CWE-250: Execution with Unnecessary Privileges

Weakness ID: 250
Vulnerability Mapping: ALLOWED
Abstraction: Base

View customized information: Conceptual | Operational | Mapping Friendly | **Complete** | Custom

Description

The product performs an operation at a privilege level that is higher than the minimum level required, which creates new weaknesses or amplifies the consequences of other weaknesses.

Extended Description

New weaknesses can be exposed because running with extra privileges, such as root or Administrator, can disable the normal security checks being performed by the operating system or surrounding environment. Other pre-existing weaknesses can turn into security vulnerabilities if they occur while operating at raised privileges.

Privilege management functions can behave in some less-than-obvious ways, and they have different quirks on different platforms. These inconsistencies are particularly pronounced if you are transitioning from one non-root user to another. Signal handlers and spawned processes run at the privilege of the owning process, so if a process is running as root when a signal fires or a sub-process is executed, the signal handler or sub-process will operate with root privileges.

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control	**Technical Impact:** *Gain Privileges or Assume Identity; Execute Unauthorized Code or Commands; Read Application Data; DoS: Crash, Exit, or Restart* An attacker will be able to gain access to any resources that are allowed by the extra privileges. Common results include executing code, disabling services, and reading restricted data.	

Potential Mitigations

Phases: Architecture and Design; Operation

Strategy: Environment Hardening

Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Architecture and Design

Strategy: Separation of Privilege

Identify the functionality that requires additional privileges, such as access to privileged operating system resources. Wrap and centralize this functionality if possible, and isolate the privileged code as much as possible from other code [REF-76]. Raise privileges as late as possible, and drop them as soon as possible to avoid CWE-271. Avoid weaknesses such as CWE-288 and CWE-420 by protecting all possible communication channels that could interact with the privileged code, such as a secondary socket that is only intended to be accessed by administrators.

Phase: Architecture and Design

Strategy: Attack Surface Reduction

Identify the functionality that requires additional privileges, such as access to privileged operating system resources. Wrap and centralize this functionality if possible, and isolate the privileged code as much as possible from other code [REF-76]. Raise privileges as late as possible, and drop them as soon as possible to avoid CWE-271. Avoid weaknesses such as CWE-288 and CWE-420 by protecting all possible communication channels that could interact with the privileged code, such as a secondary socket that is only intended to be accessed by administrators.

Phase: Implementation

Perform extensive input validation for any privileged code that must be exposed to the user and reject anything that does not fit your strict requirements.

Phase: Implementation

When dropping privileges, ensure that they have been dropped successfully to avoid CWE-273. As protection mechanisms in the environment get stronger, privilege-dropping calls may fail even if it seems like they would always succeed.

Phase: Implementation

If circumstances force you to run with extra privileges, then determine the minimum access level necessary. First identify the different permissions that the software and its users will need to perform their actions, such as file read and write permissions, network socket permissions, and so forth. Then explicitly allow those actions while denying all else [REF-76]. Perform extensive input validation and canonicalization to minimize the chances of introducing a separate vulnerability. This mitigation is much more prone to error than dropping the privileges in the first place.

Phases: Operation; System Configuration

Strategy: Environment Hardening

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.

Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	C	657	Violation of Secure Design Principles
ChildOf	C	269	Improper Privilege Management

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name
MemberOf	C	265	Privilege Issues

Relevant to the view "Architectural Concepts" (CWE-1008)

Modes Of Introduction

Phase	Note
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.
Installation	
Architecture and Design	If an application has this design problem, then it can be easier for the developer to make implementation-related errors such as CWE-271 (Privilege Dropping / Lowering Errors). In addition, the consequences of Privilege Chaining (CWE-268) can become more severe.
Operation	

Applicable Platforms

Languages

Class: Not Language-Specific *(Undetermined Prevalence)*

Technologies

Class: Mobile *(Undetermined Prevalence)*

Likelihood Of Exploit

Medium

Demonstrative Examples

Example 1

This code temporarily raises the program's privileges to allow creation of a new user folder.

Example Language: **Python** *(bad code)*

```
def makeNewUserDir(username):
  if invalidUsername(username):

    #avoid CWE-22 and CWE-78
    print('Usernames cannot contain invalid characters')
    return False

  try:
    raisePrivileges()
    os.mkdir('/home/' + username)
    lowerPrivileges()

  except OSError:
    print('Unable to create new user directory for user:' + username)
    return False

  return True
```

While the program only raises its privilege level to create the folder and immediately lowers it again, if the call to os.mkdir() throws an exception, the call to lowerPrivileges() will not occur. As a result, the program is indefinitely operating in a raised privilege state, possibly allowing further exploitation to occur.

