Web Secirity

**1) How to Store and Secure Sensitive Data in Web Applications**

The 2020 Verizon Data Breach Investigations Report (DBIR) says that nearly half (45%) of the breaches featured hacking, and are tied to web application vulnerabilities.

It has more than doubled year over year, 22% of breaches from social attacks and malware attacks, 17% of breaches due to misconfigurations, 8% of unauthorized access and 4% of physical attacks.

To make the hosting and running of a web application possible, several web application components are needed. In a basic environment there should be at least a web server software (such as Apache or IIS), web server operating system (such as Windows, Linux, MacOS), database server (such as MySQL, MSSQL or PostgreSQL) and a network based service, such as FTP or SFTP.

For a secure web server, all of these components also need to be protected to make sure that sensitive data is secured properly. If security breaks at any point, the malicious attackers can gain access to the web application and retrieve data from the database or tamper it.

**Sensitive data in web applications**

Sensitive data can be any sort of information that needs to be protected from unauthorized access to safeguard the privacy or security of an individual or organisation. It can include any information pertaining to:

* Passwords
* Passphrases
* Encryption keys
* OAuth tokens
* Credit card numbers
* Personal contact information such as names, phone numbers, email addresses, user accounts, physical addresses, etc
* Demographic information such as gender, age, income, education, ethnicity
* In some states and countries: machine identifying information such as MAC, IP addresses, serial numbers, etc

Also it can be personally identifying information (PII) or high business impact (HBI) data. Sensitive data varies a lot from country to country and the way you have to store and secure sensitive data can also vary accordingly. Various compliance standards, such as the Payment Card Industry (PCI) compliance standard, require special measures to be taken, when collecting sensitive data to stay in compliance.

In today’s world of infrastructure security- network, host, and application-level, data security becomes more important. Data security, includes the security of:

* Data-in-transit
* Data-at-rest

And the right storage mechanisms should be chosen for storing these data. Storage mechanisms should save information more reliably, reduce bandwidth, and improve responsiveness.

**Data Model**

Data model is a subset of the implementation model which describes the logical and physical representation of persistent data in the system.

* **Structured**: Structured data conforms to a tabular format with relationships between the different rows and columns, typical of SQL DBMS, flexible and dynamic queries, where the full range of query types may not be known a priori. Example: IndexedDB.
* **Key/Value**: Key/Value datastores, and related NoSQL databases use an associative array as the fundamental data model where each key is associated with one and only one value in a collection. Examples: Cache API in the browser, Apache Cassandra on the server.
* **Byte Streams**: File systems and other hierarchically organized data, stores data as a variable length, string of bytes, leaving any form of internal organization to the application layer. Examples: file systems and cloud storage services.

**Persistence**

Storage methods for web applications can be evaluated according to scope over which data is made persistent.

* **Session Persistence**: Data is persisted only as long as there exists an active single web session or browser tab. Example: Session Storage API.
* **Device Persistence**: Data is persisted across sessions and browser tabs/windows on a particular device. Example: Cache API.
* **Global Persistence**: Data is retained across sessions and devices. It is the most robust form of data persistence. It can’t be stored on the device itself, so server-side storage is needed. Example: Google Cloud Storage.

**A) Client-side Data storage**

Client-side storage allows users to store different types of data on the client with users’ permission and then retrieve them whenever needed. This allows users to persist data for long-term storage, save sites or documents for offline usage, keep user-specific settings for the site, and more.

Data can be stored in different ways, such as session storage, local storage, cookies, webSQL, cache and indexedDB.

* **SessionStorage**

SessionStorage object is used to store data on a temporary basis and cleared when the page session ends. Since SessionStorage is tab specific, it is not accessible from web workers or service workers. It is limited to about 5 MB and can contain only strings. It may be useful for storing small amounts of session specific information, for example, IndexedDB key.

* **LocalStorage**

LocalStorage object is used to store data for the entire website on a permanent basis. LocalStorage is not accessible from web workers or service workers. It is limited to about 5MB and can contain only strings. LocalStorage should be avoided because it is synchronous and will block the main thread.

* **Cookies**

Cookies are sent with every HTTP request, so storing data in it will significantly increase the size of web requests. They are synchronous, and are not accessible from web workers. Like LocalStorage and SessionStorage, cookies are limited to only strings. Cookies have their uses, but not a good choice for storage.

* **WebSQL**

WebSQL Support has been removed from almost all major browsers. The W3C stopped maintaining the Web SQL spec in 2010, with no plans to further updates planned. WebSQL should not be used, and existing usage should be migrated to IndexedDB.

* **Cache**

Cache has been deprecated and support will be removed from browsers in the future. Application cache should not be used, and existing usage should be migrated to service workers and the Cache API.

* **IndexedDB**

Unlike most modern promise-based APIs, IndexedDB is event based. Promise wrappers like idb for IndexedDB hide some of the powerful features but more importantly, hide the complex machinery (e.g. transactions, schema versioning) that comes with the IndexedDB library. It is a low level API that requires significant setup before use, which can be particularly painful for storing simple data.

**B) Server-side Data storage**

Data storage is usually handled server-side. Data storage can occur on physical hard drives, disk drives, USB drives or virtually in the cloud. Files are backed up and easily available when systems ever crash beyond repair.

There are three broad types of data storage, including direct attached storage, network attached storage and storage area network.

