Software Supply Chain Failures: A detailed vulnerability study with applicable remediations for PHP-based applications

Dinesh Kumar Mohanty, Dr. Dragan Peraković
Research Scholar, Full-time professor,
Swiss School of Business Management
kiitkp03@gmail.com, dragan@ssbm.ch

***Abstract*—In today's digital age, where organizations move from local to global, they use web applications to automate their processes. Many organizations rely on third-party software to handle payments. However, some still use old software that they can't update, or they use outdated features. These outdated systems can make their applications weak and easy to attack. Software supply chain failure is ranked as 3 in the OWASP Top 10 list in 2025, consisting of 5 sub-vulnerabilities. In this research paper we will be talking about real-time test cases and possible mitigation methodologies. We will discuss possible remediation techniques that can be adopted by the developers when developing the product.**

***Index Terms*—OWASP, supply chain vulnerability, known vulnerability, web application vulnerability, vulnerability exception.**

I. Introduction

Software supply chain failures happen when there are problems in how software is built, shared, or updated; these issues often arise from the use of 3rd-party software, libraries, dependencies, or badly developed code or tools from an external untrusted source that the 1st-party software depends on [1]. You may be at risk if:

- You don't keep track of all the versions of the parts you use, whether on the user's device or on the server.
- The software is unsafe, no longer supported, or not up to date. This applies to the operating system, servers, databases, apps, APIs, and all other parts, environments, and libraries used.
- You don't check for security risks often and don't follow updates or warnings about the parts you use.
- You don't keep a record of changes made in the software process, like updates to development tools, code repositories, sandboxes, and where software is stored. Every step should be noted, especially when something changes.
- You haven't made sure every part of the software process is as safe as it can be, especially by limiting who has access and giving people only the rights they need.
- There's no rule that requires more than one person to review and approve changes before they go live.
- Developers, DevOps, or system admins are allowed to download and use parts of software from places they shouldn't trust for use in live systems.
- You don't update or fix the core parts of your software, frameworks, and tools in a timely way based on the risk. This often happens when updates are only done once a month or quarter, leaving systems exposed for weeks or months before changes are made.
- Software developers don't check if updated or fixed parts of the software work well with the rest of the system.
- Your system has a complex CI/CD process for building and delivering software that uses many parts but doesn't have strong security in those areas compared to the rest of the application.

II. List of Mapped Vulnerabilities

Below are the CWE IDs that are clubbed into this category of OWASP Top 10 vulnerabilities for the year 2025.

Table 1: CWE IDs under Software Supply Chain Failures

CWE Number	CWE Name
CWE-447	Use of Obsolete Function/Unimplemented or Unsupported Feature in UI
CWE-1035	Using Components with Known Vulnerabilities
CWE-1104	Use of Unmaintained Third-Party Components
CWE-1329	Reliance on Component That is Not Updateable
CWE-1395	Dependency on Vulnerable Third-Party Component

III. METHODOLOGY

This is an internal security testing method that is followed for each and every CWE number.

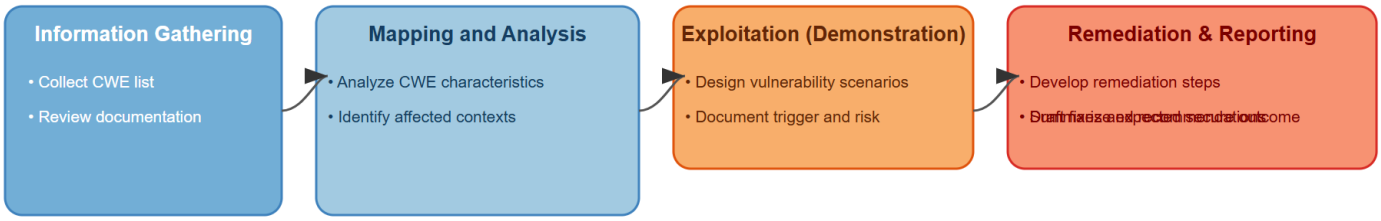

Figure 1: Methodology

1. Information gathering: This is the step where we collected a detailed list of relevant CWEs for the category with a review of specific CWE IDs.

