CRASHTEST **SECURITY**

GUIDE

PREVENTING COMMON CYBERATTACKS

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

Table of Contents

Ebook about preventing the most common cyber threats

This **big fat evergrowing ebook** shows best practices and prevention techniques on how to keep vulnerabilities away and how to secure your web apps.

CHAPTER 01

Preventing SQL Injections

Table of Contents

01 | What is SQL Injection?

SQL (Structured Query Language) injection is a code injection technique that allows the hacker to send malicious SQL queries to a web application's backend database. The SQL injection vulnerability occurs when the web page asks for a user input but accepts a SQL statement that the database can execute. The adversary can access data that the application is not built to display using these malicious SQL statements, including server configuration, user lists, and sensitive company data.

02 | Types of SQL Injections

SQL injection is the most common type of web application attack and is divided into various categories based on the techniques used to extract data. Types of SQL injections include:

SQL (Structured Query Language) injection is a code injection technique that allows the hacker to send malicious SQL queries to a web application's backend database. The SQL injection vulnerability occurs when the web page asks for a user input but accepts a SQL statement that the database can execute. The adversary can access data that the application is not built to display using these malicious SQL statements, including server configuration, user lists, and sensitive company data.

In-Band SQL Injection Attacks

This type of SQL injection occurs when the hacker uses the same communication channel to send malicious SQL database queries and collect results. In-band SQL is considered the simplest and most prevalent form of SQL attack and falls into two main categories:

1. Error-based in-band SQL Injection

In this type of attack, the hacker obtains information about the configuration of the database from error messages generated by the server. In addition, attackers can enumerate the entire database engine using malicious input that exposes information about its version and structure.

2. Union-based SQL Injection

The attacker relies on the UNION operator to combine a genuine SQL query with the supplied malicious SELECT statements of the same structure to get a single HTTP response. The information to be accessed is part of this HTTP response. Therefore, the hacker can extend the results of an original query to extract further information about the database.

Out-of-Band SQL Injection

In this SQLi attack, the malicious user cannot gather information from the same channel used to launch the attack. Instead, they rely on available functions to exfiltrate data through outbound channels. Such functions include connection establishment functions and file operations. The success of this attack depends on whether certain database server features are enabled, such as its ability to initiate an outbound DNS/HTTP request for data transfer, making the attack less prevalent.

Inferential/Blind SQL Injection

In this SQL injection attack, the application does not transfer data from the database to the attacker. As a result, hackers have to learn about the server's structure by observing its behavior and response. These attacks are much slower since successful attacks require continuous observation of HTTP response patterns. Blind SQL injection attacks are typically categorized into:

1. Boolean/content-based attacks

The attacker crafts malicious SQL statements that ask the database TRUE or FALSE questions, then validating whether these queries modify the information within the server's HTTP response. The attacker then makes inferences about the server's structure and configuration by determining if each message generates a TRUE or FALSE statement.

2. Time-based SQL Injection attacks

The attacker crafts a query that forces the database server to wait for a set period before sending the response. This is followed by sending a malicious SQL payload and observing whether the server responds instantly or after a delay. Attackers use the response to infer whether a statement used is TRUE or FALSE, thus enumerating the database without relying on data in its response.

03 | What is the Severity Level of SQL Injection Attacks?

The impacts and severity of an SQL injection attack depend on the security countermeasures in place and the attacker's skill. The **Online Web Application Security Project (OWASP)** recognizes injection attacks as one of the most common and far-reaching attacks, consistently ranking them as one of the top 10 application security risks. The attack is mainly found in application frameworks with legacy functional interfaces and is a common vulnerability of ASP and PHP web applications.

By injecting malicious commands into the data plane input, attackers influence how the application executes SQL commands while obtaining access to information stored and processed by the application. As a result, such attacks have severe effects, including:

Identity spoofing

Attackers use SQL injections to collect credentials of an application's users in the database. They can then impersonate the database user and carry out privileges and tasks according to the victim.

Repudiation issues

SQL injection vulnerabilities allow attackers access to data stored in a database server. As a result, they can alter the data in the database resulting in repudiation errors such as **mismatched balance** and **null transactions**, affecting the application's data credibility.

Unauthorized data access

SQLi vulnerability lets adversaries select and output database table names and column names, thereby allowing them to access sensitive intellectual data, personal data, or the database structure, granting them complete control of the application.

Chained attacks

In applications where the OS relies on the database server, attackers can use SQLi attacks as the initial attack vector to orchestrate further attacks within the application.

04 | How to Identify SQL Injection Vulnerability with Crashtest Security?

The Crashtest Security Suite includes an automated scanning and penetration testing solution that helps reduce the risk of being hacked through SQLi vulnerabilities. The software establishes **systematic tests** across each phase of the SDLC to uncover and mitigate SQL injection attack surfaces. Crashtest Security is built to find injection vulnerabilities in data-driven applications to eliminate blind spots before attackers can leverage them to feed malicious SQL commands to the backend database.

Crashtest Security's vulnerability scanner relies on test cases developed using SQL payloads built to generate specific error messages. In addition, the vulnerability testing tool includes an extensive list of queries and variables that enable security and QA teams to identify vulnerabilities of many SQLi attack vectors.

The solution also blends Blind SQL injection with **machine learning techniques** and **application security mechanisms** (DAST) to emulate different attack patterns used in exploiting the vulnerability. This helps to uncover various approaches attackers can leverage to orchestrate a chain attack once they have identified the injection flaw.

The Crashtest Security Suite natively integrates with all modern web development stacks, enabling teams to instantly scan web applications and APIs for SQL injection vulnerabilities. The quick security assessment compares the website against **OWASP's top 10 benchmarks** to ensure the website is free of security gaps that are commonly used for SQL injection.

05 | SQL Injection Prevention Techniques

Though prevention techniques differ with the use case, here are some common ways to prevent SQL injection attacks:

+ Enforce server and client-side input validation
+ Utilize parameterized queries
+ Implement stored procedures
+ Enforce the law of least privilege
+ Enable Object Relational Mapping (ORM)
+ Escape user input
+ Use a Web Application Firewall

Enforce server and client-side input validation

Enforcing adequate measures to ensure users only submit the allowed input type is recommended. Whenever there is a user-controlled input interface, policies should define the accepted length, style, input format, and other characteristics used to identify valid input. This should also be supplemented with whitelists for the input type that passes validation checks and enforces maximum control on the strings that can be accepted through input fields.

To ensure strong input validation for whitelists, developers should also use regular expressions (Regex) for structured data. Using a pre-fixed value set, such as a drop-down list, also ensures that the data supplied at the input matches the defined characteristics exactly.

Finally, developers should treat all user-supplied input as malicious, ensuring that all data received from external entities is validated, keeping internal and unauthorized parties from executing malicious SQL statements.

Utilize parameterized queries

A parameterized query is a pre-built SQL statement that prompts the user to supply parameters (values/variables) before the server executes it. Prepared statements allow developers to define all acceptable SQL code, making it easier for the database server to distinguish between user input data and executable code. Since a parameterized query automatically quotes the parameter's value, the user-supplied input does not subvert the application's logic, thus preventing SQL injection attacks. If a hacker enters malicious SQL commands, the intent of a query cannot be changed, thereby protecting against the execution of unwanted statements.

Enforce the law of least privilege

Developers should avoid connecting the API to the database server using an account with administrative privileges unless it is absolutely necessary. Attackers could manipulate the database to access the root system, thus performing escalated attacks against the OS and backend server. Additionally, each application/entity should have its database access credentials with minimum access rights to data hosted within the tables. Using Role-Based Access Controls (RBAC), the teams should determine each user's access needs and create modes to ensure only the necessary permissions are granted.

Enable Object Relational Mapping (ORM)

ORM frameworks seamlessly translate the results of SQL queries into code, eliminating the need to use SQL statements in application code. ORM mappers also apply parameterized statements under the hood, which limits the type of user-supplied input to be executed. While it prevents dynamic SQL queries, the ORM model does not make the site entirely immune to attacks as web development frameworks allow fragmented SQL statements for complex operations.

Escape user input

As a last resort, developers should escape all user-supplied input strings and special characters before they are applied to a query. Using the appropriate escaping scheme for user input ensures that the server does not confuse the input with code, which prevents the execution of unwanted SQL commands.

