

운영체제론 실습 8주차

CPS LAB

ptrace (2) : 프로세스 해킹

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_ATTACH

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

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

- PTRACE_DETACH

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

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

ptrace 사용법 (2)

- PTRACE_PEEKDATA

```
ptrace(PTRACE_PEEKDATA, pid, addr, 0);
```

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

- PTRACE_POKEDATA

```
ptrace(PTRACE_POKEDATA, pid, addr, data);
```

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

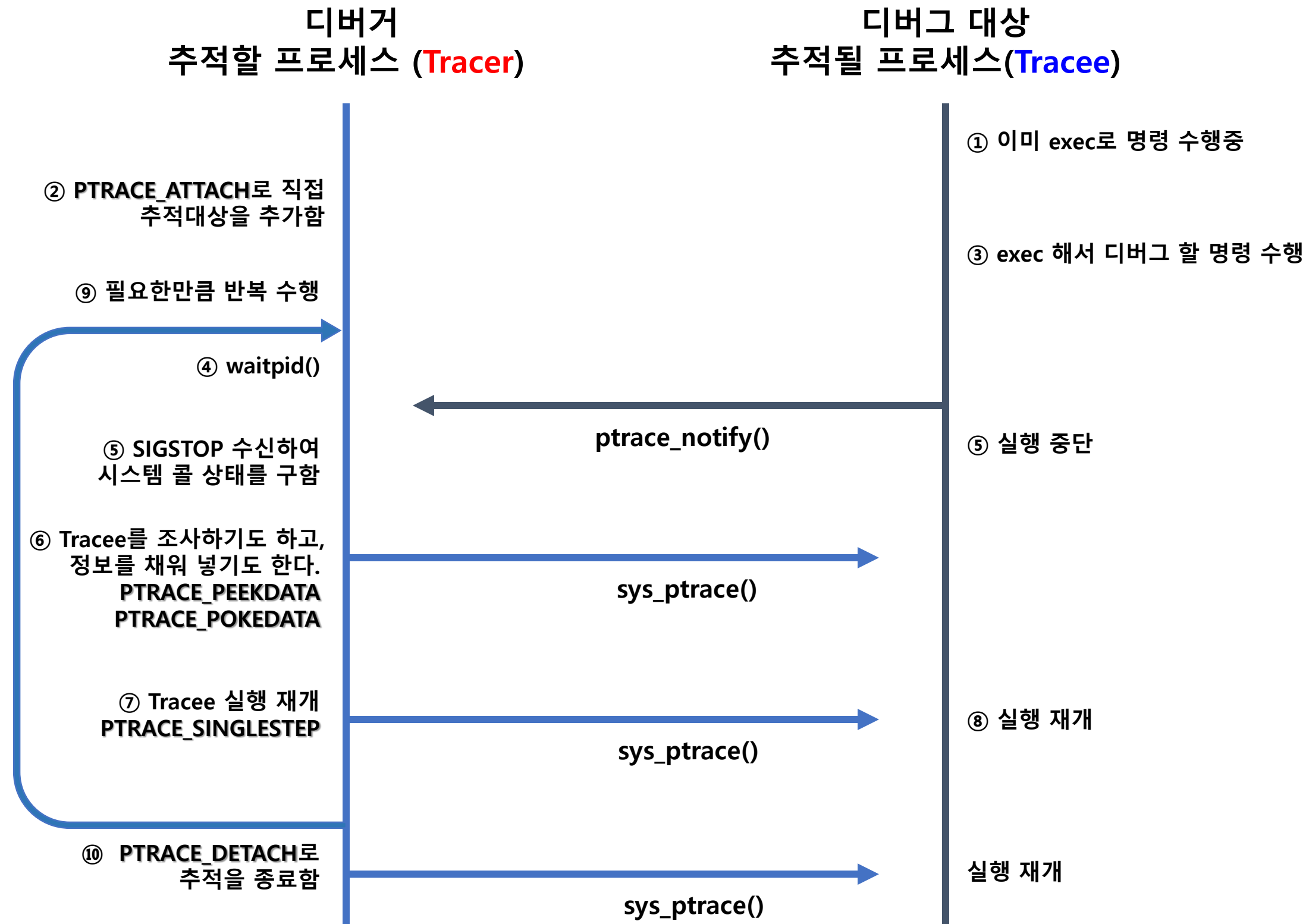
ptrace 사용법 (3)

- PTRACE_SINGLESTEP

```
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

tracer-tracee flow



Process Hacking 실습 (1)

week8/process_hacking/tracee.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;
}
```

