

# What common web exploits should I know about? [closed]

Asked 16 years, 4 months ago   Modified 6 years, 7 months ago

Viewed 13k times



53



**Closed.** This question needs to be more [focused](#). It is not currently accepting answers.



**Want to improve this question?** Update the question so it focuses on one problem only by [editing this post](#).

Closed 6 years ago.

[Improve this question](#)

I'm pretty green still when it comes to web programming, I've spent most of my time on client applications. So I'm curious about the common exploits I should fear/test for in my site.

security

testing

Share

Improve this question

Follow

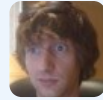
edited Aug 23, 2008 at 15:19



Chris

6,842 ● 6 ● 53 ● 67

asked Aug 22, 2008 at 18:22



Brian Leahy

35.4k ● 12 ● 46 ● 60

13 Answers

Sorted by:

Highest score (default)



34

I'm posting the [OWASP Top 2007 abbreviated list](#) here so people don't have to look through to another link and in case the source goes down.



## [Cross Site Scripting \(XSS\)](#)



- XSS flaws occur whenever an application takes user supplied data and sends it to a web browser without first validating or encoding that content. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, possibly introduce worms, etc.

## [Injection Flaws](#)

- Injection flaws, particularly SQL injection, are common in web applications. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. The attacker's hostile data tricks the interpreter into executing unintended commands or changing data.

## [Malicious File Execution](#)

- Code vulnerable to remote file inclusion (RFI) allows attackers to include hostile code and data, resulting in devastating attacks, such as total server compromise. Malicious file execution attacks affect PHP, XML and any framework which accepts filenames or files from users.

## **Insecure Direct Object Reference**

- A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. Attackers can manipulate those references to access other objects without authorization.

## **Cross Site Request Forgery (CSRF)**

- A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the web application that it attacks.

## **Information Leakage and Improper Error Handling**

- Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data, or conduct more serious attacks.

## **Broken Authentication and Session Management**

- Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys, or authentication tokens to assume other users' identities.

## **Insecure Cryptographic Storage**

- Web applications rarely use cryptographic functions properly to protect data and credentials. Attackers use weakly protected data to conduct identity theft and other crimes, such as credit card fraud.

## **Insecure Communications**

- Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications.

## **Failure to Restrict URL Access**

- Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly.

## [The Open Web Application Security Project](#)

-Adam

Share Improve this answer

answered Sep 7, 2008 at 0:09

Follow



[Adam Davis](#)

93.5k ● 60 ● 271 ● 333

---

The 2013 edition of the top ten list is here:

[owasp.org/index.php/Top\\_10\\_2013-Top\\_10](http://owasp.org/index.php/Top_10_2013-Top_10) – [Adam Davis](#)

Mar 8, 2016 at 15:57

---



28

[OWASP](#) keeps a list of the [Top 10](#) web attacks to watch out for, in addition to a ton of other useful security information for web development.



Share Improve this answer

answered Aug 22, 2008 at 19:03

Follow



[dragonmantank](#)

15.5k ● 22 ● 85 ● 92



---

These three are the most important:



27



- [Cross Site Request Forgery](#)
- [Cross Site Scripting](#)
- [SQL injection](#)



Share Improve this answer

edited Sep 11, 2008 at 22:58

Follow



answered Aug 22, 2008 at 18:23



Patrick

92.4k ● 11 ● 52 ● 61



10



Everyone's going to say "SQL Injection", because it's the scariest-sounding vulnerability and the easiest one to get your head around. Cross-Site Scripting (XSS) is going to come in second place, because it's also easy to understand. "Poor input validation" isn't a vulnerability, but rather an evaluation of a security best practice.



Let's try this from a different perspective. Here are features that, when implemented in a web application, are likely to mess you up:

- Dynamic SQL (for instance, UI query builders). By now, you probably know that the only reliably safe way to use SQL in a web app is to use parameterized queries, where you explicitly bind each parameter in the query to a variable. The places where I see web apps most frequently break this rule is when the malicious input isn't an obvious

parameter (like a name), but rather a query attribute. An obvious example are the iTunes-like "Smart Playlist" query builders you see on search sites, where things like where-clause operators are passed directly to the backend. Another great rock to turn over are table column sorts, where you'll see things like DESC exposed in HTTP parameters.

- File upload. File upload messes people up because file pathnames look suspiciously like URL pathnames, and because web servers make it easy to implement the "download" part just by aiming URLs at directories on the filesystem. 7 out of 10 upload handlers we test allow attackers to access arbitrary files on the server, because the app developers assumed the same permissions were applied to the filesystem "open()" call as are applied to queries.
- Password storage. If your application can mail me back my raw password when I lose it, you fail. There's a single safe reliable answer for password storage, which is bcrypt; if you're using PHP, you probably want PHPpass.
- Random number generation. A classic attack on web apps: reset another user's password, and, because the app is using the system's "rand()" function, which is not crypto-strong, the password is predictable. This also applies anywhere you're doing cryptography. Which, by the way, you shouldn't be doing: if you're relying on crypto anywhere, you're very likely vulnerable.

- Dynamic output. People put too much faith in input validation. Your chances of scrubbing user inputs of all possible metacharacters, especially in the real world, where metacharacters are necessary parts of user input, are low. A much better approach is to have a consistent regime of filtering database outputs and transforming them into HTML entities, like quot, gt, and lt. Rails will do this for you automatically.
- Email. Plenty of applications implement some sort of outbound mail capability that enable an attacker to either create an anonymous account, or use no account at all, to send attacker-controlled email to arbitrary email addresses.

