System Software Laboratory

KU KONKUK UNIVERSITY

System Programming

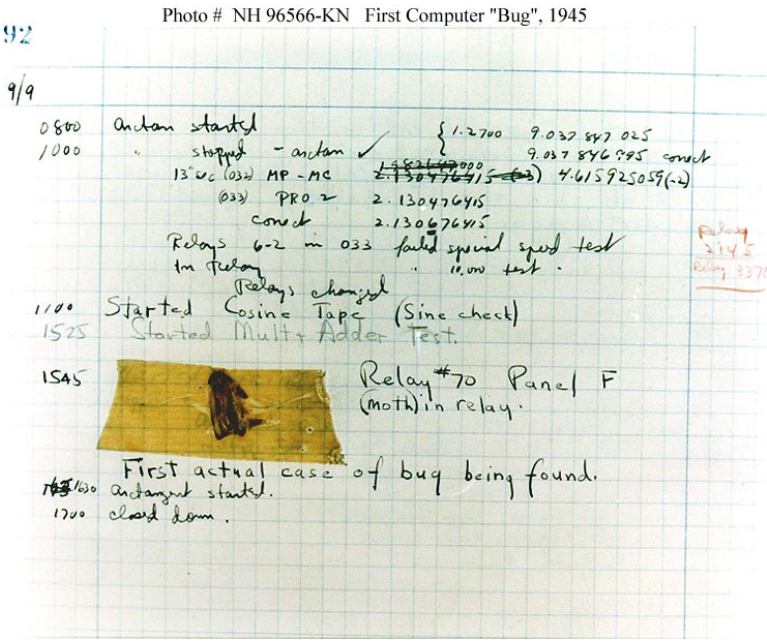
Hyun-Wook Jin
System Software Laboratory
Department of Computer Science & Engineering
Konkuk University
jinh@konkuk.ac.kr

강의 내용

- Reverse Engineering and GDB
- GDB 시작
- GDB 명령어
- GDB 실습 (예제)
- 과제 설명

버그와 디버그

- 버그의 유래
  - 하버드에서 만든 컴퓨터인 마크 II 하드웨어 발견된 나방
  - 나방이 끼어서 접촉 오류 발생
  - 이를 통해서 시스템 오류를 '버그'라 말하고, 이를 해결하는 과정을 '디버그'라고 부르게 됨



버그와 디버그

- 컴파일 오류
  - 잘못된 문법을 사용하여 컴파일러가 언어를 제대로 인식하지 못한 오류. 사전에 알 수 있음
- 런타임 오류
  - 프로그램이 실행 도중에 발생하는 오류이므로 사전에 알 수 없음
- 디버깅
  - 위와 같은 오류를 수정하는 과정

Reverse Engineering

- Reverse Engineering
  - 구조 분석을 통해서 시스템의 작동 원리를 발견하는 과정

- GDB (GNU Project Debugger)
  - 프로그램을 디버깅 하기 위한 도구
  - 실행 중인 프로그램 내부의 변수를 확인하거나, 변경할 수 있으며, 로직과 상관없이 함수 호출 가능
  - 실행을 추적하고 수정 가능
  - 어셈블리어 단위로 코드를 확인하고 추적할 수 있음

GDB 사용법

- C 코드 작성 (vi editor / nano / gedit 등 이용)
- 디버깅을 위한 컴파일 하기
  - $ gcc -Og [소스파일이름.c] –o [실행파일이름]

option	description
-g0	no debug information
-g1	minimal debug information
-g	default debug information
-g3	maximal debug information

- 디버깅 시작하기
  - $ gdb [실행파일이름]
- 디버깅 끝내기
  - (gdb) quit

GDB 사용법

- $ gdb [Program]

```
jkpark@ubuntu:~/SP1$ vim code1.c
jkpark@ubuntu:~/SP1$ gcc -o code -g code1.c
jkpark@ubuntu:~/SP1$ ls
code  code1.c
jkpark@ubuntu:~/SP1$ gdb ./code
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./code...done.
(gdb)
```