Process Hacking 실습 (2)

week8/process_hacking/tracee.c & tracee.s

- tracee.c

```
8  /* First step: Compare my_flag with condition(0xabcd) */
9  while (my_flag == 0xabcd) {
10     i++;
11 }
12
```

각 c 코드를 Assembly 코드로 변환

(HEX) 0xabcd == (DEC) 43981

- 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
```

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 저장

rip 레지스터는 다음 명령어(51번째 줄)를 가리키며,
해당 명령어 중 abcd(DEC = 43981)값을
다른 값으로 변경하는 프로세스 해킹 실습 수행

Process Hacking 실습 (3)

week8/process_hacking/tracer.c

```
pid_t tracee_pid;
struct user_regs_struct regs;
int status;
bool keep_looping = true;

int main(int argc, char **argv) {

    if (argc < 2) {
        printf("[USAGE]: ./tracer <pid-of-target-process>\n");
        return -1;
    }
    tracee_pid = atoi(argv[1]);
    ptrace(PTRACE_ATTACH, tracee_pid, NULL, NULL);
    waitpid(tracee_pid, &status, 0);

    while (keep_looping == true) {
        ptrace(PTRACE_SINGLESTEP, tracee_pid, NULL, NULL);
        waitpid(tracee_pid, &status, 0);
        ptrace(PTRACE_GETREGS, tracee_pid, NULL, &regs);
        printf("[ CURRENT REGISTER STATE ]\n");
        print_user_regs_struct(regs);
        keep_looping = peekpoke_interactively(tracee_pid, regs);
    }
    ptrace(PTRACE_DETACH, tracee_pid, NULL, NULL);
    return 0;
}
```

Process Hacking 실습 (4)

week8/process_hacking/util.c

```
void print_peek_data(pid_t tracee_pid, long long int byte_offset) {
    printf("byte_offset:%llx\n", byte_offset);
    long long int peek_output;
    peek_output = ptrace(PTRACE_PEEKDATA, tracee_pid, byte_offset, NULL);
    if (peek_output == -1 & errno != 0) {
        printf("errno: %s\n", strerror(errno));
    }
    printf ("PEEKDATA: %llx \n", peek_output);
}

void print_peek_data_interactively(pid_t tracee_pid) {
    long long int input;
    printf("peekdata hexaddr: ");
    scanf("%llx", &input);
    print_peek_data(tracee_pid, input);
}

void poke_data(pid_t tracee_pid, long long int byte_offset, long long int word) {
    long long int ptrace_output;
    printf (" poke_data called pid:%d, offset:%lld, word:%llx\n", tracee_pid, byte_offset, word);
    printf (" word:%llx\n", word);
    ptrace_output = ptrace(PTRACE_POKEDATA, tracee_pid, byte_offset, word);
    if (ptrace_output == -1 & errno != 0) {
        printf("errno: %s\n", strerror(errno));
    }
}

void poke_data_interactively(pid_t tracee_pid) {
    long long int byte_offset;
    long long int word;
    printf ("poke hexaddr: ");
    scanf ("%llx", &byte_offset);
    printf ("hexword:");
    scanf("%llx", &word);
    poke_data(tracee_pid, byte_offset, word);
}
```

프로젝트 실행

1. 코드 컴파일 수행