Example 2

The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.

Example Language: **C** *(bad code)*

```
chroot(APP_HOME);
chdir("/");
FILE* data = fopen(argv[1], "r+");
...
```

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.

Example 3

This application intends to use a user's location to determine the timezone the user is in:

Example Language: **Java** *(bad code)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
setTimeZone(userCurrLocation);
```

This is unnecessary use of the location API, as this information is already available using the Android Time API. Always be sure there is not another way to obtain needed information before resorting to using the location API.

Example 4

This code uses location to determine the user's current US State location.

First the application must declare that it requires the ACCESS_FINE_LOCATION permission in the application's manifest.xml:

Example Language: **XML** *(bad code)*

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

During execution, a call to getLastLocation() will return a location based on the application's location permissions. In this case the application has permission for the most accurate location possible:

Example Language: **Java** *(bad code)*

```
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
deriveStateFromCoords(userCurrLocation);
```

While the application needs this information, it does not need to use the ACCESS_FINE_LOCATION permission, as the ACCESS_COARSE_LOCATION permission will be sufficient to identify which US state the user is in.

Observed Examples

Reference	Description
CVE-2007-4217	FTP client program on a certain OS runs with setuid privileges and has a buffer overflow. Most clients do not need extra privileges, so an overflow is not a vulnerability for those clients.
CVE-2008-1877	Program runs with privileges and calls another program with the same privileges, which allows read of arbitrary files.
CVE-2007-5159	OS incorrectly installs a program with setuid privileges, allowing users to gain privileges.
CVE-2008-4638	Composite: application running with high privileges (CWE-250) allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file (CWE-209).
CVE-2008-0162	Program does not drop privileges before calling another program, allowing code execution.
CVE-2008-0368	setuid root program allows creation of arbitrary files through command line argument.
CVE-2007-3931	Installation script installs some programs as setuid when they shouldn't be.
CVE-2020-3812	mail program runs as root but does not drop its privileges before attempting to access a file. Attacker can use a symlink from their home directory to a directory only readable by root, then determine whether the file exists based on the response.
CVE-2003-0908	Product launches Help functionality while running with raised privileges, allowing command execution using Windows message to access "open file" dialog.

Detection Methods

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Note: These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and perform a login. Look for library functions and system calls that indicate when privileges are being raised or dropped. Look for accesses of resources that are restricted to normal users.

Note: Note that this technique is only useful for privilege issues related to system resources. It is not likely to detect application-level business rules that are related to privileges, such as if a blog system allows a user to delete a blog entry without first checking that the user has administrator privileges.

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful:

Highly cost effective:

- Compare binary / bytecode to application permission manifest

Cost effective for partial coverage:

- Bytecode Weakness Analysis - including disassembler + source code weakness analysis
- Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness: High

Manual Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

- Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies

Effectiveness: SOAR Partial

Dynamic Analysis with Automated Results Interpretation

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

- Host-based Vulnerability Scanners - Examine configuration for flaws, verifying that audit mechanisms work, ensure host configuration meets certain predefined criteria

Effectiveness: SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

- Host Application Interface Scanner

Effectiveness: SOAR Partial

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful:

Highly cost effective:

- Manual Source Code Review (not inspections)

Cost effective for partial coverage:

- Focused Manual Spotcheck - Focused manual analysis of source

Effectiveness: High

Automated Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

- Source code Weakness Analyzer
- Context-configured Source Code Weakness Analyzer

Effectiveness: SOAR Partial

Automated Static Analysis

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

