CRASHTEST **SECURITY**

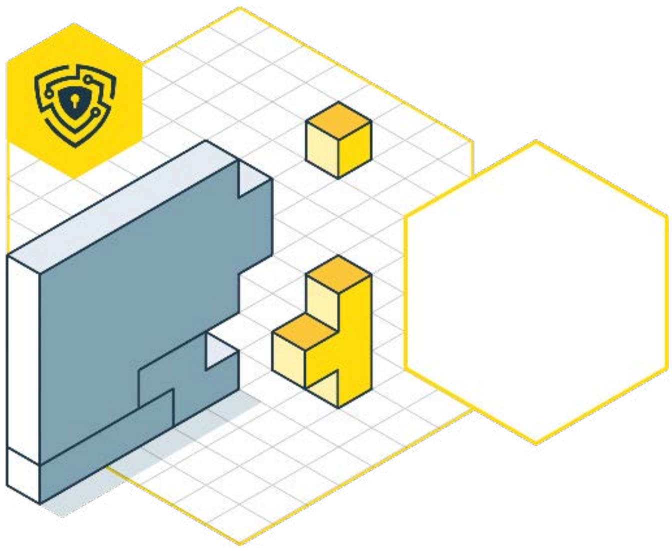
GUIDE TO

VULNERABLE AND OUTDATED COMPONENTS

WHAT ARE THE STEPS TO KEEP YOUR WEB APP OR API SAFE FROM SUCH VULNERABILITY

GUIDE FOR PREVENTING VULNERABLE AND OUTDATED COMPONENTS

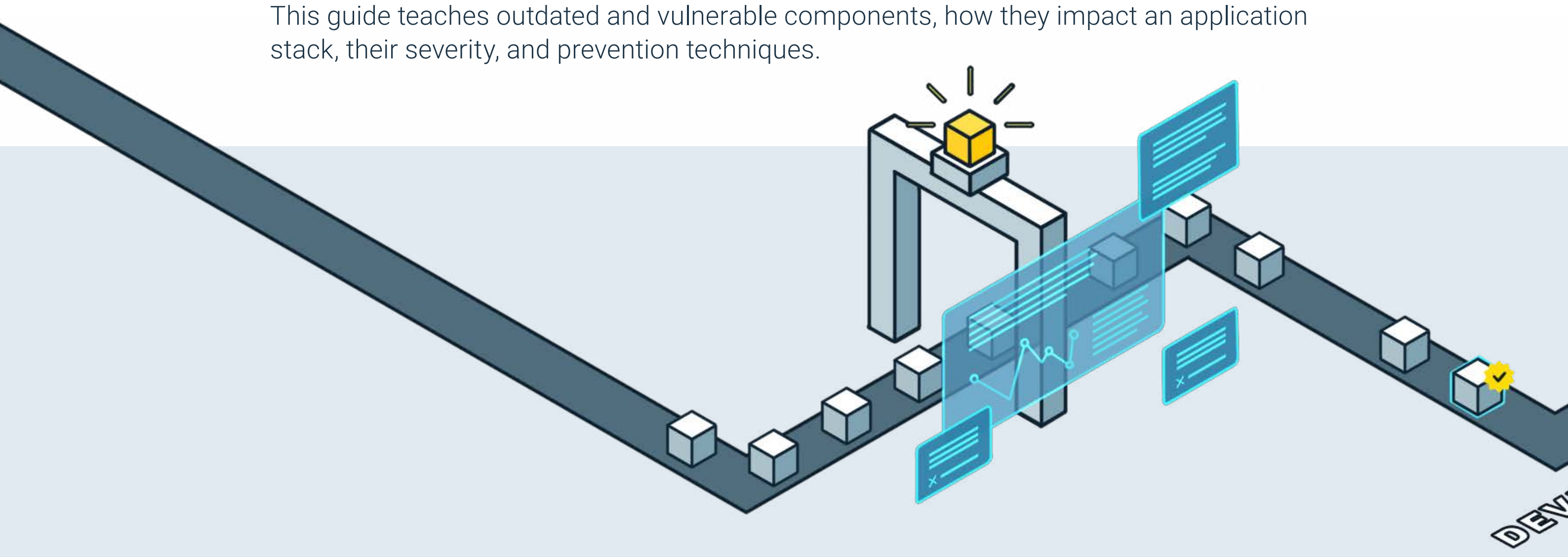
Table of Contents



INTRODUCTION TO THIS GUIDE

Modern applications follow an API-based architecture to enforce agility, where the software is composed of multiple logically distinct components. While exchanging data and functionality through APIs, these components are maintained independently and reused between various applications. In such complex architectures of multiple components, many vulnerabilities often result from the lack of appropriate component validation. Vulnerable and outdated components often introduce security issues unknown to developers, making them soft targets for hackers looking to exploit a vulnerable system.

