시스템 프로그래밍

6주차: 실습 프로젝트 - Bomblab

T A

강범우, IT/BT 701호

qjadn0630@hanyang.ac.kr

Bomblab 과제

- 과제의 목표
  - 바이너리의 역어셈블을 통해 기계수준 프로그램의 구조 이해
  - 일반적인 디버거 사용법과 역엔지니어링(리버싱) 체험
  - 6단계의 암호 입력 과정을 분석하여 바이너리 폭탄 해제

- 과제 수행 환경
  - Ubuntu (가상머신) - 18.04 버전 권장
  - gdb 디버깅 환경

- 결과 제출 방법 및 기한
  - **"bomb" 디버깅을 통해 solution_format.txt에 phase별 정답 기입 후 commit (다른 파일 수정 X)**
  - 10월 20일(화) 23:59까지 commit된 것만 인정

바이너리 폭탄 (Binary Bomb)

- 리눅스 실행 파일 (bomb)
  - 디버깅 & 리버싱 대상 파일

- 6단계의 사용자 입력을 요구
  - 내장된 암호와 일치하는 경우 다음 단계로 진행
  - 틀린 데이터 입력 시 "BOOM!!!" 메시지와 함께 종료됨

바이너리 폭탄 실행

- 반복 실행 가능
  - 폭발 메시지로 종료되어도 다시 실행할 수 있음.
- 일괄 실행 가능
  - 암호를 저장한 텍스트 파일을 인자로 주어서 실행할 수 있음.

```
prof@student2:~/bomb1$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
wrong code!!!

BOOM!!!
The bomb has blown up.
```

```
prof@student2:~/bomb1$ ./bomb solution.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?

BOOM!!!
The bomb has blown up.
```

바이너리 폭탄 실행

- 실행 중지
  - Ctrl-C 키를 입력하여 프로그램 실행을 강제로 종료할 수 있음.

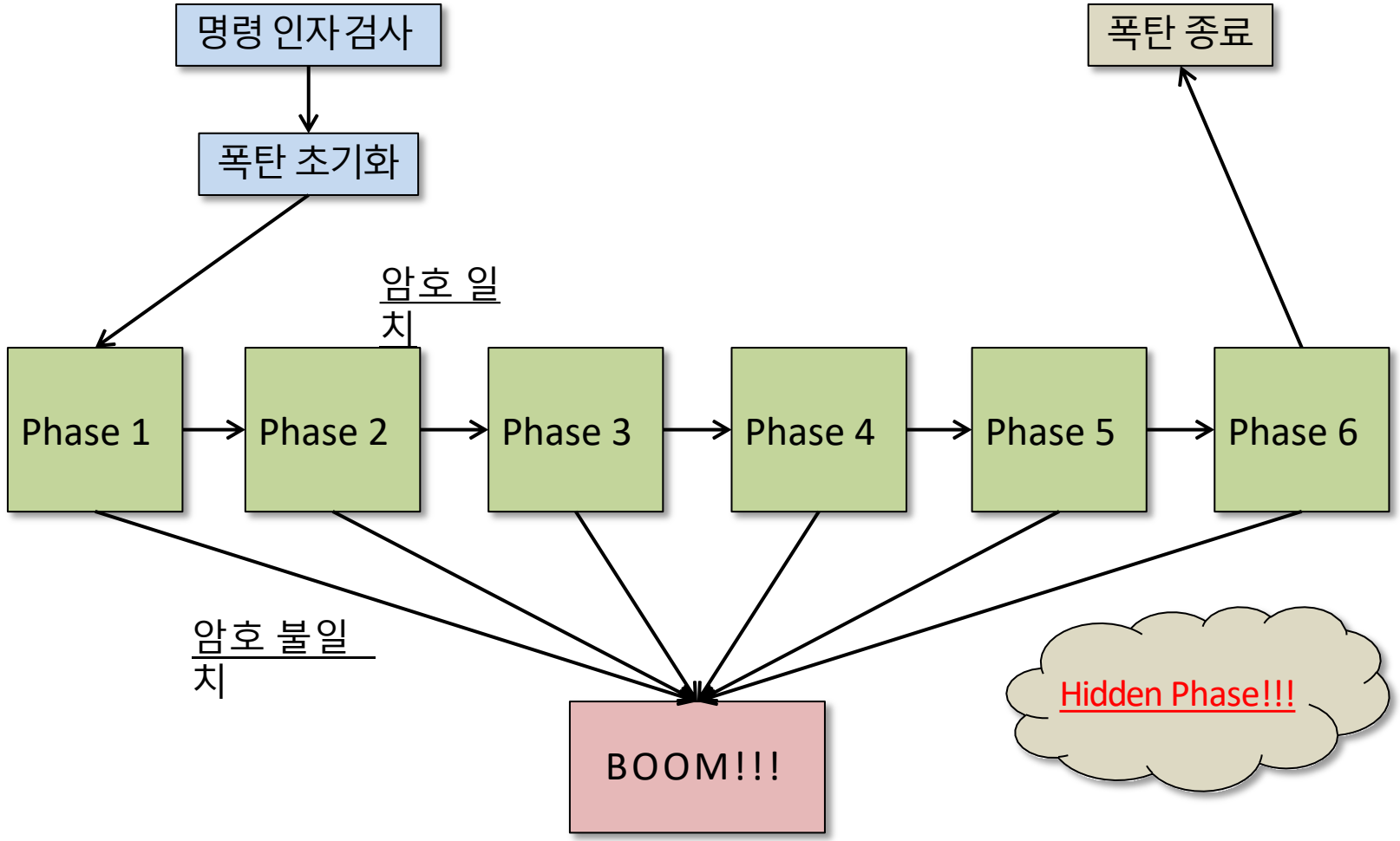
```
prof@student2:~/bomb1$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
^CSo you think you can stop the bomb with ctrl-c, do you?
Well...OK. :-)
```

