



ICS Advisory (ICSA-22-154-01)

[More ICS-CERT Advisories](#)

Vulnerabilities Affecting Dominion Voting Systems ImageCast X

Original release date: June 03, 2022

Legal Notice

All information products included in <https://us-cert.cisa.gov/ics> are provided "as is" for informational purposes only. The Department of Homeland Security (DHS) does not provide any warranties of any kind regarding any information contained within. DHS does not endorse any commercial product or service, referenced in this product or otherwise. Further dissemination of this product is governed by the Traffic Light Protocol (TLP) marking in the header. For more information about TLP, see <https://us-cert.cisa.gov/tlp/>.

1. SUMMARY

This advisory identifies vulnerabilities affecting versions of the Dominion Voting Systems Democracy Suite ImageCast X, which is an in-person voting system used to allow voters to mark their ballot. The ImageCast X can be configured to allow a voter to produce a paper record or to record votes electronically. While these vulnerabilities present risks that should be mitigated as soon as possible, CISA has no evidence that these vulnerabilities have been exploited in any elections.

Exploitation of these vulnerabilities would require physical access to individual ImageCast X devices, access to the Election Management System (EMS), or the ability to modify files before they are uploaded to ImageCast X devices. Jurisdictions can prevent and/or detect the exploitation of these vulnerabilities by diligently applying the mitigations recommended in this advisory, including technical, physical, and operational controls that limit unauthorized access or manipulation of voting systems. Many of these mitigations are already typically standard practice in jurisdictions where these devices are in use and can be enhanced to further guard against exploitation of these vulnerabilities.

2. TECHNICAL DETAILS

2.1 AFFECTED PRODUCTS

The following versions of the Dominion Voting Systems ImageCast X software are known to be affected (other versions were not able to be tested):

- ImageCast X firmware based on Android 5.1, as used in Dominion Democracy Suite Voting System Version 5.5-A
- ImageCast X application Versions 5.5.10.30 and 5.5.10.32, as used in Dominion Democracy Suite Voting System Version 5.5-A
 - **NOTE:** After following the vendor's procedure to upgrade the ImageCast X from Version 5.5.10.30 to 5.5.10.32, or after performing other Android administrative actions, the ImageCast X may be left in a configuration that could allow an attacker who can attach an external input device to escalate privileges and/or install malicious code. Instructions to check for and mitigate this condition are available from Dominion Voting Systems.

Any jurisdictions running ImageCast X are encouraged to contact Dominion Voting Systems to understand the vulnerability status of their specific implementation.

2.2 VULNERABILITY OVERVIEW

NOTE: Mitigations to reduce the risk of exploitation of these vulnerabilities can be found in Section 3 of this document.

2.2.1 IMPROPER VERIFICATION OF CRYPTOGRAPHIC SIGNATURE CWE-347

The tested version of ImageCast X does not validate application signatures to a trusted root certificate. Use of a trusted root certificate ensures software installed on a device is traceable to, or verifiable against, a cryptographic key provided by the manufacturer to detect tampering. An attacker could leverage this vulnerability to install malicious code, which could also be spread to other vulnerable ImageCast X devices via removable media.

CVE-2022-1739 has been assigned to this vulnerability.

2.2.2 MUTABLE ATTESTATION OR MEASUREMENT REPORTING DATA CWE-1283

The tested version of ImageCast X's on-screen application hash display feature, audit log export, and application export functionality rely on self-attestation mechanisms. An attacker could leverage this vulnerability to disguise malicious applications on a device.

CVE-2022-1740 has been assigned to this vulnerability.

2.2.3 HIDDEN FUNCTIONALITY CWE-912

The tested version of ImageCast X has a Terminal Emulator application which could be leveraged by an attacker to gain elevated privileges on a device and/or install malicious code.

CVE-2022-1741 has been assigned to this vulnerability.

2.2.4 IMPROPER PROTECTION OF ALTERNATE PATH CWE-424

The tested version of ImageCast X allows for rebooting into Android Safe Mode, which allows an attacker to directly access the operating system. An attacker could leverage this vulnerability to escalate privileges on a device and/or install malicious code.

CVE-2022-1742 has been assigned to this vulnerability.

2.2.5 PATH TRAVERSAL: './FILEDIR' CWE-24

The tested version of ImageCast X can be manipulated to cause arbitrary code execution by specially crafted election definition files. An attacker could leverage this vulnerability to spread malicious code to ImageCast X devices from the EMS.

CVE-2022-1743 has been assigned to this vulnerability.

