Vulnerability Report for December 2025

Executive Summary

This monthly summary captures what our AI-powered vulnerability analysis engine reviewed in **December 2025**, analyzing **200+** new or updated vulnerabilities across ecosystems and vendors.

This report highlights the **10** high-severity vulnerabilities prioritized by severity, CVSS and potential business impact. Below are this month's key stats: total analyzed, critical/high counts, average CVSS, and KEV (if present).

Σ	C	H	Avg
200+ Total Vulnerabilities	**14** Critical	**46** High	**6.9** Avg CVSS

Prefer the full picture? Visit our vulnerability database to explore everything we analyzed this month.

Table of Contents

Critical Vulnerabilities

- CVE-2025-55182: React2Shell - RSC pre-auth RCE
- CVE-2025-44005: Step CA Has Authorization Bypass in ACME and SCEP Provisi...
- CVE-2025-65854: MineAdmin has an insecure default password
- CVE-2025-65896: assyncmy is vulnerable to SQL injection via crafted dict ...
- CVE-2025-66401: MCP Watch has a Critical Command Injection in cloneRepo a...
- CVE-2025-67489: @vitejs/plugin-rsc Remote Code Execution through unsafe d...
- CVE-2025-67511: Cybersecurity AI (CAI) vulnerable to Command Injection in...
- CVE-2025-67510: Neuron MySQLWriteTool allows arbitrary/destructive SQL wh...

High Vulnerabilities

- CVE-2025-55184: Denial of Service Vulnerability in React Server Components

Medium Vulnerabilities

- CVE-2025-55183: Source Code Exposure Vulnerability in React Server Compon...

CVE-2025-55182 : React2Shell - RSC pre-auth RCE

CRITICAL

View Advisory → GHSA: **GHSA-fv66-9v8q-g76r** • Repo: **facebook/react**

Common Weakness Enumeration (CWE)

`CWE-502` — Deserialization of Untrusted Data

CVSS Score	Severity	Published Date
10.0 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H	Critical	December 03, 2025

AFFECTED PACKAGES

- `react-server-dom-webpack` (npm) — vulnerable: `= 19.0` → patched in `19.0.1`
- `react-server-dom-webpack` (npm) — vulnerable: `>= 19.1.0, < 19.1.2` → patched in `19.1.2`
- `react-server-dom-webpack` (npm) — vulnerable: `= 19.2.0` → patched in `19.2.1`
- `react-server-dom-turbopack` (npm) — vulnerable: `= 19.0` → patched in `19.0.1`
- `react-server-dom-turbopack` (npm) — vulnerable: `>= 19.1.0, < 19.1.2` → patched in `19.1.2`
- `react-server-dom-turbopack` (npm) — vulnerable: `= 19.2.0` → patched in `19.2.1`
- `react-server-dom-parcel` (npm) — vulnerable: `= 19.0` → patched in `19.0.1`
- `react-server-dom-parcel` (npm) — vulnerable: `>= 19.1.0, < 19.1.2` → patched in `19.1.2`
- `react-server-dom-parcel` (npm) — vulnerable: `= 19.2.0` → patched in `19.2.1`

VULNERABILITY ANALYSIS

The vulnerability is a Remote Code Execution (RCE) in React Server Components, caused by insecure deserialization of data sent from the client to the server. An attacker can craft a malicious request to a server action endpoint.

The server-side code uses a function, `decodeReplyFromBusboy` , to parse this request. The core of the vulnerability lies in the `ReactFlightReplyServer.js` file, which contains the logic for deserializing the client's request and executing server-side functions.

The `reviveModel` function recursively reconstructs JavaScript objects from the parsed JSON payload. The original implementation was vulnerable to prototype pollution, allowing an attacker to inject a `__proto__` key in the payload to modify `Object.prototype` .

This prototype pollution is then leveraged by `loadServerReference` , which is responsible for loading and executing the server function requested by the client. This function uses `requireModule` to load the function's code. The `requireModule` function was also vulnerable, as it accessed properties on a module object without checking if they were its own properties. An attacker could exploit the polluted prototype to make `requireModule` execute arbitrary code.