Use a web application firewall

SQL injection attacks come in various patterns, with attack vectors not covered by prevention methods such as parameterization and user input validation. The Web Application Firewall (WAF) provides holistic protection against a wide variety of web security threats, including SQL injection. The WAF helps prevent attacks by identifying & mitigating adversaries and malicious code before they reach the vulnerable web server. With a proper WAF in place, developers, security, and QA teams can improve the application's defenses against SQLi attacks without making significant changes to the application.

06 | Best practices to prevent SQL injection attacks

Validating user input and avoiding dynamic queries are central to preventing SQL injection attacks in web frameworks. The following section discusses implementing SQLi prevention measures in popular development frameworks.

Django

Django includes ***querysets*** that automatically construct parameterized queries. These sets ensure that the query parameters are defined separately from the query code. Django allows developers to execute custom SQL and raw queries in complex cases. As a best practice, it is recommended to use Django's authentication library for the ***extra()*** and ***RawSQL()*** constructs when executing arbitrary queries. Escaping input to these queries also restricts SQL input that the user can supply. Django also includes an **Object Relational Model (ORM)** that maps database tables with the same fields and implements query parameterization to eliminate the SQLi attack surface.

Laravel

Laravel ships with the **Fluent Query Builder** and the **Eloquent ORM** that aid in defining prepared statements. PHP application developers can prevent all special characters from escaping through application forms by leveraging these integrations. When hackers pass malicious SQL queries through these forms, the SQL command is escaped, and the database saves the invalid query as text.

How can the Crashtest Security Suite help?

The **Crashtest Security Suite** enables a continuous scanning and testing process that helps teams uncover vulnerabilities within their APIs, Javascript, and web applications.

In addition, the vulnerability scanner offers actionable vulnerability reports with low false positives and negatives, thereby enabling security teams to have confidence in identifying and mediating security gaps. And to spare precious time looking for solutions on how to fix the vulnerability, we also feature remediation advice directly in the report and our wiki.

Crashtest Security's scanner fits seamlessly into different development environments, helping organizations detect and remediate SQL injection flaws throughout the CI/CD pipeline.

The tool is easy to set up and can start scanning applications in minutes, granting instant access to 15+ scanners, including a **dedicated SQLi vulnerability scanner**. Start your **free, 2-week trial** to explore how you can efficiently reduce SQLi attack vectors in your web application without adding effort overheads.

START 2-WEEK TRIAL FOR FREE

CHAPTER 02

Preventing XSS Attacks

Table of Contents

01 | What is XSS?

In a cross-site request attack, the hacker injects malicious code into the application server through a user input form. If the application does not validate user-supplied input, the malicious content is included in dynamic content forwarded to the browser for execution. While XSS vulnerabilities are present in most web programming frameworks, attacks mainly target **JavaScript** since it integrates with all modern browsers and is used by most dynamic websites.

XSS is a **client-side attack** since the malicious input is delivered to and executed by the user's browser. By compromising the user's interaction with the webserver, an attacker can perform a wide range of vicious tasks, including:

- **Stealing of cookies, session tokens, and other session information**
- **Installing malicious software on the user's machine**
- **Redirecting the victim to the attacker's malicious pages**
- **Theft of personal data/user identity theft**

These vulnerabilities typically occur on application elements that accept unsanitized user input, included in the web server's dynamic response.

02 | Types of XSS?

Attackers can leverage the XSS application vulnerability to compromise an application using various tools and techniques. The most common forms of XSS attacks are:

Stored/Persistent XSS

In this type of attack, the hacker injects the malicious script into the application so that it is stored within the webserver. The Stored XSS vulnerability is more common and damaging since the script is only injected once and is included in all consequent HTTP responses. This vulnerability has a broader reach since it affects every user who logs in to the application. Attackers prefer to orchestrate this form of attack since they are self-contained and do not require the crafting of an external means to furnish client-side scripts.

Reflected XSS

In a Reflected XSS attack, the webserver immediately returns the user-supplied script within its HTTP response. This attack is not persistent because the webserver does not permanently store the vulnerable code contained within the input. The server reflects the external scripts to the client after processing the user's request. Reflected XSS, also known as **Type 2 XSS attacks**, are more common in forums where the hacker tricks an unsuspecting user into sending malicious code to the webserver using phishing attacks.

Dom-based XSS

In a DOM-based XSS attack, the malicious payload is executed when the attackers modify a website's Document Object Model (DOM) in the user's browser. While the HTTP response remains unaltered, the malicious modifications within the victim's browser make the client-side code run unexpected scripts. The attacker inputs vicious code within a user input form that supports dynamic code execution. Some most popular sources for DOM-based vulnerabilities are the **document's URL, location**, and **referrer**.

03 | What is the severity level of an attack?

While it has been exploited and researched since the 21st century, the **Cross-site Scripting** vulnerability is still a choice attack vector due to its simplicity and effectiveness. The **Cross-site Scripting vulnerability** was positioned at **number#7** in the 2017 OWASP top 10 while it was consolidated under the **Injection vulnerability** and was ranked **number#3 in 2021**.

A hacker can deface a web application, redirect users to malicious websites, or steal their session cookie with a successful attack. Additionally, by obtaining user credentials of users with elevated privileges, hackers can achieve complete control of the web application.

04 | How to identify XSS vulnerabilities with Crashtest Security?

Crashtest's Security's XSS vulnerability scanner offers a quick and reliable approach to testing for existing XSS security gaps within websites. The scanner combines **DAST** and **SAST** techniques to help developers and security professionals examine each data entry and exit point. Crashtest Security assumes entry points as those interfaces that accept user-controlled data. In contrast, exit points are components of the application where this data might be included in the server's response.

The scanner automatically submits arbitrary values at entry points, analyzing whether the data has been altered at the exit point. Once links between entry and exit points are mapped, the platform tests these links for XSS vulnerabilities using predefined security rules.

Crashtest Security Suite automates XSS testing while swiftly integrating results into your tech stack. The scanner outputs comprehensive CSV, XML, and PDF reports that can be shared across teams for enterprise-wide awareness about the state and impact of XSS vulnerabilities.

05 | XSS prevention techniques

Some ways to prevent XSS attacks in modern web applications include:

Perform proper input validation

Developers should ensure that user-controlled input interfaces don't accept code that may result in the execution of malicious scripts in user browsers. Each input form should include an allowlist that restricts user input to admissible values. In areas where custom user content is required, the developers should avoid HTML inputs. The developers should follow these and other additional measures to ensure the web server does not accept data from untrusted sources.

Avoid untrusted URLs

Attackers can leverage unsafe URLs that execute JavaScript code to send malicious payloads within the protocol JavaScript: in DOM URL locations. Web development teams should ensure their URL parsers only accept trusted transfer schemes like HTTPS.

Implement a content security policy

The Content Security Policy (CSP) is a client-side prevention technique that helps detect and prevent injection attacks like XSS. Using a CSP, developers define the domains that the browser can be viewed as trusted sources by when executing the server's response. This enables the browser to ignore scripts in source files originating from domains not listed in the CSP, thereby mitigating an XSS vulnerability.

Enforce proper output encoding

Output encoding is a common practice of converting untrusted dynamic user input into secure formats where the browser displays them as data without executing them as code. Each programming language has an entity encoding scheme that should be implemented as part of secure coding best practices by web developers to ensure proper escaping.

06 | Adopting best practices for XSS prevention

Some framework-specific XSS prevention best practices include:

PHP

In a PHP application, HTML special characters function [***htmlspecialchars()***] helps convert dynamic special characters into HTML entities. This function is recommended to be used for escaping special characters to ensure they are not executed as coded input by the browser.

PHP also includes multiple libraries that can be used to prevent XSS attacks. These include **HTML purifiers, PHP Anti-XSS**, and **htmLawed**.

JavaScript

There are multiple ways to prevent XSS vulnerabilities in Javascript event handler attributes and blocks. Developers must configure the application to accept inline scripts stored in quoted data values. With the above configuration, any scripts outside the quoted context are treated as unsafe inputs that should not be executed in-browser.

The syntax for safe Javascript would be similar to:

<script>alerts('...ENCODED UNTRUSTED DATA…')</script>

The data can also be part of a quoted expression, as shown:

<script>x='...ENCODED TRUSTED DATA…'</script>

Javascript can also be encoded inside a quoted event handler, as shown:

***<div onmouseover="x='...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...'" ***

CHAPTER 03

Privilege Escalation

Table of Contents

01 | What is Privilege Escalation?

