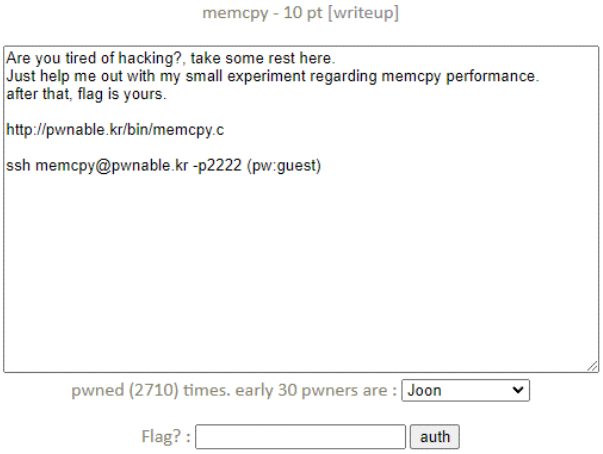
Pwnable.kr
- memcpy -

Download: http://pwnable.kr/bin/memcpy.c
ssh memcpy@pwnable.kr -p 2222
pw : guest

0. 문제 살펴보기



memcpy 동작에 대하여 얘기하고 있다.

1. SSH 접속 및 살펴보기

SSH를 이용해 상단에 표기해놓은 주소와 포트 번호로 접속한다.

디렉토리의 파일들을 살펴보자 C 코드 파일과 readme 텍스트 파일이 존재하는 것을 알 수 있다.
readme 파일을 살펴보자 nc 0 9022로 접속하여 memcpy 파일을 실행하여야 한다고 한다.
memcpy.c 파일은 ssh 접속이 아닌 다운로드를 통해 볼 수도 있으므로 이를 통해 살펴보자.

상단에 표기해놓은 다운로드 주소를 통해 파일을 다운로드 받았다.
이후 cat을 통해 memcpy.c 파일을 읽어보았더니 다음과 같은 코드가 등장하였다.

```
// compiled with : gcc -o memcpy memcpy.c -m32 -lm
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/mman.h>
#include <math.h>

unsigned long long rdtsc(){
	asm("rdtsc");
}

char* slow_memcpy(char* dest, const char* src, size_t len){
	int i;
	for (i=0; i<len; i++) {
		dest[i] = src[i];
	}
	return dest;
}

char* fast_memcpy(char* dest, const char* src, size_t len){
	size_t i;
	// 64-byte block fast copy
	if(len >= 64){
		i = len / 64;
		len &= (64-1);
		while(i-- > 0){
			__asm__ __volatile__ (
			"movdqa (%0), %%xmm0₩n"
			"movdqa 16(%0), %%xmm1₩n"
			"movdqa 32(%0), %%xmm2₩n"
			"movdqa 48(%0), %%xmm3₩n"
			"movntps %%xmm0, (%1)₩n"
			"movntps %%xmm1, 16(%1)₩n"
			"movntps %%xmm2, 32(%1)₩n"
			"movntps %%xmm3, 48(%1)₩n"
			::"r"(src),"r"(dest):"memory");
			dest += 64;
			src += 64;
		}
	}

	// byte-to-byte slow copy
	if(len) slow_memcpy(dest, src, len);
	return dest;
}
int main(void){

	setvbuf(stdout, 0, _IONBF, 0);
	setvbuf(stdin, 0, _IOLBF, 0);

	printf("Hey, I have a boring assignment for CS class.. :(₩n");
	printf("The assignment is simple.₩n");

	printf("-----------------------------------------------------₩n");
	printf("- What is the best implementation of memcpy?          -₩n");
	printf("- 1. implement your own slow/fast version of memcpy -₩n");
	printf("- 2. compare them with various size of data           -₩n");
	printf("- 3. conclude your experiment and submit report       -₩n");
	printf("-----------------------------------------------------₩n");

	printf("This time, just help me out with my experiment and get flag₩n");
	printf("No fancy hacking, I promise :D₩n");

	unsigned long long t1, t2;
	int e;
	char* src;
	char* dest;
	unsigned int low, high;
	unsigned int size;
	// allocate memory
	char* cache1 = mmap(0, 0x4000, 7, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
```

```
        char* cache2 = mmap(0, 0x4000, 7, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
        src = mmap(0, 0x2000, 7, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);

        size_t sizes[10];
        int i=0;

        // setup experiment parameters
        for(e=4; e<14; e++){      // 2^13 = 8K
                low = pow(2,e-1);
                high = pow(2,e);
                printf("specify the memcpy amount between %d ~ %d : ", low, high);
                scanf("%d", &size);
                if( size < low || size > high ){
                        printf("don't mess with the experiment.₩n");
                        exit(0);
                }
                sizes[i++] = size;
        }

        sleep(1);
        printf("ok, lets run the experiment with your configuration₩n");
        sleep(1);

        // run experiment
        for(i=0; i<10; i++){
                size = sizes[i];
                printf("experiment %d : memcpy with buffer size %d₩n", i+1, size);
                dest = malloc( size );

                memcpy(cache1, cache2, 0x4000);             // to eliminate cache effect
                t1 = rdtsc();
                slow_memcpy(dest, src, size);               // byte-to-byte memcpy
                t2 = rdtsc();
                printf("ellapsed CPU cycles for slow_memcpy : %llu₩n", t2-t1);

                memcpy(cache1, cache2, 0x4000);             // to eliminate cache effect
                t1 = rdtsc();
                fast_memcpy(dest, src, size);               // block-to-block memcpy
                t2 = rdtsc();
                printf("ellapsed CPU cycles for fast_memcpy : %llu₩n", t2-t1);
                printf("₩n");
        }

        printf("thanks for helping my experiment!₩n");
        printf("flag : ----- erased in this source code -----₩n");
        return 0;
}
```

