# Web Application Security Risks: Identifying, Modelling, and Mitigating Threats

A comprehensive guide to understanding, identifying, and defending against modern web application security threats through proactive threat modelling and robust security practices.

# Chapter 1: The Growing Threat Landscape of Web Applications

As web applications become increasingly central to business operations and daily life, they have simultaneously become prime targets for sophisticated cyber attacks. The modern threat landscape is evolving rapidly, with attackers developing new techniques to exploit vulnerabilities in complex web architectures. Understanding this landscape is the first step towards building resilient, secure applications that can withstand the challenges of today's interconnected digital ecosystem.

## $6 Trillion: The Cost of Cybercrime in 2025

**$6T**

### Global Cybercrime Cost

Projected annual cost of cybercrime by 2025

**43%**

### Targeting Web Apps

Percentage of attacks focused on web applications

**3.5M**

### Data Records Stolen

Average records compromised per web breach

The financial impact of cybercrime continues to escalate at an alarming rate, with web security breaches causing widespread disruption to businesses, significant intellectual property theft, and irreparable damage to organisational reputations. Attackers are increasingly targeting web applications due to their broad attack surface, valuable data repositories, and critical role in business operations. The sophistication of these attacks ranges from automated bot-driven campaigns to highly targeted advanced persistent threats (APTs) designed to infiltrate specific organisations.

Web applications represent an attractive target because they are publicly accessible, often handle sensitive customer data, and serve as gateways to internal systems. A single vulnerability can expose millions of user records, disrupt essential services, and result in regulatory fines, legal liabilities, and long-term reputational damage that extends far beyond the immediate financial cost.

# Common Web Application Threats Today

## SQL Injection

Malicious SQL statements inserted into input fields to manipulate databases, extract sensitive data, or bypass authentication mechanisms

## Cross-Site Scripting (XSS)

Injecting malicious scripts into web pages viewed by other users, enabling session hijacking and data theft

## Cross-Site Request Forgery

Forcing authenticated users to execute unwanted actions without their knowledge or consent

## Distributed Denial of Service (DDoS)

Overwhelming web servers with massive traffic volumes to render applications unavailable to legitimate users, causing operational disruption and financial losses

## Credential Theft & Privilege Escalation

Stealing user credentials through phishing, brute force attacks, or data breaches, then exploiting weak access controls to gain elevated permissions within the application

**Real-World Example:** In 2020, over 1.3 million WordPress sites were targeted in a massive coordinated attack exploiting vulnerabilities in outdated plugins and themes. The attack demonstrated how automated tools can rapidly scan and exploit common vulnerabilities across millions of websites simultaneously, highlighting the critical importance of timely security updates and proactive vulnerability management.

## Why Web Apps Are Vulnerable

### Architectural Complexity

Modern web applications are built upon complex architectures featuring multiple interconnected components, third-party integrations, microservices, APIs, and cloud-based infrastructure. Each component introduces potential vulnerabilities, and the interactions between them create additional attack vectors that may not be immediately apparent during development.

