Security Part 3
Programming Securely

• It’s hard!
• Some examples …
Truncated Paths

```c
int GetFile(char *dirpath, char *name) {
    char FullyQualified[1024];
    if (CheckName(dirpath) == BAD) {
        ...
    }
    strncpy(FullyQualified, dirpath, 512);
    strncat(FullyQualified, name, 512);
    return(open(FullyQualified, O_RDWR));
}

GetFile("///////////////////////////...///tmp", vmlinuz);
```
Defense

• It’s not enough to avoid buffer overflow …
• Check for truncation!
Carelessness

```c
char  buf[100];
int  len;

read(fd, &len, sizeof(len));

if (len > 100) {
    fprintf(stderr, "bad length\n");
    exit(1);
}

read(fd, buf, len);
```
A Real-Life Exploit …

• sendmail -d6,50
  – means: set flag 6 to value 50
  – debug option, so why check for min and max?
    - (shouldn’t have been turned on for production version …)
    - (but it was …)

• sendmail -d4294967269,117 -d4294967270,110 -d4294967271,113 changed etc to tmp
  – /etc/sendmail.cf identifies file containing mailer program, which is executed as root
  – /tmp/sendmail.cf supplied by attacker
    - identifies /bin/sh as mailer program
    - attacker gets root shell
What You Don’t Know …

```c
int TrustedServer(int argc, char *argv[]) {
    ...
    printf(argv[1]);
    ...
}
```

```
% TrustedServer "wxyz%n"
```

*from the printf man page:*

`%n` The number of characters written so far is stored into the integer indicated by the int * (or variant) pointer argument. No argument is converted.
Does This Work?

% setenv LD_PRELOAD myversions/libcrypt.so.1
% su
Password:
Principle of Least Privilege

• Perhaps:
  – run process with a minimal security context
    - special account, etc.
  – send it the capabilities it needs
chroot (after)

unix etc home pro dev

root

passwd shadow twd
Quiz 1

Restricting a process to a particular subtree

a) improves security by effectively running the process in a smaller protection domain
b) has little effect on security
c) potentially makes security worse
Quiz 2

After executing `chroot`, “/” refers to the process’s new root directory. Thus “..” is the same as “.” at the process’s root, and the process cannot cd to the “parent” of its root. Also, recall that hard links may not refer to directories.

a) `chroot` effectively limits a process to a subtree

b) There are other system calls one can use to set the process’s root back to the real root

c) There are ways of using `chroot` to set the process’s root closer to the real root
Escape!

chdir("/");
pfd = open ".", O_RDONLY); 
mkdir("Houdini", 0700);
chroot("Houdini");
fchdir(pfd);
for
    (i=0; i<100; i++)
        chdir("..");
chroot(".");
Fixed in BSD

• jail
  – can’t `cd` above root
  – all necessary files for standard environment present below root
  – `ps` doesn’t see processes in other jails
Linux Responds ...

• cgroups
  – group together processes for
    - resource limiting
    - prioritization
    - accounting
    - control

• name-space isolation
  – isolate processes in different name spaces
    - mount points
    - PIDs
    - UIDs
    - etc.
Docker

• Software containers for isolated applications
  – uses Linux features to isolate groups of processes
  – everything needed for execution is in the package
  – standard API for applications
    - also runs on Windows
Back to Windows

• Security history
  – DOS and early Windows
    - no concept of logging in
    - no authorization
    - all programs could do everything
  – later Windows
    - good authentication
    - good authorization with ACLs
    - default ACLs are important
      • few understand how ACLs work …
    - most users run with admin privileges
      • all programs can do everything …
Privileges in Windows

• Properties of accounts
  – administrator ≈ superuser
  – finer breakdown for service applications

• User account control (starting with Vista)
  – accounts with administrator privileges have two access tokens
    - one for normal usage
    - another with elevated rights
Least Privilege

• Easy answer
  – disable privileges
  – works only if the process has any …

• Another answer
  – restricting SIDs
    - limit what a server can do
    - two passes over ACL for access check
      • first: as previously specified
      • second: using only restricting SIDs
Least Privilege for Servers

• Pre-Vista:
  – services ran in local system account
    - all possible privileges
    - successful attackers “owned” system
    - too complicated to give special account to each service

• Vista and beyond
  – services still run in system account
  – per-service SIDs created
    - used in DACLs to indicate just what service needs
    - marked restricting in service token
Example

- Critical System File: allow administrators write
- Print service: administrator SID
- Printer: allow administrators write, allow print-service write
- Restricting print-service SID
Least Privilege for Clients

• Pre Vista
  – no
• Vista and beyond
  – windows integrity mechanism
    - a form of MAC
Print Server

• Client sends request to server
  – print contents of file $X$

• Server acts on request
  – does client have read permission?
    - server may have (on its own) read access, but client does not
    - server might not have read access, but client does
Unix Solution

• Client execs print-server, passing it file name
  – set-uid-root program
  – it (without races!) checks that client has access to file, then prints it
Windows Solution

- Server process started when system is booted
- Clients send it print requests
  - how does client prove to server it has access?
  - how does server prove to OS that client has said ok?
Impersonation

- Client sends server *impersonation token* – subset of its access token
- Server temporarily uses it in place of its own access token
Limitation of Both Approaches

- Client must trust server
  - it has full access to everything client owns!
- Is the example realistic?
  - no
  - but ...
    - password-changing program works this way
    - other examples?