Construire du code sécurisé : à quoi sensibiliser nos étudiants ?

Marie-Laure Potet, Laurent Mounier

VERIMAG, Grenoble, France

May 14, 2019

Un peu de contexte

MLP,LM,SB : Issus du domaine compilation/méthodes formelles.
Analyse de code pour la sécurité.

Cours **Software Security** dans spécialité Cybersécurité :

- Formation en apprentissage M2 CSI (SAFE) (10-12 et.)
- M2 Cybersecurity G-INP/UGA (dans Audit puis cours à part) (40 et., 36 h)
- Séminaires annuels à l'Esisar (parcours labellisé SecNumedu)

Cours **Software Security** dans parcours SI :

- ISI/Ensimag : initialement dans cours optionnel 3A (30 et.)
- Sur proposition des étudiants en 2A (ACSS, 70 et.)
- Filière aprentissage Ensimag depuis cette année (35-40 et.)
- Cours optionnel M2 Mosig (20 ét.)
- Cours M2 Mosig (20 ét.)

⇒ Quels contenus pour quels objectifs et pour quel public ? Comment être efficace (heures/acquis) ? Quelles sont les priorités ?

Objectif de la présentation

$\Rightarrow$ Sensibiliser les étudiants aux problèmes de développer du code non vulnérable, former les développeurs/outilleurs du futur.

Discussions attendues :

- Objectifs d'un cours Software Security, comment combiner au mieux attaque et défense
- Syllabus d'un tel cours suivant cursus/étudiants
- Comment préparer au mieux les étudiants à la construction de solutions sécurisées et aux outils d'analyse
- Comment les sensibiliser au développement de futurs outils (langages, smart compilateur, analyse de code . . .)

$\Rightarrow$ Concrètement partager/construire du matériel pédagogique : contenu de cours, exemples, TP mais aussi les sites intéressants.

Un premier contenu

- les vulnérabilités classiques des programmes
- les propriétés des langages aidant la sécurité
- les outils d'analyse existants
- les différentes formes de propriétés de sécurité
- les problèmes ouverts (ce sera à eux de proposer des solutions dans le futur !)

$\Rightarrow$ Des exercices/problèmes couvrant ces différents aspects.

Quelques cours/points de vue un peu similaires : Eric Poll `www.cs.ru.nl/E.Poll/ss/`, Michael Hicks www.cs.umd.edu/ mw et son blog www.pl-enthusiast.net

" Educating all future programmers about security is an important step toward increasing the security mindset. In CS 330 we illustrate common vulnerability classes and how they can be defended against by the language (e.g., by using those languages, like Rust, that are type safe) and programming patterns (e.g., by validating untrusted input). By doing so, we are hopefully making our students more fully cognizant of the task that awaits them in their future software development jobs."

Les vulnérabilités classiques

Message : le hacking est peut être un art mais la défense aussi ! Connaitre les sites expliquant les vulnérabilités, chercher des solutions existantes, ne pas réinventer. Se poser des questions sur les langages utilisés (hors la facilité de développer).

Methodologie : des aspects sémantiques, aux corrections et aux outils qui peuvent nous aider.

- les RTE
- mais aussi d'autres vulnérabilités (point délicat pour le programmeur, autres faiblesses)

Illustration : bon c'est vrai on aime bien le C avec ses 203 comportements non définis, ces 58 comportements non spécifiés et ses unsigned qui wrappent et ses signed qui provoquent des comportements non définis (-:

Un exemple

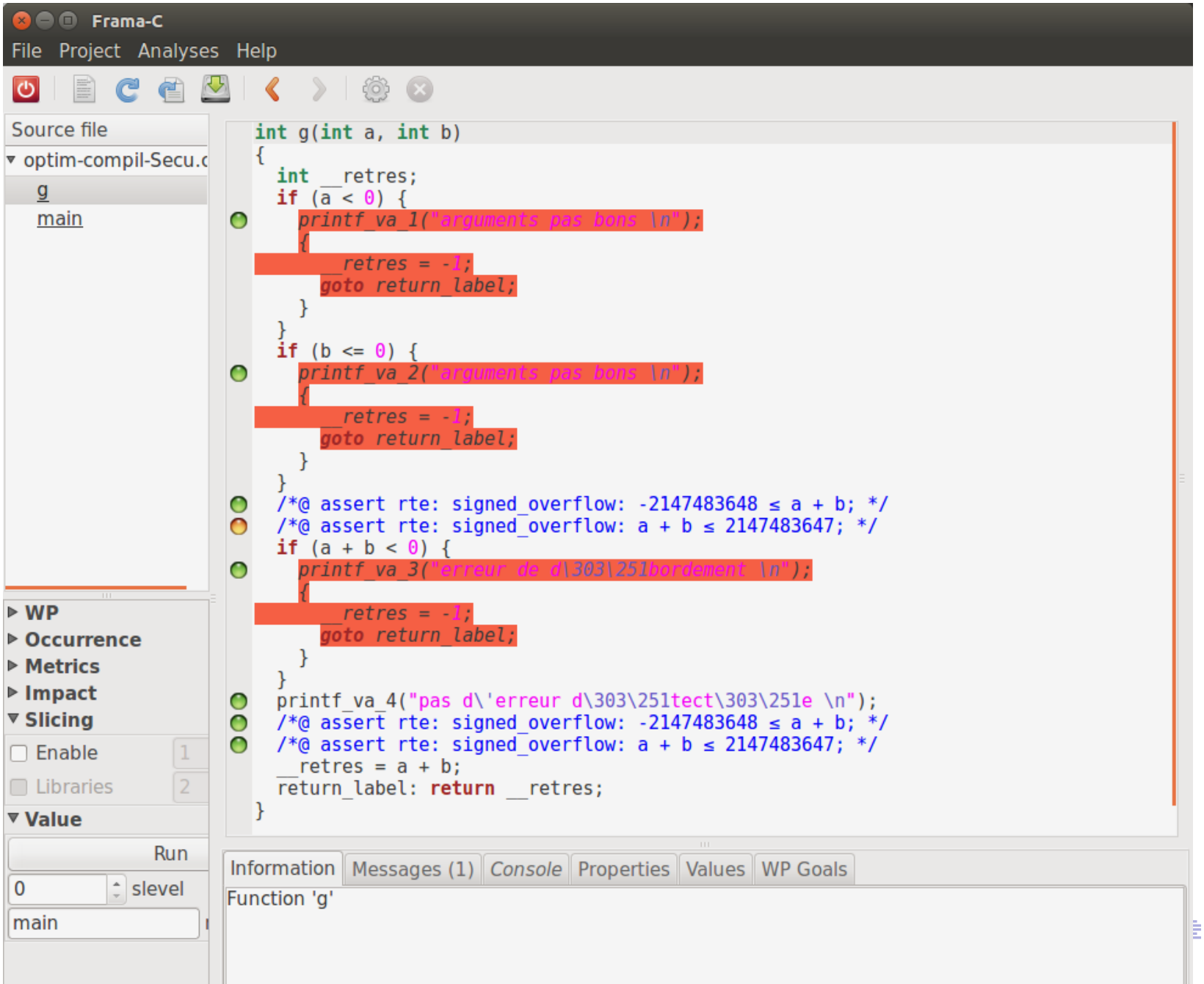
```
1  int g(int a, int b)
2  {    if (a < 0 || b <=0)
3          {printf("arguments pas bons \n");
4          return (-1) ; }
5        if (a+b < 0)
6          {printf("erreur de débordement \n");
7          return (-1) ; }
8          printf("pas d'erreur détectée \n");
9        return(a+b) ; };
```

Soit l'appel `g(2, INT_MAX)` :

- gcc optim-compil-Secu.c : imprime "erreur de débordement"
- gcc optim-compil-Secu.c -O2 : imprime "pas d'erreur détectée"