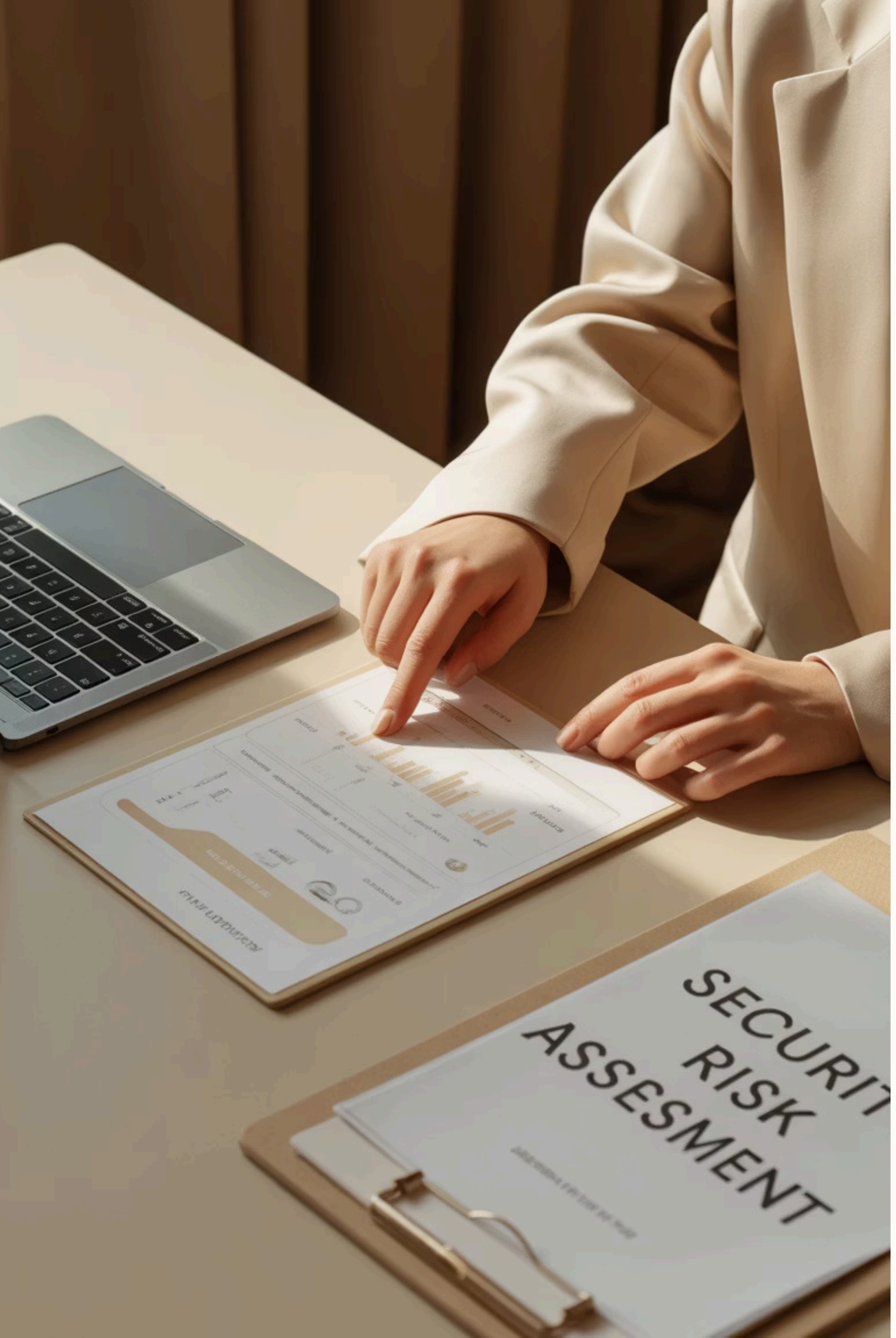
- Multiple layers: presentation, business logic, data access, external services
- Third-party libraries and dependencies with their own vulnerabilities
- Complex authentication and authorisation flows across distributed systems
- Integration points with external APIs and services

### Development Pressures

Rapid development cycles and pressure to deliver features quickly often result in security being treated as an afterthought rather than being integrated into the design phase. Organisations face constant tension between speed-to-market and security thoroughness.

- Security testing postponed until late in development cycle
- Insufficient security training for development teams
- Technical debt accumulating from rushed implementations
- Inadequate code review processes and security testing

The increasing adoption of APIs and cloud services has dramatically expanded the attack surface of modern web applications. Cloud infrastructure introduces shared responsibility models where misconfigurations can expose sensitive data, whilst APIs create new entry points that must be properly secured, authenticated, and monitored. The distributed nature of cloud-native applications means that security must be considered at every layer, from infrastructure to application code.