In a privilege escalation attack, hackers leverage vulnerabilities in access controls and resource restrictions to override the permissions and limitations of a target user account for gaining elevated access. These attacks are usually aimed at obtaining administrative privileges and utilizing them to manipulate the system's security settings by further extending the scope of the initial attack. Although most attacks rely on compromised unprivileged user accounts, attackers can also escalate privileges by exploiting other configurations and system bugs.

Some common threat vectors for privilege escalation attacks include:

Known Misconfigurations

In the absence of robust security practices during the fundamental phases of the application design, inherent misconfigurations are one of the most common exploits for malicious activities. Most privilege escalation attacks target misconfigurations in user accounts with poor or default security settings. Other misconfigurations that can be used for privilege escalation include:

- Exposing storage buckets to public networks with no authentication
- Using default credentials for root and admin accounts
- Undocumented administrative backdoors that attackers can discover
- Poorly documented configuration changes
- Improper configuration of the latest security patch in an upgraded component
- Error messages revealing stack traces or other sensitive information
- Installation and activation of unnecessary features such as ports, accounts, and services
- Missing validation for user input
- Insecure or missing security directives and headers in server responses

Privilege Vulnerabilities

Privilege vulnerabilities refer to those system flaws that allow users to adjust their permissions once they have been authenticated to the system. Examples of privilege escalation vulnerabilities in Windows and Linux systems include:

- Attackers utilize **access token manipulation** to trick the server into believing that a malicious process belongs to a legitimate user and gaining explicit permission for the malicious actor as that of the user. To get the access token, attackers orchestrate various techniques such as access token duplication, creating a login session using the LogonUser function, or creating a new process using a duplicated token.

- **Bypassing the user account control (UAC) mechanism** helps obtain elevated privileges by abusing the distinction between administrative and standard users.

- Attackers commonly **hack the DLL search order** by planting an adversarial DLL with the same name as the legitimate DLL, but in a location that the system will search before the legitimate DLL. This location is either at an upper-level folder within the working directory or a remote directory pointing to an external file volume. This results in the application finding the malicious DLL and executing it, believing it to be a legitimate DLL.

- An attacker connects to the host and employs the mechanism of **enumeration** to gather information about usernames, machine names, services, and networks. This allows them to discover potential attack vectors to access additional resources deep into the system.

- Attackers exploit known **Linux Kernel vulnerabilities** to gain root-level access to the system for executing attacks at the root privilege level, making it impractical for the system to defend.

- Attackers typically target users with SUDO system access to **exploit** their **SUDO rights** to gain root access for executing commands.

Credential Exploitation

Flaws in the authentication layer often equip malicious users with knowledge of the account names, user IDs, and passwords needed for login attempts. Over the years, attackers have developed several sophisticated ways of acquiring user credentials, such as:

- **Password exposure** through hard-coded passwords or insecure credential databases
- Utilizing **brute force attacks** to guess passwords using known and common combinations
- Using devices such as keyloggers and cameras to **shoulder surf** actions of authorized and privileged users
- Utilize **dictionary attacks** and **automated routines** to combine lists of common words that can be used as access credentials
- Leverage algorithms to encrypt and hash passwords while utilizing **Rainbow table attacks** to reconstruct hashes into original passwords
- Using the **Credential stuffing** technique to gain the lists of exposed usernames and passwords from previous exploits. This works well as users often reuse passwords across services
- When a user's device gets compromised, attackers commonly exploit the weaknesses of the process in generating, transmitting, and storing a new password when requested for an account password reset.

Malware

Malware is one of the most infamous malicious software that runs as an operational process with permissions of the compromised user account who executed it. Attackers can deploy malware at the standard user level and later escalate the privileges to the administrative level, extending the radius of their attack to an entire ecosystem. Malware that can be used for privilege escalation includes:

- **Rootkits** - These are malicious processes running with root privileges, giving the threat actor complete control over the Operating Systems.
- **Bots** - They are automated programs used to spread arbitrary code and malware and perform other malicious actions against the compromised assets.
- **Spyware** - These refer to the malicious software that performs reconnaissance and surveillance on target devices/users. Some examples of Spyware are keyboard loggers, webcam hacking software, microphone bugs, etc.

Social Engineering

These techniques manipulate users into violating security controls and exposing sensitive information. Since it preys on human weaknesses, social engineering is one of the most effective techniques for privilege escalation. Some common social engineering attacks used for privilege escalation include:

- **Phishing** - Phishing tricks the user into clicking a malicious link or attachment in a legitimate-looking message. Clicking the link or attachment typically initiates the deployment of malware or other compromises used to obtain illegal access.
- **Whaling** - Whaling is a form of phishing attack aimed at high-ranking personnel of an organization. This typically involves using fake time-sensitive opportunities and emergencies to pressurize executives into clicking malicious links to expose credentials for high-level network access.
- **Baiting** - The baiting technique lures the victim user into exposing sensitive data by being promised a gift.
- **Scareware** - The attacker uses alarming audio and flashy graphics through pop-up windows in a scareware attack. The intention is to falsely alarm the user of an impending upgrade/virus that needs to be purged. Clicking on the link sends the user to an antivirus purchase website where the scammer may attempt to steal credit card or other account information.

Impact of Privilege Escalation Attacks on an Organization

Privilege escalation remains a critical concern for an organization's web application security. The attacks are no longer restricted to the network periphery but intrude inside the organization's database to gain access to sensitive customer/organization internal data. Privilege escalation is often the first stage of a crucial complicated attack. The ultimate goal of the attack is always to gain elevated and unauthorized access to an organization's critical resources and compromise security. Impacts might range from accessing confidential data, installing malware or malicious code, and/or completely hijacking the organization's systems.

02 | Types of Privilege Escalation

Privilege escalation attacks are broadly categorized into two categories based on how the attacker extends the reach of its attack. These include:

Horizontal Privilege Escalation

In a horizontal privilege escalation attack, the hacker gains the access rights of other entities with similar permissions. To do so, attackers typically target unprotected, lower-level user accounts with similar privileges but have access to different security contexts. By orchestrating such attacks, attackers can gain access to resources and features that are available only to the victim user.

Some vulnerabilities that facilitate horizontal privilege escalation include:

- User IDs controlled by HTTP request parameters in the URL
- Exposed globally unique identifiers (GUIDs)
- Data leakage in redirects that contain user IDs

Vertical Privilege Escalation

In a vertical privilege escalation attack, attackers leverage flaws in the system to upgrade the access rights of a compromised account that allows them to view and control resources they are currently not permitted to do. Vertical escalation, also known as **privilege elevation**, typically entails moving from low-level users to a higher form of privileged access, often administrative. This typically requires going through various intermediate steps to obtain privileged credentials. Some commonly used steps include: **obtaining credentials for privileged accounts, editing high-level scripts and executables, editing application source code,** and **abusing misconfigurations to gain access to these accounts**.

Vulnerabilities that are most susceptible to vertical privilege escalations include:

- Unprotected functionality
- Parameter-based access controls
- Platform configuration flaws leading to broken access control

03 | What is the severity level of a Privilege Escalation

Privilege escalation is a broken access control flaw, which is ranked number one in the OWASP top ten list of vulnerabilities. By allowing a compromised user to act beyond their intended permissions, such vulnerabilities often lead to unauthorized access, affecting the integrity and availability of the application. With a privilege escalation attack, hackers can leak personal information belonging to the application's users, manipulate the data processed by the application or compromise the execution of processes by the webserver if the application processes financial or commercial data, horizontal privilege escalation attacks are mainly carried out for theft and fraud through account hijacking and data manipulation.

Privilege escalation vulnerability is considered highly exploitable (exploitability: two) since most threat actors are skilled at detecting and leveraging access control failures. At the same time, it is essential to note that both manual and automated techniques are available to detect such flaws within a system. When developing application code, lack of functional testing is considered one of the most common aberrations that make this vulnerability relatively common (prevalence: two).

Since a privilege escalation attack allows hackers to initiate administrative functions, the technical and business impacts vary from low to critical depending on the data and processes run by the application. Typical consequences include exposure of sensitive information, data alteration/destruction, malicious code execution, and service denial.

04 | How to identify Privilege Escalation Vulnerabilities with Crashtest Security

Crashtest Security offers an automated vulnerability testing tool to help prevent privilege escalation by identifying and remediating access control flaws. The security suite includes a custom privilege escalation scanner that helps prevent attackers from gaining administrative rights to web applications. Crashtest Security's DAST tool also helps embed security in the initial phases of development to eliminate flaws in the application's peripheral such as open ports and APIs, that may lead to account takeovers.

