

# 補助ベクトルとプロセスのロード

---

Akira Kawata

bio: <https://akawashiro.github.io/>

twitter: [@a\\_kawashiro](https://twitter.com/a_kawashiro)

2022/08/28 [Kernel/VM探検隊online part5](#)

## Linuxにおいてプロセスに渡される(広義の)引数

- ユーザ引数 (`ls -alh`)
- 環境変数 (`PYTHONPATH=. python3 hoge.py`)

```
> cat main.c
```

```
int main(int argc, char* const argv[], char** envp) {  
return 0; }
```

```
> gcc main.c -static
```

```
> ./a.out
```

# Linuxにおいてプロセスに渡される(広義の)引数

- ユーザ引数 (`ls -alh`)
- 環境変数 (`PYTHONPATH=. python3 hoge.py`)

```
> cat main.c
```

```
int main(int argc, char* const argv[], char** envp) {  
return 0; }
```

```
> gcc main.c -static
```

ユーザ引数

環境変数

```
> ./a.out
```

## 補助ベクトル(Auxiliary Vector)

- カーネルがプロセスに渡す(広義の)引数
  - ハードウェアの情報 (AT\_HWCAP)
  - メモリ上のプロセスの位置 (AT\_BASE)
  - セキュリティ目的で使うランダムな値へのポインタ (AT\_RANDOM)
- [getauxval\(3\)](#) で取得できる

# 補助ベクトルがどのようにプロセスに渡されるか

position	content	size (bytes) + comment
stack pointer ->	[ argc = number of args ]	4
	[ argv[0] (pointer) ]	4 (program name)
	[ argv[1] (pointer) ]	4
	[ argv[..] (pointer) ]	4 * x
	[ argv[n - 1] (pointer) ]	4
	[ argv[n] (pointer) ]	4 (= NULL)
	[ envp[0] (pointer) ]	4
	[ envp[1] (pointer) ]	4
	[ envp[..] (pointer) ]	4
	[ envp[term] (pointer) ]	4 (= NULL)
	[ auxv[0] (Elf32_auxv_t) ]	8
	[ auxv[1] (Elf32_auxv_t) ]	8
	[ auxv[..] (Elf32_auxv_t) ]	8
	[ auxv[term] (Elf32_auxv_t) ]	8 (= AT_NULL vector)
	[ padding ]	0 - 16
	[ argument ASCIIZ strings ]	>= 0
	[ environment ASCIIZ str. ]	>= 0
(0xbfffffff)	[ end marker ]	4 (= NULL)
(0xc0000000)	< bottom of stack >	0 (virtual)

ref: <http://articles.manugarg.com/aboutelfauxiliaryvectors.html>

## 最近ローダを作っています

- <https://github.com/akawashiro/sloader>
- ローダとは
  - プロセスをメモリにロードする
  - ユーザ引数、環境変数、補助ベクトルの準備
- sloaderでは補助ベクトルの初期化をサボっていた
  - 「補助」というぐらいだし要らないのでは？
- アセンブラオンリーのhello worldは起動できた
- libcを静的リンクしたhello worldが起動できない

