# Application Security Attacks and Countermeasures

## Testing Application Dependencies

Applications are heavily dependent on the resources of their host operating system. Testing should be done to ensure that failures in the operating system will not result in unintended or new vulnerabilities in the application. There are different attacks to test for this.

*Attack1:* *Blocking Access to Libraries*; an attacker can exploit the dependency of application software on operating system or third party software libraries for functionality, causing the application to become insecure if the libraries fail to load.

*Countermeasures:*

- Application error handlers should be executed to maintain stability and communicate the error (if appropriate)

*Attack2:* *Manipulating Registry Values*; an attacker can exploit the dependency of application software on operating system registry values to locate and access files, directories and libraries, causing the application to become insecure if the registry values are changed or absent

*Countermeasures:*

- Do not store sensitive information in the registry

*Attack3:* *Using Corrupt Files and File Names*; an attacker can exploit the dependency of application software to read from and write to the file system during normal operation, causing the application to become insecure if the files or filenames are corrupt

*Countermeasures:*

- Ensure that files used exclusively by the application cannot be altered by any other process
- Application error handlers should be executed to ensure that the application can gracefully handle corrupt files or filenames without exposing sensitive information or becoming insecure

*Attack4:* *Manipulating or Replacing Files Created by the Application*: an attacker can exploit the dependency of application software to process information, causing the application to become insecure if the information is corrupt

*Countermeasures:*

- Ensure that information used exclusively by the application cannot be altered by any other process
- Application error handlers should be executed to ensure that the application can gracefully handle corrupt information without becoming insecure

*Attack5:* *Limiting Resource Availability*; an attacker can exploit the dependency of application software to use memory for loading and operating, and disk space or network availability for read and write operations, causing the application to become insecure if the resources are limited or removed

*Countermeasures:*

- Ensure that sufficient memory and hard drive space are available to the application
- Ensure that unused memory can be released for use if necessary
- Ensure that the initial set up of the host operating system and application includes the use of disk partitioning to provide sufficient disk space for expansion

## Testing the Application User Interface

Many security issues related to the user interface are due to unintended and/or undocumented user behavior, or manipulation of the user interface functionality by an attacker. Applications should be able to handle unexpected input without becoming compromised. Attacks that may exploit the application user interface include:

*Attack1:* *Replay Attacks;* and attacker can capture an entire message and send it multiple times to a server, causing to server to repeat the requested operation, leading to the attacker gaining unauthorized access, or the server suffering a self-induced denial of service attack

*Countermeasures:*

- Utilize timestamps from trusted time servers to protect against relayed messages

*Attack2:* *Cookie Hijacking;* an attacker can exploit an application that uses persistent cookies on the user's system to compromise the user's account, if that attacker knows the user's password, has physical access to the user's computer, has administrative network access to the user's computer, has broken into the user's computer, or can see the network to sniff traffic

*Countermeasures:*

- Require a separate login each session
- Provide limited account access without re-authentication

- Ensure all cookies should have a reasonable fixed expiration date that requires re-authentication
- Tie the cookie to identifying information other than the user, such as IP address, user agent string, and so on
- Never store actual user information in cookies; store a token that points to user information on the server's database
- Cookies can be marked secure, preventing their transmittal to non-SSL web pages
- Cookies also have domain and path properties to limit a cookie's scope. If you fail to set boundaries for cookies, it may be possible for an attacker to exploit a cross-site scripting flaw on another web page or even another server to hijack a user's cookie

*Attack3:* *Altering Common Switches and Options*; an attacker can exploit a user interface which allows for the use of command line switches and options, causing the resulting change in configuration of the application (due to the use of a switch or option, such as changing memory allocation) to lead to the application being in an insecure state

*Countermeasures:*

- Test the application for stability under all combinations of common switches and options
- Restrict the code paths that can be manually specified using switches and options
- Use application error handling routines to check configuration input before it is executed

*Attack4:* *Using Escape Characters, Character Sets and Commands*; an attacker can exploit a user interface which allows for the use of special escape characters, character sets and commands, causing the application to become insecure

