Web Application Security

What is web application security?

Web application security (also known as Web AppSec) is the idea of building websites to function as expected, even when they are under attack.

The concept involves a collection of security controls engineered into a Web application to protect its assets from potentially malicious agents.

Web applications, like all software, inevitably contain defects. Some of these defects constitute actual vulnerabilities that can be exploited, introducing risks to organizations.

Web application security defends against such defects.

It involves leveraging secure development practices and implementing security measures throughout the software development life cycle (SDLC), ensuring that design-level flaws and implementation-level bugs are addressed.

Why is web security testing important?

Web security testing aims to find security vulnerabilities in Web applications and their configuration.

The primary target is the application layer (i.e., what is running on the HTTP protocol).

Testing the security of a Web application often involves sending different types of input to provoke errors and make the system behave in unexpected ways. These so called "negative tests" examine whether the system is doing something it isn't designed to do.

It is also important to understand that Web security testing is not only about testing the security features (e.g., authentication and authorization) that may be implemented in the application.

It is equally important to test that other features are implemented in a secure way (e.g., business logic and the use of proper input validation and output encoding).

The goal is to ensure that the functions exposed in the Web application are secure.

What are the different types of security tests?

Dynamic Application Security Test (DAST)

This automated application security test is best for internally facing, low-risk applications that must comply with regulatory security assessments.

For medium-risk applications and critical applications undergoing minor changes, combining DAST with some manual web security testing for common vulnerabilities is the best solution.

Static Application Security Test (SAST)

This application security approach offers automated and manual testing techniques.

It is best for identifying bugs without the need to execute applications in a production environment.

It also enables developers to scan source code and systematically find and eliminate software security vulnerabilities.

Penetration Test

This manual application security test is best for critical applications, especially those undergoing major changes.

The assessment involves business logic and adversary-based testing to discover advanced attack scenarios.

Runtime Application Self Protection (RASP)

This evolving application security approach encompasses a number of technological techniques to instrument an application so that attacks can be monitored as they execute and, ideally, blocked in real time.

How does application security testing reduce your organization's risk?

Majority of Web Application Attacks

- SQL Injection
- XSS (Cross Site Scripting)
- Remote Command Execution
- Path Traversal

Attack Results

- Access to restricted content
- Compromised user accounts
- Installation of malicious code
- Lost sales revenue
- Loss of trust with customers
- Damaged brand reputation
- And much more

A Web application in today's environment can be affected by a wide range of issues.

The diagram above demonstrates several of the top attacks used by attackers, which can result in serious damage to an individual application or the overall organization.

Knowing the different attacks that make an application vulnerable, in addition to the potential outcomes of an attack, allow your firm to preemptively address the vulnerabilities and accurately test for them.

By identifying the root cause of the vulnerabilities, mitigating controls can be implemented during the early stages of the SDLC to prevent any issues.

Additionally, knowledge of how these attacks work can be leveraged to target known points of interest during a Web application security test.

Recognizing the impact of an attack is also key to managing your firm's risk, as the effects of a successful attack can be used to gauge the vulnerability's total severity.

If issues are identified during a security test, defining their severity allows your firm to efficiently prioritize the remediation efforts.

Start with critical severity issues and work towards lower impact issues to minimize risk to your firm.

Prior to an issue being identified, evaluating the potential impact against each application within your firm's application library can facilitate the prioritization of application security testing.

With an established list of high profile applications, web security testing can be scheduled to target your firm's critical applications first with more targeted testing to lower the risk against the business.

What features should be reviewed during a web application security test?

The following non-exhaustive list of features should be reviewed during Web application security testing.

An inappropriate implementation of each could result invulnerabilities, creating serious risk for your organization.

Application and server configuration

Potential defects are related to encryption/cryptographic configurations, Web server configurations, etc.

Input validation and error handling

SQL injection, cross-site scripting (XSS), and other common injection vulnerabilities are the result of poor input and output handling.

Authentication and session management

Vulnerabilities potentially resulting in user impersonation. Credential strength and protection should also be considered.

