**운영체제 과제 #02**

**제출자 : 2019101074 안용상**

**목차 :**

**1. cat /proc/cpuinfo 결과**

**2. lsmod결과**

## 3. GPT코드 메모리 취약점 및 트리거 방법 내용설명

**4. dmesg 전체 코드(이미지 필요부분 전체 캡쳐)**

**5. 크래시 로그에 대해서 학습 후 내용을 설명**

**1. cat/proc/cpuinfo 결과**

dydtkddhkdwk@dydtkddhkdwk-virtual-machine:~$ cat /proc/cpuinfo

processor : 0

vendor\_id : AuthenticAMD

cpu family : 23

model : 113

model name : AMD Ryzen 5 3500X 6-Core Processor

stepping : 0

cpu MHz : 3593.251

cache size : 512 KB

physical id : 0

siblings : 1

core id : 0

cpu cores : 1

apicid : 0

initial apicid : 0

fpu : yes

fpu\_exception : yes

cpuid level : 16

wp : yes

flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx mmxext fxsr\_opt pdpe1gb rdtscp lm constant\_tsc rep\_good nopl tsc\_reliable nonstop\_tsc cpuid extd\_apicid tsc\_known\_freq pni pclmulqdq ssse3 fma cx16 sse4\_1 sse4\_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor lahf\_lm extapic cr8\_legacy abm sse4a misalignsse 3dnowprefetch osvw topoext ssbd ibpb vmmcall fsgsbase bmi1 avx2 smep bmi2 rdseed adx smap clflushopt clwb sha\_ni xsaveopt xsavec xgetbv1 xsaves clzero wbnoinvd arat umip rdpid overflow\_recov succor

bugs : fxsave\_leak sysret\_ss\_attrs null\_seg spectre\_v1 spectre\_v2 spec\_store\_bypass retbleed smt\_rsb

bogomips : 7186.50

TLB size : 3072 4K pages

clflush size : 64

cache\_alignment : 64

address sizes : 45 bits physical, 48 bits virtual

power management:

processor : 1

vendor\_id : AuthenticAMD

cpu family : 23

model : 113

model name : AMD Ryzen 5 3500X 6-Core Processor

stepping : 0

cpu MHz : 3593.251

cache size : 512 KB

physical id : 2

siblings : 1

core id : 0

cpu cores : 1

apicid : 2

initial apicid : 2

fpu : yes

fpu\_exception : yes

cpuid level : 16

wp : yes

flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx mmxext fxsr\_opt pdpe1gb rdtscp lm constant\_tsc rep\_good nopl tsc\_reliable nonstop\_tsc cpuid extd\_apicid tsc\_known\_freq pni pclmulqdq ssse3 fma cx16 sse4\_1 sse4\_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor lahf\_lm extapic cr8\_legacy abm sse4a misalignsse 3dnowprefetch osvw topoext ssbd ibpb vmmcall fsgsbase bmi1 avx2 smep bmi2 rdseed adx smap clflushopt clwb sha\_ni xsaveopt xsavec xgetbv1 xsaves clzero wbnoinvd arat umip rdpid overflow\_recov succor

bugs : fxsave\_leak sysret\_ss\_attrs null\_seg spectre\_v1 spectre\_v2 spec\_store\_bypass retbleed smt\_rsb

bogomips : 7186.50

TLB size : 3072 4K pages

clflush size : 64

cache\_alignment : 64

address sizes : 45 bits physical, 48 bits virtual

power management:

