VERACODE

Prevention Guide

Preventing Broken Access Control Vulnerabilities: A developer's guide

Practical steps to keep your web applications and APIs safe and secure

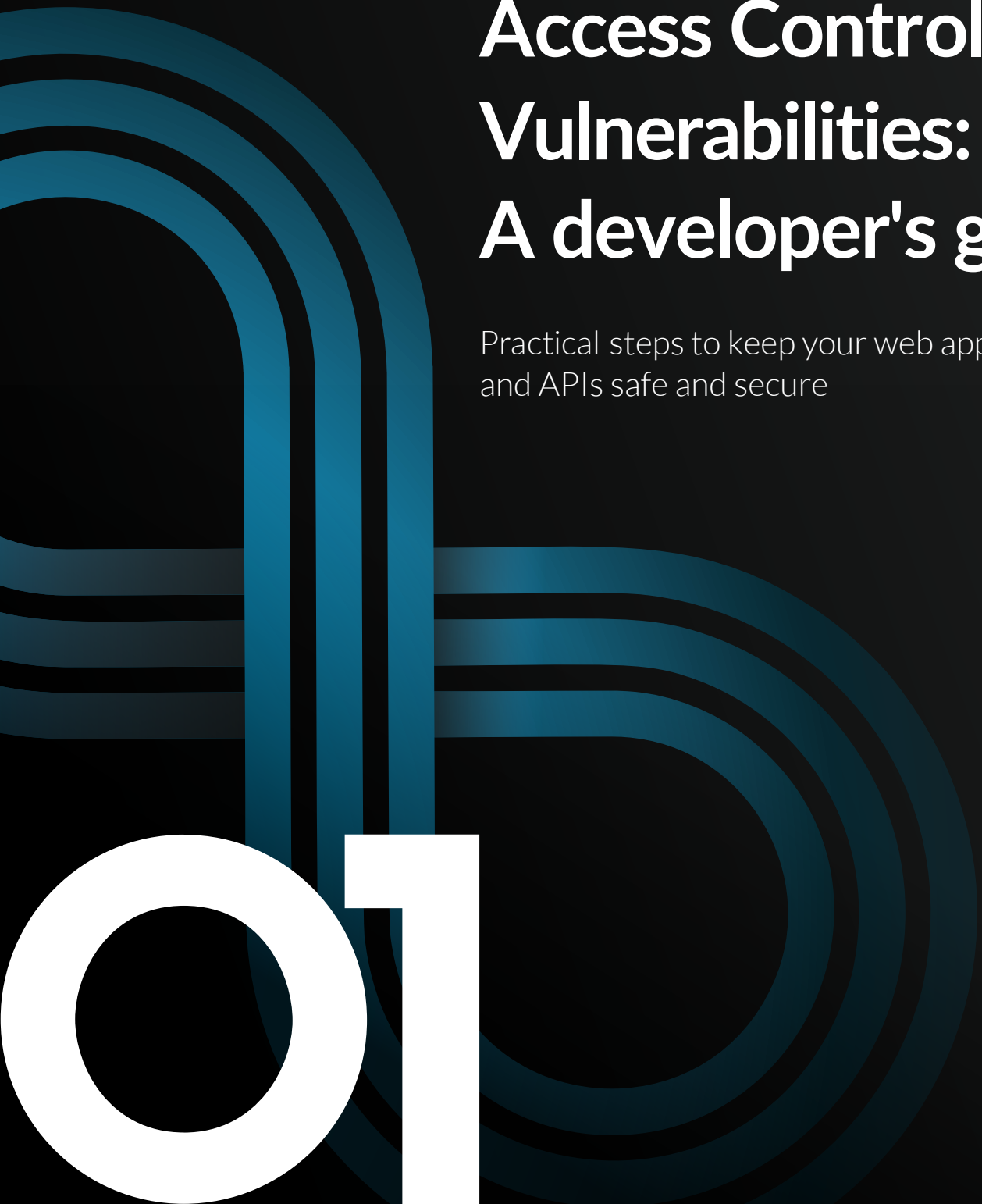

Table of Contents

Securing Web Applications and APIs

Access control is essential in modern web development as it allows for the management of permissions granted to users, processes, and devices for accessing application functions and resources. It also determines the level of access allowed and tracks activities performed by specific entities.