2.2.6 EXECUTION WITH UNNECESSARY PRIVILEGES CWE-250

Applications on the tested version of ImageCast X can execute code with elevated privileges by exploiting a system level service. An attacker could leverage this vulnerability to escalate privileges on a device and/or install malicious code.

CVE-2022-1744 has been assigned to this vulnerability.

2.2.7 AUTHENTICATION BYPASS BY SPOOFING CWE-290

The authentication mechanism used by technicians on the tested version of ImageCast X is susceptible to forgery. An attacker with physical access may use this to gain administrative privileges on a device and install malicious code or perform arbitrary administrative actions.

CVE-2022-1745 has been assigned to this vulnerability.

2.2.8 INCORRECT PRIVILEGE ASSIGNMENT CWE-266

The authentication mechanism used by poll workers to administer voting using the tested version of ImageCast X can expose cryptographic secrets used to protect election information. An attacker could leverage this vulnerability to gain access to sensitive information and perform privileged actions, potentially affecting other election equipment.

CVE-2022-1746 has been assigned to this vulnerability.

2.2.9 ORIGIN VALIDATION ERROR CWE-346

The authentication mechanism used by voters to activate a voting session on the tested version of ImageCast X is susceptible to forgery. An attacker could leverage this vulnerability to print an arbitrary number of ballots without authorization.

CVE-2022-1747 has been assigned to this vulnerability.

2.3 BACKGROUND

- **CRITICAL INFRASTRUCTURE SECTORS** Government Facilities / Election Infrastructure
- **COUNTRIES/AREAS DEPLOYED:** Multiple

- **COMPANY HEADQUARTERS LOCATION:** Denver, Colorado

TLP:WHITE

2.4 RESEARCHER

J. Alex Halderman, University of Michigan, and Drew Springall, Auburn University, reported these vulnerabilities to CISA.

3. MITIGATIONS

CISA recommends election officials continue to take and further enhance defensive measures to reduce the risk of exploitation of these vulnerabilities. Specifically, for each election, election officials should:

- Contact Dominion Voting Systems to determine which software and/or firmware updates need to be applied. Dominion Voting Systems reports to CISA that the above vulnerabilities have been addressed in subsequent software versions.
- Ensure all affected devices are physically protected before, during, and after voting.
- Ensure compliance with chain of custody procedures throughout the election cycle.
- Ensure that ImageCast X and the Election Management System (EMS) are not connected to any external (i.e., Internet accessible) networks.
- Ensure carefully selected protective and detective physical security measures (for example, locks and tamper-evident seals) are implemented on all affected devices, including on connected devices such as printers and connecting cables.
- Close any background application windows on each ImageCast X device.
- Use read-only media to update software or install files onto ImageCast X devices.
- Use separate, unique passcodes for each poll worker card.
- Ensure all ImageCast X devices are subjected to rigorous pre- and post-election testing.
- Disable the “Unify Tabulator Security Keys” feature on the election management system and ensure new cryptographic keys are used for each election.
- As recommended by Dominion Voting Systems, use the supplemental method to validate hashes on applications, audit log exports, and application exports.
- Encourage voters to verify the human-readable votes on printout.
- Conduct rigorous post-election tabulation audits of the human-readable portions of physical ballots and paper records, to include reviewing ballot chain of custody and conducting voter/ballot reconciliation procedures. These activities are especially crucial to detect attacks where the listed vulnerabilities are exploited such that a barcode is manipulated to be tabulated inconsistently with the human-readable portion of the paper ballot. (**NOTE:** If states and jurisdictions so choose, the ImageCast X provides the configuration option to produce ballots that do not print barcodes for tabulation.)

Contact Information

TLP:WHITE

For any questions related to this report, please contact the CISA at:

TLP:WHITE

Email: CISAservicedesk@cisa.dhs.gov

Toll Free: 1-888-282-0870

For industrial control systems cybersecurity information: <https://us-cert.cisa.gov/ics>
or incident reporting: <https://us-cert.cisa.gov/report>

CISA continuously strives to improve its products and services. You can help by choosing one of the links below to provide feedback about this product.

This product is provided subject to this Notification and this Privacy & Use policy.

TLP:WHITE

CWE-347: Improper Verification of Cryptographic Signature

Weakness ID: 347
Abstraction: Base

Structure: Simple

 Presentation Filter:

▼ Description

The software does not verify, or incorrectly verifies, the cryptographic signature for data.

