Principle of Least Privilege

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

Why
Applying the principle to application design limits unintended damage resulting from programming errors.

How
- language enforced protection
  - but operating systems are typically written in C for speed so this is not going to work any time soon, maybe never
- protection mechanisms supported by the operating system
  - but most operating systems distinguish only between superuser and user
- not necessary amount of granularity
  - defensive programming
- prevent errors by checking the integrity of parameters and data structures at implementation, compile or run time
Privilege Separation

What (ideas of Provos, Friedl, Honeyman)
A generic approach to limit the scope of programming bugs

Basic principle: reduce the amount of code that runs with special privilege without affecting or limiting the functionality of the service

Result: exposure to bugs seen during privileged execution is significantly reduced – maybe even eliminated

If done well: the only consequence of an error in a privilege separated service is denial of service to the adversary
Privilege Separation

**What** (ideas of Provos, Friedl, Honeyman)
A generic approach to limit the scope of programming bugs

**Basic principle:** reduce the amount of code that runs with special privilege without affecting or limiting the functionality of the service

**Result:** exposure to bugs seen during privileged execution is significantly reduced – maybe even eliminated

**If done well:** the only consequence of an error in a privilege separated service is denial of service to the adversary

**Application confinement:** app runs in a sandbox
- App subject to security policies
- App may be run in a limited or virtualized environment
- Internal state of a service run on behalf of a contained app is not known to the environment in which it is run (a feature) so the sandbox can't restrict operations that the service might perform for authenticated users
Privilege Escalation

Recall access-1.c and access-2.c for a privilege escalation attack.

```c
int main (int argc, char **argv) {
    if (!access("linker", R_OK)) {
        /*** ———— ———— ———— /**/
        printf("got access as %d\n", getuid());
        system("unlink linker");
        system("ln -s root_owned linker");
        system("chown -h franco:franco linker");
        /*** ———— ———— ———— /**/
        setuid(0);
        system("echo Hello >> linker");
    } else {
        printf("abort\n");
    }
}
```
Privilege Separation

How

Application is split into parts:
one part runs with privileges (the monitor),
other parts run without privileges (the slaves)

A slave asks the monitor to perform an operation that requires privileges

The monitor validates the request
If the request is currently permitted:
the monitor executes it
the monitor reports the results back to the slave
Privilege Separation

How

Application is split into parts:
- one part runs with privileges (the monitor),
- other parts run without privileges (the slaves)

A slave asks the monitor to perform an operation that requires privileges

The monitor validates the request

If the request is currently permitted:
- the monitor executes it
- the monitor reports the results back to the slave

Monitor

Slave

Connect me to 192.168.1.1

Lots of Privileges

Only User Privileges
Privilege Separation

How

Application is split into parts:
one part runs with privileges (the monitor),
other parts run without privileges (the slaves)

A slave asks the monitor to perform an operation that requires privileges

The monitor validates the request

If the request is currently permitted:
the monitor executes it
the monitor reports the results back to the slave

Monitor
Hmmm.. I need to check whether I can do this for this Slave

Slave
Waiting patiently

Lots of Privileges
Privilege Separation

How

Application is split into parts:
- one part runs with privileges (the monitor),
- other parts run without privileges (the slaves)

A slave asks the monitor to perform an operation that requires privileges

The monitor validates the request
If the request is currently permitted:
- the monitor executes it
- the monitor reports the results back to the slave

Looks like I can... Waiting patiently

Lots of Privileges
Privilege Separation

How

Application is split into parts:
  one part runs with privileges (the monitor),
  other parts run without privileges (the slaves)

A slave asks the monitor to perform an operation that requires privileges

The monitor validates the request

If the request is currently permitted:
  the monitor executes it
  the monitor reports the results back to the slave

Looks like I can
Here I go… All kinds of operations happen

Still waiting patiently

Lots of Privileges
Privilege Separation

How

Application is split into parts:
one part runs with privileges (the monitor),
other parts run without privileges (the slaves)

A slave asks the monitor to perform an operation that requires privileges

The monitor validates the request

If the request is currently permitted:
the monitor executes it
the monitor reports the results back to the slave

Ops are complete
Slave can use limited environment

Lots of Privileges

Getting nervous
Privilege Separation

How

Application is split into parts:
  one part runs with privileges (the monitor),
  other parts run without privileges (the slaves)

A slave asks the monitor to perform an operation that requires privileges

The monitor validates the request

If the request is currently permitted:
  the monitor executes it
  the monitor reports the results back to the slave

Lots of Privileges

Monitor

Pointer to environment

Finally!