Countermeasures:

- All allowable characters should be detailed in a documented security standard which addresses:
  - How the commands or characters are being interpreted
  - The language the application is written in
  - The libraries that are used
  - The specific words and strings reserved by the underlying operating system

*Attack5:* *Unvalidated input*; an attacker can tamper with any part of the HTTP request (such as the URL, querystring, headers, cookies, form fields or hidden fields) to try to bypass a website’s security mechanisms

*Countermeasures:*

- Use pre-tested code to ensure that all parameters are validated before they are used.
- Parameters should be validated against a positive specification that defines:
  - Data type (string, integer, and so on)
  - Allowed character set
  - Minimum and maximum length
  - Whether null is allowed
  - Whether the parameter is required or not
  - Whether duplicates are allowed
  - Numeric range
  - Specific legal ranges (enumeration)
  - Specific patterns (regular expressions)

*Attack6:* *Broken access control (authorization)*; an attacker can take advantage of a collection of access control rules for the same application, which were written for different reasons and at different times, and do not provide cohesive protection for the application

*Countermeasures:*

- Use an access control matrix to define the access control rules
- In the security standard, document access rules for types of users, the type of content they can access, and the functions they can perform
- Extensively test the access control mechanism to ensure there is no way to by pass the amalgamated collection of controls

*Attack7:* *Improper error handling*; an attacker can use detailed messages referring to internal system errors to uncover flaws in the web application

*Countermeasures:*

- Error handling should be implemented according to a documented security standard which specifies which information should be reported back to the user, which information should be logged, and so on

*Attack8:* *View Source information;* an attacker can search through the source of each page to find information such as user names, default passwords, e-mail addresses, auto-redirection information and external links in comment fields

*Countermeasures:*

- Do not store sensitive information in the comment fields of the source pages

*Attack9:* *Browsable directories;* an attacker can use default browsable directories (those which show a listing of all files in the directory) to expose unnecessary information

*Countermeasures:*

- Set permissions to prevent access to all the directories that are not necessary to the function of the web server

*Attack10:* *Hidden form fields manipulation;* an attacker can use hidden fields (those not being displayed to the user) to access information the application is storing about user names, passwords, financials, and so on

*Countermeasures:*

- Do not allow hidden input values

## Testing the Application Server

Decisions and changes in the application design and implementation process (that do not go through a proper validation and verification process) can lead to component interaction and inherent flaws that create vulnerabilities in the finished product. IT support staff should have a list of specific security requirements that emphasize:

- Which interfaces their components should extend to the rest of the application
- What form of information will the components receive
- Which computations should be performed on that information

Without this, the implementation will be vulnerable to a number of attacks, such as:

*Attack1:* *Using System Accounts*; an attacker can exploit hidden or undocumented user accounts in an application in which user actions are governed by the assigned level of access an account is given

*Countermeasures:*

- Ensure that user credentials are not cached
- Ensure that the application does not make use of any undocumented or unconfigurable system accounts with elevated privileges that may be exploited by application users

*Attack2:* *Utilizing Unprotected Test Interfaces*; an attacker can exploit applications which allow both documented and undocumented Application Program Interfaces (API's) and software hooks which bypass normal security checks, to be temporarily added to the application for testing purposes, only to become part of the eventual working product

*Countermeasures:*

- Identify all software libraries that are loaded and used by the application, and evaluate their impact on application security

*Attack3:* *Fake the Information Source*; an attacker can exploit an application's need to trust information based on the source of the information in order to function correctly, causing the application to become insecure if the information is corrupt.

