



Exploitation de control flow et protection sur systèmes embarqués

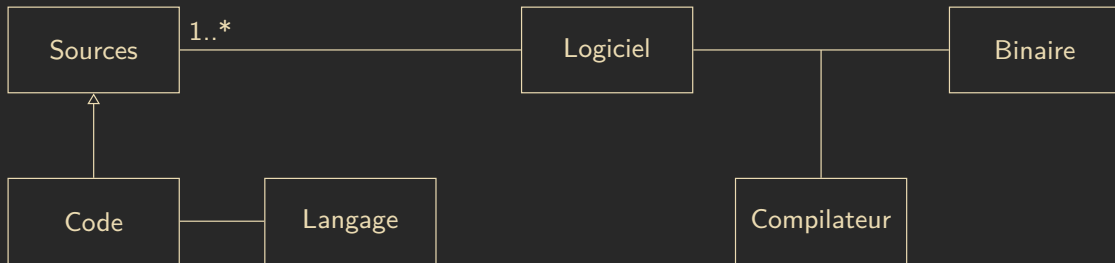
Introduction au ROP

Étienne Louboutin
September 23, 2026

- Introduction à la construction d'un logiciel
- Modélisation de l'exécution d'un logiciel
- Principe du détournement de flot de contrôle
- Exemple récent d'utilisation du RoP dans un PoC publique (CVE-2026-46242)
- Quelques protections ?

Définition d'un logiciel

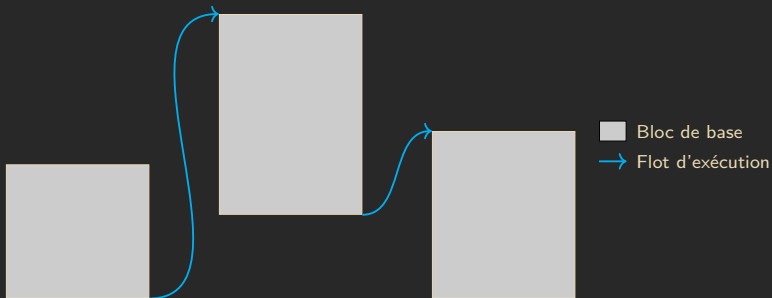
- Un logiciel est un outil informatique fournissant des services



- Sensibilité aux attaques du logiciel déterminée au niveau du binaire
- Le développeur travaille sur les sources

Des sources à l'exécution

- Les fonctionnalités du logiciel sont représentées à plusieurs niveaux
 1. les sources
 2. la représentation interne utilisée par le compilateur
 3. le binaire
- Des protections existent sur les différents niveaux
- Un modèle d'exécution générique: flot d'exécution du logiciel



Des sources à l'exécution





- Soit une fonction simple

```
1
2 int char_search(char *str, char c){
3     int i = 0;
4     while(*str != '\0'){
5         if (*str == c){
6             return i;
7         }
8         str++;
9         ++i;
10    }
11    return -1;
12 }
```



- Traduction dans une représentation interne au compilateur
- Optimisations et sécurisations faites principalement à ce niveau
- Résultat avec le compilateur gcc l'option `-Os`:

```
1  ;; Function char_search (char_search, funcdef_no=0,  
2    decl_uid=1794, cgraph_uid=0, symbol_order=0)  
3  
4  Removing basic block 6  
5  Removing basic block 7  
6  Removing basic block 8  
7  char_search (char * str, char c)  
8  {  
9    unsigned long ivtmp.5;  
10   int i;  
11   char _1;  
12   int _4;  
13  
14   <bb 2> [14.44%]:  
15     goto <bb 4>; [100.00%]  
16  
17   <bb 3> [92.50%]:  
18     ivtmp.5_12 = ivtmp.5_10 + 1;  
19     if (_1 == c_7(D))  
20       goto <bb 5>; [7.50%]  
21   else  
22     goto <bb 4>; [92.50%]
```

```
1  <bb 4> [100.00%]:  
2    # ivtmp.5_10 = PHI <0(2), ivtmp.5_12(3)>  
3    i_3 = (int) ivtmp.5_10;  
4    _1 = MEM[base: str_5(D), index: ivtmp.5_10, offset: 0B];  
5    if (_1 != 0)  
6      goto <bb 3>; [92.50%]  
7    else  
8      goto <bb 5>; [7.50%]  
9  
10   <bb 5> [14.44%]:  
11     # _4 = PHI <i_3(3), -1(4)>  
12     return _4;  
13  
14 }
```