The patch addresses these issues by: 1. Adding `hasOwnProperty` checks in `requireModule` to prevent prototype pollution. 2. Refactoring the entire promise and object resolution logic in `ReactFlightReplyServer.js` , particularly in `reviveModel` and `loadServerReference` , to handle object creation and cycles securely. 3. Adding error handling in `decodeReplyFromBusboy` to prevent crashes from malformed payloads.

IDENTIFIED VULNERABLE FUNCTIONS

`requireModule`

packages/react-server-dom-parcel/src/client/ReactFlightClientConfigBundlerParcel.js

The `requireModule` function is vulnerable to prototype pollution. It accesses a property on `moduleExports` using a key from the `metadata` object, which can be controlled by an attacker. Without the `hasOwnProperty` check, an attacker could provide a key like `__proto__` or `constructor` to access and pollute `Object.prototype` , leading to Remote Code Execution.

`requireModule`

packages/react-server-dom-turbopack/src/client/ReactFlightClientConfigBundlerTurbopack.js

The `requireModule` function is vulnerable to prototype pollution. It accesses a property on `moduleExports` using a key from the `metadata` object, which can be controlled by an attacker. Without the `hasOwnProperty` check, an attacker could provide a key like `__proto__` or `constructor` to access and pollute `Object.prototype` , leading to Remote Code Execution.

`requireModule`

packages/react-server-dom-webpack/src/client/ReactFlightClientConfigBundlerNode.js

The `requireModule` function is vulnerable to prototype pollution. It accesses a property on `moduleExports` using a key from the `metadata` object, which can be controlled by an attacker. Without the `hasOwnProperty` check, an attacker could provide a key like `__proto__` or `constructor` to access and pollute `Object.prototype` , leading to Remote Code Execution.

`requireModule`

packages/react-server-dom-webpack/src/client/ReactFlightClientConfigBundlerWebpack.js

The `requireModule` function is vulnerable to prototype pollution. It accesses a property on `moduleExports` using a key from the `metadata` object, which can be controlled by an attacker. Without the `hasOwnProperty` check, an attacker could provide a key like `__proto__` or `constructor` to access and pollute `Object.prototype` , leading to Remote Code Execution.

`reviveModel`

packages/react-server/src/ReactFlightReplyServer.js

The `reviveModel` function is responsible for deserializing JSON data from the client. The patch shows a change in how properties are assigned to the revived object, specifically with a check for `__proto__` . This indicates that the original implementation was likely vulnerable to prototype pollution, where an attacker could inject a `__proto__` property in the JSON payload to modify the prototype of all objects on the server, leading to RCE.

`loadServerReference`

packages/react-server/src/ReactFlightReplyServer.js

`loadServerReference` is responsible for taking a reference from the client, loading the corresponding server-side code via `requireModule` , and executing it with provided arguments. The extensive refactoring of this function suggests the original implementation had security flaws in how it handled this process, allowing an attacker to control which function is executed and with what arguments, leading to RCE.

`decodeReplyFromBusboy`

packages/react-server-dom-webpack/src/server/ReactFlightDOMServerNode.js

This function is the entry point for processing multipart form data from the client, which can contain the malicious payload. The patch adds `try...catch` blocks around the processing logic (`resolveField`). This suggests that a malformed payload could previously crash the server or bypass security mechanisms by triggering an unhandled exception, which is an important aspect of the overall exploit.

CVE-2025-55184 : Denial of Service Vulnerability in React Server Components

HIGH

View Advisory → GHSA: **GHSA-2m3v-v2m8-q956** • Repo: **facebook/react**

Common Weakness Enumeration (CWE)

`CWE-400` — Uncontrolled Resource Consumption `CWE-502` — Deserialization of Untrusted Data

CVSS Score	Severity	Published Date
7.5 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H	High	December 11, 2025