Slave
Privilege Separation

How

Application is split into parts:
    one part runs with privileges (the monitor),
    other parts run without privileges (the slaves)

A slave asks the monitor to perform an operation that requires privileges

The monitor validates the request

If the request is currently permitted:
    the monitor executes it
    the monitor reports the results back to the slave

Lots of Privileges

Monitor

Slave

Completes tasks via passed environment
Privilege Separation

How

Application is split into parts:
one part runs with privileges (the monitor),
other parts run without privileges (the slaves)

A slave asks the monitor to perform an operation that requires privileges

The monitor validates the request

If the request is currently permitted:
the monitor executes it
the monitor reports the results back to the slave

Coding

Operations are identified as requiring privileges or not
The number of such operations is usually very small
Privilege Separation

Benefits

Number of programming errors that occur in the privileged functions is reduced as the amount of code in those functions is small.

Small code means ability to analyze code for bugs is improved – maybe even formal methods can be applied.

Source code audits can focus on these functions so privilege escalation attacks become less likely.
Privilege Separation

Unix

One process cannot control another unrelated process “processes operate in protection domains”

Processes can fork child processes

The parent can be the monitor and the children slaves

The parent is modeled as a Finite State Machine where each state represents privileges that are allowed or denied to specific children
Privilege Separation

Interface Design

Parent should not be allowed to send sensitive information to any child – for example a private key.

Child requests may involve use of sensitive information but all of it is kept within the parent.

For example, the child may need keys and may request that keys be produced by the parent. But instead of the child using the keys directly, it may ask the parent to encrypt, sign, or even integrity check an object with the keys the parent possesses.
Privilege Separation

Privilege Separated Service Phases

**Pre-Authentication Phase:**
User has contacted service, is not yet authenticated

![Diagram](image)

Monitor

User

Provides a service
Privilege Separation

Privilege Separated Service Phases

**Pre-Authentication Phase:**
Monitor forks a slave process with unused GID and UID

- Monitor
- User
- Slave
- Clueless
Privilege Separation

Privilege Separated Service Phases

**Pre-Authentication Phase:**
- Slave changes its filesystem root to an empty directory
- Slave has no access to any filesystem
- Slave also is unprivileged

Monitor

---

User

Provides a service

Slave

Clueless
Privilege Separation

Privilege Separated Service Phases

**Post-Authentication Phase:**
- User has been authenticated and authorized
- Slave has privileges of the user including file access
- Slave holds no other privileges

[Diagram showing the relationship between Monitor, User, and Slave showing that Monitor provides a service to the User who has user privileges and the Slave who has user privileges plus file access.]
Privilege Separation

Classification of Privileged Operations

**Information Request:**
Slave sends an information request to the monitor only if getting/creating the information requires privileges.

![Diagram showing the relationship between Monitor, User, Slave, and User privileges.](image-url)
Privilege Separation

Classification of Privileged Operations

**Information Request:**

**Example:** slave needs to authenticate the user

![Diagram showing Monitor, User, Slave, and User privileges]

- Monitor: Provides a service
- User
- Slave: User privileges
Privilege Separation

Classification of Privileged Operations

**Information Request:**

**Example:** slave needs to authenticate the user
slave requests a (auth) challenge from the monitor
Privilege Separation

Classification of Privileged Operations

**Information Request:**

**Example:** slave needs to authenticate the user
Monitor computes & saves a challenge R and \( \text{Hash}(R|S) \)
where S is a secret that is shared with the User
Privilege Separation

Classification of Privileged Operations

**Information Request:**

**Example:** slave needs to authenticate the user
Monitor sends challenge to R

![Diagram showing a flow of information from Monitor to R to Slave]

Monitor

\[ R \text{ hash}(R|S) \]

Provides a service

Slave

User

User privileges
Privilege Separation

Classification of Privileged Operations

Information Request:

Example: slave needs to authenticate the user
Slave sends challenge to User

Monitor
R hash(R|S)
Provides a service

User

Slave
User privileges
Privilege Separation

Classification of Privileged Operations

**Information Request:**

**Example:** slave needs to authenticate the user
User responds by computing the Hash(R|S)
Privilege Separation

Classification of Privileged Operations

**Information Request:**

**Example:** slave needs to authenticate the user
Slave sends user response & authentication request to Monitor

Diagram:
- Monitor
  - $R \text{ hash}(R|S)$
  - Provides a service
- User
- Slave
  - User privileges
- $\text{Hash}(R|S) + \text{auth request}$
Privilege Separation

