# Concours RISC-V 2023

Yohan BEATTIE*, Julien FOLTÊTE*, Jim FOUQUET*, Quentin VACHER*,
Marina DEHEZ-CLEMENTI †, Arnaud DION †.
*Etudiant à ISAE-SUPAERO membre de *Pizzae Supatopesto*, Toulouse, France.
† Professeur-Chercheur à l'ISAE-SUPAERO, Toulouse, France

## RÉSUMÉ

Notre Projet Ingénierie Entreprise s'inscrit dans le cadre d'un concours organisé par Thalès, le GDR SoC et le CNFM prenant la forme d'un hackaton. En effet, le sujet du hackaton concerne la sécurité de l'architecture **CV32A6 RISC-V** et de son implémentation sur FPGA. L'architecture RISC-V est open-source et est de plus en plus utilisée par l'industrie des systèmes embarqués ; la **sécurité** de son implémentation devient donc un élément critique pour lequel former des futurs ingénieurs est un pari d'avenir. L'objet du concours est le développement et l'ajout de correctifs de sécurité à l'implémentation cible pour contrer des attaques lancées par le **Runtime Intrusion Prevention Evaluator (RIPE)**. Ces attaques, fournies au nombre de 10, présentent chacune une approche différente mais ont un point commun : leur finalité est la réalisation d'une attaque par dépassement de tampon (**buffer overflow**) ayant pour objectif d'exécuter du code malveillant sur le processeur. Deux catégories de correctifs peuvent être effectués, selon le support sur lequel ils sont développés : sur l'OS, ici Zephyr RTOS, ou sur l'implémentation du processeur en elle-même, écrite en SystemVerilog. Nous avons décidé de nous concentrer uniquement sur les correctifs OS et avons ainsi pu contrer toutes les attaques portant sur la Stack grâce à l'activation des **stack canaries** sur Zephyr. Notre correctif respecte les exigences de performance et a été validé par les organisateurs du concours.

## I. INTRODUCTION

Dans le cadre du Projet Innovation Entreprise (PIE) en dernière année de cycle ingénieur ISAE-SUPAERO, nous participons au RISC-V soft-core hackathon sponsorisé par Thales, le GDR SoC et le CNFM [1]. Notre objectif est de déjouer plusieurs cyberattaques sur le noyau CV32A6 tout en exécutant des applications de test sur le RTOS Zephyr. L'équipe qui réussit à contrer le plus d'attaques gagne. En cas d'égalité l'évaluation des performances des correctifs développés départage les éventuels vainqueurs. Les équipes peuvent intervenir à plusieurs niveaux, notamment en modifiant le noyau CV32A6, le RTOS Zephyr et/ou le compilateur. Les cyberattaques auxquelles nous sommes confrontés sont lancées par un logiciel nommé RIPE (Runtime Execution Prevention Evaluator) [5]. Ce logiciel est une extension du banc de test développé par *Wilander* et *Kamkar* [4] qui couvre plus de 850 formes d'attaques. Dans le cadre du concours, nous devons déjouer 10 cyberattaques fournies de type *dépassement de tampon* (*buffer overflow*) à l'aide de nos modifications.

Dans ce rapport, nous détaillons l'organisation adoptée pour répondre aux objectifs du concours et d'expliciter notre ajout de la protection adoptée contre les cyberattaques fournies ainsi que nos tentatives infructueuses.

## II. RAPPORT DES ACTIVITÉS

### A. Méthodologie

*1) Démarche employée pour répondre au problème:* Après nous être renseignés sur RIPE et les buffer overflows, nous avons décidé de concentrer nos recherches et travaux sur les correctifs OS. En effet, nous avons estimé qu'il s'agissait de la méthode d'attaque du problème la plus rentable pour nous en termes de rapport investissement de temps en amont/résultats potentiels : nos connaissances sur l'implémentation du processeur en SystemVerilog étaient bien plus faibles que celles sur l'OS.

Après quelques recherches, nous nous sommes rapidement rendu compte de la richesse de la documentation disponible de Zephyr RTOS. Cette-dernière décrit très bien les dispositifs intégrés à Zephyr pour la protection de la mémoire et des attaques par buffer overflow [3].

La plupart des protections se situent directement au sein même du code, via l'appel de fonctions `malloc()` sécurisées (appelées `k_malloc()`) ou l'utilisation au sein d'un programme de zones mémoire dédiées et fixées. Le concours nous imposant un programme d'attaque que nous ne pouvons pas modifier, ces protections à implémenter directement dans le code des programmes à sécuriser

ne sont alors malheureusement pas utilisables dans notre cas.