GDB 사용법

- list(l) : 소스 출력
  - list : main 함수를 기점으로 소스 출력
  - list 15 : 15번째 라인 주변 소스 출력(기본적으로 10줄 출력)
  - list func : func 함수의 소스 출력
  - list -5 : 현재 라인의 -5 번째 라인을 기준으로 출력
  - list test.c:func : test.c 파일의 func 함수부분을 출력
  - list test.c:10 : test.c 파일의 10번째 라인 주변의 소스 출력
  - set listsize 20 : 한번에 출력하는 라인 개수를 20개로 설정

```
(gdb) list
1       #include <stdio.h>
2
3       int main(void) {
4               printf("hi\n");
5               return 0;
6       }
(gdb)
```

GDB 사용법

- run(r), kill(k) : 프로그램 실행, 종료
  - run : 프로그램 수행(재시작)
  - run arg1 arg2 : arg1과 arg2를 인자로 프로그램 수행
  - kill : 프로그램 수행종료

```
(gdb) r
Starting program: /home/jkpark/SP1/code
hi
[Inferior 1 (process 5659) exited normally]
(gdb)
```

- backtrace(bt)
  - backtrace : 오류가 발생한 함수를 역으로 추적

GDB 사용법

- breakpoint(b)
  - b func : func 함수에 breakpoint 설정
  - b 10 : 10번째 라인에 breakpoint 설정
  - b +5 : 현재 라인에서 5 라인 이후에 breakpoint 설정
  - b *0x08048f29 : 0x08048f29 주소에 breakpoint 설정
  - b 10 if var == 0 : var의 값이 0일 때, 10번째 라인에
- 임시 breakpoint 설정
  - 해당 breakpoint가 걸리고 나면, breakpoint가 제거됨
  - tb [X]: temporary breakpoint, [X]는 위 breakpoint에 들어가는 것과 동일

GDB 사용법

- clear, delete
  - clear func : func 함수의 breakpoint 삭제
  - clear 10 : 10번째 라인의 breakpoint 삭제
  - clear tt.c:func : tt.c 파일 func함수의 breakpoint 삭제
  - delete 1 : 1번 breakpoint 삭제
  - delete : 모든 breakpoint 삭제
- enable, disable
  - enable 1 : 1번 breakpoint 활성화
  - disable 1 : 1번 breakpoint 비활성화

GDB 사용법

- step(s), next(n), continue(c), until, finish,return
  - n : 현재 라인 수행, 함수일 경우 함수를 수행함
  - s : 현재 라인 수행, 함수일 경우 함수 내부로 들어감
  - n(s) 5 : 5라인 수행 – c : 다음 breakpoint 까지 수행
  - until : loop를 빠져나와 다음 breakpoint까지 수행
  - finish : 함수를 수행하고 빠져나감
  - return : 함수를 수행하지 않고 빠져나감
  - return 0 : return 0으로 함수를 빠져나감
  - ni(si) : 현재 instruction을 수행

GDB 사용법

- print(p)
  - p var : var변수의 값 출력
  - p *var : pointer 변수의 값 출력(구조체의 값 확인)
  - p func : func 함수의 주소값 출력
  - p $rdi : %rdi register의 값 출력
- set
  - set {int}$rax=5 : %rax을 5로 설정
  - set {int}0x08048f35=5 : 0x08048f35주소의 값을 5 로 설정

GDB 사용법

- display
  - display var : var변수 값을 매번 표시
  - undisplay 1 : 1번째 display 설정을 삭제
  - disable display 1 : 1번째 display 설정을 비활성화
  - enable display 1 : 번째 display 설정을 활성화

- watch
  - 값이 변경될 때에만 출력
  - watch var : var변수가 써질 때 break
  - rwarch var : var변수가 읽혀질 때 break
  - awarch var : var변수가 읽기, 쓰기 모두 break

GDB 사용법

- x/포맷문자 : 메모리 값 확인
  - x/t addr : addr 주소의 값을 2진수로 출력
  - x/o addr : addr 주소의 값을 8진수로 출력
  - x/d addr : addr 주소의 값을 10진수(int)로 출력
  - x/u addr : addr 주소의 값을 10진수(unsigned)로 출력
  - x/x addr : addr 주소의 값을 16진수로 출력
  - x/c addr : addr 주소값의 1바이트를 문자형으로 출력
  - x/s addr : addr 주소의 값을 문자열로 출력
  - x/a addr : addr 가장 가까운 symbol의 offset 출력
  - x/i addr : 어셈블리 형식으로 출력