# Chapter 2: Identifying Application Security Risks

Effective security begins with thorough risk identification. Before implementing defensive measures, organisations must systematically identify their valuable assets, understand their application architecture, map potential attack vectors, and assess the likelihood and impact of various threats. This chapter explores proven methodologies for discovering and cataloguing security risks within web applications, providing a foundation for targeted security improvements.

## What Are We Protecting? Identifying Assets

### User Data & Privacy

Personal information, authentication credentials, payment details, health records, and other sensitive user data requiring protection under regulations like GDPR, CCPA, and PCI DSS

### Application Logic & IP

Proprietary algorithms, business logic, trade secrets, source code, and intellectual property that provide competitive advantage and represent significant organisational investment

### Infrastructure Assets

Servers, databases, APIs, cloud resources, network infrastructure, and computational resources that enable application functionality and require protection from unauthorised access

Asset identification forms the foundation of any security strategy. Understanding what needs protection enables teams to prioritise security investments, implement appropriate controls, and assess the potential impact of security incidents. Different assets require different levels of protection based on their sensitivity, value, and regulatory requirements.

> "You cannot protect what you do not know you have. Comprehensive asset inventory and classification is the cornerstone of effective security risk management."

# Mapping the Attack Surface: Architecture Diagrams

Visualising application architecture through detailed diagrams is essential for identifying potential vulnerabilities and attack vectors. Architecture diagrams reveal the flow of data between components, highlight trust boundaries where security controls must be enforced, and expose potential weaknesses in the application's design.

01

## Document Client-Side Components

Map all user-facing interfaces including web browsers, mobile apps, and single-page applications (SPAs) that interact with backend services

02

## Identify Server-Side Services

Catalogue application servers, APIs, microservices, authentication services, and business logic components that process requests

03

## Map External Integrations

Document third-party services, payment gateways, analytics platforms, and external APIs that the application depends upon

04

## Define Trust Boundaries

Mark boundaries where data crosses from untrusted to trusted zones, requiring validation, authentication, and authorisation controls

**Example:** A data flow diagram showing user input from a web form crossing into backend services illustrates a critical trust boundary. At this boundary, input validation, parameterised queries, and output encoding must be implemented to prevent injection attacks. Without clear visualisation, developers might overlook these crucial validation points.

# The STRIDE Framework for Threat Identification

STRIDE is a systematic threat modelling framework developed by Microsoft that helps security professionals identify and categorise potential threats. Each letter represents a different category of threat that can compromise application security.

### Spoofing Identity

An attacker impersonates another user or system component by stealing credentials, forging authentication tokens, or exploiting weak authentication mechanisms. Common examples include credential stuffing attacks, session hijacking, and man-in-the-middle attacks that intercept authentication data.

### Tampering with Data

Unauthorised modification of data in transit or at rest, including SQL injection attacks that alter database contents, manipulation of request parameters, or modification of files stored on the server. Attackers may change prices in e-commerce applications, alter user permissions, or corrupt critical data.

### Repudiation

Users deny performing actions they actually carried out, or attackers cover their tracks by deleting logs and audit trails. Without proper logging and non-repudiation mechanisms, organisations cannot prove that specific actions were taken, leading to disputes and potential fraud.

### Information Disclosure

Unauthorised access to sensitive information through vulnerabilities like unencrypted data transmission, exposed configuration files, verbose error messages, or inadequate access controls. Attackers may extract customer data, intellectual property, or system information that facilitates further attacks.

### Denial of Service

Disrupting application availability by overwhelming resources, crashing services, or exploiting algorithmic complexity vulnerabilities. Includes DDoS attacks, resource exhaustion, and application-layer attacks that make services unavailable to legitimate users, causing operational and financial damage.

### Elevation of Privilege

Gaining unauthorised elevated access rights by exploiting vulnerabilities in access control, privilege management, or authentication systems. Attackers may escalate from ordinary user to administrator, accessing sensitive functions and data reserved for privileged accounts.