*Countermeasures:*

- Ensure that only trusted sources are used, which cannot be compromised or imitated
- Ensure that applications have the ability to verify the source of information
- Ensure that applications have the ability to verify that the level of trust extended to that source is appropriate

*Attack4:* *Unnecessary Ports and Services*; an attacker can exploit an application which opens ports which are not used by the application, but could be exploited by the attacker

*Countermeasures:*

- Scan the application to ensure that it does not attempt to use ports or services that are not necessary for the application's functionality

*Attack5:* *Using Loops with User Input, Script or Code*; an attacker can exploit an application which allows direct user input by executing that input repetitively, causing the application to become deadlocked

*Countermeasures:*

- Ensure that direct user input should not be able to use constructs such as loops to cause denial of service or other lack of availability situations

*Attack6:* *Using Alternative Routes of Task Execution*; an attacker can exploit an application which allows the same task to be executed in more than one way, allowing a route that circumvents security controls to be utilized

*Countermeasures:*

- Each execution path should implement an appropriate security control

*Attack7:* *Forcing the System to Reset Values*; an attacker can exploit an application which allows users to leave the fields in an online input form blank, and then choose *Finish* instead of *Next*; forcing the application to provide initialized variables values where they have not been input, leading to default values and configurations leaving the application in an insecure state

*Countermeasures:*

- Assign a value to a variable as soon as it is declared
- Ensure that all variables are initialized before being used by the application

- Avoid assigning default values and configurations to any variables

*Attack8:* *Get between Time of Check Out and Time of Use*; an attacker may be able to infiltrate a transaction if too much time elapses between the time the information is checked out by the application and the time it is used, resulting in the attacker being able to force the application to perform some unauthorized action

*Countermeasures:*

- Ensure that the time delay between check out and use is minimized
- Ensure that every time sensitive operations are performed, checks are made to guarantee that they will succeed securely

*Attack9:* *Create Files with Same Name as Files Protected with a Higher Level of Classification*: an attacker can exploit an application that assigns special privileges to certain files, such as Dynamic Link Libraries, based on their location, resulting in an attack which takes advantage of execution or privilege decisions based on filename

*Countermeasures:*

- Ensure controls on privileged locations prevent writing or modifying to those locations by unauthorized applications
- Ensure that files are verified using more than filename and location alone

*Attack10:* *Force the Application to Display All Error Messages*; an attacker can use the information an application provides in error messages used to alert users of improper or disallowed actions, in order to discover a situation where no error message is displayed (meaning the error is not handled correctly) and the where the application attempts to process the bad value

*Countermeasures:*

- Use pre-tested code to ensure that all parameters are validated before they are used.
- Parameters should be validated against a positive specification that defines:

    - Data type (string, integer, and so on)
    - Allowed character set
    - Minimum and maximum length
    - Whether null is allowed
    - Whether the parameter is required or not
    - Whether duplicates are allowed
    - Numeric range
    - Specific legal ranges (enumeration)
    - Specific patterns (regular expressions)

*Attack11:* *Look for Temporary Files and Screen the File Contents for Protected Information*; an attacker can exploit applications routinely write information to temporary files, in order to gain insecure access to that information

*Countermeasures:*

- Ensure that the mechanisms for storing this information are secure
- Ensure that the mechanisms for accessing this information are secure
- Understand when, where, how the application accesses file-system information
- Identify which information should not be exposed to other potential users of the system
- Find creative ways to gain insecure access to the protected information

*Attack12:* *Passing Credentials;* an attacker can exploit web service messages (such as XML) which convert the credentials to text format prior to being sent, resulting in the attacker gaining access to a clear text version of the credentials

*Countermeasures:*

- Encrypt all protected information such as passwords and private keys

*Attack13:* *Broken Authentication and Session Management*; an attacker can make use of a session token that is not properly protected to hijack a session and assume the identity of the user

*Countermeasures:*

- Use a credential management scheme which consistently enforces the security standard, paying special attention to:

    - Password strength (minimum size and complexity)
    - Password use (defined number of allowable loin attempts per unit time)
    - Password change controls (uniformly use the same mechanism to change the password)
    - Password storage (should be stored in hashed or encrypted form for protection)
    - Protecting credentials in transit (encrypt the entire login transaction with a secure protocol as such as SSL)
    - Session ID protection (encrypt the entire user session with a secure protocol as such as SSL)
    - Account lists (avoid allowing users to gain access to a list of account names on site; if necessary display a pseudonym list that maps to the real list instead)

