MSE | MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Master of Science HES-SO in Engineering

Orientation : Technologies de l'information et de la communication

**CAHIER DES CHARGES**

# LEVEE, IMPLEMENTATION DE « CONTROL-FLOW INTEGRITY » AU SEIN DE LLVM

Fait par

Joël Gugger

Sous la direction de
Prof. Pascal Junod

Lieu, HES-SO//Master, 27 mars 2017

# Description du problème

Nos programmes sont le plus souvent écrits avec des langages bas niveaux tels le C/C++ qui forcent le développeur à gérer la mémoire lui-même. Ce qui implique que, sans de bonnes connaissances et une attention particulière, un adversaire peut facilement exploiter des bugs qui surviennent au sein de ces mécanismes de gestion. Grâce à cela, l'attaquant peut exécuter son propre code avec les privilèges du programme touché.

Sur les dix dernières années, les attaques de capables de modifier le flot de contrôle connues au sein des principaux logiciels que nous utilisons ont augmentées.

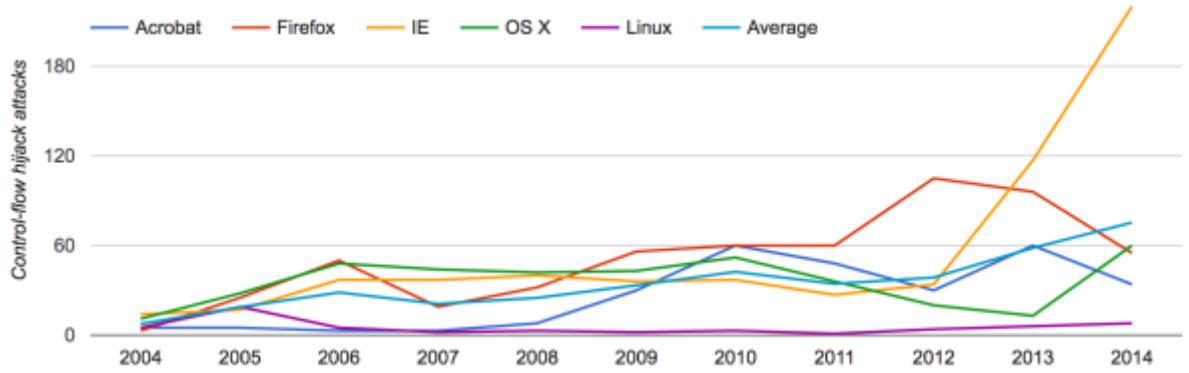


*Figure 1 http://dslab.epfl.ch/proj/cpi/*

Etant donné la dangerosité de ce type d'attaque connues depuis cinquante ans, 1998 pour le « grand public », différents concepts ont été mis en place. Parmi eux on peut retrouver ASLR, DEP, « Stack cookies », « Coarse-grained CFI » ou encore « Finest-grained CFI ».

Dans ce contexte, un laboratoire de l'EPFL propose une implémentation appelée Levee[1] qui rassemble deux concepts (**CPI**[2]/**CPS**[3]) de protection au sein de l'infrastructure de compilation LLVM. L'idée est de séparer les pointeurs jugés sensible et de les placer dans une zone mémoire sécurisée appelée **SafeStack**. La séparation des pointeurs est faite par analyse durant la phase de compilation et permet d'obtenir un « hoverhead » relativement bas (environ 8% à 10%).

Le but de ce projet d'approfondissement est de comprendre et d'expliquer plus en détail le fonctionnement des deux concepts de protection ainsi que d'expérimenter et d'analyser l'implémentation de l'EPFL appelée Levee. Une vue synoptique des attaques agissant sur le « Control-Flow » majoritairement connues doit être dressée et analysée selon Levee.