# デモ

動画撮るの疲れるのでパラパラ漫画で

```
[@goshun] (dev2~9)~/sloader/build
> cat ../hello.asm
section .text
    global _start

section .data
msg db 'Hello, world!',0xa ;our dear string
len equ $ - msg           ;length of our dear string

section .text

_start:
    mov edx,len ;message length
    mov ecx,msg ;message to write
    mov ebx,1   ;file descriptor (stdout)
    mov eax,4   ;system call number (sys_write)
    int 0x80   ;call kernel
    mov ebx,0   ;process' exit code
    mov eax,1   ;system call number (sys_exit)
    int 0x80   ;call kernel - this interrupt won't return
```

```
[@goshun] (dev2~9)~/sloader/build
```

```
> █
```



```
section .data
msg db 'Hello, world!',0xa ;our dear string
len equ $ - msg           ;length of our dear string

section .text

_start:
    mov edx,len ;message length
    mov ecx,msg ;message to write
    mov ebx,1   ;file descriptor (stdout)
    mov eax,4   ;system call number (sys_write)
    int 0x80    ;call kernel
    mov ebx,0   ;process' exit code
    mov eax,1   ;system call number (sys_exit)
    int 0x80    ;call kernel - this interrupt won't return
```

```
[@goshun] (dev2~9)~/sloader/build
```

```
> cat ../hello.c
```

```
#include <stdio.h>
```

```
int main() {
    puts("Hello from libc!\n");
    return 0;
}
```

```
[@goshun] (dev2~9)~/sloader/build
```

```
>
```

```
section .text
```

```
_start:
```

```
    mov edx,len ;message length
    mov ecx,msg ;message to write
    mov ebx,1   ;file descriptor (stdout)
    mov eax,4   ;system call number (sys_write)
    int 0x80    ;call kernel
    mov ebx,0   ;process' exit code
    mov eax,1   ;system call number (sys_exit)
    int 0x80    ;call kernel - this interrupt won't return
```

```
[@goshun] (dev2~9)~/sloader/build
```

```
> cat ../hello.c
```

```
#include <stdio.h>
```

```
int main() {
    puts("Hello from libc!\n");
    return 0;
}
```

```
[@goshun] (dev2~9)~/sloader/build
```

```
> ninja
```

```
[1/1] cd /home/akira/sloader/build && nasm -f elf64...sloader/build/hello.o && ld -s -o hello_asm hello.o
```

```
[@goshun] (dev2~9)~/sloader/build
```

```
>
```

```
mov ecx,msg ;message to write
mov ebx,1   ;file descriptor (stdout)
mov eax,4   ;system call number (sys_write)
int 0x80    ;call kernel
mov ebx,0   ;process' exit code
mov eax,1   ;system call number (sys_exit)
int 0x80    ;call kernel - this interrupt won't return
```

```
[@goshun] (dev2~9)~/sloader/build
```

```
> cat ../hello.c
```

```
#include <stdio.h>
```

```
int main() {
    puts("Hello from libc!\n");
    return 0;
}
```

```
[@goshun] (dev2~9)~/sloader/build
```

```
> ninja
```

```
[1/1] cd /home/akira/sloader/build && nasm -f elf64...sloader/build/hello.o && ld -s -o hello_asm hello.o
```

```
[@goshun] (dev2~9)~/sloader/build
```

```
> ls
```

```
build.ninja      cmake_install.cmake      CTestTestfile.cmake    hello_static_libc*
CMakeCache.txt  compile_commands.json    hello_asm*              sloader*
CMakeFiles/     CTestCustom.cmake        hello.o
```

```
[@goshun] (dev2~9)~/sloader/build
```

```
>
```

```
> ls
build.ninja      cmake_install.cmake    CTestTestfile.cmake  hello_static_libc*
CMakeCache.txt  compile_commands.json  hello_asm*            loader*
CMakeFiles/     CTestCustom.cmake     hello.o
[@goshun] (dev2~9) ~/sloader/build
> ./sloader --load hello_asm 2>&1 | tail
/home/akira/sloader/sloader.cc:116 p=0x00000000040000 head() + ph->p_offset=0x00007FAE692F7000 ph->p_filesz=0x000000000000000E8
/home/akira/sloader/sloader.cc:108 reinterpret_cast<void*>(ph->p_vaddr)=0x00000000x401000 ph->p_memsz=0x0000000000000022
/home/akira/sloader/sloader.cc:114 mmap: filename()=hello_asm p=0x000000000401000 ph->p_vaddr=0x0000000000401000
/home/akira/sloader/sloader.cc:116 p=0x000000000401000 head() + ph->p_offset=0x00007FAE692F8000 ph->p_filesz=0x0000000000000022
/home/akira/sloader/sloader.cc:108 reinterpret_cast<void*>(ph->p_vaddr)=0x00000000x402000 ph->p_memsz=0x000000000000000E
/home/akira/sloader/sloader.cc:114 mmap: filename()=hello_asm p=0x000000000402000 ph->p_vaddr=0x0000000000402000
/home/akira/sloader/sloader.cc:116 p=0x000000000402000 head() + ph->p_offset=0x00007FAE692F9000 ph->p_filesz=0x000000000000000E
/home/akira/sloader/sloader.cc:121 Load end
/home/akira/sloader/sloader.cc:133 Execute startfilename()=hello_asm
Hello, world!
[@goshun] (dev2~9) ~/sloader/build
>
```

```
[@goshun] (dev2~9) ~/sloader/build
> ./sloader --load hello_static_libc 2>&1 | tail
/home/akira/sloader/sloader.cc:116 p=0x000000000400000 head() + ph->p_offset=0x00007FEB50B03000 ph->p_fi
lesz=0x00000000000000528
/home/akira/sloader/sloader.cc:108 reinterpret_cast<void*>(ph->p_vaddr)=0x00000000x401000 ph->p_memsz=0x
000000000000966BD
/home/akira/sloader/sloader.cc:114 mmap: filename()=hello_static_libc p=0x000000000401000 ph->p_vaddr=0x
0000000000401000
/home/akira/sloader/sloader.cc:116 p=0x000000000401000 head() + ph->p_offset=0x00007FEB50B04000 ph->p_fi
lesz=0x000000000000966BD
/home/akira/sloader/sloader.cc:108 reinterpret_cast<void*>(ph->p_vaddr)=0x00000000x498000 ph->p_memsz=0x
000000000000284EC
/home/akira/sloader/sloader.cc:114 mmap: filename()=hello_static_libc p=0x000000000498000 ph->p_vaddr=0x
0000000000498000
/home/akira/sloader/sloader.cc:116 p=0x000000000498000 head() + ph->p_offset=0x00007FEB50B9B000 ph->p_fi
lesz=0x000000000000284EC
/home/akira/sloader/sloader.cc:108 reinterpret_cast<void*>(ph->p_vaddr)=0x00000000x4c17b0 ph->p_memsz=0x
000000000000B490
/home/akira/sloader/sloader.cc:114 mmap: filename()=hello_static_libc p=0x0000000004C1000 ph->p_vaddr=0x
00000000004C17B0
/home/akira/sloader/sloader.cc:115 ph->p_vaddr != reinterpret_cast<Elf64_Addr>(p)
zsh: IOT instruction (core dumped) ./sloader --load hello_static_libc 2>&1 |
zsh: done tail
[@goshun] (dev2~9) ~/sloader/build
```

```
>
```

