Professeur : Amir Hajjam El Hassani

LES OBFUSCATEURS EN JAVA

Présenté par:

AZOUMA Roseline Megane & ATIVON Kevin

LES OBFUSCATEURS EN JAVA

Table des matières

Rédigé par AZOUMA Megane & ATIVON Kevin

Introduction

Les logiciels basés sur le langage Java ont acquis une immense popularité grâce à leur portabilité, leur flexibilité et leur sécurité inhérente. Cependant, la nature même du code source Java facilite le reverse-engineering, ce qui expose les applications à des risques tels que la violation des droits d'auteur, la contrefaçon et l'exploitation des vulnérabilités. C'est là que les obfuscateurs Java entrent en jeu. Les obfuscateurs sont des outils puissants qui visent à rendre le code source Java illisible tout en préservant son fonctionnement. Ils offrent une couche de protection supplémentaire en rendant extrêmement difficile la compréhension et la reconstruction du code source original. Ce cours sur les obfuscateurs Java vise à fournir une compréhension approfondie des principes fondamentaux de l'obfuscation, des techniques utilisées et de l'importance de sécuriser le code source Java. Nous explorerons les différentes fonctionnalités offertes par les obfuscateurs et la manière dont ils peuvent être intégrés dans le processus de développement logiciel pour renforcer la sécurité des applications Java.

PARTIE I
Compréhension des obfuscateurs

I. Compréhension des obfuscateurs

A. Objectifs des obfuscateurs

Les principaux objectifs des obfuscateurs en Java sont les suivants :

- **Sécurité du code :** L'un des objectifs clés de l'obfuscation est de renforcer la sécurité du code source en rendant plus difficile pour les pirates de comprendre et d'analyser le fonctionnement interne de l'application. Cela rend plus ardu l'identification des vulnérabilités, la modification non autorisée du code ou le vol de propriété intellectuelle.

- **Protection contre l'ingénierie inverse** : L'obfuscation rend la tâche des pirates qui tentent de décompiler le bytecode Java et de reconstituer le code source d'origine plus complexe. En renommant les noms de classe, de méthode et de variable de manière aléatoire, en supprimant les informations de débogage et en réorganisant la structure du code, les obfuscateurs rendent le processus d'ingénierie inverse plus difficile et plus chronophage.

- **Réduction de la taille du code :** Certains obfuscateurs utilisent des techniques de compression et d'élimination de code inutile pour réduire la taille du fichier JAR ou du bytecode. Cela peut être bénéfique pour la distribution de l'application, en réduisant le temps de téléchargement et l'espace de stockage requis.

- **Prévention du vol de propriété intellectuelle :** L'obfuscation rend le code source difficile à comprendre et à réutiliser sans autorisation. Cela aide à dissuader le vol de propriété intellectuelle en rendant l'application moins attrayante pour ceux qui cherchent à copier ou à réutiliser du code sans permission.

- **Protection des secrets et des clés de sécurité :** Les obfuscateurs peuvent également protéger les informations sensibles, telles que les clés de chiffrement ou les jetons d'authentification, en les rendant moins évidentes dans le code. Cela renforce la sécurité de l'application en limitant l'exposition des secrets.

Il est important de noter que bien que les obfuscateurs puissent rendre le code plus difficile à comprendre, ils ne fournissent pas une sécurité absolue. Les attaquants déterminés et expérimentés peuvent toujours contourner les mesures d'obfuscation. Par conséquent, l'obfuscation est souvent utilisée en combinaison avec d'autres techniques de sécurité, telles que le chiffrement des données ou

l'implémentation de contrôles de sécurité appropriés dans l'application elle-même.

B. Différences entre obfuscation et chiffrement

L'obfuscation et le chiffrement sont deux techniques de sécurité utilisées dans le domaine de l'informatique, mais elles diffèrent dans leur objectif et leur méthode.

1. Objectif

- L'obfuscation vise à rendre le code source ou le bytecode d'une application difficile à comprendre pour les tiers. Son objectif principal est de compliquer l'analyse et la compréhension du fonctionnement interne de l'application, notamment dans le but de protéger la propriété intellectuelle, de décourager l'ingénierie inverse et de renforcer la sécurité globale de l'application.
- Le chiffrement vise à protéger les données en les rendant inintelligibles pour les personnes non autorisées. Son objectif principal est de garantir la confidentialité des informations en utilisant des algorithmes mathématiques pour convertir les données en un format illisible, qui ne peut être déchiffré que par une partie autorisée possédant la clé appropriée.

