POS37-C. Ensure that privilege relinquishment is successful

The POSIX setuid() function has complex semantics and platform-specific behavior [Open Group 2004].

If the process has appropriate privileges, setuid() shall set the real user ID, effective user ID, and the saved set-user-ID of the calling process to uid.
If the process does not have appropriate privileges, but uid is equal to the real user ID or the saved set-user-ID, setuid() shall set the effective user ID to uid; the real user ID and saved set-user-ID shall remain unchanged.

The meaning of "appropriate privileges" varies from platform to platform. For example, on Solaris, appropriate privileges for setuid() means that the PRI V_PROC_SETID privilege is in the effective privilege set of the process. On BSD, it means that the effective user ID (EUID) is zero (that is, the process is running as root) or that uid=geteuid(). On Linux, it means that the process has CAP_SETUID capability and that setuid(geteuid()) will fail if the EUID is not equal to 0, the real user ID (RUID), or the saved set-user ID (SSUID).

Because of this complex behavior, desired privilege drops sometimes may fail. For example, the range of Linux Kernel versions (2.2.0–2.2.15) is vulnerable to an insufficient privilege attack wherein setuid(geteuid()) did not drop privileges as expected when the capability bits were set to zero. As a precautionary measure, subtle behavior and error conditions for the targeted implementation must be carefully noted.

Noncompliant Code Example

This noncompliant code example compiles cleanly on most POSIX systems, but no explicit checks are made to ensure that privilege relinquishment has succeeded. This may be dangerous depending on the sequence of the preceding privilege changes.

```c
/* Code intended to run with elevated privileges */

/* Temporarily drop privileges */
if (seteuid(getuid()) != 0) {
    /* Handle error */
}

/* Code intended to run with lower privileges */

if (need_more_privileges) {
    /* Restore privileges */
    if (seteuid(0) != 0) {
        /* Handle error */
    }

    /* Code intended to run with elevated privileges */
}

/* ... */

/* Permanently drop privileges */
if (setuid(getuid()) != 0) {
    /* Handle error */
}

/* Code intended to run with lower privileges, 
* but if privilege relinquishment failed, 
* attacker can regain elevated privileges! */
```

If the program is run as a setuid root program, over time, the state of the UIDs might look like the following:

<table>
<thead>
<tr>
<th>Description</th>
<th>Code</th>
<th>EUID</th>
<th>RUID</th>
<th>SSUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program startup</td>
<td></td>
<td>0</td>
<td>User</td>
<td>0</td>
</tr>
<tr>
<td>Temporary drop</td>
<td>seteuid(getuid())</td>
<td>User</td>
<td>User</td>
<td>0</td>
</tr>
<tr>
<td>Restore</td>
<td>seteuid(0)</td>
<td>0</td>
<td>User</td>
<td>0</td>
</tr>
<tr>
<td>Permanent drop</td>
<td>setuid(getuid())</td>
<td>User</td>
<td>User</td>
<td>User</td>
</tr>
</tbody>
</table>
If the program fails to restore privileges, it will be unable to permanently drop them later:

<table>
<thead>
<tr>
<th>Description</th>
<th>Code</th>
<th>EUID</th>
<th>RUID</th>
<th>SSUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>program startup</td>
<td></td>
<td>0</td>
<td>User</td>
<td>0</td>
</tr>
<tr>
<td>Temporary drop</td>
<td>seteuid(getuid())</td>
<td>User</td>
<td>User</td>
<td>0</td>
</tr>
<tr>
<td>Restore</td>
<td>seteuid(0)</td>
<td>User</td>
<td>User</td>
<td>0</td>
</tr>
<tr>
<td>Permanent drop</td>
<td>setuid(getuid())</td>
<td>User</td>
<td>User</td>
<td>0</td>
</tr>
<tr>
<td>Restore (attacker)</td>
<td>setuid(0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Compliant Solution

