Évaluation des vulnérabilités des logiciels - TP2 : obfuscation de programmes

Sandrine Blazy - ISTIC

13 octobre 2017

Les fichiers C sont à récupérer dans le répertoire `TP2`. Le compte-rendu du TP est à déposer sur moodle (`https://foad.univ-rennes1.fr/course/view.php?id=1006722`) avant ce soir 21h.

1 Renommage de variables

Modifier le programme `obf1.c` en renommant toutes ses variables. Que constatez-vous dans le code assembleur généré ?

2 Découpage de variables

1. Obfusquer le programme `obf2.c` en remplaçant toute occurence de variable par deux variables. Plus précisément, une variable `x` sera remplacée par deux variables `a` et `b` telles que `a = x /5` (i.e. division entière) et `b = x % 5` (i.e. reste de la division entière).
2. En quoi les codes assembleur générés pour les deux programmes (i.e. le programme initial et le programme obfusqué) sont-ils différents ?
3. Obfusquer à nouveau le programme précédemment obtenu en modifiant lorsque cela est possible l'ordre d'écriture des instructions.
4. Rajouter dans le programme un prédicat opaque introduisant du code mort dans le programme.

3 Encodage d'entiers

1. Obfusquer le programme `obf3.c` en encodant ses constantes entières de la façon suivante : chaque constante `c` devient `c*6`. Le programme obfusqué se comporte-t-il comme le programme initial ?

2. Comment faut-il modifier le programme obtenu à la question précédente pour qu'il soit équivalent au programme initial ?
3. Obfusquer le programme `obf4.c` en encodant ses constantes entières en les multipliant par 6, de façon à obtenir un programme équivalent. Cette obfuscation utilisera les fonctions `encode` et `decode` fournies dans `obf4.c`. Comment peut-on s'assurer que le programme obfusqué se comporte comme le programme initial ?
4. Reprendre la dernière question en encodant les constantes entières du programme de la façon suivante : chaque constante `c` devient `c^6`.

4 Expressions mixtes arithmético-booléennes

1. Écrire un programme C permettant de tester l'obfuscation par expressions mixtes arithmético-booléennes vue en cours, dans laquelle l'instruction `R = x ^ 92;` est obfusquée en la suite d'instructions suivantes.

```
a = 229*x + 247 ;
b = 237*a + 214 + ((38*a+85) & 254) ;
c = (b + ((-2*b + 255) & 254))*3 + 77 ;
d = ((86*c + 36) & 70) * 75 + 231*c + 118 ;
e = ((58*d + 175) & 244) + 99*d + 46 ;
f = e & 148 ;
g = (f-(e&255) + f)*103 + 13 ;
r = 237*(45*g + (174*g | 34)*229 + 194 - 247) & 255 ;
```

2. Compiler ce programme d'abord sans utiliser les optimisations de `gcc`, puis en utilisant le niveau maximal d'optimisation de `gcc`. Que constatez-vosu dans les codes assembleur générés ?

5 Applatissement de graphe de flot de contrôle

1. Dessiner le graphe de flot de contrôle du programme `perlin.c`.
2. Dessiner une partie du graphe de flot de contrôle du programme `perlin-obf.c`. Que remarquez-vous ?
3. Utiliser IDA pour reconstruire le graphe de flot de contrôle du programme `perlin-obf.c`.