▼ Relationships

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

Nature	Type	ID	Name
ChildOf		345	Insufficient Verification of Data Authenticity

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

Nature	Type	ID	Name
MemberOf		1214	Data Integrity Issues
MemberOf		310	Cryptographic Issues

📘 Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

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

▼ Modes Of Introduction

Phase	Note
Architecture and Design	
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.

▼ Applicable Platforms

📘 Languages

 Class: Language-Independent (*Undetermined Prevalence*)

▼ Common Consequences

Scope	Impact	Likelihood
Access Control Integrity Confidentiality	Technical Impact: <i>Gain Privileges or Assume Identity; Modify Application Data; Execute Unauthorized Code or Commands</i> An attacker could gain access to sensitive data and possibly execute unauthorized code.	

▼ Demonstrative Examples

Example 1

In the following code, a JarFile object is created from a downloaded file.

 Example Language: **Java**

(bad code)

```
File f = new File(downloadedFilePath);
JarFile jf = new JarFile(f);
```

The JAR file that was potentially downloaded from an untrusted source is created without verifying the signature (if present). An alternate constructor that accepts a boolean verify parameter should be used instead.





▼ Observed Examples

Reference	Description
CVE-2002-1796	Does not properly verify signatures for "trusted" entities.
CVE-2005-2181	Insufficient verification allows spoofing.
CVE-2005-2182	Insufficient verification allows spoofing.

[CVE-2002-1706](#)

Accepts a configuration file without a Message Integrity Check (MIC) signature.

▼ Memberships

Nature	Type	ID	Name
MemberOf		859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)
MemberOf		884	CWE Cross-section
MemberOf		959	SFP Secondary Cluster: Weak Cryptography
MemberOf		1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures

▼ Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improperly Verified Signature
The CERT Oracle Secure Coding Standard for Java (2011)	SEC06-J		Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar

▼ Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-463	Padding Oracle Crypto Attack
CAPEC-475	Signature Spoofing by Improper Validation

▼ Content History

Submissions		
Submission Date	Submitter	Organization
2006-07-19	PLOVER	
Modifications		
Previous Entry Names		

CWE-1283: Mutable Attestation or Measurement Reporting Data

Weakness ID: 1283
Abstraction: Base

Structure: Simple

 Presentation Filter:

▼ Description

The register contents used for attestation or measurement reporting data to verify boot flow are modifiable by an adversary.

▼ Extended Description

A System-on-Chip (SoC) implements secure boot or verified boot. During this boot flow, the SoC often measures the code that it authenticates. The measurement is usually done by calculating the one-way hash of the code binary and extending it to the previous hash. The hashing algorithm should be a Secure One-Way hash function. The final hash, i.e., the value obtained after the completion of the boot flow, serves as the measurement data used in reporting or in attestation. The calculated hash is often stored in registers that can later be read by the party of interest to determine tampering of the boot flow. A common weakness is that the contents in these registers are modifiable by an adversary, thus spoofing the measurement.

▼ Relationships

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

Nature	Type	ID	Name
ChildOf	P	284	Improper Access Control

📘 Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name
MemberOf	C	1196	Security Flow Issues

▼ Modes Of Introduction

Phase	Note
Architecture and Design	Such issues can be introduced during hardware architecture or design and can be identified later during Testing or System Configuration phases.
Implementation	If the access-controls which protecting the reporting registers are misconfigured during implementation, this weakness can arise.

▼ Applicable Platforms

📘 Languages

 Class: Language-Independent (*Undetermined Prevalence*)

Operating Systems

 Class: OS-Independent (*Undetermined Prevalence*)

Architectures

 Class: Architecture-Independent (*Undetermined Prevalence*)

Technologies

 Class: Technology-Independent (*Undetermined Prevalence*)

▼ Common Consequences

Scope	Impact	Likelihood
Confidentiality	Technical Impact: <i>Read Memory; Read Application Data</i>	

▼ Demonstrative Examples

Example 1

The SoC extends the hash and stores the results in registers. Without protection, an adversary can write their chosen hash values to these registers. Thus, the attacker controls the reported results.

To prevent the above scenario, the registers should have one or more of the following properties:

1. Should be Read-Only with respect to an adversary
2. Cannot be extended or modifiable either directly or indirectly (using a trusted agent as proxy) by an adversary
3. Should have appropriate access controls or protections

▼ Potential Mitigations

Phase: Architecture and Design

Measurement data should be stored in registers that are read-only or otherwise have access controls that prevent modification by an untrusted agent.

