Stack Overflow Exploitation using Bash and Mitigation

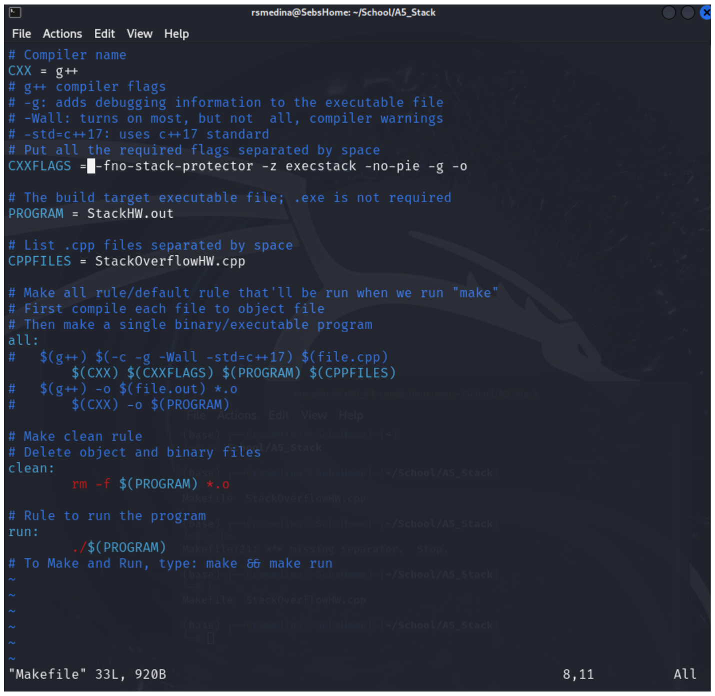
By Sebastian Medina

Content:

1. Disable all the counter measures while compiling

To complete the first step, I signed into Kail VM and from there went to D2L to Download StackOverFlowHW.cpp from the Assignment 5 tab in D2L.

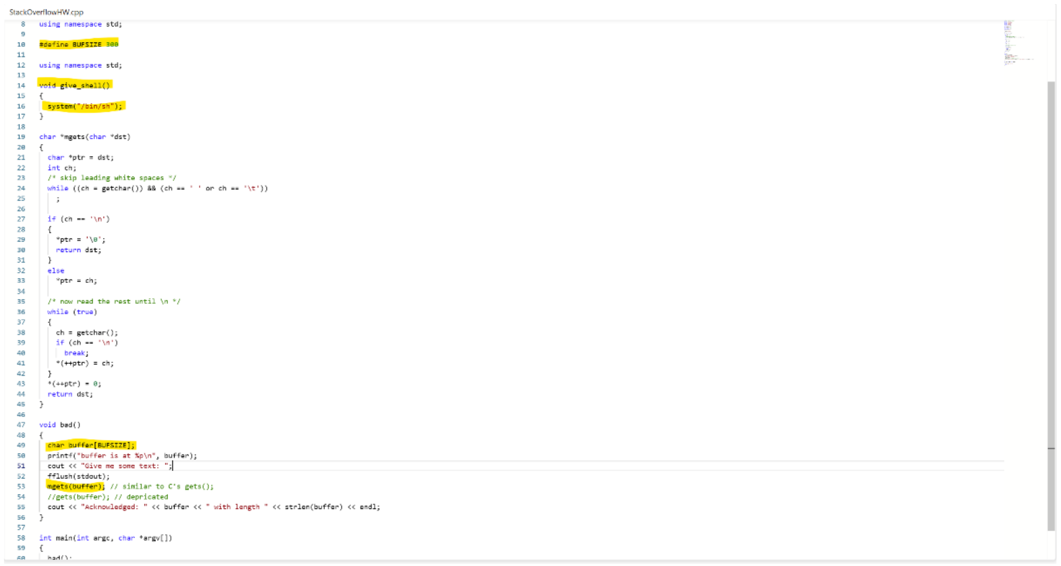
In the second step I compiled the StackOverFlowHW.cpp program as an x86 binary via a Makefile or by a bash script that is called compile.sh. For the assignment I chose to compile the program via a Makefile that I have edited to include -fno-stack-protector, z execstack, and no-pie flags that allow me to disable all the countermeasures to complete the rest of the assignment.


