Enseignement de la “Sécurité Logicielle” à l’Université de Bordeaux

RESSI 2020

Emmanuel Fleury
<emmanuel.fleury@u-bordeaux.fr>
<emmanuel.fleury@univ-rennes1.fr>

Université de Bordeaux

Vendredi, 18 décembre 2020

Sommaire

Résumé rapide

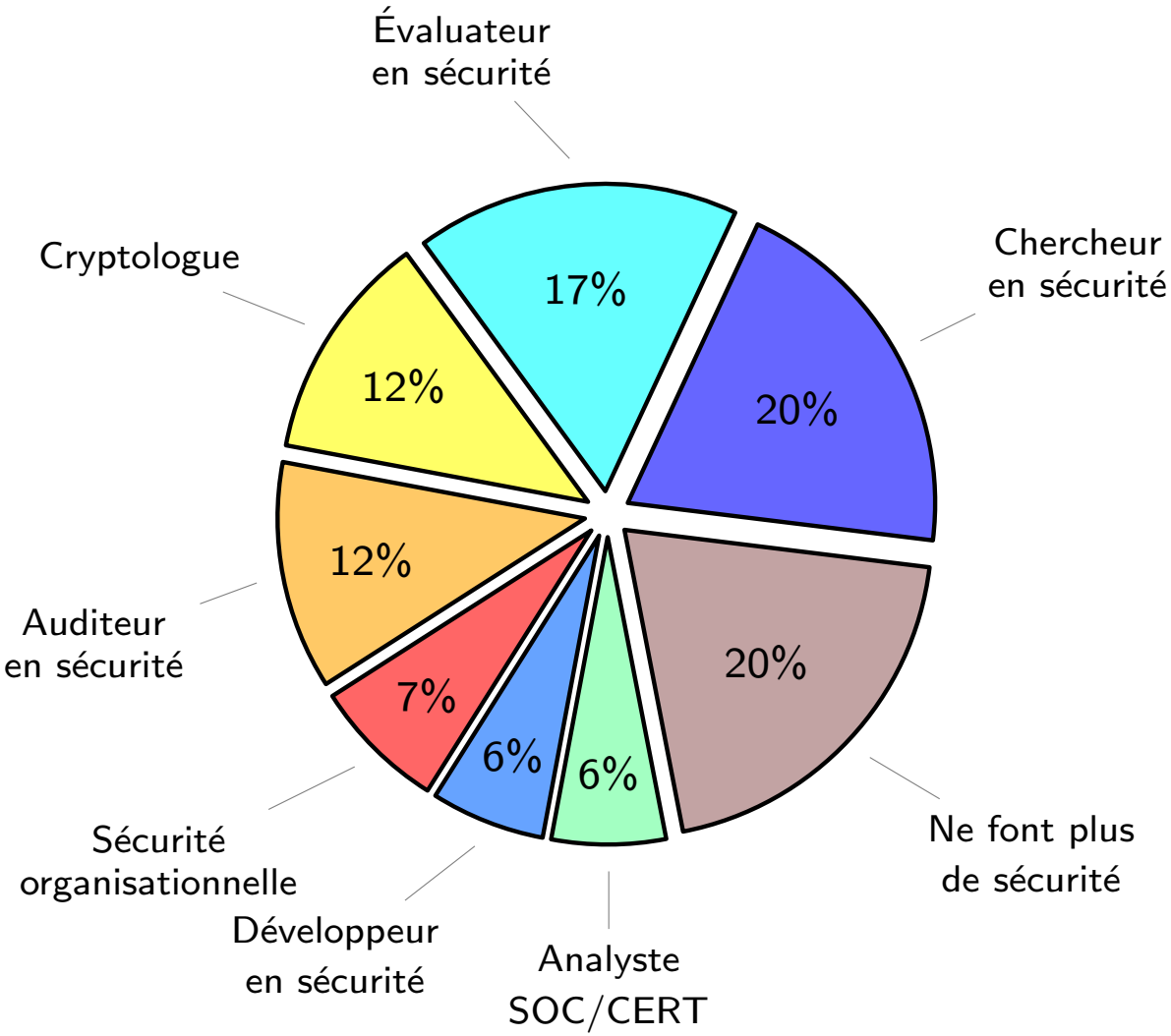
Historique

- **1998** : Ouverture du DESS *Cryptologie et Sécurité Informatique* par Christine Bachoc (Mathématiques) et Jean Bétréma (Informatique).
- **2001** : Passage au LMD, création du parcours *Cryptologie et Sécurité Informatique* au sein du *Master de Mathématiques Fondamentale* et du *Master d'Informatique de Bordeaux*.
