Lab # 3: Complete Buffer-Overflow vulnerability lab

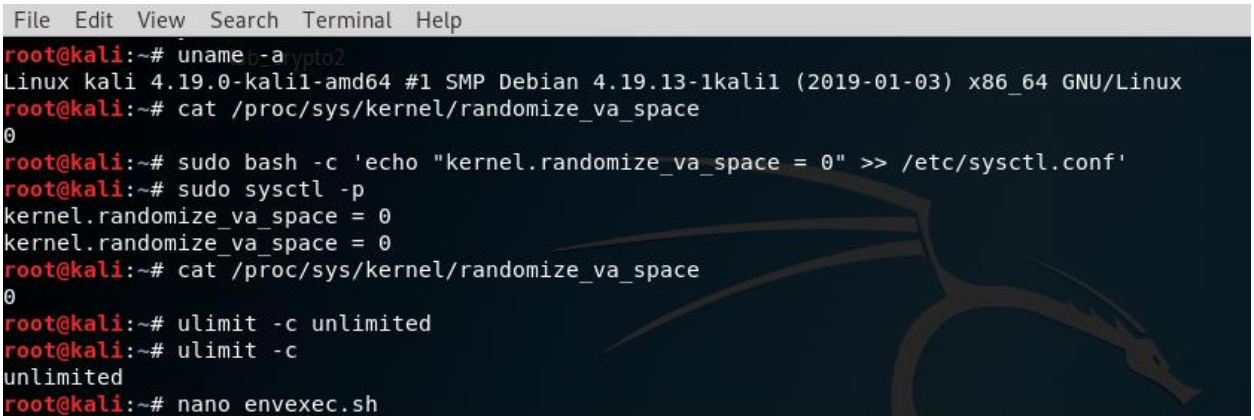
Md Rony
John Jay College of Criminal Justice

When more data is put into a fixed-sized buffer than the buffer is capable for is called buffer overflow. The extra information or data which overflowed has to go somewhere, example - into adjacent memory space, corrupting or overwriting the data held in that space. Those overflowed information or data is vulnerable which can be attacked that abuses a type of bug called a “buffer overflow”, in which a program overwrites memory adjacent to a buffer that should not have been modified intentionally or unintentionally. It causes a system can be crashed which lead to cyber-attack.