2. Méthode

- L'obfuscation modifie la structure, le format et la logique du code source ou du bytecode de manière à rendre son analyse plus difficile. Les obfuscateurs renomment les noms de classe, de méthode et de variable de manière aléatoire, réorganisent la structure du code, ajoutent du code inutile ou redondant, suppriment les informations de débogage, etc. Ces transformations altèrent la lisibilité du code sans affecter son comportement fonctionnel.
- Le chiffrement utilise des algorithmes de chiffrement pour transformer les données en un format illisible, appelé texte chiffré, en utilisant une clé spécifique. Le processus de chiffrement utilise des opérations mathématiques complexes pour masquer le contenu des données. Pour accéder aux données d'origine, il est nécessaire d'utiliser la clé de déchiffrement appropriée pour inverser le processus et restaurer les données sous leur forme d'origine, appelée texte en clair.

3. Domaines d'application

- L'obfuscation est principalement utilisée pour protéger le code source des applications, en particulier dans les environnements où le code est distribué (par exemple, les applications mobiles, les logiciels commerciaux). Son utilisation est courante dans le développement d'applications Java, Android et .NET.
- Le chiffrement est utilisé pour sécuriser les données sensibles lors de leur stockage, de leur transmission ou de leur manipulation. Il est appliqué à différents niveaux, tels que le chiffrement des disques, le chiffrement des communications réseau, le chiffrement des bases de données et le chiffrement des fichiers.

En résumé, l'obfuscation vise à rendre le code source d'une application plus difficile à comprendre, tandis que le chiffrement vise à rendre les données illisibles pour les personnes non autorisées. L'obfuscation se concentre sur la protection du code et de la logique de l'application, tandis que le chiffrement se concentre sur la confidentialité des données.

C. Principaux scénarios d'utilisation des obfuscateurs en Java

Les obfuscateurs en Java sont utilisés dans différents scénarios pour protéger le code source des applications et renforcer la sécurité. Voici les principaux scénarios d'utilisation des obfuscateurs en Java :

- Protection des applications commerciales : Lorsque des applications Java sont développées pour des besoins commerciaux, les obfuscateurs sont utilisés pour protéger le code source contre l'ingénierie inverse. Cela empêche les tiers de comprendre le fonctionnement interne de l'application, de voler la propriété intellectuelle ou de créer des versions piratées.

- Sécurité des applications mobiles : Les applications mobiles Java, notamment pour les plateformes Android, sont exposées à des risques de piratage et d'exploitation. Les obfuscateurs sont utilisés pour rendre le code source plus difficile à comprendre, à décompiler et à manipuler, ce qui réduit les risques de vol de données sensibles ou de contournement de la sécurité de l'application.

- Protection des logiciels embarqués : Dans les domaines de l'embarqué et de l'IoT, les obfuscateurs sont utilisés pour protéger le code source des dispositifs et systèmes embarqués Java. Cela vise à éviter les atteintes à la

sécurité, le vol de propriété intellectuelle ou la modification non autorisée du firmware.

- Sécurisation des algorithmes et des clés de chiffrement : Les obfuscateurs sont utilisés pour masquer les algorithmes de chiffrement et les clés sensibles utilisées dans les applications Java. Cela rend plus difficile pour les attaquants de comprendre et d'exploiter les mécanismes de sécurité utilisés, tels que les fonctions de hachage, les algorithmes de cryptographie symétrique ou asymétrique, et les clés secrètes.

- Protection des applications open source : Même si le code source des applications open source est disponible publiquement, les obfuscateurs sont utilisés pour rendre le code plus complexe à analyser et à modifier. Cela peut aider à décourager les attaques ciblées, les tentatives de vol de propriété intellectuelle et les abus de l'application.

- Respect des normes de conformité : Dans certains secteurs réglementés, tels que les services financiers ou la santé, les obfuscateurs sont utilisés pour répondre aux exigences de conformité en protégeant le code source des applications contre les menaces de sécurité et les violations de confidentialité des données.

