바이너리 실행 파일의 버퍼 오버런 분석기

최재승, 이광근 교수님
서울대학교 프로그래밍 연구실

1. 목표

- 바이너리 실행파일(binary executable)을 분석하여 **버퍼 오버런** 취약점 탐지하기
- 소스 코드 없이 바이너리 형태로만 접근 가능한 소프트웨어도 분석할 수 있도록

2. 과제

- 바이너리에는 프로그램의 메모리 구조에 대한 정보가 명시적으로 드러나지 않음
  - 각 함수에 선언된 변수, 배열 크기, 배열 접근 시 첨자(index)로 사용되는 값 등
  - C 소스 코드에는 명시적으로 드러나던 것들
- 어떤 경우에 버퍼 오버런이 일어난다고 판단할 수 있을 것인가?

3. 관찰

- 바이너리 실행파일은 스택 포인터(stack pointer) 레지스터에 일련의 정수를 더함으로써 지역 변수의 주소를 만듦
- **스택 오프셋**에 해당하는 정수값은 어셈블리에 상수로 박혀(hard-coded) 있음
- 반면, **배열 인덱스**에 해당하는 정수값은 레지스터나 메모리로부터 읽어옴
- 다음과 같은 상황을 버퍼 오버런으로 판단 가능
  - 서로 다른 스택 오프셋을 가진 두 메모리 주소가 있을 때, **이 두 주소가 접근하는 메모리 공간이 서로 겹쳐지는 것**
- 위의 관찰 및 버퍼 오버런의 정의는 확장하여 힙 메모리에서도 적용할 수 있음

4. 분석기 설계

- 요약 해석 이론의 틀에 기반하여 정적 분석기 설계
- 바이너리의 메모리 접근 패턴을 잘 포착해 낼 수 있도록 '요약 실행 의미'를 정의 (e.g. 메모리 주소의 스택 오프셋과 배열 인덱스를 구분하는 것 등)
- 프로그램의 각 지점에서 레지스터와 메모리에 저장된 값을 안전(sound)하게 요약(abstraction) 가능
  - 이 요약된 값을 토대로, 버퍼 오버런 여부를 판단

(C source)

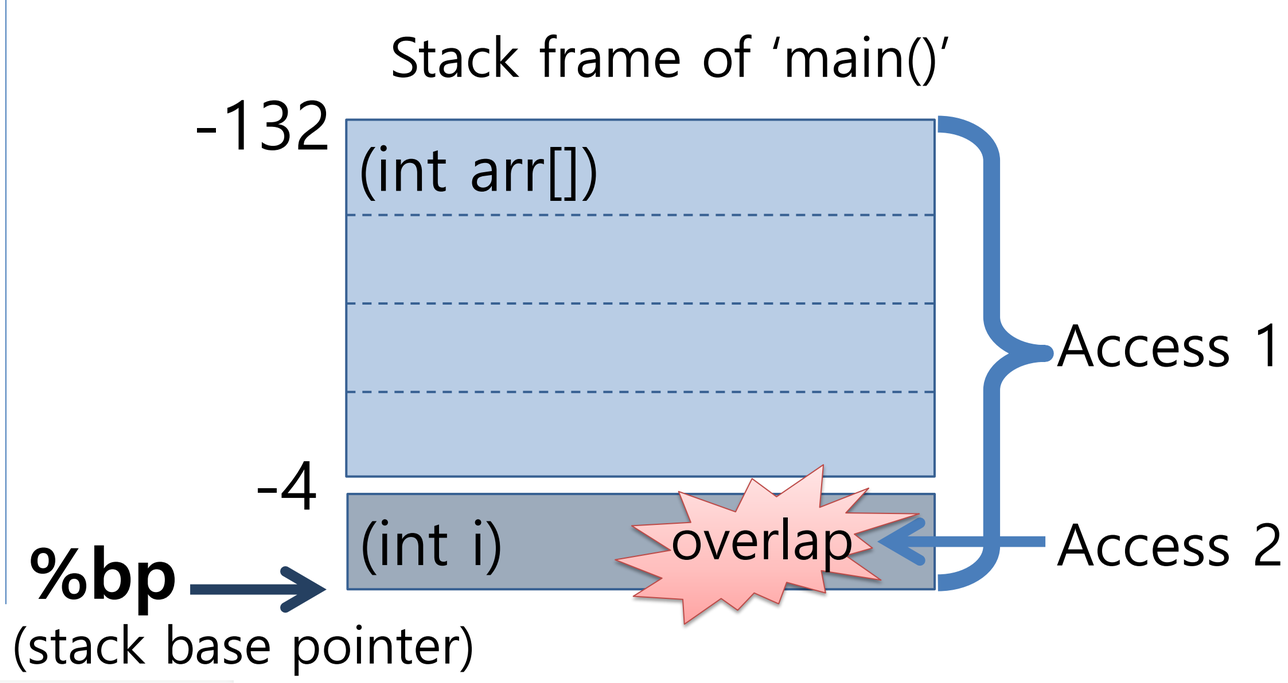
```
int main(void)
{   int arr [32];
    int i ;
    for(i = 0; i <= 32; i++)
        arr [ i ] = 1; // overflow
}
```

compile

(Binary's assembly code)

```
...
mov $0, (%bp - 4); // int i = 0;
cmp (%bp - 4), 32; // if (i > 32)
jg  label_end;     //   goto label_end;
mov (%bp - 4), %reg_1  // (load i to register)
mov $1, (%bp - 132 + 4 * reg_1) // arr[i] = 1;
...
```

stack offset *array index*



SEOUL NATIONAL UNIVERSITY

Programming Research Laboratory

ROSAEC center
Research On Software Analysis for Error-free Computing