rsmedina@SebsHome: ~/School/A5_Stack
File Actions Edit View Help

```
# Compiler name
CXX = g++
# g++ compiler flags
# -g: adds debugging information to the executable file
# -Wall: turns on most, but not  all, compiler warnings
# -std=c++17: uses c++17 standard
# Put all the required flags separated by space
CXXFLAGS = -fno-stack-protector -z execstack -no-pie -g -o

# The build target executable file; .exe is not required
PROGRAM = StackHW.out

# List .cpp files separated by space
CPPFILES = StackOverflowHW.cpp

# Make all rule/default rule that'll be run when we run "make"
# First compile each file to object file
# Then make a single binary/executable program
all:
#   $(g++) $(-c -g -Wall -std=c++17) $(file.cpp)
        $(CXX) $(CXXFLAGS) $(PROGRAM) $(CPPFILES)
#   $(g++) -o $(file.out) *.o
#       $(CXX) -o $(PROGRAM)

# Make clean rule
# Delete object and binary files
clean:
        rm -f $(PROGRAM) *.o

# Rule to run the program
run:
        ./$(PROGRAM)
# To Make and Run, type: make && make run
~
~
~
~
~
~
"Makefile" 33L, 920B                                    8,11          All
```

2. Manual static analysis

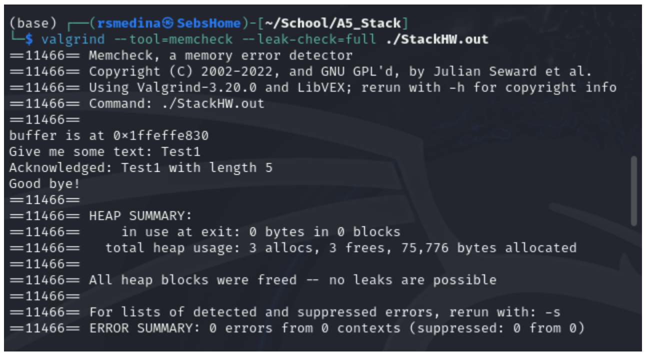
For the next step I performed a manual static analysis to see what vulnerabilities I could find. While performing the manual static analysis the first thing I saw was in the give_shell() function witch calls the (/bin/sh) witch runs a shell command. This is a vulnerability because it allows the program to access the shell which can comprise your system if the program is malicious. Another vulnerability that I noticed is that the mget() function reads inputs into buffer from the bad() function. The buffer is fixed at 300 and if the input is larger than 300 it will lead to a buffer overflow. Another instance of a possible buffer overflow is in the bad() function that creates a char array that uses the fixed BUFSIZE amount.



```
using namespace std;

#define BUFSIZE 300

using namespace std;

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

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;
}

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

int main(int argc, char *argv[])
{
  bad()
```

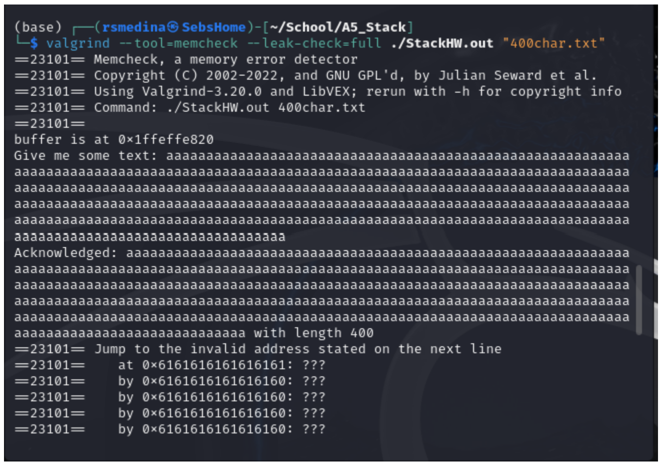
3. Valgrind Memcheck Dynamix Analysis

For the next step we had to use Valgrind Memcheck which is a tool that can be used to detect memory leaks, memory corruption, and any errors related to memory. When you first run Valgrind with the input being under the buffer size you get no errors, and the program seems to be running fine.