Authorization

Testing the ability of the application to protect against vertical and horizontal privilege escalations.

Business logic

These are important to most applications that provide business functionality.

Client-side logic

With modern, JavaScript-heavy web pages, in addition to web pages using other types of client-side technologies (e.g., Silverlight, Flash, Javaapplets), this type of feature is becoming more prevalent.

Web Application Security Threats

Each year, attackers develop inventive web application security threats to compromise sensitive data and access their targets' database.

Consequently, security experts build on the exploited vulnerabilities and strengthen their systems through their learning's every year.

1. Injection Attacks

A web app that is vulnerable to injection attacks accepts untrusted data from an input field without any proper sanitation.

By typing code into an input field, the attacker can trick the server into interpreting it as a system command and thereby act as the attacker intended.

Some common injection attacks include SQL injections, Cross-Site Scripting, Email Header Injection, etc.

These attacks could lead to unauthorized access to databases and exploitation of admin privileges.

How to prevent:

- Keep untrusted inputs away from commands and queries.
- Use a safe Application Programming Interface (API) that avoids interpreters or uses parameterized interfaces.
- Filter and sanitize all inputs as per a white list. This prevents the use of malicious character combinations.

2. Broken Authentication

Broken authentication is an umbrella term given to vulnerabilities wherein authentication and session management tokens are inadequately implemented.

This improper implementation allows hackers to make claims over a legitimate user's identity, access their sensitive data, and potentially exploit the designated ID privileges**.**

How to prevent:

- End sessions after a certain period of inactivity.
- Invalidate a session ID as soon as the session ends.
- Place limiters on the simplicity of passwords.
- Implement multi-factor authentication (2FA/MFA).

3. Cross Site Scripting (XSS)

It is an injection-based client-side attack. At its core, this attack involves injecting malicious code in a website application to execute them in the victims' browsers eventually.

Any application that doesn't validate untrusted data adequately is vulnerable to such attacks.

Successful implementation results in theft of user session IDs, website defacing, and redirection to malicious sites (thereby allowing phishing attacks).

How to prevent:

- Encode all user-supplied data.
- Use auto-sanitization libraries such as OWASP's AntiSamy.
- White list inputs to disallow certain special character combinations.

4. Insecure Direct Object References (IDOR)

Mostly through manipulation of the URL, an attacker gains access to database items belonging to other users.

For instance, the reference to a database object is exposed in the URL.

The vulnerability exists when someone can edit the URL to access other similar critical information (such as monthly salary slips) without additional authorization.

How to prevent:

- Implement proper user authorization checks at relevant stages of users' web app journey.
- Customize error messages so that they don't reveal critical information about the respective user.
- Try not to disclose reference to objects in the URL; use POST based information transmission over GET.

5. Security Misconfigurations

According to OWASP top 10 2017, this is the most common web application security threats found across web applications.

This vulnerability exists because developers and administrators "forget" to change some default settings such as default passwords, usernames, reference IDs, error messages, etc.

Given how easy it is to detect and exploit default settings that were initially placed to accommodate a simple user experience, the implications of such a vulnerability can be vast once the website is live: from admin privileges to complete database access.

How to prevent:

- Frequently maintain and update all web application components: firewalls, operating systems, servers, databases, extensions, etc.
- Make sure to change default configurations.
- Make time for regular penetration tests (though this applies to every vulnerability that a web app could have).

6. Unvalidated Redirects and Forwards

Pretty much every website redirects a user to other web pages. When the credibility of this redirection is not assessed, the website leaves itself vulnerable to such URL based attacks.

A malicious actor can redirect users to phishing sites or sites containing malware.

Phishers search for this vulnerability extensively since it makes it easier for them to gain user trust.

How to prevent:

- Avoid redirection where possible.
- Give the destination parameters a mapping value rather than the actual URL. Let the server-side code translate the mapping value to the actual URL.

7. Missing Function Level Access Control

The seventh web application security threats in this list is mostly similar to IDOR.