**2. lsmod결과**

1. dydtkddhkdwk@dydtkddhkdwk-virtual-machine:~$ lsmod
2. Module Size Used by
3. my\_module3 16384 0
4. isofs 53248 1
5. vsock\_loopback 16384 0
6. vmw\_vsock\_virtio\_transport\_common 45056 1 vsock\_loopback
7. vmw\_vsock\_vmci\_transport 32768 2
8. vsock 49152 7 vmw\_vsock\_virtio\_transport\_common,vsock\_loopback,vmw\_vsock\_vmci\_transport
9. binfmt\_misc 24576 1
10. intel\_rapl\_msr 20480 0
11. snd\_ens1371 28672 2
12. snd\_ac97\_codec 176128 1 snd\_ens1371
13. intel\_rapl\_common 40960 1 intel\_rapl\_msr
14. gameport 24576 1 snd\_ens1371
15. crct10dif\_pclmul 16384 1
16. ac97\_bus 16384 1 snd\_ac97\_codec
17. snd\_pcm 159744 2 snd\_ac97\_codec,snd\_ens1371
18. nls\_iso8859\_1 16384 1
19. ghash\_clmulni\_intel 16384 0
20. snd\_seq\_midi 20480 0
21. snd\_seq\_midi\_event 16384 1 snd\_seq\_midi
22. snd\_rawmidi 45056 2 snd\_seq\_midi,snd\_ens1371
23. vmw\_balloon 28672 0
24. aesni\_intel 376832 0
25. crypto\_simd 16384 1 aesni\_intel
26. cryptd 24576 2 crypto\_simd,ghash\_clmulni\_intel
27. snd\_seq 77824 2 snd\_seq\_midi,snd\_seq\_midi\_event
28. joydev 32768 0
29. input\_leds 16384 0
30. serio\_raw 20480 0
31. snd\_seq\_device 16384 3 snd\_seq,snd\_seq\_midi,snd\_rawmidi
32. snd\_timer 40960 2 snd\_seq,snd\_pcm
33. snd 114688 11 snd\_seq,snd\_seq\_device,snd\_timer,snd\_ac97\_codec,snd\_pcm,snd\_rawmidi,snd\_ens1371
34. soundcore 16384 1 snd
35. vmw\_vmci 90112 2 vmw\_balloon,vmw\_vsock\_vmci\_transport
36. mac\_hid 16384 0
37. sch\_fq\_codel 24576 2
38. vmwgfx 372736 6
39. drm\_ttm\_helper 16384 1 vmwgfx
40. ttm 98304 2 vmwgfx,drm\_ttm\_helper
41. drm\_kms\_helper 200704 1 vmwgfx
42. fb\_sys\_fops 16384 1 drm\_kms\_helper
43. syscopyarea 16384 1 drm\_kms\_helper
44. sysfillrect 20480 1 drm\_kms\_helper
45. sysimgblt 20480 1 drm\_kms\_helper
46. msr 16384 0
47. parport\_pc 53248 0
48. ppdev 24576 0
49. lp 28672 0
50. parport 73728 3 parport\_pc,lp,ppdev
51. drm 581632 10 vmwgfx,drm\_kms\_helper,drm\_ttm\_helper,ttm
52. pstore\_blk 16384 0
53. pstore\_zone 32768 1 pstore\_blk
54. ramoops 32768 0
55. reed\_solomon 28672 1 ramoops
56. efi\_pstore 16384 0
57. ip\_tables 32768 0
58. x\_tables 57344 1 ip\_tables
59. autofs4 45056 2
60. hid\_generic 16384 0
61. usbhid 65536 0
62. hid 159744 2 usbhid,hid\_generic
63. crc32\_pclmul 16384 0
64. psmouse 180224 0
65. ahci 49152 1
66. libahci 49152 1 ahci
67. e1000 159744 0
68. mptspi 24576 2
69. mptscsih 49152 1 mptspi
70. mptbase 106496 2 mptspi,mptscsih
71. scsi\_transport\_spi 32768 1 mptspi
72. i2c\_piix4 32768 0
73. pata\_acpi 16384 0
74. floppy 118784 0

## 3. GPT코드 메모리 취약점 및 트리거 방법 내용설명

우선 지피티가 제공해준 코드는 다음과 같다.

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/proc\_fs.h>

#include <linux/uaccess.h>

#define PROC\_ENTRY\_FILENAME "my\_proc\_entry"

static struct proc\_dir\_entry \*proc\_entry;

static ssize\_t my\_write(struct file \*file, const char \_\_user \*buffer, size\_t count, loff\_t \*f\_pos){

char input[128];

if (copy\_from\_user(input, buffer, count)){

}

input[min(count, (sizeof(input)-1))] = '\\0';

printk(KERN\_INFO "my\_proc\_entry was written: %s\\n", input);

return count;

}

static const struct proc\_ops myops = {

.proc\_write = my\_write;

}

int init\_module(){

printk(KERN\_INFO "Loading module...\\n");

proc\_entry = proc\_create(PROC\_ENTRY\_FILENAME, 0666, NULL, &myops);

if (proc\_entry == NULL){

printk(KERN\_INFO "Couldn't create proc entry \\n");

return -ENOMEM;