과제 수행 방법

- GitLab 프로젝트에 업로드 된 파일을 git pull을 사용하여 로컬 저장소로 복사 후 진행
  - 폭탄의 암호 내용은 서로 다름.
- 바이너리 폭탄의 해체
  - 리눅스 환경에서 폭탄 프로그램을 실행함.
    - 암호 문자열 입력
    - 암호를 저장한 텍스트 파일을 입력으로 줄 수 있음.
  - 암호 해독 방법
    - 폭탄 프로그램의 암호 확인 알고리즘 분석
      - 역어셈블러 및 gdb 디버거를 이용하여 어셈블리 프로그램 분석
      - 내장된 암호 문자열 조사등

과제 결과 확인 방법

- bomb.c 참조 (main 함수)



폭탄 해체 방법

- 바이너리 폭탄의 실행과 단계별로 설정된 암호 입력

- 디버거를 이용한 역엔지니어링 (reverse engineering)
  - Gdb 활용
  - 어셈블리 코드 수준에서 프로그램 구조 분석
    - 함수의 호출과 인자 전달 방식 이해
    - 데이터 영역 조사 방법 이해
    - 조건문, 순환문 등의 프로그램 구조 이해
  - 각 단계 별로 암호를 확인하는 방법(알고리즘)은 서로 다름

단계별 암호 확인 방법에 대한 힌트

- Phase_1: 문자열 비교
- Phase_2: 순환문 실행
- Phase_3: 조건문과 switch 문 실행
- Phase_4: 재귀적 호출(recursive call)과 스택 동작
- Phase_5: 배열(array) 자료구조
- Phase_6: linked list, 포인터, struct 자료구조

폭탄 해체 예시 (gdb 활용)

`$ gdb bomb`

- breakpoint를 설정하여 run-time 디버깅
  - 분석하고자 하는 함수에 breakpoint 설정
    - **`b phase_1`**
  - 프로그램의 일부분을 역어셈블
    - **`disas phase_1`**
  - Gdb로 프로그램 실행
    - **`run`**

폭탄 해체 예시 (1단계 암호 찾기)

- phase_1 함수 분석 (디스어셈블된 코드)

