

Increasing Unikraft security by implementing address space layout randomization

Master Thesis



Author: Loslever Terry (Student ID: S174777)

Supervisor: Univ.-Prof. Dr. Mathy Laurent

Co-Supervisor: Gain Gauthier

Department of Electricity, Electronics and Computer sciences

Faculty of applied sciences

University of Liège

October 29, 2021

Abstract

[Abstract goes here (max. 1 page)]

Contents		
1	Introduction	1
2	State Of The Art : Linux	2
3	Section Two	4
3.1	Exemplary Figure	4
3.2	Exemplary Figure Referencing	4
4	Section X	5
4.1	Exemplary Citation	5
A	Appendix	6
	References	7

List of Figures

1	Exemplary Figure	4
---	----------------------------	---

List of Tables

1 Introduction

2 State Of The Art : Linux

ASLR consists in the allocation of random places in virtual memory for the heap, stack and libraries.

In `include/linux/randomize_kstack.h` are the macros which compute the random offset.

In `arch/x86/kernel/process.c` -> `arch_align_stack` `arch_randomize_brk`

Puisqu'en Unikraft presque l'entièreté du système tourne grâce à des librairies internes ou externe, le code à modifier se trouve très certainement dans `lib/posix-process/include/sys`, `lib/lib/posix-process/process.c`. Ces éléments permettent d'implémenter l'ASLR sur les process par dessus le kernel.

-> au lieu d'aller direct dans la structure, peut-être regarder à `brk` -> `set` `nbr` dans l'elf pour pouvoir disable, modifier `process.c` -> Utilise des éléments de `sys/prctl.h` qui est dans `include` de la `mm` `lib` (invest sur `va_start`) -> Contient la `struct` et des `defines` -> `va_start`, `va_end` ne sont que de la gestion de listes de param -> by default unikraft doesn't support multiple process creation. -> `/fs/binfmt_elf.c` sont les fonctions qui chargent les elf -> `fs/exec.c` qui se charge de charger le programme et de le mettre en mémoire -> l'espace mémoire est donnée dans la structure `linux_binprm` de l'`exec` et on lui attribue un espace mémoire `mm_struct` et `vm_area_struct` -> `setup_arg_pages`: update les flags et la position du stack peut être relocatilisée -> Ils jouent direct avec le stack depuis la structure `vma` et font du page align. Il faut trouver l'équivalent en Unikraft pour trouver la `vma` et le stack

-> il va surement falloir changer le chargement des elfs aussi

C'EST `SETUPC.C` QU'IL FAUT CHANGER -> Il y a déjà un `_mb_init_initrd()` présent. Ressemble à de la randomisation mais ne fait rien quand on lance plusieurs fois une application Unikraft -> Le commenter ou non ne change rien -> Réussi à modifier l'address de base du stack en touchant à `_bvkmlplat_cfg.heap.start`

unikraft/lib/ukboot: -> lance tous les subsystems de l'OS avec de call la fonction `main` du programme -> Investiguer sur `uk_stack_chk_guard_setupH` -> `lib/uksp` lib qui traite le stack -> `ukarch_read_sp` permet de récupérer le `sp` dans le programme -> le stack semble être présent dans `uksched/sched.c` et dans les sous branches de cette lib -> Follow the trace of `create_stack` -> Intimement lié à `uk_alloc` qui est inclu dans la structure `uk_sched` -> mettre l'ASLR dans le `alloc.c` (peut-être) [EN PARLER AVEC MATHY] -> `alloc.h` possède les remplacement inline des certaines fonctions du C -> `unlikely()` est une fonction built par dessus un élément `cpu` (`lcpu.h`) -> utilise un pointeur sur fonction de la structure `uk_alloc` pour l'allocation [Demander à Mathy comment ça marche] -> les pointeurs orienté object -> agnostic à l'allocateur mémoire -> le `makemenu` config

en dessous de `uk_alloc` permet d'avoir plusieurs allocateurs possibles. -> regarde si je sais le faire de manière générique (même si c'est mieux) -> bcp de fonctions où on peut spécifier l'adresse que l'on voudrait aligner `posix_mem_align` -> Fonction plus spécifique pour les libraires (déjà réutiliser `posix_mem_align` (espace virtuel) IDÉE:

Mettre le kernel à une adresse arbitraire.

-> plusieurs approches possibles. -> le chargement elf par qemu est assez simple. -> compiler en pos indépendant pour pouvoir modifier les positions dans le code lui-même. -> changer l'ordre quand on génère l'image. -> méthode statique plus simple de bouger -> random pour l'elf à load a posteriori (si certains trucs dans les choix sont réutilisables le faire)

Avec Gauthier -> jouer sur les pages table: attention aux jumps relatifs. -> tables d'indirection.

Comme linux installer les fonctions dans `process.c`, et les mettre dispo dans le fichier qui se charge de charger le programme en mémoire.

`linux_binprm` est la structure qui est utilisée pour le chargement des librairies.

ASLR and its limitation in attacks. It's useful in bufferoverflows but not in format string. not the panacea + don't have enough entropy in 32bits system -> able to brute force it. (Mathias Payer, 2013)

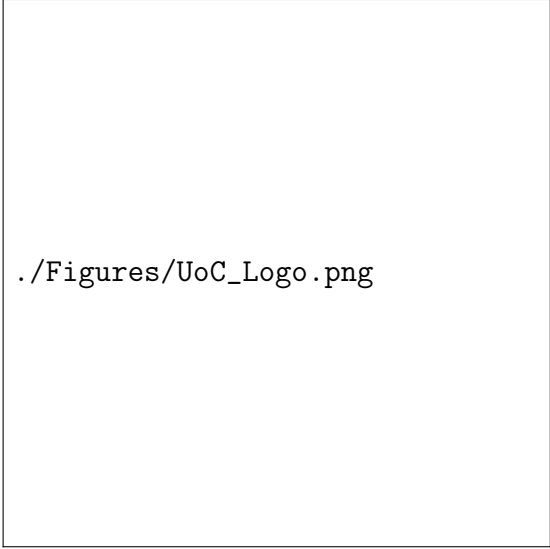
<https://xorl.wordpress.com/2011/01/16/linux-kernel-aslr-implementation/>

3 Section Two

...

3.1 Exemplary Figure

...



`./Figures/UoC_Logo.png`

Figure 1: Exemplary Figure

3.2 Exemplary Figure Referencing

See Figure 1 for details. Additional information can be found in the footnote ¹.

¹Image taken from [https://en.wikipedia.org/wiki/File:Siegel_Uni-Koeln_\(Grau\).svg](https://en.wikipedia.org/wiki/File:Siegel_Uni-Koeln_(Grau).svg).

4 Section X

4.1 Exemplary Citation

In this research we follow ? (?)...

PowerTAC is an example of a multiagent competitive gaming platform (?, ?).

A Appendix

...

References

Mathias Payer, T. R. (2013). String Oriented Programming: When ASLR is not Enough. *<https://nebelwelt.net/files/13PPREW.pdf>*(2), 447–460.