This compliant solution was implemented in sendmail, a popular mail transfer agent, to determine if superuser privileges were successfully dropped [Wheeler 2003]. If the setuid() call succeeds after (supposedly) dropping privileges permanently, then the privileges were not dropped as intended.

```c
/* Code intended to run with elevated privileges */

/* Temporarily drop privileges */
if (seteuid(getuid()) != 0) {
    /* Handle error */
}

/* Code intended to run with lower privileges */

if (need_more_privileges) {
    /* Restore Privileges */
    if (seteuid(0) != 0) {
        /* Handle error */
    }
    /* Code intended to run with elevated privileges */
}

/* ... */

/* Permanently drop privileges */
if (setuid(getuid()) != 0) {
    /* Handle error */
}
if (setuid(0) != -1) {
    /* Privileges can be restored, handle error */
}

/* Code intended to run with lower privileges;
* attacker cannot regain elevated privileges */
```

Compliant Solution

A better solution is to ensure that proper privileges exist before attempting to perform a permanent drop:
Supplementary Group IDs

A process may have a number of supplementary group IDs, in addition to its effective group ID, and the supplementary groups can allow privileged access to files. The `getgroups()` function returns an array that contains the supplementary group IDs and can also contain the effective group ID. The `setgroups()` function can set the supplementary group IDs and can also set the effective group ID on some systems. Using `setgroups()` usually requires privileges. Although POSIX defines the `getgroups()` function, it does not define `setgroups()`.

Under normal circumstances, `setuid()` and related calls do not alter the supplementary group IDs. However, a setuid-root program can alter its supplementary group IDs and then relinquish root privileges, in which case, it maintains the supplementary group IDs but lacks the privilege necessary to relinquish them. Consequently, it is recommended that a program immediately relinquish supplementary group IDs before relinquishing root privileges.

POSIX36-C. Observe correct revocation order while relinquishing privileges discusses how to drop supplementary group IDs. To ensure that supplementary group IDs are indeed relinquished, you can use the following `eql_sups()` function:
System-Specific Capabilities

Many systems have nonportable privilege capabilities that, if unchecked, can yield privilege escalation vulnerabilities. The following section describes one such capability.

File System Access Privileges (Linux)

Processes on Linux have two additional values called *fsuid* and *fsgid*. These values indicate the privileges used when accessing files on the file system. They normally shadow the effective user ID and effective group ID, but the *setfsuid()* and *setfsgid()* functions allow them to be changed. Because changes to the *fsuid* and *fsgid* normally also apply to *euid* and *egid*, a program relinquishing root privileges need not be concerned with setting *fsuid* or *fsgid* to safe values. However, there has been at least one kernel bug that violated this invariant ([Chen 2002] and [Tsafrir 2008]). Consequently, a prudent program checks that *fsuid* and *fsgid* have harmless values after relinquishing privileges.

Risk Assessment

If privilege relinquishment conditions are left unchecked, any flaw in the program may lead to unintended system compromise corresponding to the more privileged user or group account.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS37-C</td>
<td>high</td>
<td>probable</td>
<td>low</td>
<td>P18</td>
<td>L1</td>
</tr>
</tbody>
</table>

Automated Detection

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-POS37</td>
<td></td>
</tr>
<tr>
<td>Klocwork</td>
<td>2018</td>
<td>SV.FIU.PROCESS_VARIANTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SV.USAGERULES.PERMISSIONS</td>
<td></td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-POS37-a</td>
<td>Ensure that privilege relinquishment is successful</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT_C: Rule POS37-C</td>
<td>Checks for privilege drop not verified (rule fully covered)</td>
</tr>
</tbody>
</table>

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the [CERT website](https://csrc.nist.gov/).
### Related Guidelines

**Key here** (explains table format and definitions)

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Taxonomy item</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE 2.11</td>
<td>CWE-273, Failure to check whether privileges were dropped successfully</td>
<td>2017-07-07: CERT: Exact</td>
</tr>
</tbody>
</table>

### Bibliography

- **[Chen 2002]** "Setuid Demystified"
- **[Dowd 2006]** Chapter 9, "Unix I: Privileges and Files"
- **[Open Group 2004]**
  - `setuid()`
  - `getuid()`
  - `seteuid()`
- **[Wheeler 2003]** Section 7.4, "Minimize Privileges"