Computer Security
04r. Assignment 3 review

Paul Krzyzanowski
Rutgers University
Spring 2017
Question 1

How does an off-by-one overflow that only allows changing a byte of the base pointer (ebp or, on 64-bit systems, rbp_) enable malicious code execution?

• Prior to return, a function pops the stack: popq %rbp
• This restores the value of the previous frame pointer (%rbp)
• It then returns to the function that calls it
• Any references to local variables (i.e., those that live on the stack) are expressed as offsets to the frame pointer
  – If we changed the frame pointer to something in the buffer, we can change local variables
  – if the function calls another function then, on function entry, the stack pointer is set to the the current base pointer
Question 2

What damage can heap-based buffer overflows do if they cannot change the instruction pointer?

A buffer overflow a buffer and modify the value of adjacent data, which can be function pointers (which may allow code execution exploits!), or other variables such as numbers or strings (e.g., a file name).
Question 3

How can a printf format string read arbitrary stack data?

Each time you specify something to output (with a % directive, such as %x), the `printf` function assumes there is a corresponding variable on the stack. If there isn’t it just reads whatever is on the stack in that position.

```c
printf("(1) %x\n(2) %x\n(3) %x\n(4) %x\n");
```

Will print the 4 items on the top of the stack.
Question 4

How does ASLR make buffer overflow attacks more difficult?

A buffer overflow attack requires replacing the return address on the stack with a well-known return address to the exploit code (in the buffer on the stack).

If the address of the stack changes each time the program is run, the attacker will not know what to place in the overflowed buffer that will be a valid address.
• We cannot cover everything in under an hour
• Let’s go over some of the highlights
Introduction
Computer security

• What computer security addresses:

  – **Confidentiality**
    • Allow only authorized users to access data & resources
    • **Privacy**: limit what information will be shared with others
    • Privacy is a reason for confidentiality

  – **Integrity**: trustworthiness of data & resources
    • **Data integrity**: data hasn’t been corrupted
    • **Origin integrity/destination integrity**: validate who is sending and who is receiving
    • **System integrity**: system works properly and has not been subverted

  – **Availability**
    • The system is available for use and performs properly
No easy answers

• Security is hard
  – Software is incredibly complex
  – Systems are complex: cloud + local; 3rd party components; multiple admins

• If it was easy, we wouldn’t have massive security breaches year after year
  – No magic solutions
Security goals

• **Prevention**: prevent attackers from violating security policy
  – Implement mechanisms that users cannot override
  – *Example: ask for a password*

• **Detection**: detect & report attacks
  – Important when prevention fails
  – Indicates & identifies weaknesses with prevention
  – Also: detect attacks even if prevention is successful

• **Recovery**: stop the attack, repair damage
  – … Or continue to function correctly even if attack succeeds
  – Forensics: identify what happened so you can fix it
  – *Example: restoration from backups*
Policies & Mechanisms

**Policy**: description of what is or is not allowed
- E.g., users must have a password

**Mechanisms**: implement and enforce policies
- E.g., password entry & authentication
Definitions

• Vulnerability
  – A weakness in the implementation or operation of a system
  – Bugs, bad configuration, lack of access controls

• Attack
  – A means of exploiting a vulnerability
  – E.g., buffer overflow, social engineering

• Threat
  – An adversary that is capable of attacking

• Trusted Computing Base (TCB)
  – All hardware & software of a computing system critical to its security
    • Example: operating system & system software
    • If the TCB is compromised, you have no assurance that any aspect of the system is secure
Threat categories

• Disclosure: Unauthorized access to data
  – *Snooping (wiretapping)*

• Deception: Acceptance of false data
  – *Injection of data, modification of data, denial of receipt*

• Disruption: Interruption or prevention of correct operation
  – *Modification of the system, denial of service, delays*

• Usurpation: Unauthorized control of some part of a system
  – *Modification, spoofing an identity, escalation of privileges*
Protection & Access Control

Protection
- The mechanism that provides and enforces controlled access of resources to processes
- A protection mechanism *enforces* security policies

Access control
- Ensure that authorized users can do what they are permitted to do … *and no more*
The Operating System