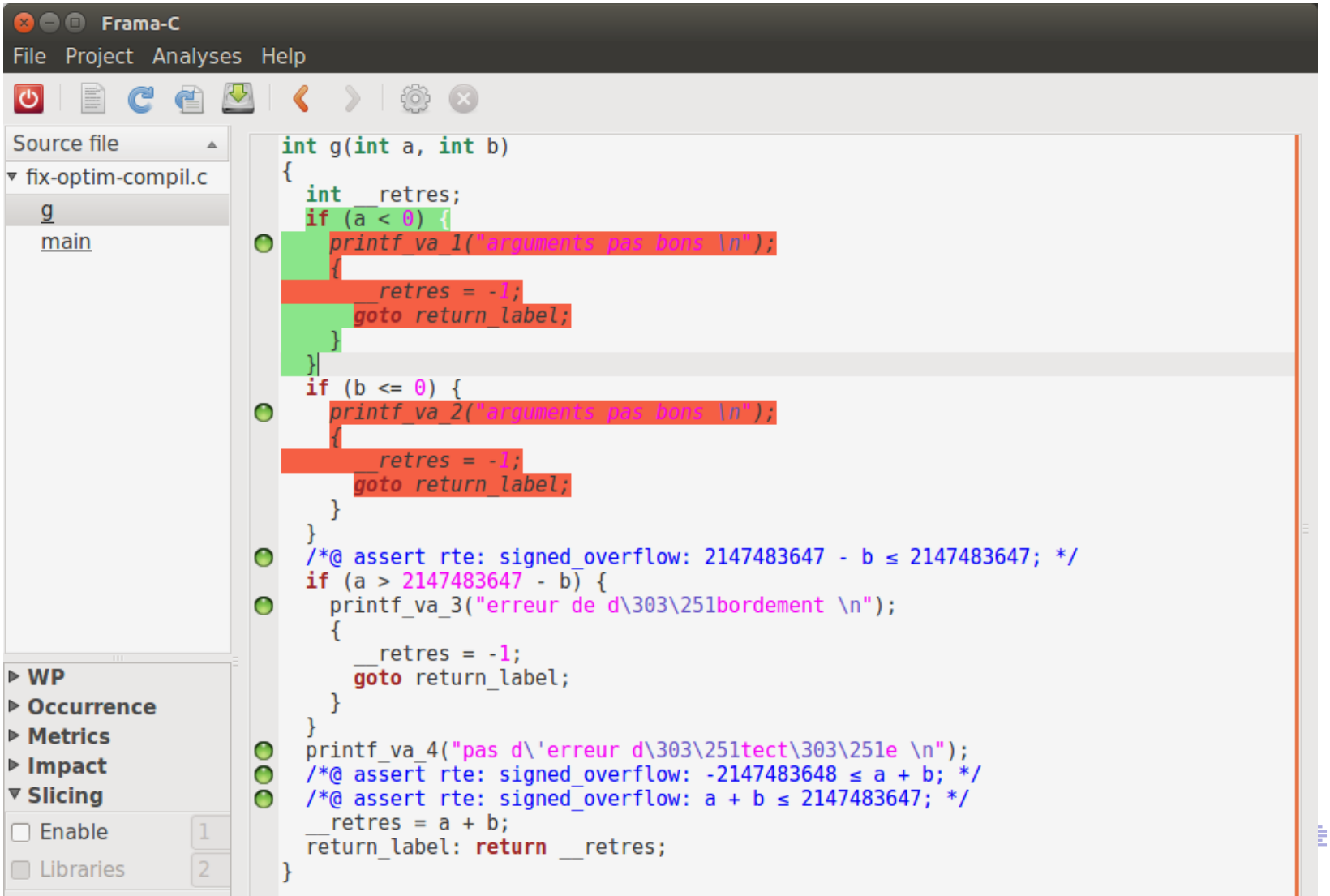
Mon compilateur est-il correct ?
⇒ Utiliser l'option `-fno-strict-overflow` (-:

Analyse statique (RTE+value)



Correction et analyse statique

INT32-C. Ensure that operations on signed integers do not result in overflow: wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+



Conclusion sur vulnérabilités détection/correction

- Un BO sur du C avec sans/canaries et en regardant l'assembleur (winloose),
- Des exemples de comportement bizarre puis correction avec site du cert
- Des exemples avec Polyspace/Bug finder (règles de codages et analyses dédiées).

Retour des étudiants (qui viennent d'écrire un compilateur !) :

- exemples simples mais illustratifs (influence de la compilation, exploitation).
- Compréhension des impacts fins de la sémantique et de vérification statique/dynamique/undefined (TB rtegen)

Example : Vulnerability and Exploitability

```
1  int main (int argc, char *argv[])
2  { char x=0 ;
3    char t1[8] ;
4    int i;
5    for (i=0;i<=atoi(argv[2]);i++)
6       t1[i]= atoi(argv[1]) ;
7    if (x != 0)     printf("You win !\n") ;
8    else            printf("You loose ...\n") ;
9    return 0 ; }
```

example1 2 7, exemple1 2 11, example1 2 17:
You loose ..., *** stack smashing detected ***, ***
stack smashing detected ***

example1-without-canary(-fno-stack-protector) :
You loose ..., You win ..., non termination

Sécurité et Propriétés des langages

Les propriétés des langages : ce qui est assuré à la compilation, à l'exécution ou pas détecté (et les vulnérabilités qui en résultent).

