CRASHTEST **SECURITY**

GUIDE FOR

PREVENTING INSECURE DESIGN ATTACKS

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

GUIDE FOR INSECURE DESIGN ATTACKS

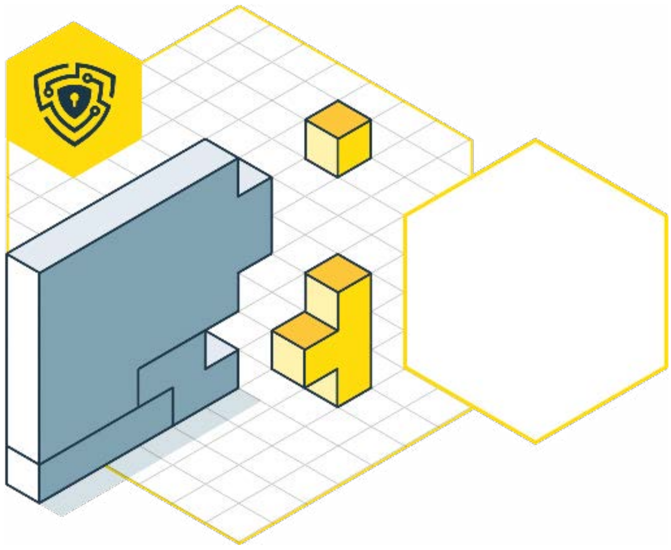
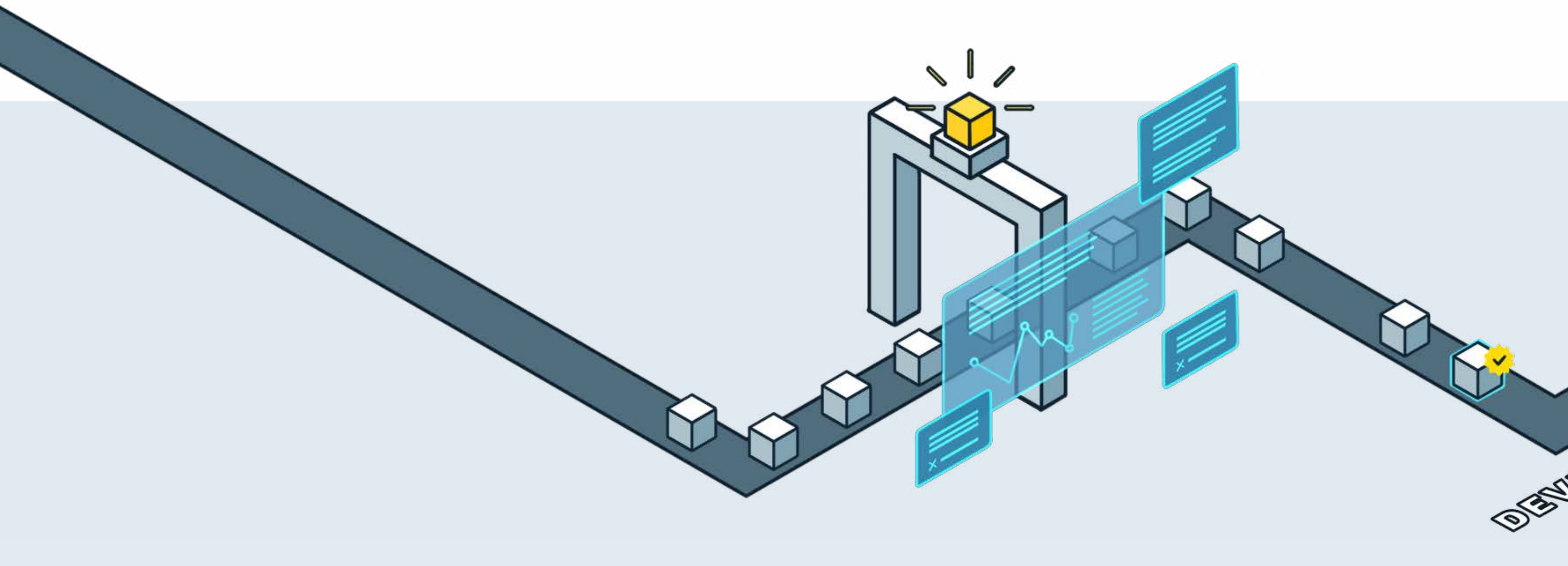


Table of Contents

INTRODUCTION TO THIS GUIDE

To tackle the continuously changing threat landscape, it is recommended to adopt preventive approaches in the initial stages of application development to help reduce the attack surface and prevent inherent design flaws. Some commonly embraced security controls encompass secure design patterns, threat modeling, and reference architectures that help mitigate application vulnerabilities in earlier stages of the software development lifecycle. An insecure design flaw is an application security risk that arises from the lack of effective implementation of these security controls. This guide discusses insecure design vulnerabilities, their types, severity level, and best prevention practices.



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.

TYPES OF INSECURE DESIGN VULNERABILITIES

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.

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.

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.

INSECURE DESIGN PREVENTION TECHNIQUES

Insecure design preventive approaches include:

TIER LAYER SEGREGATION

Application development teams should build different tier layers to establish a privilege separation for users that require different permissions. The segregation should be applied to the system and network layers to enhance isolation based on exposure and protection needs. Additionally, design teams should separate tenants by role or rate of consumption per user to limit resource consumption by different network entities.

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.

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.

DETECTING, IDENTIFYING, AND MITIGATING INSECURE DESIGN FLAWS WITH CRASHTEST SECURITY

The Crashtest Security Suite helps implement automated threat modeling through continuous vulnerability scanning and black box penetration testing. With Crashtest Security, organizations can identify common weaknesses and attack signatures for insecure design exploits through a few simple steps. Some vulnerabilities exposed by the Crashtest Security Suite include privilege escalation, file inclusion, HTTP smuggling vulnerabilities (HTTP header scanner), and more.

Sign up for a free, 14-day trial to discover how Crashtest Security's automated scanning helps verify the secure design of your organization's application stack.

Start 2-Week Trial for Free

CRASHTEST **SECURITY**

WWW.CRASHTEST-SECURITY.COM