Cependant, certaines d'entre elles concernent la configuration de l'OS en lui-même, ce sont donc ces protections que nous allons utiliser. Zephyr possède notamment une implémentation intégrée de Stack Canaries (voir II-B) qui est l'option de configuration que nous avons utilisée [6].

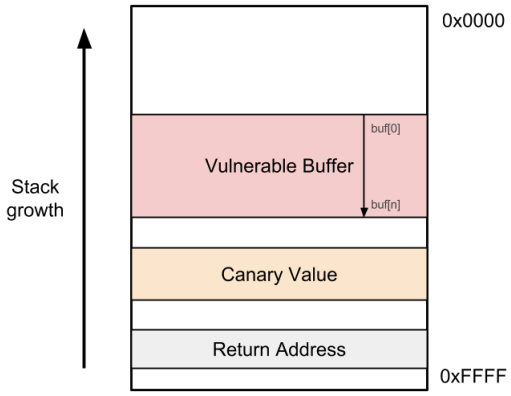


FIGURE 1. Représentation du stack canary

*2) Contribution : les stack canaries & buffer overflows:* En résumé, un buffer est une zone de mémoire qui est bornée ; un overflow signifie alors que l'on dépasse l'espace alloué initialement et que l'on écrit donc sur la mémoire qui suit, potentiellement un pointeur de retour (voir figure 1). Par exemple, lorsque l'on déclare une chaîne de caractère `str[14]` qui récupère l'entrée d'un utilisateur, alors, si des techniques de protection ne sont pas mises en place et que l'utilisateur entre une chaîne de caractères trop longue (ie. minimum 15 caractères) comprenant par exemple un code malveillant, alors ce code sera exécuté lors du prochain retour de fonction dans le code.

En effet, lorsqu'une fonction est appelée durant l'exécution d'un programme, le processeur enregistre l'adresse courante sur la stack afin qu'une fois la fonction terminée il puisse revenir à cet endroit dans le code et poursuive son exécution. Lors du retour d'une fonction, le processeur lit l'adresse de retour qui a été sauvegardée dans la stack frame. Si l'attaquant a pu corrompre cette adresse et y mettre un pointeur vers son code malveillant, alors c'est le code de notre attaquant qui va s'exécuter.

*Pour rappel, le stack canary est une valeur aléatoire placée après un buffer vulnérable, qui a pour but de vérifier l'intégrité des données suivant ce buffer. Tant que cette valeur n'est pas modifiée on sait que les données suivantes (en particulier l'adresse de retour de la fonction) n'ont pas été corrompues.*

C'est en parvenant à implémenter le stack canary dans l'OS Zephyr que nous avons déjoué 5 attaques avec un seul correctif (voir figure 2). Nous avons en particulier déjoué l'ensemble des attaques ayant lieu dans la stack.

*3) Autres travaux non fructueux:* Dans le cadre de ce projet, et afin de contrecarrer les attaques présentées, nous avons pu tenter d'implémenter de nombreuses méthodes de défense parmi celles présentées dans l'état de l'art. Malheureusement, nombre de ces dernières se sont révélées infructueuses :

- Nous avons tenté de modifier le comportement de certaines librairies liées au language C (libc) comme `memcpy` ou `malloc`. L'objectif était de protéger la mémoire en se concentrant sur la manière dont celle-ci était allouée et libérée. Nous avons tenté de forcer la vérification des tailles des données copiées (*library wrapping*) en modifiant directement ces fonctions ou en utilisant des *wrappers* dédiés comme libsafePlus. Nous avons également tenté de surveiller le dépassement des portions de la heap, par exemple à l'aide de heap canaries par exemple (suivant le même principe que les stack canaries mais pour la heap).

  Malheureusement, les modifications appliquées au librairies libc que nous avons trouvées n'avaient aucun effet sur l'exécution des programmes. Nous en avons déduit que le Docker employait d'autres librairies lors du build de Zephyr RTOS, librairies que nous n'avons pas trouvées malgré l'aide d'intervenants extérieurs au projet.
- Nous avons ensuite envisagé d'utiliser une méthode de chiffrement des pointeurs basée sur le modèle de *PointGuard* [2] . Cette solution chiffre les adresses en mémoire et les déchiffre dans le CPU (au niveau des registres pour plus de sécurité). Cette implémentation se fait à très bas niveau au sein-même des registres du processeur. Nous pensons avoir localiser le registre devant chiffrer nos pointeurs mais cela s'est

révelé sûrement infructueux du fait que le programme s'attaque lui même et donc nos adresses déchiffrées dans le registre lorsque le programme "malveillant" y accède. Nous avons été contraints de délaisser cette solution par manque de temps.