처음에 복사할 크기를 입력 받고, 메모리 복사를 진행할 때, 함수와 걸린 시간을 나타내는 것 같다.
slow_memcpy() 함수는 바이트 단위, fast_memcpy() 함수는 블록 단위로 복사를 진행하는 것 같다.

※ rdtsc

rdtsc 프로세서 타임스탬프를 반환하는 명령을 생성한다.
프로세서 타임스탬프는 마지막 재설정 이후의 클럭 주기 수를 기록한다.
즉, 코드 실행 시간을 측정하는 것이다.

※ movdqa

mov(옮기는 것) + dq(double quad word, 16 바이트를 뜻함) + a(align, 정렬된 상태)
메모리가 정렬된 상태로 레지스터로 복사할 때, 16 바이트 메모리가 모두 붙어있어 캐시에서 레지스터로 옮길 때, 부하 없이 로드할 수 있다고 한다.

2. 일단 실행해보기

```
memcpy@pwnable:~$ nc 0 9022
Hey, I have a boring assignment for CS class.. :(
The assignment is simple.
-----------------------------------------------------
- What is the best implementation of memcpy?        -
- 1. implement your own slow/fast version of memcpy -
- 2. compare them with various size of data          -
- 3. conclude your experiment and submit report      -
-----------------------------------------------------
This time, just help me out with my experiment and get flag
No fancy hacking, I promise :D
specify the memcpy amount between 8 ~ 16 : 16
specify the memcpy amount between 16 ~ 32 : 32
specify the memcpy amount between 32 ~ 64 : 64
specify the memcpy amount between 64 ~ 128 : 128
specify the memcpy amount between 128 ~ 256 : 256
specify the memcpy amount between 256 ~ 512 : 512
specify the memcpy amount between 512 ~ 1024 : 1024
specify the memcpy amount between 1024 ~ 2048 : 2048
specify the memcpy amount between 2048 ~ 4096 : 4096
specify the memcpy amount between 4096 ~ 8192 : 8192
ok, lets run the experiment with your configuration
experiment 1 : memcpy with buffer size 16
ellapsed CPU cycles for slow_memcpy : 2148
ellapsed CPU cycles for fast_memcpy : 360

experiment 2 : memcpy with buffer size 32
ellapsed CPU cycles for slow_memcpy : 516
ellapsed CPU cycles for fast_memcpy : 586

experiment 3 : memcpy with buffer size 64
ellapsed CPU cycles for slow_memcpy : 938
ellapsed CPU cycles for fast_memcpy : 100

experiment 4 : memcpy with buffer size 128
ellapsed CPU cycles for slow_memcpy : 1826
memcpy@pwnable:~$
```

memcpy 계정에서 nc를 통해 접속하자, memcpy 실행 파일이 실행된 모습이다.
memcpy를 할 양을 범위에서 최댓값을 입력하자, 따로 정의한 memcpy() 함수들이 실행되고 그에 따른 실행 시간이 출력되었다. 하지만, 4번째 시행까지만 출력이 되고 이후는 출력이 되지 않은 채 프로그램이 종료된 모습이다.

```
specify the memcpy amount between 8 ~ 16 : 8
specify the memcpy amount between 16 ~ 32 : 16
specify the memcpy amount between 32 ~ 64 : 32
specify the memcpy amount between 64 ~ 128 : 64
specify the memcpy amount between 128 ~ 256 : 128
specify the memcpy amount between 256 ~ 512 : 256
specify the memcpy amount between 512 ~ 1024 : 512
specify the memcpy amount between 1024 ~ 2048 : 1024
specify the memcpy amount between 2048 ~ 4096 : 2048
specify the memcpy amount between 4096 ~ 8192 : 4096
ok, lets run the experiment with your configuration
experiment 1 : memcpy with buffer size 8
ellapsed CPU cycles for slow_memcpy : 2340
ellapsed CPU cycles for fast_memcpy : 262

experiment 2 : memcpy with buffer size 16
ellapsed CPU cycles for slow_memcpy : 226
ellapsed CPU cycles for fast_memcpy : 248

experiment 3 : memcpy with buffer size 32
ellapsed CPU cycles for slow_memcpy : 366
ellapsed CPU cycles for fast_memcpy : 434

experiment 4 : memcpy with buffer size 64
ellapsed CPU cycles for slow_memcpy : 552
ellapsed CPU cycles for fast_memcpy : 146

experiment 5 : memcpy with buffer size 128
ellapsed CPU cycles for slow_memcpy : 988
```

위는 범위에서 최솟값을 입력한 결과이다. 이번에는 5번째 시행에서 프로그램이 종료된 모습을 볼 수 있다.