▼ Notes

Maintenance

This entry is still in development and will continue to see updates and content improvements.

▼ Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-680	Exploitation of Improperly Controlled Registers

▼ References

[REF-1107] Intel Corporation. "PCIe Device Measurement Requirements". 2018-09. <<https://www.intel.com/content/dam/www/public/us/en/documents/reference-guides/pcie-device-security-enhancements.pdf>>.

[REF-1131] John Butterworth, Cory Kallenberg and Xeno Kovah. "BIOS Chronomancy: Fixing the Core Root of Trust for Measurement". 2013-07-31. <<https://media.blackhat.com/us-13/US-13-Butterworth-BIOS-Security-Slides.pdf>>.

▼ Content History

▼ Submissions

Submission Date	Submitter	Organization
2020-04-25	Arun Kanuparthi, Hareesh Khattri, Parbati Kumar Manna, Narasimha Kumar V Mangipudi	Intel Corporation

► Modifications

CWE-912: Hidden Functionality

Weakness ID: 912
Abstraction: Class

Structure: Simple

 Presentation Filter:

▼ Description

The software contains functionality that is not documented, not part of the specification, and not accessible through an interface or command sequence that is obvious to the software's users or administrators.

▼ Extended Description

Hidden functionality can take many forms, such as intentionally malicious code, "Easter Eggs" that contain extraneous functionality such as games, developer-friendly shortcuts that reduce maintenance or support costs such as hard-coded accounts, etc. From a security perspective, even when the functionality is not intentionally malicious or damaging, it can increase the software's attack surface and expose additional weaknesses beyond what is already exposed by the intended functionality. Even if it is not easily accessible, the hidden functionality could be useful for attacks that modify the control flow of the application.

▼ Relationships

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

Nature	Type	ID	Name
ChildOf		684	Incorrect Provision of Specified Functionality
ParentOf		506	Embedded Malicious Code

▼ Modes Of Introduction

Phase	Note
Architecture and Design	
Implementation	

▼ Common Consequences

Scope	Impact	Likelihood
Other Integrity	Technical Impact: <i>Varies by Context; Alter Execution Logic</i>	

▼ Potential Mitigations

Phase: Installation

Always verify the integrity of the software that is being installed.

Phase: Testing

Conduct a code coverage analysis using live testing, then closely inspect any code that is not covered.

▼ Memberships

Nature	Type	ID	Name
MemberOf		1371	ICS Supply Chain: Poorly Documented or Undocumented Features

▼ Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-133	Try All Common Switches
CAPEC-190	Reverse Engineer an Executable to Expose Assumed Hidden Functionality

▼ Content History

Submissions		
Submission Date	Submitter	Organization
2012-12-28	CWE Content Team	MITRE

▼ Submissions

► Modifications

CWE-424: Improper Protection of Alternate Path

Weakness ID: 424
Abstraction: Class

Structure: Simple




 Presentation Filter:

▼ Description

The product does not sufficiently protect all possible paths that a user can take to access restricted functionality or resources.

▼ Relationships

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

Nature	Type	ID	Name
ChildOf		693	Protection Mechanism Failure
ChildOf		638	Not Using Complete Mediation
ParentOf		425	Direct Request ('Forced Browsing')

▼ Modes Of Introduction

Phase	Note
Architecture and Design	

▼ Applicable Platforms

📘 Languages

 Class: Language-Independent (*Undetermined Prevalence*)

▼ Common Consequences






Scope	Impact	Likelihood
Access Control	Technical Impact: <i>Bypass Protection Mechanism; Gain Privileges or Assume Identity</i>	

▼ Potential Mitigations

Phase: Architecture and Design

Deploy different layers of protection to implement security in depth.

▼ Memberships

Nature	Type	ID	Name
MemberOf		945	SFP Secondary Cluster: Insecure Resource Access
MemberOf		1306	CISQ Quality Measures - Reliability
MemberOf		1308	CISQ Quality Measures - Security
MemberOf		1309	CISQ Quality Measures - Efficiency
MemberOf		1340	CISQ Data Protection Measures

▼ Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Alternate Path Errors
Software Fault Patterns	SFP35		Insecure resource access

▼ Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-127	Directory Indexing
CAPEC-554	Functionality Bypass

▼ Content History

▼ Submissions

▼ Submissions		
Submission Date	Submitter	Organization
2006-07-19	PLOVER	
▶ Modifications		
▶ Previous Entry Names		

CWE-24: Path Traversal: '../filedir'

Weakness ID: 24
Abstraction: Variant

Structure: Simple

 Presentation Filter:

▼ Description

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize "../ sequences that can resolve to a location that is outside of that directory.

