Secure by Design Alert: Eliminating OS Command Injection Vulnerabilities

Malicious Actors Use OS Command Injection Vulnerabilities to Compromise Systems

Operating system (OS) command injection vulnerabilities are a preventable class of vulnerability in software products. Software manufacturers can eliminate them at the source by taking a secure by design approach. Despite this fact, OS command injection vulnerabilities continue to surface, allowing adversaries to exploit them to cause harm. CISA and FBI are releasing this Secure by Design Alert in response to recent well-publicized threat actor campaigns that exploited OS command injection defects in network edge devices (CVE-2024-20399, CVE-2024-3400, CVE-2024-21887) to target and compromise users. These vulnerabilities allowed unauthenticated malicious actors to remotely execute code on network edge devices.

Despite widespread knowledge and documentation of the OS command injection vulnerabilities over the past two decades, along with the availability of effective mitigations, software manufacturers have continued to develop products with this defect, which puts customers at risk.

OS command injection vulnerabilities arise when manufacturers fail to properly validate and sanitize user input when constructing commands to execute on the underlying OS. Designing and developing software that trusts user input without proper validation or sanitization can allow threat actors to execute malicious commands, putting customers at risk.

CISA and FBI urge CEOs and other business leaders at technology manufacturers to request their technical leaders to analyze past occurrences of this class of defect and develop a plan to eliminate them in the future.

To further prevent these vulnerabilities, technical leaders should:

- Ensure software uses functions that generate commands in safer ways by preserving the intended syntax of the command and its arguments,[1]
- Review their threat models,
- Use modern component libraries,
- Conduct code reviews,
- And implement aggressive adversarial product testing to ensure the quality and security of their code throughout the development lifecycle.[2]

Secure By Design Lessons to Learn

Products that are secure by design reasonably protect against malicious cyber actors exploiting the most common and dangerous classes of product defects[3]. Incorporating security at the outset—beginning in the design phase and continuing through development, release, and updates—reduces the burden on customers and risk to the public. OS command injection vulnerabilities have long been preventable by clearly separating user input from the contents of a command. Despite this finding, OS command injection vulnerabilities—many of which result from CWE-78—are still a prevalent class

[1] Input Validation Cheat Sheet. OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
[2] Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce. (n.d.). Secure Software Development Framework | CSRC | CSRC. https://csrc.nist.gov/Projects/ssdf
[3] CWE - 2023 CWE Top 25 most dangerous software weaknesses. (n.d.). https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html

of vulnerability. CISA has recently added CVE-2024-20399, CVE-2024-3400, and CVE-2024-21887 into the KEV Catalog, which documents vulnerabilities exploited in the wild. **Note:** CWE-78 is the child weakness of CWE-77 and is related to several other weaknesses.

How Can Software Manufacturers Prevent OS Command Injection Vulnerabilities?

During the design and development of a software product, developers should take steps to prevent OS command injection vulnerabilities at scale including, but not limited to:[4]

- Whenever possible, use built-in library functions that separate commands from their arguments instead of constructing raw strings that are fed into a general-purpose system command.
- Use input parameterization to keep data separate from commands; validate and sanitize all user-supplied input.
- Limit. the parts of commands constructed by user input to only what is necessary.

Example: In Python, a developer who would like to create a folder given a user's input should use the `os.mkdir()` function rather than invoking a command.[5] If such a built-in library function is not available, the developer should invoke the command in a manner that separates the command's arguments from the OS command itself—for example, by sanitizing the user's input and then invoking `subprocess.run(["mkdir", user_input])`. Software manufacturers can enforce this process by writing rules at time of pull requests to disallow risky command invocation, such as, banning `os.system`, `subprocess.run` when supplied with a string (as opposed to an array), and `subprocess.run` when supplied with the `shell=True` argument.[6]

Secure By Design Principles to Follow

CISA and FBI encourage manufacturers to learn how to protect their products from falling victim to OS command injection exploits and other preventable malicious activity by reviewing the three principles laid out in the joint guidance Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Secure by Design Software.

Principle 1: Take Ownership of Customer Security Outcomes

Software manufacturers should take ownership of their customer's security outcomes by eliminating OS command injection vulnerabilities from their products. There are key security areas manufacturers should invest in to protect their customers as well as the public. These include providing safe building blocks for their developers to ensure that a single error does not compromise the data of millions of users. The cycle of vulnerability detection, mitigation, and patch deployment for vulnerabilities that have been understood for years is not a lasting approach to security. Effective mechanisms to prevent classes of vulnerability at scale are available and software manufacturers should implement them as early in the development cycle as possible. Adopting standard best practices, such as the guidance listed above, can help manufacturers root out OS command injection vulnerabilities at the source, as opposed to relying on customers to apply fixes. Manufacturers should also implement automated mechanisms that prevent their software from using unsafe functions.

