

# NETWORK SECURITY

한양대학교 소프트웨어융합대학 소프트웨어학부  
이연준 교수

# 주요 사항

- **Buffer OverFlow (BOF)** 방어 기법에 대한 이해
- **Lab Preparation**
  - 실습 환경 구성
- **Lab Task**
- **Lab Question**
- **Evaluation**

# Memory Protection

- **BOF** 공격 기법은 주로 메모리 (**Stack, Heap**)의 경계 값을 넘어섬에 따라 발생하는 취약점을 이용한 공격기법
- 특히 **C Language**는 메모리의 경계 값을 넘어서는지를 검사하는 내부 안전 장치가 존재하지 않음
- 따라서 메모리의 경계를 넘어서지 않게 보호할 필요가 있음

# Address Space Layout Randomization (ASLR)

- 메모리 보호 기법 중의 하나로 **Stack**이나 **Heap** 및 **Library**가 배치되는 주소를 항상 무작위의 공간에 배치
  - 이는 메모리 상의 공격을 어렵게 하는 효과를 지님
  - 이전 실습에서 프로그램을 디버깅했을 때 항상 고정된 주소 값에 프로그램이 로딩되는 것을 확인할 수 있었음
- **ASLR 옵션**

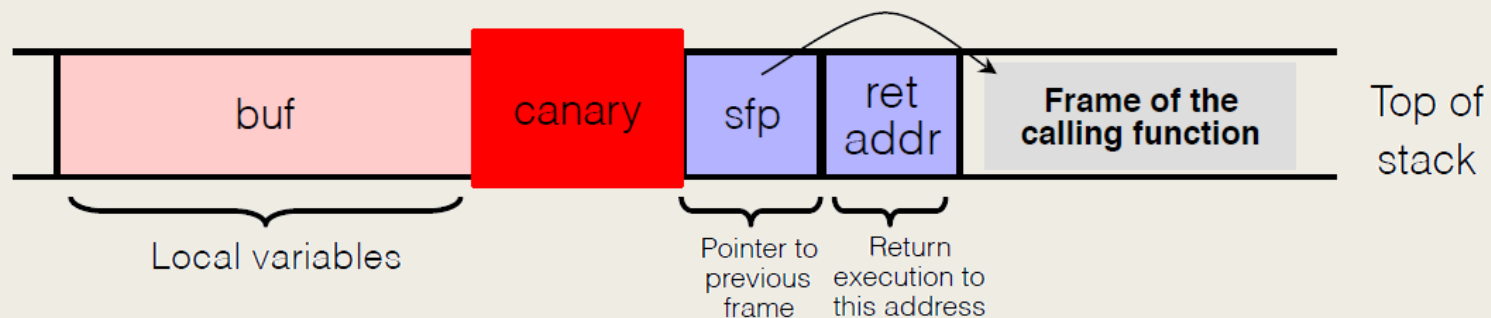
Option	설정 범위
<b>sysctl -w kernel.randomize_va_space=0</b>	<b>ASLR 해제</b>
<b>sysctl -w kernel.randomize_va_space=1</b>	<b>Stack과 Library</b>
<b>sysctl -w kernel.randomize_va_space=2</b>	<b>Stack, Heap 및 Library</b>

# Stack Canaries (Cookies)



# Stack Canaries (Cookies)

- **Function** 진입 시 **Stack**에 **Saved Frame Pointer**와 **return address** 정보를 저장할 때, 이 정보가 공격자에 의해 **overlap** 될 수도 있음
- 이를 보호하기 위해 **Stack** 상의 변수들의 공간과 **SFP** 사이에 특정한 값을 추가하는 기법
- **StackGuard, ProPolice, Stack Smashing Protector (SSP)**



# Stack Canaries (Cookies)

- 이 **Canary** 값이 변조가 되면 **Stack Smashing**으로 간주하여 실행 중인 프로그램이 종료됨
- 이전 실습에서 **gcc**를 통해 컴파일할 때 다음과 같은 옵션을 추가하여 컴파일 하였음
  - **gcc -fno-stack-protector -m32 buf0.c -o buf0**

## ■ SSP 옵션

Option	기능
<b>-fstack-protector (default)</b>	<b>SSP 적용 (취약한 Function들)</b>
<b>-fno-stack-protector</b>	<b>SSP 사용 않음</b>
<b>-fstack-protector-all</b>	<b>모든 Function에 SSP 적용</b>

# No eXecution / Data Execution Prevention (NX / DEP)

- **NX / DEP**는 **Data** 영역에서 **Code Execution**을 방지
- **Data** 영역 = **Stack / Heap**
- **Code**의 실행을 허용할 경우 **Shell Code** 삽입에 취약