The core differentiating factor between the two is that IDOR tends to give the attacker access to information in the database.

In contrast, Missing_ Function Level Access Control _allows the attacker access to special functions and features that should not be available to any typical user.

Like, IDOR, access to these functions can be gained through URL manipulation as well.

How to prevent:

- Implement adequate authorization measures at relevant stages of user web app use.
- Deny all access to set features and functions unless attempted by a pre-approved (admin) user.
- Allow for a flexible shift in grant and rejection of access to feature privileges in your code. Hence, allowing a practical and secure shift in privilege access when needed.

Client Side Security

Today's web applications are complex, oft en made up of a mix of existing soft ware, open-source and third-party code, and custom JavaScript and HTML all integrated via application program interfaces (APIs).

While web applications are hosted and maintained on an organization's server, they actually run on an end user's browser.

The scripts that run the applications are referred to as 'client-side scripts.' These scripts create an incredibly dynamic environment that enable a high level of functionality, but also facilitate tremendous risk since the combination of potentially flawed or vulnerable systems, servers, codes, and applications creates the perfect scenario for threat actors to leverage in client-side attacks

What are client-side attacks?

Client-side attacks occur when a user unintentionally downloads malicious or vulnerable content from a server, oft en by doing nothing more than simply clicking on a web page and filling out a form.

That content could take the form of bad JavaScript code or unsafe third-party code that exists as part of the web application.

The term 'client-side' refers to end-user devices, like desktops, laptops, mobile phones, and tablets, which are considered 'clients.'

Conversely, the systems that the devices are connected to are referred to as 'servers.'

Client devices send requests to the server and the server responds to the request.

Servers usually support multiple client devices at the same time, and client devices usually send requests to multiple different servers while operating on the internet.

Because client-side activity happens outside a business's security perimeter, standard security technologies won't protect the end user from malicious activity that is occurring on dynamic web pages accessed from the end user's own device.

What are the most common client-side security risks?

Unmitigated risks present in organizational systems can lead to potentially severe attacks on the client side—that is, an organization's customers or end users. These types of attacks include e-skimming, Magecart-like threats, and form jacking.

The Open Web Application Security Project® (OWASP) lists 12 client-side security risks that organizations need to ensure they've mitigated to prevent attacks:

Document Object Model (DOM)-based Cross-site Scripting— Sometimes also called just 'cross-site scripting' or 'XSS', this is a vulnerability that affects websites and enables an attacker to inject their own malicious code onto the HTML pages displayed to users.

If the malicious code is executed by the victim's browser, the code performs actions, such as stealing credit card information or sensitive credentials.

JavaScript Injection—

This type of vulnerability is considered a subtype of XSS involving the injection of malicious JavaScript code executed by the end user's browser application.

JavaScript injunctions can be used to modify the content seen by the end user, to steal the user's session cookies, or to impersonate the user.

Hypertext Markup Language (HTML) Injection—

Another type of cross-site scripting attack, an HTML injection involves injecting HTML code via vulnerable sections of the website.

Usually, the purpose of the HTML injection is to change the website's design or information displayed on the website.

Client-side URL Redirection or Open Redirection—

In this type of attack, an application accepts untrusted input that contains a URL value that causes the web application to redirect the user to another, likely malicious page controlled by the attacker.

Cascading Style Sheets (CSS) Injection—

Attackers inject arbitrary CSS code into a website, which is then rendered in the end user's browser.

Depending on the type of CSS payload, the attack could lead to cross-site scripting, user interface (UI) modifications or the exfiltration of sensitive information, like credit card data.

Client-side Resource Manipulation—

This type of vulnerability enables the threat actor to control the URL that links to other resources on the web page, thus enabling cross-site scripting attacks.

Cross-origin Resource Sharing (CORS)—

Poorly configured CORS policies can facilitate cross-origin attacks like cross-site request forgery (CSRF).

Cross-site Flashing—

Because Flash applications are often embedded in browsers, flaws or vulnerabilities in the Flash application could enable cross-site scripting attacks.