[1] Site web du projet : http://dslab.epfl.ch/proj/cpi/
[2] « Code Pointer Integrity »
[3] « Code Pointer Separation »

# Objectifs du projet d'approfondissement

**Objectifs obligatoires**

1. Présenter les deux concepts CPI/CPS implémentés au sein de Levee
2. Expliquer l'architecture de Levee au sein de LLVM
3. Exposer une vue synoptique des attaques agissant sur le « control-flow »
4. Concevoir et implémenter un « Proof of Concept » d'attaque et sa protection grâce à Levee

**Objectifs optionnels**

1. Analyser les coûts d'un portage éventuel de Levee sur ARM

## Présenter les deux concepts CPI/CPS implémentés au sein de Levee

L'étudiant décrit le concept de « Code Pointer Integrity » et de Code Pointer Separation en détail ainsi que ces composants tels que SafeStack. Il traite brièvement l'historique des autres techniques tels que ASLR, DEP, Stack cookies, Coarse-grained CFI ou encore Finest-grained CFI.

## Expliquer l'architecture de Levee au sein de LLVM

L'étudiant décrit l'état du développement de Levee au sein de LLVM. L'implémentation et la structure de Levee au sein des différentes couches de LLVM sont expliquées. Une explication du rôle et des opérations faites sur le Frontend, l'Optimizer et le Backend est rédigée.

## Exposer une vue synoptique des attaques agissant sur le « Control-Flow »

L'étudiant expose de manière synoptique une liste non exhaustive des principales attaques de Control-Flow hijack tel que Remote code injection, ROP/return-to-libc, etc. Il décrit le rayon d'action de Levee et ses limites.

## Concevoir et implémenter un « Proof of Concept » d'attaque et sa protection grâce à Levee

L'étudiant met en place un environnement contenant tous les outils nécessaires pour tester Levee. Le « Proof of Concept » d'une attaque est écrit et une analyse de l'exécutable compilé avec et sans Levee est faite. L'attaque ainsi que la protection sont décrites en détail.

## (Optionnel) Analyser les coûts d'un portage éventuel de Levee sur ARM

L'étudiant analyse la faisabilité et les enjeux de porter Levee sur ARM au sein de LLVM. Une description des modifications nécessaires à apporter au Frontend et à l'Optimizer ainsi que le travail à faire sur le Backend ARM.

# Planning et milestones

Le planning donne une idée générale des délais à respecter pour que le travail puisse être mené à bien dans les délais. L'étudiant donnera chaque lundi un compte rendu de son avancement au professeur. Des séances supplémentaires peuvent être rajoutées si besoin est.

| Date | Num Sem | Milestones |
|---|---|---|
| 20 février 2017 | 1 | Séance préliminaire |
| 27 février 2017 | 2 | Rédaction du cahier des charges |
| 6 mars 2017 | 3 | Séance de validation du cahier des charges |
| 13 mars 2017 | 4 | Création de l'environnement docker, validation du CA |
| 20 mars 2017 | 5 | Documentation et début du rapport (template, structure, etc.) |
| 27 mars 2017 | 6 | Rédaction de la présentation des concepts CPI/CPS (Obl. 1) |
| 3 avril 2017 | 7 | Séance intermédiaire, avancement du rapport |
| 10 avril 2017 | 8 | Rédaction de l'architecture de Levee (Obl. 2) |
| 17 avril 2017 | 9 | Recherche concernant la vue synoptique |
| 24 avril 2017 | 10 | Rédaction de la vue synoptique (Obl. 3) |
| 1 mai 2017 | 11 | Début de l'implémentation de l'attaque |
| 8 mai 2017 | 12 | Implémentation de l'attaque (Obl. 4) |
| 15 mai 2017 | 13 | Analyse du portage sur ARM (Opt. 1) |
| 22 mai 2017 | 14 | Finalisation du rapport et correction |
| 2 juin 2017 | 15 | Envoi et dépôt du PA |
| 26 juin 2017 | 18 | Début des défenses |

**Professeur**
Prof. Pascal Junod

**Étudiant**
Joël Gugger