eBPF 를 활용한
memory leak sanitizer

Attachable leak sanitizer

About me

저는 memory leak 문제를 해결하는 일에 진심인 LG전자 개발자 입니다.

Github: https://github.com/Bojun-Seo

Contents

- Motivation
- About eBPF and BCC
- Attachable leak sanitizer

Contents

- **Motivation**
- About eBPF and BCC
- Attachable leak sanitizer

What is memory leak?

- Memory reserved but not in use

What is memory leak?

- Memory reserved but not in use
    - Causes system memory shortage

What is memory leak?

- Memory reserved but not in use
    - Causes system memory shortage
    - Which derives “malloc” to be failed

What is memory leak?

- Memory reserved but not in use
    - Causes system memory shortage
    - Which derives “malloc” to be failed
    - Some applications cannot be executed properly

Memory leak classification

Memory leak

Memory leak classification

Memory leak

Dangling pointer

valgrind
address sanitizer(asan)
leak sanitizer(lsan)

Valgrind, asan and lsan

- Valgrind
  - 자체적인 가상 머신 위에서 프로그램을 실행시키면서 다양한 메모리 문제를 검출
  - 오버헤드가 가장 크지만, 재컴파일이 필요 없음
- Address sanitizer(asan)
  - llvm-project 의 sanitizer 중 하나
  - memory leak 뿐만 아니라 다양한 memory 문제를 검출할 수 있음
  - 재컴파일 필수
  - overhead 가 valgrind 보다는 작지만, 임베디드 환경에서 사용하기 어려움
- Leak sanitizer(lsan)
  - library preloading 방식으로 memory leak 만 검출
  - preloading 이므로 재컴파일 하지 않고도 사용할 수 있음(다만, 설정을 따로 해야 함)
  - overhead 가 적음

lsan user requirements

- 재컴파일 하지 않고 report 를 얻고 싶다.
- Daemon, service 와 같이 종료하지 않는 프로세스에 대한 report 를 얻고 싶다.

lsan user requirements

- 재컴파일 하지 않고 report 를 얻고 싶다.
- Daemon, service 와 같이 종료하지 않는 프로세스에 대한 report 를 얻고 싶다.
- **프로그램을 재시작 하지 않고 report 를 얻고 싶다.**

How we debug?

- Check the log
- Reproduce the issue

How we debug?

- Check the log
- Reproduce the issue
    - 재현을 위한 비용(시간, 금전)이 큰 경우(재현률이 낮은 경우)
    - 디버깅 툴을 붙이니 재현이 되지 않는 경우

lsan user requirements

- 재컴파일 하지 않고 report 를 얻고 싶다.
- Daemon, service 와 같이 종료하지 않는 프로세스에 대한 report 를 얻고 싶다.
- **프로그램을 재시작 하지 않고 report 를 얻고 싶다. → attachable lsan**