# Real Vulnerabilities Behind STRIDE Threats

Understanding how specific vulnerabilities map to STRIDE threat categories helps security teams prioritise remediation efforts and implement appropriate countermeasures. Real-world vulnerabilities often enable multiple STRIDE threat types simultaneously.

### SQL Injection

Enables both Tampering by allowing unauthorised database modifications and Information Disclosure by extracting sensitive data through malicious queries

### Weak Authentication

Leads directly to Spoofing as attackers can easily impersonate legitimate users through credential guessing, brute force, or exploitation of predictable session tokens

### Missing CSRF Tokens

Allows Cross-Site Request Forgery attacks where attackers force authenticated users to execute unwanted actions, enabling Tampering with data and Repudiation concerns

### Misconfigured Servers

Opens doors for Denial of Service attacks, Information Disclosure through exposed configuration files, and Elevation of Privilege via default credentials

# Chapter 3: Threat Risk Modelling – A Proactive Defence

Threat modelling represents a fundamental shift from reactive security patching to proactive risk identification and mitigation. By systematically analysing applications for potential threats during the design and development phases, organisations can identify and address vulnerabilities before they are deployed to production environments, significantly reducing both security risks and remediation costs.

# What is Threat Modelling?

Threat modelling is a structured, proactive approach to identifying, prioritising, and mitigating security risks throughout the software development lifecycle. Rather than waiting for vulnerabilities to be discovered through penetration testing or, worse, exploited by attackers, threat modelling helps development and security teams think like attackers to anticipate potential exploits before they materialise.

This iterative process should be integrated early in the SDLC and revisited as the application evolves. Each significant architectural change, new feature, or integration point warrants reassessment to ensure new threats are identified and addressed.

### Early Integration

Begin threat modelling during the design phase when architectural changes are less costly and easier to implement

### Collaborative Process

Involves developers, architects, security professionals, and business stakeholders working together to identify risks

### Living Document

Threat models are continuously updated as the application evolves, ensuring ongoing security awareness

# The Four Key Questions of Threat Modelling

Effective threat modelling centres around four fundamental questions that guide the entire process, from initial scoping through validation. These questions provide a systematic framework for thorough security analysis.

01

## What are we working on?

Define the scope of analysis, identify assets requiring protection, document the architecture, and understand the application's purpose, users, and data flows. This foundational step ensures all stakeholders have a shared understanding of what needs protection.

02

## What can go wrong?

Systematically identify potential threats and vulnerabilities using frameworks like STRIDE, analyse attack vectors, and consider various threat actors and their motivations. This step requires thinking like an attacker to anticipate potential exploits.

03

## What are we going to do about it?

Design and implement appropriate mitigations and security controls for identified threats, prioritising based on risk level. Solutions may include technical controls, architectural changes, or acceptance of certain risks with appropriate justification.

04

## Did we do a good job?

Validate that implemented controls effectively mitigate identified threats through security testing, code reviews, and penetration testing. Continuously refine the threat model based on findings and evolving threats.

> These four questions create a continuous improvement cycle, ensuring that security considerations remain central throughout the application lifecycle rather than being treated as a one-time exercise.

# Using Data Flow Diagrams (DFDs) in Threat Modelling

Data Flow Diagrams are essential tools in threat modelling that visualise how information moves through an application, making it easier to identify security-critical points where threats may materialise. DFDs help teams understand complex system interactions and pinpoint where security controls must be implemented.

## Visualise Data Movement

DFDs map data flows between components, showing how information enters, moves through, and exits the system. Each flow represents a potential attack vector that must be secured.

## Identify Trust Boundaries

Trust boundaries mark transitions between different security zones, such as user input crossing into server-side processing. These boundaries are critical points where validation, authentication, and authorisation must occur.