• Protect the OS from applications
• Make sure it stays in control
• Basic OS mechanisms
  – Hardware timer – periodically gives control to the OS
  – Scheduler – decides which process gets to run
  – Memory Management Unit (MMU) – provides private memory spaces and memory protection (read/write/execute access)
  – User & kernel mode execution – only the kernel can access privileged instructions
Access control: subjects & objects

- **Subject**: the thing that needs to access resources
  - Often the user
- **Object**: the resource the subject may access
- **Access control**: defines how subjects may access objects
Unix (POSIX) access control

• Each object (file, device) has
  – An owner and a group
  – Read, write, and/or execute permissions for the owner, group, and other (everyone else)

• Each subject (user) has
  – One user ID
  – Membership in one or more groups

• For directories
  – Execute permission = search permission
  – Write access = you can create/delete files or directories within that directory
POSIX file operations

• **chmod**: set file permissions

• **chown**: change file ownership of a file

• **chgrp**: change group ownership of a file

• **setuid**: permission bit that causes an executable file to run with the ID of the file owner, not the user who is executing the file
  
  – **WARNING!** Many set UID programs run as root (administrator) and are attractive targets. If you can take control of that program then you have administrative privileges
Principle of least privilege

• Principle of least privilege
  – At each abstraction layer, every element (user, process, function) should be able to access *only* the resources necessary to perform its task

• Privilege separation
  – Divide a program into multiple parts: high & low privilege components
Access control matrix

- Access control lists vs. capability lists

<table>
<thead>
<tr>
<th>domains of protection (subjects)</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F₀</td>
<td>F₁</td>
<td>Printer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₀</td>
<td>read</td>
<td>read-write</td>
<td>print</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₁</td>
<td>read-write-execute</td>
<td>read</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₂</td>
<td>read-execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₃</td>
<td>read</td>
<td>print</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₄</td>
<td></td>
<td>print</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Access control list*

<table>
<thead>
<tr>
<th>objects</th>
<th>F₀</th>
<th>F₁</th>
<th>Printer</th>
</tr>
</thead>
<tbody>
<tr>
<td>D₀</td>
<td>read</td>
<td>read-write</td>
<td>print</td>
</tr>
<tr>
<td>D₁</td>
<td>read-write-execute</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>D₂</td>
<td>read-execute</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>D₃</td>
<td>read</td>
<td>print</td>
<td></td>
</tr>
<tr>
<td>D₄</td>
<td>print</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Capability list*
DAC vs. MAC

• DAC = Discretionary Access Control
  – Users get to set access permissions

• MAC = Mandatory Access Control
  – Administrators set access permissions that users cannot overwrite
Multi-Level Security Models

- The **Bell-LaPadula** model is all about **confidentiality**
  - Simple **security** property:
    - You cannot read data from higher clearance levels than you are
  - **Star *-property**:
    - You cannot create data that is a lower clearance level than you are
  - Discretionary security property
    - Users can control access with ACLs only *after* MAC is enforced

- The **Biba** model is similar but is all about **integrity**
  - Simple **integrity** property:
    - You cannot read an object from a lower integrity level than you are
    - *Example*: A process will not read a system configuration file created by a lower-integrity-level process
  - **Star *-property**:
    - You cannot write to an object of a higher integrity level than you are
    - *Example*: A web browser may not write a system configuration file
Other MAC models

• **Type Enforcement (TE) Model**
  – Another access control matrix that gets checked first
  – Subjects assigned to domains; objects assigned to types
  – Matrix defines domain-domain and domain-type transitions

• **Role-Based Access Control (RBAC) model**
  – Users are assigned roles (job functions)
  – Access permissions are granted to roles
Multilateral Security

• In addition to levels, a level may have **compartments**
  – You can only access resources if you have been granted access to that compartment
  – E.g., {Top Secret, Elvis}
    • can access {Top Secret}, {Secret, Elvis}, {Secret}
    • Cannot access {Top Secret, UFO}, {Secret, UFO}

• **Lattice model**
  – Directed graph that defines access rights among clearance levels and compartment labels
Chinese Wall Model

• Defines conflict classes: groups of competing companies
  – Designed for businesses where employees have to avoid conflict of interest

• Basic rule
  – A subject can access objects from a company as long as it never accessed objects from competing companies.
Program Hijacking
Stack-based buffer overflow

• Buffer limits not checked
  – Often because unsafe functions like strcpy, strcat, and sprintf are used