- Configuration Checker
- Permission Manifest Analysis

Effectiveness: SOAR Partial

Architecture or Design Review

According to SOAR, the following detection techniques may be useful:

Highly cost effective:

- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)
- Formal Methods / Correct-By-Construction

Cost effective for partial coverage:

- Attack Modeling

Effectiveness: High

Memberships

Nature	Type	ID	Name
MemberOf	C	227	7PK - API Abuse
MemberOf	C	753	2009 Top 25 - Porous Defenses
MemberOf	C	815	OWASP Top Ten 2010 Category A6 - Security Misconfiguration
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)
MemberOf	C	866	2011 Top 25 - Porous Defenses
MemberOf	V	884	CWE Cross-section
MemberOf	C	901	SFP Primary Cluster: Privilege
MemberOf	C	1418	Comprehensive Categorization: Violation of Secure Design Principles

Vulnerability Mapping Notes

Usage: ALLOWED *(this CWE ID could be used to map to real-world vulnerabilities)*

Reason: Acceptable-Use

Rationale:

This CWE entry is at the Base level of abstraction, which is a preferred level of abstraction for mapping to the root causes of vulnerabilities.

Comments:

Carefully read both the name and description to ensure that this mapping is an appropriate fit. Do not try to 'force' a mapping to a lower-level Base/Variant simply to comply with this preferred level of abstraction.

Notes

Relationship

There is a close association with CWE-653 (Insufficient Separation of Privileges). CWE-653 is about providing separate components for each privilege; CWE-250 is about ensuring that each component has the least amount of privileges possible.

Maintenance

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category. Both CWE-272 and CWE-250 are in active use by the community. The "least privilege" phrase has multiple interpretations.

Maintenance

The Taxonomy_Mappings to ISA/IEC 62443 were added in CWE 4.10, but they are still under review and might change in future CWE versions. These draft mappings were performed by members of the "Mapping CWE to 62443" subgroup of the CWE-CAPEC ICS/OT Special Interest Group (SIG), and their work is incomplete as of CWE 4.10. The mappings are included to facilitate discussion and review by the broader ICS/OT community, and they are likely to change in future CWE versions.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Often Misused: Privilege Management
The CERT Oracle Secure Coding Standard for Java (2011)	SER09-J		Minimize privileges before deserializing from a privilege context
ISA/IEC 62443	Part 2-4		Req SP.03.05 BR
ISA/IEC 62443	Part 2-4		Req SP.03.08 BR
ISA/IEC 62443	Part 2-4		Req SP.03.08 RE(1)
ISA/IEC 62443	Part 2-4		Req SP.05.07 BR
ISA/IEC 62443	Part 2-4		Req SP.09.02 RE(4)
ISA/IEC 62443	Part 2-4		Req SP.09.03 BR
ISA/IEC 62443	Part 2-4		Req SP.09.04 BR
ISA/IEC 62443	Part 3-3		Req SR 1.1
ISA/IEC 62443	Part 3-3		Req SR 1.2
ISA/IEC 62443	Part 3-3		Req SR 2.1
ISA/IEC 62443	Part 3-3		Req SR 2.1 RE 1
ISA/IEC 62443	Part 4-1		Req SD-4
ISA/IEC 62443	Part 4-2		Req CCSC 3
ISA/IEC 62443	Part 4-2		Req CR 1.1

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-104	Cross Zone Scripting
CAPEC-470	Expanding Control over the Operating System from the Database
CAPEC-69	Target Programs with Elevated Privileges

References

[REF-6] Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. NIST. 2005-11-07. <https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf>.

[REF-196] Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. 1975-09. <http://web.mit.edu/Saltzer/www/publications/protection/>.

[REF-76] Sean Barnum and Michael Gegick. "Least Privilege". 2005-09-14. <https://web.archive.org/web/20211209014121/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-privilege>. *URL validated: 2023-04-07.*

[REF-7] Michael Howard and David LeBlanc. "Writing Secure Code". Chapter 7, "Running with Least Privilege" Page 207. 2nd Edition. Microsoft Press. 2002-12-04. <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223>.

[REF-199] NIST. "United States Government Configuration Baseline (USGCB)". <https://csrc.nist.gov/Projects/United-States-Government-Configuration-Baseline>. *URL validated: 2023-03-28.*

[REF-44] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 16: Executing Code With Too Much Privilege." Page 243. McGraw-Hill. 2010.

[REF-62] Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". Chapter 9, "Privilege Vulnerabilities", Page 477. 1st Edition. Addison Wesley. 2006.

Content History

Submissions

Submission Date	Submitter	Organization
2006-07-19 *(CWE Draft 3, 2006-07-19)*	7 Pernicious Kingdoms	

Contributions

Contribution Date	Contributor	Organization
2023-01-24 *(CWE 4.10, 2023-01-31)*	"Mapping CWE to 62443" Sub-Working Group Suggested mappings to ISA/IEC 62443.	CWE-CAPEC ICS/OT SIG
2023-04-25	"Mapping CWE to 62443" Sub-Working Group Suggested mappings to ISA/IEC 62443.	CWE-CAPEC ICS/OT SIG

Modifications

Previous Entry Names

Page Last Updated: July 16, 2024