}

return 0;

}

void cleanup\_module(){

printk(KERN\_INFO "Unloading module...\\n");

proc\_remove(proc\_entry)

}

GPT코드의 메모리 취약점을 찾아내기 위해, 먼저 GPT가 제시한 코드를 간략히 살펴보겠다

먼저 init\_module()함수는 쉘에서 insmod라는 명령어를 실행시킬 때, 실행이 되며, 모듈을 초기화하고 로딩시키며, 사용자가 그 안에서 정의한 코드를 실행시킨다. 그 안을 살펴보면 먼저 printk라는 함수가 있는데, 이는 dmesg로 띄운 로그화면에 메세지를 출력하게 해주는 함수이다.

그 다음 proc\_entry라는 변수를 초기화해주는데, proc\_entry란 proc\_dir\_entry구조체로 위에서 정의했다. 이 proc\_dir\_entry구조체는 간략하게 프로세스 파일 시스템관련 작업에 대한 핸들을 저장하는 변수라고 생각할 수 있다.

이에 할당연산자 우측의 함수 proc\_create()함수를 보자. 이 함수는 두가지 기능을 수행한다. 파라미터로 건네받은 인자들을 이용해서, 프로세스 파일을 /proc경로에 생성하고, 두번째로 그 파일에 대한 핸들을 구조체로써 반환하며 proc\_entry 변수에 저장하게 한다. 이때 /proc경로에 생성되는 파일의 이름은 PROC\_ENTRY\_FILENAME이라고 위에서 정의한 매크로에 정의된 이름이다.

그리고 파라미터의 마지막 &myops는 위에서 정의한 proc\_ops구조체의 myops구조체 인스턴스이다. 이 proc\_ops구조체는 프로세스 파일에 대한 작업 콜백 함수를 정의해놓은 구조체이다. 프로세스 파일로부터 데이터를 읽거나, 프로세스 파일에 데이터를 쓰거나 할때, 즉 프로세스 파일과 관련해서 어떤 작업을 할 때, 작업이 진행되는 시점에서 호출되어 원하는 동작을 수행하도록 하는 코드가 모인곳이다. 그러한 구조체에는 read,write 등등 관련 함수들이 멤버함수로써 정의되어있고, 그 구조체로 인스턴스를 정의하면 그 멤버함수들을 그대로 내려받는다.

그런데 이때, GPT의 코드에서는 proc\_ops구조체의 .proc\_write멤버함수를 사용자 정의함수인 my\_write()라는 함수로 대체하고 있다.

이제 my\_write()라는 함수를 사용자 정의해놓은 부분을 보자. my\_write()함수는 어떻게 만들어졌는지 살펴보면, 유저영역에서 작성한 데이터가 담긴 버퍼를 건네받아, 커널영역의 버퍼인 input이라는 이름의 버퍼에 옮겨 담는 함수이며, 그 작업이 끝나면, input버퍼안에 들어간 문자열데이터를 dmesg화면의 로그에 출력해주는 함수이다. 그리고 마지막으로 clean\_up이라는 함수는 로딩된 모듈을 언로드해주는 함수이다.

GPT가 준 이코드에서 메모리 취약점이 생길 수 있는 코드부분을 생각해봤다. 굉장히 다양할 수 있지만, 나는 my\_write()함수 내에서 그 취약점을 찾아냈다.

static ssize\_t my\_write(struct file \*file, const char \_\_user \*buffer, size\_t count, loff\_t \*f\_pos){

char input[128];

if (copy\_from\_user(input, buffer, count)){

return -EFAULT;

}

input[min(count, (sizeof(input)-1))] = '\\0';

printk(KERN\_INFO "my\_proc\_entry was written: %s\\n", input);

return count;

}

위와 같이 my\_write함수가 정의되어 있는데

input을 크기 128의 문자배열로 선언하는 코드 대신에

char \*input = NULL;의 코드로 바꿔넣는다면 어떨까?

그리고 input[0] = ‘A’;라는 코드를 넣어서 input이 가리키는 곳의 값에 접근하는 코드를 넣는다면 어떻게 될까?

