Attack	Description	Things to Look Out for	Test Cases
File Upload	Uploading arbitrary files can be problematic depending on the context in which they can be accessed. Uploading an image may be relatively benign, but if the attacker can upload script or executable files they may be able to execute arbitrary code on the server or use the server as a malware repository.	Anywhere a file can be uploaded could be vulnerable.	• Attempt tampering with file names, encoding, path, extension, or contents may cause the server to interpret the file differently. • Attempt uploading a JSP file to the server then browse to that file. Does the upload succeed and does the file execute?
Forceful Browsing	Frequently pages are simply hidden from view instead of properly authorized when accessed. Forceful browsing may be simply browsing directly to administrator pages or may be combined with parameter tampering to view pages or reports that shouldn't be visible by the user.	Keep an eye on the URL. As an administrator or other privileged user try browsing to as many authenticated pages as possible. Copy each URL to be used later.	For each authenticated page, attempt accessing the page as an unauthenticated or lower privilege user. Some pages may access different content depending on the URL or URL parameters, verify the content is properly authenticated, by browsing as an unauthenticated or lower privilege user.
Parameter/URL Tampering	Parameters when passed in the URL bar or provided by the client may be tampered with. Often times these parameters are assumed to be immutable and are not validated on the server.	• Hidden values • Dropdown values • Checkboxes • Radio Buttons • URL Parameters	Look at the source code to discover parameters, then use the Firebug plugin to manipulate the parameters before submitting any forms.
Information Disclosure	Information disclosure can help you learn about the application quickly. It can also be a security vulnerability in and of itself. Stack traces, information about the server, data quality responses, and any other information that could be useful to the attacker should be removed from production.	**Server Headers**: Search Google for server information **Error messages:** Think about what this error message tells you about the application and where the error message came from. **Stack Traces**: These allow you to learn more about the application and how it works.	Information Disclosure issues are usually discovered during the course of other testing. Every other test case in this document may provide you information.
SQL Injection (SQLi)	SQL injection vulnerabilities allow an attacker to execute arbitrary SQL commands on the server. This is due to the server failing to properly separate user-supplied data from SQL code.	Frequently simply typing a **'** (single quote) into text boxes will cause SQL error messages (Information Disclosure) to occur which can give you more information about the system and how to attack it. Look out for these types of error messages to know you're on the right track. Also think about what the SQL code may look like on the server.	**Special Characters and Commands** • **'** - a simple single quote is the first test case to use to discover SQL injection • # - the # is a SQL comment and tells the SQL interpreter to stop executing the rest of the line • ; - the semicolon is the end of a command. This may be used to string multiple SQL commands together if supported by the database. • OR, AND - SQL supports logical operators such as

		Logging in may be performed by executing the following SQL code:
		SELECT * FROM Users
		WHERE Username = '[**userprovided**]'
		AND Password = '[**userprovided**]';
		
		Typing in a single quote into the username field will cause this SQL command to be invalid, which will cause an error message.
		You may be able to bypass the authentication of this system by adding the right SQL command.
			"or" and "and"
			• <, = - SQL supports comparison operators
			#####Exploitation - ' OR 1=1 # - this statement closes a string and 1=1
			always resolves to true. Any statement or'd with true is always true. This means the rest of the statement will be true. Good for bypassing authentication and logins.

Attack	**Description**	**Things to Look Out for**	**Test Cases**
Cross Site Scripting (XSS)	XSS allows an attacker to inject client side code (HTML, JavaScript, etc.) into the page such that it is rendered on the client. XSS can come in three types: **Reflected**: script is provided by the caller and executed in the browser. e.g. an error message that is rendered in the body of the page that has been provided as a url parameter that is not properly encoded on the server. **Stored**: script is stored in a datastore and added to the body of the page as it is rendered. e.g. a forum that allows users to leave comments for one another. **DOM:** occurs on the client side only, via JavaScript rendering; primarily discovered after the # symbol on the URL or in user provided data.	Any time data or text that you have provided is reflected back to you, there is a possibility for XSS. When you discover a potential injection point look at the context. Different attacks may be possible if the attack string is in HTML, JavaScript, a header, a link or somewhere else. **Example** Consider you attempt a login with username "testAccount" and you see an error message - *Sorry there is no account called testAccount in the system.* This tells us a few things about the system: • It is performing user and password checks separately • It is leaking information about valid users (Information Disclosure Vulnerability) • It is reflecting the username provided back to you, which may be an injection point. Attempt the test cases below.	**Discovery** • <marquee> or <plaintext> - Frequently an opening marquee or plaintext tag can be very useful in flushing out potential areas for XSS injection. The attack string is very obvious when not properly validated or encoded. • ";!--"<XSS>=&{()} - This XSS locator string includes a large number of potentially dangerous characters that should always be encoded. Insert this string into the page and search for "XSS" in the page, validate that each of the characters have been properly validated for context. **Exploitation** • <script>alert(1)</script> - this is the most basic attack string. It will simply pop up an alert box if executed. Frequently this will not work because of basic blacklist filtering on the <script> tag. • <IMG SRC="javascript:alert('XSS');"> - This uses the image tag to execute javascript which can bypass simple script tag checks. • <BODY ONLOAD=alert('XSS')> - Many other tags can be used along with javascript events to execute an attack string. • https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet There are far more ways to execute JavaScript on a website than we have space for here. Go to the URL above for more examples.