Content

[1. CWE-787 Out-of-bounds Write 3](#_Toc152667072)

[Decision 3](#_Toc152667073)

[Decision 3](#_Toc152667074)

[3. CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') 3](#_Toc152667075)

[Decision 3](#_Toc152667076)

[4. CWE-416 Use After Free 3](#_Toc152667077)

[Decision 3](#_Toc152667078)

[5. CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') 3](#_Toc152667079)

[Decision 4](#_Toc152667080)

[6. CWE-20 Improper Input Validation 4](#_Toc152667081)

[Decision 4](#_Toc152667082)

[7. CWE-125: Out-of-bounds Read 4](#_Toc152667083)

[Decision 4](#_Toc152667084)

[8. CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') 4](#_Toc152667085)

[Decision 4](#_Toc152667086)

[9. CWE-352: Cross-Site Request Forgery (CSRF) 4](#_Toc152667087)

[Decision 4](#_Toc152667088)

[10. CWE-434: Unrestricted Upload of File with Dangerous Type 4](#_Toc152667089)

[Decision 5](#_Toc152667090)

[11. CWE-862: Missing Authorization 5](#_Toc152667091)

[Decision 5](#_Toc152667092)

[12. CWE-476: NULL Pointer Dereference 5](#_Toc152667093)

[Decision 5](#_Toc152667094)

[13. CWE-287: Improper Authentication 5](#_Toc152667095)

[Decision 5](#_Toc152667096)

[14. CWE-190: Integer Overflow or Wraparound 5](#_Toc152667097)

[Decision 5](#_Toc152667098)

[15. CWE-502: Deserialization of Untrusted Data 5](#_Toc152667099)

[Suggestion: 5](#_Toc152667100)

[Decision 6](#_Toc152667101)

[16. CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection') 6](#_Toc152667102)

[Decision 6](#_Toc152667103)

[17. CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer 6](#_Toc152667104)

[Decision 6](#_Toc152667105)

[18. CWE-798: Use of Hard-coded Credentials 6](#_Toc152667106)

[Decision 7](#_Toc152667107)

[19. CWE-918: Server-Side Request Forgery (SSRF) 7](#_Toc152667108)

[Decision 7](#_Toc152667109)

[20. CWE-306: Missing Authentication for Critical Function 7](#_Toc152667110)

[Decision 7](#_Toc152667111)

[21. CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') 7](#_Toc152667112)

[Decision 7](#_Toc152667113)

[22. CWE-269: Improper Privilege Management 7](#_Toc152667114)

[Decision 8](#_Toc152667115)

[23. CWE-94: Improper Control of Generation of Code ('Code Injection') 8](#_Toc152667116)

[Decision 8](#_Toc152667117)

[24. CWE-863: Incorrect Authorization 8](#_Toc152667118)

[Decision 8](#_Toc152667119)

[25. CWE-276: Incorrect Default Permissions 8](#_Toc152667120)

[Decision 8](#_Toc152667121)

# CWE-787 Out-of-bounds Write

The product writes data past the end, or before the beginning, of the intended buffer.

Typically, this can result in corruption of data, a crash, or code execution. The product may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent write operation then produces undefined or unexpected results.

**Memory Corruption:**

Often used to describe the consequences of writing to memory outside the bounds of a buffer, or to memory that is invalid, when the root cause is something other than a sequential copy of excessive data from a fixed starting location. This may include issues such as incorrect pointer arithmetic, accessing invalid pointers due to incomplete initialization or memory release, etc.

## Decision

It Applies.

1. CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

The product does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users. It is related with XSS.

Cross-site scripting (XSS) vulnerabilities occur when:

1. Untrusted data enters a web application, typically from a web request.
2. The web application dynamically generates a web page that contains this untrusted data.
3. During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTML tags, HTML attributes, mouse events, Flash, ActiveX, etc.
4. A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data.
5. Since the script comes from a web page that was sent by the web server, the victim's web browser executes the malicious script in the context of the web server's domain.
6. This effectively violates the intention of the web browser's same-origin policy, which states that scripts in one domain should not be able to access resources or run code in a different domain.