* **Direct Attached Storage (DAS)**

DAS is a storage system where servers are directly connected to the storage device. In DAS, to access data by applications, block-level access protocol is used. Some of the common devices in this category include:

* Hard Drives
* Solid-State Drives (SSD)
* CD/DVD Drives
* Flash Drives
* **Network Attached Storage (NAS)**

Network-attached storage is a file-level computer data storage server and it is connected to a computer network. It offers dedicated file serving and sharing through the network. It increases performance, reliability with features like RAID and swappable drives designed for higher multi-drive workloads.

* **Storage Area Network (SAN)**

A storage area network is a dedicated and high-performance storage system. It transfers block-level data between servers and storage devices. SAN is usually used in data centers, enterprises or virtual computing environments.

**C) Data Storage Devices**

Computer storage devices are any type of hardware that stores data. It keeps and retains information short-term or long-term. It can be a device inside or outside a computer or a server.

* **Hard Disk Drive**

Hard Disk Drive (HDD) or Fixed Disk Drive (FDD), is a non-volatile, hardware data storage device attached to a computer or server. It magnetically stores, retrieves, and outputs digital data using a series of stacked rotating metallic disks that have been coated with magnetic material. The rotating disks are paired with an actuator arm which reads and writes the digital data to the disks.

* **Solid State Drive**

Solid State Device (SSD) is a storage device that uses integrated circuit assemblies to store and retrieve data, typically using flash memory, and functioning as secondary storage in the hierarchy of computer storage. It offers swift data transfer between SSD and a smaller physical size than a disk array.

* **Compact Disk /Digital Versatile Disk**

An optical disc drive reads and writes all common Compact Disk (CD) and Digital Versatile Disk (DVD) formats. CD drives are built into computers. A DVD will hold more information than a CD, and therefore can be used for a wide variety of media and storage.

* **Hybrid Flash Arrays**

These storage devices include both flash memory drives and hard disk drives for balanced performance. Hybrid flash arrays use form factors and electrical interfaces that are compatible with common HDD bays. Hybrid flash arrays offer low-cost startup, reasonable performance costs and fast data access on demand.

* **Hybrid Cloud Storage**

Hybrid cloud storage is an approach for managing cloud storage that uses both local and off-site resources. It offers a secure and compliant option that helps to assure business continuity. It accommodates frequent backups and long-term archives as well as future scaling and always-on availability. The combination of cloud and on-premises storage adds a layer of safety to ensure data is protected and available, and storage space could potentially be unlimited.

* **Backup Software**

Computer programs used to perform a backup; creates additional exact copies of files, databases or entire computers. Software for system and enterprise backups typically comes with a license or a subscription rate billed monthly or annually.

* **Backup Appliances**

Accumulates the backup software and hardware components within a single device. Configurations may be complicated and reliability may be at risk with misconfigurations and incorrect software tuning.

* **Cloud Storage**

Complete cloud-based or online storage solutions offer virtual data storage which stores data on the internet through a cloud computing provider. They manage it and are responsible for data availability and accessibility, not just on a local computer or external hard disk. Reliability tends to be on point, but organizations need to consider a cloud storage security strategy before implementing.

**Web application vulnerabilities that lead to sensitive data leakage**

OWASP Top 10 is the list of the 10 most common application vulnerabilities with its risks, impact, countermeasures and it is updated every three to four years. The latest OWASP vulnerabilities list was released in 2017, they are:

* Injection
* Broken Authentication
* Sensitive Data Exposure
* XML External Entities (XXE)
* Broken Access Control
* Security Misconfigurations
* Cross Site Scripting (XSS)
* Insecure Deserialization
* Using Components with Known Vulnerabilities
* Insufficient Logging and Monitoring

**Sensitive data protective measures and mechanisms**

* **Authentication**

User authentication plays an important role in addressing many important data protection principles, as it is essential to meeting security, access, consent, and accountability requirements.

Maintaining confidentiality, integrity, and availability for data security is a basic factor in securing data. Authentication of users and even of communicating systems is performed by various mechanisms, but the basic factor of these is cryptography.

Authentication of users takes several forms, but all are based on the combination of authentication factors: something an individual knows (such as a password), something they possess (such as a security token), or some measurable quality (such as a fingerprint).

Single factor authentication is based on only one authentication factor. Stronger authentication requires additional factors; for instance, two factor authentication is based on two authentication factors (such as a pin and a fingerprint).

* **Access control**

Access controls are generally described as discretionary or non-discretionary, and the most common access control models are:

Discretionary Access Control (DAC) is a type of access control system that assigns access rights based on rules specified by users. Permission management can be difficult to maintain; DAC does not scale well beyond a small set of users.

Role Based Access Control (RBAC), also known as a non-discretionary access control, assigns rights based on organizational roles instead of individual user accounts within an organization and the access policy is determined by the system. A subject can access an object or execute a function, only if the set of permissions or role allows it.

Mandatory Access Control (MAC) uses a hierarchical approach to control access to files/resources. A subject’s label specifies the level of trust, and an object’s label specifies the level of trust that is required for accessing it. If a subject is to gain access to an object, the subject label must dominate or at least it should be as high as the object label. Access policy is determined by the system.

* **Encryption**

There are multiple ways for encrypting data at rest. Following is an outline of various forms of encryption that are protection methods for securing data at rest:

Full disk encryption of data at the disk level - cryptographic method that applies encryption to the entire hard drive including data, files, the operating system and software programs. This is a brute-force approach to encrypt data but this also involves performance and reliability concerns. If encryption is not done at the drive hardware level, then it affects the system in terms of performance and even minor disk corruption can be fatal as the OS, applications, and data.