Des sources à l'exécution

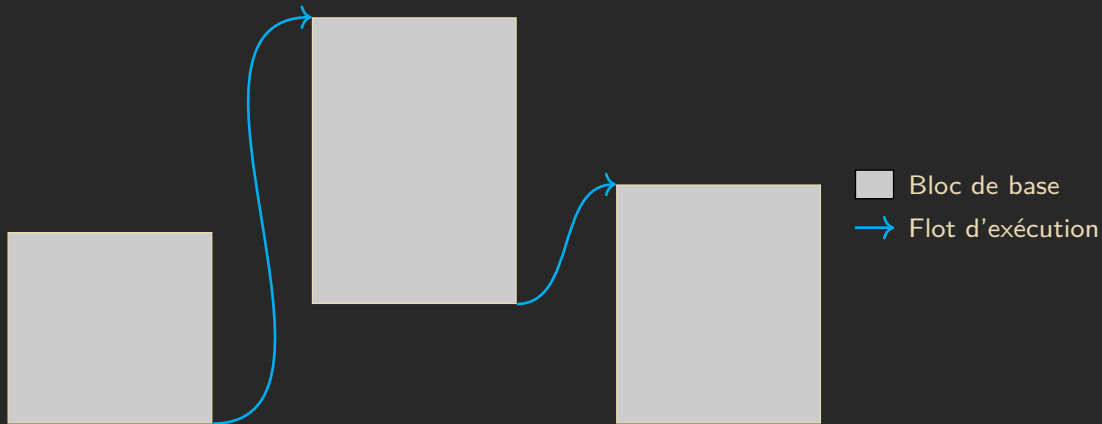


- La compilation finit par la génération du binaire
- Le binaire est représenté dans un assembleur, ici le GNU ASM

```
1
2 simple_search.o:    format de fichier elf64-x86-64
3
4
5 Deassemblage de la section .text :
6
7 0000000000000000 <char_search>:
8 0: 31 d2          xor    %edx,%edx
9 2: 8a 0c 17      mov    (%rdi,%rdx,1),%cl
10 5: 89 d0          mov    %edx,%eax
11 7: 84 c9          test   %cl,%cl
12 9: 74 0a          je     15 <char_search+0x15>
13 b: 48 ff c2      inc    %rdx
14 e: 40 38 f1      cmp    %sil,%cl
15 11: 75 ef          jne    2 <char_search+0x2>
16 13: eb 03          jmp    18 <char_search+0x18>
17 15: 83 c8 ff      or     $0xffffffff,%eax
18 18: c3            ret
```

```
1
2 simple_search_clang.o:    format de fichier elf64-x86-64
3
4
5 Deassemblage de la section .text :
6
7 0000000000000000 <char_search>:
8 0: 8a 17          mov    (%rdi),%dl
9 2: b8 ff ff ff ff mov    $0xffffffff,%eax
10 7: 84 d2          test   %dl,%dl
11 9: 74 12          je     1d <char_search+0x1d>
12 b: 31 c9          xor    %ecx,%ecx
13 d: 40 38 f2      cmp    %sil,%dl
14 10: 74 0c          je     1e <char_search+0x1e>
15 12: 8a 54 0f 01    mov    0x1(%rdi,%rcx,1),%dl
16 16: 48 ff c1      inc    %rcx
17 19: 84 d2          test   %dl,%dl
18 1b: 75 f0          jne    d <char_search+0xd>
19 1d: c3            ret
20 1e: 89 c8          mov    %ecx,%eax
21 20: c3            ret
```