There are three main kinds of XSS:

* **Type 1: Reflected XSS (or Non-Persistent)**- The server reads data directly from the HTTP request and reflects it back in the HTTP response. Reflected XSS exploits occur when an attacker causes a victim to supply dangerous content to a vulnerable web application, which is then reflected back to the victim and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces a victim to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the victim, the content is executed by the victim's browser.
* **Type 2: Stored XSS (or Persistent)**- The application stores dangerous data in a database, message forum, visitor log, or other trusted data store. At a later time, the dangerous data is subsequently read back into the application and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user. For example, the attacker might inject XSS into a log message, which might not be handled properly when an administrator views the logs.
* **Type 0: DOM-Based XSS**- In DOM-based XSS, the client performs the injection of XSS into the page; in the other types, the server performs the injection. DOM-based XSS generally involves server-controlled, trusted script that is sent to the client, such as Javascript that performs sanity checks on a form before the user submits it. If the server-supplied script processes user-supplied data and then injects it back into the web page (such as with dynamic HTML), then DOM-based XSS is possible.

## Decision

It Applies.

# CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

The product constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component. It is related to SQL Injection.

## Decision

It Applies.

# CWE-416 Use After Free

Avoid referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.

The use of previously-freed memory can have any number of adverse consequences, ranging from the corruption of valid data to the execution of arbitrary code, depending on the instantiation and timing of the flaw. The simplest way data corruption may occur involves the system's reuse of the freed memory. Use-after-free errors have two common and sometimes overlapping causes:

* Error conditions and other exceptional circumstances.
* Confusion over which part of the program is responsible for freeing the memory.

In this scenario, the memory in question is allocated to another pointer validly at some point after it has been freed. The original pointer to the freed memory is used again and points to somewhere within the new allocation. As the data is changed, it corrupts the validly used memory; this induces undefined behavior in the process.

If the newly allocated data happens to hold a class, in C++ for example, various function pointers may be scattered within the heap data. If one of these function pointers is overwritten with an address to valid shellcode, execution of arbitrary code can be achieved.

## Decision

It Applies.

# CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

The product constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component. It is related to command injection in OS. From a weakness standpoint, these variants represent distinct programmer errors. In the first variant, the programmer clearly intends that input from untrusted parties will be part of the arguments in the command to be executed. In the second variant, the programmer does not intend for the command to be accessible to any untrusted party, but the programmer probably has not accounted for alternate ways in which malicious attackers can provide input.

## Decision

It Applies.

# CWE-20 Improper Input Validation

The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly.

## Decision

It Applies.

# CWE-125: Out-of-bounds Read

The product reads data past the end, or before the beginning, of the intended buffer.

Typically, this can allow attackers to read sensitive information from other memory locations or cause a crash. A crash can occur when the code reads a variable amount of data and assumes that a sentinel exists to stop the read operation, such as a NUL in a string. The expected sentinel might not be located in the out-of-bounds memory, causing excessive data to be read, leading to a segmentation fault or a buffer overflow. The product may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent read operation then produces undefined or unexpected results.

## Decision

It Applies.

# CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

The product uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the product does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.

Many file operations are intended to take place within a restricted directory. By using special elements such as ".." and "/" separators, attackers can escape outside of the restricted location to access files or directories that are elsewhere on the system. One of the most common special elements is the "../" sequence, which in most modern operating systems is interpreted as the parent directory of the current location. This is referred to as relative path traversal. Path traversal also covers the use of absolute pathnames such as "/usr/local/bin", which may also be useful in accessing unexpected files. This is referred to as absolute path traversal.

In many programming languages, the injection of a null byte (the 0 or NUL) may allow an attacker to truncate a generated filename to widen the scope of attack. For example, the product may add ".txt" to any pathname, thus limiting the attacker to text files, but a null injection may effectively remove this restriction.

## Decision

It Applies.

# CWE-352: Cross-Site Request Forgery (CSRF)

The web application does not, or cannot, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request. When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XML HTTP Request, etc. and can result in exposure of data or unintended code execution

## Decision

It Applies.

# CWE-434: Unrestricted Upload of File with Dangerous Type

The product allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment. The phrase could be interpreted as the lack of restrictions on the size or number of uploaded files, which is a resource consumption issue. This weakness is caused by missing a security tactic during the architecture and design phase.

## Decision

It Applies.

# CWE-862: Missing Authorization

The product does not perform an authorization check when an actor attempts to access a resource or perform an action.

## Decision

It Applies.

# CWE-476: NULL Pointer Dereference

A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.

## Decision

It Applies.