This guide teaches outdated and vulnerable components, how they impact an application stack, their severity, and prevention techniques.



WHAT ARE VULNERABLE AND OUTDATED COMPONENTS?

Vulnerable components contain security vulnerabilities that attackers exploit to gain access to sensitive data or the entire system. While a complex application stack may have numerous components, web servers, databases, and operating systems are the most common targets of cyber attacks.

On the other hand, outdated components are no longer supported by the vendor. As a result, such components lack security patches addressing uncovered vulnerabilities, making them low-hanging fruit for threat campaigns.

Vulnerabilities over such components may exist for several years before they are discovered, fixed, or disclosed. Sometimes, a vulnerability may not even be identified until it has been exploited. Some of such vulnerabilities include:

INJECTION VULNERABILITIES

Code injection vulnerabilities occur when vulnerable components fail to distinguish between malicious user data and code. To orchestrate injection attacks, hackers identify components that accept user inputs as part of an executable command and then supply malicious code to manipulate the application environment and logic.

As modern applications rely on APIs to exchange data and functionality across components, security misconfigurations such as lack of input sanitation allow threat actors to inject malicious data over API requests, subsequently enabling them to manipulate and obtain unauthorized access to remote resources.

BUFFER OVERFLOWS

Buffer overflow vulnerabilities occur when the amount of data in a buffer memory exceeds the assigned storage capacity. Attackers abuse buffer overflow vulnerabilities to corrupt and manipulate data in adjacent memory buffers since overflowing data flows into these memory units. In instances where attackers gain access to a vulnerable component's memory layout, they can overwrite executable codes stored in buffers with malicious scripts that allow them to cause system crashes, alter security controls or obtain escalated privileges for deeper attacks.

CROSS-SITE SCRIPTING

Cross-site scripting is one of the most critical security risks for modern web applications, as the attack technique enables threat actors to inject malicious scripts into trusted websites.

A common approach is the stored cross-site scripting exploit, where attackers embed malicious scripts into vulnerable versions of third-party components for prolonged misuse of component vulnerabilities. When application components are not scanned or updated regularly, they are prone to become vectors for advanced persistent threats that may eventually affect multiple users and layers of the tech stack.

TYPES OF VULNERABLE AND OUTDATED COMPONENTS

In complex projects, the lack of visibility of third-party components remains a persistent challenge. Such instances lead to the use of vulnerable open-source software and the failure of security patching and regular updates.

Types of vulnerable and outdated components are categorized and mapped to the following three Common Weaknesses and Enumerations:

- **Failure to maintain components** - Flaws that occur due to the presence of unused features along with insufficient patch management and other preventative maintenance controls on modern application development pipelines
- **Out-of-date support systems** - Relying on outdated versions of software packages in the deployment
- **Component misconfiguration** - Third-party package components causing conflict with the overall system configuration

VULNERABLE AND OUTDATED COMPONENTS - SEVERITY LEVEL

Moving up three spots from the 2017 list, the vulnerability of the vulnerable and outdated components is now **ranked number 6** on OWASP's Top 10 list of 2021. The security flaw has an **average weighted exploit of 5 (moderate)** and **an average weighted impact of 5 (moderate)**.

Although the vulnerability has a **low average incidence rate (8.77%)**, it is attributed as a vulnerability that poses serious security risks with an **average coverage of 22.47%**. This is mainly because when outdated components are run on elevated privileges, a successful exploit of its vulnerabilities results in severe compromise of the entire framework.

Consequences of successful attacks on vulnerable and outdated components include:

- Mass disclosure of records intended for private access
- Distributed Denial-of-Service
- Remote Code Execution attacks
- Data integrity violations

IDENTIFY VULNERABLE AND OUTDATED COMPONENTS WITH CRASHTEST SECURITY

Crashtest Security, with its suite of vulnerability scanners, helps security teams expedite the process of detecting and identifying vulnerable components of an application stack. Some scanners offered by Crashtest Security to help identify security issues in vulnerable and outdated components include:

- **OWASP Scanner** - Tests all software packages against the latest OWASP Top 10 vulnerabilities and their mapped CWEs
- **API Vulnerability Scanner** - Checks for safe API implementation and raises an alert if the interface includes misconfigured HTTP directives
- **Command Injection Scanner** - Validates inputs such as user-supplied URLs, message forums, and comment sections to determine whether the application is vulnerable to code injection attacks.
- **Microservices Scanner** - Checks for common vulnerabilities on all containerized services and functionalities
- **XSS Scanner** - Scans software components for cross-site scripting attack vectors

Crashtest Security, with its automated penetration testing, also enables security researchers to simulate an entire threat campaign and assess how attackers may leverage vulnerable and outdated components in the real world. Through actionable security reports, Crashtest Security also helps enhance security posture, rapid assessment, and threat mitigation at the component level.