Une structure de contrôle d'exécution



Une structure de contrôle d'exécution

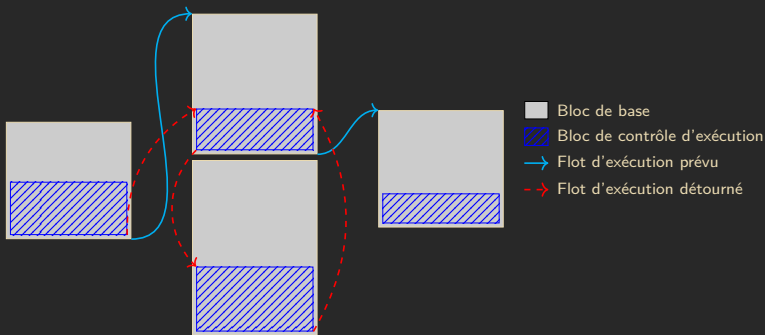
- Une suite d'instructions pour contrôler le flot:

```
or $0xffffffff,%eax; ret
```

- Qui peut être détournée

- l'instruction `ret` lit une adresse *altérable*
- cette adresse correspond à la prochaine instruction à exécuter

- Un attaquant peut altérer le flot d'exécution du logiciel:



Une définition de gadget

- Un gadget :
 - un pivot: permettant de contrôler le flot d'exécution du programme
 - le corps du gadget qui correspond aux instructions précédant le pivot

Une définition de gadget

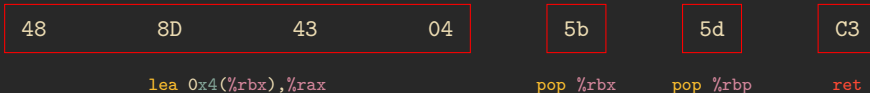
- Un gadget :
 - un pivot: permettant de contrôler le flot d'exécution du programme
 - le corps du gadget qui correspond aux instructions précédant le pivot
- `or $0xffffffff,%eax; ret`
 - le corps `or $0xffffffff,%eax`
 - le pivot `ret`

Une définition de gadget

- Un gadget :
 - un pivot: permettant de contrôler le flot d'exécution du programme
 - le corps du gadget qui correspond aux instructions précédant le pivot
- `or $0xffffffff,%eax; ret`
 - le corps `or $0xffffffff,%eax`
 - le pivot `ret`
- `lea 0x4(%rbx), %rax; pop %rbx; pop %rbp; ret`
 - le corps `lea 0x4(%rbx), %rax; pop %rbx; pop %rbp`
 - le pivot `ret`

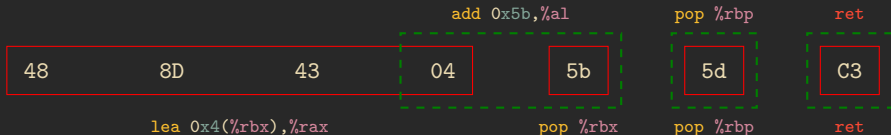
Une définition de gadget

- Un gadget :
 - un pivot: permettant de contrôler le flot d'exécution du programme
 - le corps du gadget qui correspond aux instructions précédant le pivot
- `or $0xffffffff,%eax; ret`
 - le corps `or $0xffffffff,%eax`
 - le pivot `ret`
- `lea 0x4(%rbx), %rax; pop %rbx; pop %rbp; ret`
 - le corps `lea 0x4(%rbx), %rax; pop %rbx; pop %rbp`
 - le pivot `ret`
- D'autres gadgets peuvent être découverts (surtout sur x86)



Une définition de gadget

- Un gadget :
 - un pivot: permettant de contrôler le flot d'exécution du programme
 - le corps du gadget qui correspond aux instructions précédant le pivot
- `or $0xffffffff,%eax; ret`
 - le corps `or $0xffffffff,%eax`
 - le pivot `ret`
- `lea 0x4(%rbx), %rax; pop %rbx; pop %rbp; ret`
 - le corps `lea 0x4(%rbx), %rax; pop %rbx; pop %rbp`
 - le pivot `ret`
- D'autres gadgets peuvent être découverts (surtout sur x86)



La chaîne de détournement de flot d'exécution

- Suite d'adresses de gadgets composant le programme d'attaque
- Inclusion possible de données supplémentaires
- Exemple pour l'équivalent du code C suivant, après un buffer overflow sur la stack:

```
mprotect(stack, 0x1000, RWX);
```