2. Mapping and Analysis: Analysis of each CWE and why it is being exploited.

3. Exploitation (Demonstration): Design vulnerable code for each CWE that developers are left with.

4. Remediation & Reporting: Develop remediation strategies for each vulnerability scenario with fixed code a developer should adopt while developing and its best practices.

IV. RESULT DISCUSSION

1. CWE-447 Use of Obsolete Function

Information:
A UI function for a security feature appears to be supported and gives feedback to the user that suggests that it is supported, but the underlying functionality is not implemented [2].

Vulnerabilities example:

1. Use of mysql_* (e.g., mysql_connect(), mysql_query(), mysql_fetch_assoc()) functions that required older version of PHP
2. Use of ereg_* (e.g., ereg(), ereg_replace()) functions that is used for regular expression matching
3. Use of split() function to split a string into an array based on a regular expression
4. Use of session_register() function to register global variables within session

Exploitation:

```
// Obsolete function usage
$conn = mysql_connect("localhost", "user", "password");
$result = mysql_query("SELECT * FROM users WHERE id = $user_id");
```

Figure 2: CWE-447 Vulnerable code

```
// Example: If $user_id is "1 OR 1=1"
$result = mysql_query("SELECT * FROM users WHERE id = 1 OR 1=1");
```

Figure 3: CWE-447 exploitable code

- If $user_id is directly taken from user input without proper input validation, an attacker could exploit this to execute arbitrary SQL queries.

- Hidden but Still Active Admin Buttons—A PHP application shows an "Admin Settings" button for all users but hides the button using CSS, which points to /admin/settings.php:

  <button id="adminSettings" style="display:none">Admin Settings</button>

  The attacker can gain access to the said URL with user privilege.

- UI Shows a Disabled Feature; Backend Still Accepts Input: <input type="file" disabled> that and the user can manipulate from the Document Object Model (DOM) by removing the disabled option that may allow them to upload malicious files to gain internal server access.
- Unsupported Feature Generates Errors Revealing Sensitive Info— A UI prompts users for "Advanced Analytics" even though the feature is not implemented. When users click the button, the backend throws an exception: Fatal error: Call to undefined function generateReport().

Remediation:

1. Use of the function mysqli_* (e.g., mysqli_connect(), mysqli_query(), mysqli_fetch_assoc()) and PDO_* (PHP Data Objects) extension, which supports prepared statements to mitigate SQL injection.
2. Do not use ereg_*, instead use the preg_*
3. Do not use split(), instead use explode()
4. Use $_SESSION['variable'] = "value"; to register user session
5. Regularly audit your codebase for obsolete functions, especially when migrating between major PHP versions.

2. CWE-1035 Using Components with Known Vulnerabilities

Information:
If an application uses third-party components (say jQuery, PHPMailer, Symfony), libraries, plugins, or frameworks that contain known security vulnerabilities with exploitable techniques using which an attacker could exploit your application [3].

Exploitation:
PHPMailer incorrectly sanitized sender addresses that have been identified as part of CVE-2016-10045 (remote code execution), where an attacker sends a crafted From: parameter containing injected shell commands → PHPMailer processes it → arbitrary code executes on the server [4]. The isMail transport in PHPMailer before 5.2.20 might allow remote attackers to pass extra parameters to the mail command and consequently execute arbitrary code by leveraging improper interaction between the escapeshellarg function and internal escaping performed in the mail function in PHP [4].

How attackers exploit known vulnerable components

1. Checking version numbers, where attackers do information gathering on:
   - /vendor/composer/installed.json
   - /readme.html (WordPress)
   - library-specific headers
2. Public exploit availability—If a version is known to be vulnerable, exploits are often published on:
   - Exploit-DB
   - GitHub
   - Metasploit
3. Direct exploitation—The attacker sends crafted requests known to trigger the CVE.

Zero skill is needed, as attackers simply reuse publicly available PoCs.