런타임에 이러한 코드가 그대로 실행이 된다면, 널포인터의 값에 접근하려 하는 것이기 때문에, 유효한 메모리 주소에 대한 값접근이 아니라서 예외가 발생하게 된다. 컴파일 시간에 이를 잡아내지 못하기 때문에, 컴파일 오류 없이 진행이되고, 런타임에서 커널 크래시가 일어난다.

static ssize\_t my\_write(struct file \*file, const char \_\_user \*buffer, size\_t count, loff\_t \*f\_pos){

char \*input = NULL;

input[0] = 'A';

if (copy\_from\_user(input, buffer, count)){

return -EFAULT;

}

input[min(count, (sizeof(input)-1))] = '\\0';

printk(KERN\_INFO "my\_proc\_entry was written: %s\\n", input);

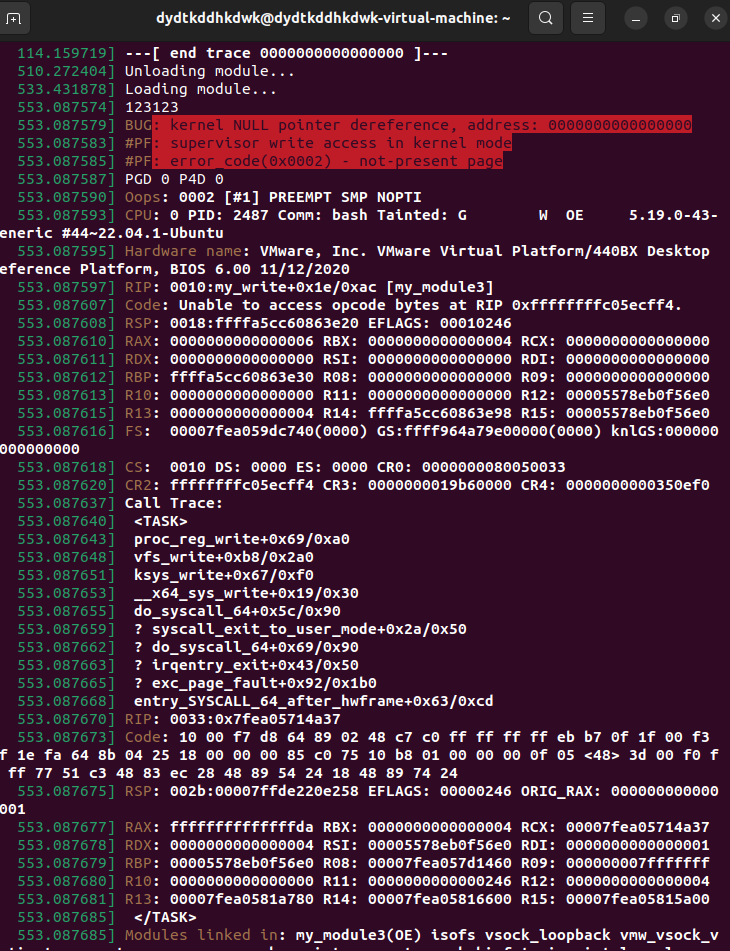
return count;

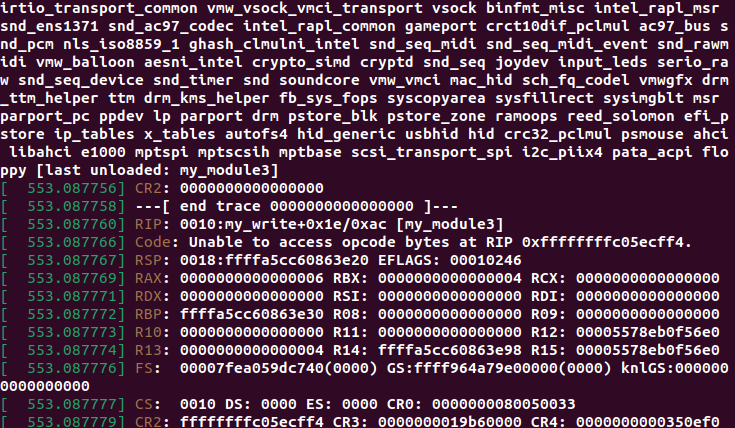
}

실제로 이렇게 해서 컴파일을 시킨뒤, 모듈을 적재하고 echo로 아무 값이나 전달해서 my\_write콜백함수를 호출했을 때, 실제로 그 부분이 크래시의 트리거가 되는 것을 직접 확인했다.

