시스템프로그래밍(가) 과제2

소프트웨어학부 20192800 권대현

1. 개발 환경

* 운영체제: Windows 11 Home
* 하위 시스템: GNU/Linux 5.15.146.1-microsoft-standard-WSL2 x86\_64
* 리눅스 버전: Ubuntu 22.04.2 LTS

1. 소스코드 설명

* 20192800.h

#include <stdio.h>

#include <assert.h>

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <errno.h>

#include <stdbool.h>

#include <elf.h>

bool is\_elf(Elf64\_Ehdr eh);

void read\_elf\_header(int32\_t fd, Elf64\_Ehdr \*elf\_header);

void print\_elf\_header(Elf64\_Ehdr elf\_header);

char \*read\_section(int32\_t fd, Elf64\_Shdr sh);

void print\_section\_headers(int32\_t fd, Elf64\_Ehdr eh, Elf64\_Shdr sh\_table[]);

void modify\_rodata(int32\_t fd, Elf64\_Shdr sh\_table);

//Implement this function to check whether an input file is an elf file or not.

bool is\_elf(Elf64\_Ehdr elf\_header)

{

    if(!strncmp((char\*)elf\_header.e\_ident, "\177ELF", 4)){

        return true;

    }else{

        printf("It's not ELF file\n");

        return false;

    }

}

void read\_elf\_header(int32\_t fd, Elf64\_Ehdr \*elf\_header)

{

    assert(elf\_header != NULL);

    assert(lseek(fd, (off\_t)0, SEEK\_SET) == (off\_t)0);

    assert(read(fd, (void \*)elf\_header, sizeof(Elf64\_Ehdr)) == sizeof(Elf64\_Ehdr));

}

void print\_elf\_header(Elf64\_Ehdr elf\_header)

{

    printf("Storage class\t= ");

    switch(elf\_header.e\_ident[EI\_CLASS])

    {

        case ELFCLASS32:

            printf("32-bit objects\n");

            break;

        case ELFCLASS64:

            printf("64-bit objects\n");

            break;

        default:

            printf("Unknwon CLASS\n");

            break;

    }

    printf("Data format\t= ");

    switch(elf\_header.e\_ident[EI\_DATA])

    {

        case ELFDATA2LSB:

            printf("2's complement, little endian\n");

            break;

        case ELFDATA2MSB:

            printf("2's complement, big endian\n");

            break;

        default:

            printf("Unknwon Format\n");

            break;

    }

    printf("OS ABI\t\t= ");

    switch(elf\_header.e\_ident[EI\_OSABI])

    {

        case ELFOSABI\_SYSV:

            printf("UNIX System V ABI\n");

            break;

        case ELFOSABI\_HPUX:

            printf("HP-UX\n");

            break;

        case ELFOSABI\_NETBSD:

            printf("NetBSD\n");

            break;

        case ELFOSABI\_LINUX:

            printf("Linux\n");

            break;

        case ELFOSABI\_SOLARIS:

            printf("Sun Solaris\n");

            break;

        case ELFOSABI\_AIX:

            printf("IBM AIX\n");

            break;

        case ELFOSABI\_IRIX:

            printf("SGI Irix\n");

            break;

        case ELFOSABI\_FREEBSD:

            printf("FreeBSD\n");

            break;

        case ELFOSABI\_TRU64:

            printf("Compaq TRU64 UNIX\n");

            break;

        case ELFOSABI\_MODESTO:

            printf("Novell Modesto\n");

            break;

        case ELFOSABI\_OPENBSD:

            printf("OpenBSD\n");

            break;

        case ELFOSABI\_ARM\_AEABI:

            printf("ARM EABI\n");

            break;

        case ELFOSABI\_ARM:

            printf("ARM\n");

            break;

        case ELFOSABI\_STANDALONE:

            printf("Standalone (embedded) app\n");

            break;

        default:

            printf("Unknown (0x%x)\n", elf\_header.e\_ident[EI\_OSABI]);

            break;

    }

    printf("Filetype \t= ");

    switch(elf\_header.e\_type)

