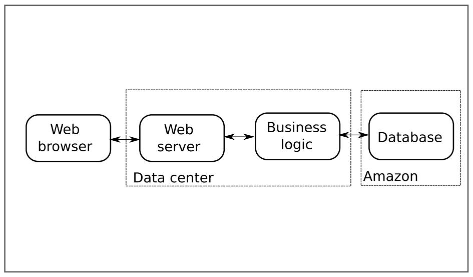
CSci 4271W
Development of Secure Software Systems
Day 5: Memory safety attacks

Stephen McCamant

University of Minnesota, Computer Science & Engineering

Outline

Trust boundaries example



Attacks come with data flows

- Principle: attacks propagate along data flows
- Therefore, enumerate flows to enumerate attacks
  - A more specific prompt, but does not eliminate the need for imagination
  - Other half is types of attacks, see next slide

STRIDE threat taxonomy

- Spoofing (vs authentication)
- Tampering (vs integrity)
- Repudiation (vs. non-repdiation)
- Information disclosure (vs. confidentiality)
- Denial of service (vs. availability)
- Elevation of privilege (vs. authortization)

What to do about threats

- Mitigate: add a defense, which may not be complete
- Eliminate: such as by removing functionality
- Transfer functionality: let someone else handle it
- Transfer risk: convince another to bear the cost
- Accept risk: decide that the risk (probability $\cdot$ loss) is sufficiently low

Spoofing threat examples

- Using someone else's account
- Making a program use the wrong file
- False address on network traffic

Tampering threat examples

- Modifying an important file
- Rearranging directory structure
- Changing contents of network packets

Repudiation threat examples

- Performing an important action without logging
- Destroying existing logs
- Add fake events to make real events hard to find or not credible

Info. disclosure threat examples

- Eavesdropping on network traffic
- Reading sensitive files
- Learning sensitive information from meta-data

DoS threat examples

- Flood network link with bogus traffic
- Make a server use up available memory
- Make many well-formed but non-productive interactions

Elevation of privilege threat examples

- Cause data to be interpreted as code
- Change process to run as root/administrator
- Convince privileged process to run attacker's code

Outline

Threat modeling, cont'd

Shellcode techniques

Binary-level GDB commands and demo

In-person lab logistics reminders

Project 1 bcimgview intro and demo

Exploiting other vulnerabilities

Basic definition

- Shellcode: attacker supplied instructions implementing malicious functionality
- Name comes from example of starting a shell
- Often requires attention to machine-language encoding

Classic execve `/bin/sh`

- `execve(fname, argv, envp)` system call
- Specialized syscall calling conventions
- Omit unneeded arguments
- Doable in under 25 bytes for Linux/x86

Avoiding zero bytes

- Common requirement for shellcode in C string
- Analogy: broken 0 key on keyboard
- May occur in other parts of encoding as well

More restrictions

- No newlines
- Only printable characters
- Only alphanumeric characters
- "English Shellcode" (CCS'09)

Transformations

- Fold case, escapes, Latin1 to Unicode, etc.
- Invariant: unchanged by transformation
- Pre-image: becomes shellcode only after transformation

Multi-stage approach

- Initially executable portion unpacks rest from another format
- Improves efficiency in restricted environments
- But self-modifying code has pitfalls

NOP sleds

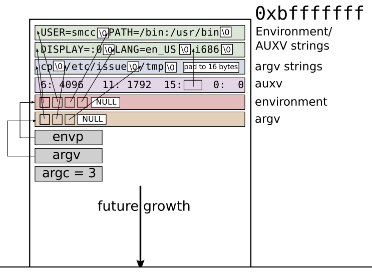
- Goal: make the shellcode an easier target to hit
- Long sequence of no-op instructions, real shellcode at the end
  - x86: 0x90 0x90 0x90 0x90 0x90 ... shellcode

Where to put shellcode?

- In overflowed buffer, if big enough
- Anywhere else you can get it
  - Nice to have: predictable location
- Convenient choice of Unix local exploits:

Where to put shellcode?

