리버싱 교육 1회차

201702075 조수환

ARGOS

INDEX

ARGOS

리버싱 교육 1회차

교육 소개

- 교육자/튜터 소개
- 리버싱?

ARGOS

교육자/튜터 소개

교육부장 - 전반적인 교육 진행 담당

- 17 조수환

튜터 - 실습 진행 중 막히는게 있으면 질문해주세요

- 18 박준서
- 17 박정필
- 18 서연주
- 17 김재훈

리버싱?

Reverse Engineering

완성된 제품을 분석하여 설계 단계로 돌리는 기술

(설계 -> 개발 -> 제품화의 역과정)

- 교육 목표
  - 바이너리(실행파일), 어셈블리어와 친해지기
  - 기본적인 컴퓨터 시스템에 대한 이해도 올리기
  - ~~쉬운 과제 점수 한입~~ ★★★

리버싱 교육 1회차

어셈블리어 기초

- 레지스터
- 메모리 구조
- 연산자

ARGOS

레지스터

Register

프로세서가 연산을 하기 위해 필요한 저장공간

CPU에 내장되어 CPU가 요청을 처리하는데에 있어서

필요한 데이터를 일시적으로 저장하는 역할을 수행

밤랩 실습하는 내내 볼겁니다

레지스터

ARGOS

범용 레지스터

용도가 특별하게 정해져 있지 않은 레지스터로 변수와 같은 역할을 한다.

보편적인 규약, 관습에 의거해 쓰임새가 정해져 있는 경우도 있다.

ex) rax = 함수 리턴 값, rsi = 함수 인자 값

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

레지스터

함수 호출 인자

함수가 호출될 때 필요한 인자들을 담는 역할

인자의 순서대로 rdi, rsi, rcx, rdx...

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

레지스터

스택 포인터

스택의 가장 위쪽을 가리킨다.

스택 메모리는 함수에 필요한 변수들이 저장됩니다.

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

레지스터

프로그램 카운터

rip는 범용 레지스터가 아닌 프로그램 카운터의 역할만 한다.

프로그램 카운터는 다음에 실행될 명령어의 위치를 가리킨다.

rax rbx rcx rdx rsi rdi

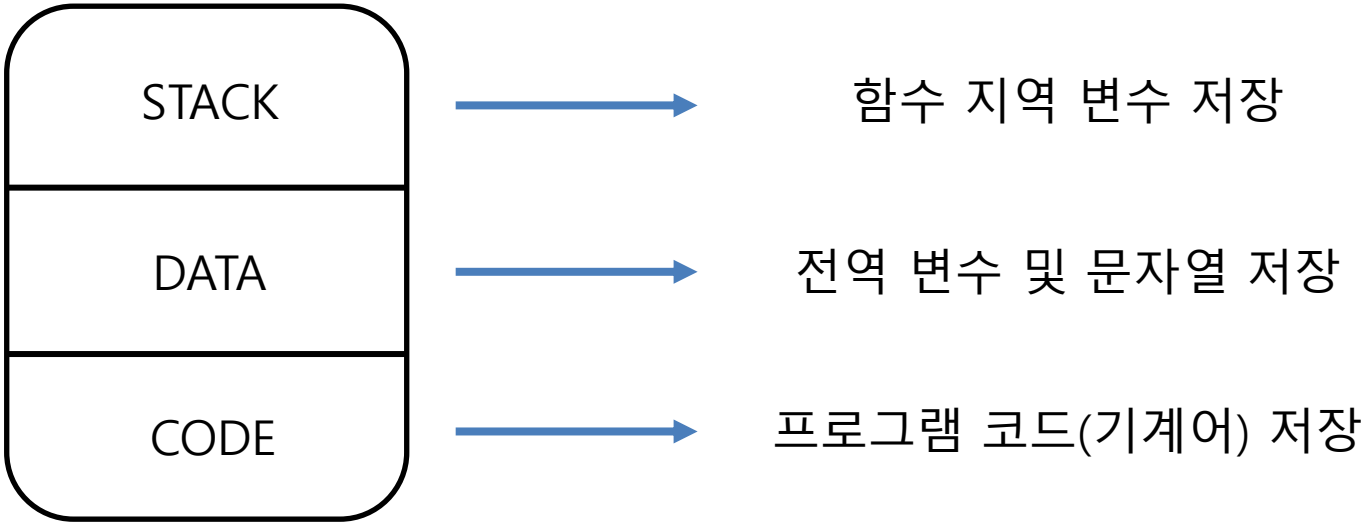
rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