Remediation

- Use Composer's security tools
- Pin minimum-allowed versions, not exact versions
- Maintain a Software Bill of Materials (SBOM) and use OWASP Dependency-Check
- Regularly patch server-side components such as PHP, MySQL, OpenSSL, Apache/nginx

3. CWE-1104 Use of Unmaintained Third-Party Components

Information:
The use of third-party components that lack active support or maintenance prevents fixes for bugs, vulnerabilities, or quality issues, which indirectly heightens security risks in applications [5]. Third-party components can be a big security risk because it's hard to know how secure they really are. Companies sometimes use these parts without checking where they come from, how they were made, or how often they get updated [6]. This can mean they end up with old or unsafe code that has known problems, which hackers can use to break into systems [6]. Open-source software is often used a lot because it's free and people work together on it. But since it's developed by many different people, it can be hard to keep security up-to-date [6]. If there are no strict checks and updates are delayed, companies might end up using software that's been tampered with, which can hurt the whole system [6].

- Unpatched CVEs—Old packages often contain security vulnerabilities that will never be fixed.
- Outdated dependencies—Unmaintained packages depend on outdated libraries.
- Broken security mechanisms—old cryptography, broken auth flows, unsafe APIs.
- Forces old PHP versions—Using an abandoned package may block upgrading PHP itself.
- Permanent attack surface—Vulnerability is forever exploitable until the component is removed.

Exploitation:

An example of CWE 1104 is

- Using old phpmailer < 5.2 (no longer maintained)
- Using the deprecated PHP mysql extension (mysql_*)
- Composer package marked as "abandoned"
- Laravel/Symfony apps depending on outdated packages

Attackers commonly follow this process:

1. Identify outdated components

They look for:

- /vendor/composer/installed.json
- Publicly readable composer.lock files
- WordPress plugin version numbers
- GitHub public repos showing composer.json
- Old JS/CSS libraries shipped in PHP CMSes

2. Check for known CVEs

If an outdated library is found, attackers search:

- CVE databases
- Exploit-DB
- GitHub PoCs
- Metasploit modules

3. Trigger the vulnerability

Typical attacks include:

- RCE on old mailer/file-upload libraries
- SQL Injection in old plugins
- XXE/SSRF in old HTTP clients
- Broken authentication bypass in old framework versions

Because the component is unmaintained, the vulnerability never gets fixed. A classic example is:

1. A site uses a legacy API client released in **2013**.
2. The client uses insecure cURL options and verifies no TLS certificate.
3. Attacker performs a **man-in-the-middle attack**.
4. The app trusts attacker-controlled API responses.
5. Attacker injects commands or steals sensitive data.

The weakness existed solely because the component was no longer maintained.

Remediation:

1. Do not continue using components with no updates, and replace them with maintained alternatives.
2. Enable dependency automation to detect outdated or abandoned packages.
3. Maintain SBOM (Software Bill of Materials) to track which components are unmaintained or vulnerable across the supply chain.

4. CWE-1329 Reliance on Component That is Not Updateable

Information:

If the component is discovered to contain a vulnerability or critical bug, but the issue cannot be fixed using an update or patch, then the product's owner will not be able to protect against the issue [7]. The only option might be replacement of the product, which could be too financially or operationally expensive for the product owner [7]. As a result, the inability to patch or update can leave the product open to attacker exploitation or critical operation failures [7]. This weakness can be especially difficult to manage when using ROM, firmware, or similar components that traditionally have had limited or no update capabilities [7].

A component may be non-updateable because:

- The vendor no longer publishes updates.
- Source code is unavailable,
- It's embedded / proprietary / compiled,
- The system architecture blocks updates,
- Updating it would break core functionality.
- No compatible updates exist.

Exploitation:

- Closed-source library embedded into a PHP extension
- PHP framework depends on an old version of a library that cannot be replaced
- Legacy PHP application depends on deprecated "mcrypt"

Sample:

```
function encryptData($data) {
   return mcrypt_encrypt(MCRYPT_RIJNDAEL_256, KEY, $data, MCRYPT_MODE_CBC);
}
```

- Vendor SDK that is discontinued, example: curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);

Remediation:

1. Use OWASP Dependency-Check
2. Replace with updateable alternatives
3. Sandbox or isolate vulnerable components

5. CWE-1395 Dependency on Vulnerable Third-Party Component

Information:

Many products are large enough or complex enough that part of their functionality uses libraries, modules, or other intellectual property developed by third parties who are not the product creator [8]. For example, even an entire operating system might be from a third-party supplier in some hardware products [8]. Whether open or closed source, these components may contain publicly known vulnerabilities that could be exploited by adversaries to compromise the product [8].

