S-SDLC SOCKET DEV ALERT LISTING

Brief Overview

This document serves as a structured guide from Socket.Dev Alerts, outlining potential vulnerabilities and suggested mitigations for developers when selecting and ensuring the security of packages and their dependencies within their applications. It is crucial to refer to this guide throughout the development process, while remaining vigilant as alerts are constantly evolving in real-time. Stay proactive in seeking the latest updates and alerts.

Defining a Risk

Risk is the possibility of damage when a threat takes advantage of a weakness, which calls for strong security measures to protect personal data and preserve its sensitive information and maintain its integrity, confidentiality, and availability. Given our collaboration with the Department of Defense (DoD), due to the sensitive nature of the mission and the serious consequences of a security breach, we must be alert to external security considerations, such as foreign languages on package documentation.

This document categorizes security vulnerabilities into four severity levels: critical, high, medium, and low. Each category indicates the possible impact and urgency of resolving the identified risk. It's vital to note that the severity classification does not automatically decide whether a problem should be halted or continued. Many vulnerabilities, regardless of severity level, can be effectively addressed with the right methods. We will describe each severity level and examine alternative solutions that can be adopted to manage and resolve these vulnerabilities, ensuring that risks are minimized while security is maintained.

Severity Levels for Vulnerabilities

- ▲ Critical
- ◆ High
- ■ Medium
- ● Low

Security Risks

▲ Hidden Behavior or code
Mitigation: Any code that might be hidden or bypasses normal installation procedures, it's essential to thoroughly review and understand its purpose, potential security implications, integration impact.

▲ Possible Typosquat Attack

Mitigation: Ensure the correct spelling of packages, as sometimes they can appear to be spelled correctly at a quick glance even if they are slightly off.

◆ Potential Vulnerability
Mitigation: Potential vulnerability alerts should try to be avoided, if necessary, apply appropriate mitigations with proper documentation.

◆ Network or Environment Access
Mitigation: Network and environment access alerts should be restricted to the correct SCOPE in that part of the application, ensuring data protection and security is well maintained.

■ AI Detected Risk
Mitigation: Review the AI detected security risk details page noting the vulnerability and where the alert is located inside the package. Mitigate potential risks and ensure the package's security and integrity are thoroughly evaluated.

■ External Install Features
Mitigation: A package that requires external installation and execution of code should be subjected to a comprehensive review to verify its installation procedures and the SCOPE of its access and operations.

● Code Execution:
Mitigation: Any code that is executed is essential to review understan its purpose, potential security implications, integration impact.

Package Integrity and Reliability

◆ Remote or External Dependency
Mitigation: Any remote or external dependency should undergo thorough review to verify its installation procedures and the SCOPE of its access and operations.

◆ Unstable Ownership
Mitigation: Alerts about ownership should be reviewed and carefully selected choosing an unstable or nonexistent author can possibly lead to a lower quality package.

■ Package and Dependency Inconsistency
Mitigation: Thoroughly review inconsistencies in packages and their dependencies to ensure version compatibility, and ensure updates are regularly helping maintain stability.

■ Deprecated or Unmaintained Package
Mitigation: Deprecated or Unmaintained packages should go through rigorous review before use, signaling that the package is no longer actively maintained and may have unresolved vulnerabilities due to lack of updates.

License and Legal Usage

- ■ Explicitly Unlicensed Item
- ■ Nonpermissive License
- ● Legal notice
- ● (Experimental) SPDX and License issues

Mitigation for all License and Legal Usage:

First the license (or legal notice) should be examined, one can do this by going to the hosted page and investigating the license, then if the issue is greater than an SPDX disjunction it should be communicated with a supervisor to provide guidance on being able to use the package, things to consider are:

Is it possible to use a functionally similar package without the issue? Can the function of the package be created with code?
Can the function of the package be avoided?

Can the issue associated with the package be accepted?

Ensure that the package and the license issue associated with the package are documented if used.

Malware and Vulnerabilities

- ▲ Known Malware
- ▲ Critical CVE
- ◆ High CVE
- ■ Medium CVE
- ● Low CVE

Mitigation for all Malware and Vulnerabilities Usage:

Both known malware and critical CVEs (Common Vulnerabilities and Exposures) should always be avoided. If a CVE is encountered, it needs to be investigated, you can do this by visiting the GitHub page linked to the CVE through Socket. After the investigation, communicate with the associated supervisor to find a suitable solution that should be determined based on the specific vulnerability. The solution could be one of the following:

Avoiding the vulnerability: This means taking steps to completely avoid the risk associated with the vulnerability.

Addressing the vulnerability directly: This involves implementing controls to prevent the vulnerability from being exploited.

Accepting the vulnerability: In some cases, it might be decided that the vulnerability will be accepted, usually due to cost or other factors.

Ensure that the CVE is documented with the associated solution and package.