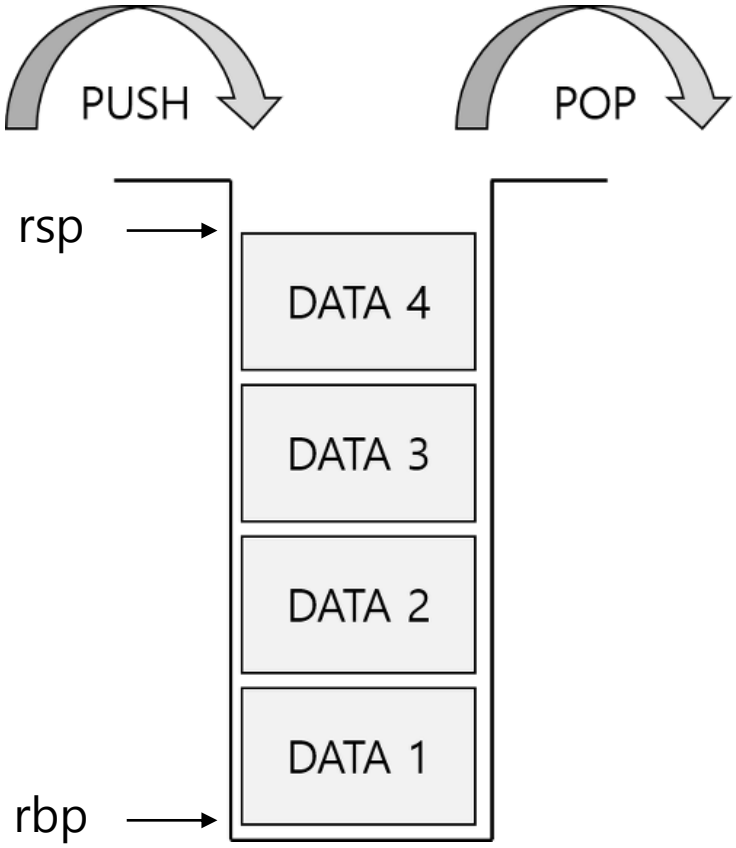
rip

메모리 구조

메모리는 많은 영역으로 나뉘고, 각자 분담하는 역할이 다르다.



메모리 구조



스택(Stack)은 데이터를 쌓는 형태의

자료구조를 말한다.

스택 메모리는 이러한 구조를 갖는

저장 공간을 말한다.

스택을 용이하게 다루기 위해서는

꼭대기의 위치를 알아야 한다.(스택 포인터)

연산자

어셈블리어 형태 = [연산자] [피연산자]

Ex) add rax, rbx (연산자 = add, 피연산자 = rax, rbx)

- 자주 보는 연산자

  - mov a, b -> b를 a에 복사한다. (a = b)

  - lea a, b -> b의 주소에 있는 값을 a에 복사한다. (a = *b)

연산자

- 자주 보는 연산자

  - add a, b -> a + b의 값을 a에 넣는다. (a += b)

  - sub a, b -> a – b의 값을 a에 넣는다. (a -= b)

  - imul a, b -> a * b의 값을 a에 넣는다. (a *= b)

  - cmp a, b -> a와 b를 비교한다.

  - jmp [위치] -> 해당 위치로 이동(jump)한다.

  - jl, jg, je -> jmp의 비교 조건 (l = less than, g = greater than, e = equal)

리버싱 교육 1회차

Bomb Lab

- Bomb Lab이란?
- gdb 명령어
- Phase 1 실습

ARGOS

Bomb Lab이란?