- type safety (typage fort, coercions vérifiées à l'exécution, . . .)
- memory safety (initialisation, durée de vie des objets, vérification à l'exécution des accès mémoire . . .)
- control flow safety (maitrise des points d'entrée dans le code)

Exemple :

	memory safety	type safety
C, C ++	-	-
Java, C#, Rust	+	+

On insiste sur ce qui peut être pris en charge statiquement ou dynamiquement, comment et avec quelles garanties (correction, complétude, théorème de Rice).

Un peu de matériel

Un exemple qui viole memory et type safety :

```
char *c ;
f() {char cc = 'a' ; c = &cc ;}
g() {int i = - 99 ; }
main () { f() ; g() ; printf(" %c ", *c) ; }
```

$\Rightarrow$ peut imprimer le caractère ÿ de code -1 (undefined dans la sémantique)

Formalisation des propriétés :

- OK type safety: le typage statique garantit que la représentation mémoire est toujours un objets du type
- Plus complexe memory safety : modèle mémoire fin et durée de vie des objets

Illustration concrète

Deux exemples intéressants à étudier :

- Java qui offre les bonnes propriétés précédentes (revérifiées sur le byte-code) et permet ainsi de mettre en place du contrôle d'accès portant sur l'appel de méthodes.
- Rust qui offre une gestion fine de la mémoire tout en évitant les références pendantes.

⇒ Aucun langage/plate-forme n'est parfait. Il faut en maitriser/comprendre les faiblesses (ex. étude Javasec).

Message : choisir un langage/API/plate-forme en se posant les bonnes questions en terme de sécurité et mettre en place les protections/vérifications adaptées à ce choix.

⇒ Intérêt des étudiants pour ce point de vue plus méta sur les langages + s'interrogent pourquoi on a des mauvais outils (i.e. non adaptés pour se défendre).

Exemple d'exercices

Exercice 1 : Variables non initialisées. Discuter des différentes solutions : 1) rien n'est fait 2) détection d'une erreur à l'exécution 3) initialisation par défaut par le compilateur 4) vérification de l'initialisation des variables par le compilateur vis-à-vis des critères suivants : a)coût de la solution b) erreurs potentielles induites pour le développeur c) assurances en sécurité.

Exercice 2 : Traitement des entiers. Discuter des solutions C (undef.), Java (wrap.), C# (excep.) Python/Rust (prec. inf.) ?

Exercice 3 : Ne pas déréférencer le pointeur null. Que pourrait-on proposer au niveau du langage, d'une analyse statique, d'une vérification à l'exécution ? (type annotation NonNull Java 8)

Les outils d'analyse de code

Sensibiliser les étudiants à l'existence des outils d'analyse de code et comprendre les apports et les limites de chaque technique. Expérimentation sur des petits exemples sécurité.

- Fuzzing (AFL) : automatisation, passage à l'échelle, caractère partiel des résultats susceptibles d'être obtenus.
- Exécution symbolique ou concolique (KLEE, angr) : techniques SMT pour la génération de séquences de tests. Problèmes classiques du symbolique (stub, état initial, . . .)
- Interprétation abstraite/WP : Comprendre les principes, les forces et faiblesses.

Message : une bonne compréhension de l'offre existante. Quels outils pour quels objectifs dans le processus de développement sécurisé ?