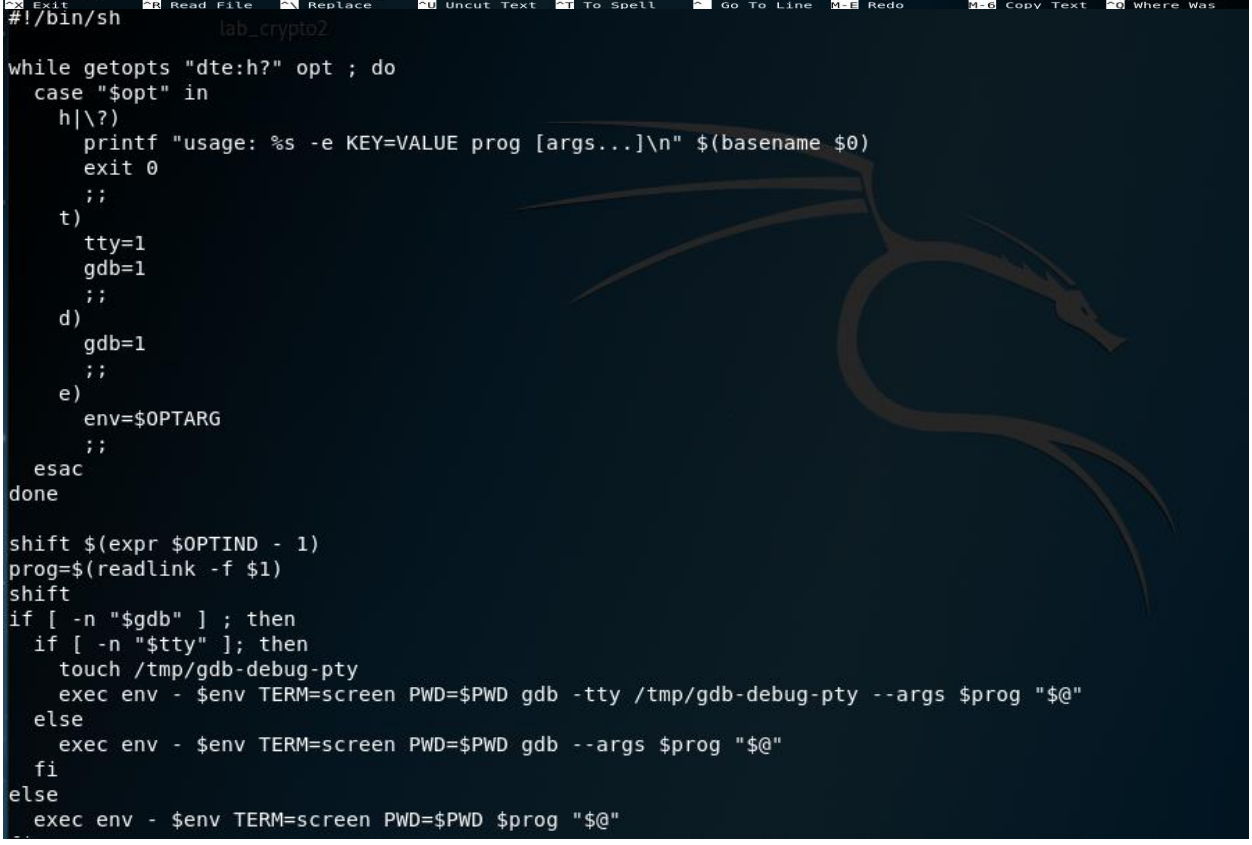
It is important to understand what buffer overflows are and how they pose to attack applications, and what techniques are used to exploit these vulnerabilities effective way.

I exploited on kali Linux operating system and how I did my lab is below.

I disabled memory randomization and enabled core dumps and verified that it is “0” and is unlimited.



I wrote a envexec.sh script in bash typing command “nano” I copied a code from https://gist.github.com/apolloclark/6cffb33f179cc9162d0a



sameway I created another vulnerable code file named vuln.c

```
root@kali:~# nano vuln.c
root@kali:~# cat vuln.c
#include <stdio.h>
#include <string.h>

int main (int argc, char** argv)
{
        char buffer[500];
        strcpy(buffer, argv[1]);

        return 0;
}
```

```
  GNU nano 3.2
#include <stdio.h>
#include <string.h>

int main (int argc, char** argv)
{
        char buffer[500];
        strcpy(buffer, argv[1]);

        return 0;
}
```

then I compiled the code and debugged
and run gbd to laod the program to analyze it.

```
root@kali:~# gcc -z execstack -fno-stack-protector -mpreferred-stack-boundary=2 -g vuln.c -o vuln
cc1: error: -mpreferred-stack-boundary=2 is not between 3 and 12
root@kali:~# gcc -z execstack -fno-stack-protector -mpreferred-stack-boundary=3 -g vuln.c -o vuln
root@kali:~# chmod +x envexec.sh
root@kali:~# ls
'?'          Downloads                libevent-1.4.14b-stable.tar.gz   Pictures          Python-3.3.0.tgz     Python-3.3.0.tgz.3   vuln
 Desktop     envexec.sh               libevent-1.4.14b-stable.tar.gz.1 Public            Python-3.3.0.tgz.1   Templates            vuln.c
 Documents   libevent-1.4.14b-stable  Music                            Python-3.3.0      Python-3.3.0.tgz.2   Videos
root@kali:~# ./envexec.sh -d vuln
GNU gdb (Debian 8.2-1) 8.2
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 /root/vuln...done.
(gdb)
```