Bomb Lab

총 6개의 phase로 구성된 bomb 바이너리를 분석하여

폭탄을 터트리지 않고 phase를 모두 통과하는 과제

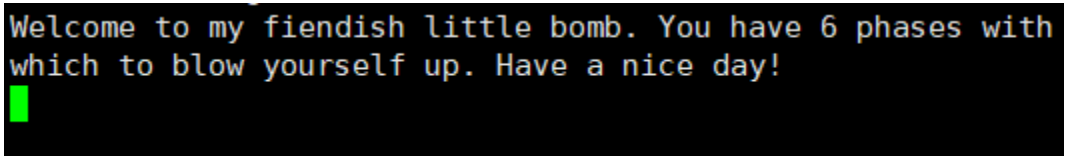
어렵고 딱딱한 어셈블리어와 리버싱을

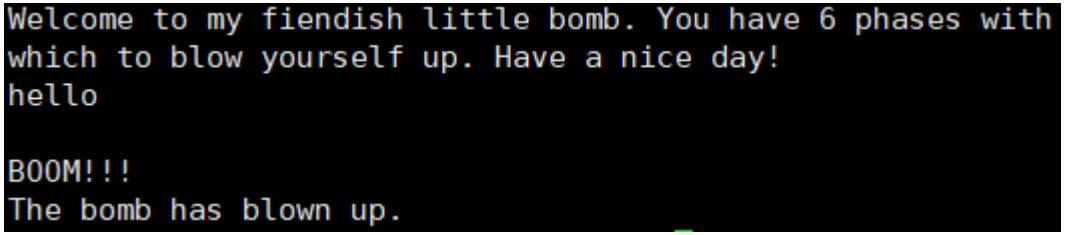
(그나마)재미있게 실습을 해볼 수 있다.

Bomb Lab이란?

각 phase 마다 입력해야하는 답이 있고, 틀리면 폭탄이 터지는 구조.

(시프 과제에서는 터질 때 마다 점수가 깎입니다.)





지금부터 바이너리를 분석하여 답을 알아내 봅시다.

Bomb Lab이란?

git clone https://github.com/4RG0S/2021-argos-bomblab-edu.git

```
sh2358@edu-argos:~$ git clone https://github.com/4RG0S/2021-argos-bomblab-edu.git
Cloning into '2021-argos-bomblab-edu'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), 13.55 KiB | 2.26 MiB/s, done.
```

cd 2021-argos-bomblab-edu ; chmod +x bomb

(디렉토리 이동 후 bomb 바이너리의 권한 변경)

준비 끗!

gdb 명령어

ARGOS

GDB(GNU Debugger)

```
sh2358@edu-argos:~/2021-argos-bomblab-edu$ gdb bomb
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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 bomb...
(gdb)
```

C, C++ 등으로 만들어진 실행 파일을 디버깅 하는 도구

bomb 바이너리를 분석하는데 쓸 겁니다.

gdb 명령어

자주 쓰는 명령어 모음

시작/종료
- 시작 : gdb [프로그램명]
- 종료 : quit or q

intel 문법 설정
- set disassembly-flavor intel

분석
- 해당 함수 코드 : disas [함수명]
- 실행 : run or r

중요!
- 브레이크 포인트 : b [위치]

- 브레이크 포인트 위치의 코드 : disas
- 다음 명령어 : ni
- 진행 : c
- 강제 이동 : jump [위치]
- info func : 사용한 함수 보기
- info r : 사용한 레지스터 보기

정보 수집 -x 명령어
x/[범위][출력 형식]

<출력 형식>
t : 2 진수
o : 8 진수
d : 10 진수
x : 16 진수
s : 문자열
i : 어셈블리

정보 수집 –p 명령어
p/[출력 형식][계산식] : 계산 결과 확인
계산식에는 여러가지가 들어 갈 수 있다.
(ex : 레지스터, 변수)

gdb 명령어

ARGOS

* break point의 중요성 *

틀리면 폭탄이 터진다 -> 폭탄이 터지면 감점이다.