3. 문제점 파악

4번째 시행 때 종료된 부분을 보면, slow_memcpy() 함수는 모두 실행되어 그 시간이 출력되었고,
fast_memcpy() 함수의 시간이 출력되지 않은 것을 보아 이 함수가 실행될 때, 종료된 것으로 파악된다.
fast_memcpy() 함수를 살펴보면 처음 len의 값이 64보다 클 경우, movdqa 부분이 실행되지만, 그렇지 않을 경우에는 다시 slow_memcpy() 함수가 실행된다.
또한, fast_memcpy() 함수에서 movdqa는 16 바이트씩 복사를 진행한다. 따라서, 처음 memcpy 크기를 정할 때, 16 바이트 단위로 입력해야 된다고 생각된다.

```
memcpy@pwnable:~$ cp memcpy.c /tmp/sun/
memcpy@pwnable:~$ cd /tmp/sun
```

```
memcpy(cache1, cache2, 0x4000);         // to eliminate cache effect
t1 = rdtsc();
fast_memcpy(dest, src, size);           // block-to-block memcpy
t2 = rdtsc();
printf("src: %p\n", src);
printf("dest: %p\n", dest);
printf("ellapsed CPU cycles for fast_memcpy : %llu\n", t2-t1);
printf("\n");
```

```
memcpy@pwnable:/tmp/sun$ vi memcpy.c
memcpy@pwnable:/tmp/sun$ gcc -o memcpy memcpy.c -m32 -lm
```

무엇이 문제인지를 파악하기 위해 /tmp 디렉토리에 있는 개인 디렉토리에 C 코드 파일을 복사하고, src와 dest의 주소를 비교하기 위해 fast_memcpy() 함수 이후에 위와 같은 printf() 코드들을 추가한 후 컴파일 해보았다.

```
experiment 1 : memcpy with buffer size 8
ellapsed CPU cycles for slow_memcpy : 2086
src: 0×f7765000
dest: 0×8356410
ellapsed CPU cycles for fast_memcpy : 212

experiment 2 : memcpy with buffer size 16
ellapsed CPU cycles for slow_memcpy : 316
src: 0×f7765000
dest: 0×8356420
ellapsed CPU cycles for fast_memcpy : 308

experiment 3 : memcpy with buffer size 32
ellapsed CPU cycles for slow_memcpy : 496
src: 0×f7765000
dest: 0×8356438
ellapsed CPU cycles for fast_memcpy : 580

experiment 4 : memcpy with buffer size 64
ellapsed CPU cycles for slow_memcpy : 910
src: 0×f7765000
dest: 0×8356460
ellapsed CPU cycles for fast_memcpy : 104

experiment 5 : memcpy with buffer size 128
ellapsed CPU cycles for slow_memcpy : 9846
Segmentation fault (core dumped)
```

이후, 같은 조건으로 프로그램을 실행해본 결과 위와 같았다.
우리는 16 바이트씩 복사를 진행해야 한다. 하지만, 현재 experiment 3에서의 dest 주소를 보면 끝이 0x8로 끝나는 것을 볼 수 있다. 여기에서 우리는 16 바이트씩 복사된 것이 아닌 것을 볼 수 있다.
그렇다면 8 바이트가 추가된 것이므로 이곳에 8 바이트를 추가적으로 사이즈를 주면 16 바이트씩 나눠질 것이다. 따라서, 우리는 fast_memcpy() 함수에서 len의 크기가 64이상이 되는 experiment 4부터 size의 값을 기존보다 8 높은 수로 줄 것이다.

그 이유는 uaf에서의 마지막 설명처럼 64비트 환경에서의 heap 영역은 16 바이트씩 할당하지만 8 바이트의 여유 공간을 더 가진다. 따라서, 위는 heap 영역에 할당될 때, 일어나는 문제로 8 바이트를 추가적으로 더 줘야 한다.

4. 공격

```
specify the memcpy amount between 8 ~ 16 : 8
specify the memcpy amount between 16 ~ 32 : 16
specify the memcpy amount between 32 ~ 64 : 32
specify the memcpy amount between 64 ~ 128 : 72
specify the memcpy amount between 128 ~ 256 : 136
specify the memcpy amount between 256 ~ 512 : 264
specify the memcpy amount between 512 ~ 1024 : 520
specify the memcpy amount between 1024 ~ 2048 : 1032
specify the memcpy amount between 2048 ~ 4096 : 2056
specify the memcpy amount between 4096 ~ 8192 : 4104
```

```
experiment 10 : memcpy with buffer size 4104
ellapsed CPU cycles for slow_memcpy : 31438
ellapsed CPU cycles for fast_memcpy : 1294

thanks for helping my experiment!
flag : [redacted]
```

위와 같이 memcpy amount를 입력하자, experiment 10까지 모두 실행되었고 이후, flag 값을 얻을 수 있었다.