The scanner outputs a privilege escalation report that lists attack vectors based on categories, severity, and proposed mitigation solutions. Crashtest Security also includes a port, OWASP, and HTTP header scanner to help track down specific flaws and misconfigurations commonly exploited to orchestrate privilege escalation attacks.

05 | Privilege Escalation Prevention Techniques

Given a large number of exploitation techniques, preventing privilege escalation attacks requires multiple defense strategies like privileged access management and identity-centric patterns. Some robust approaches to prevent privilege escalation attacks include:

Strong Password Policies

Developers and security professionals should enforce password policies that promote the creation of strong, secure passwords which cannot be compromised using social engineering techniques. Users should also be discouraged from reusing passwords across services that may get compromised in other data breaches. Additionally, more robust authentication controls such as Multi-Factor Authentication (MFA) and device-based sign-in restrict malicious players from accessing administrative accounts using stolen credentials.

Enforce the Principle of the least Privilege

Every entity that interacts with the application should only be granted the privileges required to complete its task. The assignment of access rights should be controlled by the role and function of the entity rather than the identity. These principles confine processes to a small domain, reducing the blast radius of a successful attack. A policy must be enforced for developers to mandatorily declare and document the permissions required to access each resource while administrators should perform appropriate security checks before granting access. Assigning only the needed permissions makes it difficult for hackers to perform activities outside their intended scope.

Secure Management of Privileged Accounts

It is crucial to secure privilege accounts against exposure and unintended access. The security teams should catalog all privileged accounts along with the processes they execute. An application should have minimal privilege accounts with defined scopes. Additionally, security professionals should set up continuous logging and monitoring for these accounts to avoid any exploit. These accounts should also be constantly tested and hardened against potential threats and vulnerabilities that may lead to abuse of the privilege.

06 | Best Practices in Preventing Privilege Escalation

Some of the recommended practices to prevent privilege escalation include:

Keep Components Updated

Most privilege escalation attacks target known vulnerabilities that are readily published in the Common Vulnerability Enumeration database. Security teams should ensure the systems and applications that interact with the webserver are regularly patched with the latest updates to minimize the risk of hacking attacks. A trusted patch management system should also be integrated with appropriate software implementations to help applications not miss any critical security updates.

Eliminate File Transfer Functionality

Attackers mostly use file transfer functions such as FTP, get, and curl to download and execute malware. These tools and functions should be used when necessary with utmost precaution and should only be enabled for specific users, directories and applications.

Invest in Security Training and Awareness

As most attacks begin with social engineering techniques, the human factor is considered the weakest link in a cyberattack chain. All employees and third-party vendors should be educated on common cyber security risks and their roles in avoiding them.

Vulnerability Assessment and Mitigation

Automated scanners should be used to constantly scan, detect and identify flaws that could lead to privilege escalation. Identified vulnerabilities should be urgently addressed with patches, updates, and fixing misconfigurations to strengthen the web application's security posture.

How can the Crashtest Security Suite help?

Crashtest Security Suite helps reduce the attack surface by enforcing **continuous penetration testing** and **automated scanning** for broken access controls and security misconfigurations. The suite comes bundled with multiple automated scanners with simple integration for end-to-end cyber security hardening. Start your **free, fourteen-day trial** to see how Crashtest Security can help eliminate privilege vulnerabilities.

START 2-WEEK TRIAL FOR FREE

CHAPTER 04

Prevent SSL/TLS Vulnerabilities

Table of Contents

01 | What are SSL/TLS Vulnerabilities?

Given the heavy reliance on TLS/SSL protocols in modern web and network traffic, attacks targetting vulnerabilities within such protocols are prevalent. Although such protocol suites are constantly evolving to address advanced threats, configurations of a deprecated version or those with an original protocol continue to be prime targets for hackers.

Significant vulnerabilities in TLS/SSL include:

Attacker encrypted communication

If the attacker can get a hold of data in transit, they may encrypt, misuse and lock sensitive user and system data. This also potentially builds the foundation of a ransomware attack that leads to attackers demanding ransom from the affected user before restoring their access to the data.

Wildcard certificates

Some enterprises use a single, self-signed certificate to secure all their subdomains of a single domain name. While this allows for the quick deployment of secure transport protocols across all subdomains, it also introduces many security risks. If a public-facing web server using a wildcard is compromised, attackers can impersonate any subdomain protected by the wildcard key certificate. Attackers can also use a webserver's identity to host malicious content for phishing attacks and other social engineering campaigns.

Untrusted certificate authorities

Certificate Authorities (CAs) are responsible for issuing SSL/TLS certificates. Attackers exploit inherent security misconfigurations to act as a CA and issue forged self-signed certificates. Cryptographic configurations may fail to detect malicious activity in such instances where the CA is compromised or untrusted. Since the browser trusts the CA, users continue to transact sensitive data being under the impression that they are connected to an authentic, secure webserver.

TLS/SSL stripping vulnerabilities

TLS/SSL stripping is a popular vector for modern man-in-the-middle attacks built on the SSL strip vulnerability. The vulnerability allows hackers to downgrade the security potency of web systems by depriving them of the encryption layer that offers HTTPS security. After a successful attack, attackers intercept users' HTTP requests and proxy them to a malicious server during HTTPS redirection. Since users miss noticing the redirect, they continue interacting with the attacker's server over HTTP.

RSA key transport weaknesses

Devices that use RSA encryption keys and certificates are susceptible to the Trusted Platform Module firmware vulnerability that mishandles key generation. This allows hackers to access private keys and generate their certificates. The keys generated by this encryption standard are not genuinely random and allow for a practical factorization or birthday attack.

CBC-mode cipher vulnerabilities

Ciphers that use the CBC mode of operation use padding to ensure a uniform block size. CBC-cipher vulnerabilities facilitate Padding Oracle attacks - exploits in which the attacker tampers with the ciphertext to check whether it causes an error in padding format. If the server returns an implementation error while decrypting invalid security content, the hacker can deceptively create malicious ciphertext and extract enough information to reconstruct the plaintext data.

02 | Types of SSL/TLS Attacks

With the changing threat landscape, hackers have developed different attack patterns to exploit vulnerabilities in TLS/SSL. Some most advanced forms of TLS/SSL attack include:

POODLE Attack

In October 2014, the **Padding Oracle On Downgraded Legacy Encryption (POODLE)** attack exploited an SSL 3.0 protocol vulnerability. In this type of attack, hackers leverage protocol negotiation capabilities to enforce the use of SSL 3.0. This starts with the attacker eavesdropping on all communication between the server and the client to manipulate the network traffic to impersonate entities. Following this, the attacker tricks the server into thinking that the client can not communicate with newer protocols by dropping connections. This convinces the server to use the SSL 3.0 protocol while the attacker tricks the client browser into running JavaScript that helps decrypt sections of encrypted data.

POODLE attacks are carried out on servers that use 128 and 64-bit block ciphers and a symmetric encryption algorithm. The cipher suite selection used on these systems relies on the CBC mode of operation and uses the Message Authentication Code (MAC) to encrypt input plain text and padding values. Hackers can reconstruct several bytes of the original plain text by crafting similar malicious requests.

BEAST Attack

The **Browser Exploit Against TLS/SSL (BEAST)** attack allows hackers to capture encrypted sessions and obtain plaintext data. The BEAST vulnerability can be found on any webserver/website that supports application protocol SSL, TLS 1.0, or older cryptographic protocols with weak ciphers.

Given the number of input-key combinations, breaking cryptographic algorithms typically requires considerable time and effort. However, the BEAST vulnerability simplifies the deciphering of cryptographic algorithms by allowing the attacker to guess a single byte at a time.

CRIME Attack

Compression Ratio Info-leak Made Easy (CRIME) attack targets a client-side implementation error on connections that use a stream cipher, HTTPS/SPDY, TLS data compression, or web cookies. A man-in-the-middle attack can hijack session cookies for web servers with this implementation error while the user is authenticated to the web application. The primary objective of a CRIME attack is to infer the value of the session cookie by observing the change in compression method or ciphertext length.

When orchestrating a CRIME attack, a hacker typically abuses vulnerabilities in the compression mechanism, allowing them to control the path taken by new requests and inject malicious data into the stream. A successful attack allows the hacker to obtain data on the size of security content sent by the browser, the compression method, and the payload compression ratios.

