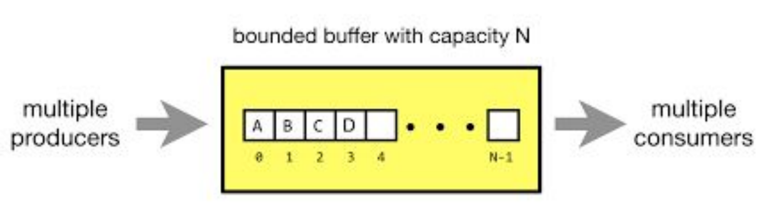
Buffer Overflow Attacks and Defenses (with demo)

Parth Laturia
Devansh Chandak
Rajat Jain
Anish Deshpande
Sunil Meena

Introduction

- A **buffer** is a region of physical memory storage, used to to store data (usually temporarily). We look at buffers implemented in RAM as part of user programs.
- Buffer overflow vulnerabilities dominate the area of remote network penetration vulnerabilities, where an anonymous Internet user seeks to gain partial or total control of a host.
- A **buffer overflow** occurs when:
  - More data is written to the buffer than it can handle.
  - Data is written out of bounds because of errors in the length condition. (**Off-by-One**)
  - Overwriting dynamic memory allocation linkages.
  - Return addresses of subroutines are changed.
- Programs typically written in C or C++ language are inherently susceptible to buffer overflow attacks, in which methods are often passed pointers or arrays as parameters without any indication of their size, and such malpractices are exploited later.

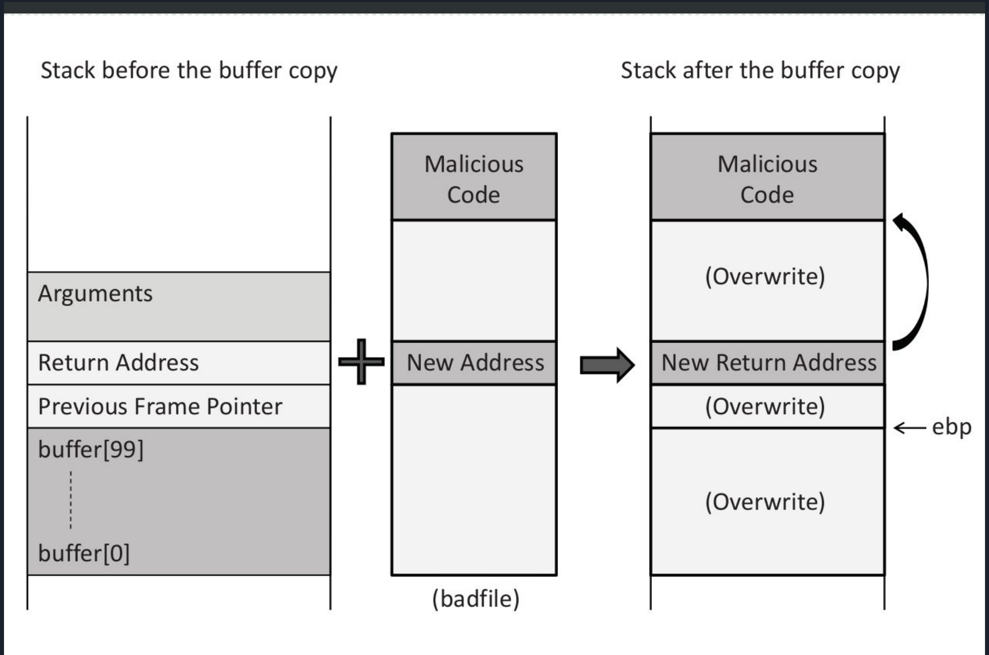


- We survey the various types of buffer overflow vulnerabilities and attacks, and survey the various defensive measures that mitigate buffer overflow vulnerabilities