gdb commands
To get information I typed “list” , “disas main”, ” info os” ,“info functions” ,”info variables”

```
(gdb) list
293     in ../sysdeps/x86_64/multiarch/strcpy-sse2-unaligned.S
(gdb) info os
Type        Description
cpus        Listing of all cpus/cores on the system
files       Listing of all file descriptors
modules     Listing of all loaded kernel modules
msg         Listing of all message queues
processes   Listing of all processes
procgroups  Listing of all process groups
semaphores  Listing of all semaphores
shm         Listing of all shared-memory regions
sockets     Listing of all internet-domain sockets
threads     Listing of all threads
(gdb) info functions
All defined functions:

File ../argp/argp-fmtstream.h:
266:    size_t __argp_fmtstream_point(argp_fmtstream_t);
220:    int __argp_fmtstream_putc(argp_fmtstream_t, int);
207:    int __argp_fmtstream_puts(argp_fmtstream_t, const char *);
230:    size_t __argp_fmtstream_set_lmargin(argp_fmtstream_t, size_t);
242:    size_t __argp_fmtstream_set_rmargin(argp_fmtstream_t, size_t);
254:    size_t __argp_fmtstream_set_wmargin(argp_fmtstream_t, size_t);
194:    size_t __argp_fmtstream_write(argp_fmtstream_t, const char *, size_t);

File ../argp/argp.h:
526:    void __argp_usage(const argp_state *);
544:    int __option_is_end(const argp_option *);
532:    int __option_is_short(const argp_option *);
```

```
0x00007ffff7faa360  buffer_size.11974
0x00007ffff7faa370  start_fct.12547
0x00007ffff7faa378  startp.12546
0x00007ffff7faa380  startp_initialized.12545
0x00007ffff7faa3a0  buffer_size.11914
0x00007ffff7faa3c0  resbuf.11918
0x00007ffff7faa408  result
0x00007ffff7faa410  start_fct
0x00007ffff7faa418  startp
0x00007ffff7faa420  asc
0x00007ffff7faa438  start_fct.10680
0x00007ffff7faa440  startp.10679
0x00007ffff7faa4e0  startp
0x00007ffff7faa4e8  startp_initialized
0x00007ffff7faa55c  once
0x00007ffff7faa580  buffer_size
0x00007ffff7faa5a0  resbuf
0x00007ffff7faa5e0  resbuf
0x00007ffff7faa600  buffer_size
0x00007ffff7faa610  start_fct
0x00007ffff7faa618  startp
0x00007ffff7faa620  startp_initialized
0x00007ffff7faa624  lock
0x00007ffff7faa628  not_first
0x00007ffff7faa630  functions
0x00007ffff7faa660  tmpbuf
0x00007ffff7faa8e0  lock
0x00007ffff7faa8e4  once
0x00007ffff7faa900  _res@GLIBC_2.2.5
0x00007ffff7faabc0  default table
0x00007ffff7faad90  startp
0x00007ffff7faada0  svcauthdes_stats@GLIBC_2.2.5
0x00007ffff7faadc0  buffer_size
0x00007ffff7faadd0  resbuf
0x00007ffff7faadf0  resbuf
0x00007ffff7faae08  buffer_size
0x00007ffff7faae20  resbuf.10662
0x00007ffff7faae38  buffer_size.10661
0x00007ffff7faae68  start_fct
0x00007ffff7faae70  startp
0x00007ffff7faae78  startp_initialized
0x00007ffff7faae80  start_fct.10689
0x00007ffff7faae88  startp.10688
0x00007ffff7faae90  startp_initialized.10687
0x00007ffff7faaea0  cm
0x00007ffff7faaf18  start_fct
0x00007ffff7faaf20  startp
0x00007ffff7faaf40  once
0x00007ffff7fab068  crud
0x00007ffff7fab26c  devpts_mounted
0x00007ffff7fab270  have_no_dev_ptmx
0x00007ffff7fa49a0  _sys_siglist
0x00007ffff7fa4bc0  sys_sigabbrev
0x00007ffff7fa89b8  _obstack
(gdb) run Hello
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/vuln Hello
[Inferior 1 (process 5410) exited normally]
(gdb)
```

I Run the program with input hello.

I was able to write the return address and code executed. I have checked the rip register.