BREACH Attack

The Browser Reconnaissance and Exfiltration via Adaptive Compression (BREACH) attack targets HTTPS connections using HTTP-level compression. BREACH attacks are typically targeted at applications that include at least one of the following flaws:

- Reflect user input in the body of HTTP responses
- Reflect security content within the response bodies
- Connect to a server that uses compression at the HTTP level

Attackers can exploit these vulnerabilities to extract cookies, client certificates, authentication tokens, and other sensitive security content. A BREACH attack involves sending several requests to the server while observing data returned in the server responses.

FREAK Attack

A FREAK attack is a commonly found SSL/TLS vulnerability that forces target clients and servers to use export-grade cryptography by intercepting HTTPS connections. In a FREAK attack scenario, threat actors trick cipher suite selection mechanisms to establish a connection and downgrade the application protocol to a vulnerable version.

A FREAK attack is commonly orchestrated on servers that meet at least one of the following conditions:

- The server supports RSA export cipher suites
- Clients offer the RSA cipher suite/use vulnerable OpenSSL version
- Clients use Secure Channel or Apple Secure Transport for enterprise transport security

DROWN Cross-Protocol Attacks

Decrypting RSA with Obsolete and Weakened eNcryption (DROWN) is a collection of downgrade attacks aimed at servers that support SSL v2 connections. To do so, attackers leverage cross-protocol vulnerabilities that allow modern, TLS-enabled websites to use the insecure SSL 2.0 protocol. SSLv2 being vulnerable to a **Bleichenbacher RSA padding oracle** attack, attackers exploit its vulnerability to decrypt RSA security content without possessing the private key. Successful DROWN attacks result in the exposure of sensitive, encrypted data, such as initialization vectors, passwords, trade secrets, financial data, and credit card information.

Renegotiation Attacks

A renegotiation procedure is essential for TLS/SSL communications since it allows for creating a new transport layer security handshake inside an existing successful connection. This helps re-establish secure connection after a terminated negotiation phase while enabling entities to change cipher suites and renew client certificates if they need to. In instances where a client is allowed to initiate renegotiation handshakes, attackers can inject malicious content into the protocol stream and perform denial-of-service attacks to lead the webserver into a downtime.

03 | SSL/TLS Vulnerability Severity Level

Transport Layer Security plays a crucial role in modern web communications as it encompasses various encryption protocols and techniques designed to provide integrity and confidentiality. Vulnerabilities in the middlebox security protocol are severe since they allow attackers to access hidden information. Unsurprisingly, two of OWASP's top 10 vulnerabilities of 2021 -- **Broken Access Control (A01:2021)** and **Cryptographic failures (A02:2021)** -- are directly associated with misconfigurations in transport layer security.

These vulnerabilities allow hackers to intercept encrypted communications and snoop on traffic between web servers and clients. Successful TLS/SSL attacks lead to the disclosure of sensitive information, thereby allowing attackers to unleash various attacks based on enterprise transport security flaws, such as:

- Account takeover
- Data breaches
- Denial-of-service
- Exposure of encryption algorithms in proprietary networks with weak ciphers
- Ransomware attacks
- Theft of client certificates using FREAK attack
- Phishing attacks
- Deployment of advanced persistent malware
- Privilege escalation attacks
- Loss of business reputation and customer trust

04 | How to identify SSL/TLS Vulnerabilities with Crashtest Security

The Crashtest Security Suite includes a new generation TLS/SSL scanner to identify and report application vulnerabilities in a single click. The scanner automatically connects to target ports and cross-validates the cipher suite list and protocol versions to discover common vulnerabilities that lead to transport layer security attacks. The scanner also checks chain certificates to ensure they are from a valid CA and point to a legitimate server. Crashtest Security provides actionable vulnerability reports with comprehensive details such as supported TLS/SSL versions, the server's handshake preference, certificate details, and encryption cipher details to ensure complete hardening of the transport layer.

Crashtest Security also offers a heartbleed tester out of the box that checks the configuration of every webserver to prevent heartbleed vulnerabilities in SSL client certificates. As heartbleed attacks are aimed at OpenSSL, the scanner helps avoid vectors that enable hackers to obtain a private key and impersonate the webserver.

The security suite additionally comes with an extensive list of scanners to detect, identify and remediate various attack vectors associated with transport layer security. These include **CSRF scanner, port scanner, FREAK attack vector detection, CRIME vulnerability scanner, BEAST vulnerability scanner, HTTP Header scanner,** and **API vulnerability scanner**.

05 | SSL/TLS Vulnerabilities Prevention Techniques

Some ways to prevent transport layer security vulnerabilities include:

Using the latest TLS version (TLS 1.3)

Use the most stable, up-to-date version of the TLS/SSL cryptographic protocol as it includes the most recent enhancements to avoid known vulnerabilities of previous versions. The current TLS version has dropped support for legacy insecure protocol and encryption algorithms, such as export cipher suites, making it less susceptible to obsolete cryptographic algorithm attacks. TLS 1.3 features fewer and faster handshakes, with TLS handshakes requiring one round trip instead of two in the negotiation phase. TLS 1.3 also features a more extensive cipher suite list that replaces weak ciphers with GCM ciphers for a successful connection.

Choosing an appropriate Certificate Authority

Certification Authority Authorization (CAA) records specify which CAs can issue digital certificates for a particular domain, eliminating the need for self-signed certificates. Any CA who is missing from the CAA records should be avoided for issuing a valid certificate. The CA should be well known and trusted automatically by browsers and operating systems for public-facing applications. Organizations can also use internal CAs for proprietary networks, although these can only generate key certificates for internal users and servers within the organization‘s network.

Using HTTP Strict Transport Security (HSTS)

Using a special response header, modern web servers specify HSTS configuration as an extended enterprise transport security protection layer. Although most browsers support HSTS, it is recommended to explicitly include it in the server configuration to ensure it is active for all internet-facing web applications. Once this header reaches the client, the browser ensures all communications sent to the specified domain are in HTTPS and not HTTP. HSTS also prevents browser prompts for HTTPS click-throughs. By automatically redirecting HTTP requests to HTTPS, HSTS forms the first line of defense against man-in-the-middle attacks.

Enforcing CSRF Protection

For applications susceptible to cross-site request forgery, attackers can force authenticated users to inject malicious payloads into the data stream by submitting requests to malicious websites through phishing attacks. These malicious payloads can be used to modify ciphertext as in POODLE attacks or abuse the application's compression mechanism. CSRF protection techniques keep attackers from successfully instigating man-in-the-middle attacks, hence preemptively avoiding enterprise transport security threats.

06 | Best Practices in Preventing SSL/TLS Attacks

Best practices to avoid compromising TLS/SSL security include:

Use cryptographically strong ciphers

TLS supports an extensive cipher suite list that includes ciphers with varying encryption strength and security levels. As a recommended practice, an organization's cipher suite selection should involve a cipher suite list that covers the most appropriate encryption algorithms. Whenever possible, it is also recommended to enforce the use of Galois/Counter Mode ciphers that offer enhanced security by leveraging universal hashes over binary Galois fields for authenticated encryption. Weak ciphers such as anonymous, null, and EXPORT cipher suites should also be avoided as these are considered vulnerable versions that expand the attack surface.

Avoid TLS Compression

TLS compression is known for its vulnerability to leaking plaintext information, such as session cookies and other security-related HTTP headers, subsequently allowing for CRIME attacks. Hackers can steal cookie data and hijack user sessions using a CRIME attack. Although modern upgraded browsers do not support TLS compression, it is recommended that browser versions are upgraded regularly while TLS compression is explicitly disabled in browsers.

Perform regular server configuration tests

The OWASP security bulletin offers guidance on SSL/TLS testing to help identify weaknesses in transport layer security. This allows teams to establish whether they have implemented adequate security controls to protect the TLS/SSL layer. The tests include **server cipher suite selection validation, configuration validation, certificate strength, validity scans**, and **appropriate implementation of TLS**.

Crashtest Security Suite's TLS/SSL vulnerability scanner allows you to scan your web application directly through an efficient webhook integration to help test multiple attack vectors, including **portscan, common misconfiguration**, and **breach attacks**.

Use wildcard certificate when absoultely necessary

Since wildcards make a single certificate valid for all associated subdomains, they violate the principles of **same-origin** and **least privilege**. As multiple server systems commonly share a wildcard certificate, the certificate's private key must exist on numerous systems, increasing the likelihood of compromise. Since attackers value the single private key highly, it becomes a prime target for breaches. It is recommended that instead of using wildcard certificates on production systems, developers should leverage **short-lived, sub-domain-specific certificates** that are regularly rotated to prevent malicious attacks.