## Enforce Security Controls

By visualising data flows and trust boundaries, teams can determine exactly where validation, encryption, and access controls must be enforced to prevent unauthorised access or data manipulation.

**Example:** An IoT device transmitting sensor data to a cloud platform crosses multiple trust boundaries. The DFD would show: device → gateway → cloud API → database. At each boundary, different security controls are needed: device authentication, encrypted transmission, API authorisation, and encrypted storage. Without visualising these flows, teams might overlook critical security checkpoints.

# Prioritising Risks: Likelihood vs Impact

Not all threats pose equal risk. Effective security requires prioritising threats based on both their likelihood of occurrence and their potential impact on the organisation. This risk-based approach ensures that security resources are allocated efficiently to address the most significant threats first.

## Assessing Likelihood

Consider factors that influence the probability of a threat materialising:

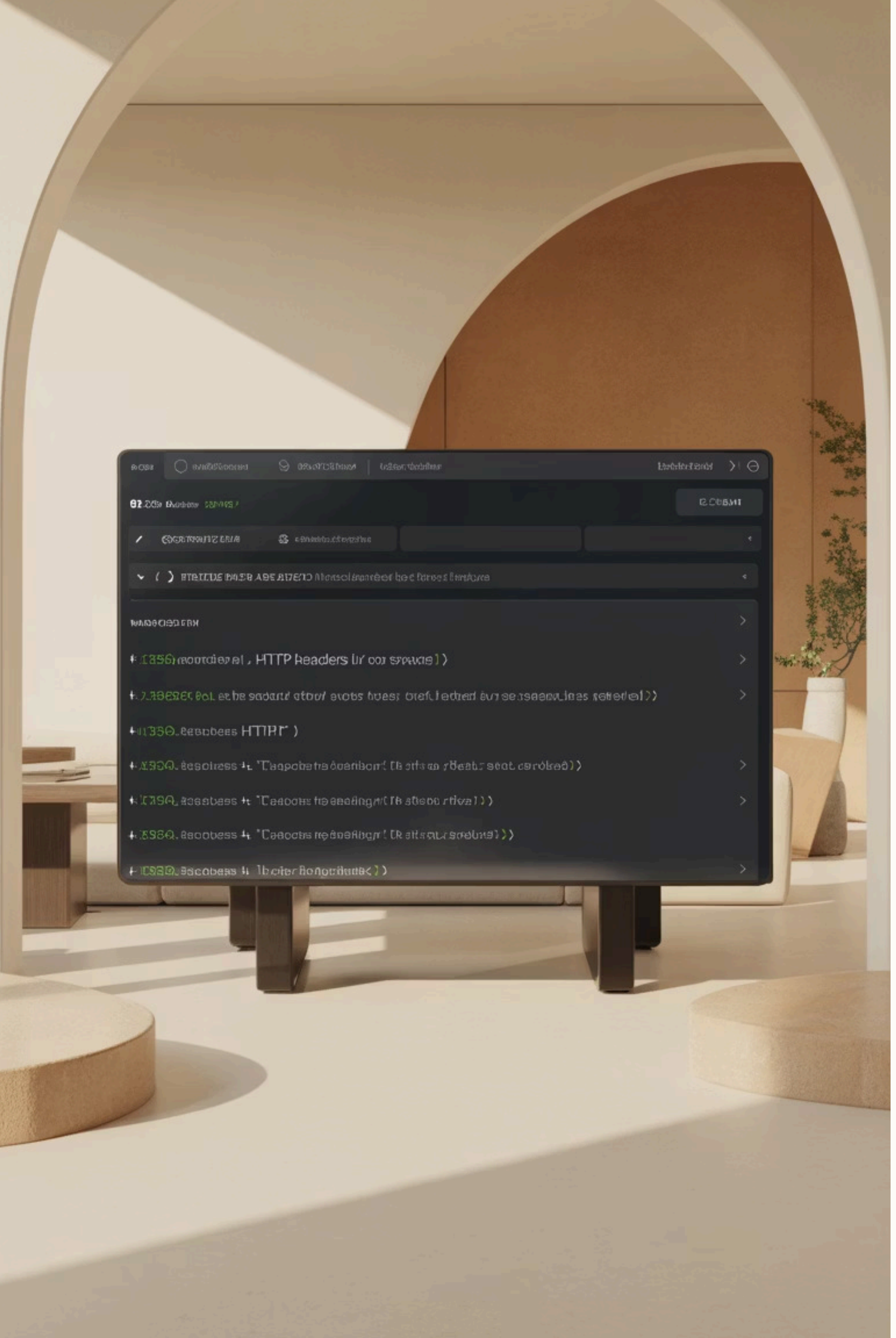
- **Threat actor motivation:** How attractive is your application to attackers?
- **Attack complexity:** How difficult is the vulnerability to exploit?
- **Existing controls:** What security measures are already in place?
- **Historical data:** Have similar attacks occurred previously?
- **Exposure:** How accessible is the vulnerable component?