PARTIE II
Techniques d'obfuscation

II.Techniques d'obfuscation

A. Renommage des noms de classe, de méthode et de variable :

L'une des techniques couramment utilisées par les obfuscateurs en Java est le renommage des noms de classe, de méthode et de variable. Cette technique vise à rendre le code source moins lisible en modifiant les noms de manière aléatoire ou en utilisant des noms générés de manière cryptique. Voici comment fonctionne le renommage :

Original Source Code Before Rename Obfuscation	Reverse-Engineered Source Code After Rename Obfusaction
private boolean areRelativelPrime( ) {if (get(term1, term2) ... 1){ return true; } else { return false; } } }	private boolean c() { if (b(a, f) .... 1) { return true; } else { return false; } } }

1. Noms de classe :

- Les obfuscateurs renomment les classes en leur attribuant des noms aléatoires ou générés de manière cryptique. Par exemple, une classe initialement nommée "MyClass" peut être renommée en "a1b2c3".
- Les références aux classes renommées sont mises à jour dans tout le code source pour garantir que les appels aux classes renommées sont correctement résolus.

2. Noms de méthode :

- Les obfuscateurs renomment également les méthodes en leur attribuant des noms aléatoires ou générés de manière cryptique. Par exemple, une méthode initialement nommée "calculateSum" peut être renommée en "x9y8z7".
- Les appels aux méthodes renommées sont ajustés dans tout le code source pour refléter les nouveaux noms de méthode.

3. Noms de variable :

- Les variables locales et les champs de classe peuvent également être renommés par les obfuscateurs. Les noms des variables peuvent être modifiés en utilisant des noms aléatoires ou générés de manière cryptique, tout comme les noms de classe et de méthode.
- Les références aux variables renommées sont mises à jour dans le code source pour s'aligner sur les nouveaux noms de variable.

L'objectif principal du renommage est de rendre le code source plus difficile à comprendre en rendant les noms de classe, de méthode et de variable moins explicites. Cela complique l'analyse du code par des tiers, car la signification des noms devient moins évidente. Le renommage n'affecte pas le fonctionnement de l'application, car les liens entre les différentes parties du code sont ajustés pour refléter les nouveaux noms.

Cependant, il est important de noter que le renommage seul ne garantit pas une sécurité absolue. Les méthodes d'analyse statique et dynamique peuvent encore être utilisées pour comprendre le comportement de l'application. Le renommage est souvent utilisé en combinaison avec d'autres techniques d'obfuscation pour renforcer la sécurité du code source.

B. Suppression des informations de débogage

La suppression des informations de débogage est une autre technique utilisée par les obfuscateurs en Java pour renforcer la sécurité du code source. Les informations de débogage fournissent des détails sur la structure interne de l'application et peuvent être utiles aux développeurs pour le débogage et l'analyse du code. Cependant, elles peuvent également être exploitées par des tiers malveillants pour comprendre le fonctionnement de l'application et découvrir d'éventuelles vulnérabilités.

1. Informations de débogage incluses dans le bytecode

- Le bytecode généré par le compilateur Java contient souvent des informations de débogage, telles que les noms de variables, les numéros de ligne, les fichiers sources associés, etc.
- Les obfuscateurs suppriment ces informations de débogage du bytecode, les rendant ainsi indisponibles pour les tiers qui pourraient analyser le code.

2. Suppression des fichiers de symboles

- Les obfuscateurs peuvent également supprimer les fichiers de symboles (par exemple, les fichiers JAR, les fichiers de débogage associés) qui contiennent des informations de débogage supplémentaires.
- Cela empêche l'accès direct aux informations de débogage par des outils de décompilation ou d'analyse du code.

3. Anonymisation des exceptions

- Les obfuscateurs peuvent anonymiser les exceptions en remplaçant les noms d'exception par des noms génériques ou cryptiques. Cela rend plus difficile l'identification des erreurs spécifiques et limite les informations potentiellement exploitables disponibles pour les attaquants.

La suppression des informations de débogage rend le code source plus opaque, car les indices qui facilitent la compréhension et l'analyse du code sont supprimés. Cela rend l'ingénierie inverse plus difficile et décourage les attaquants potentiels.

Cependant, il est important de noter que la suppression des informations de débogage peut rendre le processus de débogage et de maintenance plus complexe pour les développeurs. Par conséquent, il est courant de garder une copie du code non obfusqué avec les informations de débogage intactes pour faciliter le débogage lors du développement et du test de l'application.