AFFECTED PACKAGES

- `react-server-dom-parcel` (npm) — vulnerable: `>= 19.0.0, < 19.0.2` → patched in `19.0.2`
- `react-server-dom-turbopack` (npm) — vulnerable: `>= 19.0.0, < 19.0.2` → patched in `19.0.2`
- `react-server-dom-webpack` (npm) — vulnerable: `>= 19.0.0, < 19.0.2` → patched in `19.0.2`
- `react-server-dom-parcel` (npm) — vulnerable: `>= 19.1.0, < 19.1.3` → patched in `19.1.3`
- `react-server-dom-parcel` (npm) — vulnerable: `>= 19.2.0, < 19.2.2` → patched in `19.2.2`
- `react-server-dom-turbopack` (npm) — vulnerable: `>= 19.1.0, < 19.1.3` → patched in `19.1.3`
- `react-server-dom-turbopack` (npm) — vulnerable: `>= 19.2.0, < 19.2.2` → patched in `19.2.2`
- `react-server-dom-webpack` (npm) — vulnerable: `>= 19.1.0, < 19.1.3` → patched in `19.1.3`
- `react-server-dom-webpack` (npm) — vulnerable: `>= 19.2.0, < 19.2.2` → patched in `19.2.2`

VULNERABILITY ANALYSIS

The vulnerability is a Denial of Service in React Server Components, categorized under CWE-400 (Uncontrolled Resource Consumption) and CWE-502 (Deserialization of Untrusted Data). The root cause lies in the improper handling of specially crafted payloads during deserialization on the server.

The analysis of the patch commit `ac60e735881439ae074706e194fd22781805411f` reveals two main attack vectors that were fixed: 1. **Promise/Thenable Cycles**: The primary vector was the possibility of creating a cyclic promise (a "thenable" that resolves to itself) within the serialized data. The deserialization logic in `packages/react-server/src/ReactFlightReplyServer.js` would enter an infinite loop when trying to resolve such a promise. The functions `ReactPromise.then` , `fulfillReference` , and `getOutlinedModel` were directly involved in this looping behavior. The patch mitigates this by adding explicit cycle detection within `ReactPromise.then` and removing the vulnerable looping logic from `fulfillReference` and `getOutlinedModel` . Furthermore, checks were added to `loadServerReference` and `createModel` to prevent the creation of malicious promise-like objects from properties named "then".

1. **Resource Re-initialization**: The second vector involved the repeated initialization of resources like TypedArrays and Streams. The functions `parseTypedArray` , `parseReadableStream` , and `parseAsyncIterable` could be tricked into processing the same resource ID multiple times. This would lead to unnecessary resource allocation and processing, eventually causing a Denial of Service. The patch addresses this by adding checks to ensure that a resource ID is only initialized once.

The identified vulnerable functions are all part of the server-side deserialization pipeline in `ReactFlightReplyServer.js` . An unauthenticated attacker could send a malicious payload to a server using React Server Components, triggering these vulnerabilities and causing the server to become unresponsive.

IDENTIFIED VULNERABLE FUNCTIONS

`ReactPromise.then`

packages/react-server/src/ReactFlightReplyServer.js

The original implementation of the `then` method on the internal `ReactPromise` object did not have a mechanism to detect cyclic promise chains. A malicious actor could craft a serialized payload containing a promise that resolves to itself, causing the `then` method to enter an infinite recursive loop, which would lead to a Denial of Service (DoS) due to a stack overflow.

`fulfillReference`

packages/react-server/src/ReactFlightReplyServer.js

This function is responsible for resolving object references from the serialized payload. The vulnerable version of the code contained a loop designed to unwrap nested promises. A malicious payload could exploit this by creating a reference to a promise that forms a cycle. This would cause the function to loop indefinitely, resulting in a Denial of Service (DoS).

`getOutlinedModel`

packages/react-server/src/ReactFlightReplyServer.js

Similar to `fulfillReference` , this function processes parts of the serialized model and was vulnerable to an infinite loop. A crafted payload with a cyclically referenced promise would cause this function to loop indefinitely while trying to resolve the promise, leading to a Denial of Service (DoS).

`loadServerReference`

packages/react-server/src/ReactFlightReplyServer.js

This function loads server references during the deserialization process. The vulnerability existed because it did not check for the property name 'then'. This allowed an attacker to craft a payload that could create a malicious promise-like object when the framework attempted to access a property named 'then', which could then be used to trigger a promise cycle DoS vulnerability.