VULNERABLE AND OUTDATED COMPONENTS - PREVENTION TECHNIQUES

While organizations may choose to adopt several modern tools and practices to prevent outdated component vulnerabilities, here are some techniques that ensure the application framework is built to prevent vulnerable and outdated components at the foundation level:

SOFTWARE COMPOSITION ANALYSIS

Software Composition Analysis (SCA) involves identifying and manifesting every open-source software component in the application code. While performing SCA, it is also recommended to test each of such components for known security flaws and vulnerabilities. An ideal approach to doing so is to look for insecure design flaws that may arise from integrating multiple open-source software packages and assess if those collectively impact the security compliance, posture, and code quality of the environment.

MULTI-FACTOR AUTHENTICATION (MFA)

Since most third-party components run privileged functions, they operate with root permissions. In such instances, when an attacker gets hold of one vulnerable component may result in the attacker obtaining access to other components of the application.

To prevent such interwoven chain abuses, multi-factor authentication helps reduce the attack blast surface while boosting critical access control measures by administering more than one authentication mechanism to grant resource permissions at the component level.

PATCH MANAGEMENT

Unpatched versions of software components used in deployments are a favorite target of most threat campaigns. Such components often contain known vulnerabilities, making it easy for threat actors to orchestrate successful exploits. Patch management ensures that outdated versions of components are either replaced or upgraded with secured versions. A scheduled patch management lifecycle enables security teams to proactively prevent insecure default configurations, data integrity violations, and other random threats introduced with unsecured third-party components.

SECURE DESIGN PATTERNS

Secure design patterns enforce the adoption of best practices to develop secure, reusable applications that are purpose-built to prevent common vulnerabilities and attack vectors. These design patterns help security teams avoid known security flaws in application code and define a roadmap of actions to follow in case of a successful exploit. Embracing secure design principles also helps outline the blueprint of an effective patch management lifecycle while ensuring no vulnerable components are used across any stages of the SDLC.

BEST PRACTICES IN PREVENTING VULNERABLE AND OUTDATED COMPONENTS

Some commonly recommended practices in preventing outdated and vulnerable components include:

DEPLOY WITH A MINIMAL SETUP

Installing unnecessary features and plugins complicates maintaining a robust security posture since an unnecessarily extended setup adds to the manual overhead of monitoring and patching vulnerabilities. New software packages also broaden an application stack‘s attack surface while introducing new system-level conflicts that can be misused for possible attacks. As a recommended practice, it is essential to audit the setup regularly to identify only critical features and plugins installed while discarding unused features to reduce the attack surface.

EMBRACE MICROSERVICE-BASED ARCHITECTURE

Adopting a microservices architecture allows software teams to achieve adequate segregation between different components of a workload. The architecture helps enhance application performance and reduces the blast radius of a successful exploit.

A service-oriented logical separation enables the isolation of insecure components and mitigation of security threats without impacting user experience or application functionality. Microservice-based isolation also protects application servers from direct attacks since software teams can enforce discrete security controls, such as cryptographic algorithms at the service and component levels.

ENFORCE CONTINUOUS MONITORING AND VULNERABILITY SCANNING

Monitoring and vulnerability scanning form a critical quantum of the threat modeling tool stack and are often considered the first line of defense against unknown threats. More importantly, tackling new attack vectors in a changing threat landscape remains a continuous challenge. As a recommended practice, adopting a comprehensive monitoring framework that includes the right tools to evaluate failures and identify security defects as they arise is crucial. Regular monitoring helps detect suspicious activities in real time, such as access control violations and malicious API requests.

SECURING WEB APPLICATIONS AGAINST VULNERABLE AND OUTDATED COMPONENTS WITH CRASHTEST SECURITY

Crashtest Security helps improve security posture and mitigate critical security risks through automated penetration testing and vulnerability scanning. With Crashtest Security, organizations can automate threat modeling to proactively spot security defects and prevent vulnerable and outdated component attacks.

To know more about how Crashtest Security can help reduce the risk of being hacked through vulnerable, third-party components, try a free 14-day trial here.

Start 2-Week Trial for Free

CRASHTEST **SECURITY**

WWW.CRASHTEST-SECURITY.COM