Avoid mixing TLS and Non-TLS Content

A page that uses TLS/SSL protocols should not include resources loaded over unsecured HTTP connections. Attackers can use these resources (CSS and Javascript) to inject malicious code or sniff session cookies from a target webpage. It is also recommended to use updated versions of modern browsers that are configured to deny a successful connection when dynamic loading content served over HTTP to HTTP-secured pages.

CHAPTER 05

Prevent Broken Access Control

Table of Contents

01 | What is Broken Access Control?

Access control issues enable unauthorized users to access, modify and delete resources or perform actions beyond the permissions. Broken access control encompasses various security vulnerabilities typically exploited to elevate privilege levels. Developing secure and effective access control schemes is often a complex undertaking that spans multiple application functions that were not designed deliberately but have evolved with the application. Software developers often overlook how entities access resources when implementing these schemes, resulting in hidden authorization flaws. Such control flaws are typically easy to discover and exploit, making them a popular target for common attacks.

02 | Types of Broken Access Control

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 obtain access to the accounts of other regular users with the same level of permissions. An attacker can leverage these vulnerabilities to get the legitimate user's data and use it for a wide range of malicious acts such as ransomware attacks, financial fraud/unauthorized money transfer, exposure of sensitive files, and data deletion. A horizontal privilege escalation attack usually does not require sophisticated attack tooling and can be orchestrated with a few simple steps, such as:

- By modifying the URL's request ID parameter with legitimate user details obtained through some form of social engineering
- By 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

Vertical Privilege Escalation

Vertical privilege escalation, also known as privilege elevation, allows an unauthorized user to gain higher privilege levels, typically admin privileges. Privilege elevation usually follows an initial attack, as the malicious user intends to obtain permissions beyond what the compromised subject already has. When compared to horizontal escalation, vertical privilege escalation attacks are more sophisticated since the hacker is required to perform root or kernel-level modifications to obtain administrative access.

Once the attackers gain access rights of admin users, they can inject malicious payloads at the code level, disrupt a sensitive business function or impact the availability of the application's critical resources. Some common techniques hackers use to abuse vertical access controls include:

- 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

Context-Based Privilege Escalation

A hybrid attack in which the malicious user first obtains access to regular user accounts and then uses broken vertical access controls to gain administrative rights. Context-based privilege escalation attacks also involve business logic exploitation that allows users to perform usually impossible actions 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 for access to functionality and sensitive files beyond their permitted context
- Location-based attacks

03 | Broken Access Control Severity Level

Broken access control is ranked number one on the 2021 OWASP Top 10 security vulnerabilities for web application environments. These vulnerabilities enable attackers to masquerade as different types of users and take control of legitimate user accounts. Depending on the actual vulnerability exploited, the consequences of a privilege escalation attack can be severe.

Specific 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 - The file permission 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 efficiency.

Prevalence of the CSRF vulnerability is considerably common in modern web applications. Attackers also often leverage the fact that most web applications use predictable parameters for actions. However, using appropriate code analysis and penetration testing techniques, the detectability of a CSRF vulnerability is relatively uncomplicated. Since such an identification technique requires **multi-stage delivery of payloads** and is well documented, a CSRF vulnerability's overall severity is typically regarded as of ***average 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

Access control vulnerabilities are commonly exploited, with a maximum incidence rate of 55.97% and an average incidence rate of 6.82%. Broken access control vulnerabilities have an average weighted impact of 5.93 and an average coverage rate of 47.72%.

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 Control Sphere
- CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory

04 | Identifying Broken Access Control With Crashtest Security

Through AI-driven testing and comprehensive vulnerability scanning, Crashtest Security Suite helps generate an in-depth analysis of a tech stack's security and access control. Crashtest Security Suite includes a list of scanners that collectively help analyze broken access control. The list of scanners includes:

- 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 - A security scanner used to detect attack surfaces that expose application server implementations, privacy laws, and the web application's access control policy to external domains.

Crashtest Security's automated scanning reduces manual efforts, and lets developers focus quickly on implementing secure design and threat mitigation policies. The platform also offers actionable security reports that can be shared across cross-functional teams, clients and executives, subsequently helping to administer security as a shared responsibility across all verticals of an organization.

05 | Broken Access Control Prevention Techniques

Some techniques to prevent access control issues include:

Multi-Factor Authentication

Multi-Factor authentication (MFA) is a zero-trust approach to administering security that deploys a series of access control checks that make it difficult for a hacker to perform malicious activities even after acquiring legitimate user credentials. This multi-layered defense strategy combines different authentication mechanisms to validate a user's identity. Mandatory requirement of two or more proofs of identification (such as authentication tokens or biometric IDs) for granting access essentially blocks unauthenticated users from exploiting a normal user account, thereby preventing access control attack attempts.

Proper Authorization Schemes

Using CSRF tokens within custom request headers offers a more robust defense mechanism against CSRF attacks as it enforces the same-origin policy restriction. Modern browsers do not support custom headers to be transported with Cross-Domain requests by default. Custom headers can only be added in JavaScript or within the script's origin. This CSRF mitigation technique is stateless and requires no changes to the user experience, making it particularly useful for CSRF mitigation on a REST service.

Apart from considering scalability and agility as the critical features of an application, software developers should build the software with robust access controls and security in mind. Authorization models that enable the creation of effective control units include:

- Discretionary access control - Limiting access based on the subject's identity and/or the groups they belong to. A subject with direct access permission can indirectly assign that permission to other subjects of their choosing.
- Mandatory access control - This mechanism involves securing access to resources based on the sensitivity of the information held by the resource. Administrators label the data sensitivity consisting of a security level and one or more security categories. This allows subjects to only access information held by a resource whose security label applies to them.
- Role-based access control - This access control scheme divides network subjects' access levels into roles. The user is attached to a role, which allows access to the information and functionality needed to perform their duties effectively. Roles are typically built upon job competency, authority, and responsibility.
- Attribute-based access control - Permits or denies information exchange based on properties of the requesting entity, requested action, the context of information exchange, or the resource requested. Some properties used in attribute-based authorization include:
    - Location
    - Threat level evaluation
    - Time of day
    - Security measures implemented on the requested resource

Unit and Integration Tests

Development teams should implement unit tests at each stage of the software lifecycle to prevent access control flaws at the code level. Unit testing helps evaluate individual modules to ensure appropriate implementation of access control on application code and uncover class-level weaknesses in privilege management. On the other hand, integration tests cover a more extensive scope that includes third-party and open source components used in building the application, thereby helping to evaluate the overall security posture.

Session Management

Session management is a critical consideration for building secure software. As such, the appropriate implementation of session IDs, authentication tokens, and cookies collectively prevent session hijacking attacks. Such deployments are provisioned to forcefully destroy session-associated data on an application server after a subject logs out of the application. Implementing session timeouts that require re-authentication and a fresh token when a user connects to the server after logout is also recommended. Designers and developers should also ensure not to expose session IDs in URLs, as attackers could exploit these for session theft techniques.

06 | Best Practices in Preventing Broken Access Control

Some recommended practices to ensure effective access control include:

Enforce the Law of Least Privilege

Denying access by default for all public resources is one of the first steps to controlling misuse of access privileges. This implies that each user should be granted permissions required to complete a particular function and no more. The law of least privilege implements a zero-trust approach to information exchange that ensures subjects don't have privileges beyond what is necessary.

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 functionalities at the code level. Once the system has authenticated a subject, their privileges to objects should be limited by their roles and identity.

Use Centralized Authorization Routines

Administering access control policies and routines in a centralized location is a recommended approach that helps expedite the application of vulnerability fixes as soon as those are identified. Centralization also eliminates the manual effort required to apply access control policies to every page containing sensitive files and information.

Perform Server-Side Controls

User authentication, input validation, and request processing should be performed at the server-side, as this simplifies the remote management of access control routines. Server-side access control mechanisms also eliminate reliance on traditional keys to enforce privilege decisions, making tracking all access attempts and activities easy.

Test and Audit Access Controls Frequently

Apart from manually testing control mechanisms, it is also recommended to adopt automated scanning tools for continuous monitoring of access control flaws that misalign with an organization's security policy. While continuous testing and vulnerability scanning help teams evaluate whether access control mechanisms are working as intended, such tools also help uncover emerging vulnerabilities within access control systems.