Différentes formes de propriétés

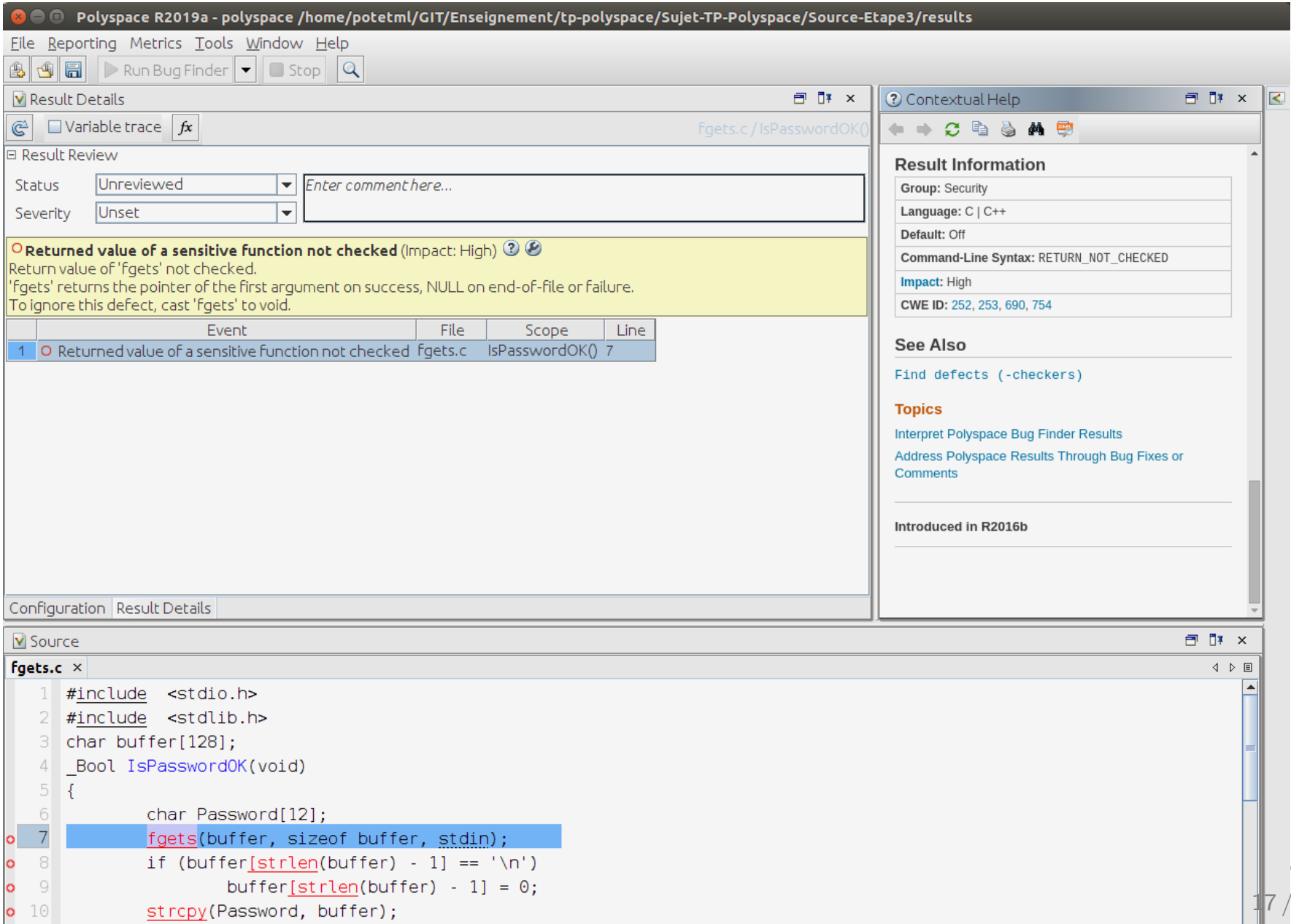
- Propriétés fonctionnelles et règles de codage
- Propriétés sur les flux : teinte, non-interférence, isolation
- Propriétés structurelles sur le code : exemple cryptocoding.net

Exemple Polyspace BugFinder :
`https://fr.mathworks.com/help/bugfinder/defects-runtime-checks-misra-coding-rules.html`