`createModel`

packages/react-server/src/ReactFlightReplyServer.js

This function is part of the model creation process during deserialization. Similar to `loadServerReference` , it lacked a check for the 'then' property. This oversight allowed a crafted payload to create a malicious promise-like object, which could then be used to trigger a Denial of Service (DoS) attack.

`parseTypedArray`

packages/react-server/src/ReactFlightReplyServer.js

This function is responsible for parsing typed arrays from the payload. The vulnerability allowed for the re-initialization of a typed array by referencing the same ID multiple times. This could lead to excessive resource allocation and processing, ultimately causing a Denial of Service (DoS).

`parseReadableStream`

packages/react-server/src/ReactFlightReplyServer.js

This function parses readable streams and was vulnerable to a re-initialization attack, similar to `parseTypedArray` . A malicious payload could cause the same stream to be processed multiple times, leading to resource exhaustion and a Denial of Service (DoS).

`parseAsyncIterable`

packages/react-server/src/ReactFlightReplyServer.js

This function parses async iterables and was vulnerable to the same re-initialization attack as `parseReadableStream` and `parseTypedArray` . This allowed an attacker to trigger a Denial of Service (DoS) through resource exhaustion by forcing the same iterable to be processed multiple times.

CVE-2025-55183 : Source Code Exposure Vulnerability in React Server Components

MEDIUM

View Advisory → GHSA: **GHSA-925w-6v3x-g4j4** • Repo: **facebook/react**

Common Weakness Enumeration (CWE)

`CWE-497` — Exposure of Sensitive System Information to an Unauthorized Control Sphere

`CWE-502` — Deserialization of Untrusted Data

CVSS Score	Severity	Published Date
5.3 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	Medium	December 11, 2025

AFFECTED PACKAGES

- `react-server-dom-parcel` (npm) — vulnerable: `>= 19.0.0, < 19.0.2` → patched in `19.0.2`
- `react-server-dom-turbopack` (npm) — vulnerable: `>= 19.0.0, < 19.0.2` → patched in `19.0.2`
- `react-server-dom-webpack` (npm) — vulnerable: `>= 19.0.0, < 19.0.2` → patched in `19.0.2`
- `react-server-dom-parcel` (npm) — vulnerable: `>= 19.1.0, < 19.1.3` → patched in `19.1.3`
- `react-server-dom-parcel` (npm) — vulnerable: `>= 19.2.0, < 19.2.2` → patched in `19.2.2`
- `react-server-dom-turbopack` (npm) — vulnerable: `>= 19.1.0, < 19.1.3` → patched in `19.1.3`
- `react-server-dom-turbopack` (npm) — vulnerable: `>= 19.2.0, < 19.2.2` → patched in `19.2.2`
- `react-server-dom-webpack` (npm) — vulnerable: `>= 19.1.0, < 19.1.3` → patched in `19.1.3`
- `react-server-dom-webpack` (npm) — vulnerable: `>= 19.2.0, < 19.2.2` → patched in `19.2.2`

VULNERABILITY ANALYSIS

No vulnerability analysis available.

CVE-2025-44005 : Step CA Has Authorization Bypass in ACME and SCEP Provisioners

CRITICAL

View Advisory → GHSA: **GHSA-h8cp-697h-8c8p** • Repo: **smallstep/certificates**

Common Weakness Enumeration (CWE)

`CWE-306` — Missing Authentication for Critical Function

CVSS Score	Severity	Published Date
10.0 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N	Critical	December 03, 2025

AFFECTED PACKAGES

- `github.com/smallstep/certificates` (go) — vulnerable: `<= 0.28.4` → patched in `0.29.0`

VULNERABILITY ANALYSIS

The vulnerability is an authorization bypass in Step CA's ACME and SCEP provisioners. The root cause is a flaw in the token authorization logic within the `authority.UseToken` function. This function was responsible for validating and marking tokens as used but failed to properly handle cases where a provisioner, such as ACME or SCEP, does not support token-based authentication.

