Goals for Today

- TOCTOU
- Integer Overflow, Casting
- Randomness
- Timing Attacks
- Defensive Mechanisms
- Software Development Design Principles

TOCTOU

- TOCTOU == Time of Check to Time of Use

```c
int openfile(char *path) {
  struct stat s;
  if (stat(path, &s) < 0)
    return -1;
  if (!S_ISRREG(s.st_mode)) {
    error("only allowed to regular files");
    return -1;
  }
  return open(path, O_RDONLY);
}
```

Goal: Open only regular files (not symlink, etc)

Attacker can change meaning of path between `stat` and `open` (and access files he or she shouldn't)

Integer Overflow and Implicit Cast

```c
char buf[80];
void vulnerable() {
  int len = read_int_from_network();
  char *p = read_string_from_network();
  if (len > sizeof buf) {
    error("length too large, nice try!");
    return;
  }
  memcpy(buf, p, len);
}

void *memcpy(void *dst, const void * src, size_t n);
typedef unsigned int size_t;
```

- If len is negative, may copy huge amounts of input into buf
Integer Overflow and Implicit Cast

```
size_t len = read_int_from_network();
char *buf;
buf = malloc(len+5);
read(fd, buf, len);
```

- What if `len` is large (e.g., `len = 0xFFFFFFFF`)?
- Then `len + 5 = 4` (on many platforms)
- Result: Allocate a 4-byte buffer, then read a lot of data into that buffer.

(from www-inst.eecs.berkeley.edu—implflaws.pdf)

Randomness issues

- Many applications (especially security ones) require randomness
- "Obvious" uses:
  - Generate secret cryptographic keys
  - Generate random initialization vectors for encryption
- Other "non-obvious" uses:
  - Generate passwords for new users
  - Shuffle the order of votes (in an electronic voting machine)
  - Shuffle cards (for an online gambling site)

C's rand() Function

```
unsigned long next = 1;
/* rand: return pseudo-random integer on 0..32767 */
int rand(void) {
    next = next * 1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}
/* srand: set seed for rand() */
void srand(unsigned int seed) {
    next = seed;
}
```

- Problem: don't use rand() for security-critical applications!
  - Given a few sample outputs, you can predict subsequent ones

Dr. Dobbs' Portal

Windows/.NET

Randomness and the Netscape Browser

How secure is the World Wide Web?

Ian Goldberg and David Wagner

No one was more surprised than Netscape Communications when a pair of computer-scientists bugged the Netscape encryption scheme. Ian and David describe how they attacked the popular Web browser and what they found next.
Problems in Practice

- One institution used (something like) `rand()` to generate passwords for new users
  - Given your password, you could predict the passwords of other users
  - Random number generator improperly seeded
  - Possible to trivially break into machines that rely upon Kerberos for authentication
- Online gambling websites
  - Random numbers to shuffle cards
  - Real money at stake
  - But what if poor choice of random numbers?
Obtaining Pseudorandom Numbers

- For security applications, want “cryptographically secure pseudorandom numbers”
- Libraries include:
  - OpenSSL
  - CryptoAPI (Microsoft)
- Linux:
  - `/dev/random`
  - `/dev/urandom`
- Internally:
  - Pool from multiple sources (interrupt timers, keyboard, …)
  - Physical sources (radioactive decay, …)

Timing Attacks

- Assume there are no “typical” bugs in the software
  - No buffer overflow bugs
  - No format string vulnerabilities
  - Good choice of randomness
  - Good design
- The software may still be vulnerable to timing attacks
  - Software exhibits input-dependent timings
- Complex and hard to fully protect against

Password Checker

- Functional requirements
  - `PwdCheck(RealPwd, CandidatePwd)` should:
    - Return `TRUE` if `RealPwd` matches `CandidatePwd`
    - Return `FALSE` otherwise
  - `RealPwd` and `CandidatePwd` are both 8 characters long
- Implementation (like TENEX system)
  ```java
  PwdCheck(RealPwd, CandidatePwd) // both 8 chars
  for i = 1 to 8 do
    if RealPwd[i] != CandidatePwd[i] then
      return FALSE
  return TRUE
  ```
- Clearly meets functional description
Attacker Model

```plaintext
PwdCheck(RealPwd, CandidatePwd) // both 8 chars
for i = 1 to 8 do
    if RealPwd[i] != CandidatePwd[i] then
        return FALSE
return TRUE
```

- Attacker can guess CandidatePwds through some standard interface
- Naive: Try all 256^8 = 18,446,744,073,709,551,616 possibilities
- Better: Time how long it takes to reject a CandidatePassword. Then try all possibilities for first character, then second, then third, ...
  - Total tries: 256*8 = 2048

Other Examples

- Plenty of other examples of timings attacks
  - AES cache misses
    - AES is the "Advanced Encryption Standard"
    - It is used in SSH, SSL, SSH, PGP, ...
  - RSA exponentiation time
    - RSA is a famous public-key encryption scheme
    - It's also used in many cryptographic protocols and products