- **2005** : Emmanuel Fleury remplace Jean Bétréma pour l'Informatique.
- **2006** : Gilles Zémor remplace Christine Bachoc pour les Mathématiques.
- **2016** : Nouvelle Accréditation du Master et création du programme actuel.
- **2017** : Obtention du label SecNumEdu de l'ANSSI.

Quelques statistiques

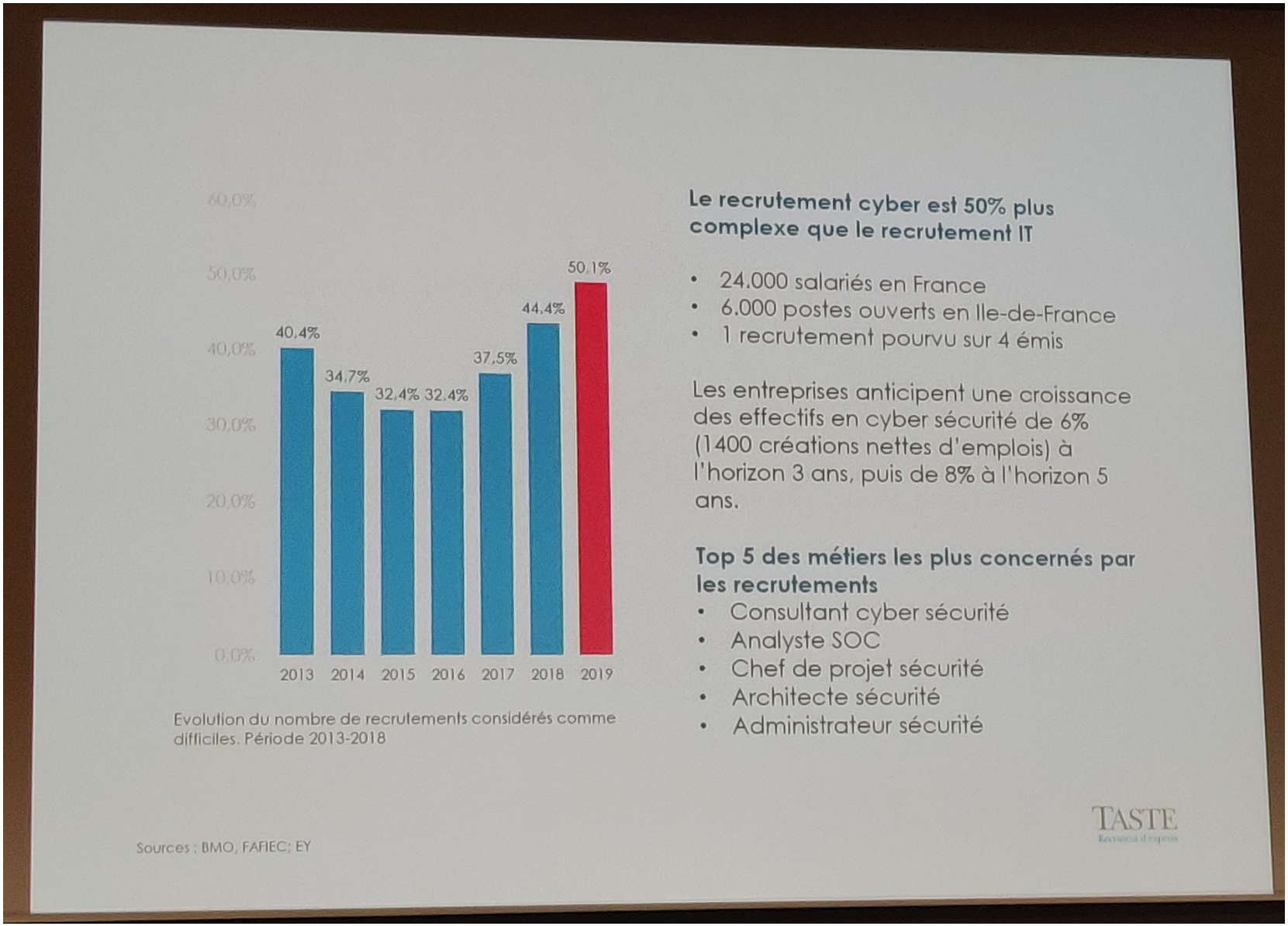
- **Nombre d'étudiants** : 30–40 (Master1), 20–30 (Master2), 20–30 Diplômés/an.
- **Taux d'insertion** : 94% (dont 30% en thèse, 70% en CDD/CDI).
- **Salaire médian de sortie** : 2400 euros/mois