The `GetTokenID` method on these provisioners would return an error, but the calling `UseToken` function would ignore it, allowing the request to proceed without proper authorization. This could be exploited to bypass authorization checks for operations like certificate renewal and revocation.

The fix addresses this in several ways: 1. The `GetTokenID` methods for ACME and SCEP provisioners were modified to return a specific error, `ErrTokenFlowNotSupported` . 2. The `UseToken` function was updated to recognize this specific error and reject any request where the provisioner does not support the token flow. 3. An additional check was added to the `Revoke` function to ensure the serial number in the request matches the one in the token, preventing unauthorized revocations.

The identified vulnerable functions are those central to the token authorization flow that either contained the flawed logic or called the vulnerable function, as well as the `Revoke` function which was missing a critical validation check.

IDENTIFIED VULNERABLE FUNCTIONS

`authority.UseToken`

authority/authorize.go

The original implementation of `UseToken` incorrectly handled errors from `prov.GetTokenID(token)` . It would only proceed if `err == nil` , effectively ignoring cases where a provisioner did not support token-based authentication (like ACME and SCEP), which would return an error. This allowed authorization to be bypassed. The patched version explicitly checks for `provisioner.ErrTokenFlowNotSupported` and denies the request, closing the security hole.

`authority.authorizeToken`

authority/authorize.go

This function is a caller of the vulnerable `UseToken` function. By passing a token from an unsupported provisioner (like ACME or SCEP), an attacker could bypass the authorization check within this function.

`authority.AuthorizeRenewToken`

authority/authorize.go

This function, used for certificate renewal, was also a caller of the vulnerable `UseToken` function. This could have allowed an attacker to renew certificates without proper authorization.

`authority.Revoke`

authority/tls.go

The `Revoke` function was missing a check to ensure that the serial number in the revocation request matched the serial number in the authorization token (`claims.Subject`). This vulnerability could allow an attacker to revoke a certificate they do not own by providing a valid token for a different certificate.

`provisioner.ACME.GetTokenID`

authority/provisioner/acme.go

The ACME provisioner did not support token-based authentication, but it returned a generic error that was ignored by the `UseToken` function. The patch changes this to return a specific `ErrTokenFlowNotSupported` error, which is now correctly handled by the updated `UseToken` function. While not vulnerable in itself, this function was a key component of the bypass.

`provisioner.SCEP.GetTokenID`

authority/provisioner/scep.go

Similar to the ACME provisioner, the SCEP provisioner returned a generic error for the unsupported `GetTokenID` method. This was a contributing factor to the authorization bypass vulnerability in `UseToken` . The fix ensures it returns a specific, actionable error.

CVE-2025-65854 : MineAdmin has an insecure default password

CRITICAL

View Advisory → GHSA: **GHSA-x6mh-4w8x-p34v** • Repo: **mineadmin/mineadmin**

Common Weakness Enumeration (CWE)

`CWE-94` — Improper Control of Generation of Code ('Code Injection')

CVSS Score	Severity	Published Date
9.8 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	Critical	December 12, 2025

AFFECTED PACKAGES

- `mineadmin/mineadmin` (composer) — vulnerable: `<= 3.0.9` → patched in **None**

VULNERABILITY ANALYSIS

The vulnerability in MineAdmin is a critical issue that stems from two interconnected weaknesses. First, the `Mine\Command\InstallProjectCommand::initUserData` function sets a hardcoded, weak default password ('admin123') for the 'superAdmin' account during the installation process. This allows an attacker to easily gain privileged access to the application's administrative interface.

Once an attacker has administrative access, they can exploit a second vulnerability in the scheduled tasks feature. The `Mine\Crontab\MineCrontab::execute` function, which handles the execution of scheduled tasks, has a feature to evaluate and run arbitrary PHP code. By creating a new scheduled task of type 'Eval' and providing a malicious payload in the task's target, an attacker can achieve Remote Code Execution on the server. The combination of a default administrative password and a feature that allows for code execution makes this a critical vulnerability.

