d'intents, dans ce cas vous chercherez à obtenir un maximum d'informations concernant la cible (implicite, explicite, nom de l'action ou de la classe, données transmises)
- tout autre point qui vous semblerait pertinent

L'analyse sera intra-procédural, c'est à dire que l'on ne cherchera pas à reconstruire l'information contenu dans l'intent lorsque celui est manipulé au travers d'appels de méthodes.

Une attention toute particulière devra être portée à l'architecture. Une mauvaise architecture ne permettra pas de tirer profit de la similarité des points 1 et 2. Ce point, ainsi que la documentation des sources, seront pris en compte dans la notation. Idéalement, le moteur d'analyse (qui calcule l'information pour chaque point de programme) devrait être indépendant du domaine abstrait utilisé (le type des états abstraits et les fonctions de transformations des états associées aux différentes instructions).

Fonctionnement de l'outil Votre outil prendra en entrée un paquet `apk`, le nom `C` d'une classe à analyser et un flag indiquant le type d'analyse à réaliser. Si l'analyse valide le code, l'outil produira en sortie un fichier `C.report` contenant pour chaque méthode

- le résumé de la méthode : nom, signature, information sur les registres utilisés
- pour chaque position (offset) de la méthode, l'instruction et l'état abstrait calculé par l'analyse pour cette position

En cas d'échec, l'outil produira un message d'erreur expliquant le plus clairement possible l'origine de l'erreur. Il peut s'agir

- de l'impossibilité d'exécuter une instruction (calcul des out en fonction des in)
- de l'impossibilité de fusionner les informations de deux chemins (calcul des in en fonction des out)

Rendu Le rendu sera fait dans la page web du cours sous forme de fichier zip. Il devra contenir

- un rapport décrivant votre outil et son architecture
- le code de votre outil
- pour les analyses 1 et 2 un apk de test contenant plusieurs classes correctes
- pour les analyses 1 et 2 un apk de test contenant plusieurs classes incorrectes (à construire avec Smali car le compilateur ne générera, à priori, que des classes correctes)
- pour l'analyse 3 un apk permettant de tester de manière pertinente votre analyse
- un fichier Readme.ml détaillant
  - les étapes permettant d'utiliser l'outil
  - les classes des APK pouvant être utilisées pour le test avec une description du problème pour les classes incorrectes

Attention : l'outil doit être fonctionnel pour être évalué. Il vaut mieux un outil qui ne traite pas toutes les instructions mais qui fonctionne sur les autres. Dans ce cas le rapport devra mentionner les instructions non traitées.