Detecting Broken Access Control Vulnerabilities within Web Apps and APIs with Crashtest Security

Crashtest Security Suite's vulnerability scanners help establish a continuous, automated security testing process that allows teams to uncover access control flaws with extremely low false positives. The security suite integrates with almost all popular software stacks and security platforms, helping to initiate penetration testing within minutes.

Try Crashtest Security for a free, 14-day demo to understand how the security suite can help eliminate access control blind spots while saving time and budget on blackbox pentesting.

START 2-WEEK TRIAL FOR FREE

CHAPTER 06

Cryptographic Failure Prevention

Table of Contents

01 | What is Cryptographic Failure?

Cryptographic failures occur when third-party entities unintentionally expose sensitive data. Insufficient cryptography may arise due to inadequate protection, security misconfiguration, or inappropriate usage of cryptographic algorithms. Also known as sensitive data exposure, a cryptographic failure breaks the cybersecurity trust chain, providing unauthorized access to restricted content.

The vulnerability encompasses a wide range of security flaws when attackers compromise the cryptographic protocol to intercept communication between a sender and the intended recipient. While it is not possible to build a completely uncrackable encryption algorithm, stronger protocols are less likely to be exploited through basic computational capabilities and knowledge. Even with a robust encryption protocol, security professionals may ignore data transmission secure design best practices, subsequently exposing sensitive information.

02 | Types of Cryptographic Failure

Cryptographic failures are categorized according to the specific flaw that results in the exposure of sensitive data, such as user credentials and health records. Some common forms of cryptographic failures include:

Use of Hard-coded Password

This is a commonly found security flaw in which the software exposes plain-text passwords and other secrets within the source code. Listed as CWE-59, the flaw has two variations:

Inbound

The software contains embedded credentials within its authentication mechanism in this scenario. When an account is created, the application assigns a simple default password that is hard-coded and associated with the account. This password is the same for each product deployment that super-users cannot change without considerable modification of the application's source code repositories. If the password is ever exposed to attacker-controlled password databases, then anybody with knowledge of the client application can exploit it to gain access. Since all installations use this default password for new accounts, attackers can orchestrate deep system-level attacks such as denial of service.

Outbound

This flaw occurs when the vulnerable application contains a hard-coded password for connecting to other systems/components. The vulnerability applies to front-end services authenticating to back-end systems, typically requiring a fixed, complex password. If developers hard-code strong passwords, threat actors can employ advanced hacking techniques to obtain and use them for advanced attacks.

Insufficient Entropy

Cryptographic functions use a pseudo-random number generator (PRNG) to generate hashes, salts, and other random values in encryption protections. In vulnerable applications, a cryptographically weak pseudo-random number generator creates values that are easy for the attacker to guess. Insufficient randomness/entropy offers attackers access to the seed or internal state of PRNG, which generates values or patterns that are easy to predict. Since the application uses these values for authentication and authorization, attackers can gain access to sensitive, restricted information. Insufficient entropy is a base-level weakness in the Common Weakness Enumeration database as entry CWE-331.

Use of Risky/Broken Cryptographic Algorithms

Developers are advised to use standard algorithms such as SHA 256, SHA 384, and SHA 512 that are robust and have been developed to tackle modern threats. When the web application uses a less-secure algorithm for encryption, a determined attacker can break it to compromise the protected message contents. Applications that use outdated, weak encoding and encryption algorithms allow attackers to intercept secured communications and successfully orchestrate a man-in-the-middle attack. The flaw is listed as CWE-331 in the Common Weakness Enumeration database and is attributed to a high-severity vulnerability because of the severe consequences of a successful attack.

Improper Following of a Certificate‘s Chain of Trust

The server certificate is a crucial security feature that helps secure communications by verifying the website‘s ownership and preventing attackers from creating a site clone. In most cases, the certificate traverses multiple intermediate entities who vouch for one another, with the user at the end of the trust chain. Improper following of the certificate‘s trust chain (CWE-296) reduces the usefulness of the certificate. In such cases, the client may receive a certificate and only check the first entity within the chain, which derives no real trust since a valid certificate should be traced back to the source.

A trust chain can be broken due to several different ways, including:

- Self-signed, non-root certificates
- Improper validation of intermediate certificates
- Lack of expected primary constraints and crucial extensions in intermediate certificates
- Compromised/broken root certificates

Reversible One-Way Hash

This vulnerability occurs in websites that use hashing algorithms that can be used to generate the original input. Hackers leverage a brute-force attack to determine the actual value or find a value that can generate the same hash functions. This enables successful cryptanalysis of the hashing algorithms. Attackers also exploit the reversible one-way hash (CWE-328) flaw to bypass authentication by entering an alternate password that generates the identical pre-calculated hashes, enabling them to take over a victim‘s account entirely.

Unprotected Transport of Credentials

This flaw occurs when an application fails to protect the username and password while in transit from the login database to the application server. The login page fails to apply effective security measures to the credentials as they are transmitted from the client machine to the server. Websites commonly use a secure sockets layer (SSL) to ensure the integrity and confidentiality of credential data. An improper SSL implementation for the login page allows attackers to eavesdrop and alter user credentials, leading to compromised accounts.

03 | Cryptographic Failure - Severity Level

Moving up from number five on the 2017 top vulnerabilities list, the cryptographic failure vulnerability is ranked number two on the 2021 OWASP top 10 lists. As of 2021, 29 common weaknesses and enumerations were mapped to the cryptographic failure vulnerability. The flaw has an average incidence rate of 4.49%, a weighted exploit of 7.29, and an average impact score of 6.81, making it a high-priority vulnerability.

04 | How to Identify Cryptographic Failure with Crashtest Security?

The Crashtest Security Suite provides advanced ethical hacking and automated vulnerability scanning functionalities to keep websites safe from cryptographic attacks. Crashtest Security's black box penetration testing helps security professionals identify cryptographic attack vectors in web applications and APIs even with limited knowledge of client libraries, application architecture, or internal traffic procedures.

The platform employs various automated scanners for identifying potential vulnerabilities within cryptographic systems. These include:

- **Privilege escalation scanner** helps identify accounts that may have been compromised through an insufficient transport of credentials
- **HTTP header scanner** helps developers identify cryptographic attack attempts via malformed requests.
- **SSL/TLS scanner** identifies flaws in implementing HTTPS communication, which may lead to encrypted content interception.

Crashtest Security also offers comprehensive security reports that help analyze every identified vulnerability in detail and provide remediation guidance so that developers and security professionals don't waste time researching security controls.

05 | Cryptographic Failure Prevention Techniques

Encryption protections form the first line of defense against cryptographic failures. Other security measures to prevent cryptographic failures include:

HTTP Strict Transport Security (HSTS)

The HSTS policy mechanism helps prevent man-in-the-middle attacks by informing the client application that the website should only be accessed via HTTPS. Since attackers can intercept the initial HTTP connection, all incoming HTTP requests should be converted to HTTPS by default and are more effective than a server-side HTTP to HTTPS redirect.

HSTS is defined as a header field within the HTTP response named strict-transport-security. The HSTS protocol is supported by all major browsers and is effective against active or passive cryptographic attacks.

Server-side Cipher Prioritization

The server should be configured to evaluate the list of supported cipher suites, check which ones are compatible with the client, and use it during the handshake of the connection. The selected cipher suite is a combination of:

- The key generation and exchange algorithm
- Authentication algorithm
- Bulk encryption algorithm
- Message Authentication Code (MAC) algorithm

06 | Best Practices in Preventing Cryptographic Failures

Some best practices to prevent cryptographic attacks include:

Discard Unnecessary Data

To avoid exposure to sensitive data, developers should avoid storing the data unnecessarily. A recommended approach is to employ message truncation or PCI-DSS compliant tokenization to replace sensitive data with non-sensitive placeholders or remove a portion of the data altogether.

Enforce Cryptographic Randomness Using Appropriate Initialization Vectors (IVs)

An IV should be chosen appropriately to suit the same mode of operation. For most applications, this means using a cryptographically secure pseudo-random number generator. It is also recommended that the initial vector is not used twice for proper key management.

Avoid Legacy Protocols When Transporting Sensitive Data

Legacy protocols such as SMTP and FTP include insecure design flaws that hackers can crack with minimal resources and within a reasonable time frame. Modern, advanced, mainstream algorithms address these flaws to ensure data is protected at rest and in transit.

