NETWORK SECURITY

한양대학교 소프트웨어융합대학 소프트웨어학부
이연준 교수

주요 사항

- **Buffer OverFlow (BOF)** 공격 기법에 대한 이해
- **Lab Preparation**
  - 실습 환경 구성 **(radare2** 설치**,** 기타 설정**)**
- **Lab Task**
  - **HTTP Header Live Tool**
  - 다양한 **XSS Attack** 실습
  - **XSS Attack** 대응법
- **Lab Question**
- **Evaluation**

Buffer OverFlow (BOF)

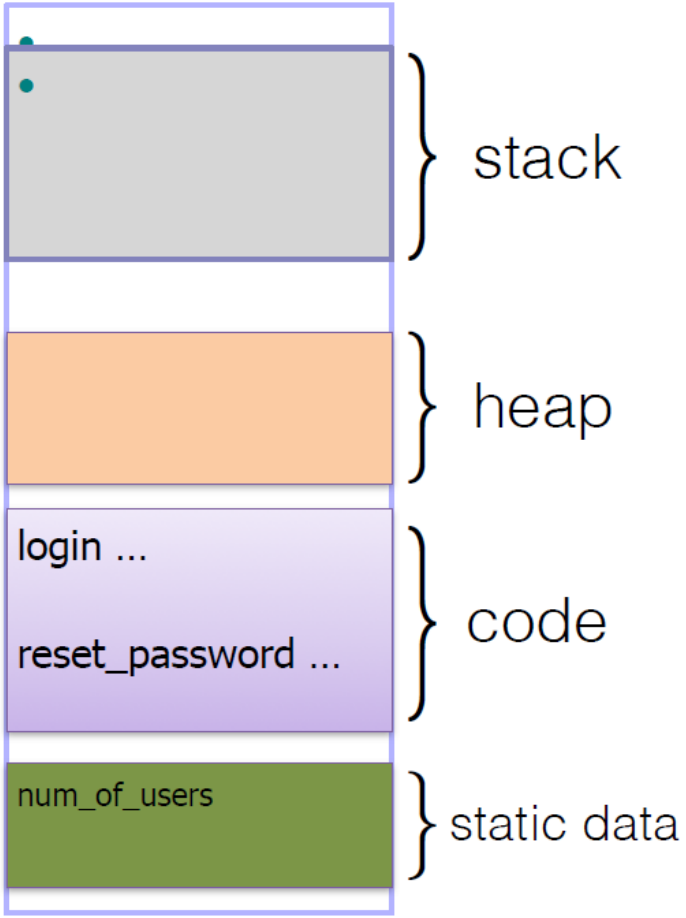
- 프로그램에서 할당한 메모리 **(Buffer)**의 양보다 많은 입력에 의해 **Buffer**의 경계를 넘어서는 것.
- 위와 같은 현상이 발생할 경우 프로그램은 비정상적으로 종료되며 예기치 못한 결과를 초래함
- **C Language**는 메모리의 경계값을 벗어나지 않는지 검사하는 내부 안전장치가 없음
- 메모리 영역에 따라 **Stack, Heap** 기반의 **Buffer Overflow**
  - 인접한 메모리 영역 **Overwrite, Off by One,** 해제한 메모리에 다시 **write (Double free)** 등
- 특수한 **Code Pattern**에 의한 **Buffer Overflow**
  - **Format String, Integer Over/Underflow**

Memory Layout of C language

```
bool num_of_users = 0;

bool login () {
  ……
  if (password_expires())
    reset_password();
  ……
}

void reset_password() {
  ……
  char usr[20], char pwd[100];
  gets(&usr); gets(&pwd);
  update_hash_file(usr,
      compute_hash(pwd, salt));
}
```



main (int argc, char * argv[])

- **argc = arguments count**
  - **main** 함수에 전달될 **argument**의 개수
  - 기본 값은 **1**
- **argv = arguments vector**
  - **main** 함수에 전달될 **argument vector** (주로 문자열)
  - **argv[0] =** 프로그램명
- 따라서 프로그램에 붙는 옵션의 실제 개수 **= argc-1**
- 예) **ls -al**
  - **argc = 2**
  - **argv[0] = "ls"**
  - **argv[1] = "-al"**

