漏洞解析之一：缓冲区溢出

1 概述

缓冲区溢出是最常见的代码漏洞之一，通常是指数据存储的位置超出了缓冲区的范围。一般在向缓冲区存储数据时发生。

2 分类

2.1 按照缓冲区的创建方式分类

- **栈缓冲区溢出**

  缓冲区在栈上创建的，比如数组变量对应的缓冲区。栈缓冲区的溢出称为栈缓冲区溢出。

- **堆缓冲区溢出**

  缓冲区在堆上创建的，例如 malloc 函数所对应的缓冲区。堆缓冲区的溢出称为堆缓冲区溢出。

2.2 按照溢出的方向分类

- **上溢**

  数据存储到缓冲区上边界之外，成为缓冲区上溢。

- **下溢**

  数据存储到缓冲区下边界之外，成为缓冲区下溢。

3 后果

- **系统崩溃**
- **过度内存消耗**
- **缓冲区中的敏感信息被覆盖**
  - 可能会导致系统的安全机制失效，进而被侵入。
  - 被覆盖部分可能对应任意的代码，因此会导致执行任意代码。

4 示例

- **示例 1**

```
typedef struct _charVoid
{
    char charFirst[16];
    void * voidSecond;
    void * voidThird;
} charVoid;


void
CWE121_Stack_Based_Buffer_Overflow__char_type_overrun_memcpy_01_bad()
{
    {
        charVoid structCharVoid;
        structCharVoid.voidSecond = (void *)SRC_STR;
        /* Print the initial block pointed to by structCharVoid.voidSecond */
```

```
                printLine((char *)structCharVoid.voidSecond);
                /* FLAW: Use the sizeof(structCharVoid) which will overwrite the
        pointer voidSecond */
                memcpy(structCharVoid.charFirst,                    SRC_STR,
        sizeof(structCharVoid));

        structCharVoid.charFirst[(sizeof(structCharVoid.charFirst)/sizeof(char))-1] = '\0'; /*
        null terminate the string */
                printLine((char *)structCharVoid.charFirst);
                printLine((char *)structCharVoid.voidSecond);
            }
        }
```

- **示例 2**

```
void CWE122_Heap_Based_Buffer_Overflow__char_type_overrun_memcpy_01_bad()
{
    {
        charVoid * structCharVoid = (charVoid *)malloc(sizeof(charVoid));
        if (structCharVoid == NULL) {exit(-1);}
        structCharVoid->voidSecond  = (void *)SRC_STR;
        /* Print the initial block pointed to by structCharVoid->voidSecond  */
        printLine((char *)structCharVoid->voidSecond);
        /* FLAW: Use the sizeof(*structCharVoid) which will overwrite the pointer y */
        memcpy(structCharVoid->charFirst,  SRC_STR, sizeof(*structCharVoid));
        structCharVoid->charFirst[(sizeof(structCharVoid->charFirst)/sizeof(char))-1] = '\0';
/* null terminate the string */
        printLine((char *)structCharVoid->charFirst);
        printLine((char *)structCharVoid->voidSecond);
        free(structCharVoid);
    }
}
```

- **示例 3**

```
void CWE124_Buffer_Underwrite__char_alloca_cpy_01_bad()
{
    char * data;
    char * dataBuffer = (char *)ALLOCA(100*sizeof(char));
    memset(dataBuffer, 'A', 100-1);
    dataBuffer[100-1] = '\0';
    /* FLAW: Set data pointer to before the allocated memory buffer */
    data = dataBuffer - 8;
    {
        char source[100];
        memset(source, 'C', 100-1); /* fill with 'C's */
        source[100-1] = '\0'; /* null terminate */
```

```
        /* POTENTIAL FLAW: Possibly copying data to memory before the destination buffer */
        strcpy(data, source);
        printLine(data);
    }
}
```

5 应对措施

- 写入缓冲区前要进行边界检查
- 使用可靠的 API
- 使用可靠的编译器
- 使用可靠的操作系统

6 相关漏洞

缓冲区溢出可以导致“返回地址覆盖”、“堆栈指针覆盖”、 “函数指针覆盖”、或“任意写入”等漏洞。