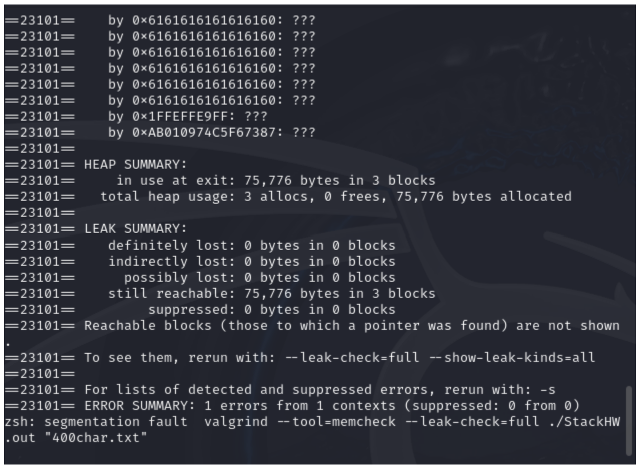


```
(base) ┌──(rsmedina㉿SebsHome)-[~/School/A5_Stack]
└─$ valgrind --tool=memcheck --leak-check=full ./StackHW.out
==11466== Memcheck, a memory error detector
==11466== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==11466== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==11466== Command: ./StackHW.out
==11466==
buffer is at 0x1ffeffe830
Give me some text: Test1
Acknowledged: Test1 with length 5
Good bye!
==11466==
==11466== HEAP SUMMARY:
==11466==     in use at exit: 0 bytes in 0 blocks
==11466==   total heap usage: 3 allocs, 3 frees, 75,776 bytes allocated
==11466==
==11466== All heap blocks were freed -- no leaks are possible
==11466==
==11466== For lists of detected and suppressed errors, rerun with: -s
==11466== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

However, whenever you enter in an input that is larger than your buffer size Valgrind will show that a segmentation fault has occurred.



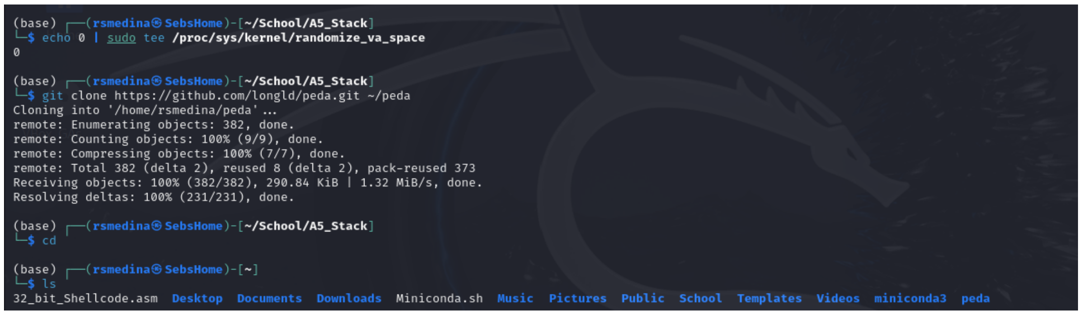
```
(base) ┌──(rsmedina㉿SebsHome)-[~/School/A5_Stack]
└─$ valgrind --tool=memcheck --leak-check=full ./StackHW.out "400char.txt"
==23101== Memcheck, a memory error detector
==23101== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==23101== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==23101== Command: ./StackHW.out 400char.txt
==23101==
buffer is at 0x1ffeffe820
Give me some text: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Acknowledged: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa with length 400
==23101== Jump to the invalid address stated on the next line
==23101==    at 0x6161616161616161: ???
==23101==    by 0x6161616161616160: ???
==23101==    by 0x6161616161616160: ???
==23101==    by 0x6161616161616160: ???
==23101==    by 0x6161616161616160: ???
```