Dynamic Memory Allocation

- 프로그램이 실행되는 도중 **Memory**를 할당하는 방식
- **Heap** 영역에 해당 메모리가 할당 된다.
- **malloc (Memory allocate)**
  - **Heap** 영역에 해당 크기만큼의 메모리를 할당할 때
  - **void* malloc(size_t size);**
- **free**
  - **Heap** 영역에 할당한 메모리를 해제(반환)할 때
  - **void* malloc(size_t size);**
  - **void free(void* ptr);**
- 항상 할당한 만큼 해제를 해줘야 한다.

→ **malloc**과 **free**는 항상 짝을 이룬다.

LAB PREPARATION

실습 환경 구성 준비

- **radare2** 설치
- 실습에 필요한 **Package** 설치
  - **sudo apt-get install build-essential**
  - **sudo apt-get install libc6-dev-i386**
- **git clone https://github.com/radare/radare2**
- **cd radare2**
- **sys/user.sh**
- 설치를 마친 후 다음 **command**를 실행
  - **sudo sysctl -w kernel.randomize_va_space=0**

LAB TASK

Buffer OverFlow - (1)

■ 다음과 같은 **Code**를 작성한다. **(File Name : buf0.c)**

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[])
{
        char str1[6]="hello";
        char str2[6];

        gets(str2);

        if (strncmp(str1, str2, 6) == 0)
                printf("Both Strings are equal to %s \n", str1);
        else
                printf("strings are not equal \n");
}
```

Buffer OverFlow - (1)

- 작성을 마친 후 다음 명령어로 **compile** 한다.
- **gcc -fno-stack-protector -m32 buf0.c - o buf0**
- **hello**를 입력하여 정상 동작하는지 확인
- 이후 **BOF**를 발생시켜 동일한 값으로 인식하게 하는 입력값 **3**가지를 찾기

Buffer OverFlow - (2)

■ 다음과 같은 **Code**를 작성한다. **(File Name : buf1.c)**

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
typedef struct password
{
        char str1[8];
        char str2[8];
        int pass;
} password_t;

int main(int argc, char *argv[])
{
        password_t check_password;
        strncpy(check_password.str1, "fuzzball", 8);
        check_password.pass = 0;
        printf("Enter the password:\n");
        gets(check_password.str2);
        printf("pass is %d\n",check_password.pass);
        if (strncmp(check_password.str1, check_password.str2,8) == 0)
                check_password.pass = 1;
        if(check_password.pass == 1)
                printf("Password is correct! \n");
        else
                printf("Incorrect password %s %s \n", check_password.str1, check_password.str2);
}
```

Buffer OverFlow - (2)

- 작성을 마친 후 다음 명령어로 **compile** 한다.
- **gcc -fno-stack-protector -m32 buf1.c - o buf1**
- **fuzzball**을 입력하여 정상 동작하는지 확인
- 이후 이전 **Task**처럼 **BOF**를 발생시켜 **fuzzball**이 아닌 완전 다른 **8**개의 문자를 사용하여 **Password check**를 우회하는 것이 가능함
- **Hint :** 입력한 비밀번호는 **8**자리보다 커야함

Buffer OverFlow - (2)

- 해당 **Task**를 완료하기 위해서는 **ASCII**문자 이외의 값을 입력해야 한다.
- 온라인에서 **Hex – ASCII Converter**를 사용하여 **0x61**이 나타내는 문자가 무엇인지 확인할 것.
- **0x20**은 무엇을 나타내는지 확인하고 문자 **'5'**의 **Hex**값 표현을 찾을 것.
- 문자 **'5'**가 아닌 실제 숫자 **0x5**를 입력하기 위해 프로그램에 **ASCII** 문자 범위가 아닌 값을 입력하는 방법을 찾아낼 것.
- **Hint : bash script, python, perl** 등 다양한 방법이 존재
- 최종적으로는 **BOF**를 발생시켜 **fuzzball**이 아닌 완전 다른 **8**개의 문자를 입력하여 **Password check**를 우회하는 방법을 찾을 것