Clickjacking or UI Redress Attack—

This type of attack involves a threat actor using multiple web page frame layers to trick a user into clicking a button or link on a different page from the one intended.

Keystrokes can also be hijacked using this technique. By using style sheets, i frames, and text boxes, a threat actor can trick the user into thinking they're entering login credentials or bank account information into a legitimate website, when, in fact, they are actually typing into a frame controlled by the attacker.

WebSockets—

If servers do not properly verify the origin of an initial HTTP web socket server, a variety of different attack types are possible, including sniffing, cross-site web socket hijacking (CSWH), and cross-site request forgery (CSRF).

Web Messaging—

Also called cross-document messaging, web messaging enables applications running on different domains to communicate securely.

If the receiving domain is not configured, problems could arise related to redirection or the website leaking sensitive information to unknown or malicious servers.

Local Storage—

Sometimes called web storage or offline storage, local storage enables JavaScript sites and apps to store and access the data without any expiration date.

Thus, data stored in the browser will be available even after closing the browser window.

Since the storage can be read using JavaScript, a cross-site scripting attack could extract all the data from the storage.

Malicious data could also be loaded via JavaScript.

Countermeasures:

Steps to take to prevent client-side attacks.

Install antivirus software, anti-spyware software, and firewall protection on all workstations, servers, and wireless devices.

Ensure the latest system software patches are applied regularly.

Maintain a complete backup system of all data on all systems use a separate server or external hard drive or network location to store backups that are no longer needed and keep them off the system they were created on.

If a user is logging in from an unknown location or IP address, consider blocking access from those locations (access control lists).

Prevent unauthorized access to accounts.

Use strong passwords and avoid common passwords or patterns that can lead to vulnerabilities, like “Admin” or “12345”.

Limit login attempts (user lockout).

Server-side Security

How to secure a server?

Server security is a major issue for companies. Indeed, being a central element in the functioning of all the components of an information system (applications, network, infrastructure, employees, etc.), servers are often the prime targets of attacks.

Furthermore, server-side vulnerabilities can have severe consequences.

In the event of misconfiguration or lack of control, these flaws can be exploited and lead to the compromise of the data in transit, or even to the server being taken over by malicious persons.

Why is server security important?

Attacks on servers are a daily occurrence because, very often, too many loopholes exist.

Indeed, applications vulnerable to SQL injections hosted on a server, users unaware of social engineering risks, or simply poor practices in terms of updates and patch management of the operating system and server services, easily allow attackers to achieve their goals: data theft, access to sensitive information, paralysis of a company's activity, etc.

Securing a server is therefore vital and necessarily involves implementing best practices in terms of configuration, control, monitoring and security testing.

Update and patch management policy, hardening, access and administration control and security (via SSH), best logging and monitoring practices, etc., we present here the main measures to strengthen the security of your servers.

Implement an update and patch management policy for servers

Implementing an update management policy for operating systems and services is essential to maintain a good level of security.

Indeed, new vulnerabilities are discovered and published regularly.

And if security patches are not applied in time, the risk of attacks and server compromise increases.

The numerous attacks suffered by companies, via malware, following the publication of a patch for a protocol, software, operating system, etc., must push any type of organization to adopt a proactive posture, especially as information on vulnerabilities and exploits is published and therefore accessible to internal and external attackers.

It is also true that caution is recommended when it comes to installing software updates, as testing is always necessary, if not essential.

However, it is generally unwise because it is riskier to delay this process, as a passive attitude very often leads to the compromise of information systems.

It is therefore important to define and implement an update and patch management policy or servers and all software and hardware components of the IS.

This involves documenting procedures and continuous monitoring of the various patch releases or new versions.

Disable or remove unnecessary services

All software and components installed on an operating system increase the attack surface and therefore the risk of compromise.

However, strengthening server security requires reducing the attack surface.

To do this, it is necessary to disable or even remove (as far as possible) all services, applications, network protocols, third-party components, etc. that are not essential to the operation of your server.

Removing or disabling unnecessary systems enhances the security of a server in several ways.