Insertion du Master



- Sondage de janvier 2020 ;
- Liste tirée du référentiel des métiers de la sécurité (ANSSI) ;
- Sur les promotions 2005-2019 (environs 300 étudiants) ;
- 112 réponses (taux de réponse de ≈30%) ;
- 80% des "*anciens*" continuent dans le domaine de la sécurité (au sens large).

Les enjeux futurs



Source Taste RH, 2019.

Programme Master CSI

université de BORDEAUX

Semestre 7 (Master1)

Renforts en math/info	Théorie de la Complexité	Arithmétique	Programmation	Théorie de l'information	Option
0 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS

Option (1 choix) : "Analyse, classification, indexation des données", "Système d'exploitation".

Semestre 8 (Master1)

Cryptologie	Sécurité Logicielle	Calcul Formel	Option	TER	Anglais
6 ECTS	6 ECTS	6 ECTS	6 ECTS	3 ECTS	3 ECTS

Option (1 choix) : "Administration Réseau", "Introduction à la vérification", "Optimisation combinatoire", "Programmation des architectures parallèles".

Semestre 9 (Master2)

Cryptanalyse	Cryptologie avancée	Algorithmique arithmétique	Courbes elliptiques	Sécurité réseau	Sécurité système	Cartes à puces	Vérification des logiciels
6 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS

Choix de 5 UEs parmi les 8 UEs disponibles.

Semestre 10 (Master2)

Projet	Séminaires	Stage (Mars–Août) + Mémoire + Soutenance
6 ECTS	0 ECTS	24 ECTS

Programme Master CSI

université de BORDEAUX

Semestre 7 (Master1)

Renforts en math/info	Théorie de la Complexité	Arithmétique	Programmation	Théorie de l'information	Option
0 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS

Option (1 choix) : "Analyse, classification, indexation des données", "Système d'exploitation".

Semestre 8 (Master1)

Cryptologie	Sécurité Logicielle	Calcul Formel	Option	TER	Anglais
6 ECTS	6 ECTS	6 ECTS	6 ECTS	3 ECTS	3 ECTS

Option (1 choix) : "Administration Réseau", "Introduction à la vérification", "Optimisation combinatoire", "Programmation des architectures parallèles".

Semestre 9 (Master2)

Cryptanalyse	Cryptologie avancée	Algorithmique arithmétique	Courbes elliptiques	Sécurité réseau	Sécurité système	Cartes à puces	Vérification des logiciels
6 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS	6 ECTS

Choix de 5 UEs parmi les 8 UEs disponibles.

Semestre 10 (Master2)

Projet	Séminaires	Stage (Mars–Août) + Mémoire + Soutenance
6 ECTS	0 ECTS	24 ECTS

Comment enseigner la sécurité ?

université de BORDEAUX

- **La sécurité doit rester un prétexte !**

  La sécurité est juste une *attitude* face à problème, pas une fin en soi.

- **Le Diable est dans les détails !**

  Comprendre en détails le fonctionnement bas-niveau des programmes informatiques.

- **Cherchez l'erreur !**

  Enseigner les failles classiques et comment les trouver dans un programme.

- **La méthode scientifique...**

  Enseigner à penser contre soi-même et à adopter un attitude scientifique et méthodique.

- **Challengez vos étudiants !**

  Certains challenges (pas tous) sont très utiles pédagogiquement et motivent les étudiants.

Programmation (Semestre 7, Master1)

université de BORDEAUX

Syllabus

Obtenir une certaine autonomie en programmation C et Java.

