Appendix: Remediation Advice and Resources

Command Injection (CMDi)

Command Injection attacks can be ameliorated through input validation, whitelisting of allowed characters and the use of APIs versus direct system calls. Limiting the privileges of the application user is also useful since the injected commands run with the same environment as the application user.

Cross-Site Scripting (XSS)

The primary defense against XSS is contextually appropriate escaping of user input. The extent and approach to this will depend on whether you need to accept actual code from users and of what sort. Since there are several potential pitfalls in implementing this, the use of a security-focused encoding library is wise.

An XSS attack could also read a user's cookie and then use that information on the originating web site. If your application uses cookies, you can invalidate the session if the user's IP changes. For an extensive list of remediation steps, see the XSS Prevention Cheat Sheet.

Cross-Site Request Forgery (CSRF)

CSRF is essentially an issue of authentication and user intent: actions are taken by an attacker on behalf of an authorized user without their knowledge. To protect against it, web applications need strong authentication measures and a reliable way to judge the user intended to take a particular action. The latter is commonly missed, but without it otherwise secure authentication methods can still be abused.

While there are a large number of defenses against CSRF with varying effectiveness, one common approach is to add a secret random token valid for the current user or even per request, also known as the Synchronizer Token pattern. This token is then validated by the server to ensure that the request was from the proper user and that the user intended to make the request (since the value is inserted into forms before submission). There are also a number of libraries that can be plugged into applications which can make CSRF protection much easier to implement.

It's important to note that the effect of a CSRF attack is entirely dependent on the application privileges of the user being attacked. Limit application permissions to the minimum necessary to reduce your risk profile. For an extensive list of remediation steps, see the CSRF Prevention Cheat Sheet.

Insecure Direct Object Reference (DOR)

By directly referencing internal objects such as filenames or database record IDs in URLs, a web application becomes susceptible to an attacker altering those references and attempting to take actions not otherwise allowed or intended. The result can be varied and potentially disastrous, such as a user viewing sensitive files on the application server to removing another user's account.

The main defenses against DOR are mapping of objects versus direct reference as well as proper authentication of all sensitive requests. By mapping selections made on a per user or per session basis to their actual values on the server side, no database details are revealed to the end user.

Redirects and files in URL parameters deserve special attention. Redirects must be implemented with a white list in order to prevent users from being redirected to a malicious site in phishing attacks. A similar approach is needed for serving files with their path in the URL. If the file requested isn't verified the user could guess the relative path to a sensitive file from the web root and modify the URL to serve it. For any resource that is directly referenced, the main defense is verifying the requesting user and that they have permission to access the resource.

Local and Remote File Includes (LFI, RFI)

RFI attacks are caused by unvalidated user input interacting with functions that load resource files. By manipulating the values of variables in a page's URL for example, external files hosted by the attacker can be dynamically rendered, resulting in malicious code being run. To protect against this attack you must prevent unvalidated user input from being used in file include requests. For more information and examples, see this article: http://www.esecurityplanet.com/browser-security/how-to-prevent-remote-file-inclusion-rfi-attacks.html

LFI attacks are allowed by the same lack of input validation in resource loading requests as RFIs, but the result is that local files (i.e. on the target webserver) can be viewed and exfiltrated. Never allow unchecked user input to be directly used in resource calls. Limiting your web server application's permissions to only those needed will prevent access to sensitive files not needed by the web application.

Publicly Writeable Directory

Using publicly or world-writeable directories in your web application can be dangerous because the files they contain can be changed by a user with any access level on the host machine. Since some web applications use such directories to store, for example, session information this can open a number of security holes. To avoid this, ensure your application only uses specific subdirectories with the needed permissions, and that they are otherwise locked down. Avoid publicly-writeable directories altogether if possible.

Session Fixation

Session fixation is made possible through poor web application authentication. If the server does not provide a new session ID when authentication occurs it is possible for the original ID to serve as a valid authenticated session ID. An attacker can set this original ID to a value she knows and place it in a URL parameter, cookie or hidden form field that is then presented to the victim. Thus the best way to avoid this vulnerability is to always invalidate the existing session ID and create a new one whenever authentication occurs, which should be required before any sensitive or important request.