Broken access control vulnerabilities occur when malicious users exploit limitations on their actions or access to objects. Attackers often exploit access control failures to gain unauthorized access to application resources, execute malicious commands, or obtain privileged user permissions. This guide covers broken access control vulnerabilities, the severity of associated attacks, and common prevention techniques.

Understanding Broken Access Control

Access control issues allow unauthorized users to access, modify, or delete resources beyond their intended permissions. Broken access control includes security vulnerabilities commonly exploited to gain higher privilege levels. Creating secure and efficient access control systems can be challenging, especially when dealing with application functions that have evolved without deliberate design.

Developers often overlook how entities access resources, leading to hidden authorization flaws. These flaws are easily discovered and exploited, making them a prime target for common attacks.

Types of Broken Access Control Vulnerabilities

Broken access control vulnerabilities mostly lead to privilege escalation attacks and are characterized by how the malicious user exploits and modifies access rights. The primary forms of access control vulnerabilities include:

Horizontal Privilege Escalation

Horizontal privilege escalation vulnerabilities occur when a user can access the accounts of other regular users with the same level of permissions. Attackers exploit these vulnerabilities to gain access to legitimate user data and carry out various malicious activities, including ransomware attacks, financial fraud, unauthorized money transfers, exposure of sensitive files, and data deletion. Unlike vertical privilege escalation, horizontal attacks do not typically require advanced tools and can be executed with simple steps, such as:

- Modifying the URL's request ID parameter with legitimate user details obtained through some form of social engineering
- Reviewing the application code to identify authentication vulnerabilities at the source code level
- Using third-party code review tools combined with security testing tools
- Enumerating user accounts on Linux machines maintains their hold of the identification process

Context-based Privilege Escalation

A hybrid attack involves a two-step process where the attacker first gains access to regular user accounts and then exploits broken vertical access controls to obtain administrative rights. Context-based privilege escalation attacks exploit business logic to enable users to perform actions that are typically restricted within their security context. Examples of context-based privilege escalation include:

- Leveraging Insecure Direct Object Reference vulnerabilities to access critical resources via user-supplied input
- Using corrupt HTTP referer headers to access functionality and sensitive files beyond their permitted context
- Location-based attacks

Vertical Privilege Escalation

Vertical privilege escalation, also known as privilege elevation, occurs when an unauthorized user gains higher privilege levels, typically admin privileges. This type of attack typically follows an initial breach, as the attacker aims to acquire permissions beyond those of the compromised subject. Compared to horizontal escalation, vertical privilege escalation attacks are more sophisticated, requiring root or kernel-level modifications to gain administrative access.

Once attackers obtain admin rights, they can inject malicious code, disrupt critical business functions, or compromise the availability of essential application resources. Hackers employ various techniques to exploit vertical access controls, including:

- Using the Windows Sysinternals suite to create backdoor administrative users
- Using process injection to mimic administrative functions
- Leveraging directory listing vulnerabilities to disclose information about the access control policy
- Using social engineering for direct access to admin accounts

Severity Level of Broken Access Control

Broken access control is recognized by the Online Web Application Security Project (OWASP) as a prevalent and highly critical vulnerability. It enables attackers to impersonate various user types and gain control over legitimate user accounts. The consequences of a privilege escalation attack can be severe, depending on the specific vulnerability exploited. Some attack scenarios include:

- **Use of insecure IDs**
  Attackers randomly guess the references for users, roles, objects, contents, and functions. Hackers can easily manipulate access control rules to obtain elevated privilege levels if the vulnerable application does not sanitize the supplied user input.

- **Forced browsing**
  Most applications use multiple security checks to grant access to critical resources within the website's backend. Hackers use brute force techniques to bypass the pages running authentication checks, obtaining direct access to web resources.

- **Path transversal attacks**
  The hacker includes a relative path within a URL request, which may grant them direct access to sensitive files.

- **Cache attacks**
  Web browsers store frequently accessed web pages locally within the cache memory. Attackers can obtain cache data and exploit them to replicate administrative functions and orchestrate deeper attacks.