    {

        case ET\_NONE:

            printf("N/A (0x0)\n");

            break;

        case ET\_REL:

            printf("Relocatable\n");

            break;

        case ET\_EXEC:

            printf("Executable\n");

            break;

        case ET\_DYN:

            printf("Shared Object\n");

            break;

        default:

            printf("Unknown (0x%x)\n", elf\_header.e\_type);

            break;

    }

    printf("Machine\t\t= ");

    switch(elf\_header.e\_machine)

    {

        case EM\_NONE:

            printf("None (0x0)\n");

            break;

        case EM\_386:

            printf("INTEL x86 (0x%x)\n", EM\_386);

            break;

        case EM\_X86\_64:

            printf("AMD x86\_64 (0x%x)\n", EM\_X86\_64);

            break;

        case EM\_AARCH64:

            printf("AARCH64 (0x%x)\n", EM\_AARCH64);

            break;

        default:

            printf(" 0x%x\n", elf\_header.e\_machine);

            break;

    }

    printf("\n");

}

char \*read\_section(int32\_t fd, Elf64\_Shdr sh)

{

    char\* buff = malloc(sh.sh\_size);

    if(!buff) {

        printf("%s:Failed to allocate %ldbytes\n",

                \_\_func\_\_, sh.sh\_size);

    }

    assert(buff != NULL);

    assert(lseek(fd, (off\_t)sh.sh\_offset, SEEK\_SET) == (off\_t)sh.sh\_offset);

    assert(read(fd, (void \*)buff, sh.sh\_size) == sh.sh\_size);

    return buff;

}

void print\_section\_headers(int32\_t fd, Elf64\_Ehdr eh, Elf64\_Shdr sh\_table[])

{

    uint32\_t i;

    char\* sh\_str;

    assert(lseek(fd, (off\_t)eh.e\_shoff, SEEK\_SET) == (off\_t)eh.e\_shoff);

    for(i=0; i<eh.e\_shnum; i++) {

        assert(read(fd, (void \*)&sh\_table[i], eh.e\_shentsize) == eh.e\_shentsize);

    }

    /\* section-header string-table \*/

    sh\_str = read\_section(fd, sh\_table[eh.e\_shstrndx]);

    for(i=0; i<eh.e\_shnum; i++) {

        if(!strncmp((sh\_str + sh\_table[i].sh\_name), ".rodata", 7))

        {

            printf("%s section info\n", (sh\_str + sh\_table[i].sh\_name));

            printf("    file offset = 0x%08lx\n", sh\_table[i].sh\_offset);

            printf("           size = 0x%08lx\n", sh\_table[i].sh\_size);

            //rodata 수정하는 함수 호출

            modify\_rodata(fd, sh\_table[i]);

        }

    }

}

//rodata 수정하는 함수

void modify\_rodata(int32\_t fd, Elf64\_Shdr sh\_table)

{

    char \*rodata = read\_section(fd, sh\_table);

    char \*ptr = rodata;

    //rodata 영역 크기만큼 처음부터 끝까지 탐색

    while(ptr < rodata + sh\_table.sh\_size)

    {

        //만약 문자열이 software라면

        if (!strncmp(ptr, "software", 8))

        {

            strncpy(ptr, "hackers!", 8);

            break;

        }

        ptr++;

    }

    //파일 커서를 검색한 rodata 위치로 이동

    assert(lseek(fd, (off\_t)sh\_table.sh\_offset, SEEK\_SET) == (off\_t)sh\_table.sh\_offset);

    //저장된 ptr을 파일 커서에 해당하는 위치에 저장

    assert(write(fd, rodata, sh\_table.sh\_size) == sh\_table.sh\_size);