Directory level (or Filesystem) - Entire data directories are encrypted or decrypted as a container and to access those files encryption keys are required. Used for segregating data of identical sensitivity or categorization into directories that are individually encrypted with different keys.

File level - Only specific files with sensitive data are encrypted rather than encrypting an entire hard drive or even a directory. It can be more efficient to encrypt individual files.

Application level - Allows to encrypt entire files or specific fields of data at the application level, before it is stored. The application manages encryption and decryption of application-managed data.

The two goals of securing data in motion are preventing data from being compromised with its confidentiality, integrity, availability. To protect data in motion:

* Implement security framework for data by enforcing end-to-end encryption, strong authentication, automation of file based tasks, rules and policy management, user Ad Hoc secure file transfers, guaranteed delivery, integration with existing security controls, etc
* Restrict cloud sharing/alternative transfer methods
* Identify critical assets and vulnerabilities

The most common way to protect data in motion is to utilize encryption combined with authentication to create a conduit to safely pass data.

**2 Session Security**

Session security plays a key factor in building secure web applications. A web application is not secure unless it is protected from external attacks like XSS. These malicious scripts are designed to gain access to sensitive data in web applications, including cookies, as they act as a key to store session tokens.

Attackers can exploit and gain unauthorized access to the web application because of the improper implementation of authorisation or authentication. According to OWASP (Open Web Application Security Project) Top 10, broken authentication is the second biggest risk to web application security.

**What is a session?**

HTTP is a stateless protocol, enabling the communication between a client (front-end) and a server (back-end). Sessions or tokens are used to overcome the stateless nature of HTTP requests.

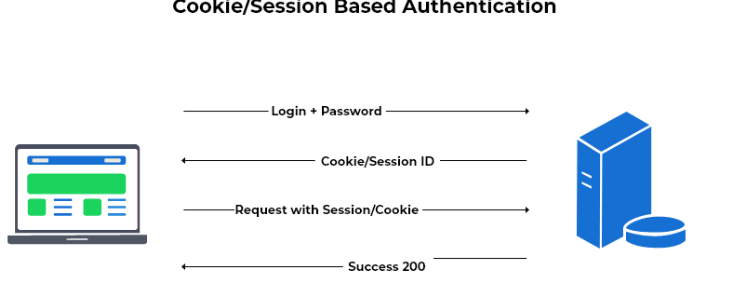
Somehow if cyber attackers gain control over cookies, they can impersonate the user, thereby retrieving their sensitive data. Over the years, web application security began with sessions and now it is based on tokens to improve overall session security.

**Types of authentication**

**Session based authentication**

A session identifier (session ID) is created at server-side to uniquely identify each user login. This session ID is sent to the browser. The session ID is stored in a cookie in the client-side (browser). While the user is logged in, the cookie will be sent along with each subsequent request. For example,

Cookie: Session\_Id=bhbsbdljfbsjkd9784a49hjihfgkdh4iuhuihnh43i65743;



**How session based authentication works**

* User tries to log in using their login credentials.
* The server verifies the user with the given credentials and creates a session with a session ID.
* The session ID is placed in the client (browser) as a cookie.
* On subsequent requests, the session ID stored in the client-side is verified against the session ID stored in the server-side, and if a match is found it is considered as valid, then the request is processed.
* And if a user logs out of the application, then the session is destroyed in both client-side and server-side.

**Advantages of session based authentication**

Cookies are small-sized values, easy to use and implement and can revoke the validity of the cookies.

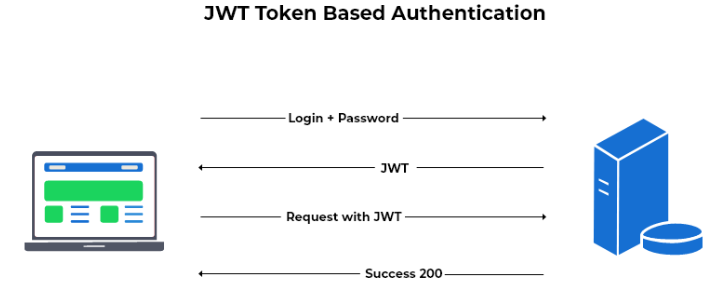
**Disadvantages of session based authentication**

Cookies are prone to XSS and CSRF attacks and the sessions are stored in the server’s memory. Thus scaling becomes an issue when a large number of users log in.

**Token based authentication**

Now, many web applications use JSON Web Token (JWT) instead of sessions for authentication. In a token based authentication, the server creates a JWT token with a secret and sends the JWT token to the client. The client stores the JWT token and includes JWT token in the header with every subsequent request. The server would then validate the JWT token with every request from the client and send a response. For example,

Authorization: Bearer eyJhgGciOiJIUzI5NiIsInR5cCI6IihXVCJ9.eyJzdWIihiIxMjM0NTY3tDkwIiwibmFtoSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfr.SflKxvRJSMeKKF2QT4fwpMeJf36POk6yd\_madQsswd3



**How token based authentication works**

* User tries to log in using their login credentials.
* The server verifies the credentials, and it returns a signed token.
* This token is stored in the client-side, can be stored in local storage, session storage or in a cookie.
* This token is included in the header as an authorization header in each subsequent request to the server. The server decodes the token and if the token is valid, then processes the request.
* And if a user logs out of the application then the token is destroyed on client-side, it’s not necessary to interact further with the server.