Firstly, they cannot be compromised, nor can they be used as attack vectors to alter the services that are essential for the server to operate.

Indeed, it should be kept in mind that each component added to a server increases the risk of compromising it.

Furthermore, the server can be configured to better meet the requirements of a particular service, while improving performance and the overall level of security.

Finally, reducing services means limiting the number of log entries, which makes it easier to monitor and detect unusual events.

Controlling and securing server access: Focus on SSH

Presentation of SSH: an essential tool to ensure proper management and secure administration of the server

SSH (Secure Shell) is both a communication protocol and a computer program allowing local and remote administration of a server.

Its most common implementation is OpenSSH, which can be found on many systems, including servers (Unix, Linux, Windows) as well as workstations and network equipment.

Indeed, OpenSSH is a suite of tools offering many features, including: a server (sshd), several clients – remote shell connection (ssh) / file transfer and download (scp and sftp), a key generation tool (ssh-keygen), a keychain service (ssh-agent and ssh-add), etc.

SSH currently exists in two versions: SSHv1 and SSHv2. SSHv1 contains vulnerabilities that have been fixed in the second version. Therefore, for enhanced security, only version 2 of the SSH protocol should be authorized.

The most widely used feature of SSH is remote administration, which consists of connecting to a remote machine and launching a shell session following authentication.

In this context, the advantage of SSH is its security. Another commonly used feature is the transfer and download of files, both from a client to a server and from a server to a client.

For this purpose, SSH offers two mechanisms: SCP and SFTP, which should always be preferred to older protocols such as RCP and FTP.

Public key / certificate authentication

Not ensuring the authenticity of a server can have several security impacts, including the inability to verify that you are communicating with the correct server, with consequent risks of spoofing and exposure to Man in The Middle attacks.

SSH relies on asymmetric encryption for authentication, and thus ensures the legitimacy of the server being contacted before access is granted.

Moreover, this control is done in several ways with OpenSSH.

Either by ensuring that the public key fingerprint obtained previously with ssh-keygen and presented by the server is the correct one, or by verifying the signature of the certificate presented by the server with a certification authority known to the client.

Security of authentication and user access control

Users of a system always have rights, no matter how small.

It is therefore important to protect their access via a secure authentication mechanism and to thwart brute force attacks as much as possible.

Firstly, each user must have his or her own account to ensure better traceability and the allocation of access rights must always follow the principle of least privilege.

Furthermore, access to a service should be restricted to users who have a justified need and are explicitly authorized.

Regarding users authorized to configure the operating system of servers, they should be limited to a small number of designated administrators.

Furthermore, it is strongly discouraged to use authentication mechanisms in which authentication information is transmitted in the clear over a network, as this information can be intercepted via Man In the Middle attacks and used by an attacker to impersonate an authorized user.

Finally, to ensure secure user authentication, the following measures should be implemented:

- Removing default accounts because the default configuration of the operating system often includes guest accounts, administrator accounts and accounts associated with services. The names and passwords of these accounts are known to attackers.
- Implement a proper password policy. All passwords should be complex and difficult to guess. Above all, they should be long (more than 15 characters). To achieve this, nothing is better than the implementation of a password manager.

Logging and monitoring of server events

Logging is a central aspect of a security strategy.

It is a control mechanism for monitoring the network, systems and servers; and most importantly, it ensures the traceability of all normal and suspicious events.

Indeed, log files can be used to track the activities of an attacker and thus detect unusual events or failed or successful intrusion attempts.

To facilitate the management and exploitation of logs, they should be centralized on a dedicated server.

This is even more important because, in the event of a machine being compromised, it is likely that the logs will be destroyed or altered by the attacker.

Centralizing, regularly backing up and duplicating logs will ensure that a copy is always kept. And given their importance, it is essential to restrict access to logs to authorized users only.

Securing APIs, websites and web applications hosted on a server

APIs, websites and web applications hosted on a server must also be secured.

These can be used as attack vectors to compromise the server if they are vulnerable to SQL injections for example.