デモ終わり

## gdbで見る

```
Program received signal SIGSEGV, Segmentation fault.
```

```
0x0000000000402535 in ?? ()
```

```
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
```

```
RAX 0x0
```

```
RBX 0x400518 ← 0
```

0x0から読もうとしている

```
▶ 0x402535    movq    (%rax), %rax
0x402538    xorb   %al, %al
0x40253a    movq   %rax, %fs:0x28
0x402543    cmpl   $0, 0xbf2de(%rip)
```

## 0x402535周りをobjdumpで見る

```
40252e:  mov 0xbd52b(%rip),%rax # 4bfa60 <_dl_random>
402535:  mov (%rax),%rax
402538:  30 c0  xor %al,%al
```

\_dl\_randomが初期化されていない



## \_dl\_randomはどこで初期化される

- glibcの中でAT\_RANDOMの値が書き込まれる
- [dl-support.c#L316-L318](#)

```
case AT_RANDOM:
```

```
    _dl_random = (void *) av->a_un.a_val;
```

```
    break;
```

## なぜ\_dl\_randomが必要なのか？

- [libc-start.c#L331-L333](#)
- スタックガードで使われる
- スタックガードとは
  - スタックオーバーフロー攻撃を防ぐための手段
  - スタックの終わりにランダムな値を書き込んでおく
  - スタックを超えて書き込みがあると値が変わるので気づく

# それカーネルから渡す必要ある？

## 経緯

1. glibc側からスタックガード用にランダムなバイト列が欲しい
2. 最初は/dev/urandomを使おうとしていた
3. プロセスを起動するたびに/dev/urandomを開くのは高価
4. カーネルに実装される

- [Re: \[PATCH\] get\\_random\\_long\(\) and AT\\_ENTROPY for auxv, kernel 2.6.21.5](#)
- [randomized stack protector value](#)
- [ELF: implement AT\\_RANDOM for future glibc use](#)

ローダを実装するとき  
は補助ベクトルもちゃ  
んと実装しないと謎の  
SEGVで苦しむよ