따라서 GPT가 제공해준 코드는, NULL POINTER에 대해 메모리 취약점을 가지고 있었다고 할 수 있다

**4. dmesg 전체 코드**





**5. 크래시 로그에 대해서 학습 후 내용을 설명**

커널 크래시에 대한 로그정보는 매우 길다. 그래서 이번 경우에 해당하는 커널크래시 로그정보를 크게 5가지 구획으로 위에서부터 나온순서대로 나눠 정리해봤다

| 첫번째 | 크래시의 종류에 대한 핵심 요약 |
| --- | --- |
| 두번째 | 현재 하드웨어 스펙에 대한 설명 |
| 세번째 | 크래시가 유발된 순간의 명령어 위치에 대한 상세정보(오프셋까지) 요약 |
| 네번째 | 크래시가 발생한 그 순간에 수많은 다양한 레지스터들이 갖고있던 값들을 레지스터이름과 짝지어서 출력하는 부분 |
| 다섯 번째 | Call Trace라고 불리는 호출추적에 대한 로깅 정보를 출력하는 부분 |

이 다섯가지 구획에서 물론 모든 정보가 중요하지만 첫번째 세번째 다섯 번째에 대한 정보가 보다 더 중요하다고 생각한다. 그래서 그 부분에 대한 설명을 요약해보겠다.

**첫번째 구획. 크래시의 종류에 대한 핵심 요약은 다음과 같다.**

[ 553.087579] BUG: kernel NULL pointer dereference, address: 0000000000000000 부터 [ 553.087590] Oops: 0002 [#1] PREEMPT SMP NOPTI 까지 이어지는 로그는 다음과 같다. 가장 위의 로그 문장은 커널에서 NULL 포인터 역 참조 오류가 발생했고, 해당 버그가 0x0000000000000000 주소에서 발생했다는 것을 알려주며, 그 뒤로 이어지는 로그들은 NULL 포인터 역 참조 오류의 과정을 보다 상세하게 기술해준다. 커널 모드에서 슈퍼 바이저 쓰기 접근을 시도했지만, 쓰고자 하는 값의 주소가 널 포인터이기 때문에, 그에 대응하는 페이지에 대한 권한을 현재 프로세스는 갖고 있지 않으며, 그에 따라 PageFault오류가 나게 되었다는 의미를 함축하고 있다. 또한 마지막 줄은 커널 패닉이 오류코드 0002로써 발생했다는 것을 알려준다.

**세번째 구획. 크래시가 유발된 순간의 명령어 위치에 대한 상세정보(오프셋까지) 요약한 부분**

[ 553.087597] RIP: 0010:my\_write+0x1e/0xac [my\_module3] [ 553.087607] Code: Unable to access opcode bytes at RIP 0xffffffffc05ecff4

위 두 로그는 현재 실행중인 명령어의 위치인 RIP의 값을 알려주고 있다. 즉 커널 크래시가 난 순간에 위치했던 명령어 위치. 다시 말해 크래시가 일어난 명령어의 위치를 로깅하고 있다. 그 위치를 my\_module3라는 모듈의 my\_write함수에서 그 함수 전체크기 0xac중에 0x1e만큼 떨어진 곳으로 설명하고 있다. 또한 RIP 주소 0xffffffffc05ecff4에서 명령어 바이트에 접근할 수 없음을 알려주고 있다.

**다섯 번째 구획. Call Trace라고 불리는 호출추적에 대한 로깅 정보를 출력하는 부분**

Call Trace아래부분은 다음과 같다. <Task>태그부터 </Task>태그까지 아래로 갈 수록 호출순서의 역순으로 함수들의 호출을 추적한 로그를 출력해준다. 이 정보로 현재 스택 상태를 알 수 있고, 현재 호출상태를 파악할 수 있다.

추가로 Modules linked in으로 시작하는 긴 문장이 있는 부분이 있는데, 시스템에서 현재 로드 된 모듈의 목록을 보여준다. 그 부분을 살펴보면 가장 먼저, 크래시를 유발한 코드를 담고 있는 my\_module3 모듈이 출력이 되는 것을 볼 수 있다.