Auditing is...

Reading code to find security bugs
Threat modeling comes first, tells you what kinds of bugs you're looking for
Bug fixing comes next (might be someone else's job)

Tiers and triage

You might not have time to do a complete job, so use auditing time strategically
Which bugs are most likely, and easiest to find?
Triage into definitely safe, definitively unsafe, hard to tell
Hard to tell might be improved even if safe

Threat model and taint

Vulnerability depends on what an attacker might control
Another word for attacker-controlled is “tainted”
Threat model is the best source of tainting information
  Of course, can always be conservative

Where to look for problems

If you can't read all the code carefully, search for indicators of common danger spots
  For format strings, look for printf
  For buffer overflows, look at buffers and copying functions

Ideal: proof

Given enough time, for each dangerous spot, be able to convince someone:
  Proof of safety: reasons why a bug could never happen, could turn into assertions
  Proof of vulnerability: example of tainted input that causes a crash
Integer overflow to buffer overflow

- One common pattern: overflow causes an allocation to be too small
- In machine integers, multiplication doesn't always make a value larger

Overflow example questions

1. What's a value of num_objs that would trigger an overflow?
   - Think back to 2021 on how multiplication overflows
2. Why is the p->ident check relevant to exploitability?

Overflow example

```c
struct obj { short ident, x, y, z; long b; double c;};
struct obj *read_objs(int num_objs) {
    unsigned int size = num_objs*(unsigned)sizeof(obj);
    struct obj *objs = malloc(size);
    struct obj *p = objs;
    for (i = 0; i < num_objs; i++) {
        fread(p, sizeof(struct obj), 1, stdin);
        if (p->ident == 0x4442) return 0;
        /* ... */ p++; }
    return objs; }
```

Overflow in multiplication

- Struct size is 24 bytes, or 11000 (16+8) in binary
- \(24 \times (x \ll 4) + (x \ll 3)\)
- Top three bits fall off
- Interpreted as unsigned after multiplication, and by malloc

Vulnerability condition

- Overflow happens if we write more than we allocated
- Allocation won't fail on this 64-bit machine (4GB available)
- \(24 \cdot \max(x,0) > (24 \cdot x) \mod 2^{32}\)
- Safe if:
  - Count interpreted as negative
  - Overflow does not occur

Computing overflow values

- One approach: input must be bigger than \(2^{32}/24\) to overflow
- No-calculator approach: pick numbers where multiplication is easy
  - Compare in decimal: 1001 \cdot 42 = 42042
Outline
- Code auditing
- Integer overflow discussion
- Threat modeling

Why threat modeling?
- Think about and describe the security design of your system
- Enumerate possible threats
- Guide effort spent on combating threats
- Communicate to customers and other developers

Why a structured approach?
- Goal is to avoid missing a threat
- Enumerate vectors for threats
- Enumerate kinds of threats per vector
- Convince readers of the model’s completeness

Data-flow modeling
- Break down software into smaller modules
  - Modules drawn with rounded rectangles
  - More detail is better, within reason
- Show data flows among modules and external parties
  - Rectangles for external parties
  - Most data flows will be bi-directional

Trust boundaries
- A trust boundary groups components with the same privilege, which therefore trust each other
  - Drawn as labeled dotted box
  - Attacks usually don’t originate within a trust group
- The boundary also corresponds to an attack surface

Data flow example

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-repudiation)
- Information disclosure (vs. confidentiality)
- Denial of service (vs. availability)
- Elevation of privilege (vs. authorization)

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