= 틀려도 폭탄만 안 터지면 장땡이다!?

```
(gdb) r
Starting program: /home/sh2358/2021-argos-bomblab-edu/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
hello

Breakpoint 1, 0x0000000000400ee0 in phase_1 ()
(gdb) c
Continuing.

Breakpoint 7, 0x000000000040143a in explode_bomb ()
```

```
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 2098162) killed]
```

gdb 명령어

ARGOS

- break point 설정 위치
  - 각 phase 마다 설정해놓기
  - explode bomb 함수에 설정해놓기

```
(gdb) b* phase_1
Breakpoint 1 at 0x400ee0
(gdb) b* phase_2
Breakpoint 2 at 0x400efc
(gdb) b* phase_3
Breakpoint 3 at 0x400f43
(gdb) b* phase_4
Breakpoint 4 at 0x40100c
(gdb) b* phase_5
Breakpoint 5 at 0x401062
(gdb) b* phase_6
Breakpoint 6 at 0x4010f4
(gdb) b* explode_bomb
Breakpoint 7 at 0x40143a
```

Phase 1 실습

ARGOS

disas phase_1

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    rsp,0x8
   0x0000000000400ee4 <+4>:     mov    esi,0x402400
   0x0000000000400ee9 <+9>:     call   0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   eax,eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    call   0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    rsp,0x8
   0x0000000000400efb <+27>:    ret
End of assembler dump.
```

test a, b = 두 피연산자의 and 연산 진행 (값이 0이면 ZF = 1)

je = jump if equal, 값이 같거나 ZF = 1이면 jump

즉 eax의 값이 0이면 통과

Phase 1 실습

ARGOS

eax는 함수 strings_not_equal의 리턴값이다.

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    rsp,0x8
   0x0000000000400ee4 <+4>:     mov    esi,0x402400
   0x0000000000400ee9 <+9>:     call   0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   eax,eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    call   0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    rsp,0x8
   0x0000000000400efb <+27>:    ret
End of assembler dump.
```

그렇다면 위의 함수도 뜯어봐야 하나 싶지만 그럴 필요는 없다.

의도한건지 그냥 친절한건지 함수명이 아주 직관적이기 때문

문자열이 같은지 아닌지를 판별하는 함수 같다.

Phase 1 실습

함수를 호출하려면 레지스터에 인자를 담아서 보내야 한다.

```
0x0000000000400ee4 <+4>:     mov    esi,0x402400
0x0000000000400ee9 <+9>:     call   0x401338 <strings_not_equal>
```

esi는 함수의 인자를 담는 레지스터다. 이 값을 한번 보자.

```
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
```

esi는 함수의 두번째 인자를 담는다.

문자열을 비교하는 함수니까 첫번째 인자인 edi와 esi를 비교하여

리턴 값을 결정하지 않을까?

Phase 1 실습

ARGOS

프로그램을 실행해서 phase_1 test라는 문자열을 입력했다.

```
(gdb) r
Starting program: /home/sh2358/2021-argos-bomblab-edu/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
phase_1 test

Breakpoint 1, 0x0000000000400ee0 in phase_1 ()
```

여기서 disas 명령어를 입력하면 break point 위치의 어셈블리 코드가 나온다.

```
(gdb) disas
Dump of assembler code for function phase_1:
=> 0x0000000000400ee0 <+0>:     sub    rsp,0x8
   0x0000000000400ee4 <+4>:     mov    esi,0x402400
   0x0000000000400ee9 <+9>:     call   0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   eax,eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    call   0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    rsp,0x8
   0x0000000000400efb <+27>:    ret
End of assembler dump.
```

Phase 1 실습

ni 명령어를 입력하면 다음 줄을 실행한다.

(입력 후 다시 enter 누르면 한번 더 함)

```
(gdb) ni
0x0000000000400ee4 in phase_1 ()
(gdb)
0x0000000000400ee9 in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    rsp,0x8
   0x0000000000400ee4 <+4>:     mov    esi,0x402400
=> 0x0000000000400ee9 <+9>:     call   0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   eax,eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    call   0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    rsp,0x8
   0x0000000000400efb <+27>:    ret
```

strings_not_equal 함수를 실행하기 바로 전이다.