```
==23101==    by 0×6161616161616160: ???
==23101==    by 0×6161616161616160: ???
==23101==    by 0×6161616161616160: ???
==23101==    by 0×6161616161616160: ???
==23101==    by 0×6161616161616160: ???
==23101==    by 0×6161616161616160: ???
==23101==    by 0×1FFEFFE9FF: ???
==23101==    by 0×AB010974C5F67387: ???
==23101==
==23101== HEAP SUMMARY:
==23101==     in use at exit: 75,776 bytes in 3 blocks
==23101==   total heap usage: 3 allocs, 0 frees, 75,776 bytes allocated
==23101==
==23101== LEAK SUMMARY:
==23101==    definitely lost: 0 bytes in 0 blocks
==23101==    indirectly lost: 0 bytes in 0 blocks
==23101==      possibly lost: 0 bytes in 0 blocks
==23101==    still reachable: 75,776 bytes in 3 blocks
==23101==         suppressed: 0 bytes in 0 blocks
==23101== Reachable blocks (those to which a pointer was found) are not shown
.
==23101== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==23101==
==23101== For lists of detected and suppressed errors, rerun with: -s
==23101== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
zsh: segmentation fault  valgrind --tool=memcheck --leak-check=full ./StackHW
.out "400char.txt"
```

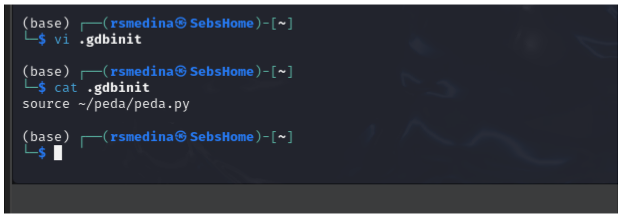
4. Exploit the Program

To start Exploiting the programs I had to disable the randomization of the buffer address which will make the buffer address predictable and easier to launch a buffer overflow attack. To do this I ran the command "echo 0 | sudo tee /proc/sys/kernel/randomize_va-space". Also, I installed Peda which is a gdb extension that boosts the normal gdb debugger. Peda boost gdb by allowing more pertinent information to be viewable. The way I installed peda is with the command "git clone https://github.com/longid/peda.git ~/peda". This command installs peda in my home directory.



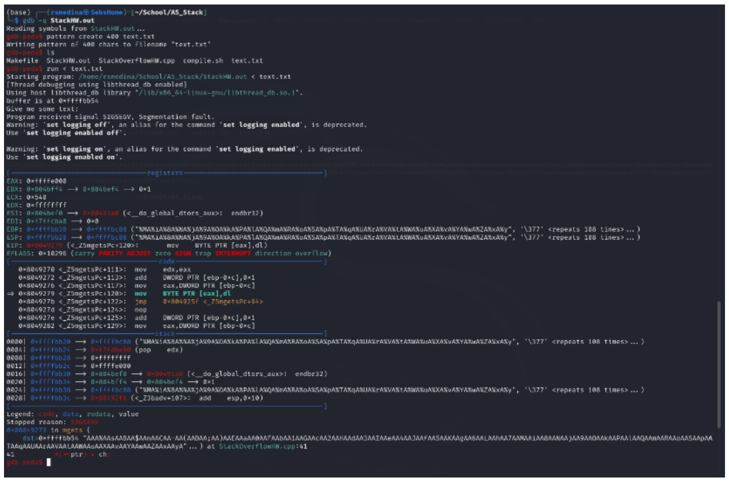
```
(base) ┌──(rsmedina㉿SebsHome)-[~/School/A5_Stack]
└─$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
0

(base) ┌──(rsmedina㉿SebsHome)-[~/School/A5_Stack]
└─$ git clone https://github.com/longld/peda.git ~/peda
Cloning into '/home/rsmedina/peda'...
remote: Enumerating objects: 382, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 382 (delta 2), reused 8 (delta 2), pack-reused 373
Receiving objects: 100% (382/382), 290.84 KiB | 1.32 MiB/s, done.
Resolving deltas: 100% (231/231), done.

(base) ┌──(rsmedina㉿SebsHome)-[~/School/A5_Stack]
└─$ cd

(base) ┌──(rsmedina㉿SebsHome)-[~]
└─$ ls
32_bit_Shellcode.asm  Desktop  Documents  Downloads  Miniconda.sh  Music  Pictures  Public  School  Templates  Videos  miniconda3  peda
```

To make sure that the peda is configured properly I went into .gdbinit file and make sure that it said “source ~/peda/peda.py” as seen in the image below.

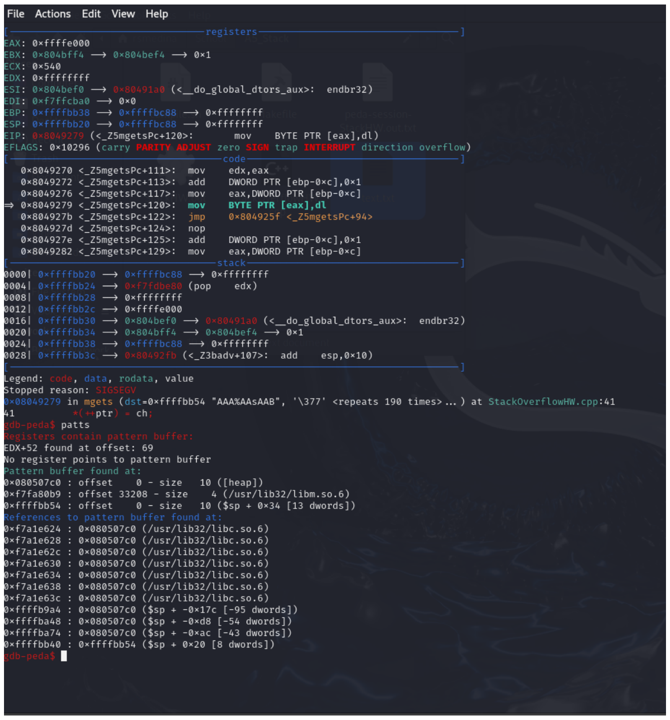


