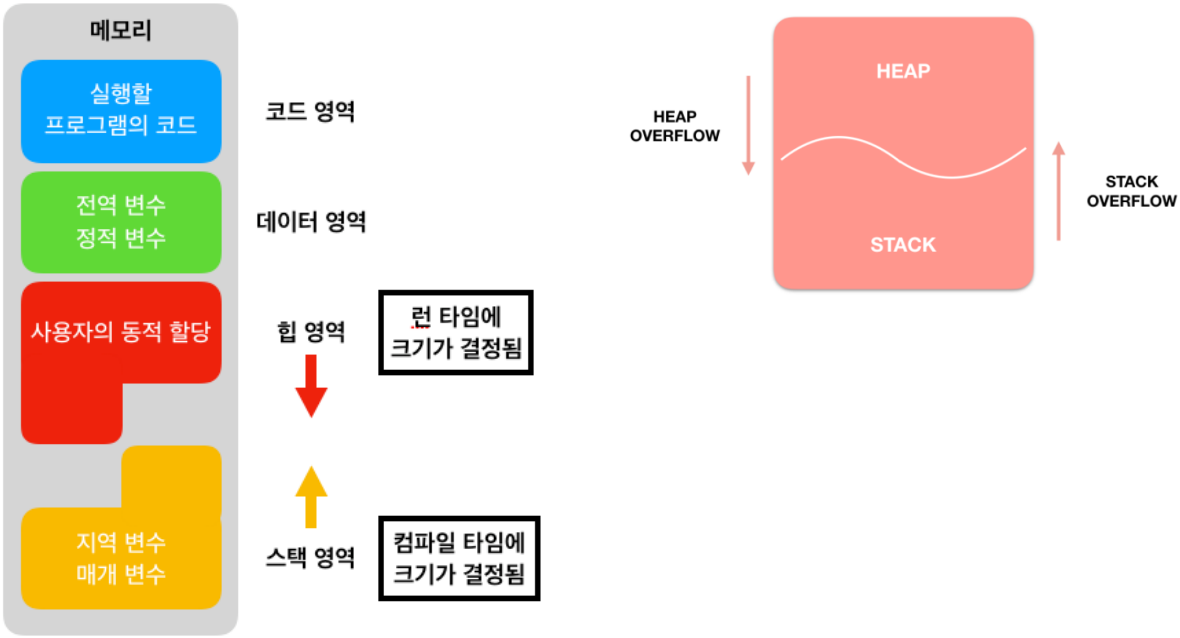
#자료조사

0. memory



메모리 구조는 크게 code, data, heap, stack으로 구성

힙과 스택은 버퍼 공간으로 사용자가 정해주는 대로 변하는 크기를 가지기 때문에, 이를 이용하여 버퍼 오버 플로우 공격을 수행할 수 있다.

힙은 동적으로 할당된 것들에 대해 버퍼 공간이다. 따라서 대표적으로 `malloc`을 이용한 공격이 가능하다.

스택은 지역 변수에 대한 버퍼 공간이다. 따라서 `입력 값`을 이용한 공격이 가능하다.

버퍼 오버플로우 공격의 원리는 오른쪽 그림과 같다.

1. 스택 버퍼 공격

1. FSB(Format String Bug) (Format String Buffer Overflow)

   https://eunice513.tistory.com/16

   - 데이터 형태에 대한 불명확한 정의에 의한 문제

- 메모리를 조작하여 printf의 형식을 바꾸어 주소값을 출력하도록 한다.
- 알아낸 주소 값을 이용해 임의의 메모리 주소에 데이터를 쓸 수 있다.

→ 버퍼 오버플로우 공격 아님

2. RTL(Return to Libc)
   - 리눅스의 메모리 보호 기법 중 하나인 NX bit를 우회하기 위해 사용되는 공격 기법, 메모리에 적재되어 있는 공유 라이브러리에서 원하는 함수를 사용할 수 있는 공격 기법
   - NX bit : Stack Segment의 실행 권한 제한 → stack에 쉘 코드를 저장하고 이를 실행하여 악용되는 것을 막기 위한 메모리 기법 (한 마디로 관리자 권한을 탈취하는 쉘 코드의 실행을 막음)
   - 함수 호출에 대한 이해가 필요하다.
   - libc에 적재되어 있는 함수 중 하나인 system() 함수에 대한 예제다. c 코드에 `system("echo 0")`을 작성하면 프롬프트에서 `echo 0`을 한 것과 같다.
   - **system 명령어의 주소와, 명령어에 넘겨주는 인자의 주소를 확인한다. 이 정보를 이용해 버퍼 오버플로우 공격 (쓰레기값 + system 명령어의 위치 + 4바이트 쓰레기 값 + 명령어에 넘겨주는 인자의 주소) → 이미 배운 것과 마찬가지로 RET을 덮어쓰는 방식**
3. RTL Chaining

   https://d4m0n.tistory.com/80
   - RTL 기법을 응용해 라이브러리 함수의 호출을 연계
   - SFP에 명령어의 주소를 넣어 스택의 포인터를 다음 함수로 위치
4. GOT Overwrite

   https://d4m0n.tistory.com/83?category=796362
   - 동적 라이브러리 호출 시 사용되는 PLT & GOT을 이용하는 공격 기법
   - PLT → GOT → 함수의 실제 주소 : GOT을 원하는 함수의 주소로 변조시키는 기법
   - ex. GOT에 system() 함수의 주소 overwrite
5. ROP(Return Oriented Programming)
   - Nxbit, ASLR을 우회하는 기법
   - ASLR : 프로세스 실행 시, 메모리를 매핑할 때마다 매번 주소값을 바꿔준다.
   - 기본적으로 RTL 기법 사용 → RTL을 연속적으로 사용하는 기법

- RTL, RTL Chaining, GOT Overwrite 사용

2. 힙 버퍼 오버플로우

https://ii4gsp.tistory.com/64

- 힙은 스택과 다르게 요구하는 블록의 크기나 요구, 횟수 순서가 일정한 규칙이 없다. 힙의 기억 장소는 포인터를 통해 동적으로 할당받고 돌려준다 - 리스트, 트리, 그래프 등
- 동적 메모리 할당을 위해 할당, 해제를 수동으로 해줘야 한다. 해제를 제대로 하지 않았을 때의 메모리 취약점을 이용하는 경우도 있으니 주의!
- 스택 오버플로우처럼 직접적인 RET 조작은 불가능, **프로그램의 함수 포인터를 조작**
- 함수 포인터를 조작하여 원하는 함수가 실행되도록 한다. (오버플로우를 발생 시킴)

3. 공격을 위해 필요한 개념

1. ASLR(Address Space Layout Randomization) : 스택, 힙, 라이브러리의 주소를 랜덤화하는 메모리 보호 기법

   리눅스에서 메모리에 대한 공격을 할 때 일반적으로 이것을 해제하고 시작한다. 과제에서도 적용되었다.

2. DEP(Data Execution Prevention)

   실행되어서는 안 되는 메모리 영역에서 코드의 실행을 방지해 임의의 코드가 실행되는 것을 방지하는 방어 기법