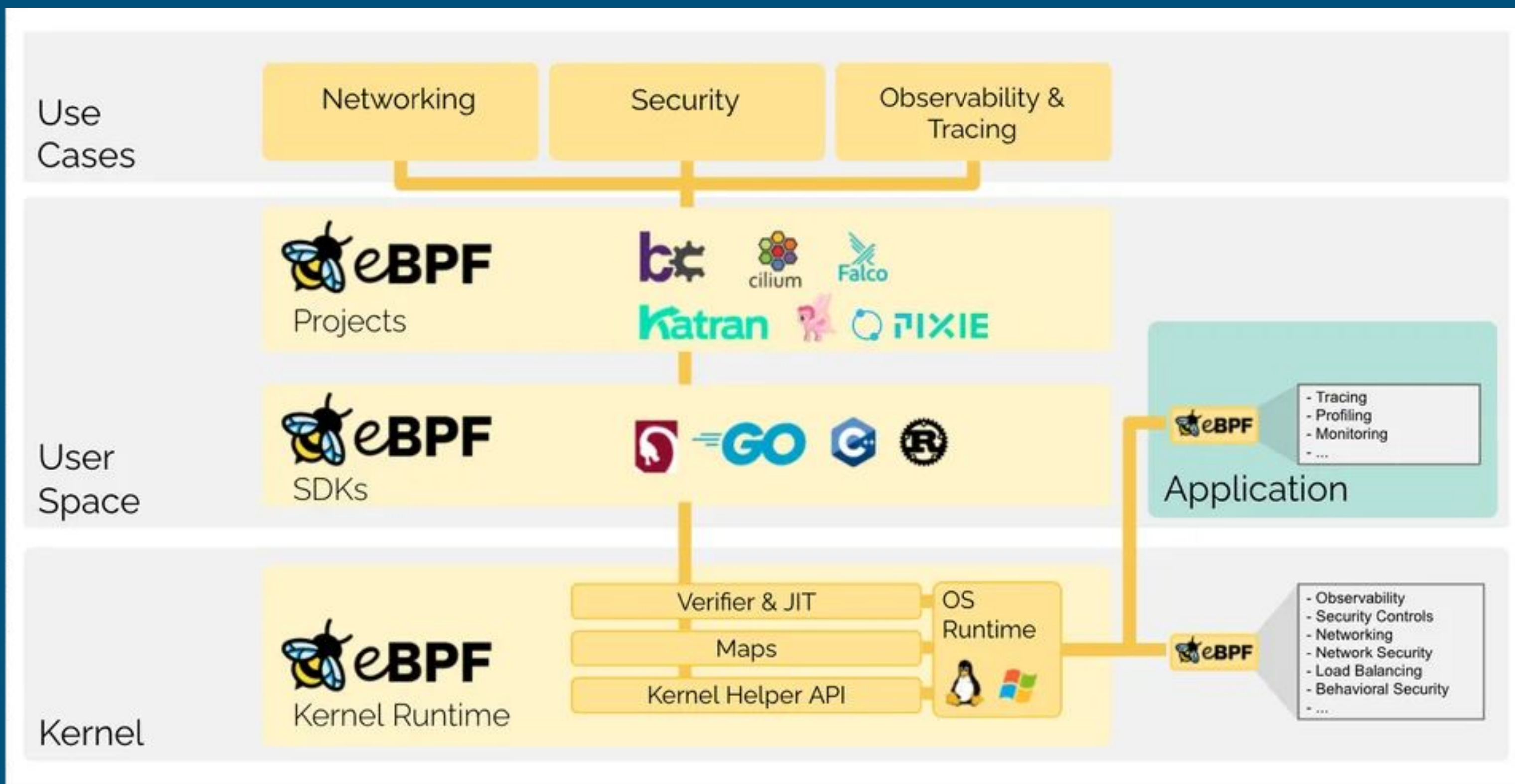
Contents

- Motivation
- **About eBPF and BCC**
- Attachable leak sanitizer

eBPF

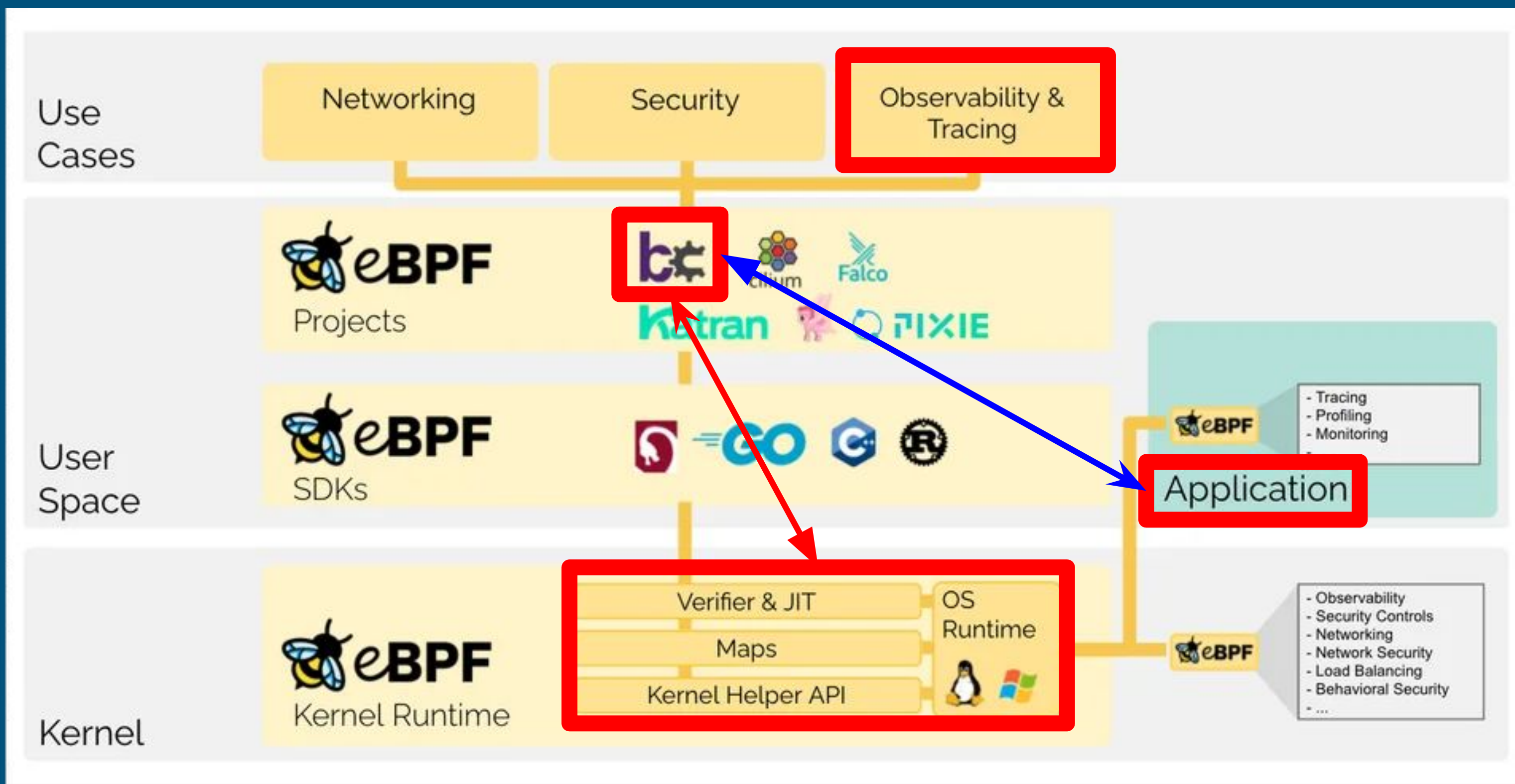
- BPF(Berkeley Packet Filter) 란?
  - 네트워크 메시지 필터링을 위한 리눅스 커널 내부 가상 머신
- BPF 를 확장(eBPF: extended BPF)
  - 가상 머신 강화(JIT, 레지스스터 크기 증가, 등)
  - User defined code 가 kernel 내부에서 실행할 수 있게 됨