**Advantages of token based authentication**

A token is stored on the client-side, so there is no problem with scaling and there is no need to store session information in the server.

**Disadvantages of token based authentication**

Since JWT contains more user information, the size of JWT is much bigger when compared with the session ID stored in a cookie and it can’t revoke the access to a user.

**Attacks on session security and their safety measures**

**Manipulator in the middle attack (MITM)** is when an attacker intercepts a request between a user and the server. If the request is not encrypted, an attacker can view the sensitive information in it. To protect against this type of attack, use HTTPS and secure cookies throughout the application.

**Cross-site scripting (XSS)** is a security exploit which allows an attacker to inject malicious scripts into a website. Eg: <script>alert(document.cookie);</script> may alert the cookie in the browser if the application is vulnerable to XSS. To protect against this type of attack, implement proper sanitization for user inputs.

**Cross-site request forgery (CSRF)** is an attack that forces an authenticated end-user to execute unwanted actions on a web application. An active session can be used by this vulnerability so that the necessity of cookies can be compromised. To protect against this type of attack, use an anti-CSRF token and make sure that the token is always validated.

**Session fixation** is an attack that permits an attacker to hijack in to a valid user session. The attacker has to provide a valid session ID of the web application and try to make the victim’s browser use it. To protect against this type of attack, do not use anonymous data as cookies and properly validate and invalidate the cookies.

**Brute force attack** is an attempt to predict cookies, once that attacker has encrypted or unencrypted cookies. The best way to prevent this is to use long auth tokens or cookies with high entropy.

**Best practices for token authentication**

**Validate JWT tokens**- Reject a JWT token that does not conform to the signature algorithm and validate all claims, issuers, expiration date and audience. Avoid sensitive information in payload- Certain algorithms sign tokens to protect against manipulation, which can be easily decoded.

**Give tokens an expiration time**- A signed token is valid forever unless the signing key is changed or expiration is set explicitly. This could pose potential issues. So, consider implementing a secure and suitable strategy for managing tokens.

**HTTPS communication**- Do not send tokens over non-secure connections as these requests can be intercepted and tokens can be compromised.

**Do not hardcode token**s- Hardcoding tokens in web applications can make the process very simple, but it helps an attacker to compromise the web application with less effort.

**Best practices for session authentication**

**Lengthy and random session ID**- Make sure the length of the session ID is long enough and in a completely random way to prevent brute force attacks. The recommended length is 128 bits.

**Session ID without user-specific data**- Ensure that the session ID does not contain any additional user-related sensitive data. The data should be a random string of characters without any meaning.

**HTTPS communication**- HTTPS should be used for all session-based applications.

**Secure and HTTPonly cookies**- Session cookies should be created with secure and HttpOnly attributes.

**Manage sessions properly**- Destroy sessions upon closing browser, timeout, logout or log-in from a separate location.

**Conclusion**

To sum up, web application security has come a long way, and it is no longer bound by cookies and sessions. However, the session involved in the tokens is still a topic to focus on. In order to keep sessions safe, take proper session security measures and adopt secure and suitable methods.

**3 Secure Firewall Configuration for Web Applications**

Security threats have always been and continue to be a menace for small and big organisations alike. There’s a possibility of many security threats existing in a typical enterprise distributed application including the likes of DDoS, SQL injection, Cross-Site Scripting, etc.

Hackers are getting increasingly sophisticated and, thankfully, so are cyber defence systems. We have the option to set up the first line of defence against security threats with the help of a web application firewall (WAF).

**What is a Web Application Firewall?**

A web application firewall or WAF has a promising role in an IT infrastructure. WAF helps protect web applications from attacks such as cross-site forgery, cross-site-scripting (XSS), file inclusion, and SQL injection, etc by filtering and monitoring HTTP traffic between a web application and the internet.

Deploying a WAF in front of a web application establishes an initial level of protection between the web application and the internet.

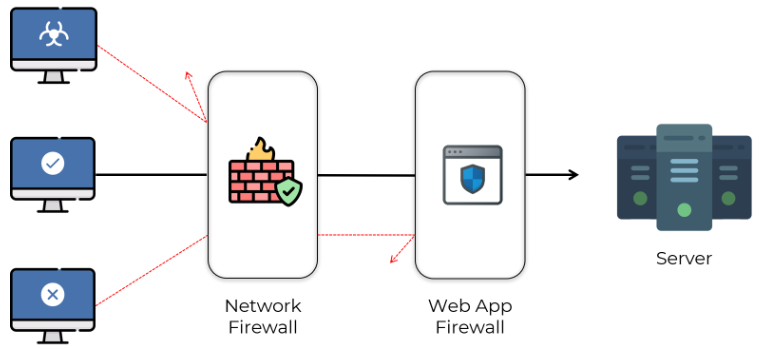
It is a type of reverse-proxy, protecting the server from exposure by having clients pass through the WAF before reaching the server.

WAFs may come in the form of an appliance, server plugin, or filter, and maybe customized to an application. The effort to perform this customization can be significant and needs to be maintained as the application is modified.

Web application firewalls, positioned between web client and web server, monitor every request and response within the application layers. They search for specific “attack signatures” to identify a specific incoming attack, whilst monitoring for abnormal behaviours that are incompatible with past traffic patterns.

A web application firewall is an essential tool that ensures protection against malicious users and bots.

However, many of the web application firewalls are not configured securely. It is easier for attackers to take advantage of a misconfigured web application firewall.