## Evaluating Impact

Assess the potential consequences if a threat is realised:

- **Data breach:** Scope and sensitivity of potentially exposed data
- **Financial loss:** Direct costs, fines, and revenue impact
- **Operational disruption:** Downtime and service unavailability
- **Reputational damage:** Customer trust and brand perception
- **Regulatory compliance:** Legal and regulatory consequences

**Example:** SQL injection vulnerabilities in a customer database warrant immediate attention due to high likelihood (commonly exploited, well-known attack) and severe impact (potential exposure of millions of customer records, regulatory fines, reputational damage). Conversely, a minor UI rendering bug with no security implications represents low risk and can be addressed in regular maintenance cycles.



# Chapter 4: Other HTTP Fields and Their Security Implications

HTTP headers are often overlooked components of web security, yet they play a crucial role in defending against common attacks. Properly configured HTTP security headers provide additional layers of defence that complement application-level security controls. This chapter explores essential HTTP headers and their security implications, demonstrating how correct configuration strengthens overall application security posture.

## Beyond the Basics: HTTP Headers That Matter for Security

Modern browsers support numerous security-focused HTTP headers that enable web applications to implement additional defensive measures. These headers instruct browsers on how to handle content, connections, and embedded resources, significantly reducing the risk of various attacks.

### Content-Security-Policy (CSP)

CSP is one of the most powerful security headers, preventing Cross-Site Scripting (XSS) attacks by restricting which resources can be loaded and executed on a web page. By defining approved sources for scripts, styles, images, and other content, CSP ensures that only trusted code runs in users' browsers.

```
Content-Security-Policy: default-src 'self'; script-src 'self' https://trusted.cdn.com; style-src 'self' 'unsafe-inline';
```

This policy restricts content to the same origin by default, allows scripts only from the application itself and a trusted CDN, and permits inline styles whilst preventing inline scripts.

### Strict-Transport-Security (HSTS)

HSTS enforces HTTPS connections, protecting against protocol downgrade attacks and cookie hijacking. Once a browser receives an HSTS header, it will refuse to connect to the server over insecure HTTP for the specified duration, ensuring all communication remains encrypted.

```
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
```

This configuration enforces HTTPS for one year, includes all subdomains, and enables preloading so browsers enforce HTTPS even on first visit.

### X-Frame-Options

This header protects against clickjacking attacks by controlling whether the application can be embedded in frames or iframes on other websites. Clickjacking tricks users into clicking on hidden elements, potentially leading to unauthorised actions.

```
X-Frame-Options: DENY
```

DENY prevents any framing, whilst SAMEORIGIN allows framing only by pages from the same origin. For more granular control, use the newer frame-ancestors directive in CSP.