eBPF use cases



Source: https://ebpf.io/static/e293240ecccb9d506587571007c36739/f2674/overview.png

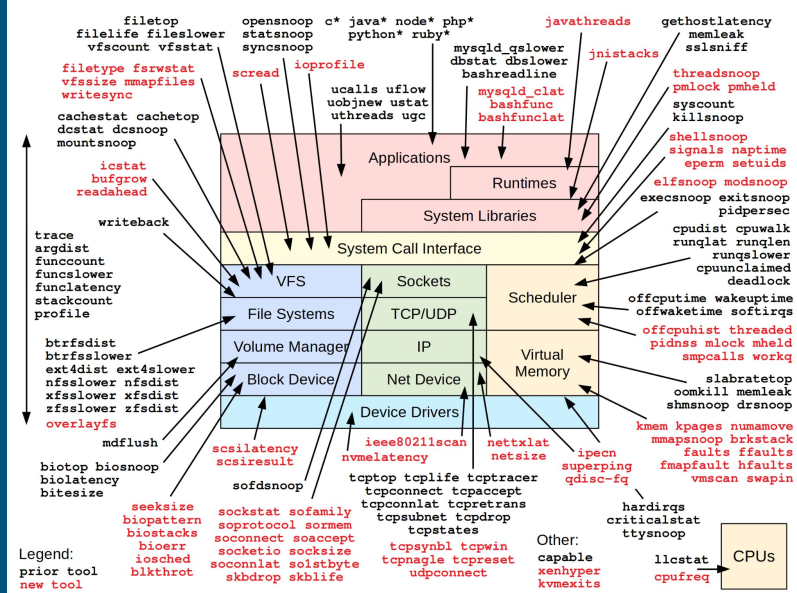
eBPF use cases



Source: https://ebpf.io/static/e293240ecccb9d506587571007c36739/f2674/overview.png

BCC(BPF Compiler Collection)

- 효율적인 커널 추적 및 조작 프로그램을 만들기 위한 toolkit
- 유용한 도구와 예제가 포함되어 있음
- https://github.com/iovisor/bcc



Source: https://www.brendangregg.com/BPF/bpf_performance_tools_book.png

Contents

- Motivation
- About eBPF and BCC
- **Attachable leak sanitizer**

Attachable leak sanitizer

- https://github.com/iovisor/bcc/pull/4120
- 재컴파일 필요 없음
- Daemon, service 와 같이 종료하지 않는 프로세스에 대한 report 를 얻을 수 있음
- **프로그램을 재시작 하지 않고 report 를 얻을 수 있음**
    - **단, attach 한 이후의 report 만 얻을 수 있음**

Attachable lsan 의 동작

- UPROBE 를 이용하여 user memory (de)allocator 함수 진출입을 hooking
- eBPF kernel runtime 에서 allocation 정보를 bpf map 에 저장/삭제
- Attachable lsan user process 에서 bpf map 에 있는 정보를 읽어옴
- 읽어온 정보를 바탕으로 dangling pointer 여부를 판별

Target user process

```
void* malloc(size_t size)
{
   // do something
   return ptr;
}
```

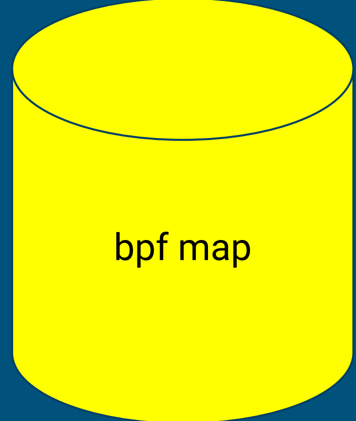
Kernel

eBPF

```
void on_allloc_enter()
{
   // save size
}
```

eBPF

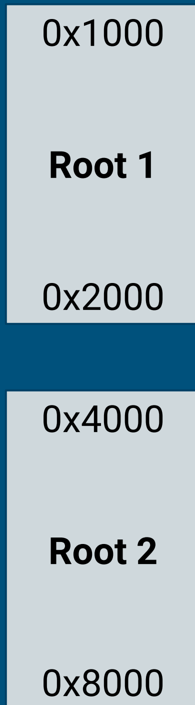
```
void on_allloc_exit()
{
   // get stack backtrace
   // save information on bpf map
   // key: ptr, value: (size, backtrace)
}
```