Additionally, senior executives at software manufacturers must take accountability for the security of their customers starting by regularly testing and conducting code reviews to determine product susceptibility to exploitation. The Open Web Application Security Project (OWASP) and other entities provide guidance on testing methods with available techniques.

---

[4] OS Command Injection Defense - OWASP Cheat Sheet Series. (n.d.). https://cheatsheetseries.owasp.org/cheatsheets/OS_Command_Injection_Defense_Cheat_Sheet.html#defense-option-3-parameterization-in-conjunction-with-input-validation; A03 Injection - OWASP Top 10:2021. (n.d.).

[5] Ensure that the arguments to os.mkdir() do not create a path traversal vulnerability. See previous Secure by Design Alert: https://www.cisa.gov/resources-tools/resources/secure-design-alert-eliminating-directory-traversal-vulnerabilities-software.

[6] Command injection prevention for Python | Semgrep. (2023, April 12). https://semgrep.dev/docs/cheat-sheets/python-command-injection.

Principle 2: Embrace Radical Transparency and Accountability

Manufacturers should lead with transparency when disclosing product vulnerabilities. To that end, manufacturers should track the vulnerability associated with their products and disclose these to their customers via the CVE program. Manufacturers should ensure that their CVE records are correct and complete. In addition to providing CVEs, it is especially important that manufacturers supply an accurate CWE mapping so the industry can track classes of software defect, and customers can understand areas where a given vendor's development practices may require improvement.[7]

Many, but not all, OS command injection vulnerabilities are the result of CWE-78. As such, manufacturers should identify and document the root causes of OS command injection vulnerabilities and declare it a business goal to work toward eliminating the entire class. Software manufacturers should also maintain a modern vulnerability disclosure program (VDP). **Note:** CISA provides resources to assist organizations in establishing and maintaining a VDP.

Principle 3: Build Organizational Structure and Leadership to Achieve These Goals

Technology manufacturing executives should:

- Give the security of their products the same level of care they give to cost.
- Consider the full picture: that customers, our economy, and our national security are currently bearing the brunt of business decisions to not build security into their products—as illustrated by the recent threat actor campaigns referenced in this Alert.
- Be aware that fully implementing secure by design software development can reduce financial and productivity costs as well as complexity.
- Make the appropriate investments and develop the right incentive structures that promote security as a stated business goal.
- Lead programs to root out entire classes of vulnerability rather than addressing them on a case-by-case basis.
- Establish organizational structures that prioritize proactive measures, such as adopting standard best practices, to root out OS command injection vulnerabilities at the source.
- Ensure their organization conducts reviews to detect common and well-known vulnerabilities, like OS command injection, to determine their susceptibility, and implement the existing effective and documented mitigations.
    - Organizations should conduct these reviews continually to root out classes of vulnerability, as some vulnerabilities may change or develop over time.
    - Executives should request regular updates to assess: (1) the company's progress at identifying recurring classes of vulnerability, (2) the company's progress to eliminate them, and (3) the appropriate resources needed to continue making progress.

Action Item For Software Manufacturers

To demonstrate their commitment to building their products to be secure by design, software manufacturers should take the Secure by Design Pledge. The pledge lays out seven key goals that the signers commit to demonstrating measurable progress towards, including reducing systemic classes of vulnerability like OS command injection.

This Secure by Design Alert is part of an ongoing series that aims to advance industry-wide best practices that eliminate entire classes of vulnerability during the design and development phases of the product development lifecycle. Through the Secure by Design initiative, we seek to foster a cultural shift across the industry by normalizing the development of technology products that are secure to use out of the box. Visit cisa.gov to learn more about the principles of Secure by Design, take the Secure by Design Pledge, and stay informed on the latest Secure by Design Alerts.

[7] Common Weakness Enumeration (CWE) classification identifies classes of software/hardware weaknesses (including vulnerabilities and defects); Common Vulnerabilities and Exposures (CVE) classification identifies and labels unique vulnerabilities in specific software/hardware products.

Disclaimer

The information in this report is being provided “as is” for informational purposes only. The authoring organizations do not endorse any commercial entity, product, company, or service, including any entities, products, or services linked within this document. Any reference to specific commercial entities, products, processes, or services by service mark, trademark, manufacturer, or otherwise, does not constitute or imply endorsement, recommendation, or favoring by the authoring organizations.