Types of Attacks

Stack Based Buffer Overflow

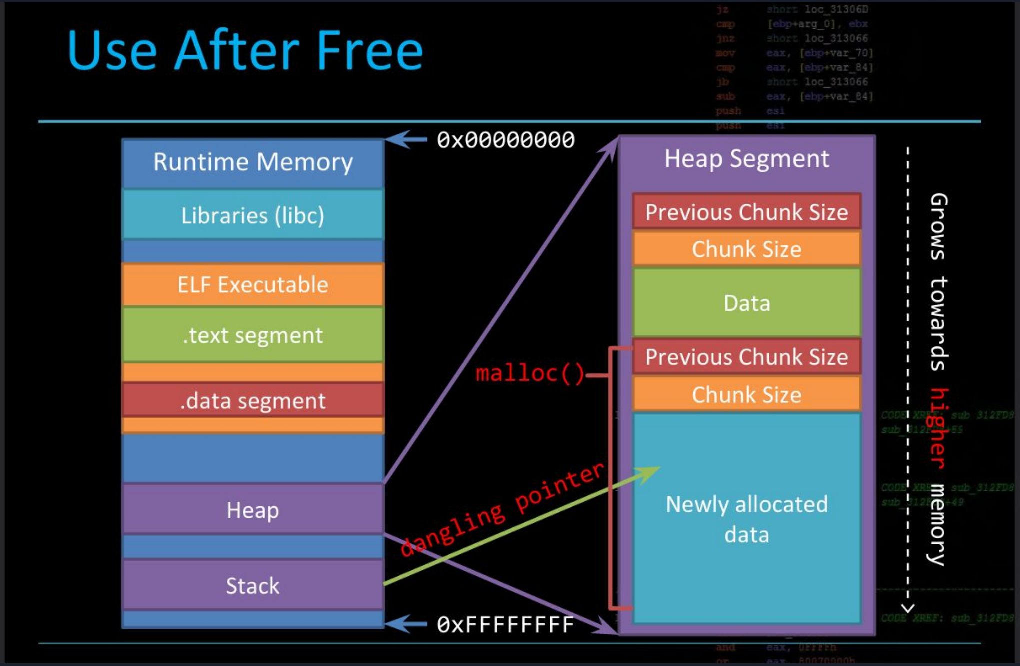
- The stack stores various frames, variables, pointers, addresses etc.
- Data → Copied to target buffer.
- Overflow consequence → Critical values like **return addresses** are modified.
- Exploits → We can cause the program to **crash** or **run other malicious code.**
- Defenses →
  - Hardware Architecture (**NX bit** to separate code from data)
  - OS loader program implements **Address Space Layout Randomisation**
  - Compiler uses **StackGuard**, a way to gauge whether a return address has been overwritten.



Types of Attacks

Heap Based Buffer Overflow

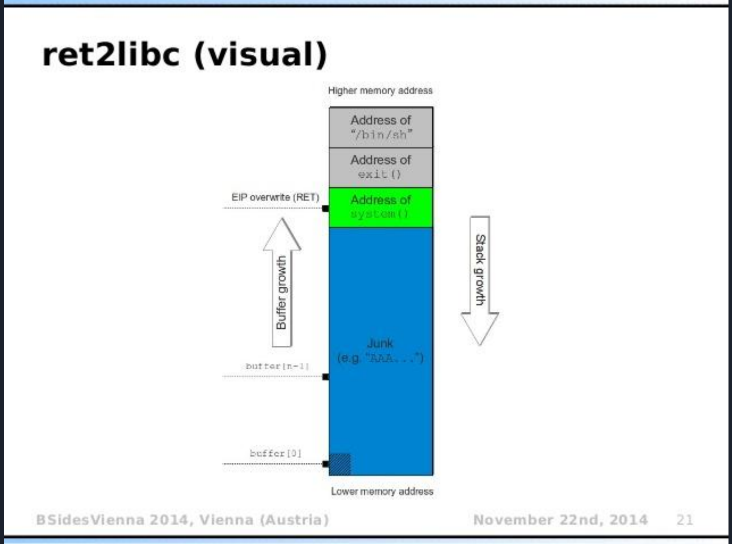
- Heap memory is dynamically allocated, contains program data.
- Overflow → we overwrite:
  - Dynamic memory allocation linkages
  - Program function pointers
- Exploit → **Use After Free**
  - A **dangling pointer** references already freed data, may contain anything now!
  - It’s simply pointer mismanagement. We can corrupt the memory- obtain the EIP!
- Defenses →
  - Safer unlinking (pointer consistency, etc)
  - Heap Entry Header Cookies
  - Heap base address randomization, function pointer encoding, ASLR etc.