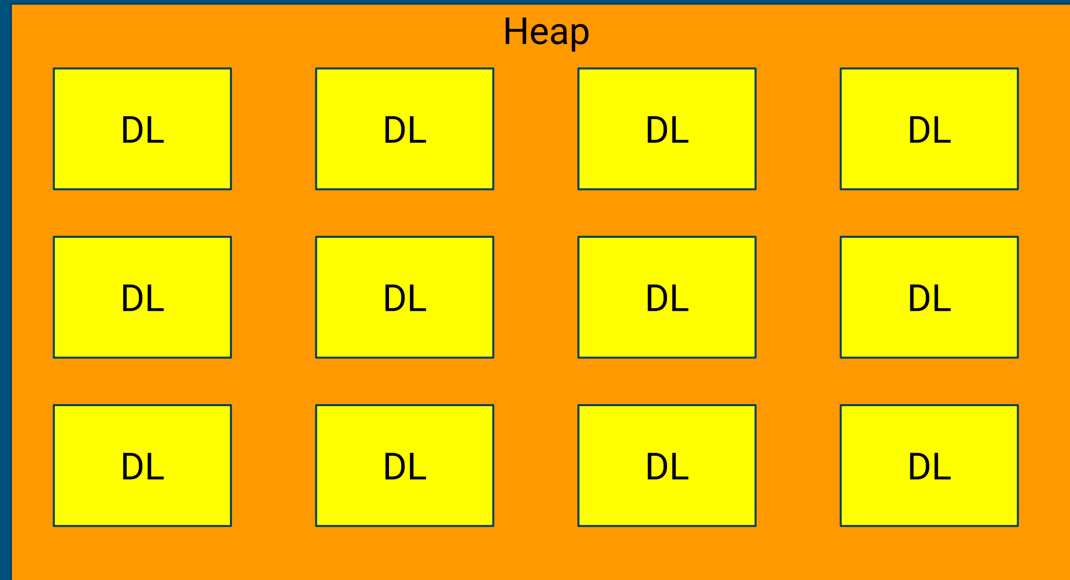


Attachable lsan user process

Run dangling pointer
detecting algorithm

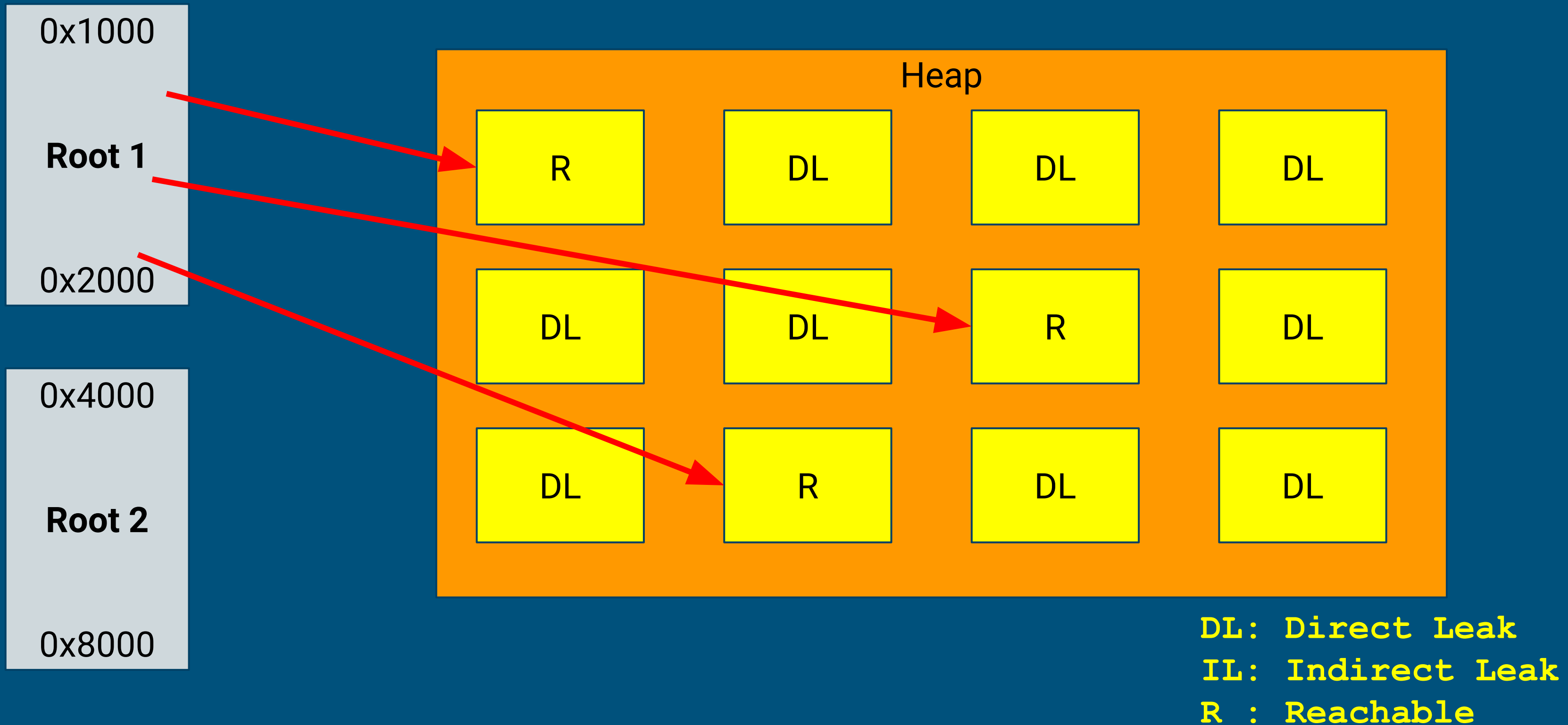
Dangling pointer detecting algorithm



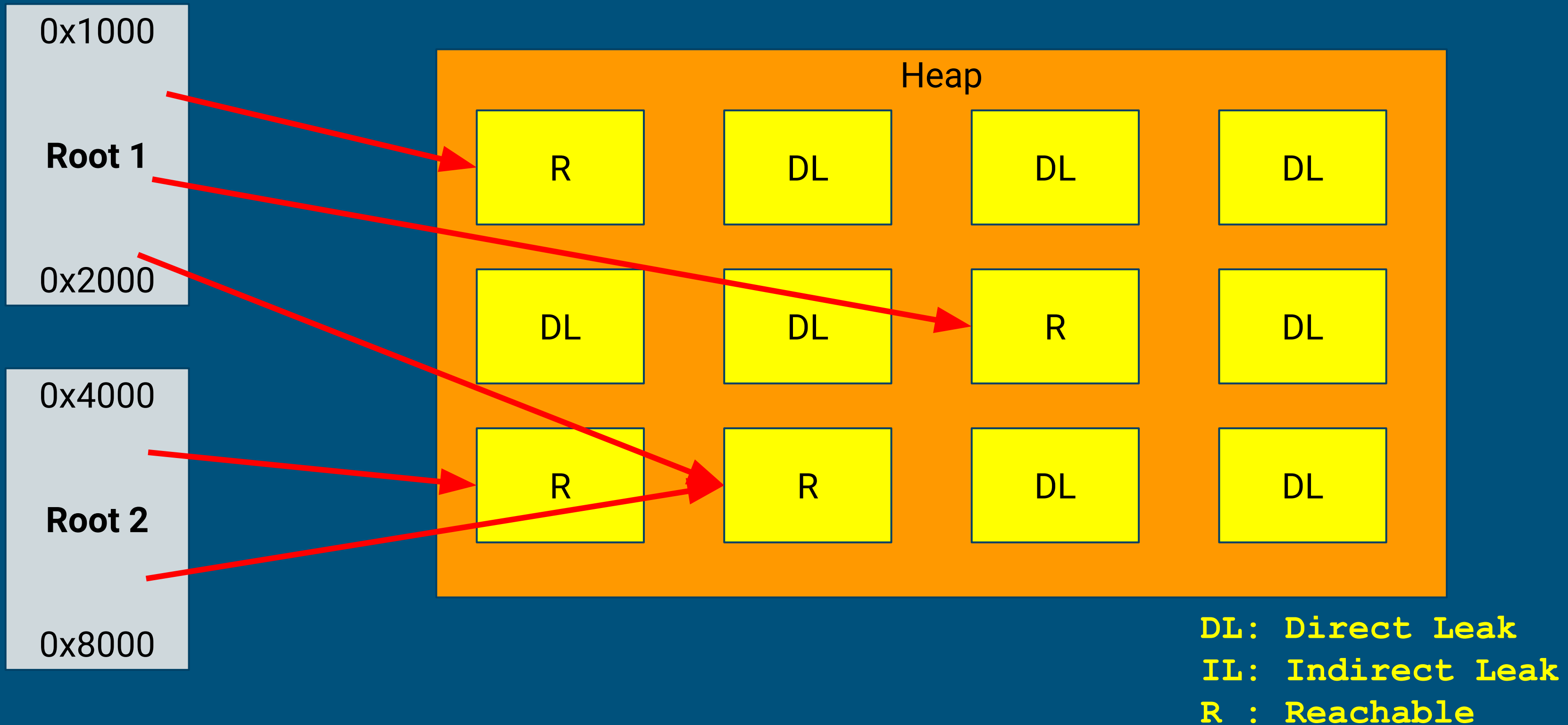


DL: Direct Leak
IL: Indirect Leak
R : Reachable

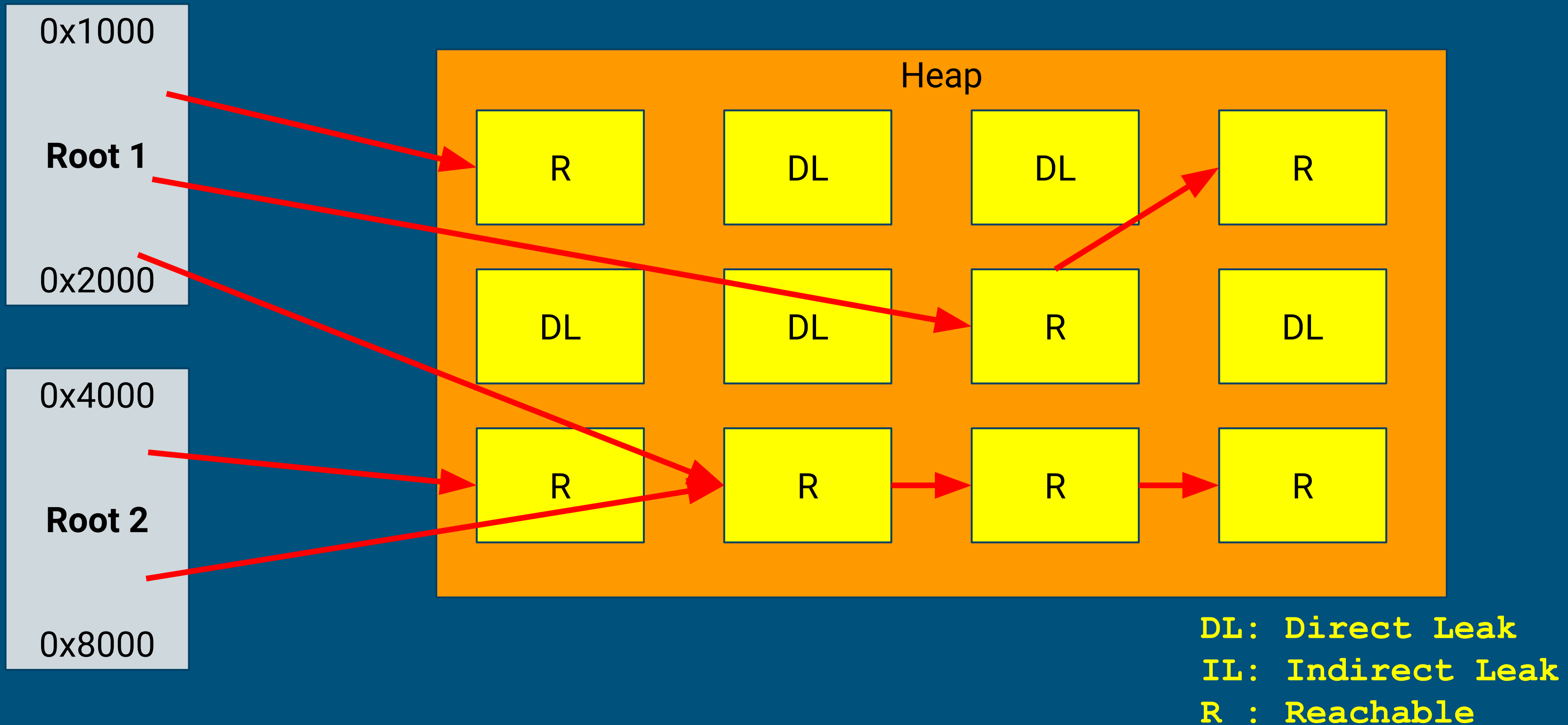
Dangling pointer detecting algorithm



Dangling pointer detecting algorithm

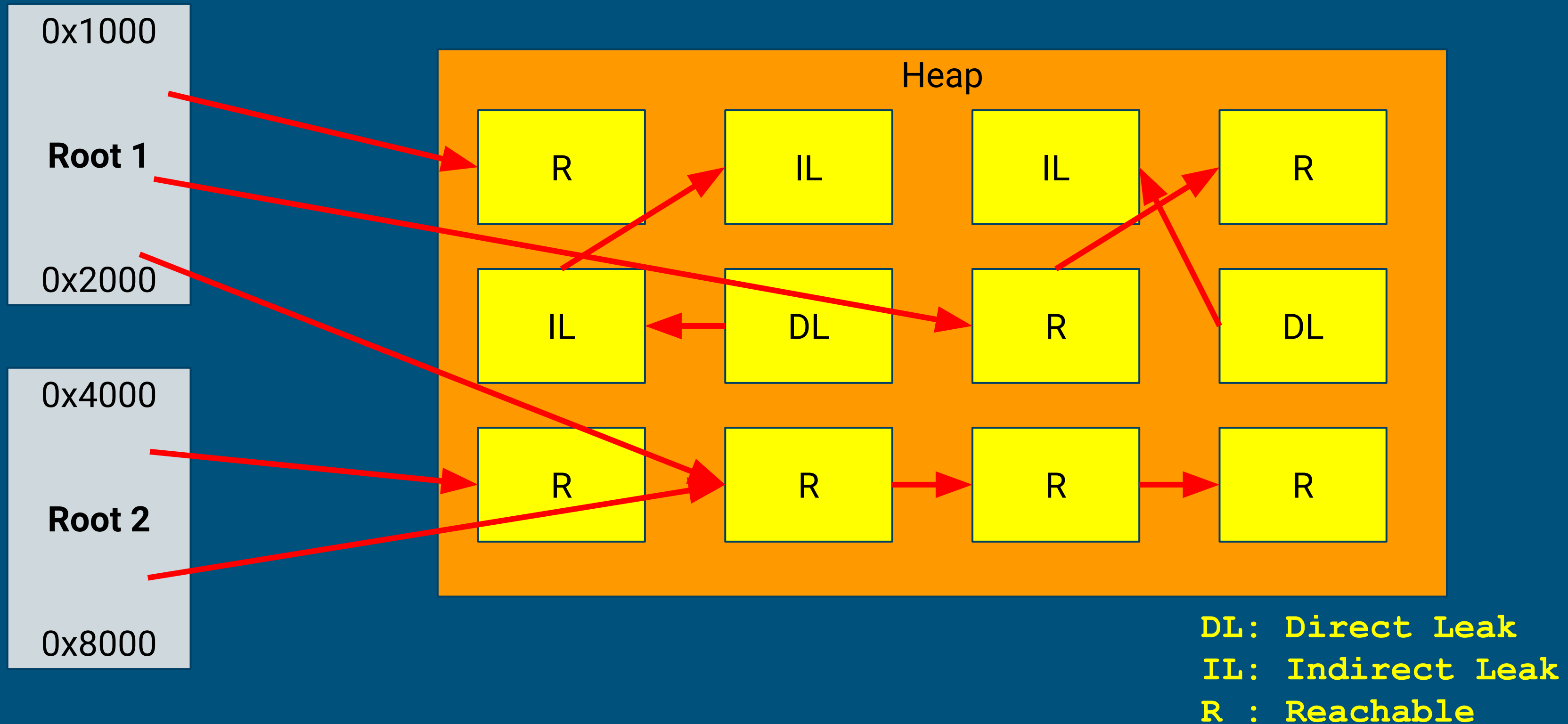


Dangling pointer detecting algorithm



DL: Direct Leak
IL: Indirect Leak
R : Reachable

Dangling pointer detecting algorithm



Demo

```
# original leak sanitizer usage
```

```
# original leak sanitizer usage
$ cat leak.c
#include <stdlib.h>
int main(int argc, char* argv[]) {