GDB 사용법

- information(info)
  - info reg : 모든 register 정보 출력
  - info reg $rax : %rax register의 정보 출력
  - info breakpoint(b) : 모든 breakpoint의 정보 출력
  - info breakpoint 1 : 1번째 breakpoint 정보 출력
  - info args : 프로그램이 시작될 대 주어진 인자값 출력
  - info func : 프로그램이 사용하는 함수들에 대한 출력
  - info frame : frame에 대한 정보 출력
  - info display : display로 설정된 정보 출력

GDB 사용법

- disassemble(disas)
  - disas func : func 함수의 assembly code 출력
  - disas 0x08048f29 0x08048f35 : 특정 주소 사이의assembly code 출력
- call
  - call func(arg1, arg2) : func함수 호출, return값 출력
- jump
  - jump *0x08048f35 : 해당 주소로 jump
  - jump 10 : 10번째 라인으로 jump
  - jump func : func함수로 jump

GDB 예상순서

- 코드작성
- 디버깅을 위한 컴파일
- gdb [prog] – GDB실행
- list – 코드를 확인
- b , info b , delete – breakpoint 설정
- run – 디버깅환경에서 실행
- step, next, continue – 진행
- print, display - 변수확인
- quit - 디버깅종료

디버깅 실습 #1

```c
#include <stdio.h>

static void do_swap( int *lhs, int *rhs, int *tmp )
{
	tmp = 0;

	*tmp = *lhs;
	*lhs = *rhs;
	*rhs = *tmp;
}

void swap ( int *lhs, int *rhs )
{
	int t;
	do_swap( lhs, rhs, &t );
}
```

```c
int main ( void )
{
	int i = 0;
	int j = 1;

	printf( "i = %d, j = %d\n", i, j );
	swap( &i, &j );

	printf( "i = %d, j = %d\n", i, j );
	return 0;
}
```

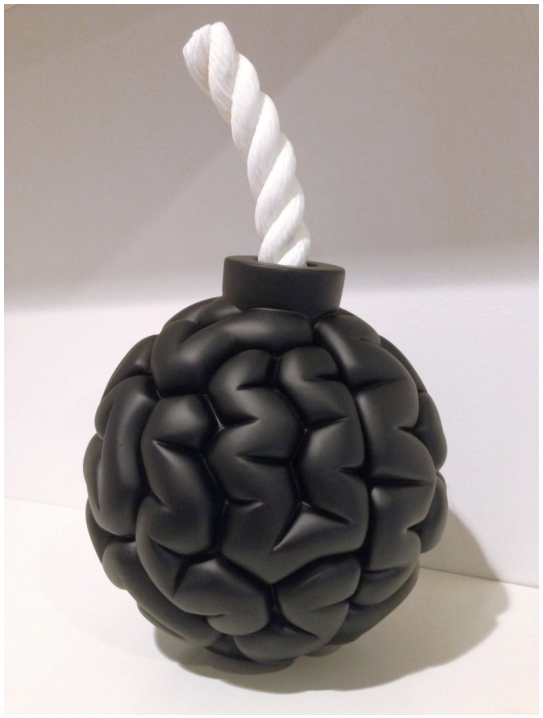
디버깅 실습 #2

```c
#include <stdio.h>

void swap(int x, int y)
{
	int temp;
	temp=x;
	x=y;
	y=temp;
}
```

```c
int main()
{
	int a=10;
	int b=30;

	printf("Before swap: a=%d, b=%d\n", a, b);

	swap(a,b);

	printf("After swap: a=%d, b=%d\n", a, b);

	return 0;
}
```

BombLab

- BombLab
  - 본 과제의 목적은 binary code를 reverse engineering하고 주어진 binary code의 수행 내용을 이해
  - 본 과제는 Dr. Evil이 만든 폭탄을 제거하는 시나리오를 가짐
  - 폭탄을 실행시키면 각 단계마다 암호를 입력 해야 하고, 정확한 암호를 입력하면 폭탄이 제거되고 다음 단계로 넘어감
  - 잘못된 암호를 입력할 경우 폭탄은 "BOMB!!!"이라는 메시지와 함께 터지고 종료됨