**Web Application Firewall Policies**

A WAF operates through a set of rules often called policies. In the WAF policy configurations, you have options to enable rules to detect attacks at the request line, query-string, URI, request headers, request body, response code, or response body.

Some of the WAF policies are listed below:

**Web attack signature policy**

The signature database includes signatures that can detect known attacks and exploits found in 22 scan points. In your policy configuration, you can choose classes of scan points to process: HTTP headers, HTTP request body, and HTTP response body.

**URL protection policy**

This policy enables you to create rules that detect patterns in the URL or the file extension.

**HTTP protocol constraint policy**

The HTTP protocol constraint policy enables you to create rules that restrict URI, header, and body length, HTTP method, or HTTP response code.

**SQL/XSS injection detection policy**

This policy includes rules to detect SQL/XSS injection in the HTTP request URI, HTTP referer header, HTTP cookie header, or HTTP request body.

**Bot detection**

This policy includes rules to detect bots. A bot is an application that runs automated tasks over the internet. The WAF supports two methods for detecting bad bots: signature detection and behaviour detection. Whitelisting can also be employed for detecting the good ones.

**Security Mechanisms of Web Application Firewalls**

The main aim of using a WAF is to secure the existing, often productive web applications. Besides the basic protection via blacklisting and whitelisting, the WAF can also act as a central service point for completing tasks which should otherwise be on the application side.

**The table below shows the most common issues faced by a web application and how WAF protects the application from them.**

|  |  |
| --- | --- |
| **Issues** | **Countermeasures** |
| Cookie protection | Cookies can be signed. Cookies can be encrypted. |
| Information leakage | Cloaking filter, outgoing pages can be “cleaned” (error messages, comments, undesirable information). |
| Session riding (CSRF) | URL encryption/token. |
| File upload | Virus check (generally via external systems) via ICAP linked to the WAF. |
| Parameter tampering | Parameter manipulation can be prevented via URL encryption (GET) and parameter encryption (GET and POST). Site usage enforcement, meaning the possible sequence of URLs can be fixed or can be detected. |
| Forced browsing | Can be prevented via URL encryption. Site usage enforcement. |
| Path traversal (URL) link validation | Can be prevented via URL encryption. Site usage enforcement. |
| Logging | All or only specific/permitted parts of the data of a request and the connected tests can be logged |
| SSL | WAF can force SSL with pre-defined encryption strength. SSL termination on the WAF forwarding of the SSL data to the application. An SSL connection is possible from WAF to application. |
| HTTP request smuggling | Is prevented via strict testing of the conformity to standards of each request. |
| Data validation and injections | Can be tested to a very detailed degree (length, constant value/range of values, e.g. for SELECT, character area); validation possible with whitelist and/or blacklist (signature). Rules can in part be generated automatically. |

**Best Practices for Web Application Firewall Configuration**

Even though many companies have implemented a WAF, most of the web application firewall configurations are not secure. It would be more effective if the following points are considered while configuring a web application firewall.

**Coordinate Web Application Firewall Configuration and Web Application Update**

Configuring the web application firewall in parallel with a web application’s update cycle prevents significant issues. Web application firewalls should be configured when the website is being updated or when patches are implemented.

It is most probable that all or a part of the website will not be protected if the configuration and update do not happen simultaneously. And if the WAF is unable to cover those parts, then the consequences can be severe for the site’s users. Maintain the web application firewall along with a change in the security surface of your application.

**Check for vulnerabilities**

Always ensure that the code does not contain bugs or vulnerabilities before implementing the web application firewall.

Even though WAF is the first line of defence, if the website itself is not secure, it will drag the whole web infrastructure into danger. So make sure to scan your website regularly to make sure there are no security issues.

**Use a web application firewall with built-in acceleration**

Prefer a WAF with a built-in acceleration feature, which mitigates web latency and speeds overall traffic so that it will not affect user experience.

**Lockdown the backdoor**

The protection of databases and back-end applications is also important as front end applications. Securing both ends can be achieved with a database activity monitoring product along with a proper web application firewall configuration and regular vulnerability assessment.

**Understanding the web application environment**

A greater level of understanding about the application environment, security posture and possible loopholes are necessary to make sure you get the best out of a web application firewall implementation.

**Summary**

Web applications of all kinds have in recent years increasingly become the target of hackers. It’s our responsibility to upgrade our defence systems to overcome the challenges and threats and keep our applications secure.

A web application firewall acts as the first line of defence and it allows you to make sure that an application is safe by putting in a reasonable amount of initial effort. In addition to implementing them, it is also necessary to properly configure them so that it can be an effective defense mechanism in your bid to keep away hackers.

**4 How to Improve Web Application Security?**

Web applications play a key role in determining the success of a business. Many companies solely depend on web applications for their business, offering a SaaS product for other customers and also building web apps for internal use.

Yet it’s a fact that many companies don’t know how to keep track of their web application security and improve it.

Content management systems (CMS) like WordPress, Joomla and website builders have made it easy for everyone to create a website. Most of the website owners forget the fact that the attack surface of internet-facing web applications is much wider and that they need adequate security.

Whenever a customer or visitor is on your website you have to make sure that their data is safe.

If you fail to keep your customers’ data safe, you could be at the receiving end of a cyber attack which can lead to huge business loss and it can also get you sued. You have to keep in mind that no methods can guarantee your web application will be safe from attackers forever.