Detect Cryptographic Failures with Crashtest Security

Crashtest Security Suite ensures web applications, web servers, and APIs are free of cryptographic failures by leveraging a comprehensive vulnerability scanning and penetration testing process. The security suite integrates seamlessly into modern development stacks, ensuring configuration effectiveness in cryptographic operations.

To know more about how Crashtest Security can help detect potential vulnerabilities in your web application before they lead to a security incident, try a 14-day, free demo:

START 2-WEEK TRIAL FOR FREE

CHAPTER 06

Insecure Design Attack Prevention

Table of Contents

01 | What is Insecure Design?

Insecure design is a broad vulnerability set encompassing multiple implementation defects introduced during an application's design phase. The vulnerability set covers over 40 Common Weaknesses and Enumerations and critical security risks in application architecture.

When developing business logic, design flaws often stem from ignoring 'shift-left' practices and the lack of sufficient profiling. Contrary to popular belief, flaws introduced by insecure design cannot be fixed in the implementation stages; hence it is recommended to ensure such flaws are addressed during the initial stages of design.

02 | Types of Insecure Design Vulnerability

The insecure design includes over 40 vulnerabilities that introduce application architecture and design flaws. Some of the most common insecure design flaws include:

Unprotected Storage of Credentials

Some web deployments include plaintext storage of user passwords that may compromise system access through unauthorized access to user accounts. Due to improper configuration, user credentials get stored within the web server's memory, configuration manifests, or application properties, allowing malicious users to access and exploit legitimate accounts.

External Control of Path or File Name

CWE-73 represents a highly-susceptible form of insecure design that allows malicious users to modify critical flows or sensitive files crucial to application logic. This flaw introduces file/path manipulation errors when the application relies on user input to control key flows and influence the file paths used in software operations. Attackers leverage file manipulation errors to specify paths or resources used in these operations, granting them privileges that are otherwise restricted.

Storing Passwords in a Recoverable Format

Some applications rely on recoverable encryption for passwords, which provides no significant benefits over storing cleartext passwords. Malicious users can leverage insufficient credential recovery workflows to orchestrate password reuse attacks. Passwords that can be recovered directly or discovered through a brute force search of available information can also offer hackers unlimited access to user accounts.

Unrestricted Upload of Files

This vulnerability occurs when an application fails to validate the file extension, eventually allowing an attacker to transfer or upload a malicious file to be processed within the software environment. Uploaded files introduce critical security risks, including broken access controls and complete system takeover. In instances where the application fails to implement restrictions on the number or size of uploaded files, attackers may initiate a denial of service attack by overloading the web server or database.

Inconsistent Interpretation of HTTP Requests

This vulnerability is commonly found in intermediate agents between web servers and clients, such as HTTP proxies and firewalls. When an intermediate agent interprets client requests inconsistently, the flaw allows hackers to smuggle malformed requests to one entity without the other entity being aware. In some cyber-attacks, malicious users can also add a new, malicious request to persistent data flow via HTTP pipelining, which the intermediate agent interprets as part of the first legitimate request, enabling undetected HTTP request smuggling.

Use of a Browser Cache Containing Sensitive Information

Most deployments rely on client-side caching to improve performance by pooling previously processed information locally. This reduces the time required to initialize, process, and access information requested by users. Design flaws in the caching process often leave these resources available to malicious actors. Some clients expose cache data in public terminals, allowing hackers to access and exploit sensitive information such as credit card numbers and login credentials.

Insufficient Compartmentalization

This weakness results from the lack of isolation between application logic or functions that require different levels of rights, privileges, or permissions. Additionally, in some web environments, security professionals and development teams use only one factor for a security decision, reducing the effectiveness of integrity checks. Once an attacker has gained unauthorized access to a single functionality, they can orchestrate vertical or horizontal privilege escalation, allowing them to increase the exploit surface and blast radius of their attack.

External Control of Critical State Data

A severe vulnerability is found in applications that store critical state information about the software or user sessions in a location that is available to attack. If the application allows the hackers to modify this state data without being detected, they can craft malformed requests for restricted resources or perform unauthorized actions. Applications store state information in various locations, such as a web form field, input argument, database record, or a publicly accessible cookie. Attackers also use information stored in these locations to change state information for security controls, creating an unforeseen attack vector.

03 | Insecure Design - Severity Level

Insecure design vulnerabilities rank fourth on the OWASP 2021 Top 10 vulnerabilities list (A04:2021). The exposure is mapped to 40 Common Weaknesses and Enumerations, with an average incidence rate of 3%. Insecure design flaws have an average weighted exploit score of 6.46 (moderate) and an average impact score of 6.78 (medium).

Some potential effects of attacks targeting insecure design flaws include:

Loss of income - Attackers can exploit insecure design vulnerabilities to threaten the model of a business process and avoid revenue-generating paywalls
User and system enumeration - Attackers rely on exposed credentials, cached information, file paths, and other insecure design errors to build a logical map of application servers and user accounts. Hackers commonly use enumeration information to create and test possible attack scenarios to achieve this.
Data breaches - By leveraging weaknesses in file path implementation, attackers can build exploits to facilitate the disclosure of records holding sensitive information. This information can be sent to attacker-controlled databases or used for ransomware attacks.
Denial of service - Malicious actors can spoof web servers or databases with multiple malformed requests or by uploading malicious files that reduce the server's ability to handle legitimate user requests.
Advanced cyber attacks - Hackers leverage insufficient compartmentalization errors to expand the scope of their exploit. Attackers also use compromised passwords to escalate privileges, targeting a more significant number of user accounts in applications lacking strong boundaries.

04 | How to Identify Insecure Design?

As insecure design flaws typically occur in the earlier stages of the software development life cycle, it is essential to detect and identify them during the initial design phases of application development. Some common approaches to identifying insecure design vulnerabilities include:

Requirements and Resource Examination

When collecting business requirements for application conceptualization and development, security teams should proactively identify various defense mechanisms that will be leveraged to protect underlying resources and data assets. It is also essential that during the design phase, the development team considers all resources and additional components that will be exposed to the public and thereby implement integrity checks for safe access. Developers should also take into account the separation of privilege required to ensure there are robustly secure boundaries.

Vulnerability Scanning

Automated vulnerability scanning can help identify insecure design flaws across all components required to build an application. As a recommended practice, pre-code vulnerability scanning should be performed on all infrastructure resources identified during the requirement gathering phase to mitigate common weaknesses and enumerations before they get used in the SDLC.

Penetration Testing

Pentests form the core of pre-code threat modeling, as they offer developers and security professionals a glimpse into common patterns of attacker activity. Using penetration tests, application design teams assess how attackers identify and exploit insecure design flaws, allowing for proactive threat remediation.

05 | Insecure Design Prevention Techniques

Insecure design preventive approaches include:

Tier Layer Segregation

Most deployments rely on client-side caching to improve performance by pooling previously processed information locally. This reduces the time required to initialize, process, and access information requested by users. Design flaws in the caching process often leave these resources available to malicious actors. Some clients expose cache data in public terminals, allowing hackers to access and exploit sensitive information such as credit card numbers and login credentials.

Threat Modeling

Security teams should adopt diligent threat modeling practices for all business logic, access controls, authentication, and key flows within the application to proactively identify threat patterns and related countermeasures.

Unit and Integration Testing

Unit tests help identify design flaws in each component/unit of code used to design and build an application. Integration tests help identify and mitigate critical vulnerabilities at the interface of two or more components used in the application development cycle. These tests help application developers and security professionals address design flaws before deploying code in production, which helps prevent implementation defects from being pushed further down the deployment pipeline.

06 | Best Practices in Preventing Insecure Design Vulnerabilities

Some best practices to prevent insecure design vulnerabilities include:

Establishing Secure Design Patterns

Security professionals should establish a library of secure design patterns that can be reused for comprehensive application security. These patterns should include base design components and complementary program components that provide a repeatable, hardened process for rapid, secure code development.

Eliminating Unused Features and Components

Idle features and frameworks introduce an extensive attack surface that adds to the workload for security professionals. Using minimal libraries and frameworks without unnecessary features, components, samples, and documentation eliminate avoidable attack vectors, helping achieve a secure design at the granular level.

Embracing Microservice Architectures

The microservice-based design offers effective compartmentalization between tenants, functions, and entities, which reduces the blast surface of a successful insecure design exploit. Microservice architectures also aid granular resource management for easier network and system layer tier segregation.

CRASHTEST **SECURITY**

WWW.CRASHTEST-SECURITY.COM