운영체제론 실습

- ptrace (2): 프로세스 해킹

목 차

- Ptrace 설명
  - 서로 독립된 프로세스 간 추적
  - 사용할 request 목록 및 사용법
- 프로젝트: Process Hacking (Basic)

예제 코드 다운로드 경로

아래 명령어를 linux 환경에서 치면 다운받을 수 있음.

```
$ wget http://ce.hanyang.ac.kr/week8.zip
$ unzip week8.zip
```

Ptrace

사용할 request 목록

```
long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
```

request	설 명
PTRACE_ATTACH	pid로 지정한 프로세스에 붙어서 그 프로세스를 호출 프로세스의 피추적자로 만든다.
PTRACE_DETACH	정지된 피추적자를 재시작하되 먼저 그 프로세스에서 떨어진다.
PTRACE_PEEKDATA	피추적자 메모리의 주소 addr에서 워드를 읽고 그 워드를 ptrace() 호출의 결과로 반환한다.
PTRACE_POKEDATA	워드 data를 피추적자 메모리의 주소 addr로 복사한다.
PTRACE_SINGLESTEP	한 인스트럭션 단위로 피추적자의 실행을 재개한다.

Ptrace 사용법 (1)

```
ptrace(PTRACE_ATTACH, pid, 0, 0);
```

- Tracer가 **pid**에 해당되는 프로세스(Tracee)에게 **추적자로 붙을 때 사용함**
- Tracee 에게 **SIGSTOP** 시그널을 보내어 멈춤

```
ptrace(PTRACE_DETACH, pid, 0, sig);
```

- Tracer가 **Tracee로부터 떨어질 때 사용**하며, Tracee를 재개시킴
- **sig**가 **0**이면, Tracer가 시그널을 보내지 않고 떨어짐

Ptrace 사용법 (2)

ptrace(PTRACE_PEEKDATA, pid, addr, 0);

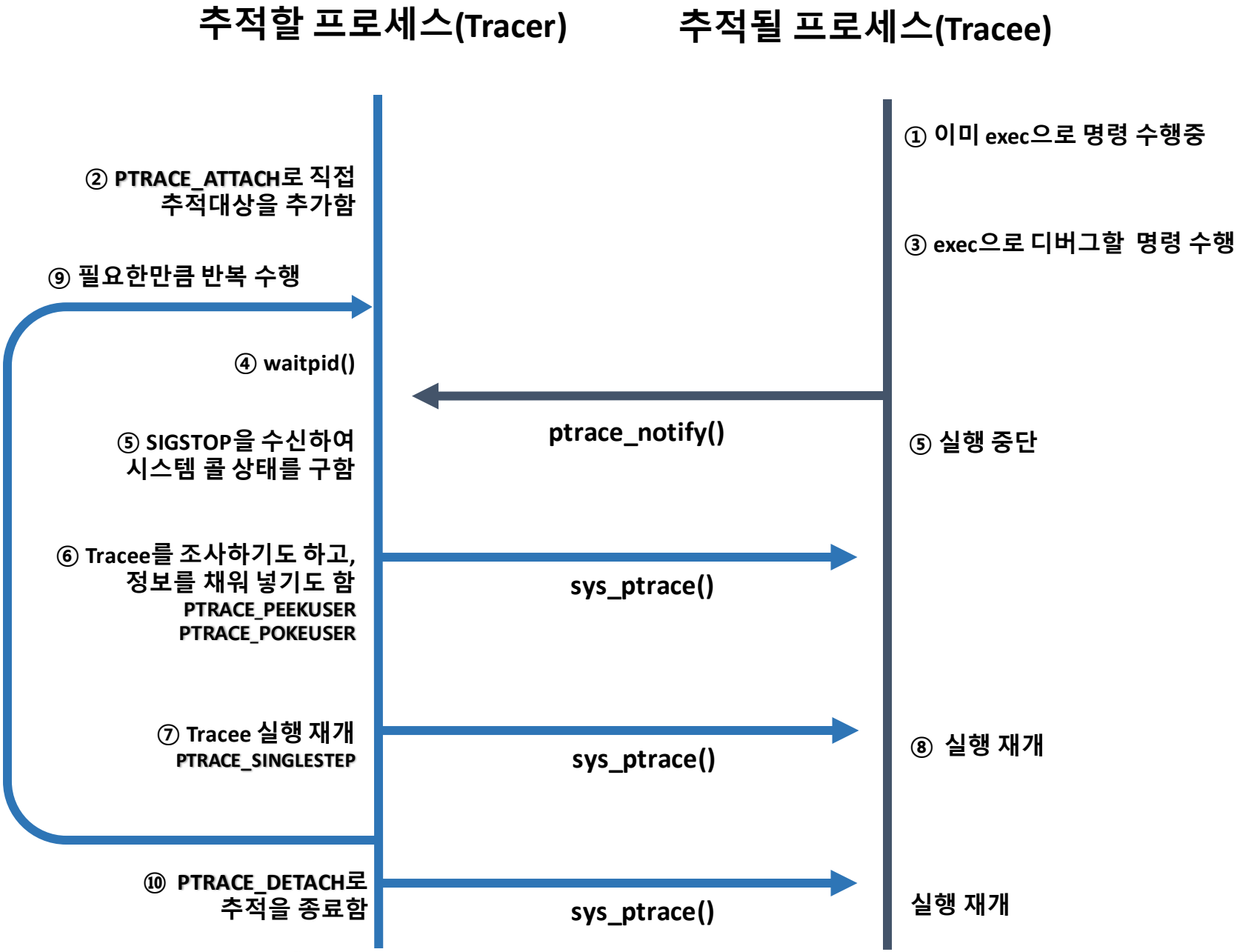
- Tracee 메모리의 주소 **addr**에서 워드를 읽음
- ptrace() 호출의 결과로 반환함. 예) **long data = ptrace(**PTRACE_PEEKDATA,...);