```
(base) ┌──(rsmedina㉿SebsHome)-[~]
└─$ vi .gdbinit

(base) ┌──(rsmedina㉿SebsHome)-[~]
└─$ cat .gdbinit
source ~/peda/peda.py

(base) ┌──(rsmedina㉿SebsHome)-[~]
└─$ 
```

Next it time to do the attack since peda is all installed and configured already. However, before we do this attack we must find the offset. This is a very important step because it lets me know where exactly to launch the exploit so I can perform the attack. However, for me I ran into a snag. I compiled “StackOverflowHW.cpp” three different ways and gdb would not successfully locate the buffer size. I compiled “StackOverflowHW.cpp” via my make file which has all the counter measure turned off, also I compiled “StackOverflowHW.cpp” with compile.sh and lastly with the command “g++ -m32 -fno-stack-protector -z execstack -no-pie -o StackHW StackOverflowHW.cpp” . I took all binaries from all the ways I complied and ran all of them though gdb with the command “gdb StackHW”. Once in gdb I created a pattern of 400 characters named text.txt and ran that though gdb program. This is where I encountered a problem. For some reason no matter how, I ran or changed the binary the text.txt file would not enter into the program and then when I ran patts in gdb the correct pattern buffer would not show up.



```
(base) ┌──(rsmedina㉿SebsHome)-[~/School/A5_Stack]
└─$ gdb -q StackHW.out
Reading symbols from StackHW.out...
gdb-peda$ pattern create 400 text.txt
Writing pattern of 400 chars to filename "text.txt"
gdb-peda$ ls
Makefile  StackHW.out  StackOverflowHW.cpp  compile.sh  text.txt
gdb-peda$ run < text.txt
Starting program: /home/rsmedina/School/A5_Stack/StackHW.out < text.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
buffer is at 0×ffffbb54
Give me some text:
Program received signal SIGSEGV, Segmentation fault.
Warning: 'set logging off', an alias for the command 'set logging enabled', is 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'.
[──────────────────────────────registers──────────────────────────────]
EAX: 0×ffffe000
EBX: 0×804bff4 ⟶ 0×804bef4 ⟶ 0×1
ECX: 0×540
EDX: 0×ffffffff
ESI: 0×804bef0 ⟶ 0×80491a0 (<__do_global_dtors_aux>:  endbr32)
EDI: 0×f7ffcba0 ⟶ 0×0
EBP: 0×ffffbb38 ⟶ 0×ffffbc88 ("%MA%iA%8A%NA%jA%9A%OA%kA%PA%lA%QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y", '\377' <repeats 188 times>...)
ESP: 0×ffffbb20 ⟶ 0×ffffbc88 ("%MA%iA%8A%NA%jA%9A%OA%kA%PA%lA%QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y", '\377' <repeats 188 times>...)
EIP: 0×8049279 (<_Z5mgetsPc+120>:       mov    BYTE PTR [eax],dl)
EFLAGS: 0×10296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[─────────────────────────────────code─────────────────────────────────]
   0×8049270 <_Z5mgetsPc+111>:  mov    edx,eax
   0×8049272 <_Z5mgetsPc+113>:  add    DWORD PTR [ebp-0×c],0×1
   0×8049276 <_Z5mgetsPc+117>:  mov    eax,DWORD PTR [ebp-0×c]
⇒  0×8049279 <_Z5mgetsPc+120>:  mov    BYTE PTR [eax],dl
   0×804927b <_Z5mgetsPc+122>:  jmp    0×804925f <_Z5mgetsPc+94>
   0×804927d <_Z5mgetsPc+124>:  nop
   0×804927e <_Z5mgetsPc+125>:  add    DWORD PTR [ebp-0×c],0×1
   0×8049282 <_Z5mgetsPc+129>:  mov    eax,DWORD PTR [ebp-0×c]
[─────────────────────────────────stack────────────────────────────────]
0000| 0×ffffbb20 ⟶ 0×ffffbc88 ("%MA%iA%8A%NA%jA%9A%OA%kA%PA%lA%QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y", '\377' <repeats 108 times>...)
0004| 0×ffffbb24 ⟶ 0×f7fdbe80 (pop    edx)
0008| 0×ffffbb28 ⟶ 0×ffffffff
0012| 0×ffffbb2c ⟶ 0×ffffe000
0016| 0×ffffbb30 ⟶ 0×804bef0 ⟶ 0×80491a0 (<__do_global_dtors_aux>:  endbr32)
0020| 0×ffffbb34 ⟶ 0×804bff4 ⟶ 0×804bef4 ⟶ 0×1
0024| 0×ffffbb38 ⟶ 0×ffffbc88 ("%MA%iA%8A%NA%jA%9A%OA%kA%PA%lA%QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y", '\377' <repeats 108 times>...)
0028| 0×ffffbb3c ⟶ 0×80492fb (<_Z3badv+107>:  add    esp,0×10)
[──────────────────────────────────────────────────────────────────────]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0×08049279 in mgets (
    dst=0×ffffbb54 "AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAA
TAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA"...) at StackOverflowHW.cpp:41
41          *(++ptr) = ch;
gdb-peda$ █
```

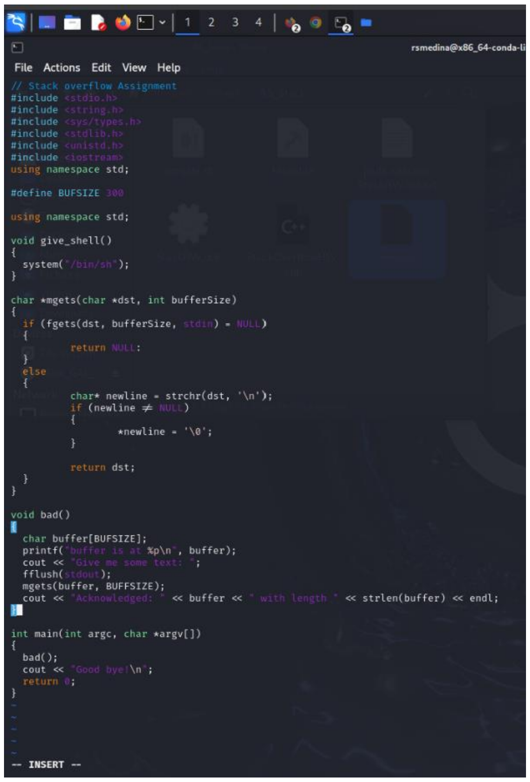