It is important to protect against XSS in this same context since session fixation is rarely feasible if the attacker cannot inject malicious code that sets the desired session ID in user's cookie.

SQL Injection (SQLi)

As with other common vulnerabilities, user input validation is key to preventing SQLi. Directly applying user input to raw SQL queries should never be allowed. This can be accomplished through a number of methods such as binding user input to parameters that are passed into DB statements, using ORM libraries and escaping characters that have special meaning to SQL.

In addition, audit the permissions of your web application's DB user to ensure it does not have more access than is needed. This will help reduce the impact if an injection attack does make it through. For an extensive list of remediation steps, see the SQL Injection Prevention Cheat Sheet.

Insufficient Transport Layer Security

The transport layer of a web application needs to use SSL for all sensitive requests. In addition, the protocol, cipher type and key length of the SSL implementation selected must not have known weaknesses. Ensure that all sensitive actions occur over a connection established with an acceptable implementation of SSL. Your SSL certificate should also be valid and current to avoid training users into thinking certificate related warnings and errors are normal. Without such sufficient protections in place an attacker could compromise sensitive transferred data, potentially leading to compromise of the entire application aside from data exfiltration. For more information on see the OWASP Transport Layer Protection Cheat Sheet.

Unauthorized Access

The issue of what users should be allowed to perform what functions in a web application is both a simple and a complex issue. While the notion of access control is straighforward in concept, the manifestations of it in a real world application can be sprawling, difficult to understand and potentially vulnerable. If a single function called from a URL is not properly validated for the user making the request, sensitive data could be exposed, records could be deleted, etc.

The solution is also simple in notion and complex in manifestation: select an effective authentication mechanism and ensure that all pages in your application are properly validated with that mechanism. If a page should not be accessible to every user, there must be a verification that only allowed users can access it, that they are authenticated and made the request. Your role-based access control scheme should be kept as simple and configurable as possible to make such audits possible. Additional information on many aspects of this topic can be found in the OWASP Guide to Authorization.

Unvalidated Redirect

If your web application doesn't verify the destinations of redirects (AKA forwards, transfers) then it can be used as a vector to send users to malicious sites. A user is much more likely to click a link in a phishing attack if the site linked to is one she knows and trusts. Improper configuration of the application can allow an attacker to use an altered URL to redirect a user from the known site to one of the attacker's choosing. This site might be a copy of a valid site designed to steal credentials, install malware on the user's machine, etc.

Audit your application for any redirects that employ a URL parameter to specify all or part of the destination path. When you need to employ redirects, try to avoid storing the path in a parameter. If you must do this, verify that the destination is allowed via a white list and that the user making the request is authorized to access it. You might also consider mapping the needed path to a value that is translated by the server as well as presenting a confirmation page to the user when a redirect occurs.

Vulnerable Version

Avoiding the use of components with known vulnerabilities in your application is an ongoing process of awareness and auditing. Applications almost always employ a number of external components to save development time and follow accepted practices. These components, as well as the sometimes non-trivial number of dependencies they themselves have, must be audited carefully to avoid the possibility of a component being compromised, which could result in the entire application being compromised. Keeping a record of all known components and their current version is a good start to this process. Making it an explicit responsibility to regularly monitor the providers of these components and decide if an upgrade is warranted in the face of an update is crucial, as is monitoring vulnerability disclosure databases such as nvd.nist.gov and cve.mitre.org.

Appendix: Glossary

Command Injection (CMDi)

An attack method that injects and runs commands in a compromised application. Most command shell attacks take advantage of unvalidated user input, which enables an attacker to append a command sequence or escape string to execute arbitrary commands. If an command inject attack is successful, the attacker can use the exploited application as a pseudo shell to execute malicious code.

Cross-Site Scripting (XSS)

An attack method that injects a client-side script to exploit web applications and web pages. Most cross-site scripting attacks execute malicious code when a target visits an infected web page. Typically, the infected web page redirects the victim to a spoofed web page, which injects malicious code that enables the attacker to take over the session.

Direct Object Reference (DOR)

An attack method that exploits a vulnerability in web applications that exposes an internal implementation object, such as a database record or file, to a user. Examples of common examples of direct object vulnerabilities include open redirects and direct traversals.