▼ Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The "../ manipulation is the canonical manipulation for operating systems that use "/" as directory separators, such as UNIX- and Linux-based systems. In some cases, it is useful for bypassing protection schemes in environments for which "/" is supported but not the primary separator, such as Windows, which uses "\" but can also accept "/".

▼ Relationships

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

Nature	Type	ID	Name
ChildOf	ⓑ	23	Relative Path Traversal

▼ Modes Of Introduction

Phase	Note
Architecture and Design	
Implementation	

▼ Applicable Platforms

📘 Languages

 Class: Language-Independent (*Undetermined Prevalence*)

▼ Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity	Technical Impact: <i>Read Files or Directories; Modify Files or Directories</i>	

▼ Potential Mitigations

Phase: Implementation

Strategy: Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as [CWE-23](#), and exclude directory separators such as "/" to avoid [CWE-36](#). Use a list of allowable file extensions, which will help to avoid [CWE-434](#).

Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete ([CWE-184](#)). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data ([CWE-182](#)). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.

Phase: Implementation

Strategy: Input Validation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated ([CWE-180](#)). Make sure that the application does not decode the same input twice ([CWE-174](#)). Such errors could be used to bypass allowlist validation schemes by introducing dangerous inputs after they have been checked.

Memberships

Nature	Type	ID	Name
MemberOf		981	SFP Secondary Cluster: Path Traversal

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			'./filedir
Software Fault Patterns	SFP16		Path Traversal

Content History

Submissions		
Submission Date	Submitter	Organization
2006-07-19	PLOVER	
Modifications		
Previous Entry Names		

CWE-250: Execution with Unnecessary Privileges

Weakness ID: 250
Abstraction: Base

Structure: Simple

 Presentation Filter:

▼ Description

The software 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.

▼ Relationships

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

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

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

Nature	Type	ID	Name
MemberOf		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: Language-Independent (*Undetermined Prevalence*)

Technologies

 Class: Mobile (*Undetermined Prevalence*)

▼ 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.	

▼ 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 <code>setuid</code> 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 <code>setuid</code> 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	<code>setuid</code> root program allows creation of arbitrary files through command line argument.
CVE-2007-3931	Installation script installs some programs as <code>setuid</code> 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.

▼ 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 Federal Desktop Core Configuration (FDCC) [[REF-199](#)] or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

▼ 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

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. Both [CWE-272](#) and [CWE-250](#) are in active use by the community. The "least privilege" phrase has multiple interpretations.

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.

▼ 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

▼ 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://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html>>.

[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. "Federal Desktop Core Configuration". <<http://nvd.nist.gov/fdcc/index.cfm>>.

[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	7 Pernicious Kingdoms	
► Modifications		
► Previous Entry Names		

CWE-290: Authentication Bypass by Spoofing

Weakness ID: 290
Abstraction: Base

Structure: Simple






 Presentation Filter:

▼ Description

This attack-focused weakness is caused by improperly implemented authentication schemes that are subject to spoofing attacks.

▼ Relationships

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

Nature	Type	ID	Name
ChildOf		287	Improper Authentication
ParentOf		291	Reliance on IP Address for Authentication
ParentOf		293	Using Referer Field for Authentication
ParentOf		350	Reliance on Reverse DNS Resolution for a Security-Critical Action
PeerOf		602	Client-Side Enforcement of Server-Side Security

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

Nature	Type	ID	Name
MemberOf		1211	Authentication Errors

📘 Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

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

▼ Modes Of Introduction

Phase	Note
Architecture and Design	COMMISSION: This weakness refers to an incorrect design related to an architectural security tactic.
Implementation	

▼ Common Consequences

Scope	Impact	Likelihood
Access Control	Technical Impact: <i>Bypass Protection Mechanism; Gain Privileges or Assume Identity</i> This weakness can allow an attacker to access resources which are not otherwise accessible without proper authentication.	

▼ Demonstrative Examples

Example 1

The following code authenticates users.

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

```
String sourceIP = request.getRemoteAddr();
if (sourceIP != null && sourceIP.equals(APPROVED_IP)) {
    authenticated = true;
}
```

The authentication mechanism implemented relies on an IP address for source validation. If an attacker is able to spoof the IP, they may be able to bypass the authentication mechanism.

Example 2

Both of these examples check if a request is from a trusted address before responding to the request.