}

* + 20192800.h에는 20192800.c에서 사용하는 함수의 선언과 원형이 작성돼 있다.
  + 기존 readelf.h 파일에서 modify\_rodata 함수를 추가로 선언 및 구현했다.
  + 함수 설명
    - is\_elf: 해당 파일이 elf 포맷인지 검사하는 함수이다.
    - read\_elf\_header: 파일 디스크립터를 통해 해당 파일의 elf 헤더를 읽어들여서 elf\_header 인자로 넘겨주는 함수이다.
    - print\_elf\_header: 인자로 받은 elf\_header를 통해 헤더의 정보를 printf로 출력하는 함수이다.
    - read\_section: 인자로 받은 섹션의 영역을 읽어서 char 배열 버퍼에 저장하여 리턴해주는 함수이다.
    - print\_section\_headers: 섹션을 읽어들여서 rodata 영역이 나오면 해당 섹션의 이름, offset, 크기를 출력하는 함수이다. 여기에 추가적으로 rodata를 수정하기 위한 modify\_rodata 함수를 호출해줬다.
    - modify\_rodata: 파일 디스크립터와 sh\_table(rodata)을 인자로 넘겨받아서, rodata에 “software” 문자열이 존재하는지 검사하고, 이를 “hackers!”로 수정하는 함수이다.
  + modify\_rodata() 상세 설명
    - 해당 함수는 인자로 파일 디스크립터 번호와 헤더 테이블을 넘겨받는다.
      * modify\_rodata()함수는 print\_section\_headers 함수에서 rodata 영역일 때 호출되므로, 항상 헤더 테이블로 rodata영역을 넘겨받는다.
    - rodata의 data를 저장할 char 배열 변수 rodata를 선언하여 read\_section 함수를 호출하여 buffer를 넘겨받는다.
    - 문자열 검사를 위한 포인터 \*ptr을 선언하고, rodata의 첫 번째 인덱스를 가리키게 한다.
    - While문 안에서 rodata의 처음부터 끝까지 검사한다.
      * Strncmp 함수를 통해 ptr이 “software” 문자열과 같은지를 검사한다.
        + 만약 같다면, ptr에 “hackers!”를 복사하고 while문을 빠져나온다.
      * ptr값을 1 증가시켜서 이전 ptr에서 다음 문자를 가리키는 문자열로 변경시킨다.
    - lseek 함수를 통해 write를 수행하기 전, 파일 커서를 rodata 섹션의 offset으로 이동시킨다.
    - Write을 수행하여 변경된 buffer인 rodata를 rodata 섹션에 덮어씌운다.
* 20192800.c

#include <20192800.h>

int32\_t main(int32\_t argc, char \*argv[])

{

    int32\_t fd;

    if(argc!=2) {

        printf("Usage: 20192800 <file>\n");

        return 0;

    }

    fd = open(argv[1], O\_RDWR|O\_SYNC);

    if(fd<0) {

        printf("Error %d Unable to open %s\n", fd, argv[1]);

        return 0;

    }

    Elf64\_Ehdr ehdr;

    Elf64\_Shdr\* sh\_tbl;

    read\_elf\_header(fd, &ehdr);

    if(!is\_elf(ehdr)) {

        return 0;

    }

    print\_elf\_header(ehdr);

    sh\_tbl = malloc(ehdr.e\_shentsize \* ehdr.e\_shnum);

    if(!sh\_tbl) {

        printf("Failed to allocate %d bytes\n", (ehdr.e\_shentsize \* ehdr.e\_shnum));

    }

    print\_section\_headers(fd, ehdr, sh\_tbl);

    return 0;

}

* + 20192800.c는 기존 파일인 readelf.c에서 참조할 헤더파일 명을 20192800.h로 변경한 것 외엔 변경 사항이 없다.
  + 먼저 argc의 개수를 검사하여 프로그램에 올바른 인자 개수가 들어왔는지 판단한다.
  + 두 번째 인자, 즉 rodata를 변경할 대상의 파일을 open하여 파일 디스크립터 번호를 저장할 fd에 넣어 준다.
  + read\_elf\_header를 수행하여 fd를 통해 elf 헤더 정보를 ehdr 변수에 넣어준다.
  + is\_elf 함수를 통해 ehdr에 등록된 파일이 elf 포맷인지 확인한다.
  + print\_elf\_header를 통해 ehdr의 헤더 정보를 출력한다.
  + 헤더 테이블을 저장할 변수 sh\_tbl의 크기를 동적 할당해준다.
  + print\_section\_headers 함수를 통해 섹션의 헤더 정보를 출력함과 동시에 rodata 영역 내 “software” 문자열을 검사하여 “hackers!”로 바꿔준다.
* Makefile

CC=gcc

CFLAGS=-I.