- L'utilisation de la DEP (Data Execution Prevention) a été essayée mais finalement abandonnée car d'après la documentation Zephyr RTOS , l'activation de cette fonction n'était pas disponible sur notre carte. De plus, il se trouve que cette défense était seulement susceptible de contrer l'attaque 6 car les autres attaques qui auraient été susceptibles d'être contrées par cette défense l'ont déjà été par les stack canaries. En effet, les attaques restantes (hormis l'attaque 6) sont de type "return-to-libc", "varleak" et "ROP". Elles consistent donc en l'exploitation de fonctions autorisées (libc) et en la fuite de données. Il n'y a alors pas de tentative d'exécution de code malveillant dans la stack ni dans la heap et la DEP est inutile.

### B. *Implémentation et résultats*

*1) Description détaillée du correctif:* L'implémentation des stack canaries dans Zephyr se réalise par son activation dans un fichier de configuration du build de l'OS. La documentation spécifie la nécessité d'activer également un générateur aléatoire, ce que nous avons bien évidemment effectué également dans le même fichier.

Cependant, le fichier de configuration décrit par la documentation officielle n'est pas présent dans la version de Zephyr fournie par le concours. Il a donc fallu retrouver ce fichier, stocké à l'emplacement suivant :

```
zephyr-docker/workspace/zephyr/
  boards/riscv/cv32a6_zybo/
    cv32a6_zybo_defconfig
```

Nous avons donc ajouté à ces fichier après les lignes de configuration déjà présentes les configurations suivantes :

```
CONFIG_TEST_RANDOM_GENERATOR=y
CONFIG_STACK_CANARIES=y
```

Et, de cette manière, les stack canaries sont activés sur Zephyr.

*2) Résultats sur les différentes attaques:* Si nous nous attendions à ce qu'une majorité des attaques portant sur la stack, notamment les trois premières, soient déjouées par l'activation des stack canaries, il s'agit finalement de toutes les attaques sur la stack qui ont été déjouées. Ce n'est pas vraiment surprenant puisque les stack canaries sont censés être efficaces peu importe la méthode utilisée, mais nous n'avions pas réellement réfléchi à l'application de cette solution aux attaques n°5 et 9. Nous avons donc été agréablement surpris lors de nos tests après activation des stack canaries.

Le tableau suivant récapitule les attaques déjouées par notre correctif. (voir figure 2)

*3) Validation du correctif:* Nous avons pu constater que toutes les attaques portant sur la Stack ont été déjouées par ce correctif. Or, comme mentionné précédemment, nous étions étonné car il était indiqué que les attaques étaient de difficulté croissante, et l'attaque n°9, qui devait être parmi les plus difficiles, est ici contrecarrée. Cela nous a amenés à questionner la validité de notre solution et c'est pourquoi nous avons contacté les organisateurs du concours afin de nous assurer que notre patch était valide. Après en avoir débattu en interne, ils nous ont confirmé la validité de notre solution.

Concernant l'exigence de performance, l'exécution du script fourni donne les informations suivantes :

```
Begining of execution with depth
12, call number 50, seed value
63728127.0
SUCCESS: computed value 868200.0 -
duration: 25.367454 sec
634186341 cycles
```

Son exécution avant implémentation de notre correctif donne les résultats suivants :

```
Begining of execution with depth
12, call number 50, seed value
63728127.0
SUCCESS: computed value 868200.0 -
duration: 25.300457 sec
632511434 cycles
```

Selon la documentation fournie, il s'agit d'un script réalisant diverses opérations plus ou moins gourmandes sur la mémoire afin d'en contrôler les performances. La *computed value* est bien identique comme attendu, et l'écart de durée de l'ordre de 0.26%, ce qui rentre dans les exigences de performance.