- Browser caching (authentication pages should be marked with a no cache tag to prevent someone from using the back button in a user's browser to access the login page and resubmit the credentials)
- Trust relationships (avoid implicit trust between components whenever possible; each component should have to authenticate itself to the other component)

*Attack14:* *Cross site scripting (XSS) flaw*; an attacker can cause a web application to send malicious code (generally in the form of a script) to be executed through a victim's browser

*Countermeasures:*

- ❑ Input filtering: properly sanitize user input information by validating all headers, cookies, query strings, form fields and hidden fields
- ❑ Output filtering: filter and properly sanitize user information when it is sent back to the user's browser
- ❑ Use of firewall: use third party application firewall which intercepts and blocks cross site script before it reaches the web server or vulnerable scripts
- ❑ Disable client side scripting: The best protection is to disable scripting when it is not required
- ❑ Use signed scripting: use signed scripting such that any script with an invalid or untrusted signature will not be run automatically

*Attack15:* *Buffer overflows;* an attacker can send crafted input to a web application, causing it to execute arbitrary code which corrupts the execution stack, allowing the attacker to take over the system

*Countermeasures:*

- ❑ Apply the latest security patches to the web application
- ❑ Periodically scan the web code looking for buffer overflow flaws in the web server or application
- ❑ Properly sanitize user input information by validating all headers, cookies, query strings, form fields and hidden fields

*Attack16:* *Injection flaws:* an attacker can relay malicious code through one web application to another

*Countermeasures:*

- ❑ Avoid using external operating system shell commands to pass function calls, relying instead on internal language specific libraries to do the same function
- ❑ For calls to backend databases, carefully validate the information provided to ensure it does not contain any malicious content

*Attack17:* *Insecure storage:* an attacker can take advantage of an application's need to store protected information by locating insecurely stored information

*Countermeasures:*

- Encrypt all critical information
- Encrypt all keys, certificates and passwords
- Encrypt all secrets in memory
- Choose strong algorithms
- Use proven encryption algorithms
- Provide supporting mechanisms for encryption key changes, and so on
- Whenever reasonable, rather than store protected information in an encrypted form, force the user to re-enter the information

*Attack18:* *Denial of service;* an attacker can use a web application's inability to tell the difference between valid traffic and traffic generated for an attack, to force the web application to attempt to handle excessive numbers of concurrent users or traffic volumes, causing the web application to cease functioning in a normal manner

*Countermeasures:*

- Establish quotas to limit the amount of load a given user can generate
- Handle one request per user at a time by synchronizing on the user's session
- Drop any requests currently being processed for a user when another request from that user arrives
- Check the error handling scheme to ensure an error cannot affect the overall operation of the application

*Attack19:* *Insecure configuration management:* an attacker can use improper system configuration to exploit the web application

*Countermeasures:*

- Patch all security flaws in the server software
- Configure the application software to limit directory listing or directory traversal
- Remove unnecessary default, backup or sample files; including scripts, applications, configuration files and web pages
- Correctly configure file and directory permissions
- Correctly configure user, group and role permissions
- Disable unnecessary services, including content management and remote administration
- Change default passwords on default accounts
- Disable unnecessary administrative or debugging functionality
- Correctly configure SSL certificates and encryption settings
- Use signed certificates for authentication

- Ensure proper authentication with external systems

*Attack20:* *Identifying the web server vendor and version by banner grabbing;* an attacker may use the disclosure of unnecessary information in the web server banner to attempt to gain access to the web server

*Countermeasures:*
- If possible, change the server tag in response header.

*Attack21:* *Identifying the web server vendor and version by using default files;* an attacker may use the normal behavior of the server to expose default directories, file extensions, and pages in the default installation

*Countermeasures:*
- Set permissions to prevent access on default pages of the server.

*Attack22:* *Identifying the web server vendor and version by identifying the modules running on the web server;* An attacker may use the response header to identify the modules running, which in turn will identify the operating system and which modules can be exploited

*Countermeasures:*
- Change the server tag

*Attack23:* *Product specific issues;* an attacker can use knowledge of the modules running on the web server to get access to the remote machine

*Countermeasures:*
- Patch the web server and web applications regularly