In this article, we’ll be looking into certain best practices that will help you improve web application security and prevent being an easy target for cyber attackers.

**Choose a secure host**

Even if your website has top of the line security it won’t do you any good if you are not using a secure host.

Do some research and choose a hosting company which has a good reputation and does not have much downtime issues. It is also recommended to check whether they meet your other unique requirements depending on your business needs.

Some of the key points to consider while choosing a hosting server is:

* Does the web host offer a Secure File Transfer Protocol (SFTP)?
* Is FTP used by Unknown User disabled?
* Does it use a Rootkit Scanner?
* Does it offer file backup to a remote server?
* How well do they keep up to date on security upgrades?
* Whether they provide technical support whenever necessary.

**Know your web applications and prioritize them**

It’s quite surprising that most organizations are still unaware of how many web applications they have or where they are hosted.

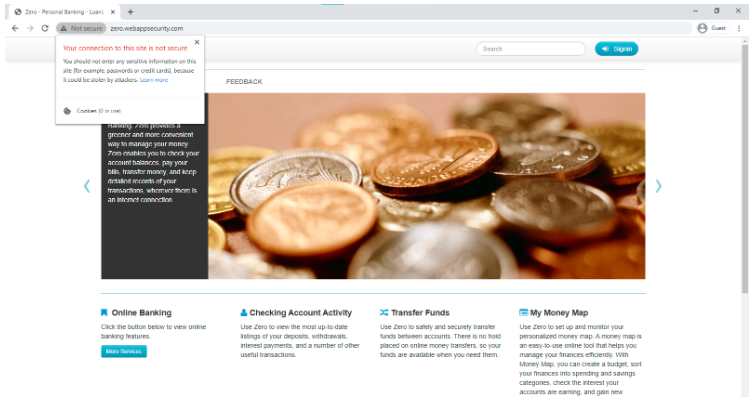
It is important to have a list of web applications including your organization’s and other third-party applications and prioritize it according to the amount of damage that could be done if something goes wrong.

**Secure your login pages using SSL (HTTPS) encryption**

To keep your website safe, you need a URL with SSL (or even better TLS) encryption enabled.

HTTPS encrypts data sent from your browser to the web server and prevents a 3rd party from reading it while in transit. So, even if an attacker tries to intercept the data (manipulator-in-the-middle attacks), it will be useless.

If your website doesn’t have a valid SSL certificate, it is often flagged by most of the popular browsers as insecure. Browsers warn users not to send any personal, payment or password information in such cases.



**Always sanitize and validate user input**

Never trust any user input. This is a very common security mistake found on many websites.

If the user input data is not properly sanitized and validated, your website has a high risk of being targeted by attacks like XSS, SQL injection and other types of injection attacks. Sanitizing a user input may include the elimination of unwanted characters by removing, replacing, encoding, or escaping the characters.

Eg: <script>alert("XSS:);</script>

Html encoded &lt;script&gt;alert(&quot;XSS:);&lt;/script&gt;

Validation is the process of ensuring that the user input data falls within the expected characters. You can create a whitelist or a blacklist to achieve this. In whitelisting, only the approved characters will be allowed and the rest of the characters will be rejected.

For example, if your website has a field for accepting phone numbers, you could whitelist numbers from 0 to 9. If the user tries to input any other character it won’t be accepted. On the contrary, in blacklisting, the list of defined characters will not be accepted as a valid input.

If possible, use whitelisting rather than blacklisting. When using a blacklist, you have to consider all the possible invalid options and if you miss something, you could expose your web application to hackers. This is why it’s much better to simply whitelist what is valid.

**Have a good password policy**

Whenever there is a talk about web application security, good passwords policies are always part of it.

Most companies nowadays have standard password policies to improve their online security. Even with these password policies, there are so many websites, databases, and programs, an admin or website owner has to keep password protected.

As a result, a lot of people end up using the same password in almost all places in order to remember their login information. But it’s a significant security mistake.

Nowadays, attackers use automated brute-forcing softwares to check whether sites are vulnerable. To protect against brute force, always use unique and complex passwords containing both uppercase, lowercase, numbers and special characters.

Use hard to guess passwords and also try not to use any personal information as passwords. If you try to keep a password in your memory, it is almost always an easy one. So, it is recommended to use a password manager for storing your passwords.

Also, if two-factor authentication (2FA) is available, always opt-in for it. Besides password, this will add an extra layer of security for your accounts.

**Limit access rights and credentials**

Whenever you think of giving access to your application, databases, etc, always follow the principle of least privilege (PoLP).

In simple terms, give users access only to data and tools they need to do their job. At first, you may feel comfortable giving high privileges to certain employees or your close friends thinking that they would use it with caution. Most of the time employees don’t think about website security when logging into the application or databases. This could pose a significant security risk to your application.

Make sure that a person has prior experience using your application before you give admin privileges to them. Check if they’re aware of how to avoid a security breach or a misconfiguration.

In most companies, employees are not permanent. If an employee leaves your company, you have to immediately revoke all the access given to that employee and also change the passwords if necessary. A best practice is to have a record of all employees with the privileges they have and what are the changes made by them.

**Keep your website clean**

Every database, application, or plugin on your website is another possible point of entry for hackers. You should delete any unwanted files or databases connected to your web application and close all unwanted open ports in your server that are no longer in use.

It is also important to keep your file structure organized to keep track of changes and make it easier to delete old files.

**Ensure everything is up to date**

It’s very important to keep all softwares, plugins and extensions you have up to date.