  int* a = (int*)malloc(sizeof(int));
  return 0;
}
$
```

```
# original leak sanitizer usage
$ cat leak.c
#include <stdlib.h>
int main(int argc, char* argv[]) {
  int* a = (int*)malloc(sizeof(int));
  return 0;
}
$ clang leak.c -fsanitize=leak
$
```

```
# original leak sanitizer usage
$ cat leak.c
#include <stdlib.h>
int main(int argc, char* argv[]) {
  int* a = (int*)malloc(sizeof(int));
  return 0;
}
$ clang leak.c -fsanitize=leak
$ ./a.out
=================================================================
==136151==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 4 byte(s) in 1 object(s) allocated from:
    #0 0x429518 in __interceptor_malloc (/home/worker/test/a.out+0x429518)
    #1 0x42b8cf in main (/home/worker/test/a.out+0x42b8cf)
    #2 0x7f47f13bbd8f  (/lib/x86_64-linux-gnu/libc.so.6+0x29d8f)

SUMMARY: LeakSanitizer: 4 byte(s) leaked in 1 allocation(s).
$
```

Attachable leak sanitizer usage

```
# Attachable leak sanitizer usage

~$ cat leak_daemon1.c
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char* argv[]) {
  while (1) {
    int* a = (int*)malloc(sizeof(int));
    sleep(1);
  }
  return 0;
}
~$
```

```
# Attachable leak sanitizer usage

~$ cat leak_daemon1.c
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char* argv[]) {
  while (1) {
    int* a = (int*)malloc(sizeof(int));
    sleep(1);
  }
  return 0;
}
~$ gcc leak_daemon1.c
~$
```

Attachable leak sanitizer usage

```
~$ cat leak_daemon1.c
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char* argv[]) {
  while (1) {
    int* a = (int*)malloc(sizeof(int));
    sleep(1);
  }
  return 0;
}
~$ gcc leak_daemon1.c
~$ ./a.out &
[1] 84150
~$
```

```
# Attachable leak sanitizer usage

~$ sudo ./lsan -p 84150
Warn: Failed to open: /usr/etc/suppr.txt
[2023-09-07 05:54:01] Print leaks:
40 bytes direct leak found in 10 allocations from stack id(24244)
  #1 0x0055601e3a6186 main+0x1d (/home/bojun/a.out+0x1186)
  #2 0x007fbba9823510 __libc_init_first+0x90 (/usr/lib/libc.so.6+0x23510)

[2023-09-07 05:54:11] Print leaks:
80 bytes direct leak found in 20 allocations from stack id(24244)
  #1 0x0055601e3a6186 main+0x1d (/home/bojun/a.out+0x1186)
  #2 0x007fbba9823510 __libc_init_first+0x90 (/usr/lib/libc.so.6+0x23510)

^C
~$
```

Questions?

