Secure Software Programming and Vulnerability Analysis

Christopher Kruegel chris@auto.tuwien.ac.at
http://www.auto.tuwien.ac.at/~chris

Automation Systems Group

Race Conditions

Overview

- Parallel execution of tasks
  - multi-process or multi-threaded environment
  - tasks can interact with each other

- Interaction
  - shared memory (or address space)
  - file system
  - signals

- Results of tasks depends on relative timing of events

➢ Indeterministic behavior

Race Conditions

- Race conditions
  - alternative term for indeterministic behavior
  - often a robustness issue
  - but also many important security implications

- Assumption needs to hold for some time for correct behavior, but assumption can be violated

- Time window when assumption can be violated
  - ➢ window of vulnerability

Race Conditions

- Window of vulnerability can be very short
  - race condition problems are difficult to find with testing and difficult to reproduce
  - attacker can slow down victim machine to extend window and can often launch many attempts

- Deadlock
  - special form of race condition
  - two processes are preventing each other from accessing a shared resource, resulting in both processes ceasing to function

Race Conditions

- General assumption
  - sequence of operations
    - is not atomic
    - can be interrupted at any time for arbitrary lengths
  - use proper countermeasures to ensure deterministic results

➢ Synchronization primitives

- Locking
  - can impose performance penalty
  - critical section has to be a small as possible

Race Conditions

- Case study

```
public class Counter extends HttpServlet {
    int count = 0;
    public void doGet(HttpServletRequest in,
                      HttpServletResponse out)
    {
        out.setContentType("text/plain");
        Printwriter p = out.getWriter();
        count++;
        p.println(count + " hits so far!");
    }
}
```


Race Conditions

- Time-of-Check, Time-of-Use (TOCTOU)
  - common race condition problem
  - problem:

    Time-**O**f-**C**heck (t_1): validity of assumption *A* on entity *E* is checked

    Time-**O**f-**U**se (t_2): assuming *A* is still valid, E is used

    Time-Of-Attack (t_3): assumption *A* is invalidated

    $t_1 < t_3 < t_2$

- Program has to execute with elevated privilege
  - otherwise, attacker races for his own privileges

TOCTOU

- Steps to access a resource
  1. obtain reference to resource
  2. query resource to obtain characteristics
  3. analyze query results
  4. if resource is fit, access it

- Often occurs in Unix file system accesses
  - check permissions for a certain file name (e.g., using `access(2)`)
  - open the file, using the file name (e.g., using `fopen(3)`)
  - four levels of indirection (symbolic link - hard link - inode - file descriptor)

- Windows uses file handles and includes checks in API open call

Overview

- Case study

```
/* access returns 0 on success */
if(!access(file, W_OK)) {
     f = fopen(file, "wb+");
     write_to_file(f);
} else {
     fprintf(stderr, "Permission denied when trying
             to open %s.\n", file);
}
```

- Attack

```
$ touch dummy; ln –s dummy pointer
$ rm pointer; ln –s /etc/passwd pointer
```

Examples

- TOCTOU Examples
  - Filename Redirection
    - Paper: Checking for Race Conditions in File Accesses
  - Setuid Scripts
    1. exec() system call invokes seteuid() call prior to executing program
    2. program is a script, so command interpreter is loaded first
    3. program interpreted (with root privileges) is invoked on script name
    4. attacker can replace script content between step 2 and 3

Examples

- TOCTOU Examples
  - Directory operations
    - `rm` can remove directory trees, traverses directories depth-first
    - issues `chdir("..")` to go one level up after removing a directory branch
    - by relocating subdirectory to another directory, arbitrary files can be deleted
  - SQL `select` before `insert`
    - when select returns no results, insert a (unique) element
    - when DB does not check, possible to insert two elements with same key

Examples

- TOCTOU Examples
  - LOMAC
    - Linux kernel level monitor
    - checks system calls (similar to “Secure execution environment” paper)
    - arguments copied to module and checked
    - then, arguments are copied again to invoke actual system call
  - Web site user management
    - user is authenticated at portal page
    - no session management used
    - further pages are not checked because unauthorized user cannot “know” about them

Examples

- TOCTOU Examples
  - File meta-information
    - `chown(2)` and `chmod(2)` are unsafe
    - operate on file names
    - use `fchown(2)` and `fchmod(2)` that use file descriptors
  - Joe Editor
    - when joe crashes (e.g., segmentation fault, xterm crashes)
    - unconditionally append open buffers to local DEADJOE file
    - DEADJOE could be symbolic link to security-relevant file

Temporary Files

- Similar issues as with regular files
  - commonly opened in `/tmp` or `/var/tmp`
  - often guessable file names