IDENTIFIED VULNERABLE FUNCTIONS

`Mine\Command\InstallProjectCommand::initUserData`

src/Command/InstallProjectCommand.php

This function is called during the installation of MineAdmin and creates a super administrator account with a hardcoded default password 'admin123'. An attacker can use these default credentials to gain administrative access to the application.

`Mine\Crontab\MineCrontab::execute`

src/Crontab/MineCrontab.php

This function is responsible for executing scheduled tasks. When a task of type '3' (Eval) is executed, this function takes the 'target' field of the task and executes it as PHP code using `eval()` . An attacker with access to the admin panel (e.g., through the default password) can create a malicious scheduled task to achieve Remote Code Execution (RCE). The provided PoC `eval('system("ping -c 4 xxxxxxxx.dnslog.cn");');` confirms this execution vector.

CVE-2025-65896 : assyncmy is vulnerable to SQL injection via crafted dict keys

CRITICAL

View Advisory → GHSA: **GHSA-qhqw-rrw9-25rm** • Repo: **long2ice/asyncmy**

Common Weakness Enumeration (CWE)

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

CVSS Score	Severity	Published Date
9.8 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	Critical	December 02, 2025

AFFECTED PACKAGES

- `asyncmy` (pip) — vulnerable: `<= 0.2.10` → patched in **None**

VULNERABILITY ANALYSIS

The vulnerability exists in the `asyncmy.converters` module, specifically within the `escape_dict` function. This function is responsible for preparing Python dictionaries for use in SQL queries. The analysis of the source code for `asyncmy/converters.pyx` reveals that while dictionary values are properly escaped, the keys are not. The function `escape_dict` iterates over the dictionary items and applies escaping only to the values, leaving the keys untouched. This flaw can be exploited if an attacker can supply a dictionary with specially crafted keys, which will then be incorporated directly into the SQL query, leading to SQL injection. The `escape_item` function is also identified as vulnerable because it acts as a dispatcher, and when it receives a dictionary, it directs it to the flawed `escape_dict` function. The vulnerability is confirmed by the GitHub issue #134 in the `long2ice/asyncmy` repository, which explicitly points out this flaw. Since there is no patched version available yet, the code from the latest version serves as evidence of the vulnerability.

IDENTIFIED VULNERABLE FUNCTIONS

`escape_dict`

asyncmy/converters.pyx

The function `escape_dict` processes a dictionary to be used in a SQL query. It iterates through the dictionary's items and escapes the values using `escape_item` . However, it fails to escape the keys. If an attacker can control the keys of the dictionary, they can inject malicious SQL, leading to a SQL injection vulnerability.

`escape_item`

asyncmy/converters.pyx

The `escape_item` function is the entry point for escaping various data types. When it receives a dictionary, it calls the vulnerable `escape_dict` function. Therefore, any call to `escape_item` with a dictionary whose keys are controlled by an attacker can lead to SQL injection.

CVE-2025-66401 : MCP Watch has a Critical Command Injection in cloneRepo allows Remote Code Execution (RCE) via malicious URL

CRITICAL

View Advisory → GHSA: **GHSA-27m7-ffhq-jqrm** • Repo: **kapilduraphe/mcp-watch**

Common Weakness Enumeration (CWE)

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

CVSS Score	Severity	Published Date
9.8 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	Critical	December 02, 2025

AFFECTED PACKAGES

- `mcp-watch` (npm) — vulnerable: `<= 0.1.2` → patched in **None**

VULNERABILITY ANALYSIS

The vulnerability is a classic command injection in the `cloneRepo` function of the `MCPScanner` class. The provided patch in commit `e7da78c5b4b960f8b66c254059ad9ebc544a91a6` clearly shows the remediation. The vulnerable code was using `execSync` to shell out to `git clone` , and the user-controlled `url` was concatenated directly into the command string. This is a textbook example of command injection. The fix replaces `execSync` with `spawnSync` , which correctly handles arguments by passing them as an array to the executable, thus preventing shell interpretation of special characters in the URL. The vulnerable function is therefore `MCPScanner.cloneRepo` as it is the function that contains the dangerous call to `execSync` with unsanitized user input.

