Analyzing BufferOverflow with GDB

Pre-requisite: GDB (Step by Step Introduction)

A **BufferOverflow** often occurs when the content inside the defined variable is copied to another variable without doing **Bound Checks** or considering the size of the buffer. Let's analyze buffer overflow with the help GNU Debugger (GDB) which is inbuilt every Linux system.

The motive of this exercise is to get comfortable with debugging code and understand how does buffer overflow works in action.

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

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

   cantoverflowme = 0;
   gets(buffer);

   if (cantoverflowme != 0) {
      printf("You OVERFLOW'ed Me\n");
   }
   else {
      printf("Can't Overflow Me\n");
   }
}
```

- **Step 1**
  Let's compile this code with the following flags :

  ```
  gcc overflow.c -o overflow  -fno-stack-protector -z execstack -no-pie
  ```

  The above code is going to create a compiled binary that disables various stack protections

  ```
  -z execstack : Disables Executable Stack
  -fno-stack-protector : Disables Stack Canaries
  -no-pie : Disables Position Independent Executables
  ```

- **Step 2**
  Now that stack protections are disabled we can load the code in GDB by typing

  ```
  gdb ./overflow
  ```

- **Step 3**
  Once the code is open we can look at the functions that are inside the binary by using typing

  ```
  info functions
  ```

root@kali: ~/BOF-1

```
root@kali:~/BOF-1# gdb ./overflow
GNU gdb (Debian 8.1-4+b1) 8.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./overflow...(no debugging symbols found)...done.
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x0000000000401000  _init
0x0000000000401030  puts@plt
0x0000000000401040  gets@plt
0x0000000000401050  _start
0x0000000000401080  _dl_relocate_static_pie
0x0000000000401090  deregister_tm_clones
0x00000000004010c0  register_tm_clones
0x0000000000401100  __do_global_dtors_aux
0x0000000000401130  frame_dummy
0x0000000000401132  main
0x0000000000401190  __libc_csu_init
0x0000000000401200  __libc_csu_fini
0x0000000000401204  _fini
(gdb)
```

We can see there's a gets call which is being used which is vulnerable in nature as it doesn't do any bound checks.

- **Step 4**
  Let's type

  ```
  disas main
  ```

  and disassemble the main function

```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000401132 <+0>:     push   %rbp
   0x0000000000401133 <+1>:     mov    %rsp,%rbp
   0x0000000000401136 <+4>:     sub    $0x60,%rsp
   0x000000000040113a <+8>:     mov    %edi,-0x54(%rbp)
   0x000000000040113d <+11>:    mov    %rsi,-0x60(%rbp)
   0x0000000000401141 <+15>:    movl   $0x0,-0x4(%rbp)
   0x0000000000401148 <+22>:    lea    -0x50(%rbp),%rax
   0x000000000040114c <+26>:    mov    %rax,%rdi
   0x000000000040114f <+29>:    mov    $0x0,%eax
   0x0000000000401154 <+34>:    callq  0x401040 <gets@plt>
   0x0000000000401159 <+39>:    mov    -0x4(%rbp),%eax
   0x000000000040115c <+42>:    test   %eax,%eax
   0x000000000040115e <+44>:    je     0x40116e <main+60>
   0x0000000000401160 <+46>:    lea    0xe9d(%rip),%rdi        # 0x402004
   0x0000000000401167 <+53>:    callq  0x401030 <puts@plt>
   0x000000000040116c <+58>:    jmp    0x40117a <main+72>
   0x000000000040116e <+60>:    lea    0xea2(%rip),%rdi        # 0x402017
   0x0000000000401175 <+67>:    callq  0x401030 <puts@plt>
   0x000000000040117a <+72>:    mov    $0x0,%eax
   0x000000000040117f <+77>:    leaveq
   0x0000000000401180 <+78>:    retq
End of assembler dump.
```

- **Step 5**

  Let's put a breakpoint by typing

  ```
  b * main+39
  ```

  so that we can analyze the content of stack when the program hits the breakpoint.

- **Step 6**

  Type

  ```
  r
  ```

  to run the code and input any number of A's as we already know from the code above.

  Let's input **63 A's and 78 A's** and see the change in the result.

- **Step 7**

  You can use python code to print A's by typing after leaving the GDB.