C. Transformation de la structure du code

La transformation de la structure du code est une autre technique utilisée par les obfuscateurs en Java pour compliquer davantage l'analyse et la compréhension du code source. Elle vise à altérer la structure logique du code sans en affecter le fonctionnement. Voici comment fonctionne la transformation de la structure du code :

Original Source Code To Swap Two Variables	Machine Level Instructions Created by the Complier	Machine Level Instructions After Transient Variable Caching	Original Source Code To Swap Two Variables ?

temp=a;
b=temp;
a=b;
	iload_1
	istore_3
	iload_2
	istore_1
	iload_3
	istore_2
		iload_1
		iload_2
		istore_1
		istore_2
			There is no equivalent high level source

1. Réarrangement des instructions

- Les obfuscateurs peuvent réorganiser l'ordre des instructions dans les méthodes pour modifier la séquence logique du code.
- Cela rend plus difficile la compréhension du flux d'exécution du programme, car les instructions ne suivent plus une logique linéaire et prévisible.

2. Désassemblage et réassemblage du code

- Certains obfuscateurs peuvent désassembler le bytecode Java en code intermédiaire, puis le réassembler en un nouveau bytecode.
- Cette technique peut modifier la structure du code, notamment en réorganisant les blocs de code, en remplaçant les instructions par des équivalents, etc.

3. Ajout de code inutile ou redondant

- Les obfuscateurs peuvent insérer du code inutile ou redondant dans le code source.
- Cela rend le code plus complexe et augmente le bruit dans le code, rendant plus difficile la compréhension de la logique réelle.

4. Transformation de la représentation

- Les obfuscateurs peuvent convertir le code source d'une représentation lisible en une représentation plus cryptique.
- Par exemple, les noms de variables et de méthodes peuvent être transformés en utilisant des caractères non alphanumériques, des valeurs ASCII ou des valeurs hexadécimales.

La transformation de la structure du code rend l'analyse et la compréhension du code plus difficiles en introduisant des modifications qui ne modifient pas la logique fonctionnelle de l'application. Cela complique l'ingénierie inverse et décourage les tentatives de vol de propriété intellectuelle.

Cependant, il est important de noter que la transformation de la structure du code peut rendre le processus de développement, de maintenance et de débogage plus complexe pour les développeurs. Il est essentiel de s'assurer que les transformations ne perturbent pas le comportement de l'application et ne génèrent pas de bogues indésirables.

D. Ajout de code inutile ou redondant

L'ajout de code inutile ou redondant est une technique utilisée par certains obfuscateurs en Java pour rendre le code source plus difficile à comprendre et à analyser. Cette technique consiste à introduire du code supplémentaire qui n'a pas d'impact sur le fonctionnement de l'application, mais qui augmente la complexité du code et rend plus ardues les tentatives de compréhension. Voici comment fonctionne l'ajout de code inutile ou redondant :

- Ajout de méthodes ou de blocs de code inutiles :
  - Les obfuscateurs peuvent insérer des méthodes supplémentaires ou des blocs de code qui ne sont pas appelés dans le flux d'exécution normal de l'application.
  - Ces méthodes ou blocs de code ne contribuent pas à la logique fonctionnelle de l'application, mais rendent plus difficile la distinction entre le code réellement utilisé et le code superflu.

- Introduction de fausses dépendances :
  - Les obfuscateurs peuvent ajouter des dépendances factices entre les classes ou les packages de l'application.
  - Cela implique d'ajouter des références ou des appels de méthodes entre des parties du code qui n'ont pas réellement besoin de communiquer, créant ainsi une complexité supplémentaire.

- Insertion de variables redondantes :
  - Les obfuscateurs peuvent introduire des variables supplémentaires qui ne sont pas utilisées dans le code.
  - Ces variables peuvent avoir des noms cryptiques ou générés aléatoirement, contribuant à la confusion lors de l'analyse du code.

L'ajout de code inutile ou redondant rend le code source plus complexe et moins lisible pour les tiers. Cela rend l'analyse et la compréhension du code plus difficiles, car les développeurs doivent trier le code réellement utilisé du code superflu. Cela rend également plus difficile l'identification des parties critiques du code, ce qui rend plus ardues les tentatives de modification ou de compréhension de la logique de l'application.