```
[──────────────────────────────registers──────────────────────────────]
EAX: 0×ffffe000
EBX: 0×804bff4 ⟶ 0×804bef4 ⟶ 0×1
ECX: 0×540
EDX: 0×ffffffff
ESI: 0×804bef0 ⟶ 0×80491a0 (<__do_global_dtors_aux>:  endbr32)
EDI: 0×f7ffcba0 ⟶ 0×0
EBP: 0×ffffbb38 ⟶ 0×ffffbc88 ⟶ 0×ffffffff
ESP: 0×ffffbb20 ⟶ 0×ffffbc88 ⟶ 0×ffffffff
EIP: 0×8049279 (<_Z5mgetsPc+120>:       mov    BYTE PTR [eax],dl)
EFLAGS: 0×10296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[─────────────────────────────────code─────────────────────────────────]
   0×8049270 <_Z5mgetsPc+111>:  mov    edx,eax
   0×8049272 <_Z5mgetsPc+113>:  add    DWORD PTR [ebp-0×c],0×1
   0×8049276 <_Z5mgetsPc+117>:  mov    eax,DWORD PTR [ebp-0×c]
⇒  0×8049279 <_Z5mgetsPc+120>:  mov    BYTE PTR [eax],dl
   0×804927b <_Z5mgetsPc+122>:  jmp    0×804925f <_Z5mgetsPc+94>
   0×804927d <_Z5mgetsPc+124>:  nop
   0×804927e <_Z5mgetsPc+125>:  add    DWORD PTR [ebp-0×c],0×1
   0×8049282 <_Z5mgetsPc+129>:  mov    eax,DWORD PTR [ebp-0×c]
[─────────────────────────────────stack────────────────────────────────]
0000| 0×ffffbb20 ⟶ 0×ffffbc88 ⟶ 0×ffffffff
0004| 0×ffffbb24 ⟶ 0×f7fdbe80 (pop    edx)
0008| 0×ffffbb28 ⟶ 0×ffffffff
0012| 0×ffffbb2c ⟶ 0×ffffe000
0016| 0×ffffbb30 ⟶ 0×804bef0 ⟶ 0×80491a0 (<__do_global_dtors_aux>:  endbr32)
0020| 0×ffffbb34 ⟶ 0×804bff4 ⟶ 0×804bef4 ⟶ 0×1
0024| 0×ffffbb38 ⟶ 0×ffffbc88 ⟶ 0×ffffffff
0028| 0×ffffbb3c ⟶ 0×80492fb (<_Z3badv+107>:  add    esp,0×10)
[──────────────────────────────────────────────────────────────────────]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0×08049279 in mgets (dst=0×ffffbb54 "AAA%AAsAAB", '\377' <repeats 190 times>...) at StackOverflowHW.cpp:41
41          *(++ptr) = ch;
gdb-peda$ patts
Registers contain pattern buffer:
EDX+52 found at offset: 69
No register points to pattern buffer
Pattern buffer found at:
0×080507c0 : offset     0 - size   10 ([heap])
0×f7fa80b9 : offset 33208 - size    4 (/usr/lib32/libm.so.6)
0×ffffbb54 : offset     0 - size   10 ($sp + 0×34 [13 dwords])
References to pattern buffer found at:
0×f7a1e624 : 0×080507c0 (/usr/lib32/libc.so.6)