Local File Inclusion (LFI)

An attack method that uses an include method to inject local files from the exploited server. LFI attacks typically exploit vulnerable parameters that enable an attacker to include code that is already hosted on the server. LFI attacks enable the attacker to gather user names, gather information from the log files, and remotely execute commands on the exploited server.

Cross-site Request Forgery (CSRF)

An attack method that uses a victim's active session to exploit the victim's identity and privileges. During a Cross-site Request Forgery attack, an attacker sends a victim a URL to a web page that contains a link, form button, or some Javascript that performs an action. When the action is requested, it executes malicious actions on the attacker's behalf. For example, an attacker may force a victim to unknowingly perform tasks like make purchases, modify account information, transfer funds, or pretty much any action that the web application allows.

Publicly Writable Directory

A directory that has write permissions that grants all users the ability to modify the directory and the files that it contains, including creating, deleting, and renaming files.

Remote File Inclusion (RFI)

An attack method that uses server-side scripts to exploit vulnerable web applications and enables the attacker to upload a remote file to the victim's server. RFI attacks typically exploit vulnerable include functions to link to remote scripts. These scripts will commonly allow an attacker to execute shell commands that enable them to upload files, create directories, and modify websites.

Session Fixation

An attack method that enables an attacker to hijack an established user session by forcing the session identifier (ID) to a specific value. During a session fixation attack, the attacker sends a victim a URL that contains the fixed session, which forces the victims's browser to use the selected session.When the victim clicks on the URL, the web application establishes that a session already exists for the user and does not create a new session. Therefore, when the victim logs into web application, the attacker is able to access the account using the same session ID.

Sites

Refers to a website, or a collection of web pages, that is defined by a fully qualified domain name or IP address. A site can also refer to a web application.

SQL Injection (SQLi)

An attack method that exploits user input vulnerabilities to pass SQL queries through a web application to the database.The database executes the SQL queries, which typically enables the attacker to modify or gain access to the database.

Transport Layer Encryption

An Internet protocol that enables the ability to securely transmit and receive encrypted data between servers and clients that support Transportation Layer Security or Secure Socket Layer (SSL). TLS/SSL prevents non-trusted devices from allowing a third party to monitor or alter communication between a server and a client.

Unauthorized Access

Refers to the ability to obtain entry to system and network resources without valid permissions. An attacker can exploit vulnerabilities in authentication services, FTP services, and web services to obtain unauthorized access in order to do things like modify security policies, steal user names and passwords, and escalate privileges.

Unvalidated Redirect

A request that accepts untrusted and unvalidated user-supplied parameters to specify the redirection of the target. If the application does not validate the input value, the victim can be redirected to a malicious URL. This attack method is typically used in phishing attacks to get victims to unknowingly visit a malicious site. To exploit an unvalidated redirect, an attacker may craft a URL that uses a domain of a trusted site, such as http://www.yoursite.com. However, the URL may include a redirect function, such as http://www.yoursite.com/redirect.aspx?url=http://www.mysite.com, that sends the victim to a malicious site designated by the attacker.

Vulnerable Version

Refers to a version of an application or software that has known security vulnerabilities.

vhost (Virtual Host)

Refers to the fully qualified domain name of a virtual host or server. Typically, vhosts are devices that can be accessed remotely by users to host data or utilize software services.

Web Audit

A feature that performs vulnerability checks for XSS, LFI, RFI, and SQLi flaws.

Web Exploit

A feature that matches exploits to known web vulnerabilities to create an attack plan. Web exploit runs the attack plan after it has been created and attempts to exploit the identified vulnerabilities.

Web Crawl

A feature that recursively parses a website or namespace for hyperlinks that point to other web pages and follows the links to those other pages.

Web Page

Refers to an HTML document that resides on the World Wide Web.

Web Scan

A feature that analyzes web application configurations and security. A web scan crawls websites, audits them for misconfigurations and common vulnerability types, such as XSS, LFI, RFI, and SQLi vulnerabilities, and exploits the identified vulnerabilities.

Web Vulnerability

A security weakness or flaw that enables an attacker to compromise a web application. Most web vulnerabilities are caused by unvalidated user input, cross-site scripting, and injection flaws. These security flaws provide attackers with an opportunity to leverage the vulnerability to gain access to the web application to run arbitrary code.