Oregon State University

COLLEGE OF ENGINEERING | School of Electrical Engineering and Computer Science

Software Vulnerabilities - II

Software Vulnerability Categories

- Program Input (Input Checking)
  - e.g., code and data injection
- Program Code (Program Logic Errors)
  - e.g., broken access control, bad random numbers or seeds
- Interacting with Operating systems and other Programs
  - e.g., memory leaks, race conditions, environment variables
- Program Output (Output Checking)
  - e.g., cross site scripting (XSS)

OS Interaction Vulnerabilities

Correct Use of Memory

- Memory Leak
  - Run process out of memory. DoS.
- Free/Allocation errors
  - Heap overflow, can enable arbitrary execution
- Could be solved by
  - Tools to track heap utilization
    - Valgrind
    - Duma

Race Conditions and Shared Memory

- Multiple threads of control accessing a common memory location
  - Subtle (and not so subtle) errors in synchronization are possible
  - Multiple writers
  - Writing while another thread is reading
  - Deadlocks

- Errors vary from invocation to invocation

- Attacker could attempt to trigger a latent threading error

Environment Variables

- Another way for the program to get input
  - **and should be treated as such**

- Generally set up for the user
  - Sysadmin creates a profile for the user that initializes the environment

- Environment variables read by compiled programs and scripted programs

Example Vulnerable Scripts

- Using PATH or IFS environment variables
- Cause script to execute attackers program with privileges granted to script
  - SetUID root scripts would be attractive
- Almost impossible to prevent in some form
  - Though the use of IFS has been restricted in most modern shells

```
#!/bin/bash
user=`echo $1 | sed 's/@.*$//'`
grep $user /var/local/accounts/ipaddrs
```

```
#!/bin/bash
PATH=”/sbin:/bin:/usr/sbin:/usr/bin”
export PATH
user=`echo $1 | sed 's/@.*$//'`
grep $user /var/local/accounts/ipaddrs
```

Example from Stallings and Brown Reference Book

Path Attack On Libraries

- Dynamic libraries are loaded at invocation time

- Loader must search the system to find the libraries needed by the executable
  - LD_LIBRARY_PATH

- Flexibility vs. Attack avenue

Least Privilege

- Ideally run a program with **only as many privileges and access rights as it needs but no more**
  - What's wrong with too much access?

- Root in Unix is not a good example of least privilege

- Web servers and file access
  - What files does the web server process needs to read? Needs to write?

- How long does a program need special privilege?
  - e.g., a low port network service program

- Divide program into sets of processes
  - Move the privilege required elements into smaller, simpler processes

System Calls and Standard Library Functions

- Programs use system calls and standard library functions for common operations
  - and make assumptions about their operation
  - unexpected behavior may be a result of system optimizing access to shared resources
    - by buffering, re-sequencing, modifying requests
  - can conflict with program goals

Secure File Shredder Example

```
/* Generic Overwriting patterns*/
patterns = [10101010, 01010101, 11001100, 00110011, 00000000, 11111111, … ]
open file for writing
for each pattern
        overwrite file contents with pattern
close file
delete file
```

```
/* Generic Overwriting patterns*/
patterns = [10101010, 01010101, 11001100, 00110011, 00000000, 11111111, … ]
open file for writing
for each pattern
        overwrite file contents with pattern
        flush application buffer (to kernel)
        sync kernel/file system buffers with device
close file
delete file
```

Is the problem solved?
What if it is not a magnetic disk, or if it is a journaling file system?

Example by Wietse Venema, IBM

Race Conditions

- Files can be used to synchronize access to OS resources between processes

```
If [ ! –e $file ]
then
   touch $file
else
      echo “You don’t have the lock”
fi
```

- Time of check to time of use (TOCTOU)

Temporary Files

- Many programs create temporary intermediate files
  - – Can create unique names based on process id
  - – How could an attacker leverage this?

```
do {
   filename = tempnam(NULL, “foo”);
   fd = open(filename, O_CREAT | O_EXCL | …., 0600)
   free(filename);
} while (fd == -1);
```

Summary

- There are many interactions with the OS that might impact a programs control flow

- If not done carefully these interaction might be hijacked or might be manipulated to change the control flow of a program or to take control