Assembleur du call

```
movl    7, %edx  
movl    4096, %esi  
movabsq 0x7fffffff000, %rdi  
call    mprotect
```

Chaîne de détournement après override de rip avec l'adresse de

`pop rdi; ret`

```
.quad    _stack_pointer & ~ 0xfff  
.quad    0x7e43b    %pop rdx; pop rsi; ret  
.quad    7  
.quad    0x1000  
.quad    0x4eb8c    % call mprotect
```

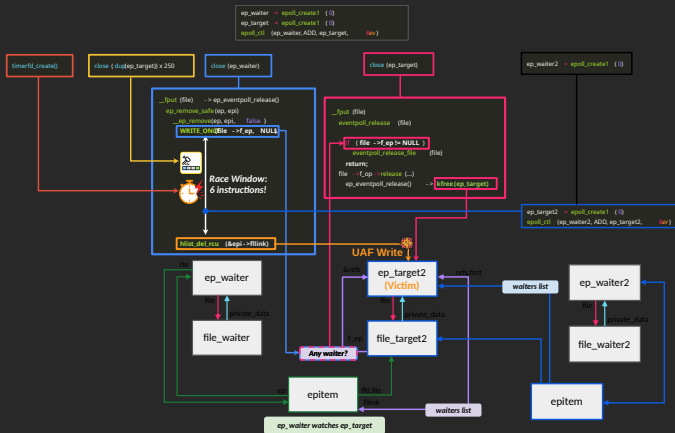
- Les gadgets présents ne correspondent pas forcément exactement aux besoins de l'attaquant

En pratique: CVE-2026-46242 (0)

- Explication complète sur le github de J-jaeyoung: bad-epoll
- Bug kernel linux sur toutes les versions du 2023-04-08 au 2026-04-24
- L'API `epoll` est vulnérable à un UAF (Use After Free)
- Race condition sur une application multithreadée
- Une chaîne ROP dans le kernel pour une escalade de privilège en post-exploit

En pratique: CVE-2026-46242 (1)

■ Flût de l'exploitation de la vuln



En pratique: CVE-2026-46242 (2)

■ Principes généraux:

- Race condition sur deux free concurrent du même objet
- Le premier thread détruit l'objet pendant que le second écrit
- UAF du premier permet de faire une écriture 8 octet dans la structure traversée dans le second
- La `struct file` est contrôlée par complètement par le thread attaquant
- Pointeur sur fonction dans cette struct, déréférencé dans la fonction `vfg_poll`:
`return file->f_op->poll(file, pt);`
- Contrôle du flot d'exécution à partir de l'écriture de poll par le premier gadget/pivot
- Registre `rdx` maîtrisé: `file->f_count + 1`
- Cette valeur est utilisé pour pointée sur une page mémoire maîtrisée

■ Prérequis: Fuite de l'offset du KASLR (side-channel)

■ Lecture du writeup complet fortement recommandé pour les intéressés

- Deux registres maîtrisés: RIP et RDX
- La stack est inconnue et non maîtrisée
- Écrire une chaîne sans utiliser la stack est non trivial
 - Gestion d'une FSM
 - Vérification de la cohérence des offsets
- Idéalement: écraser RSP pour maîtriser la stack $\Rightarrow$ besoin d'un pivot de stack

- Deux registres maîtrisés: RIP et RDX
- La stack est inconnue et non maîtrisée
- Écrire une chaîne sans utiliser la stack est non trivial
 - Gestion d'une FSM
 - Vérification de la cohérence des offsets
- Idéalement: écraser RSP pour maîtriser la stack $\Rightarrow$ besoin d'un pivot de stack
- Le pivot est présenté dans l'ordre d'exécution
- Ce n'est pas forcément (rarement) l'ordre d'écriture

- Deux registres maîtrisés: RIP et RDX
- La stack est inconnue et non maîtrisée
- Écrire une chaîne sans utiliser la stack est non trivial
 - Gestion d'une FSM
 - Vérification de la cohérence des offsets
- Idéalement: écraser RSP pour maîtriser la stack $\Rightarrow$ besoin d'un pivot de stack
- Le pivot est présenté dans l'ordre d'exécution
- Ce n'est pas forcément (rarement) l'ordre d'écriture
- On a une page de mémoire allouée maîtrisée dans l'espace kernel: `virt`
- On essaie d'écrire l'adresse de notre nouvelle stack dans RSP