IDENTIFIED VULNERABLE FUNCTIONS

`MCPScanner.cloneRepo`

src/scanner/McpScanner.ts

The `cloneRepo` method in the `MCPScanner` class was vulnerable to command injection. It used `child_process.execSync` to construct and execute a `git clone` command, directly embedding the user-provided `url` into the command string without proper sanitization. This allowed an attacker to inject arbitrary shell commands by crafting a malicious URL, leading to remote code execution.

CVE-2025-67489 : @vitejs/plugin-rsc Remote Code Execution through unsafe dynamic imports in RSC server function APIs on development server

CRITICAL

View Advisory → GHSA: **GHSA-j76j-5p5g-9wfr** • Repo: **vitejs/vite-plugin-react**

Common Weakness Enumeration (CWE)

`CWE-94` — Improper Control of Generation of Code ('Code Injection')

CVSS Score	Severity	Published Date
9.8 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	Critical	December 08, 2025

AFFECTED PACKAGES

- `@vitejs/plugin-rsc` (npm) — vulnerable: `<= 0.5.5` → patched in `0.5.6`

VULNERABILITY ANALYSIS

The vulnerability lies in the unsafe use of dynamic `import()` with user-controlled input within the `@vitejs/plugin-rsc` package, specifically affecting the development server. The analysis of the patch commit `fe634b58210d0a4a146a7faae56cd71af3bb9af4` reveals that the core issue is in the anonymous `load` functions defined inside an `initialize` function in both `packages/plugin-rsc/src/rsc.tsx` and `packages/plugin-rsc/src/ssr.tsx` . These functions are responsible for dynamically loading modules based on an `id` parameter.

The vulnerability description states that server function APIs like `loadServerAction` , `decodeReply` , and `decodeAction` are the entry points. These APIs internally use the vulnerable `load` function to resolve and import modules. An attacker could craft a request to the development server where the module `id` is a malicious `data:` URL containing arbitrary JavaScript. The `import(id)` call would then execute this code with Node.js privileges.

The patch mitigates this by introducing a validation mechanism. Before the dynamic import is executed, another `import()` is performed on a virtual module (`virtual:vite-rsc/reference-validation?...`). This triggers a new Vite plugin (`rsc:reference-validation`) that checks if the provided `id` is a legitimate and known server or client reference. If the `id` is not found in the list of valid references, the plugin throws an error, preventing the subsequent unsafe import and stopping the attack.

IDENTIFIED VULNERABLE FUNCTIONS

`initialize.load`

packages/plugin-rsc/src/rsc.tsx

This anonymous `load` function, configured via `setRequireModule` , is called by server function APIs like `decodeAction` and `decodeReply` . It performs a dynamic import using a module `id` that can be controlled by an attacker through HTTP requests. Before the patch, there was no validation of the `id` , allowing a malicious `data:` URL to be imported, leading to remote code execution on the development server.

`initialize.load`

packages/plugin-rsc/src/ssr.tsx

Similar to the `load` function in `rsc.tsx` , this anonymous function in `ssr.tsx` is used for dynamically loading client-side modules during server-side rendering. It was vulnerable to the same dynamic import attack, where a malicious `id` could lead to code execution. The patch adds a validation step to ensure the `id` corresponds to a known client reference.

CVE-2025-67511 : Cybersecurity AI (CAI) vulnerable to Command Injection in run_ssh_command_with_credentials Agent tool

CRITICAL

View Advisory → GHSA: **GHSA-4c65-9gqf-4w8h** • Repo: **aliasrobotics/cai**

Common Weakness Enumeration (CWE)

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

CVSS Score	Severity	Published Date
9.7 CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H	Critical	December 09, 2025

AFFECTED PACKAGES

- `cai-framework` (pip) — vulnerable: `<= 0.5.9` → patched in **None**

VULNERABILITY ANALYSIS

The security vulnerability is a command injection flaw within the `run_ssh_command_with_credentials` function located in the file `src/cai/tools/command_and_control/sshpass.py` . The root cause of the vulnerability is the improper construction of a shell command. The function directly concatenates user-supplied input for `username` , `host` , and `port` into the `ssh_command` string without adequate sanitization. While the `password` and `command` parameters had some form of escaping, the others did not. This allows an attacker to inject arbitrary shell commands by crafting malicious values for the `username` or `host` parameters, as demonstrated in the Proof of Concept. The fixing commit `09ccb6e0baccf56c40e6cb429c698750843a999c` addresses this by applying `shlex.quote()` to all user-controlled parts of the command, ensuring they are treated as single, non-executable string literals by the shell, and by adding validation for the `port` parameter.