ptrace(PTRACE_POKEDATA, pid, addr, data);

- 워드 **data**를 Tracee 메모리의 주소 **addr**에 복사
- **data**의 타입은 **long** 값임

ptrace(PTRACE_SINGLESTEP, pid, 0, sig);

- Tracee의 실행을 재개하지만, 1개의 명령을 실행한 시점에서 실행을 정지시킴
- **sig**는 정지된 Tracee에게 보낼 시그널 번호임
  - 0이면 시그널을 보내지 않음


추적할 프로세스(Tracer)
추적될 프로세스(Tracee)
① 이미 exec으로 명령 수행중
② PTRACE_ATTACH로 직접
추적대상을 추가함
③ exec으로 디버그할 명령 수행
⑨ 필요한만큼 반복 수행
④ waitpid()
ptrace_notify()
⑤ 실행 중단
⑤ SIGSTOP을 수신하여
시스템 콜 상태를 구함
⑥ Tracee를 조사하기도 하고,
정보를 채워 넣기도 함
PTRACE_PEEKUSER
PTRACE_POKEUSER
sys_ptrace()
⑦ Tracee 실행 재개
PTRACE_SINGLESTEP
sys_ptrace()
⑧ 실행 재개
⑩ PTRACE_DETACH로
추적을 종료함
sys_ptrace()
실행 재개

프로젝트) Process Hacking (Basic)

In **tracee.c**

```c
#include <stdio.h>

int i = 0;
int my_flag = 0xabcd;

int main(void) {

  /* First step: Compare my_flag with condition(0xabcd) */
  while (my_flag == 0xabcd) {
    i++;
  }

  /* Last step: Check if the value of the flag has changed */
  char hex_flag[20];
  sprintf(hex_flag, "%x", my_flag);
  printf("Tracee exits with flag: %s\n", hex_flag);
  return i;
}
```

프로젝트) Process Hacking (Basic)

In **tracee.c**

```
 9    while (my_flag == 0xabcd) {
10      i++;
11    }
```

In **tracee.s**

```
42      jmp .L2
43  .L3:
44      .loc 1 10 0
45      movl    i(%rip), %eax
46      addl    $1, %eax
47      movl    %eax, i(%rip)
48  .L2:
49      .loc 1 9 0
50      movl    my_flag(%rip), %eax
51      cmpl    $43981, %eax
52      je  .L3
53      .loc 1 15 0
```

각 c 코드를 Assembly 코드로 변환

16진수인 0xabcd 를 10진수로 변환➔ 43981

레지스터 설명:
1. rax: 일반 목적으로 사용 (eax와 동일)
2. rip: Instruction pointer 다음 수행할 명령어 주소

50번째 줄의 movl을 수행하고 나면,
rax 레지스터는 my_flag의 값(0xabcd)을 저장함.
rip 레지스터는 다음 명령어(51번째 줄)를 가리키며,
해당 명령어 중 abcd(10진수로는 43981)값을
다른 값으로 변경하는 프로세스 해킹을 수행

프로젝트) Process Hacking (Basic)