Beyond these features, the #1 mistake you are likely to make in your application is to expose a database row ID somewhere, so that user X can see data for user Y simply by changing a number from "5" to "6".

[Share](#) [Improve this answer](#)

answered Sep 10, 2008 at 21:33

[Follow](#)



[tqbf](#)

9,101 ● 3 ● 24 ● 13



**10**



```
bool UserCredentialsOK(User user)
{
    if (user.Name == "modesty")
        return false;
    else
```





```
// perform other checks  
}
```

Share Improve this answer

edited May 18, 2018 at 9:13

Follow



ColdFire

6,859 ● 6 ● 36 ● 53

answered Aug 22, 2008 at 18:25



Rob Cooper

28.9k ● 26 ● 105 ● 142

---

+1 for funny, yet I don't wanna know how many times I found hard coded authentications in systems... – [tekiegreg](#) Jan 19, 2009 at 23:50

---

In addition, this can open the door to time-based attacks where a valid username with an invalid password would take far less time than an invalid username. A hacker could then figure out a list of user names and work from there. – [nicbou](#) Mar 5, 2014 at 19:19

---



3



SQL INJECTION ATTACKS. They are easy to avoid but all too common.

NEVER EVER EVER EVER (did I mention "ever"?) trust user information passed to you from form elements. If your data is not vetted before being passed into other logical layers of your application, you might as well give the keys to your site to a stranger on the street.

You do not mention what platform you are on but if on ASP.NET get a start with good ol' Scott Guthrie and his article "[Tip/Trick: Guard Against SQL Injection Attacks](#)".

After that you need to consider what type of data you will permit users to submit into and eventually out of your database. If you permit HTML to be inserted and then later presented you are wide-open for Cross Site Scripting attacks (known as XSS).

Those are the two that come to mind for me, but our very own Jeff Atwood had a good article at [Coding Horror](#) with a review of the book "[19 Deadly Sins of Software Security](#)".

Share Improve this answer

answered Aug 22, 2008 at 18:31

Follow



Ian Patrick Hughes

10.4k ● 3 ● 32 ● 37



3



Most people here have mentioned SQL Injection and XSS, which is *kind of* correct, but don't be fooled - the most important things you need to worry about as a web developer is INPUT VALIDATION, which is where XSS and SQL Injection stem from.



For instance, if you have a form field that will only ever accept integers, make sure you're implementing something at both the client-side AND the server-side to sanitise the data.

Check and double check any input data especially if it's going to end up in an SQL query. I suggest building an escaper function and wrap it around anything going into a query. For instance:

```
$query = "SELECT field1, field2 FROM table1 WHERE  
field1 = ' " . myescapefunc($userinput) . "'";
```

Likewise, if you're going to display any user-inputted information onto a webpage, make sure you've stripped any `<script>` tags or anything else that might result in Javascript execution (such as `onLoad=` `onMouseOver=` etc. attributes on tags).

Share Improve this answer

answered Aug 23, 2008 at 11:03

Follow



Steve M

10.6k ● 12 ● 53 ● 63



1

This is also a short little presentation on security by one of wordpress's core developers.

[Security in wordpress](#)



it covers all of the basic security problems in web apps.



Share Improve this answer

answered Aug 22, 2008 at 18:27

Follow



icco

3,084 ● 4 ● 35 ● 49



0

The most common are probably database injection attacks and cross-site scripting attacks; mainly because those are the easiest to accomplish (that's likely because those are the ones programmers are laziest about).



Share Improve this answer

answered Aug 22, 2008 at 18:24



Follow



TheSmurf

15.6k ● 3 ● 41 ● 48



0



You can see even on this site that the most damaging things you'll be looking after involve code injection into your application, so XSS (Cross Site Scripting) and SQL injection (@Patrick's suggestions) are your biggest concerns. Basically you're going to want to make sure that if your application allows for a user to inject any code whatsoever, it's regulated and tested to be sure that only things you're sure you want to allow (an html link, image, etc) are passed, and nothing else is executed.

Share Improve this answer

answered Aug 22, 2008 at 18:26

Follow



btw

7,156 ● 9 ● 41 ● 40



SQL Injection. Cross Site Scripting.

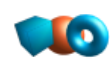
0



Share Improve this answer

answered Aug 22, 2008 at 18:28

Follow



Kibbee

66.1k ● 28 ● 144 ● 184



Using stored procedures and/or parameterized queries will go a long way in protecting you from sql injection.

0



Also do *NOT* have your web app access the database as sa or dbo - set a up a standard user account and set the permissions.



AS for XSS (cross site scripting) ASP.NET has some built in protections. The best thing is to filter input using validation controls and Regex.

Share Improve this answer

answered Aug 22, 2008 at 18:33

Follow



Booji Boy

4,582 ● 4 ● 41 ● 45



0

I'm no expert, but from what I learned so far the golden rule is not to trust any user data (GET, POST, COOKIE).

Common attack types and how to save yourself:



1. [SQL Injection Attack](#): Use prepared queries
2. [Cross Site Scripting](#): Send no user data to browser without filtering/escaping first. This also includes user data stored in database, which originally came from users.

Share Improve this answer

answered Aug 23, 2008 at 9:30

Follow



Imran

90.8k ● 26 ● 100 ● 131