여기서 함수의 인자인 edi의 값을 보자

Phase 1 실습

ARGOS

i r 명령어를 입력하면 레지스터의 정보들을 볼 수 있다.

```
End of assembler dump.
(gdb) i r
rax            0x603780         6305664
rbx            0x402210         4203024
rcx            0xc              12
rdx            0x1              1
rsi            0x402400         4203520
rdi            0x603780         6305664
```

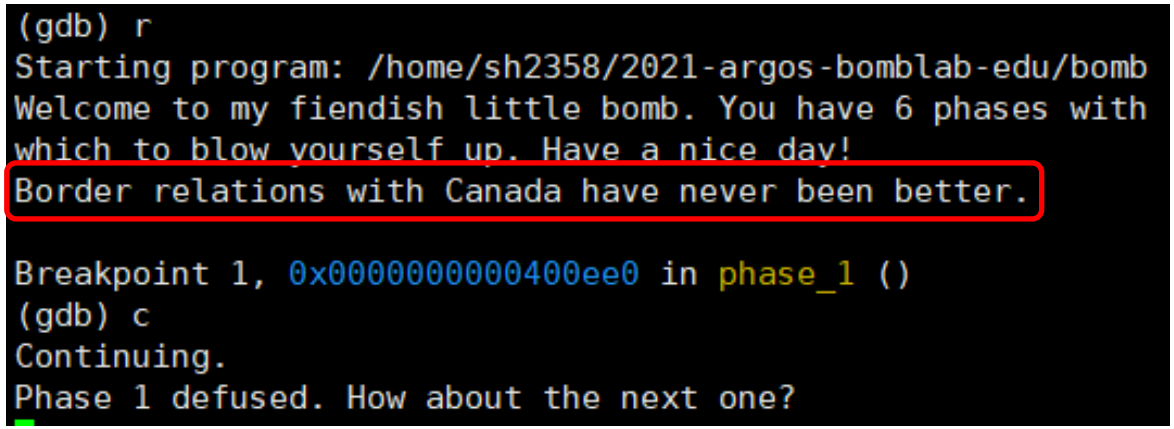
위의 정보를 통해 rdi의 값을 뜯어보자.

```
(gdb) x/s 0x603780
0x603780 <input_strings>:       "phase_1 test"
```

아까 입력한 문자열이 들어 있다.

즉, esi에 있는 문자열과 같은 문자열을 입력하면 되는 것이다.

Phase 1 실습



이런 식으로 explode bomb의 호출 없이 phase가 통과된다.

이후의 다른 phase들 또한 이번 실습처럼

분석하고 입력해보고 뜯어보면서 답을 찾아 입력하는 방식으로 해결한다.

리버싱 교육 1회차

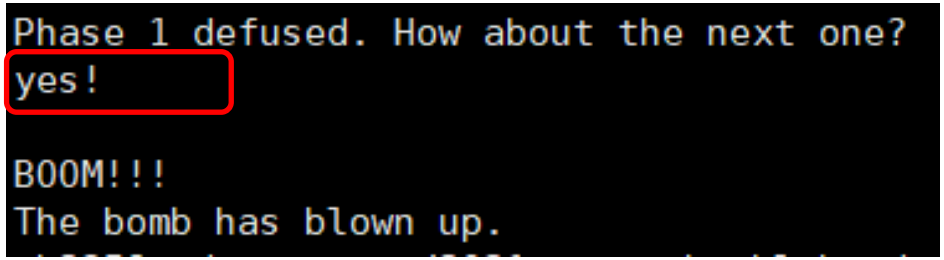
과제

- phase 2

ARGOS

phase 2

phase 1을 통과하면 바로 phase 2를 시작합니다.



phase 1 처럼 phase_2 함수를 분석해서

올바른 입력을 하면 통과 할 수 있습니다.

다시 시작 할때마다 phase_1을 통과해서 와야 하므로

각 phase의 정답을 메모장 같은 곳에 적어놓으세요.

Q & A

Thank You for Listening