- **File permissions**
  This vulnerability affects files stored on a web server that should not be publicly available. If the server's OS mechanism allows these directories to be readable, attackers can modify application scripts, configuration files, and other default files to cause operational inefficiency.

Cross-site Request Forgery (CSRF) is another vulnerability in the Broken Access Control category. These vulnerabilities are widespread in modern web applications due to the use of predictable parameters for actions. However, detecting CSRF vulnerabilities is relatively straightforward through code analysis and application security testing. The identification technique involves multi-stage payload delivery and is well-documented. As a result, the severity of a CSRF vulnerability is generally considered to have an average level of exploitability.

Impacts of a privilege escalation attack include:

- Takeover of site administration functions
- Modification or deletion of site content
- User account takeover
- Delivery of malicious payloads
- Distributed denial-of-service
- Unauthorized money transfer

Vulnerabilities associated with broken access control fall under several common weaknesses and enumerations, including:

- CWE-23: Relative Path Traversal
- CWE-59: Improper Link Resolution Before File Access (Link Following)
- CWE-201: Exposure of Sensitive Information Through Sent Data
- CWE-219: Storage of File with Sensitive Data Under Web Root
- CWE-275: Permission Issues
- CWE-284: Improper Access Control
- CWE-352: Cross-Site Request Forgery (CSRF)
- CWE-377: Insecure Temporary File
- CWE-402: Transmission of Private Resources into a New Sphere (Resource Leak)
- CWE-425: Direct Request (Forced Browsing)
- CWE-441: Unintended Proxy or Intermediary (Confused Deputy)
- CWE-497: Exposure of Sensitive System Information to an Unauthorized Cont-rol Sphere
- CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory

Detect Broken Access Control Vulnerabilities

Through AI-driven testing and application security testing, the **Veracode Software Security Platform** helps you generate an in-depth analysis of your tech stack's security and access control. The platform includes dynamic application security testing that collectively analyzes for broken access control vulnerabilities such as:

- **Privilege Escalation Scanner**
  A vulnerability scanner built to alert admins of any flaws that may lead to abuse of existing access control mechanisms

- **CSRF Scanner**
  Helps prevent access control attacks using malicious payloads submitted through a trusted normal user

- **URL Fuzzer Scanner**
  Prevents privilege escalation attacks orchestrated through forced browsing or modifying URL request parameters with a relevant admin URL

- **HTTP Header Scanner**
  Prevents the use of modified HTTP referer headers to access critical resources beyond the current security context

- **Fingerprinting Scanner**
  Detect attack surfaces that expose application server implementations, privacy laws, and the web application's access control policy to external domains

Veracode's dynamic application security solution streamlines manual efforts, enabling developers to prioritize secure design and threat mitigation policies. It provides actionable security reports that can be easily shared with cross-functional teams, clients, and executives, fostering a collaborative security approach that spans across all areas of an organization.

Broken Access Control Prevention Techniques

1.

Multi-factor Authentication

Multi-Factor Authentication (MFA) is a zero-trust security approach that employs multiple access control checks to deter hackers from carrying out malicious activities, even if they possess legitimate user credentials. This defense strategy combines various authentication mechanisms to verify a user's identity. During implementation, two or more forms of identification (such as tokens or biometric IDs) are required before granting access. This effectively prevents unauthorized users from exploiting user accounts and mitigates broken access control attempts.

2.

Unit and Integration Tests

To prevent access control flaws in code, development teams should incorporate unit tests throughout the Software Development Life Cycle (SDLC). Unit testing assesses individual modules to verify proper implementation of access control in application code and identify any privilege management vulnerabilities at the class level. Additionally, integration tests encompass a broader scope, including third-party and open source components utilized in the application's construction. This comprehensive testing approach evaluates the overall security posture of the application.

3.

Session Management

Session management is a crucial aspect of secure software development. It involves the proper implementation of session IDs, authentication tokens, and cookies to prevent session hijacking attacks. These measures include forcibly deleting session-related data on the application server when a user logs out. It is also advisable to implement session timeouts that require re-authentication and a new token when a user reconnects to the server after logging out. Additionally, designers and developers should avoid exposing session IDs in URLs, as this could be exploited by attackers for session theft.