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000555555555204 <+0>:     sub    $0x8,%rsp
   0x0000555555555208 <+4>:     lea    0x15e1(%rip),%rsi        # 0x5555555567f0
   0x000055555555520f <+11>:    callq  0x55555555556e0 <strings_not_equal>
   0x0000555555555214 <+16>:    test   %eax,%eax
   0x0000555555555216 <+18>:    jne    0x55555555521d <phase_1+25>
   0x0000555555555218 <+20>:    add    $0x8,%rsp
   0x000055555555521c <+24>:    retq
   0x000055555555521d <+25>:    callq  0x5555555557ec <explode_bomb>
   0x0000555555555222 <+30>:    jmp    0x555555555218 <phase_1+20>
```

상수를 레지스터 **%rsi**에 저장하고
strings_not_equal 호출함.
(암호 문자열 비교로 짐작됨)

%eax != 0 이면 (return value)
explode_bomb 호출로 점프
(폭탄 폭발 함수로 짐작됨)
* 리턴값이 1인 경우에 정상적으로 실행

폭탄 해체 예시 (1단계 암호 찾기)

- 실행 분석 및 메모리 영역 조사
  - **gdb bomb**
    - **(gdb) x/s 0x5555555567f0**

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000555555555204 <+0>:     sub    $0x8,%rsp
   0x0000555555555208 <+4>:     lea    0x15e1(%rip),%rsi        # 0x5555555567f0
   0x000055555555520f <+11>:    callq  0x5555555556e0 <strings_not_equal>
   0x0000555555555214 <+16>:    test   %eax,%eax
   0x0000555555555216 <+18>:    jne    0x55555555521d <phase_1+25>
   0x0000555555555218 <+20>:    add    $0x8,%rsp
   0x000055555555521c <+24>:    retq
   0x000055555555521d <+25>:    callq  0x5555555557ec <explode_bomb>
   0x0000555555555222 <+30>:    jmp    0x555555555218 <phase_1+20>
End of assembler dump.
(gdb) x/s 0x5555555567f0
0x5555555567f0: "I turned the moon into something I call a Death Star."
```