Notre correctif est donc valide et répond aux critères du concours.

| Num. | Technique | Code d'attaque | Code_Ptr | Localisation | Fonction | Déjouée | Contre-mesure |
|---|---|---|---|---|---|---|---|
| **1** | Directe | No_NOP | Ret_Addr | Stack | Memcpy | ✓ | Stack canary |
| **2** | Directe | No_NOP | Func_Ptr_Stack_Var | Stack | Memcpy | ✓ | Stack canary |
| **3** | Indirecte | No_NOP | Func_Ptr_Stack_Var | Stack | Memcpy | ✓ | Stack canary |
| **4** | Directe | Data Only | Var_Leak | Heap | Sprintf | ✗ | NA |
| **5** | Directe | Retour vers libc | Ret_Addr | Stack | Memcpy | ✓ | Stack canary |
| **6** | Indirecte | Sans_NOP | Func_Ptr_Heap | Heap | Memcpy | ✗ | NA |
| **7** | Indirecte | Retour vers libc | Struct_Func_Ptr_Heap | Heap | Homebrew | ✗ | NA |
| **8** | Indirecte | Retour vers libc | Longjmp_Buf_Heap | Heap | Memcpy | ✗ | NA |
| **9** | Directe | Return Oriented Programming | Ret_Addr | Stack | Memcpy | ✓ | Stack canary |
| **10** | Directe | Return Oriented Programming | Struct_Func_Ptr_Heap | Heap | Sprintf | ✗ | NA |

FIGURE 2. Bilan du patch des attaques

## III. CONCLUSION

Notre Projet Ingénierie et Entreprise (PIE) consistait à participer à un concours organisé par Thales, le GDR SoC et le CNFM, qui portait sur la sécurité de l'architecture CV32A6 RISC-V et de son implémentation sur une architecture FPGA. Malgré des problèmes importants rencontrés lors de l'installation et nos difficultés à anticiper de manière précise la durée des différentes tâches du projet, nous avons pu effectuer de nombreuses tentatives de protection contre les 10 attaques fournies.

En considérant que notre travail portait sur l'architecture RISC-V en SystemVerilog, ce dernier requérait de devenir familier avec les outils et le langage dédiés avec pour seules ressources les autoformations disponibles en ligne. Ce dernier nous aurait demandé une charge de travail considérable dans un délai raccourci à cause des problèmes d'installation rencontrés si nous n'avions pas décidé de focaliser nos recherches sur les correctifs au sein de l'OS Zephyr. Cette démarche a porté ses fruits puisque nous avons pu, en mettant en place des stack canaries au sein de l'OS, contrecarrer les 5 attaques portant sur la stack. Ce correctif a été validé par l'organisation du concours et respecte largement les exigences de performance.

Par la suite, notre choix de consacrer nos recherches uniquement aux correctifs appliqués à l'OS a montré ses limites : nos différentes tentatives de correction des attaques portant sur la Heap se sont montrées infructueuses. Il est évident que dans une deuxième partie de projet, avec plus de temps, nous aurions dû nous consacrer à l'étude approfondie de l'architecture CV32A6 en elle-même.

Nous sommes reconnaissants d'avoir pu acquérir des compétences en matière de correction de vulnérabilités dans les systèmes embarqués. En outre, ce projet nous a permis de capitaliser des connaissances sur la sécurité des systèmes embarqués et sur les mécanismes de protection contre les attaques de type buffer overflow qui font partie des attaques informatiques les plus répandues et critiques.

## RÉFÉRENCES


[1] « 3rd national RISC-V student contest 2022-2023 ». In : (). URL : https://web-pcm.cnfm.fr/wp-content/uploads/2022/10/Annonce-RISC-V-contest-2022-2023-v1.pdf.

[2] Crispin COWAN et al. « Pointguard TM : protecting pointers from buffer overflow vulnerabilities ». In : (jan. 2003), p. 7-7.

[3] Memory Protection Design. Mars 2023. URL : https://docs.zephyrproject.org/latest/kernel/usermode/memory_domain.html#.

[4] John WILANDER et Mariam KAMKAR. « A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention ». In : (2003).

[5] John WILANDER et al. « RIPE: Runtime Intrusion Prevention Evaluator ». In : ACSAC '11 (2011), p. 41-50.

[6] Zephyr stack canary. Mars 2023. URL : https://docs.zephyrproject.org/latest/kconfig.html#CONFIG_STACK_CANARIES.


Nous, soussignés, M. Yohan BEATTIE, M. Julien FOLTETE, M. FOUQUET Jim, M. VACHER Quentin, certifions qu'il s'agit d'un travail original et que toutes les sources utilisées ont été indiquées dans leur intégralité. Nous certifions également que nous n'avons pas copié ou utilisé des idées ou des formulations provenant de tout autre travail, article ou mémoire, sous forme imprimée ou électronique, sans mentionner leur origine précise, et que les citations complètes sont indiquées entre guillemets.

Fait à Toulouse, le 11/05/2023

Signatures :