# CWE Detail – CWE-732

## Description

The product specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.

## Extended Description

When a resource is given a permission setting that provides access to a wider range of actors than required, it could lead to the exposure of sensitive information, or the modification of that resource by unintended parties. This is especially dangerous when the resource is related to program configuration, execution, or sensitive user data. For example, consider a misconfigured storage account for the cloud that can be read or written by a public or anonymous user.

## Threat-Mapped Scoring

Score: 3.25

Priority: P2 - Serious (High)

## Observed Examples (CVEs)

**•** CVE-2022-29527: Go application for cloud management creates a world-writable sudoers file that allows local attackers to inject sudo rules and escalate privileges to root by winning a race condition.

**•** CVE-2009-3482: Anti-virus product sets insecure "Everyone: Full Control" permissions for files under the "Program Files" folder, allowing attackers to replace executables with Trojan horses.

**•** CVE-2009-3897: Product creates directories with 0777 permissions at installation, allowing users to gain privileges and access a socket used for authentication.

**•** CVE-2009-3489: Photo editor installs a service with an insecure security descriptor, allowing users to stop or start the service, or execute commands as SYSTEM.

**•** CVE-2020-15708: socket created with insecure permissions

**•** CVE-2009-3289: Library function copies a file to a new target and uses the source file's permissions for the target, which is incorrect when the source file is a symbolic link, which typically has 0777 permissions.

**•** CVE-2009-0115: Device driver uses world-writable permissions for a socket file, allowing attackers to inject arbitrary commands.

**•** CVE-2009-1073: LDAP server stores a cleartext password in a world-readable file.

**•** CVE-2009-0141: Terminal emulator creates TTY devices with world-writable permissions, allowing an attacker to write to the terminals of other users.

**•** CVE-2008-0662: VPN product stores user credentials in a registry key with "Everyone: Full Control" permissions, allowing attackers to steal the credentials.

**•** CVE-2008-0322: Driver installs its device interface with "Everyone: Write" permissions.

**•** CVE-2009-3939: Driver installs a file with world-writable permissions.

**•** CVE-2009-3611: Product changes permissions to 0777 before deleting a backup; the permissions stay insecure for subsequent backups.

**•** CVE-2007-6033: Product creates a share with "Everyone: Full Control" permissions, allowing arbitrary program execution.

**•** CVE-2007-5544: Product uses "Everyone: Full Control" permissions for memory-mapped files (shared memory) in inter-process communication, allowing attackers to tamper with a session.

**•** CVE-2005-4868: Database product uses read/write permissions for everyone for its shared memory, allowing theft of credentials.

**•** CVE-2004-1714: Security product uses "Everyone: Full Control" permissions for its configuration files.

**•** CVE-2001-0006: "Everyone: Full Control" permissions assigned to a mutex allows users to disable network connectivity.

**•** CVE-2002-0969: Chain: database product contains buffer overflow that is only reachable through a .ini configuration file - which has "Everyone: Full Control" permissions.

## Related Attack Patterns (CAPEC)

* CAPEC-1
* CAPEC-122
* CAPEC-127
* CAPEC-17
* CAPEC-180
* CAPEC-206
* CAPEC-234
* CAPEC-60
* CAPEC-61
* CAPEC-62
* CAPEC-642

## Attack TTPs

**•** T1553.002: Code Signing (Tactics: defense-evasion)

**•** T1505.005: Terminal Services DLL (Tactics: persistence)

**•** T1548: Abuse Elevation Control Mechanism (Tactics: privilege-escalation, defense-evasion)

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

**•** T1083: File and Directory Discovery (Tactics: discovery)

**•** T1134.001: Token Impersonation/Theft (Tactics: defense-evasion, privilege-escalation)

**•** T1554: Compromise Host Software Binary (Tactics: persistence)

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

**•** T1550.004: Web Session Cookie (Tactics: defense-evasion, lateral-movement)

## Modes of Introduction

**•** Architecture and Design: N/A

**•** Implementation: REALIZATION: This weakness is caused during implementation of an architectural security tactic. The developer might make certain assumptions about the environment in which the product operates - e.g., that the software is running on a single-user system, or the software is only accessible to trusted administrators. When the software is running in a different environment, the permissions become a problem.

**•** Installation: The developer may set loose permissions in order to minimize problems when the user first runs the program, then create documentation stating that permissions should be tightened. Since system administrators and users do not always read the documentation, this can result in insecure permissions being left unchanged.

**•** Operation: N/A

## Common Consequences

**•** Impact: Read Application Data, Read Files or Directories — Notes: An attacker may be able to read sensitive information from the associated resource, such as credentials or configuration information stored in a file.

**•** Impact: Gain Privileges or Assume Identity — Notes: An attacker may be able to modify critical properties of the associated resource to gain privileges, such as replacing a world-writable executable with a Trojan horse.

**•** Impact: Modify Application Data, Other — Notes: An attacker may be able to destroy or corrupt critical data in the associated resource, such as deletion of records from a database.

## Potential Mitigations

**•** Implementation: When using a critical resource such as a configuration file, check to see if the resource has insecure permissions (such as being modifiable by any regular user) [REF-62], and generate an error or even exit the software if there is a possibility that the resource could have been modified by an unauthorized party. (Effectiveness: N/A)

**•** Architecture and Design: Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully defining distinct user groups, privileges, and/or roles. Map these against data, functionality, and the related resources. Then set the permissions accordingly. This will allow you to maintain more fine-grained control over your resources. [REF-207] (Effectiveness: Moderate)

**•** Architecture and Design: Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails. (Effectiveness: Limited)

**•** Implementation: During program startup, explicitly set the default permissions or umask to the most restrictive setting possible. Also set the appropriate permissions during program installation. This will prevent you from inheriting insecure permissions from any user who installs or runs the program. (Effectiveness: High)

**•** System Configuration: For all configuration files, executables, and libraries, make sure that they are only readable and writable by the software's administrator. (Effectiveness: High)

**•** Documentation: Do not suggest insecure configuration changes in documentation, especially if those configurations can extend to resources and other programs that are outside the scope of the application. (Effectiveness: N/A)

**•** Installation: Do not assume that a system administrator will manually change the configuration to the settings that are recommended in the software's manual. (Effectiveness: N/A)

**•** Operation: Ensure that the software runs properly under the United States Government Configuration Baseline (USGCB) [REF-199] or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software. (Effectiveness: N/A)

**•** Implementation: When storing data in the cloud (e.g., S3 buckets, Azure blobs, Google Cloud Storage, etc.), use the provider's controls to disable public access. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** After running this program on a UNIX system, running the "ls -l" command might return the following output:

**•** Because the optional "mode" argument is omitted from the call to mkdir(), the directory is created with the default permissions 0777. Simply setting the new user as the owner of the directory does not explicitly change the permissions of the directory, leaving it with the default. This default allows any user to read and write to the directory, allowing an attack on the user's files. The code also fails to change the owner group of the directory, which may result in access by unexpected groups.

**•** The first time the program runs, it might create a new file that inherits the permissions from its environment. A file listing might look like:

**•** If the admin file doesn't exist, the program will create one. In order to create the file, the program must have write privileges to write to the file. After the file is created, the permissions need to be changed to read only.

**•** If this command is run from a program, the person calling the program might not expect that all the files under the directory will be world-readable. If the directory is expected to contain private data, this could become a security problem.

**•** However, "Allow Blob Public Access" is set to true, meaning that anonymous/public users can access blobs.

**•** Suppose the command returns the following result:

## Notes

**•** Maintenance: The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-693).