Attackers always keep an eye on the latest security vulnerabilities and know how to exploit them. If you don’t update regularly your company might become a likely target for attackers.

Always keep a note of every plugin or extensions you have and update whenever new versions are available. Be aware of the latest technologies and vulnerabilities found in the components you use by exploring different reputable web application security blogs.

**Keep regular backups**

Your web applications’ data is always at a constant risk. One of the best methods to keep your site safe is to have a good backup system in place.

Do not store your backups on the same server as your website is hosted on because if the server is under attack, the backups are also vulnerable. It is better to keep your backup on a home computer or hard drive. If possible find a climate-controlled off-site place to store your data and to protect it from hardware failures, attacks, and viruses.

Another best option is to backup your website in a cloud-based platform which makes storing data easy and allows access from anywhere.

It is recommended to use a solution where you can schedule your site backups automatically. Most of the content management systems have plugins or extensions that can automatically backup your website. Also, make sure your solution has a reliable recovery system too.

By doing this, you can restore files from any point to any other point you want to before a virus or ransomware wrecks havoc.

**Make sure to tweak the default settings of your CMS**

It is recommended not to run your CMS with default configurations. These configurations include default usernames/passwords, user permissions, login attempts etc.

Attackers use automated bots to scan websites. If your website is running using defaults usernames or passwords or with any misconfigured default settings, then your website will become a likely target.

**For example,** WordPress and other popular CMS platforms come with a default username “admin” and a password “password”.

The default login page of a WordPress website will be like example.com/wp-admin or example.com/wp-login.php. By default, WordPress doesn’t limit login attempts which could result in a brute force attack. Another overlooked security misconfiguration is specifying file permissions. Each file has three permissions which are represented by a number.

* Read (4): View or read the file contents.
* Write (2): Change the file contents.
* Execute (1): Run the program file or script.

Other than these default file permissions there are also three user types.

1. Owner – Often, the creator of the file, but ownership can be changed. Only one user can be the owner at a time.
2. Group – Each file is assigned to a group. Users who are part of that specific group will gain access to the permissions of the group.
3. Public – Everyone else.

Assign file and user permission accordingly and never keep the default permissions as it is because at some point you may run into security issues.

**Run security tests on your website for vulnerabilities**

You should perform security checks and scans regularly if you want to stay on top and improve web application security.

It is recommended to perform a security scan at least once in a month and after each major changes, upgrades or downgrades you do to your website. You can use an automated penetration testing tool for this like the one we offer at Beagle Security.

**Get professionals to “hack” your application**

You should consider hiring professional hackers to try to penetrate your application if your business revolves around a web app that your company is selling or is using internally.

It’s very difficult to stay on top of web application security on your own. White-hat hackers can help you identify and fix the vulnerabilities present in your website before a real attacker finds and exploits it.

Also, instead of hiring professional hackers from a firm, you could organise a bug bounty program where you pay a reward to someone who finds a vulnerability in your web application.

**Summary**

Nowadays companies mainly depend on web applications for their business.

Every website owner assumes that their website is safe and secure and forgets how exposed their applications are. Creating a website is very easy but keeping it secure is the difficult part.

It’s quite surprising how many options companies have for improving web application security but they hardly take the right measures.

**5 A Comprehensive Guide to Web Application Security**

“If one thinks that they know it all about cybersecurity- this discipline was probably not explained appropriately to them.”- Stephane Nappo

Digital adoption amongst modern-day businesses has become more prominent than ever. Today, every business wants a digital presence to reach a global audience.

Cybercriminals realize the need for businesses to connect with their customers. Therefore, they pose a comprehensive challenge in front of global businesses.

After COVID-19 struck the world, web app security has become a topic of debate. Growing reliance on eCommerce, eLearning, and digital payment systems have forced global businesses to adopt desperate measures to keep their website out of scrutiny and data theft.

So, what is web application security? And why do hackers do what they do?

Let’s understand it:

**What is web app security, and why do cybercriminals choose to compromise websites?**

Web application security is a series of steps taken to protect a website from digital security threats. Since hackers can compromise the application codes, WAS aims to protect it by restricting unsolicited access and promoting identity verification.

Hackers would generally target SaaS companies, CMS platforms, and hosting providers who serve websites on a large scale.

Once they are compromised, every website that they are facilitating also collapses.

So, the question is, why do hackers do that?

Well, hackers have a range of motives behind compromising websites. They want businesses to endure economic harm, social harm, reputational harm, etc.

Some maniacs also compromise websites for fun as they get a sick sense of pleasure.

Back in 2020, hackers targeted 130 high-profile Twitter accounts, including the likes of Elon Musk, hackers (for doing so) received 400 payments in bitcoins up to $121,000.

Therefore, even tech giants can get compromised if they are vulnerable. Another question that stems from here is- what ways can hackers adopt to compromise our web applications?

Let us look at them:

**How can hackers attack your web application?**

**1. SQL injection**

Websites use SQL or Structured Query Language to connect with databases. Using SQL, a website can store, delete, retrieve, update, or create databases. Moreover, SQL stores user transaction details and logs them on a website.

When an SQL injection happens, hackers use search queries used by the database to exploit loopholes in the database.

For example, a hacker can type ‘or 1=1 instead of a normal username and password. If a website adds this string to an SQL command used to check user existence in a DB, then the query will return as “true”. Thus, a hacker can easily gain access to a sensitive area.