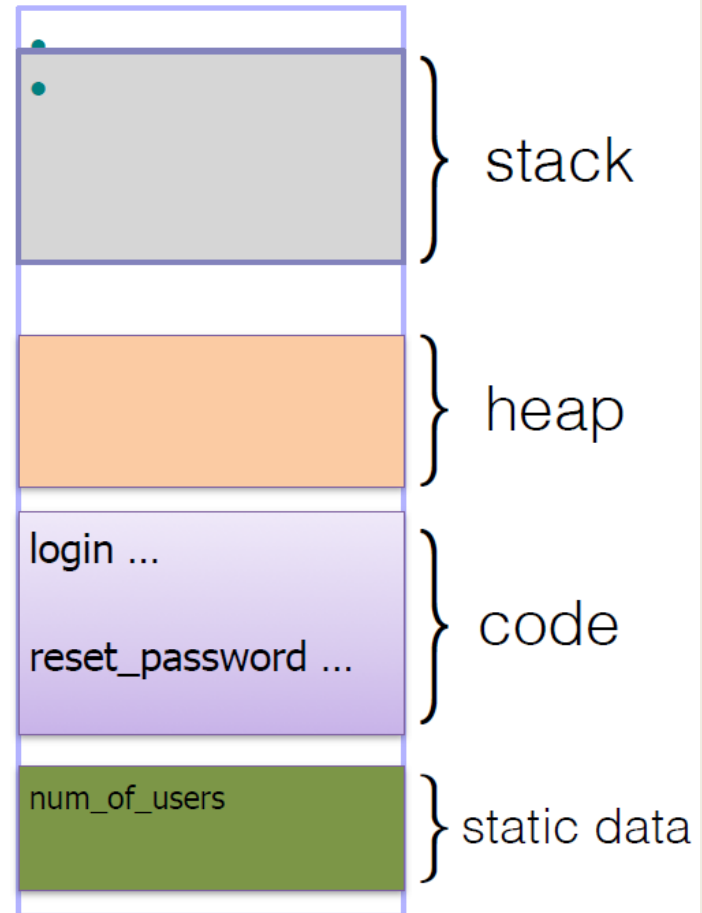


# No eXecution / Data Execution Prevention (NX / DEP)

```
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));
}
```



# **LAB PREPARATION**

# 실습 환경 구성 준비

## ■ dos2unix 설치

- **sudo apt-get install dos2unix**

## ■ checksec.sh 다운로드

- **wget <http://www.trapkit.de/tools/checksec.sh>**
- **chmod +x checksec.sh**
- **dos2unix checksec.sh**

# **LAB TASK**

# SSP Activate

- 이전 실습에서 작성했던 코드들은 공통적으로 **SSP**를 적용하지 않고 컴파일을 하였음
  - **fno-stack-protector**
- 그러므로 이번에는 **SSP**를 적용한 상태로 컴파일을 할 것
  - **e.g., gcc -m32 buf1.c -o buf1**
- 컴파일을 마친 이후에는 **checksec**를 통해 **stack canary**가 적용되었는지 확인하고 캡처할 것
  - **./checksec.sh --file 실행파일명**
- 이전처럼 **BOF** 공격을 실행하고 어떤 결과가 나오는지 캡처할 것

# ASLR Activate

- 이전 실습에서는 **ASLR**을 비활성화한 상태로 **BOF** 실습을 진행하였음
- **ASLR**을 다시 활성화 한 후 이전 실습의 결과와 비교할 것
  - **sudo sysctl -w kernel.randomize\_va\_space=2**
- 이전 실습의 **BOF-(3)**에서 생성한 실행파일을 **debugger**를 통해서 주소값이 달라지는 것을 확인하여 캡처할 것

# **LAB QUESTION**

# Lab Question

**1.Stack Canary** 기법은 특정한 값을 삽입하여 **canary** 값의 변조 여부를 확인하여 **BOF**가 발생했는지를 확인하는 기법입니다. 하지만 해당 기법에는 취약점이 존재합니다. 이 취약점은 무엇이며, 해당 기법의 파훼법에 대해 간략하게 설명하세요.

**2.ASLR**은 **Stack**이나 **Heap, Library**를 임의의 메모리 주소로 배치하는 기법입니다. 이를 통해 메모리 상의 공격을 어렵게 하는 효과가 있습니다. 해당 기법을 통해 모든 **BOF**를 막을 수 있습니까? 이에 대한 답과 그 이유를 서술하세요.

**3.C, C++**에서 사용되는 함수 중 **strcat(), strcpy(), gets(), scanf()**와 같이 **BOF**에 취약한 함수들이 있습니다. 해당 함수들이 가지고 있는 취약점은 무엇이며 이를 해결하기 위한 방법에 대해서 설명하세요.



# Evaluation

## ■ Lab Task 진행

- 2개의 **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 : 2019/11/18 23:59
- 과제 제출 시 메일 제목 및 파일명은 ‘본인 이름\_학번’으로 제출  
– 예) 이석원\_2019101059
- [sevenshards00@gmail.com](mailto:sevenshards00@gmail.com)으로 보낼 것.

# Q&A