Cependant, il est important de noter que l'ajout de code inutile ou redondant peut augmenter la taille du code et avoir un impact sur les performances de l'application, notamment en termes de temps de chargement et d'utilisation de la mémoire. Il est donc essentiel de trouver un équilibre entre l'obfuscation du code et l'impact sur les performances.

E. Compression du code

La compression du code est une technique utilisée par certains obfuscateurs en Java pour réduire la taille du code source ou du bytecode de l'application. Elle vise à compresser le code sans en altérer la logique ou le fonctionnement. Voici comment fonctionne la compression du code :

- Compression des fichiers JAR ou des archives de classe :
  - Les obfuscateurs peuvent compresser les fichiers JAR ou les archives de classe en utilisant des algorithmes de compression tels que ZIP ou GZIP.
  - Cette compression réduit la taille du fichier exécutable ou du bytecode, ce qui peut faciliter le déploiement de l'application, réduire le temps de téléchargement et économiser de l'espace de stockage.

- Compression des chaînes de caractères :
  - Les obfuscateurs peuvent compresser les chaînes de caractères dans le code source en utilisant des algorithmes de compression spécifiques.
  - Au lieu de stocker les chaînes de caractères sous leur forme lisible, elles sont compressées en une représentation plus compacte et décompressées lors de leur utilisation dans l'application.

- Compression du bytecode ou du code intermédiaire :
  - Certains obfuscateurs peuvent compresser directement le bytecode Java ou le code intermédiaire généré lors de l'obfuscation.
  - Cela réduit la taille du code, tout en veillant à ce que le code puisse être décompressé et exécuté normalement lors de l'exécution de l'application.

La compression du code permet de réduire la taille du code source ou du bytecode sans altérer son fonctionnement. Cela peut être bénéfique pour le déploiement de l'application, notamment lorsqu'il s'agit de distribuer l'application sur des réseaux à bande passante limitée ou de la stocker sur des dispositifs avec un espace de stockage limité.

Il convient de noter que bien que la compression du code puisse réduire la taille du fichier, elle peut également entraîner une légère surcharge en termes de temps de décompression lors de l'exécution de l'application. Il est donc important de trouver un équilibre entre la réduction de la taille et l'impact sur les performances lors de l'utilisation de la compression du code.

F. Outils d'obfuscation populaires

On peut donc citer quelques outils tels que : ProGuard, Allatori, DexGuard, Zelix KlassMaster, vGuard

PARTIE III

Méthodologie de mise en œuvre des obfuscateurs en JAVA

III. Méthodologie de mise en œuvre des obfuscateurs en Java

La mise en œuvre des obfuscateurs en Java suit généralement une méthodologie qui implique plusieurs étapes.

A. Étapes de préparation avant l'obfuscation

- Analyse des besoins de sécurité : Identifiez les exigences spécifiques en matière de sécurité de l'application, telles que la protection contre l'ingénierie inverse, la sécurisation des clés de chiffrement, etc.

- Évaluation des obfuscateurs : Effectuez une recherche et une évaluation des obfuscateurs disponibles pour déterminer lequel répond le mieux à vos besoins en termes de fonctionnalités, de compatibilité et de performances.

B. Configuration des options d'obfuscation

- Sélection des transformations à appliquer : Choisissez les techniques d'obfuscation spécifiques que vous souhaitez utiliser, telles que le renommage des noms, la suppression des informations de débogage, etc.

- Configuration des paramètres d'obfuscation : Définissez les options d'obfuscation spécifiques fournies par l'outil, telles que les règles de renommage, les exclusions de classes ou de packages, les niveaux de compression, etc.

C. Traitement des exceptions et des bibliothèques tierces

- Gestion des exceptions : Certains obfuscateurs peuvent rencontrer des problèmes avec certaines constructions du code ou des bibliothèques tierces. Identifiez ces problèmes et configurez les exceptions appropriées pour éviter toute altération indésirable du code.

- Traitement des bibliothèques tierces : Assurez-vous de comprendre l'impact de l'obfuscation sur les bibliothèques tierces utilisées dans votre application. Certaines bibliothèques peuvent nécessiter des configurations spécifiques pour garantir leur fonctionnement correct après l'obfuscation.

D. Validation du code obfusqué

- Vérification de la fonctionnalité : Effectuez des tests approfondis sur l'application obfusquée pour vous assurer que toutes les fonctionnalités sont préservées et qu'aucun bogue majeur n'a été introduit pendant le processus d'obfuscation.