# Lesser-Known HTTP Fields with Security Impact

### Referrer-Policy

Controls the information sent in the Referer header when users navigate away from your site. This protects user privacy and prevents leaking sensitive information in URLs to third parties.

```
Referrer-Policy: strict-origin-when-cross-origin
```

### Permissions-Policy

Formerly Feature-Policy, this header restricts which browser features and APIs can be used, reducing the attack surface by disabling unnecessary capabilities like geolocation, camera access, or payment APIs.

```
Permissions-Policy: geolocation=(), camera=(), microphone=()
```

## Critical Set-Cookie Flags for Session Security

Cookie security flags are essential for protecting user sessions from theft and manipulation. Proper cookie configuration prevents common session-based attacks.

### HttpOnly Flag

Prevents JavaScript from accessing cookies, mitigating XSS attacks that attempt to steal session tokens through client-side scripts.

```
Set-Cookie: sessionId=abc123; HttpOnly
```

### Secure Flag

Ensures cookies are only transmitted over HTTPS connections, preventing interception over insecure networks through man-in-the-middle attacks.

```
Set-Cookie: sessionId=abc123; Secure
```

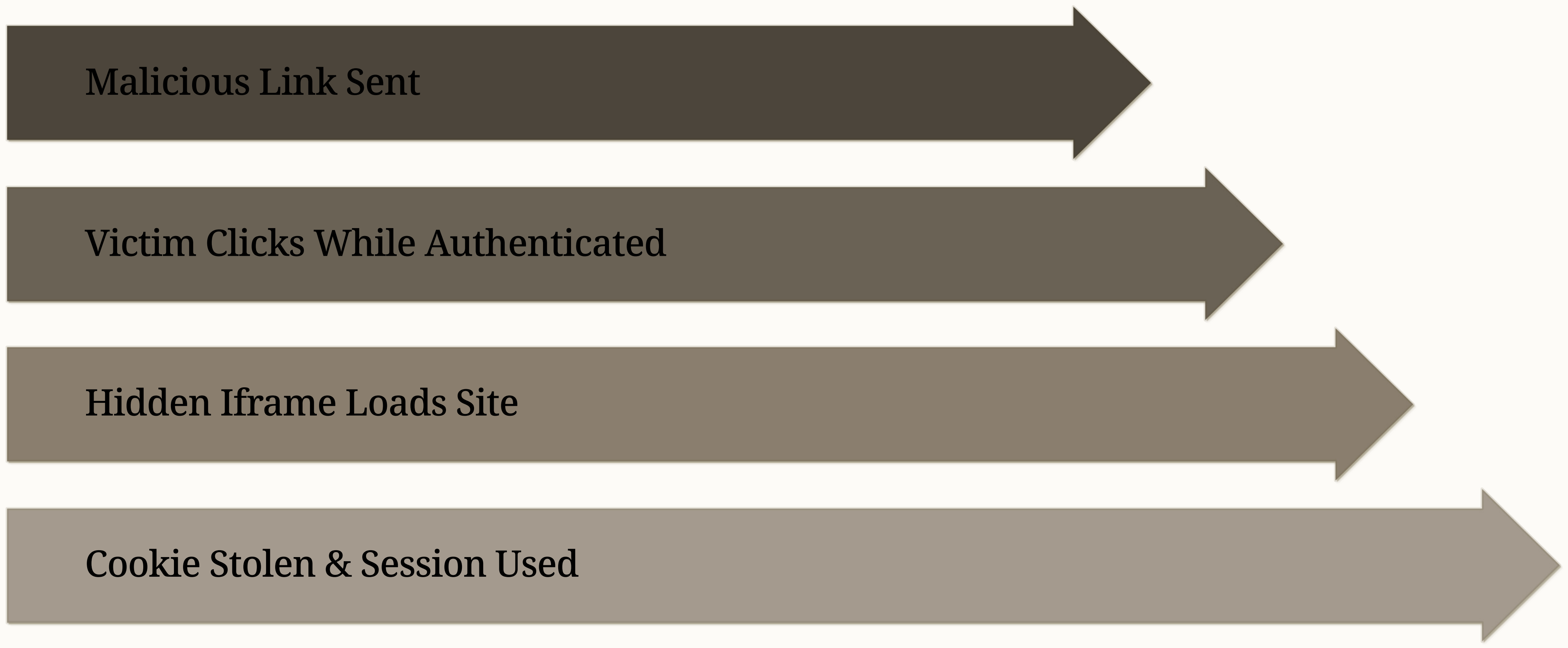
### SameSite Flag

Protects against CSRF attacks by controlling when cookies are sent with cross-site requests. SameSite=Strict provides strongest protection.

```
Set-Cookie: sessionId=abc123;
SameSite=Strict
```

## Visualising an Attack Scenario Using HTTP Fields

Understanding how missing or misconfigured HTTP headers enable attacks helps demonstrate their critical importance. This scenario illustrates how multiple header vulnerabilities can be chained together in a sophisticated attack.


Malicious Link Sent
Victim Clicks While Authenticated
Hidden Iframe Loads Site
Cookie Stolen & Session Used

### Attack Breakdown: Session Hijacking via Multiple Header Vulnerabilities

**Initial Compromise**

Attacker sends phishing email containing link to malicious page, exploiting missing CSP to load malicious scripts and absent X-Frame-Options to embed victim site in iframe

1

2

**Session Extraction**

Malicious JavaScript accesses session cookie (missing HttpOnly flag) and exfiltrates it to attacker's server, facilitated by lack of SameSite protection

3

**Session Hijacking**

Attacker uses stolen session cookie to impersonate victim, gaining full access to their account and sensitive data without needing credentials

This attack succeeds due to **multiple missing security headers**. Each header provides a defensive layer, and when all are absent, attackers can easily chain vulnerabilities together. Proper implementation of CSP, X-Frame-Options, HttpOnly, Secure, and SameSite flags would have prevented this attack at multiple stages.