BombLab 준비

- 본인에게 학번으로 할당된 폭탄을 다운로드 받는다.
  - 다운로드 페이지 : https://drive.google.com/open?id=1uXQlWlbjstiLqUBUflU4QtWGxAhmnuTN
  - 다운로드 기간 : 2017.11.09 ~ 2017.11.12
    - 다운로드 기간이 지나면 과제를 다운받을 수 없음
  - 반드시 본인의 폭탄을 다운로드 받을 것
    - 타인의 폭탄을 받아 과제 진행 시 0점
- 압축해제 : tar xvf bomb#.tar
- 압축 해제된 파일은 다음과 같이 구성된다.
  - bomb : 실행 가능한 binary 폭탄
  - bomb.c : bomb의 메인 루틴 소스파일
  - solution#.txt : bomb 과제 폭탄 암호를 적는 텍스트 파일
    - #는 자신의 고유번호 (파일 이름 변경 시 과제 0점)

BombLab 풀이 도구

- 다음 tool들의 기능을 사용하여 폭탄을 제거할 수 있다.
  - gdb : breakpoint, disassemble, etc.
  - objdump
    - -t : symbol table출력. 함수, 전역변수 등의 이름과 주소를 알 수 있음
    - -d : assembly code 생성
  - strings [Program]: 출력 가능한 문자열들을 출력

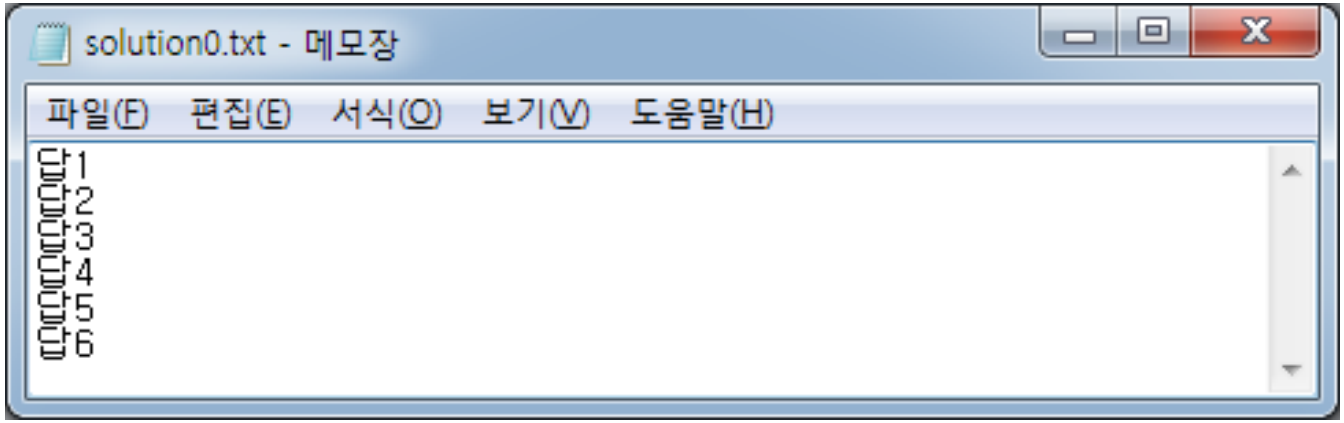
과제 TIP

- Bomb는 여섯개의 단계 + 추가문제로 구성되어 있다.
  - 1단계: 문자열 비교
  - 2단계: 배열
  - 3단계: 조건문
  - 4단계: 함수 콜
  - 5단계: 배열
  - 6단계: 비공개
  - 추가문제: 비공개
- 각 단계의 형태는 어셈블리 코드를 이해하는데 힌트가 될 수 있음

과제 TIP

- 각각의 학생마다 폭탄의 답은 다름

- 폭탄의 암호를 solution#.txt에 저장하여 폭탄 실행 시 "./bomb solution#.txt" 로 수행하면 파일에 입력된 각 line 의 암호가 입력됨.



주의사항

- 다른 사람의 과제를 copy할 경우 0점!!
- 제출 양식 반드시 엄수!! 어길 시 미제출로 처리
- 지연 제출시 미제출로 처리
- 질의응답은 수업게시판을 이용

제출 방법

- 제출방법
  - E-Mail 제출 : 2017sslab@gmail.com
  - 제목 : [2017_시스템프로그래밍#2_A]학번_이름
    - A는 화요일 오전 반(과목코드 2195)
    - B는 화요일 오후 반(과목코드 2196)
  - 제출파일
    - solution#.txt : 폭탄 암호 파일
    - 학번_이름_report.pdf : 과제 수행 방법 및 과정을 report로 작성
      - Ex) 201612345_홍길동_report.pdf
- 제출기한 : 11월 19일 일요일 23시 59분까지