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

```

sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));

while (1) {
    memset(msg, 0x0, MAX_MSG);
    clien = sizeof(cli);
    if (inet_ntoa(cli.sin_addr)==getTrustedAddress()) {
        n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clien);
    }
}

```

Example Language: **Java**

(bad code)

```

while(true) {
    DatagramPacket rp=new DatagramPacket(rData,rData.length);
    outSock.receive(rp);
    String in = new String(p.getData(),0, rp.getLength());
    InetAddress clientIPAddress = rp.getAddress();
    int port = rp.getPort();

    if (isTrustedAddress(clientIPAddress) & secretKey.equals(in)) {
        out = secret.getBytes();
        DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port); outSock.send(sp);
    }
}

```

The code only verifies the address as stored in the request packet. An attacker can spoof this address, thus impersonating a trusted client.

Example 3

The following code samples use a DNS lookup in order to decide whether or not an inbound request is from a trusted host. If an attacker can poison the DNS cache, they can gain trusted status.

Example Language: **C**

(bad code)

```

struct hostent *hp;struct in_addr myaddr;
char* tHost = "trustme.example.com";
myaddr.s_addr=inet_addr(ip_addr_string);

hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
    trusted = true;
} else {
    trusted = false;
}

```

Example Language: **Java**

(bad code)

```

String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
    trusted = true;
}

```

Example Language: **C#**

(bad code)

```

IPAddress hostIPAddress = IPAddress.Parse(RemoteIpAddress);
IPEndPoint hostInfo = Dns.GetHostByAddress(hostIPAddress);
if (hostInfo.HostName.EndsWith("trustme.com")) {
    trusted = true;
}

```

IP addresses are more reliable than DNS names, but they can also be spoofed. Attackers can easily forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

▼ Observed Examples

Reference	Description
CVE-2009-1048	VOIP product allows authentication bypass using 127.0.0.1 in the Host header.

▼ Memberships

Nature	Type	ID	Name
MemberOf	V	884	CWE Cross-section
MemberOf	C	956	SFP Secondary Cluster: Channel Attack
MemberOf	C	1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures

▼ Notes

Relationship

This can be resultant from insufficient verification.

▼ Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication bypass by spoofing

▼ Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-21	Exploitation of Trusted Identifiers
CAPEC-22	Exploiting Trust in Client
CAPEC-459	Creating a Rogue Certification Authority Certificate
CAPEC-461	Web Services API Signature Forgery Leveraging Hash Function Extension Weakness
CAPEC-473	Signature Spoof
CAPEC-476	Signature Spoofing by Misrepresentation
CAPEC-59	Session Credential Falsification through Prediction
CAPEC-60	Reusing Session IDs (aka Session Replay)
CAPEC-667	Bluetooth Impersonation AttackS (BIAS)
CAPEC-94	Adversary in the Middle (AiTM)

▼ References

[REF-62] Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". Chapter 3, "Spoofing and Identification", Page 72. 1st Edition. Addison Wesley. 2006.

▼ Content History

▼ Submissions		
Submission Date	Submitter	Organization
2006-07-19	PLOVER	
► Modifications		

CWE-266: Incorrect Privilege Assignment

Weakness ID: 266
Abstraction: Base

Structure: Simple







 Presentation Filter:

▼ Description


A product incorrectly assigns a privilege to a particular actor, creating an unintended sphere of control for that actor.

▼ Relationships

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

Nature	Type	ID	Name
ChildOf		269	Improper Privilege Management
ParentOf		9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods
ParentOf		520	.NET Misconfiguration: Use of Impersonation
ParentOf		556	ASP.NET Misconfiguration: Use of Identity Impersonation
ParentOf		1022	Use of Web Link to Untrusted Target with window.opener Access
CanAlsoBe		286	Incorrect User Management

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

Nature	Type	ID	Name
MemberOf		265	Privilege Issues

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

▼ Modes Of Introduction

Phase	Note
Architecture and Design	
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.

▼ Applicable Platforms

📘 Languages

 Class: Language-Independent (*Undetermined Prevalence*)

▼ Common Consequences

Scope	Impact	Likelihood
Access Control	Technical Impact: <i>Gain Privileges or Assume Identity</i> A user can access restricted functionality and/or sensitive information that may include administrative functionality and user accounts.	

▼ Demonstrative Examples

Example 1

The following example demonstrates the weakness.

Example Language: C
(bad code)

```
seteuid(0);
/* do some stuff */

seteuid(getuid());
```

Example 2

The following example demonstrates the weakness.

