# CWE Detail – CWE-272

## Description

The elevated privilege level required to perform operations such as chroot() should be dropped immediately after the operation is performed.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Related Attack Patterns (CAPEC)

* CAPEC-17
* CAPEC-35
* CAPEC-76

## Attack TTPs

**•** T1027.009: Embedded Payloads (Tactics: defense-evasion)

**•** T1574.005: Executable Installer File Permissions Weakness (Tactics: persistence, privilege-escalation, defense-evasion)

**•** T1574.010: Services File Permissions Weakness (Tactics: persistence, privilege-escalation, defense-evasion)

**•** T1564.009: Resource Forking (Tactics: defense-evasion)

**•** T1027.006: HTML Smuggling (Tactics: defense-evasion)

## Modes of Introduction

**•** Implementation: REALIZATION: This weakness is caused during implementation of an architectural security tactic.

**•** Operation: N/A

## Common Consequences

**•** Impact: Gain Privileges or Assume Identity, Read Application Data, Read Files or Directories — Notes: An attacker may be able to access resources with the elevated privilege that could not be accessed with the attacker's original privileges. This is particularly likely in conjunction with another flaw, such as a buffer overflow.

## Potential Mitigations

**•** Architecture and Design: Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software. (Effectiveness: N/A)

**•** Architecture and Design: Follow the principle of least privilege when assigning access rights to entities in a software system. (Effectiveness: N/A)

**•** Architecture and Design: Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges. (Effectiveness: N/A)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** N/A

**•** N/A

**•** Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

## Notes

**•** Maintenance: CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category.

**•** Other: If system privileges are not dropped when it is reasonable to do so, this is not a vulnerability by itself. According to the principle of least privilege, access should be allowed only when it is absolutely necessary to the function of a given system, and only for the minimal necessary amount of time. Any further allowance of privilege widens the window of time during which a successful exploitation of the system will provide an attacker with that same privilege. If at all possible, limit the allowance of system privilege to small, simple sections of code that may be called atomically. When a program calls a privileged function, such as chroot(), it must first acquire root privilege. As soon as the privileged operation has completed, the program should drop root privilege and return to the privilege level of the invoking user.