Fiches techniques
Obfuscation de code

Louka Boivin

19 novembre 2021

1 Problématique

1.1 Introduction

Le but de ce projet est de réaliser un outil permettant de protéger des logiciels propriétaires contre des utilisations illégales et contre la copie. Pour cela, le but premier est de créer un outil de gestion et de protection des licences afin d'éviter les utilisations illégales du logiciel.

Dans un second temps, il faut donc protéger ce logiciel contre la copie et contre le reverse engineering. Le reverse engineering est le fait de dé-compiler ou désassembler un exécutable afin d'en retrouver le code source. Pour ce faire plusieurs méthodes peuvent être utilisées et combinées pour augmenter la résistance du code. Ces méthodes sont détaillées dans la section n°2.

1.2 Mots-clés

Propriété intellectuelle, obfuscation de code, software watermarking, code tamper-proofing.

2 Méthodes

Les principales méthodes utilisées sont les suivantes :
— Obfuscation de code
— Software watermarking
— Code tamper-proofing

2.1 Obfuscation de code

Étant donné que aucun code n'est sûre à 100% le but est de rendre le reverse-engineering plus coûteux en efforts et en temps que de produire un logiciel équivalent. Pour cela l'objectif est de rendre le code source flou, afin de le rendre difficilement lisible et compréhensible par l'humain.

Il y a différents niveaux d'obfuscation [2] :

2.1.1 Layout Obfuscation

La chose la plus simple et de modifier le code source pour le rendre obscure et incompréhensible à première vue. Par exemple :

- Renommer les variables (ex : changer `licenseFile` par `z6zdf50klp`),
- Retirer les commentaires,
- Changer l'agencement du code, retirer tabulations, sauts de ligne, espaces...

2.1.2 Control Obfuscation

Ensuite on peut modifier pour rendre plus flou l'exécution du programme en ajoutant différents éléments :

- Ajouter du code "mort" : du code jamais exécuté, mais présent pour complexifier la compréhension,
- Des instructions conditionnelles inutiles : toujours vrai, juste présent pour faire croire que le test est important,
- Ajouter des opérandes ou fonctions redondantes,
- Paralléliser le code à plusieurs endroits,
- Modifier l'étendue des boucles, itérer à l'envers etc...

2.1.3 Data Obfuscation

Une autre chose importante et de pouvoir cacher les données comprises dans notre code source. Par exemple, les chaînes de caractères restent telles quelles dans le code compilé et donc il est nécessaire de les rendre complexe à décoder.

Exemple : on peut chiffrer nos chaînes de caractères à l'aide d'un simple XOR.

On peut également agir sur les variables : modifier leur types pour quelque chose de plus flou, les scinder en deux variables complémentaires qui permettrait par l'intermédiaire de fonctions de les ré-assembler et les modifier etc...

2.1.4 Preventive Obfuscation

Enfin, une dernière méthode consiste à prévenir notre code contre un logiciel de désobfuscation précis, si on sait contre qui ou quoi notre logiciel doit être protégé.

2.1.5 Influence sur les performances

L'obfuscation de code ne doit pas avoir d'influence significative sur les performances du programme ou de l'application à l'exécution.

Pour évaluer l'efficacité de l'obfuscations et les performances plusieurs critères sont utilisés :

- Potentielle d'obfuscation,

— Résistance aux outils de désobfuscation automatique,
— L'intégration du code obfusqué au reste,
— Le coût.

2.2 Software watermarking

Le software-watermarking [1] ou "tatouage numérique [5]" en français, est une méthode qui permet d'ajouter des informations de copyright ou de vérification à un document numérique. Cela permet ensuite de vérifier la validité ou l'intégrité du code. Il existe plusieurs types de tatouages, visible, invisible et fragile.

Pour plus de détails, voir les références.

2.3 Code tamper-prooofing

Le code tamper-proofing [4] est un autre ensemble de méthode qui ont pour but de protéger les logiciels contre les utilisateurs illégaux ou malveillants.

Cela consiste par exemple, à chiffre le code source, ajouter des sommes de contrôles pour en vérifier l'intégrité etc...

Le tamper-proofing est généralement utilisé dans les schémas de licence pour empêcher l'attaquant de désactiver la vérification de celle-ci [3].

3 Exemples

— Outils d'obfuscation payants :
  — Stunnix (C/C++/JavaScript/Perl/VBS)
  — Themida (C++)
— Outils gratuits :
  — ConfuserEx.

Réferences externes

[1] Dr James HAMILTON. *What is software watermarking ?* URL : https://jameshamilton.eu/research/what-software-watermarking.

[2] Rajan PATEL et Nimisha PATEL. "A way to protect software secrets from reverse engineering using code obfuscation techniques". In : (2010). URL : https://www.researchgate.net/publication/269694315_A_WAY_TO_PROTECT_SOFTWARE_SECRETS_FROM_REVERSE_ENGINEERING_USING_CODE_OBFUSCATION_TECHNIQUES.

[3] *What is software tamper-proofing ?* URL : http://www.whitehawksoftware.com/white-hawk-technology/what-is-software-tamper-proofing/.

[4] Article WIKIPÉDIA. *Anti-tamper software*. URL : https://en.wikipedia.org/wiki/Anti-tamper_software.

[5] Article WIKIPÉDIA. *Tatouage numérique*. URL : https://fr.wikipedia.org/wiki/Tatouage_num\%C3\%A9rique.