Way to build and run attachable lsan on Ubuntu 22.04.3

```
# Way to build and run attachable lsan on Ubuntu 22.04.3
# Need to install some packages
```

```
# Way to build and run attachable lsan on Ubuntu 22.04.3
# Need to install some packages
~$ sudo apt install git cmake libclang-dev libelf-dev llvm clang
```

```
# Way to build and run attachable lsan on Ubuntu 22.04.3
# Need to install some packages
~$ sudo apt install git cmake libclang-dev libelf-dev llvm clang

~$ git clone https://github.com/Bojun-Seo/bcc.git -b lsan
~$
```

```
# Way to build and run attachable lsan on Ubuntu 22.04.3
# Need to install some packages
~$ sudo apt install git cmake libclang-dev libelf-dev llvm clang

~$ git clone https://github.com/Bojun-Seo/bcc.git -b lsan
~$ cd bcc/
~/bcc$
```

```
# Way to build and run attachable lsan on Ubuntu 22.04.3
# Need to install some packages
~$ sudo apt install git cmake libclang-dev libelf-dev llvm clang

~$ git clone https://github.com/Bojun-Seo/bcc.git -b lsan
~$ cd bcc/
~/bcc$ mkdir build
~/bcc$
```

```
# Way to build and run attachable lsan on Ubuntu 22.04.3
# Need to install some packages
~$ sudo apt install git cmake libclang-dev libelf-dev llvm clang

~$ git clone https://github.com/Bojun-Seo/bcc.git -b lsan
~$ cd bcc/
~/bcc$ mkdir build
~/bcc$ cd build/
~/bcc/build$
```

```
# Way to build and run attachable lsan on Ubuntu 22.04.3
# Need to install some packages
~$ sudo apt install git cmake libclang-dev libelf-dev llvm clang

~$ git clone https://github.com/Bojun-Seo/bcc.git -b lsan
~$ cd bcc/
~/bcc$ mkdir build
~/bcc$ cd build/
~/bcc/build$ cmake ..
~/bcc/build$
```

```
# Way to build and run attachable lsan on Ubuntu 22.04.3
# Need to install some packages
~$ sudo apt install git cmake libclang-dev libelf-dev llvm clang

~$ git clone https://github.com/Bojun-Seo/bcc.git -b lsan
~$ cd bcc/
~/bcc$ mkdir build
~/bcc$ cd build/
~/bcc/build$ cmake ..
~/bcc/build$ cd ../libbpf-tools/
~/bcc/libbpf-tools$
```

```
# Way to build and run attachable lsan on Ubuntu 22.04.3
# Need to install some packages
~$ sudo apt install git cmake libclang-dev libelf-dev llvm clang

~$ git clone https://github.com/Bojun-Seo/bcc.git -b lsan
~$ cd bcc/
~/bcc$ mkdir build
~/bcc$ cd build/
~/bcc/build$ cmake ..
~/bcc/build$ cd ../libbpf-tools/
~/bcc/libbpf-tools$ make lsan
~/bcc/libbpf-tools$
```

```
# Attachable leak sanitizer another usage

~$ cat leak_daemon2.c
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char* argv[]) {
  while (1) {
    int* a = (int*)malloc(sizeof(int));
    free(a);
    a = calloc(1, 1);
    sleep(1);
  }
  return 0;
}
~$ gcc leak_daemon2.c
~$ ./a.out &
[1] 84357
~$
```

```
# Attachable leak sanitizer another usage

~$ sudo ./lsan -p 84357
Warn: Failed to open: /usr/etc/suppr.txt

[2023-09-07 06:14:52] Print leaks:
10 bytes direct leak found in 10 allocations from stack id(45954)
  #1 0x00559487c9f1e5 main+0x3c (/home/bojun/a.out+0x11e5)
  #2 0x007fa400223510 __libc_init_first+0x90 (/usr/lib/libc.so.6+0x23510)

[2023-09-07 06:15:02] Print leaks:
20 bytes direct leak found in 20 allocations from stack id(45954)
  #1 0x00559487c9f1e5 main+0x3c (/home/bojun/a.out+0x11e5)
  #2 0x007fa400223510 __libc_init_first+0x90 (/usr/lib/libc.so.6+0x23510)

^C
~$
```