INF226 – Software Security

Håkon Robbestad Gylterud

2019–09-16

System calls and file descriptors

The application and the OS

The operating system provides a rich interface for programs.

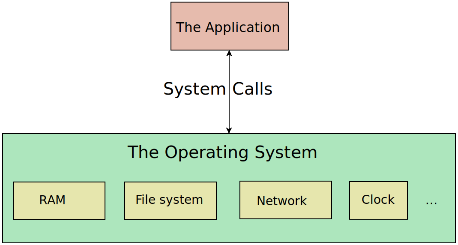

Figure 1: System calls

File descriptors

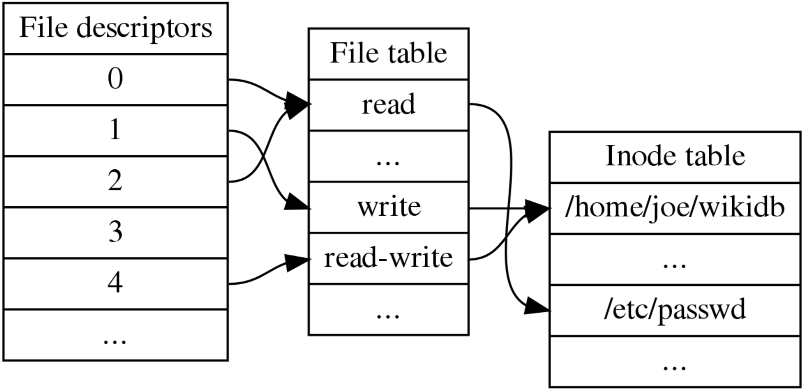

Figure 2: System calls

System call / file descriptor demo

Let us look at the system calls made by some simple programs.

Principle of least priviledge

From the PrivSep article:

> *Every program and every user should operate using the least amount of privilege necessary to complete the job.*

(Similiar formulations to be found in the course books.)

Principle of least priviledge

From the PrivSep article:

> *Every program and every user should operate using the least amount of privilege necessary to complete the job.*

(Similiar formulations to be found in the course books.)

How do we reconcile this with the plethora of system calls available?

Last time

We saw various ways of *restricting process priviledges*:

- Running as unpriviledged user (UID and GID on Linux).
- Quotas and limits on network, filesystem and RAM.
- `chroot`
- Namespaces
- `pledge` or `seccomp`

Today: Priviledge separation in SSH

OpenSSH:

- is an implementation of the SSH (Secure SHell) protocols,
- part of the OpenBSD project
- found on most modern unix-like systems
- provides secure remote access to machine (PKI)
- extensive feature set:
  - remote terminal access
  - X11 forwarding (for GUI)
  - port forwarding (network routing)
  - ...

Preventing priviledge escalation

Motivation

Typical service behaviour:

- Accept requests from network (untrusted)
- Authenticate user
- Allow priviledged operations to authenticated users

Problem: Difficult to safely escalate priviledges once the user is authenticated.

Example

```
void login(int connection) {
   // Get user authentication data from network
   char buffer[1024];
   read_auth_info(buffer);

   if(verify_auth(buffer))
      // User is authenticated!
      escalate_priviledges();
   else
      exit();
}
```

Question: What potential security problems could arise from this code?

Priviledge separation in SSH

Provos, Friedl, Honeyman: Preventing Priviledge Escalation (2003)

Provides a general general pattern of **monitor/slave** processes:

- Monitor:
  - Priviledged
  - Provides an interface for slave to perform priviledged operations.
  - Validates the requests to perform operations.
  - Finite state machine
- Slave:
  - Unpriviledged
  - Does most of the work
  - Calls on monitor when priviledged operations must be performed

Applies it to OpenSSH.

Priviledge separation overview

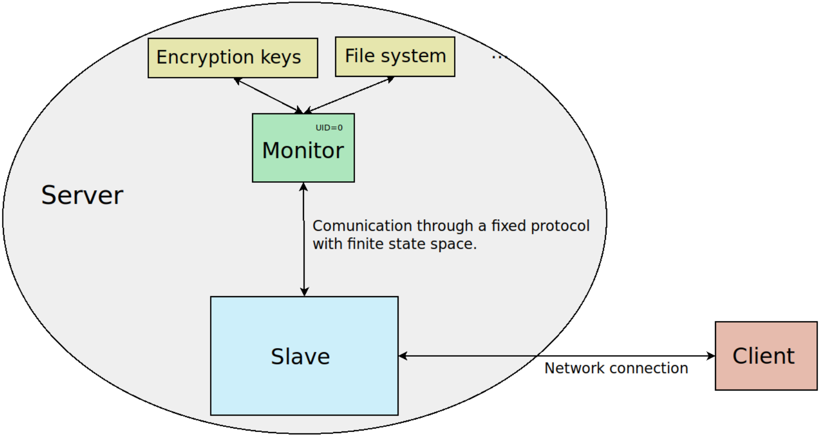

Figure 3: Priviledge separation

Motivation

Basic principle: Limit the amount of code running in a priviledged process.