Programmation C (6 semaines)

- C11 Language
- Memory Management in C
- Bit-level Programming
- `gcc` (or `clang`) Compiler
- `make` Build-system
- `git` Version Control System
- Debugging with `gdb` and `valgrind`
- Profiling and Testing Software

Programmation Java (6 semaines)

- Java Language (8+) ;
- Basics on Object Oriented Programming ;
- Inheritance and Extension ;
- Eclipse and Ant.

Évaluations

Projet (8 semaines)

Un logiciel de moyenne envergure (1000-1500 loc) qui résout un problème NP (ou plus) avec une structure de donnée non-triviale et nécessitant des manipulations bit-à-bit (Sudoku, Reversi, Slitherlink, Gomoku).

- Projet individuel ;
- Imposer un workflow basé sur `git` ;
- Tests logiciels à chaque étape ;
- Vérification du suivi des coding-styles ;
- Revue régulière du code ;
- Lancez des défis aux étudiants ;
- Être **impitoyable** !

Examen sur machine (3 heures)

- Tout Internet est autorisé ;
- Exercice équivalent aux Google Code Jam ;
- Donner des jeux de tests : *petit*, *moyen* et *avancé* (“avancé” nécessite une réflexion algorithmique).

Sécurité Logicielle (Semestre 8, Master1)

université de BORDEAUX

Syllabus

Comprendre les aspects bas niveaux des logiciels en espace-utilisateur et savoir exploiter les failles les plus courantes.

- Introduction to software security
- Usual programming flaws
- x86 assembly
- Executable Files
- Obfuscation and Reverse-engineering
- Shellcodes
- Stack buffer-overflows exploitation
- Advanced stack buffer-overflows exploitation (`ret2libc`, ROP)
- Heap buffer-overflows exploitation
- Format strings and other stories...
- Fuzzing Techniques
- Metasploit Framework

Évaluations

Projet

Résolution d'un maximum de challenges sur Root-me dans les catégories **App-script**, **App-system** et **Cracking**. Puis, remise d'un rapport qui explique la résolution de chaque challenge. Les points sont calculés comme suit (afin d'éviter la triche entre les groupes) :

- 1 si moins de 10% des équipes l'ont résolu.
- .75 si moins de 50% des équipes l'ont résolu.
- .5 si moins de 90% des équipes l'ont résolu.
- .25 si au moins une équipe ne l'a pas résolu.
- .125 si toutes les équipes l'ont résolu.

Examen sur table (3 heures)

- Petit exercice de compréhension de code (C, assembleur, exploit, ...) ;
- Lecture d'un article de recherche sur un sujet vu en cours et répondre à une douzaine de questions associées.

Syllabus

Apprendre à programmer au sein d'un système d'exploitation Linux et à en exploiter les failles les plus courantes.

- Introduction to Linux Kernel
- Kernel Modules Programming
- Programming Device Drivers
- Kernel Rootkit Programming (Part 1) (Syscall Hooking, Root Backdoor, Keylogger)
- Kernel Rootkit Programming (Part 2) (Hiding modules, files and processes)
- Kernel Rootkit Programming (Part 3) (Network Backdoor)
- Kernel Exploitation (Part 1) (NULL pointer, Stack-overflow)
- Kernel Exploitation (Part 2) (Slab/Slub/Slob, Heap-overflow)
- Kernel Exploitation (Part 3) (SMEP/SMAP, KASLR, ROP)

Évaluations

Présentation

Les étudiants exposent un sujet lié à la sécurité des systèmes d'exploitation devant leurs camarades. La liste des sujets est imposée et choisie parmi les sujets courants du moment par l'enseignant ou sur suggestion des étudiants (à valider par l'enseignant).

Examen sur table (3 heures)

- Petit exercice de compréhension de code (C, assembleur, exploit, ...) ;
- Lecture d'un article de recherche sur un sujet vu en cours et répondre à une douzaine de questions associées.

Conclusion

Que retenir de tout ça ?

- La sécurité est un prétexte pour enseigner des choses compliquées.
- La base de la sécurité logicielle reste la programmation.
- La pratique doit prendre une part très importante des cours.

Ce qu'il me reste à faire

- Automatisation des corrections du cours de "*Programming*".
- Améliorer les documents de cours de "*Software Security*".
- Finir le cours "*System Security*" qui est encore expérimental.
- Créer un cours de "*Software Reverse-Engineering*".

Quelques perspectives

- Former plus d'enseignants !
- Monter une plate-forme (inter-universitaire) de challenges.

Questions ?