- Analyse de la performance : Évaluez les performances de l'application obfusquée pour détecter tout impact négatif sur les temps de démarrage, l'utilisation de la mémoire, la vitesse d'exécution, etc.

E. Tests de régression et débogage post-obfuscation

- Effectuez des tests de régression approfondis pour détecter les problèmes éventuels qui pourraient survenir après l'obfuscation.

- Si des problèmes sont identifiés, revenez en arrière pour examiner les configurations d'obfuscation, les exceptions ou les exclusions afin de résoudre les problèmes rencontrés.

En suivant une méthodologie structurée, vous pouvez mettre en œuvre efficacement les obfuscateurs en Java et renforcer la sécurité de votre application.

PARTIE IV

Avantages et considérations liés à l'obfuscation en JAVA

IV.Avantages et considérations liés à l'obfuscation en Java

L'obfuscation en Java présente plusieurs avantages en termes de sécurité et de protection du code source. Cependant, il est important de prendre en compte certaines considérations lors de l'utilisation de techniques d'obfuscation.

A. Avantages de l'obfuscation en Java

- **Protection contre l'ingénierie inverse**
- **Réduction de la taille du code**
- **Prévention du vol de propriété intellectuelle**

B. Considérations liées à l'obfuscation en Java

- **Limitations et risques potentiels de l'obfuscation** : Il est donc essentiel de mettre en place d'autres couches de sécurité et de prendre en compte d'autres mesures de protection.

- **Impact sur les performances de l'application**

- **La maintenance plus difficile**

- **Interopérabilité avec les bibliothèques tierces** : L'obfuscation peut entraîner des problèmes d'interopérabilité avec les bibliothèques tierces utilisées dans l'application. Certaines bibliothèques peuvent nécessiter des configurations spécifiques ou peuvent ne pas fonctionner correctement après l'obfuscation. Il est important de tester et de vérifier la compatibilité avec les bibliothèques tierces avant de déployer l'application obfusquée.

PARTIE V

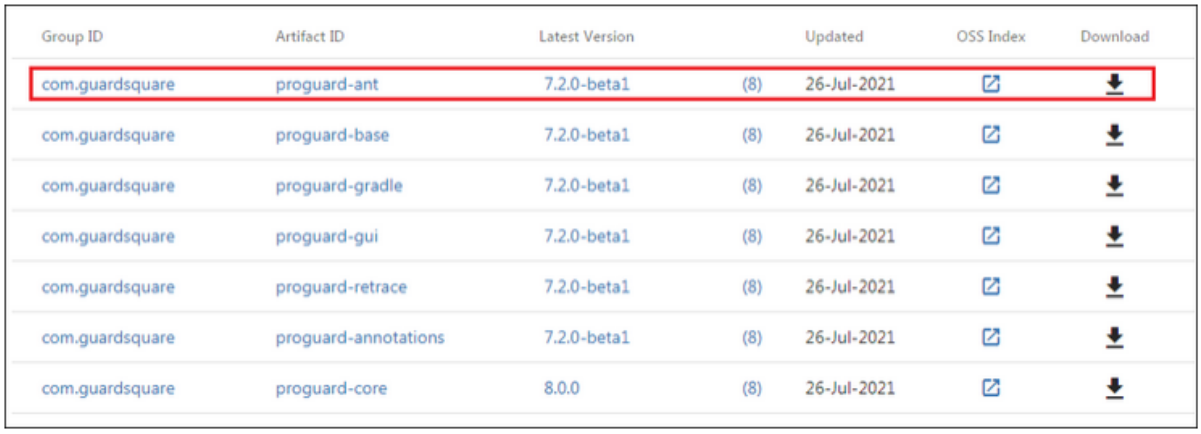
MISE EN PRATIQUE DE L'OBFUSCATION SUR UNE APPLICATION JAVA AVEC PROGUARD

V. Mise en pratique de l'obfuscation sur une application Java avec ProGuard

ProGuard est un raccourcisseur, optimiseur, obfuscateur et prévérificateur de fichiers de classe Java gratuit. Il optimise le bytecode et renomme les classes, les fichiers, les méthodes et les variables en utilisant le nom court sans signification. En outre, il optimise également le code et détecte les classes, méthodes, attributs et instructions inutilisés. Il convient également pour Android et Kotlin.