20192800: 20192800.c

    $(CC) -o 20192800 20192800.c $(CFLAGS)

* + Makefile 파일에는 소스코드를 컴파일하기 위한 정보와 명령어가 저장되어 있다.
  + 먼저 컴파일러의 종류를 담을 CC 변수를 선언하여 GCC를 사용하도록 지정했다.
  + CFLAGS 변수를 선언하여 -I. 옵션을 사용하도록 했다. 이 옵션은 컴파일할 때 헤더 파일을 검색할 위치를 현재 폴더에서 찾도록 한다.
  + Makefile을 통해 만들어진 실행 파일의 이름을 20192800로 설정하고 20192800.c 소스 코드를 사용하도록 명시했다.
  + 실제 빌드할 때의 명령어를 선언했던 변수를 이용하여 명시했다. gcc -o 20192800 20192800.c -I. 을 최종적으로 수행할 것이다.

1. 문제점 및 해결 방법

* rodata 영역에 저장된 데이터 검출 방법에 대한 문제
  + readelf.h의 print\_section\_headers 함수를 통해 rodata의 offset과 size를 알아내는 방법에 대해서 이해할 수 있었다. 그러나, 어떻게 해당 주소에 접근해 데이터를 검출해야 할 지 감이 오지 않았다.
  + 애초에 print\_section\_headers 함수가 .rodata임을 알아내는 방법이 문자열 비교 함수인 strncmp를 통해 확인하는 것을 보고, read\_section 함수를 통해 섹션의 데이터를 가져온다는 사실을 알게 되었다. 이를 통해 임시로 데이터를 저장할 char 형 배열 변수를 선언하여 read\_section에서 불러온 buffer를 저장해 주어서 문제를 해결할 수 있었다.
* Buffer 문자열 검사에 대한 문제
  + 위의 해결 과정을 통해 buffer를 rodata에 저장하는 것까지는 해결했다. 그러나, 어떻게 문자열에서 “software”를 검사해낼 지 방법이 떠오르지 않았다.
  + 보통 문자열은 for문이나 while문 같은 반복문과 검사를 진행할 index를 통해 검사한다는 사실을 떠올렸다. 나는 while문을 통해 검사를 진행 해야겠다고 생각했고, 이를 위한 포인터 \*ptr을 선언해서 rodata의 첫 부분을 가리키게 했다. rodata의 끝 부분은 첫 부분에서 section 크기만큼 더한 값일 테니, rodata + sh\_table.sh\_size로 설정하였다. 마지막으로, 어떻게 문자열이 검출되는 지를 확인하기 위해 printf를 돌려봤고, 다음 이미지와 같이 문자열이 출력되는 것을 확인했다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* 변경 사항 저장에 대한 문제
  + 바로 위 해결 방안을 통해 software를 검출하여 hackers!로 변경하는 과정까진 구현할 수 있었다. 그러나, 단순히 함수 내 buffer에서 변경하는 것으론 rodata의 영역에 변경 사항을 갱신할 수 없었다.
  + 이를 해결하기 위해 다시 readelf.h를 참고했다. readelf.h 내 함수 중에서 read\_section이라는 함수의 마지막 부분에 파일 디스크립터를 통해 파일에 접근하고 이를 read를 통해 불러오는 코드가 존재한다. 여기서 영감을 받아 read를 단순히 write으로 바꿔준다면 문제가 해결될 것 같다고 생각했다. 과정은 다음과 같다. 먼저 lseek 함수를 통해 파일에서 write을 수행할 offset을 설정해준다. modify\_rodata의 경우, sh\_table.sh\_offset이 offset에 해당할 것이다. 다음으론 write함수를 통해 변경된 파일 커서를 기준으로 write 작업을 수행한다. 변경된 buffer인 rodata와 sh\_table.sh\_size가 크기에 해당한다. 다행히 해결 방법이 도움이 됐고, 다음과 같은 결과를 얻을 수 있었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명