# CWE-287: Improper Authentication

When an actor claims to have a given identity, the product does not prove or insufficiently proves that the claim is correct. This weakness can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or even execute arbitrary code.

## Decision

It Applies.

# CWE-190: Integer Overflow or Wraparound

The product performs a calculation that can produce an integer overflow or wraparound, when the logic assumes that the resulting value will always be larger than the original value. This can introduce other weaknesses when the calculation is used for resource management or execution control.

## Decision

It Applies.

# CWE-502: Deserialization of Untrusted Data

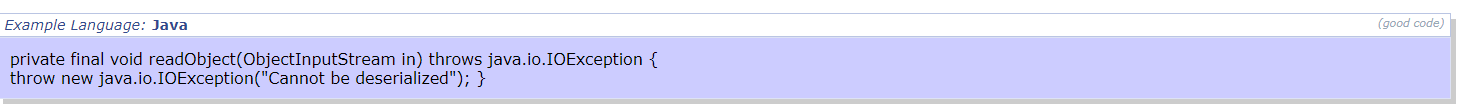
The product deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

It is often convenient to serialize objects for communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it does not use cryptography to protect itself. Furthermore, any cryptography would still be client-side security -- which is a dangerous security assumption.

Data that is untrusted can not be trusted to be well-formed.

When developers place no restrictions on "gadget chains," or series of instances and method invocations that can self-execute during the deserialization process (i.e., before the object is returned to the caller), it is sometimes possible for attackers to leverage them to perform unauthorized actions, like generating a shell.

Suggestion:  
To mitigate this, explicitly define final readObject() to prevent deserialization. An example of this is:



## Decision

It Applies.

# CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')

The product constructs all or part of a command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended command when it is sent to a downstream component.

Command injection vulnerabilities typically occur when:

1. Data enters the application from an untrusted source.

2. The data is part of a string that is executed as a command by the application.

3. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

Many protocols and products have their own custom command language. While OS or shell command strings are frequently discovered and targeted, developers may not realize that these other command languages might also be vulnerable to attacks. Command injection is a common problem with wrapper programs.

## Decision

It Applies.

# CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

The product performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer. As a result, an attacker may be able to execute arbitrary code, alter the intended control flow, read sensitive information, or cause the system to crash.

## Decision

It Applies.

# CWE-798: Use of Hard-coded Credentials

The product contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.



## Decision

It Applies.

# CWE-918: Server-Side Request Forgery (SSRF)

The web server receives a URL or similar request from an upstream component and retrieves the contents of this URL, but it does not sufficiently ensure that the request is being sent to the expected destination. By providing URLs to unexpected hosts or ports, attackers can make it appear that the server is sending the request, possibly bypassing access controls such as firewalls that prevent the attackers from accessing the URLs directly. The server can be used as a proxy to conduct port scanning of hosts in internal networks, use other URLs such as that can access documents on the system (using file://), or use other protocols such as gopher:// or tftp://, which may provide greater control over the contents of requests. It can cause XSPA:Cross Site Port Attack

## Decision

It Applies.

# CWE-306: Missing Authentication for Critical Function

The product does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources.

## Decision

It Applies.

# CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

The product contains a code sequence that can run concurrently with other code, and the code sequence requires temporary, exclusive access to a shared resource, but a timing window exists in which the shared resource can be modified by another code sequence that is operating concurrently.

## Decision

It Applies.

# CWE-269: Improper Privilege Management

The product contains a code sequence that can run concurrently with other code, and the code sequence requires temporary, exclusive access to a shared resource, but a timing window exists in which the shared resource can be modified by another code sequence that is operating concurrently.

The product does not properly assign, modify, track, or check privileges for an actor, creating an unintended sphere of control for that actor.

## Decision

It Applies.

# CWE-94: Improper Control of Generation of Code ('Code Injection')

The product constructs all or part of a code segment using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the syntax or behavior of the intended code segment.

## Decision

It Applies.

# CWE-863: Incorrect Authorization

The product performs an authorization check when an actor attempts to access a resource or perform an action, but it does not correctly perform the check. This allows attackers to bypass intended access restrictions.

## Decision

It Applies.

# CWE-276: Incorrect Default Permissions

The product performs an authorization check when an actor attempts to access a resource or perform an action, but it does not correctly perform the check. This allows attackers to bypass intended access restrictions.

## Decision

It does not apply.