0×f7a1e628 : 0×080507c0 (/usr/lib32/libc.so.6)
0×f7a1e62c : 0×080507c0 (/usr/lib32/libc.so.6)
0×f7a1e630 : 0×080507c0 (/usr/lib32/libc.so.6)
0×f7a1e634 : 0×080507c0 (/usr/lib32/libc.so.6)
0×f7a1e638 : 0×080507c0 (/usr/lib32/libc.so.6)
0×f7a1e63c : 0×080507c0 (/usr/lib32/libc.so.6)
0×ffffb9a4 : 0×080507c0 ($sp + -0×17c [-95 dwords])
0×ffffba48 : 0×080507c0 ($sp + -0×d8 [-54 dwords])
0×ffffba74 : 0×080507c0 ($sp + -0×ac [-43 dwords])
0×ffffbb40 : 0×ffffbb54 ($sp + 0×20 [8 dwords])
gdb-peda$ █
```

Since I was not able to figure out exactly where the buffer was, I cannot complete the exploit. However, these are the next steps I would take if I could figure out my problem and get the buffer size. I would start by making the binary “StackHW.out” have root privileges and change the file permissions to 4755. Also, when I know the correct size, I can use the “nm” command to obtain the return address of the shell. Once I gain the return address of the shell, I can craft a payload using python. However, since I do not know I can not create the correct payload needed for this lab. After the payload was completed and built, I would run the payload with “StackHW.out” to successfully launch the attack.

5. Patching the Vulnerability

Now the task is to fix the vulnerability within StackOverflowHW.cpp so that buffer overflow attacks are less likely to occur. So the change that I made was within the char *mgets () function I changed where it used to say mgets() to fgets(). This means that the inputs are read form “stdin.fgets. Also another change I made is that under this same function was to mgets() because now mgets() can take in two arguments instead of one. This also ensures that the function mgets knows the maximum Buff size that can be read into it and stops people from going beyond the buffer size. Also, the new fixes will print out NULL when there is a weird input.



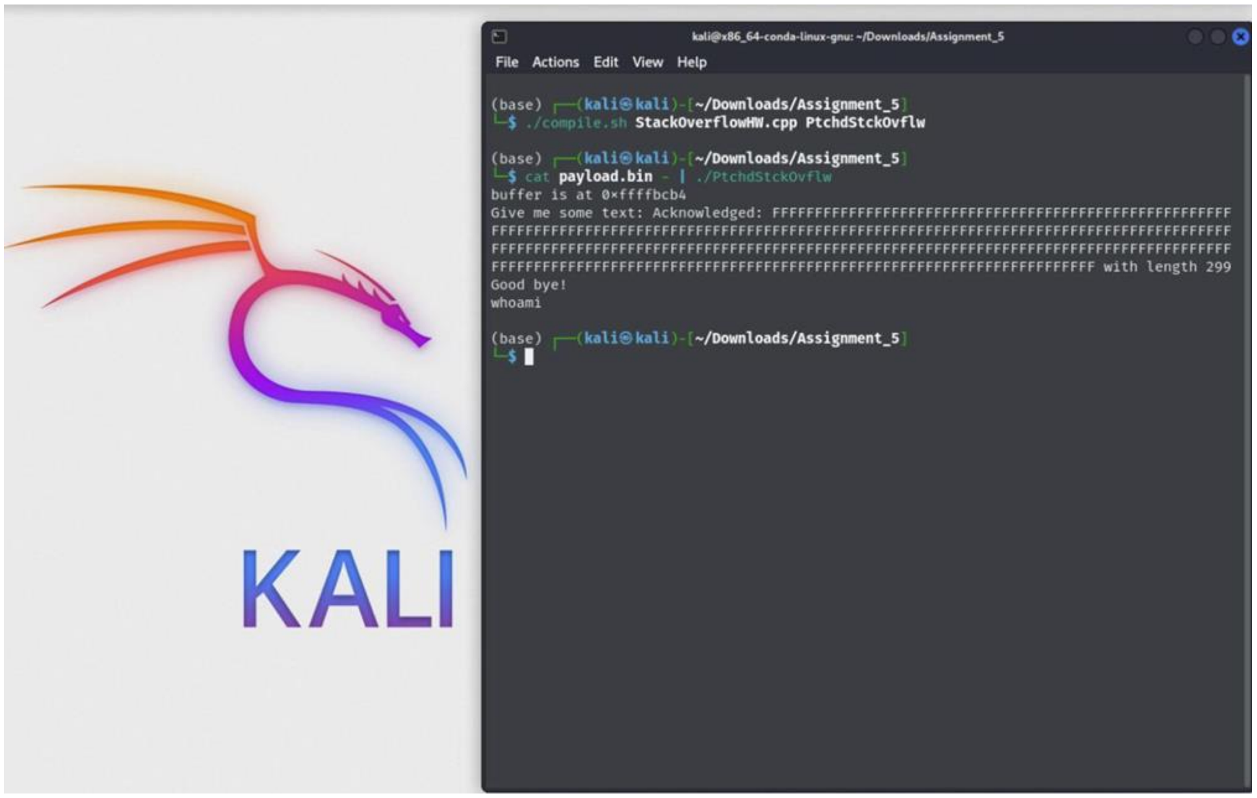
```
// Stack overflow Assignment
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <iostream>
using namespace std;