  ```
  python -c "print 'A' * 63"
  ```

```
root@kali:~/BOF-1# python -c "print 'A' * 63"
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
root@kali:~/BOF-1#
```

- **Step 8**

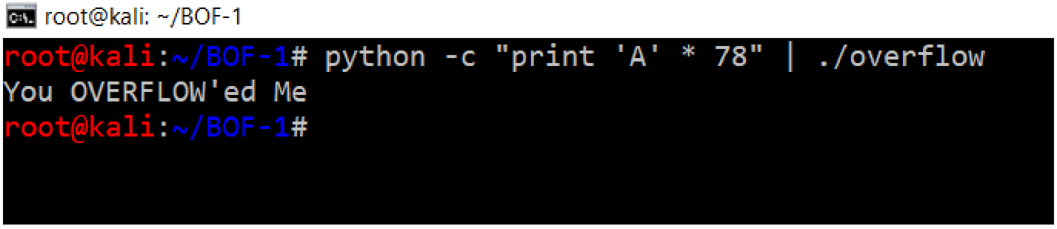
  Now that we have 63 A's let's run the code and paste it when it ask's us for the input.

```
root@kali:~/BOF-1# python -c "print 'A' * 63"
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
root@kali:~/BOF-1#
```

  Let's try the whole process again and this time let's input any number of A's let's say 78.

  A cool way to do this can be

  ```
  python -c "print 'A' * 78" | ./overflow
  ```



As we can see once the **overflow occurs** it changes the variable because of memory being leaked on the stack and changing values of variables

- **Step 9**
  Let's check the stack which it over writes, so we have to set a break point at