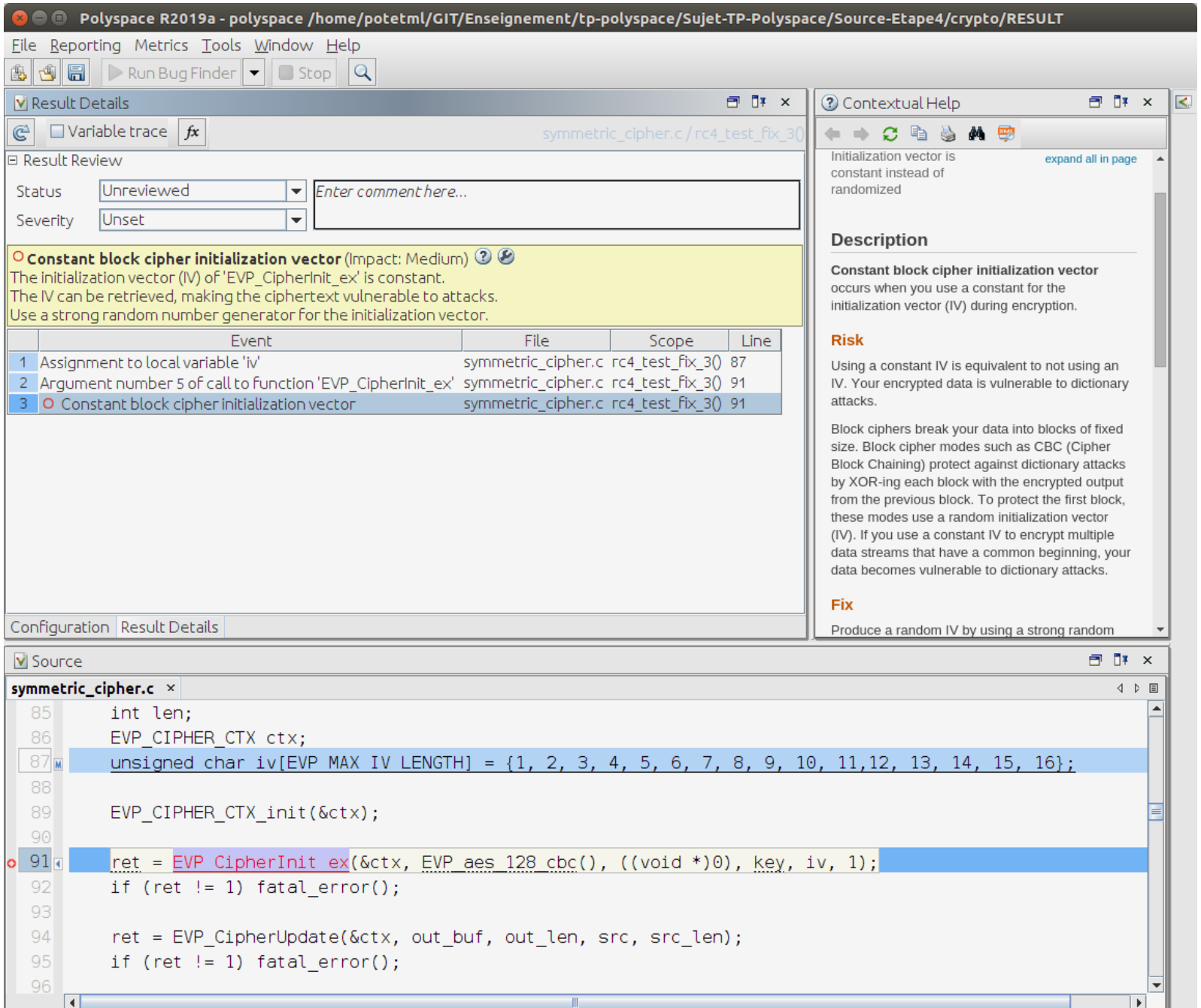
Message : garantir la sécurité du code ne se réduit pas à la recherche de vulnérabilités classiques mais nécessite de faire un cahier des charges précis, en identifiant ce qui doit être protégé et contre quoi, et en prenant en compte le contexte d'utilisation (d'où peuvent provenir les attaques).

⇒ Construire des outils et des chaines de développement qui prennent en compte la sécurité sont des problèmes ouverts.

Tester les retours de fonctions de la libc



Règles crypto



Un contenu M1 info/sécu

- vulnérabilités " classiques " (RTE, vérification i/o, exécution de données, élévation privilèges . . .)
- les outils à disposition (analyses de code mais aussi sites et principes de développement sécurisé)
- Plate-formes, compilateurs et langages : faiblesses et contre-mesures
- GL de la sécurité : robustesse plutôt que conformité, analyse de sécurité

Ne pas en faire trop, ça les surprend déjà . . . mais les sensibiliser au fait qu'il y a d'autres aspects.

On continue en M2 ISI : un cours mixant attaque/défense et CdC (exemple firewall + vérif de protocoles + étude d'une cible CSPN et CC)

Un contenu M2 cybersécurité

Dans l'idéal :

- reverse et exploitabilité
- offuscation de code/malware (et technique de déoffuscation)
- les outils pour audit/reverse (statique/dynamique/monitoring)
- programmation sécurisée (crypto/embarqué)
- modèles d'attaquant actif (observation/action) et analyses de sécurité
- retour sur langage et compilation pour le développement sécurisé

Question 1 : ou mettre (comment) les aspects contrôle d'accès i.e. au sens maîtrise des flux ?

Question 2 : qu'a t-on oublié ?

Question 3 : ...