• Overflow overwrites frame pointer & stack pointer

• If the stack pointer is changed, the return address is changed
  – Write code into the buffer
  – Overflow the buffer to set the return address
  – When the function returns, it branches to the new code
Off-by-one Buffer Overflows

• An off-by-one stack overflow can only modify one byte of the top of the stack, which holds the frame pointer
  – When a function returns, the modified frame pointer becomes the reference point for all local variables
  – It also becomes the new stack pointer when a new function is called
  – (see homework assignment)
Heap & text segment overflows

• A buffer overflow can overwrite adjacent variables that are allocated in higher memory
  – The program will use these modified variables
Printf format attacks

If an attacker can change the printf format string

• Read the stack
  – Read any address on the stack (using %x, for example)
  – If you don’t supply arguments, printf will match %x with the next item on the stack

• Modify memory
  – Use ”%x” to set where we write in memory: each %x skips one word on the stack
  – Use “%.Nx” to generate N bytes of output – this allows you to set the value you will write
  – Use %n to write the value – it prints the # of bytes output so far
Defenses

• Data Execute Protection (DEP)
  – Operating system turns off execute permission for stack and heap memory
  – Attacks:
    • return-to-libc: overflow a return address to a desired point in the C library
    • Return-Oriented-Programming (ROP): overflow a stack of return addresses to various points in libraries or the program – the return from one function takes you to the next entry point

• Address Space Layout Randomization (ASLR)
  – Load programs and libraries into different memory locations so addresses are different each time

• Stack Canaries
  – Compiler places a random # on the top of the stack and checks it before returning from a function
SQL Injection Attacks

• If user input becomes part of a SQL query, it can change the type of query – or add additional commands

```sql
SELECT * from logininfo WHERE username = paul AND password = 'abcde'
```

```sql
SELECT * from logininfo WHERE username = paul AND password = '' OR 1=1 -- ;'
```

– Validate all input!
– Safest prevention = use parameterized queries – don’t make user input part of the command
Shell injection attacks

- **IFS** – internal field separator: change the definition of how the shell parses arguments (it doesn’t have to be whitespace)

- Use of `system()` and `popen()` in programs
  - These invoke the shell. Same risk as SQL injection if user input is part of the command

- **PATH** variable: change the order in which the shell looks for programs

- **LD_PRELOAD**: preload libraries, possibly overriding functions that the program uses with your own

- **LD_LIBRARY_PATH**: similar attack – tell the OS where to look for libraries
App-level name parsing

• Parsing pathnames to make sure a user-supplied name stays within a subdirectory can be tricky
  – http://poopybrain.com/../../etc/passwd

• Escaped unicode characters make it harder
  – Single-byte characters have multi-byte equivalents: "/" = 0x2f = 0xc0af

• Homograph Attacks
  – Not an injection attack but a social engineering one – confuses users
  – Similar characters may be represented in different languages or different symbols in Unicode: you can often write the same text using different characters
TOCTTOU Attack

• Time Of Check To Time Of Use
  – If you check the condition and then do something, you may introduce a race condition
  – An attacker may change something after you check the condition but before you do the operation
    • Example: change a link to a user-readable file to a privileged file
RASQ

• The more a program interacts with its environment (users, files, networks), the more opportunities there are for attack

• Relative Attack Surface Quotient
  – A metric Microsoft created to estimate how vulnerable a system might be
  – Each type of attack gets a “bias” – a risk assessment
  – Roughly, sum up all possible interactions with each interaction multiplied by its bias
App confinement

• **chroot**: change root directory for a process & its children
  – If an attacker becomes root, he may be able to escape by creating a device
    file that gives access to the disk or to memory

• **FreeBSD Jails**
  – Same namespace protection like chroot
  – But you can take power away from root for processes in the jail
    • No ability to create devices, raw sockets, mounting filesystems
  – Way more secure
App confinement

• **Linux namespaces**
  – Provide a private namespace for directory structure, network, process ID, user/group IDs, IPC, hostname

• **Linux capabilities**
  – Selectively take away power if a process becomes root.
  – Disallow file owner changes, permission changes, sending signals, creating raw sockets, changing root, etc.

• **Linux control groups**
  – Limit how much resources a process can use (CPU, memory, files, network)
The end