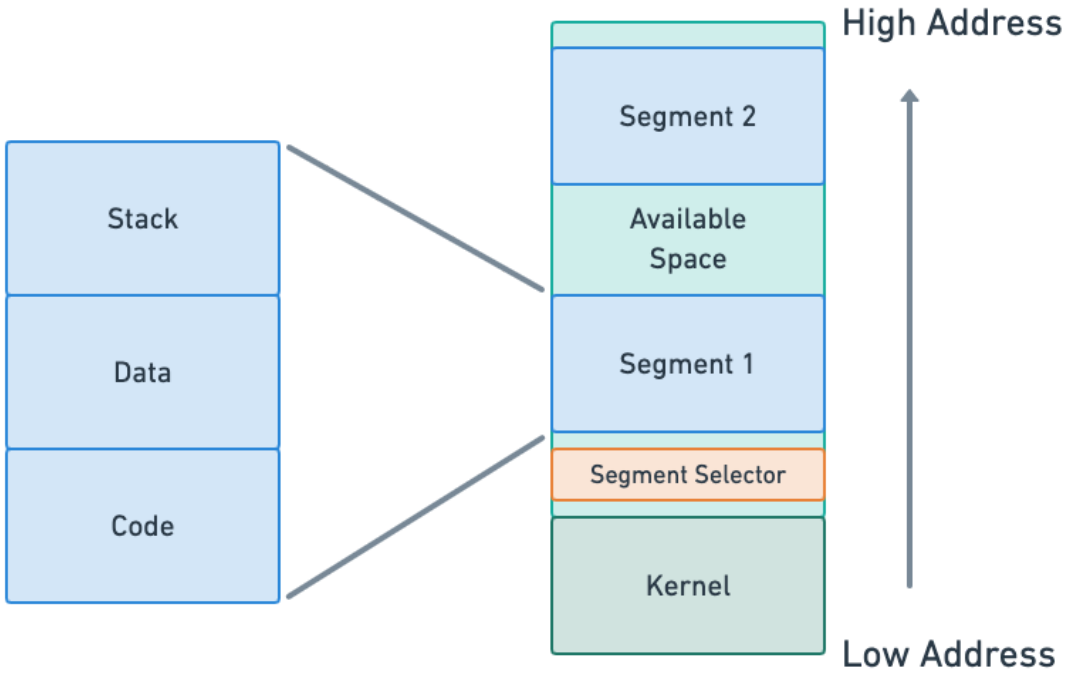
해커 지망자들이 알아야 할 Buffer Overflow Attack 의 기초 - 정리

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/e5cdcf8e-7068-455c-bb88-7226099bba5c/buffer_overflow_foundation_pub.pdf

By 달고나 (Dalgona@wowhacker.org), Wowhacker Team

- 메모리 & 레지스터 구조
  - Week 2 - 레지스터 정리 안의 내용과 비슷하니 간단하게만 다뤄보자.



  - 시스템 운영에 필요한 명령어 집합은 Kernel에서 찾기에 Kernel의 위치는 항상 고정됨
  - 하나의 프로그램 → 하나의 프로세스 : Segment라는 단위로 묶여 Available Space에 저장시킴

- 현재의 시스템은 멀티 태스킹이 가능하므로, 여러 개의 Segment가 같이 Available Space 위에 올라갈 수 있음.
- Segment는 Code Segment, Data Segment, Stack Segment로 구성됨.
  - Code Segment : 시스템이 알아들을 수 있는 명령어, 즉 기계어 코드들이 저장되는 장소. Segment가 어디에 위치할지 모르기에 Logical Adress를 사용하여 작성이 된다.
  - Data Segment : 프로그램 실행시 사용되는 데이터 (전역 변수)들이 저장됨. 이는 다시 현재 모듈의 data structure, 상위 레벨로부터 받아들이는 데이터 모듈, 동적 생성 데이터, 다른 프로그램 과 공유하는 공유 데이터 부분으로 나뉘어짐.
  - Stack Segment : 현재 수행되고 있는 handler, task, program이 저장하는 데이터 영역. 곧 사용하게 될 버퍼가 여기에 자리잡음

프로그램의 실행과 스택의 모습에 관련된 내용은 Week 2에서 자세히 다루었다. 이 PDF를 통해서는 실제 프로그램이 실행될 때 이론과 다르게 실제로 어떻게 진행되는지 알 수 있었고, 그 내용들을 정리하겠다.

- Stack은 Word (4byte)단위로 자란다. 따라서 프로그램에서 할당한 버퍼의 크기와 실제로 할당한 버퍼의 크기는 다를 수 있다.
- 또한, 컴파일러의 종류와 버전에 따라 dummy값이 들어가는 등 실제 할당되는 버퍼의 크기는 크게 달라질 수 있다.
- RET instruction은 EIP 레지스터에 return address를 POP하는 것과 같다. 물론 EIP 레지스터는 직접 수정할 수는 없지만 말이다.

Buffer Overflow의 이해

- Buffer : 시스템이 연산 작업을 하는데 있어 필요한 데이터를 일시적으로 메모리 위에 저장하는 그 공간
  - 함수가 끝나도 free( ) 없이는 반환되지 않는 malloc( )과 같은 데이터 저장 공간과는 다르다.
- 발생 원인 : 미리 준비된 Buffer의 크기보다 큰 데이터가 들어갈 때 발생. 이 때, 이전 함수의 base pointer, return address가 저장되어 있는 공간을 침범하여 공격자의 의도대로 프로그램을 이용할 수 있게 된다.

- 이용 : Return Address의 값을 임의로 공격 코드의 주소로 바꾸면 프로그램의 실행 중 EIP에 공격 코드의 주소가 들어가 공격을 시도할 수 있다.
- Byte order : 실제 메모리에 들어가는 정보는 입력한 정보와 다를 수 있다. 바로 Little Endian 방식과 Big Endian 방식의 차이 때문이다.
    - Little Endian : 바이트의 순서가 높은 메모리 주소에서 낮은 메모리 주소로 이루어짐
    - Big Endian : 바이트의 순서가 낮은 메모리 주소에서 높은 메모리 주소로 이루어짐
        - ex) 74E3FF59라는 값을 저장, LE : 59 FF E3 74 | BE : 74 E3 FF 59