  ```
  main+39
  ```

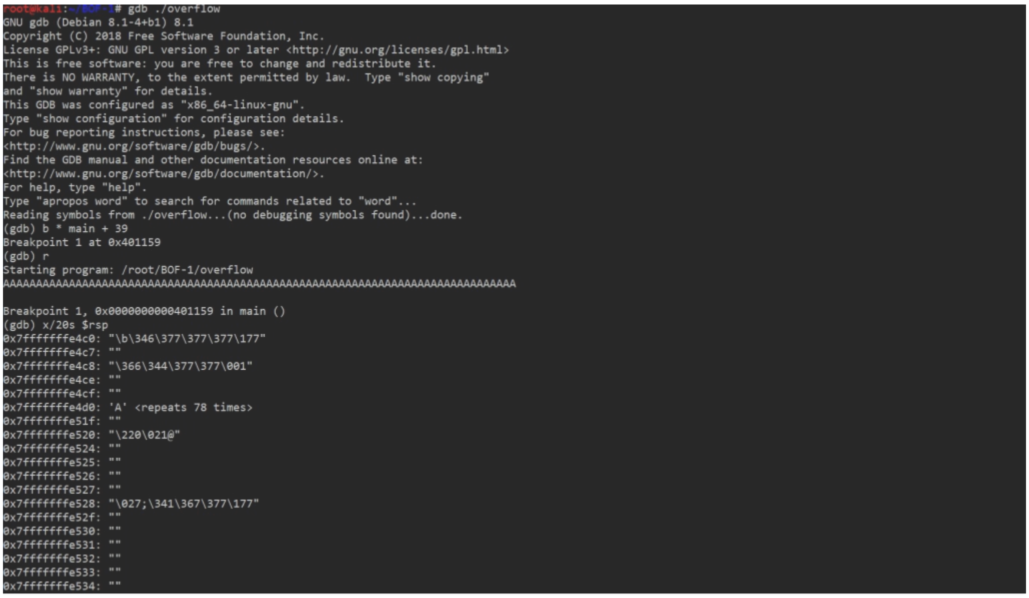
  then type

  ```
  r
  ```

  and then we can type

  ```
  x/20s $rsp
  ```

  ```
  x : eXamine
  20s : 20 values in string
  $rsp : for register RSP (Stack Pointer)
  ```



Hence we can see how **78 A's are written on the stack and are overflowing the memory**.

Recommended Posts:

System Protection in Operating System
C-SCAN Disk Scheduling Algorithm
Difference between Loading and Linking
asctime() and asctime_s() functions in C with Examples
Difference between Spoofing and Phishing
Difference between File and Folder
Reader-Writer problem using Monitors (pthreads)

Find all unique pairs of maximum and second maximum elements over all sub-arrays in O(NlogN)

time.h header file in C with Examples

SetUID, SetGID, and Sticky Bits in Linux File Permissions

Python | Stack using Doubly Linked List

InfyTQ 2019 : Find the position from where the parenthesis is not balanced

Classical problems of Synchronization with Semaphore Solution

SCAN (Elevator) Disk Scheduling Algorithms

Coding_Karma
Check out this Author's contributed articles.

If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please Improve this article if you find anything incorrect by clicking on the "Improve Article" button below.

Article Tags : C | Operating Systems | Stack | Cyber-security | Information-Security

Practice Tags : Operating Systems | Stack | C

Be the First to upvote.

To-do Done

0

No votes yet.

Feedback/ Suggest Improvement | Add Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Previous

Check if the two given stacks are same

Next

C Program to count the Number of Characters in a File

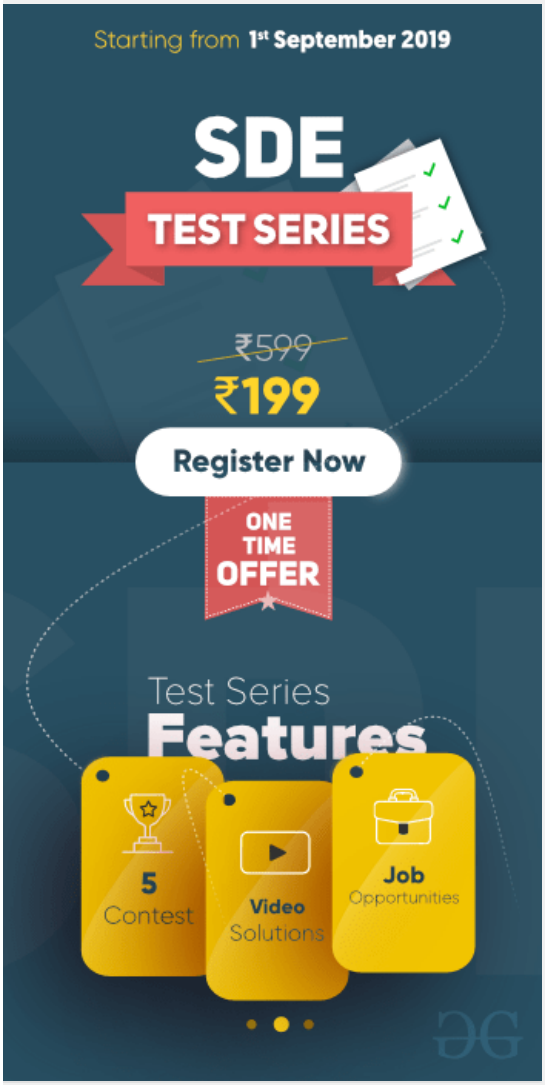
Load Comments

Most popular articles

- Must Do Coding Questions for Companies like Amazon, Microsoft, Adobe, ...
- Must Do Coding Questions Company-wise
- Sapient Interview Experience (ASDE1)
- Citrix Interview Experience (On Campus 2019)
- Samsung Interview Experience through Co-cubes (2019)

Most visited in Operating Systems

- Difference between Virtual memory and Cache memory
- Message Passing in Java
- Classical problems of Synchronization with Semaphore Solution
- Difference between Process and Thread
- Difference between Multiprocessing and Multithreading


Starting from 1st September 2019
SDE
TEST SERIES
₹599
₹199
Register Now
ONE TIME OFFER
Test Series
Features
5 Contest
Video Solutions
Job Opportunities