Benefits:

- Without further holes in the monitor, RCE vulnerabilities are confined to the slave.
- Bugs in unpriviledged part will ideally only result in denial of service for the misbehaving client.
- More intense scrutiny can be given to priviledged parts.
- Simplifying the priviledged part makes reasoning about its security easier.

Implementing the monitor/slave pattern

Indentifying priviledged operations

- File access
- Accessing cryptographic keys
- Data base access
- Spawning pseudo-terminals
- Binding to a network interface

From these operations a service specific **monitor/slave interface** is defined.

This is an example of *functional decomposition*.

Monitor

Monitor does not give **sensitive resources** to slave, but performs actions on its behalf.

Example: Instead of giving access to keys, monitor will make a signature upon request.

Different types of requests

- Information requests
- Capabilities (passing file descriptors)
- Change of identity

Phases

- **Pre-authentication phase**
  - Slave has as little priviledge as possible
  - Monitor only accepts authentication related requests from slave
- **Post-authentication phase**
  - Slave has normal user priviledges
  - Monitor validates requests requiring additional priviledges

Implementating priviledge separation

Once a network connection is made, service spawns a separate monitor/slave pair for that connection.

Slave process is created by:

- Changing UID and GID to unused values
- Chrooted into an empty, unwritable directory
- Marked as `P_SUGID` (prevent information leakage between slaves)
- `pledge("stdio",NULL)`

Slave is given the file descriptor for the network connection.

Slave/master communication

Slave communicates with master through an IPC mechanism such as

- pipe
- shared memory
- **socket-pair**

Change of identity

Once the user is authenticated, the slave should run as a normal user.

Problem: Unix does not support changing UID of a process without UID=0.

Solution:

1. Terminate slave and
2. Monitor spawns a new process with correct UID/GID

To be able to meaningfully continue the session, **slave state must be retained**.

Retaining slave state

The suggested way to retain slave state is by:

- Serializing data structures and transfer to master.
- Allocate dynamic memory resources on memory shared with master.

When new slave is spawned:

- Serialized data structures are passed through IPC
- Memory shared with new slave

Priviledged operations in `sshd`

SSHD priviledged operations in **pre-authentication phase**:

- Access to allowed Diffie-Hellman parameters
- Signing a challenge with server private key to authenticate the connection.
- User validation
- Password authentication
- Public key authentication

The number of requests allowed by slave is limited.

Change of identity

As mentioned:

- data structures are serialized
- shared memory transferred

But a slight complication is the `zlib` compression of the data stream:

- special hooks in `zlib` for custom memory allocation

Priviledged operations in `sshd`

SSHD priviledged operations in **post-authentication phase**:

- Key exchange:
  - SSHv2 supports renewing cryptographic keys
- Pseudo terminal creation (PTY)
  - Requires root to change ownership of a device file
  - Passes the file descriptor to the client

Results

Results

Required updates in code base:

- 950 lines changed (2% of 44 000 total in sshd)
- Additional code added.
- Separate library, `privman` for the general parts.
- Used by other services

Results

Division of code into priviledged and un-priviledged parts:

- 67.70% unpriviledged
- 32.30% priviledged

Security analysis

Assumption: RCE gives attacker control over the slave.

Possible further escalation paths:

- Taking over other system processes
  - Restricted by UID
  - Other slave processes protected by P_SGUID
- System calls to change the file system:
  - File system root empty and unwritable

(cont.)

Possible further esclalation paths (cont.):

- Local network connections:
  - Not preventable by this mechanism
  - May abuse IP based trust relationships
- Gaining information about the system:
  - System time
  - PID of processes
  - Depends on the system if these are accessible through file system or system calls

Other ways to harm the system

The attacker can also attempt using up system resources

- Fork bomb
- Intensive computations

Mitigated by system limits.

Quotas and limits

Resource limits:

- Process number limit
  - preventing DoS by fork bombs, `:(){ :|:& };:`
- File descriptor limit
- Memory limits (data,stack)
- Disk quotas
- Niceness (CPU priority)

Default values in `/etc/login.conf`

Exercises

Read the article *Preventing Priviledge Escalation*, and answer the questions:

1. Which operating system mechanisms does this approach to priviledge separation rely upon?
2. Why does the slave process have to restart when going from pre-authentication phase to post-authentication phase?
3. What does the `P_SUGID` flag do?

Muddiest point

Fill in the form linked from `mitt.uib.no`.

Next lecture: Authentication

- Passwords, entropy and policies
- Storing passwords:
  - Hashing
  - Salting (to protect agains rainbow tables)
  - Key derivation functions
  - Other schemes (PAKE)
- Two-factor authentication

Before the lesson, take a few minutes to watch the TED talk with Larrie Faith Cranor: *What's wrong with your pa$$w0rd?* (link on the Syllabus page).