CSE 484 / CSE M 584: Computer Security and Privacy

Software Security (Misc)

Fall 2017

Franziska (Franzi) Roesner
franzi@cs.washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
Last Words on Buffer Overflows...
ASLR Issues

• NOP slides and heap spraying to increase likelihood for custom code (e.g., on heap)
• Brute force attacks or memory disclosures to map out memory on the fly
  – Disclosing a single address can reveal the location of all code within a library
Other Possible Solutions

• Use safe programming languages, e.g., Java
  – What about legacy C code?
  – (Though Java doesn’t magically fix all security issues 😃)
• Static analysis of source code to find overflows
• Dynamic testing: “fuzzing”
• LibSafe: dynamically loaded library that intercepts calls to unsafe C functions and checks that there’s enough space before doing copies
  – Also doesn’t prevent everything
Beyond Buffer Overflows...
Another Type of Vulnerability

• Consider this code:

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

• **Goal:** Open only regular files (not symlink, etc)
• **What can go wrong?**
TOCTOU (Race Condition)

- 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_ISREG(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)
Another Type of Vulnerability

• Consider this code:

```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;
```
Implicit Cast

• Consider this code:

```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`. 
Another Example

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

(from [www-inst.eecs.berkeley.edu—implflaws.pdf](https://www-inst.eecs.berkeley.edu—implflaws.pdf))
Integer Overflow

- 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)
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)

```plaintext
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

• 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 CandidatePasswd. Then try all possibilities for first character, then second, then third, ....
  
  – Total tries: $256^8 = 2048$

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

• Plenty of other examples of timings attacks
  – AES cache misses
    • AES is the “Advanced Encryption Standard”
    • It is used in SSH, SSL, IPsec, PGP, ...
  – RSA exponentiation time
    • RSA is a famous public-key encryption scheme
    • It’s also used in many cryptographic protocols and products
Software Security: So what do we do?
Fuzz Testing

• Generate “random” inputs to program
  – Sometimes conforming to input structures (file formats, etc.)

• See if program crashes
  – If crashes, found a bug
  – Bug may be exploitable

• Surprisingly effective

• Now standard part of development lifecycle
General Principles

• Check inputs
Shellshock

• Check inputs: not just to prevent buffer overflows

• Example: Shellshock (September 2014)
  – Vulnerable servers processed input from web requests
  – Passed (user-provided) environment variables (like user agent, cookies...) to CGI scripts
  – Maliciously crafted environment variables exploited a bug in bash to execute arbitrary code

```bash
env x='() { ::}; echo Vulnerable'
bash -c "echo Real Command"
```
General Principles

• Check inputs
• Check all return values
• Least privilege
• Securely clear memory (passwords, keys, etc.)
• Failsafe defaults
• Defense in depth
  – Also: prevent, detect, respond

• NOT: security through obscurity
General Principles

• Reduce size of trusted computing base (TCB)
• Simplicity, modularity
  – But: Be careful at interface boundaries!
• Minimize attack surface
• Use vetted component
• Security by design
  – But: tension between security and other goals
• Open design? Open source? Closed source?
  – Different perspectives
Does Open Source Help?

• Different perspectives…

• Happy example:

• Sad example:
  – Heartbleed (2014)
    • Vulnerability in OpenSSL that allowed attackers to read arbitrary memory from vulnerable servers (including private keys)
http://xkcd.com/1354/

SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).

User Meg wants these 6 letters: POTATO. User Ada wants pages about "irl games". Unlocking secure records with master key 5130985733435444. Marge (chrome user) sends this message: "HTTP..."
Server, are you still there? If so, reply "BIRD" (4 letters).

User Olivia from London wants pages about "bees in car why". Note: Files for IP 375.381.383.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 346 connections open. User Brendan uploaded the file self.png (contents: 834ba962e3eb9ff99b33b6f9).

Hmm...

User Olivia from London wants pages about "bees in car why". Note: Files for IP 375.381.383.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 346 connections open. User Brendan uploaded the file self.png (contents: 834ba962e3eb9ff99b33b6f9).

BIRD
Server, are you still there? If so, reply "HAT" (500 letters).

User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "ColoRaSt". User Nadia requests pages.

HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "ColoRaSt". User Nadia requests pages.

http://xkcd.com/1354/
Vulnerability Analysis and Disclosure

• What do you do if you’ve found a security problem in a real system?

• Say

  – A commercial website?
  – UW grade database?
  – Boeing 787?
  – TSA procedures?