```
$ make
```

2. tracee 코드 실행

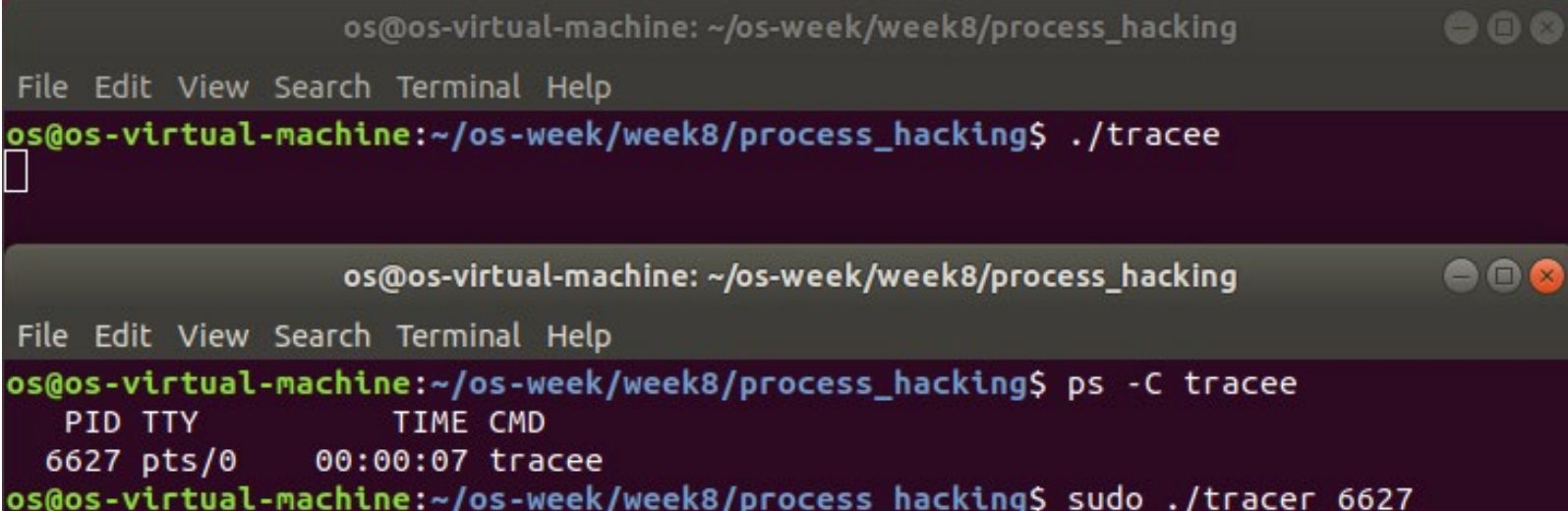
```
$ ./tracee
```

3. tracee 프로세스의 pid 확인

```
$ ps -C tracee
```

4. tracer 코드 실행 (pid를 가진 프로세스를 대상으로 추적)

```
$ sudo ./tracer {tracee's pid}
```



The image shows two terminal windows. The top window shows the command `./tracee` being executed. The bottom window shows the command `ps -C tracee` being executed, which displays the following output:

PID	TTY	TIME	CMD
6627	pts/0	00:00:07	tracee

Below the table, the command `sudo ./tracer 6627` is shown being executed.

터미널 2개로 나누어서
하나는 tracee Code,
나머지 하나는 tracer Code 실행

프로젝트 실행 결과

```
os@os-virtual-machine:~/os-week/week8/process_hacking$ sudo ./tracer 6627
[ CURRENT REGISTER STATE ]
r15: 0 general purpose registers
r14: 0
r13: 7ffc36e9e1c0
r12: 55fd602005f0
rbp: 7ffc36e9e0e0
rbx: 0
r11: 0
r10: 0
r9: 7f7bcf1cbd80 6.
r8: 7f7bcf1cbd80 5.
rax: abcd
rcx: 55fd60200790 4.
rdx: 7ffc36e9e1d8 3.
rsi: 7ffc36e9e1c8 2.
rdi: 1 1. function/syscall argument
orig_rax: ffffffff
rip: 55fd60200728 instruction pointer
cs: 33
eflags: 286
rsp: 7ffc36e9e0c0 Stack Pointer (current location in stack)
ss: 2b
fs_base: 7f7bcf3e54c0
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: 55fd60200728
byte_offset: 55fd60200728
PEEKDATA: 8be4740000abcd3d
P
poke hexaddr: 55fd60200728
hexword: 8be4740000aaaa3d
poke_data called pid: 6627, offset: 94546727798568, word: 8be4740000aaaa3d
q
```

```
os@os-virtual-machine:~/os-week/week8/process_hacking$ ./tracee
Tracee exits with flag: abcd
```

rax 값이 abcd가 될 때 까지
(s)tep 수행

rip에 담겨있는 다음 명령어의
주소에 접근하여 해킹함

메모리 주소의 데이터 값
abcd만 aaaa로 변경

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

Question 정답

정답 : 코드 세그먼트 (Code Segment)

데이터 세그먼트 들	스택(stack)	지역변수 (Local variable), 매개변수 (Parameter)	0xFFFFFFFF
	힙(heap)		동적 할당 영역
	데이터	전역변수 (Global variable), 정적변수 (static variable), 문자열 리터럴 (String Literal)	
코드 세그먼트 들	...	명령어 (Instruction)	정적 할당 영역
	printf		
	IsPrimeNumber		
	scanf		
	main		0x00000000

참고 : <https://ultradream.tistory.com/entry/%EC%84%B8%EA%B7%B8%EB%A8%BC%ED%8A%B8Segment-%EB%9E%80>

감사합니다.

CPS LAB