Eg: IOS Jailbreaking corrupts heap memory to replace the kernel

Types of Attacks

Return-to-LibC



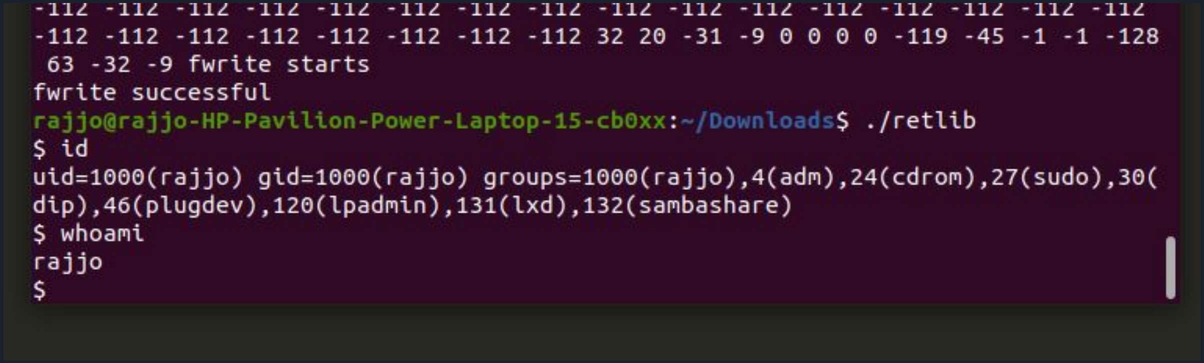
ret2libc (visual)

BSidesVienna 2014, Vienna (Austria) November 22nd, 2014 21

- Invokes a buffer overflow →
  - a subroutine return address on the call stack is replaced.
  - A pre-existing subroutine address (on the executable) takes its place.
  - The **No-Execute bit** is bypassed
- Advantage → No code need be injected
- Disadvantage → Only other functions on the executable can be called.

- Defense →
  - ASLR makes this attack unlikely on 64-bit systems.
  - 32-bit systems have 1 bits for randomization. So a brute force return from LibC may be used.

Eg: Spawning a shell as shown



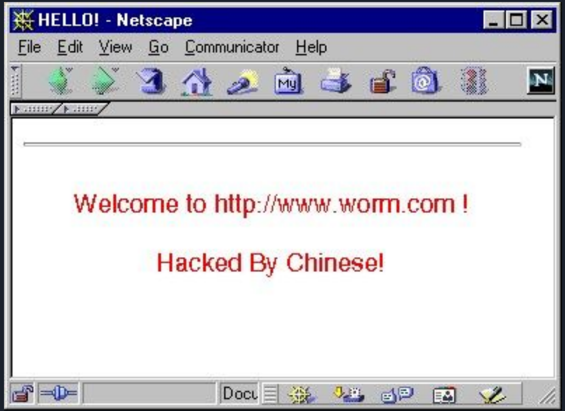
```
-112 -112 -112 -112 -112 -112 -112 -112 -112 -112 -112 -112 -112 -112 -112 -112
-112 -112 -112 -112 -112 -112 -112 -112 32 20 -31 -9 0 0 0 0 -119 -45 -1 -1 -128
 63 -32 -9 fwrite starts
fwrite successful
rajjo@rajjo-HP-Pavilion-Power-Laptop-15-cb0xx:~/Downloads$ ./retlib
$ id
uid=1000(rajjo) gid=1000(rajjo) groups=1000(rajjo),4(adm),24(cdrom),27(sudo),30(
dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
$ whoami
rajjo
$
```

Case Study

Code Red: A worm which exploited buffer overflow: It spread itself on Microsoft IIS servers. Then , it launched **DoS** attack by overflowing a vulnerable buffer with the string “NNN…”., followed by the code which when executed was the payload of the worm.

→ It targeted systems with fixed IP addresses.
→ It defaced affected websites to display a fake message.
→ Apache HTTP servers (a cross-platform web server software, which was popular at the time), had access logs which were lists of requests for a file or some data from a website.
The following was frequently seen in the logs:

```
GET/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6858%uc
bd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00
c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a  HTTP/1.0
```



Welcome to http://www.worm.com !

Hacked By Chinese!

Conclusion:

We have presented a detailed categorization and analysis of buffer overflow vulnerabilities, attacks, and defenses. Buffer overflows constitute a majority of security vulnerability issues, and a substantial majority of remote penetration security vulnerability issues. The results of this analysis of both stack and heap-based attacks show that ASLR, a combination of the StackGuard defense and the non-executable stack defense, header cookies (for heaps) serve to defeat many contemporary buffer overflow attacks.

There have been several attacks like the CodeRed, Slammer worms which cause huge damage (CodeRed costed $2.6 billion). The experience of such attacks demonstrates that widespread vulnerabilities in Internet hosts can be exploited quickly and dramatically, and that techniques other than host patching are needed to mitigate internet worms.

Since high-speed worms are no longer simply a theoretical threat, worm defenses need to be automatic; there is no conceivable way for system administrators to respond to threats of this speed