- Secure procedure
  1. pick a prefix for your filename
  2. generate at least 64 bits of high-quality randomness
  3. base64 encode the random bits
  4. concatenate the prefix with the encoded random data
  5. set umask appropriately (0066 is usually good)
  6. use `fopen(3)` to create the file, opening it in the proper mode
  7. delete the file immediately using `unlink(2)`
  8. perform reads, writes, and seeks on the file as necessary
  9. finally, close the file

Temporary Files

- Library functions to create temporary files can be insecure
  - `mktemp(3)` is not secure, use `mkstemp(3)` instead
  - old versions of `mkstemp(3)` did not set umask correctly

- Temp Cleaners
  - programs that clean “old” temporary files from `temp` directories
  - first `lstat(2)` file, then use `unlink(2)` to remove files
  - vulnerable to race condition when attacker replaces file between `lstat(2)` and `unlink(2)`
  - arbitrary files can be removed
  - delay program long enough until temp cleaner removes active file

Prevention

- "Handbook of Information Security Management" suggests
  1. increase number of checks
  2. move checks closer to point of use
  3. immutable bindings
- Only number 3 is acceptable!
- Immutable bindings
  - operate on file descriptors
  - do not check access by yourself (i.e., no use of `access(2)`)
    drop privileges instead and let the file system do the job
- Use the `O_CREAT | O_EXCL` flags to create a new file with `open(2)` and be prepared to have the open call fail

Prevention

- Some calls require file names

  `link(), mkdir(), mknod(), rmdir(), symlink(), unlink()`
  - especially `unlink(2)` is troublesome
- Secure File Access
  - create "secure" directory
  - directory only write and executable by UID of process
  - check that no parent directory can be modified by attacker
  - walk up directory tree
    checking for permissions and links at each step

Locking

- Ensures exclusive access to a certain resource
- Used to circumvent accidental race conditions
  - advisory locking (processes need to cooperate)
  - not mandatory, therefore not secure
- Often, files are used for locking
  - portable (files can be created nearly everywhere)
  - “stuck” locks can be easily removed
- Simple method
  - open file using the `O_EXCL` flag

Locking

- Problem
  - NFS up to version 2 does not support `O_EXCL`
  - multiple processes can capture the lock
- Solution (man page for `open(2)`)
  - create unique file on file system (e.g., using host name)
  - use `link(2)` to make a link to lock file
  - when `link(2)` succeeds, or when the link count of the unique file is 2, then the locking operation was successful
- POSIX record locks
  - using `fcntl(2)` calls
  - can lock portions of files, and are automatically removed on process exit

Non-FS Race Conditions

- Linux / BSD kernel `ptrace(2)` / `execve(2)` race condition

- `ptrace(2)`
  - debugging facility
  - used to access other process' registers and memory address space
  - can only attach to processes of same UID, except being run by root

- `execve(2)`
  - execute program image
  - setuid functionality (modifying the process EUID)
    - not invoked when process is marked as being traced

Non-FS Race Conditions

- Problem with `execve(2)`

  1. first checks whether process is being traced
  2. open image (may block)
  3. allocate memory (may block)
  4. set process EUID according to setuid flags

- Window of vulnerability between step 1 and step 4
  - attacker can attach via ptrace
  - blocking kernel operations allow other user processes to run

Non-FS Race Conditions

- Signaler handler race conditions

- Signals
  - used for asynchronous communication between processes
  - signal handler can be called in response to multiple signals
  - signal handler must be written re-entrant
    or block other signals

- Example
  - sendmail up to 8.11.3 and 8.12.0.Beta7
    - `syslog(3)` is called inside the signal handler
    - race condition can cause heap corruption because of double free vulnerability

Non-FS Race Conditions

- Windows DCOM / RPC vulnerability

  - `RPCSS` service
  - multiple threads process single packet
  - one thread frees memory,
    while other process still works on it
  - can result in memory corruption
  - and thus denial of service

Detection

- Static code analysis

  1. specify potentially unsafe patterns
     and perform pattern matching on source code
     - Paper -- “Checking for Race Conditions in File Accesses”
     - RATS (Rough Auditing Tool for Security)

  2. source code analysis and model checking
     - based on finite state machines
     - more precise analysis possible
     - MOPS (MOdel-checking Programs for Security properties)

Detection

- Static code analysis

  3. source code analysis and annotations / rules
     - RacerX (found problems in Linux and commercial software)
     - rccjava (found problems in java.io and java.util)

- Dynamic analysis

  1. inferring data races during runtime
     - “Eraser: A Dynamic Data Race Detector for Multithreaded Programs”