Pour utiliser l'outil ProGuard, nous devons d'abord le télécharger . Après cela, suivez les étapes.

Étape 1 : Téléchargez le fichier **proguard-ant-7.2.0-beta1.jar** .



Group ID	Artifact ID	Latest Version		Updated	OSS Index	Download
com.guardsquare	proguard-ant	7.2.0-beta1	(8)	26-Jul-2021		
com.guardsquare	proguard-base	7.2.0-beta1	(8)	26-Jul-2021		
com.guardsquare	proguard-gradle	7.2.0-beta1	(8)	26-Jul-2021		
com.guardsquare	proguard-gui	7.2.0-beta1	(8)	26-Jul-2021		
com.guardsquare	proguard-retrace	7.2.0-beta1	(8)	26-Jul-2021		
com.guardsquare	proguard-annotations	7.2.0-beta1	(8)	26-Jul-2021		
com.guardsquare	proguard-core	8.0.0	(8)	26-Jul-2021		

Étape 2 : Créez un fichier de configuration contenant toutes les informations sur l'application. Dans notre cas, nous avons créé le fichier **democonfig.pro** .

-injars : spécifiez le chemin des fichiers .jar (application Java compilée).

-outjars : Il s'agit d'un fichier jar qui est créé par le ProGuard, après obfuscation. Il contient toutes les conventions de nommage confuses et obscures des méthodes et des variables dans le fichier de classe.

-printmapping : Le fichier contient toutes les informations de mappage pour notre référence.

-keep : dans ce fichier, nous spécifions les fichiers de classe et les méthodes que nous ne voulons pas masquer. Par exemple, **com.javatpoint.DemoClass**

contient le point d'entrée de l'application avec la classe principale qui ne sera pas masquée dans l'exemple suivant.

democonfig.pro

```
$ cat democonfig.pro
-injars /home/jsmith/demoapp.jar
-outjars /home/jsmith/demoapp-obfuscated.jar This is the obfuscated jar file
-libraryjars /usr/java/jdk1.5.0_14/jre/lib/rt.jar
-printmapping proguard.map
-verbose
-keep public class com.javatpoint.DemoClass
```

Maintenant, nous allons exécuter l'application ProGuard en utilisant les commandes suivantes.

```
$ cd /home/jsmith/proguard4. 2 /lib
$ java -jar proguard.jar  @democonfig .pro
```

Après avoir exécuté ProGuard, il crée les deux fichiers suivants :

- **myapp-obfuscated.jar :** il contient les fichiers de classe obfusqués de l'application. Nous pouvons partager ce fichier sans nous soucier de la rétro-ingénierie de notre application.
- **map :** Il contient les informations de mappage pour notre référence.

Voyons les fichiers de démonstration.

proguard.map

Le fichier affiche le nom d'origine des objets source Java et le nouveau nom masqué correspondant. Par exemple, considérez ce qui suit :

```
demoapp.ProExample -> ApplicationDemo.class:
areRelativePrime() -> c()
get() -> b
term1 -> a
term2 -> f
```

Demo.java (avant obfuscation)

```
HashSet<String> obj=new HashSet(list);
set.add("Gaurav");
Iterator<String> i=set.iterator();
while(i.hasNext())
{
System.out.println(i.next());
}
```

Nous obtenons le code source Java suivant qui a été décompilé à partir du fichier de classe après obfuscation.

```
A<m> o=new A(list);
o.q("#$%#^&&*!");
P<m> o=o.u();
while(s.k())
{
System.out.println(s.l());
}
```

Nous observons que l'instruction **HashSet<String> obj=new HashSet(list);** s'est transformé en l'instruction **A<m> o=new A(list);** Par conséquent, personne ne peut deviner à partir du code ci-dessus ce que contient l'application réelle.

Conclusion

Les obfuscateurs en Java jouent un rôle essentiel dans la protection des applications contre l'ingénierie inverse et la sécurisation du code source. Grâce à des techniques telles que le renommage, la suppression des informations de débogage et la transformation de la structure du code, les obfuscateurs rendent la tâche des pirates plus difficile. Cependant, il est important de comprendre les limitations et les risques associés à l'obfuscation, ainsi que l'impact potentiel sur les performances de l'application. Avec une mise en œuvre appropriée et une utilisation judicieuse des outils d'obfuscation, il est possible de renforcer considérablement la sécurité des applications Java.