Environment variables



0xbfffffff
USER=smcc \0 PATH=/bin:/usr/bin \0
DISPLAY=:0 \0 LANG=en_US \0 i686 \0
cp \0 /etc/issue \0 /tmp \0 pad to 16 bytes
6: 4096 11: 1792 15: 0: 0
NULL
NULL
envp
argv
argc = 3
future growth
Environment/ AUXV strings
argv strings
auxv
environment
argv

Code reuse

- If can't get your own shellcode, use existing code
- Classic example: `system` implementation in C library
  - "Return to libc" attack
- More variations on this later

Outline

GDB commands

- (See separate slides)

Outline

In-person and online labs this week

- Starting tomorrow, in-person labs will be available
- In 1-250 Keller, same time as online labs
- Online labs also still available, no name split

Electronic collaboration in-person

- Because of social distancing, in-person labs still need screen sharing
  - Zoom room posted for staff interaction
- Any groups you want, in one area of the lab
  - But still not too close together
  - Collaboration may use `tmate`, Zoom, anything else

Labs safety reminders

- Mask or other face covering is required
- Stay 6 feet apart, don't come if you're sick
- Self-service disinfecting wipes
  - You might also consider gloves
- Safety rules are a compromise, no pressure to attend if it's not worth it

Outline

Project 1 aspects

- Threat modeling (from now)
- Code auditing
- Attack creation
- Security report
- Revision and fixes (second submission)

Project 1 scenario

- Benign but buggy image viewer
- Key threat class: opening untrusted images
  - Imagine web or email downloads
  - Similar to many historical problems

Project 1 logistics

- Individual project
- Submission deadlines finalized with code release
  - Planned for Thursday

`bcimgview` demo

- Simple image viewer functionality

Outline

Threat modeling, cont'd

Shellcode techniques

Binary-level GDB commands and demo

In-person lab logistics reminders

Project 1 bcimgview intro and demo

Exploiting other vulnerabilities

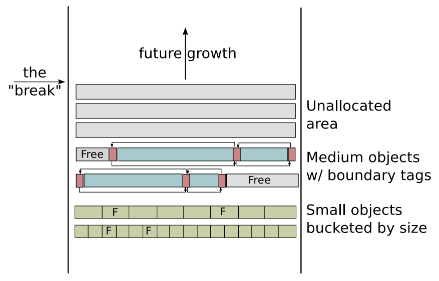
Non-control data overwrite

- Overwrite other security-sensitive data
- No change to program control flow
- Set user ID to 0, set permissions to all, etc.

Heap meta-data

- Boundary tags similar to doubly-linked list
- Overwritten on heap overflow
- Arbitrary write triggered on `free`
- Simple version stopped by sanity checks

Heap meta-data


future growth
the "break"
Unallocated area
Free
Free
Medium objects w/ boundary tags
F
F
F
F
Small objects bucketed by size

Use after free

- Write to new object overwrites old, or vice-versa
- Key issue is what heap object is reused for
- Influence by controlling other heap operations

Integer overflows

- Easiest to use: overflow in small (8-, 16-bit) value, or only overflowed value used
- 2GB write in 100 byte buffer
  - Find some other way to make it stop
- Arbitrary single overwrite
  - Use math to figure out overflowing value

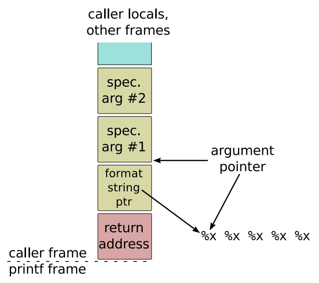
Null pointer dereference

- Add offset to make a predictable pointer
  - On Windows, interesting address start low
- Allocate data on the zero page
  - Most common in user-space to kernel attacks
  - Read more dangerous than a write

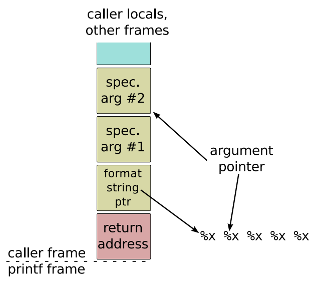
Format string attack

- Attacker-controlled format: little interpreter
- Step one: add extra integer specifiers, dump stack
  - Already useful for information disclosure

Format string attack layout



Format string attack layout



Format string attack: overwrite

- `%n` specifier: store number of chars written so far to pointer arg
- Advance format arg pointer to other attacker-controlled data
- Control number of chars written with padding
- On x86, can use unaligned stores to create pointer