This includes both:

- direct dependencies (you install it), and
- transitive dependencies (your dependency depends on it).

Examples are:

- Old PHPMailer Version with RCE
- GuzzleHTTP SSRF Vulnerability
- Framework Plugin Using Outdated JS Library
- PHP image processing library bundled with vulnerable C library
- Old WordPress Plugin with File Upload Vulnerability

Exploitation:

Attackers intentionally trigger exceptions by providing:

1. **Identify component versions**
   - Public composer.lock
   - /vendor/ files exposed
   - WordPress plugin version banners
   - JavaScript library version numbers
   - GitHub repo showing composer.json
2. **Look up CVEs**
   - NVD (National Vulnerability Database)
   - Exploit-DB
   - GitHub PoCs
   - Security advisories
3. **Use known exploits**

Exploiting requires no skill—the vulnerability is already documented. Common attack types:

- RCE (remote code execution)
- SQL injection
- File upload bypass
- Authentication bypass
- XSS from outdated JS components
- SSRF via HTTP client vulnerabilities
- Crypto weaknesses from old crypto libraries

Remediation:

1. Use OWASP Dependency-Check
2. Replace with updateable alternatives
3. Sandbox or isolate vulnerable components

V. CONCLUSION

Modern software systems rely heavily on external components—libraries, plugins, frameworks, and SDKs—making the security of applications inseparable from the security of their supply chain. Weaknesses such as CWE-1104 (Use of Unmaintained Third-Party Components), CWE-1329 (Reliance on Components That Are Not Updateable), and CWE-1395 (Dependency on Vulnerable Third-Party Components) demonstrate how vulnerabilities can arise even when application code itself is sound. These CWEs draw attention to distinct aspects of the same failure pattern: relying on unreliable, out-of-date, or unpatchable components results in persistent, predictable attack surfaces that adversaries frequently take advantage of.

To build resilient systems, organizations must move beyond traditional code review and adopt continuous dependency monitoring, automated security tooling, SBOMs, and strict update policies. Essential practices include replacing abandonware, avoiding vendor lock-in, verifying component health, and upholding a methodical upgrade process. In the end, protecting the software supply chain is a continuous operational duty rather than a one-time event. Proactively addressing these CWEs improves overall application integrity, lowers technical debt, and reduces vulnerability exposure windows.

REFERENCES

[1] "A03 Software Supply Chain Failures - OWASP Top 10:2025 RC1." Accessed: Dec. 01, 2025. [Online]. Available: https://owasp.org/Top10/2025/A03_2025-Software_Supply_Chain_Failures/
[2] "CWE - CWE-447: Unimplemented or Unsupported Feature in UI (4.18)." Accessed: Dec. 01, 2025. [Online]. Available: https://cwe.mitre.org/data/definitions/447.html
[3] "CWE - CWE-1035: CWE CATEGORY: OWASP Top Ten 2017 Category A9 - Using Components with Known Vulnerabilities (4.18)." Accessed: Dec. 01, 2025. [Online]. Available: https://cwe.mitre.org/data/definitions/1035.html
[4] "CVE Record: CVE-2016-10045." Accessed: Dec. 01, 2025. [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2016-10045
[5] "CWE - CWE-1104: Use of Unmaintained Third Party Components (4.18)." Accessed: Dec. 01, 2025. [Online]. Available: https://cwe.mitre.org/data/definitions/1104.html
[6] "Understanding Third-Party Software Risks | RunSafe Security." Accessed: Dec. 01, 2025. [Online]. Available: https://runsafesecurity.com/blog/understanding-third-party-software-risks/
[7] "CWE - CWE-1329: Reliance on Component That is Not Updateable (4.18)." Accessed: Dec. 01, 2025. [Online]. Available: https://cwe.mitre.org/data/definitions/1329.html
[8] "CWE - CWE-1395: Dependency on Vulnerable Third-Party Component (4.18)." Accessed: Dec. 01, 2025. [Online]. Available: https://cwe.mitre.org/data/definitions/1395.html