## Conclusion: Building Resilient Web Applications

Web application security is not a destination but a continuous journey requiring vigilance, adaptation, and collaboration across all stakeholders. As we've explored throughout this presentation, building resilient applications demands a multi-layered approach combining proactive threat modelling, comprehensive risk identification, and proper security configuration at every level.

1

### Early and Continuous Threat Modelling

Integrating threat modelling early in the SDLC and maintaining it throughout the application lifecycle is fundamental to reducing risks and costs. Identifying vulnerabilities during design is exponentially cheaper than discovering them in production. Make threat modelling a standard practice for every significant feature or architectural change.

2

### Defence in Depth

Understanding and properly configuring HTTP security headers, cookies, and other protocol-level protections strengthens your defensive layers. These configurations complement application-level security controls, ensuring that even if one layer fails, others provide protection. Never rely on a single security measure.

3

### Collaborative Security Culture

Security is a team effort requiring collaboration between developers, architects, security professionals, operations teams, and business stakeholders. Foster a culture where security is everyone's responsibility, not just the security team's domain. Regular security training and open communication channels are essential.

---

## Call to Action: Start Today

The threat landscape continues evolving, with attackers developing new techniques and exploiting emerging vulnerabilities. Organisations that treat security as an afterthought will inevitably face costly breaches, regulatory penalties, and reputational damage. Those that embrace proactive security practices, integrate threat modelling into their SDLC, and maintain robust security configurations will be positioned to withstand these challenges.

> **Begin your security transformation now:** Conduct a threat modelling session for your current projects, audit your HTTP security headers, review your authentication and session management practices, and establish a regular cadence for security reviews. The investment you make today in proactive security will pay dividends in reduced risk, lower remediation costs, and greater customer trust.

Remember: **it's far easier and less expensive to build security in from the start than to retrofit it after a breach occurs.** Take action today to protect your applications, your users, and your organisation's future.