#define BUFSIZE 300

using namespace std;

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

char *mgets(char *dst, int bufferSize)
{
  if (fgets(dst, bufferSize, stdin) = NULL)
  {
          return NULL:
  }
  else
  {
          char* newline = strchr(dst, '\n');
          if (newline ≠ NULL)
          {
                  *newline = '\0';
          }

          return dst;
  }
}

void bad()
{
  char buffer[BUFSIZE];
  printf("buffer is at %p\n", buffer);
  cout << "Give me some text: ";
  fflush(stdout);
  mgets(buffer, BUFFSIZE);
  cout << "Acknowledged: " << buffer << " with length " << strlen(buffer) << endl;
}

int main(int argc, char *argv[])
{
  bad();
  cout << "Good bye!\n";
  return 0;
}
~
~
~
~
~
-- INSERT --
```

6. Recompiling and doing Dynamically Analysis

In this task I must complete a manual analysis with the new "StackOverflowHW.cpp" with Valgrind. I compiled the new program I remade as "NstackHW" and then ran that program in Valgrind. When I run this with more than 300 characters it does not error out on me at all. This is because of the new function fget(). This makes sure that the bash overflow will not occur.

7. Reenabling Countermeasures and Attempting Exploit

For this assignment since I was not able to exploit the program, I sent the modified better version to my friend Sam Evans to see if he can run this and see if he can do a Buffer overflow attack on it. This is what he sent me back and buffer overflow was successful. He changed the name on the file.


kali@x86_64-conda-linux-gnu: ~/Downloads/Assignment_5
File Actions Edit View Help
KALI

```
(base) ┌──(kali㉿kali)-[~/Downloads/Assignment_5]
└─$ ./compile.sh StackOverflowHW.cpp PtchdStckOvflw

(base) ┌──(kali㉿kali)-[~/Downloads/Assignment_5]
└─$ cat payload.bin - | ./PtchdStckOvflw
buffer is at 0×ffffbcb4
Give me some text: Acknowledged: FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF with length 299
Good bye!
whoami

(base) ┌──(kali㉿kali)-[~/Downloads/Assignment_5]
└─$ █
```