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

```

AccessController.doPrivileged(new PrivilegedAction() {
    public Object run() {
        // privileged code goes here, for example:
        System.loadLibrary("awt");
        return null;
        // nothing to return
    }
}

```

Example 3

This application sends a special intent with a flag that allows the receiving application to read a data file for backup purposes.

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

```

Intent intent = new Intent();
intent.setAction("com.example.BackupUserData");
intent.setData(file_uri);
intent.addFlags(FLAG_GRANT_READ_URI_PERMISSION);
sendBroadcast(intent);

```

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

```

public class CallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Uri userData = intent.getData();
        stealUserData(userData);
    }
}

```

Any malicious application can register to receive this intent. Because of the FLAG_GRANT_READ_URI_PERMISSION included with the intent, the malicious receiver code can read the user's data.

▼ Observed Examples

Reference	Description
CVE-1999-1193	untrusted user placed in unix "wheel" group
CVE-2005-2741	Product allows users to grant themselves certain rights that can be used to escalate privileges.
CVE-2005-2496	Product uses group ID of a user instead of the group, causing it to run with different privileges. This is resultant from some other unknown issue.
CVE-2004-0274	Product mistakenly assigns a particular status to an entity, leading to increased privileges.

▼ Potential Mitigations**Phases: Architecture and Design; Operation**

Very carefully manage the setting, management, and handling of privileges. Explicitly manage trust zones in the software.

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.

▼ Weakness Ordinalities

Ordinality	Description
Resultant	<i>(where the weakness is typically related to the presence of some other weaknesses)</i>

▼ Affected Resources

- System Process

▼ Memberships

Nature	Type	ID	Name
MemberOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)
MemberOf	V	884	CWE Cross-section
MemberOf	C	901	SFP Primary Cluster: Privilege
MemberOf	C	1149	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 15. Platform Security (SEC)
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design

▼ Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incorrect Privilege Assignment
The CERT Oracle Secure Coding Standard for Java (2011)	SEC00-J		Do not allow privileged blocks to leak sensitive information across a trust boundary
The CERT Oracle Secure Coding Standard for Java (2011)	SEC01-J		Do not allow tainted variables in privileged blocks

▼ References

[REF-76] Sean Barnum and Michael Gegick. "Least Privilege". 2005-09-14. <<https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html>>.

▼ Content History

▼ Submissions		
Submission Date	Submitter	Organization
2006-07-19	PLOVER	
▶ Modifications		

CWE-346: Origin Validation Error

Weakness ID: 346
Abstraction: Base

Structure: Simple

 Presentation Filter:

▼ Description

The software does not properly verify that the source of data or communication is valid.

▼ Relationships

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

Nature	Type	ID	Name
ChildOf	P	284	Improper Access Control
ChildOf	🟢	345	Insufficient Verification of Data Authenticity
ParentOf	🟡	1385	Missing Origin Validation in WebSockets
PeerOf	🟢	451	User Interface (UI) Misrepresentation of Critical Information

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

Nature	Type	ID	Name
MemberOf	🔴	1214	Data Integrity Issues
MemberOf	🔴	417	Communication Channel Errors

📘 Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (CWE-1003)

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

▼ Modes Of Introduction

Phase	Note
Architecture and Design	
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.

▼ Applicable Platforms

📘 Languages

 Class: Language-Independent (*Undetermined Prevalence*)

▼ Common Consequences

Scope	Impact	Likelihood
Access Control	Technical Impact: Gain Privileges or Assume Identity; Varies by Context	
Other	An attacker can access any functionality that is inadvertently accessible to the source.	

▼ Demonstrative Examples

Example 1

This Android application will remove a user account when it receives an intent to do so:

 Example Language: **Java**

(bad code)

```

IntentFilter filter = new IntentFilter("com.example.RemoveUser");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);

public class DeleteReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int userID = intent.getIntExtra("userID");
        destroyUserData(userID);
    }
}

```

```
}
}
```

This application does not check the origin of the intent, thus allowing any malicious application to remove a user. Always check the origin of an intent, or create an allowlist of trusted applications using the manifest.xml file.