Buffer OverFlow - (3)

■ 다음과 같은 **Code**를 작성한다. **(File Name : buf2.c)**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void secret_function()
{
        printf("\n**********************************\nProgram failed successfully!\n**********
**********************\n\n");
}

int get_from_user()
{
        char input_number[8];
        printf("we are now in get_from_user function ... \n");
        printf("enter your group number: ");
        gets(input_number);
        return atoi(input_number);
}

int main(int argc, char **argv[])
{
        printf("we are now in the main function ... \n");
        int groupnumber = get_from_user(groupnumber);
        printf("your group number is %d\n", groupnumber);

        return 0;
}
```

Buffer OverFlow - (3)

- 이번 **Task**는 코드의 수정 없이 **secret_function()**을 호출하는 것이 목표이다.
- **radare2**를 사용하여 **Stack**을 분석
- **r2 –d ./buf2**
- 다양한 명령어들이 있으므로 **?**를 입력하면 그에 관련된 도움말들이 출력됨
- 크게 **db, dc, afl, s, px, pxr@esp**와 같은 명령어들을 사용하게 될 것임

Buffer OverFlow - (3)

- 이번 **Task**는 코드의 수정 없이 **secret_function()**을 호출하는 것이 목표이다.
- **radare2**를 사용하여 **Stack**을 분석
- **r2 –d ./buf2**
- 다양한 명령어들이 있으므로 **?**를 입력하면 그에 관련된 도움말들이 출력됨
- 크게 **db, dc, afl, s, px, pxr@esp**와 같은 명령어들을 사용하게 될 것임

Buffer OverFlow - (3)

- **BOF(Stack Smashing)**이 **secret_function()**을 호출하게 하는데 어떤 역할을 하는지 설명할 것.
- **radare2**를 통해 **secret_function**의 주소를 찾을 것
- **radare2**를 이용하여 **get_from_user()**에서 **breakpoint**를 설정하고 코드를 계속 실행하여 **return address**를 찾을 것.
- **return address**를 통해 입력해야 하는 문자열의 최소 길이를 확인하고 이를 **secret_function()**의 **address**를 **little-endian** 순서로 보내어 **BOF**를 성공시킬 것.

LAB QUESTION

Lab Question

1. **BOF-(1)**에서 **hello**를 입력하여 **Both strings are equal to hello**가 올바르게 나왔을 경우 **gets()**가 **call**되기 전과 된 이후의 경우를 생각하여 다음 표를 완성하세요.

	str1	str2
Before gets()		
After gets()		

2. **BOF-(1)**에서 **hello**를 입력하지 않고도 **Both strings are equal to XXX**이 나오는 경우, 왜 정상적으로 동작하는 것처럼 보이는지 설명하세요.

Lab Question

3. 다음 표는 **BOF-(2)**에서 변수 **check_password**가 있는 스택입니다. 각 주소마다 단일 바이트가 들어있고 또한 주소 역시 **0x01**부터 시작한다고 가정했을 때 **Task**에서 입력한 값이 어떻게 입력되는지 표를 완성하세요.

Address	Stores a byte for variable	Value	Address	Stores a byte for variable	Value
0x01	check_password.str1	f	0x0a		
0x02			0x0b		
0x03			0x0c		
0x04			0x0d		
0x05			0x0e		
0x06			0x0f		
0x07			0x10		
0x08			0x11		
0x09					

Evaluation

- **Lab Task** 진행
  - **3**개의 **Task**에서 진행한 과정을 캡처하고 설명할 것
- **Lab Question**
  - 주어진 문항에 대한 답과 해결 방안에 대해 간략하게 서술
- **Lab Task** 수행 결과를 위와 같이 명시한 대로 캡처하여 **MS Word** 또는 **PDF** 파일로 결과를 제출할 것.

Evaluation

- 과제 제출 기한 : 2019/11/11 23:59
- 과제 제출 시 메일 제목은 ‘본인 이름_학번’으로 제출
  - 예) 이석원_2019101059
- sevenshards00@gmail.com으로 보낼 것.

Q&A