4.

Proper Authorization Schemes

Using CSRF tokens in custom request headers provides a stronger defense against CSRF attacks by enforcing the same origin policy restriction. However, it's important to note that modern browsers do not support the transportation of custom headers with Cross-Domain requests by default. Custom headers can only be added in JavaScript or within the script's origin. This stateless CSRF mitigation technique requires no changes to the user experience, making it particularly valuable for securing Representational State Transfer (REST) services.

In addition to scalability and agility, software developers should prioritize robust access controls and security when building applications. Effective control units can be created through authorization models.

- **Discretionary access control**
  This model limits access based on the subject's identity or group membership. Subjects with direct access permission can also assign that permission to other subjects.

- **Mandatory access control**
  In this mechanism, access to resources is secured based on the sensitivity of the information they hold. Administrators label data with security levels and categories, allowing subjects to access only the information that matches their security label.

- **Role-based access control**
  This scheme divides network subjects into roles, with each role having specific access levels. Users are assigned roles that grant them access to the necessary information and functionality for their duties. Roles are typically based on job competency, authority, and responsibility.

- **Attribute-based access control**
  This model permits or denies information exchange based on various properties, such as the requesting entity, requested action, context of information exchange, or the requested resource. Properties used in attribute-based authorization include location, threat level evaluation, time of day, and security measures implemented on the requested resource.

Security Best Practices to Prevent Broken Access Control

1.

Enforce the Law of Least Privilege

One of the initial steps in controlling access privileges is to deny access by default for all public resources. This means that each user should only be granted the permissions necessary to perform specific functions, and no more. This approach, known as the principle of least privilege, follows a zero-trust philosophy in information exchange, ensuring that subjects are not given privileges beyond what is required.

3.

Writing Application Code and Business Logic With Authorization Controls in Mind

Software developers and business designers must ensure their program and business logic includes rules that define access to resources and functionality at the code level. Once the system has authenticated a subject, their privileges to objects should be limited by their roles and identity.

3.

Perform Server-side Controls

To simplify the remote management of access control routines, it is recommended to perform user authentication, input validation, and request processing at the server-side. This approach eliminates the need for traditional keys to enforce privilege decisions and allows for easy tracking of all access attempts and activities.

2.

Use Centralized Authorization Oversight

Implementing access control policies and routines from a centralized location is a recommended approach. This allows for quick application of vulnerability fixes as soon as they are identified. Centralization also eliminates the need for manual effort in applying access control policies to every page containing sensitive files and information.

4.

Test and Audit Access Controls Frequently

In addition to manual testing, it is advisable to use automated scanning tools for continuous monitoring of access control flaws that do not align with an organization's security policy. Continuous testing and vulnerability scanning help evaluate the effectiveness of access control mechanisms and uncover emerging vulnerabilities within the system.

Strengthen Your Web Applications and APIs Against Attacks

Veracode Dynamic Analysis is a dynamic application security testing solution that establishes a continuous and automated process for security testing. It effectively uncovers access control flaws with minimal false positives. This solution seamlessly integrates with popular software stacks and security platforms, allowing teams to initiate dynamic application security testing within minutes

When used in conjunction with Veracode's Software Security Platform, it becomes a comprehensive solution that identifies security risks across your entire tech stack. By combining Dynamic Analysis with Static Analysis and Software Composition Analysis, you can prevent the introduction of new flaws and significantly reduce risk over time in your modern software development processes.

Strengthen your software against broken access control vulnerabilities by trying Veracode Dynamic Analysis for free with a 14-day trial.

Start Your Free, 14-Day Trial

VERACODE

Veracode is intelligent software security. The Veracode Software Security Platform continuously finds flaws and vulnerabilities at every stage of the modern software development lifecycle. Prompted by powerful AI trained by trillions of lines of code, Veracode customers fix flaws faster with high accuracy. Trusted by security teams, developers, and business leaders from thousands of the world's leading organizations, Veracode is the pioneer, continuing to redefine what intelligent software security means.

Learn more at **www.veracode.com**,
on the **Veracode blog** and on **Twitter.**