IDENTIFIED VULNERABLE FUNCTIONS

`run_ssh_command_with_credentials`

src/cai/tools/command_and_control/sshpass.py

The function is vulnerable to command injection because it uses f-string formatting to construct a shell command, and it fails to properly sanitize the `username` , `host` , and `port` parameters. An attacker can provide a malicious payload in these parameters, which will be executed by the shell. The patch mitigates this by using `shlex.quote()` to escape all user-provided inputs before they are passed to the shell.

CVE-2025-67510 : Neuron MySQLWriteTool allows arbitrary/destructive SQL when exposed to untrusted prompts (agent “footgun”)

CRITICAL

View Advisory → GHSA: **GHSA-898v-775g-777c** • Repo: **neuron-core/neuron-ai**

Common Weakness Enumeration (CWE)

`CWE-250` — Execution with Unnecessary Privileges `CWE-284` — Improper Access Control

CVSS Score	Severity	Published Date
9.4 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:H	Critical	December 09, 2025

AFFECTED PACKAGES

- `neuron-core/neuron-ai` (composer) — vulnerable: `<= 2.8.11` → patched in `2.8.12`

VULNERABILITY ANALYSIS

The vulnerability described is an arbitrary SQL execution in the `MySQLWriteTool` component of the `neuron-core/neuron-ai` package. The analysis began by identifying the vulnerable and patched versions, which are `<= 2.8.11` and `2.8.12` respectively. By comparing the git tags for these versions, a set of commits related to the patch was identified.

The core of the vulnerability lies in the `__invoke` method of the `MySQLWriteTool` class, which is responsible for executing the provided SQL query. The security patches revolve around the introduction and enhancement of a `validate` method, which is called by `__invoke` to check the query against a list of forbidden, high-risk SQL statements.

Commit `74a19aa0a94cbe982cf28fb1452d45ddca7a76c1` modifies the `__invoke` function, confirming that it calls the `validate` function. This makes `__invoke` a key function in the exploitation process, as it's the entry point for the malicious query.

Commit `44bab85d92bf162898ee48d0bcef6ba0d29b59c9` modifies the `validate` function itself. This function contains the security logic to deny dangerous queries. Therefore, any attempt to exploit this vulnerability would involve this function being called.

Both `__invoke` and `validate` from the `Neuron\Tools\Toolkits\MySQL\MySQLWriteTool` class are therefore identified as the vulnerable functions that would appear in a runtime profile during an exploitation attempt.

IDENTIFIED VULNERABLE FUNCTIONS

`Neuron\Tools\Toolkits\MySQL\MySQLWriteTool::__invoke`

src/Tools/Toolkits/MySQL/MySQLWriteTool.php

This function is the entry point for executing SQL queries. It takes a user-provided query string and executes it. The vulnerability lies in the fact that, prior to the patch, this function did not properly validate the query, allowing arbitrary and destructive SQL statements to be executed. The patch adds a call to the `validate` function to check for forbidden keywords.

`Neuron\Tools\Toolkits\MySQL\MySQLWriteTool::validate`

src/Tools/Toolkits/MySQL/MySQLWriteTool.php

This function is responsible for validating the SQL query to prevent malicious statements from being executed. In the vulnerable version, this function was either non-existent or its checks were insufficient. The patch introduces or strengthens this function to include a denylist of dangerous SQL keywords like `DROP` , `TRUNCATE` , `ALTER` , etc. An attempt to exploit the vulnerability would involve a query that is processed by this function.

This report was generated by Miggo Security's AI-powered vulnerability analysis engine.