테스트 결과 예시

```
beomwoo@beomwoo-desktop:~$ ./bomb < solution_format.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

바이너리 파일의 역어셈블 팁

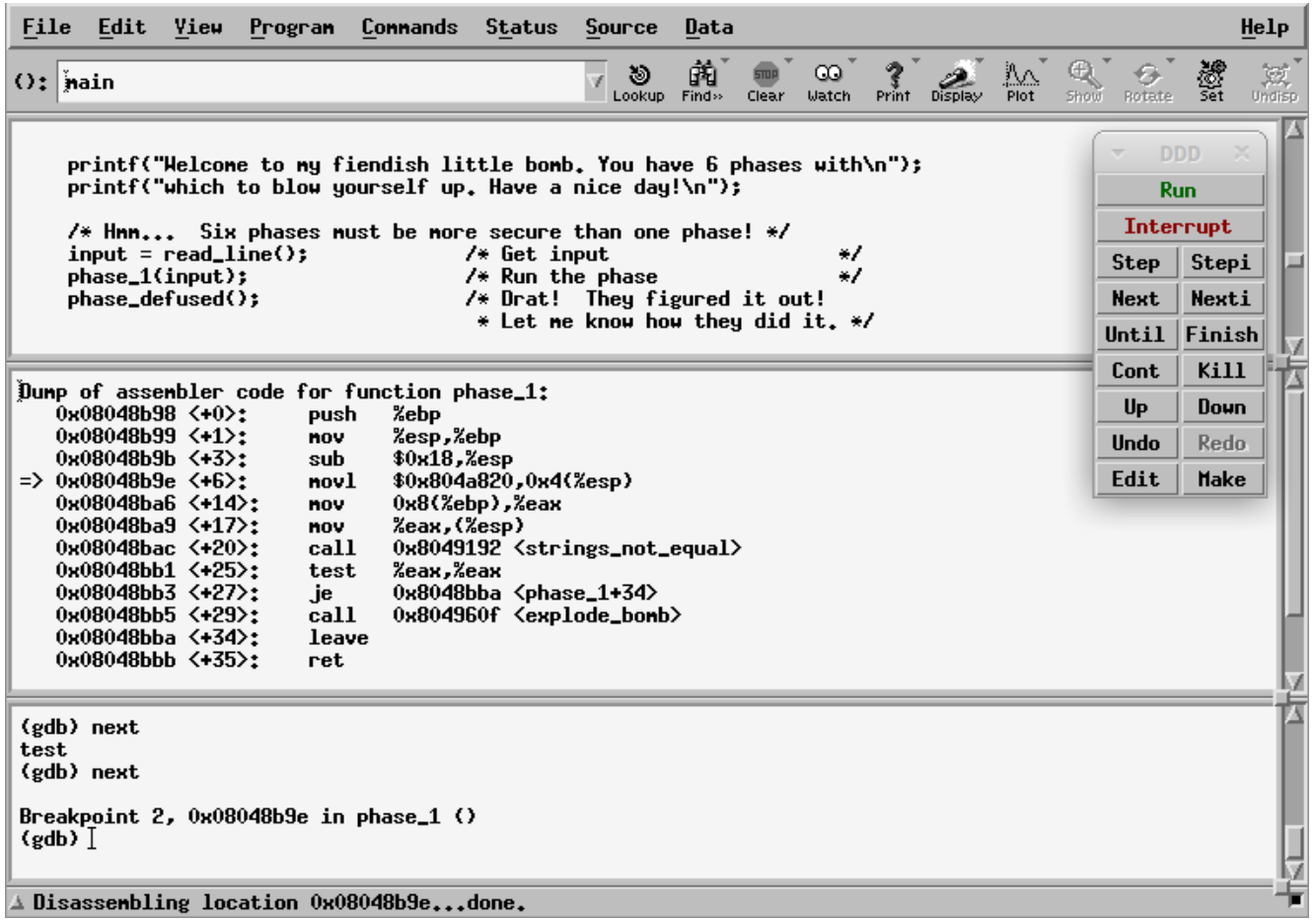
- objdump 명령어
  - **`objdump -d bomb`**
    - disassemble 된 프로그램 코드 출력
  - **`objdump -s bomb`**
    - 전체 memory contents 출력 (코드와 데이터)
- strings 명령어
  - **`strings bomb`**
    - object 파일 내의 문자열 데이터 출력

기본적인 GDB 명령어

- 실행
  - run [명령 인자]
  - continue, kill
  - stepi [k], nexti [k]
- 역어셈블
  - disas 함수 / 주소 [주소]
- 브레이크 포인트
  - break 함수 / *주소
  - delete [번호]
  - info break
- 레지스터 검사
  - print [/형식] $eax
  - info register (i r)
- 메모리 검사
  - print [/형식] 변수 / *주소 / *($Reg+k)
  - x/[개수][형식][단위] 주소 / 함수
- 스택 확인
  - bt
  - info frame / locals / args

참고: DDD (Data Display Debugger)

- gdb의 그래픽 프런트-엔드 프로그램



주요 GDB 명령어

- 프로그램 실행
  - run [명령 인자]
    - 디버깅할 프로그램 실행
  - continue, kill
    - 중단된 프로그램 실행 재개, 프로그램 실행 종료
  - step / stepi, next / nexti [숫자]
    - 소스 수준 / 명령어 수준에서 단계적 실행
  - finish / return [return 값]
    - 현재 함수의 return까지 실행 / 실행하지 않고 return
  - quit
    - gdb 종료

주요 GDB 명령어

- 중단점 (breakpoint)
  - break <함수이름 / 라인넘버 / *메모리주소>
    - 지정된 위치에 breakpoint 설정
  - info breakpoints
    - 설정된 breakpoint 표시
  - delete <breakpoint 번호>
    - 설정된 breakpoint 삭제
  - enable, disable <breakpoint 번호>
    - breakpoint 활성/비활성
  - clear <함수이름>
    - 함부 내의 모든 breakpoint 삭제

주요 GDB 명령어

- 메모리 확인
  - display [/표시형식] <변수이름>
    - 변수의 값이 유효한 코드 부분에서 값을 계속해서 표시
  - undisplay
    - display 모드를 종료
  - print [/표시형식] <변수이름 / *메모리주소 / 레지스터이름>
    - 현재 / 지정된 위치의 프로그램 소스 코드 표시
    - 표시형식: /d, /x, /t
  - x/[개수][표시형식][단위크기] <주소 / 함수>
    - 지정된 위치의 메모리 데이터를 지정된 형식으로 표시
    - 표시형식: d, o, x, t, f, s, i 등
    - 단위 크기: b, h, w, g

주요 GDB 명령어

- 프로그램 코드 확인
  - list [함수이름 / 라인넘버 / 변수이름]
    - 현재 / 지정된 위치의 프로그램 소스 코드 표시
  - disas <함수이름 / 메모리주소>
    - 현재 / 지정된 위치의 프로그램 어셈블리 코드 표시
- 변수 값 변경 감지
  - watch <변수이름>
    - 변수에 데이터가 쓰여질 때 실행 멈춤 (breakpoint 역할)
  - rwatch / awatch <변수이름>
    - 변수를 읽을 때 / 모든 경우에 실행 멈춤

주요 GDB 명령어

- 기타 명령어
  - backtrace, where
    - 현재 실행 위치의 주소와 스택 상태 표시
  - info locals
    - 모든 지역 변수 목록과 현재 값을 표시
  - info program
    - 프로그램 실행 정보 표시
  - info registers
    - 레지스터 사용 정보 표시