In **tracer.c**

```
13 pid_t tracee_pid;
14 struct user_regs_struct regs;
15 int status;
16 bool keep_looping = true;
17
18 int main(int argc, char **argv) {
19
20   if (argc < 2) {
21     printf("[USAGE]: ./tracer <pid-of-target-process>\n");
22     return -1;
23   }
24   tracee_pid = atoi(argv[1]);
25   ptrace(PTRACE_ATTACH, tracee_pid, NULL, NULL);
26   waitpid(tracee_pid, &status, 0);
27
28   while (keep_looping == true) {
29     ptrace(PTRACE_SINGLESTEP, tracee_pid, NULL, NULL);
30     waitpid(tracee_pid, &status, 0);
31     ptrace(PTRACE_GETREGS, tracee_pid, NULL, &regs);
32     printf("[ CURRENT REGISTER STATE ]\n");
33     print_user_regs_struct(regs);
34     keep_looping = peekpoke_interactively(tracee_pid, regs);
35   }
36   ptrace(PTRACE_DETACH, tracee_pid, NULL, NULL);
37   return 0;
38 }
```

프로젝트) 결과화면

```
jsbaik@jsbaik: ~/OS2019/week8/process_hacking 54x55
jsbaik@jsbaik:~/OS2019/week8/process_hacking$ ./tracee

Tracee exits with flag: abcd
[1]+  죽었음                  ./tracee
jsbaik@jsbaik:~/OS2019/week8/process_hacking$
jsbaik@jsbaik:~/OS2019/week8/process_hacking$
```

생각해보기:
my_flag 변수의 값은 변경되지 않은 것을 볼 수 있다. 그렇다면 프로세스 내의 어느 메모리 영역의 값이 변경된 걸까?

프로젝트
실행 결과 확인 받은 후,
퇴실 가능

```
jsbaik@jsbaik: ~/OS2019/
jsbaik@jsbaik:~/OS2019/week8/process_hacking$ ps -C tracee
  PID TTY          TIME CMD
 1934 pts/0    00:00:21 tracee
jsbaik@jsbaik:~/OS2019/week8/process_hacking$ sudo ./tracer 1934
[ CURRENT REGISTER STATE ]
r15:                    0      general purpose registers
r14:                    0
r13:          7ffd37def140
r12:          55a7e805e5f0
rbp:          7ffd37def060
rbx:                    0
r11:                    7
r10:                    2
r9:           7ff4ca5a5d80   6.
r8:           7ff4ca5a5d80   5.
rax:                  abcd
rcx:          55a7e805e790   4.
rdx:          7ffd37def158   3.
rsi:          7ffd37def148   2.
rdi:                    1    1. function/syscall argument
orig_rax:ffffffffffffffff
rip:          55a7e805e728
cs:                    33
eflags:               286
rsp:          7ffd37def040   Stack Pointer (current location in stack)
ss:                    2b
fs_base:      7ff4ca7b54c0
gs_base:                0
ds:                     0
es:                     0
fs:                     0
gs:                     0
(q)uit, next (s)tep, (p)eek data, (P)oke data, print (r)egisters
p
peekdata hexaddr: 55a7e805e728
byte_offset:55a7e805e728
PEEKDATA: 8be4740000abcd3d
P
poke hexaddr: 55a7e805e728
hexword:8be4740000aaaa3d
  poke_data called pid:1934, offset:94179640600360, word:8be4740000aaaa3d
  word:8be4740000aaaa3d
q
jsbaik@jsbaik:~/OS2019/week8/process_hacking$
```

rax의 값이 abcd가 되는 순간!

rip에 담겨있는 다음 명령어의 주소에 접근해 해킹함

생각해보기 답

<table>
<tr><td rowspan="2">데이터 세그먼트들</td><td>스택(stack)
↓
↑
힙(heap)</td><td>지역변수 (Local variable),
매개변수 (Parameter)</td><td>0xFFFFFFFF</td></tr>
<tr><td></td><td></td><td>동적 할당 영역</td></tr>
<tr><td></td><td>데이터</td><td>전역변수(Global variable), 정적변수(static variable),
문자열 리터럴(String Litreal)</td><td rowspan="5">정적 할당 영역</td></tr>
<tr><td rowspan="4">코드 세그먼트들</td><td>...</td><td rowspan="4">명령어 (Instruction)</td></tr>
<tr><td>printf</td></tr>
<tr><td>IsPrimeNumber</td></tr>
<tr><td>scanf</td></tr>
<tr><td>main</td><td>0x00000000</td></tr>
</table>

수고하셨습니다.

- 다음 시간:

  다중 스레드 정렬 응용