Classification of Privileged Operations

Information Request:

**Example:** slave needs to authenticate the user
Monitor compares saved Hash(R|S) with Value received from Slave and sends success on match – otherwise fail
Privilege Separation

Classification of Privileged Operations

**Information Request:**

**OpenSSH:** most operations are implemented using informational requests
Privilege Separation

Classification of Privileged Operations

**Capability Request:**

Capability request is an information request where the slave expects the monitor to respond with a control message containing a file descriptor `fd`. 
Privilege Separation

Classification of Privileged Operations

**What is a pseudo terminal:**
A master/slave pair of device nodes (may not represent a real device such as /dev/urandom) where the slave emulates a text terminal and the master is the monitor that provides privileges for the slave to operate successfully.

The user inputs commands to the slave.
The slave sends the commands to /bin/bash where it is decided what to do and then do it if allowed.
The monitor sends results back to the slave and therefore the user.
Privilege Separation

Classification of Privileged Operations

**Capability Request:**
Capability request is an information request where the slave expects the monitor to respond with a control message containing a file descriptor \( fd \)

**Example:** slave needs to open a pseudo-terminal for user slave requests file descriptor for PT from monitor monitor execs \texttt{posix	extunderscore openpt()}\texttt{, creates }/dev/ptmx monitor passes \( fd \) to \texttt{grantpt()} user ID of device set to user ID of slave group set to unspecified ID access rights set to \texttt{crx--w----} monitor execs \texttt{unlockpt()} to unlock the slave monitor passes the \( fd \) to the slave
Privilege Separation

Classification of Privileged Operations

Change of Identity:
Issued when a service changes from pre-authenticated to post-authenticated phase (usually)

Problem:
The user id associated with a process can be changed but only by su. But the slave process needing this change does not have su rights

A Solution:
slave exports its entire state to the monitor
monitor kills slave, opens a new slave with user rights
monitor passes state info to new slave
address space is not copied – addresses remain as they were before

Note: monitor needs protection from memory corruption – special structure is used to hold slave data
Privilege Separation

OpenSSH:

- SSH daemon is started and binds to port 22 listening for a connection.
Privilege Separation

OpenSSH:

A new connection is handled by a forked child.
The child needs to use privileges
to spawn user pseudo-terminals
to authenticate key exchanges when keys are replaced
to clean up pseudo-terminals when session ends
to create a process with user privileges
Privilege Separation

OpenSSH:

The child must ask the monitor to sign a crypto hash of all appropriate data during DH key exchange for purposes of authentication (information request)
The child must ask the monitor to sign a crypto hash of all appropriate data during DH key exchange for purposes of authentication (information request). The session key obtained in the key exchange is kept by the slave, slave may encrypt and decrypt.
Privilege Separation

OpenSSH:

The child must ask the monitor to sign a crypto hash of all appropriate data during DH key exchange for purposes of authentication (information request).

The session key obtained in the key exchange is kept by the slave, slave may encrypt and decrypt.

In case of login, the password info is passed to the monitor which checks its user database and answers y/n.
Privilege Separation

OpenSSH:

If the login succeeds, the child must change its identity to that of the user exported state: crypto algorithms, sequence numbers,... privileged ops needed: key exchange, PT creation
Privilege Separation

OpenSSH Security Analysis:

Lines of Code

<table>
<thead>
<tr>
<th>Unprivileged:</th>
<th>Privileged:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ciphers</td>
<td>Authentication</td>
</tr>
<tr>
<td>Packet Handling</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>Monitor</td>
</tr>
<tr>
<td>Priv Sep Interface</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>267</td>
<td>803</td>
</tr>
<tr>
<td>1093</td>
<td>1700</td>
</tr>
<tr>
<td>7944</td>
<td>900</td>
</tr>
<tr>
<td>1056</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>10360</td>
<td>3403</td>
</tr>
</tbody>
</table>

Assume slave may be completely compromised - then
Cannot signal other processes due to different UID
Cannot signal other slave processes – each is marked
and can be signaled only by root
Cannot escape from slave's empty directory
Cannot read /proc or /sys directories
reconnaissance curtailed