Example 2

These Android and iOS applications intercept URL loading within a WebView and perform special actions if a particular URL scheme is used, thus allowing the Javascript within the WebView to communicate with the application:

Example Language: **Java**

(bad code)

```
// Android
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url){
    if (url.substring(0,14).equalsIgnoreCase("examplescheme:")){
        if(url.substring(14,25).equalsIgnoreCase("getUserInfo")){
            writeToView(view, UserData);
            return false;
        }
        else{
            return true;
        }
    }
}
```

Example Language: **Objective-C**

(bad code)

```
// iOS
-(BOOL) webView:(UIWebView *)exWebView shouldStartLoadWithRequest:(NSURLRequest *)exRequest
navigationType:(UIWebViewNavigationType)exNavigationType
{
    NSURL *URL = [exRequest URL];
    if ([[URL scheme] isEqualToString:@"exampleScheme"]){
        {
            NSString *functionString = [URL resourceSpecifier];
            if ([functionString hasPrefix:@"specialFunction"]){
                {
                    // Make data available back in webview.
                    UIWebView *webView = [self writeToView:[URL query]];
                }
            }
        }
        return NO;
    }
    return YES;
}
```

A call into native code can then be initiated by passing parameters within the URL:

Example Language: **JavaScript**

(attack code)

```
window.location = examplescheme://method?parameter=value
```

Because the application does not check the source, a malicious website loaded within this WebView has the same access to the API as a trusted site.




Observed Examples

Reference	Description
CVE-2000-1218	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning
CVE-2005-0877	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning
CVE-2001-1452	DNS server caches glue records received from non-delegated name servers
CVE-2005-2188	user ID obtained from untrusted source (URL)
CVE-2003-0174	LDAP service does not verify if a particular attribute was set by the LDAP server
CVE-1999-1549	product does not sufficiently distinguish external HTML from internal, potentially dangerous HTML, allowing bypass using special strings in the page title. Overlaps special elements.
CVE-2003-0981	product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS.

Weakness Ordinalities

Ordinality	Description
Primary	(where the weakness exists independent of other weaknesses)
Resultant	(where the weakness is typically related to the presence of some other weaknesses)

▼ Memberships

Nature	Type	ID	Name
MemberOf		949	SFP Secondary Cluster: Faulty Endpoint Authentication
MemberOf		1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures
MemberOf		1382	ICS Operations (& Maintenance): Emerging Energy Technologies

▼ Notes

Maintenance

This entry has some significant overlap with other CWE entries and may need some clarification. See terminology notes.

Terminology

The "Origin Validation Error" term was originally used in a 1995 thesis [REF-324]. Although not formally defined, an issue is considered to be an origin validation error if either (1) "an object [accepts] input from an unauthorized subject," or (2) "the system [fails] to properly or completely authenticate a subject." A later section says that an origin validation error can occur when the system (1) "does not properly authenticate a user or process" or (2) "does not properly authenticate the shared data or libraries." The only example provided in the thesis (covered by OSVDB:57615) involves a setuid program running command-line arguments without dropping privileges. So, this definition (and its examples in the thesis) effectively cover other weaknesses such as [CWE-287](#) (Improper Authentication), [CWE-285](#) (Improper Authorization), and [CWE-250](#) (Execution with Unnecessary Privileges). There appears to be little usage of this term today, except in the SecurityFocus vulnerability database, where the term is used for a variety of issues, including web-browser problems that allow violation of the Same Origin Policy and improper validation of the source of an incoming message.

▼ Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Origin Validation Error

▼ Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-111	JSON Hijacking (aka JavaScript Hijacking)
CAPEC-141	Cache Poisoning
CAPEC-142	DNS Cache Poisoning
CAPEC-160	Exploit Script-Based APIs
CAPEC-21	Exploitation of Trusted Identifiers
CAPEC-384	Application API Message Manipulation via Man-in-the-Middle
CAPEC-385	Transaction or Event Tampering via Application API Manipulation
CAPEC-386	Application API Navigation Remapping
CAPEC-387	Navigation Remapping To Propagate Malicious Content
CAPEC-388	Application API Button Hijacking
CAPEC-510	SaaS User Request Forgery
CAPEC-59	Session Credential Falsification through Prediction
CAPEC-60	Reusing Session IDs (aka Session Replay)
CAPEC-75	Manipulating Writeable Configuration Files
CAPEC-76	Manipulating Web Input to File System Calls
CAPEC-89	Pharming

▼ References

[REF-324] Taimur Aslam. "A Taxonomy of Security Faults in the UNIX Operating System". 1995-08-01. <<http://cwe.mitre.org/documents/sources/ATaxonomyofSecurityFaultsintheUNIXOperatingSystem%5BAslam95%5D.pdf>>.

▼ Content History

▼ Submissions		
Submission Date	Submitter	Organization
2006-07-19	PLOVER	
► Modifications		