The pre-Rop Chain: stack pivot

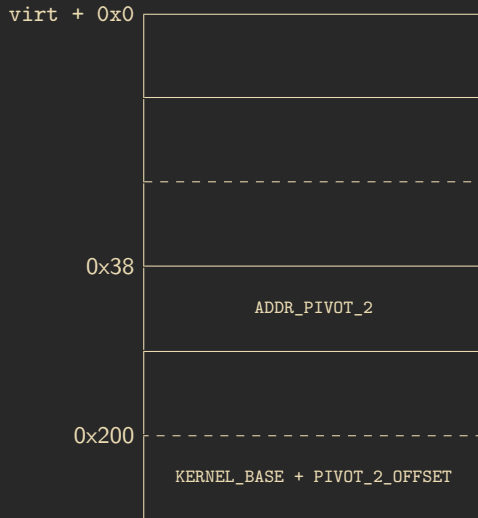
- Premier gadget: copie de RDX dans RDI
- RDX contient `virt`

```
mov rax, [rdx+0x38]
```

```
mov rdi, rdx
```

```
mov rax, [rax]
```

```
call rax
```



The pre-Rop Chain: stack pivot

- Deuxième gadget: copie de RDI dans RBX
- Même type de gadget

```
mov rax,[rdi]
mov rbx, rdi
mov rax, [rax+0x148]
call rax
```



The pre-Rop Chain: stack pivot

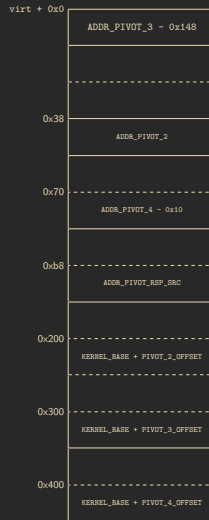
- Troisième gadget: Placer `virt + ADDR_RSP_SRC` dans `RCX`
- `RDI == RBX == virt`

```
mov rax,[rdi+0x70]
```

```
mov rcx, [rdi+0xb8]
```

```
mov rax, [rax+0x10]
```

```
jmp rax
```



The pre-Rop Chain: stack pivot

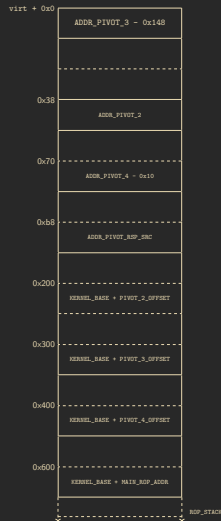
- Quatrième gadget: Pivot de Stack, et suite de la chaîne
- $RCX == virt + ADDR_RSP_SRC$; $RBX == virt$

```
push [rcx]
```

```
rcr byte [rbx+5d], 0x41
```

```
pop rsp
```

```
ret
```



- Après un stack pivot, RSP est maintenant maîtrisé
- Une chaîne ROP simple peut être utilisée
- Objectif:
 - `commit_creds(&init_cred)`
 - `switch_task_namespaces(find_task_by_vpid(1), &init_nsproxy)`
 - `ret2usr =>` retourner dans le User Space
 - `execve("/bin/sh");`
- "Left as an exercise" (le poc complet est dispo sur le github de J-jaeyoung)

Quelles protections ?

■ Protection logiciel type CFI

- `clang --fsanitize=cfi-*` (besoin de LTO)
- Des compilateurs spécifiques (GFree, ...)
- Kernel CFI in GCC/Clang (`-fsanitize=kcfi`) (2022 CLang, 2026 GCC)

■ Des protections matérielles

- Intel Control Flow Enforcement Technology
 - ▶ Shadow stack
 - ▶ Instruction `endbr64`
- Aarch64
 - ▶ Branch Target Instruction (`bti c`; `bti j`)
 - ▶ Pointer Authentication Codes (PAC) (`paciasp`; `autiasp`)