**Solution:**

Since hackers can use automated tools to inject SQL, you need to filter the user input properly. Programming languages have special features to ensure the proper filtration of user input.

**2. XSS attacks**

XSS or Cross-Site Scripting attack involves injecting JavaScript codes into websites as hyperlinks.

When users click on any such hyperlink, their data can be stolen, ads displayed on the page may be altered, and sometimes even the entire session can be highjacked.

XSS scripts are tough to spot as hackers add them to social media posts, comments, recommendations, and reviews as a valuable piece of information on which a user is compelled to click.

**Solution**

Since hackers can insert malicious codes as user inputs in social media, web forums, and websites where users are most likely to click, website owners need to ensure that user input gets filtered appropriately and malicious codes get erased.

**3. DDoS attacks**

DDoS or Distributed Denial of Service attacks is carried out by malware-infected computers that send data requests to your website.

In most cases, the computer owner may not know that their computer is being used to overwhelm a website’s server.

Hackers use many such computers to overwhelm a server by sending traffic to a point where the website crashes.

In some cases, hackers demand huge ransom amounts to let the website go live again.

**Solution**

To mitigate DDoS attacks, you need to add filtration processes so that malicious, spoofed, and malformed packets from unknown sources get dropped. Also, have an aggressive strategy for connection timeouts. If you are using firewalls, make sure they come with DDoS security.

**4. CSRF attacks**

CSRF or Cross-Site Request Forgery is a type of malicious attack that hackers use after accessing a web application.

Here a hacker can give unauthorized commands from a user’s account and trick the web application into believing them. The biggest downside to these attacks is that no hindrance can stop hackers from transferring funds and obtaining sensitive account information and user data.

Now, you must be wondering, how is CSRF different from XSS? The main difference is that CSRF attacks are made after obtaining account access. Unlike XSS, a hacker has all the credentials to claim an account here.

**Solution**

To prevent CSRF attacks, you need to check HTTP headers to conclude whether the request is coming internally from an application or outside from an external source.

**5. DNS spoofing**

DNS spoofing attacks aim to divert website traffic from a legit website to a malicious one.

Hackers also use this technique to gain reconnaissance about where the traffic is being diverted.

The biggest downside to this attack is that neither the website owner nor the user will know that their connection has been interrupted and transferred to an illegitimate site.

After redirection, users are tricked into sharing their sensitive data like bank details, credit/debit card numbers, and phone numbers to steal their data.

**Solution**

To prevent DNS spoofing, you need to set up a TTL (Time-To-Live) or Hop limit to reduce the period of computer data. Also, keep clearing DNS catches from the computer.

**6. Social engineering attacks**

Speaking of convincing users and website admins to share their data, social engineering attacks are also in full swing these days.

Here are some of the common social engineering attacks:

**Phishing emails**

In a phishing attack, emails impersonating a brand’s identity are sent to users to trick them into believing that they are coming from a legitimate source.

Once the trust is established, emails asking for contact details, bank account numbers, and addresses are sent on behalf of a legit organization. Also, they may ask you to click unsolicited links containing malicious files.

**Baiting**

Another common form of social engineering attack is baiting. Here hackers can show files containing valuable information like money hacks or free Netflix access. Once you click on them, malicious codes will be automatically downloaded into your system.

**Pretexting**

In these attacks, hackers impersonate one of your clients or employees and call or text you, asking for your sensitive bank, username, password, or company details.

**Solution**

The only way to prevent these attacks is by staff training and increasing customer awareness about these attacks. When people are educated about them, they are more likely to understand their risks.

**7. Non-targeted attacks**

As the name suggests, these attacks are not meant to compromise your website.

You must be wondering, what is the purpose of these attacks then?

These attacks target web hosts and CMS platforms instead of a specific website. They believe in capturing big guns instead of spending their resources fighting a foot soldier.

Non-targeted attacks compromise CMS platforms like WordPress and Joomla by targeting a specific outdated version.

Since not all websites are kept up to date, hackers use automated bots to find out such outdated websites that become easy targets for them.

**Solution**

The solution to stop non-targeted attacks is simple. Keep your plugins, CMS platforms, and web hosting software up to date.

**8. Memory corruption**

In memory corruption, hackers modify a space in the memory for installing unsolicited and malicious software.

Hackers can further use that software to access all devices, networks, and programs connected to that computer.

**Solution**

To avoid memory corruption, you need to scan it through an anti-malware tool regularly. If your memory has already been corrupted, unclip the faulty memory or replace it with a new one.

**9. Buffer storage**

Buffer overflow results from data containing malware getting overwritten multiple times in a storage location, especially in a target memory space.

Since the data gets multiplied in the storage, its malicious content is also multiplied. As a result, more vulnerabilities occur in the system.

**Solution**

To stop buffer storage, evaluate your codes properly. Also, have objective quantifiers for your code.

Thus, these are some ways hackers can enter your website and breach user data.

If cyberattacks are not mitigated, then they can have a serious impact on your reputation. Customers like to leave their data in safe hands. So, what else can you do to protect your website from such attacks?

Here are 8 best practices that you can inculcate in your website to repel cyberattacks.

**Essential Practices for Web Application Security**

**1. Attack your website**

The best way to protect against your enemy is by thinking like your enemy. You can install the best-in-class security tools, but how do you know that they will perform optimally come the doomsday?

Therefore, you must hire experts who can run an attack in an isolated environment so that you don’t damage anything in the process.

The reason why we don’t recommend DIYing it is because a layman can do more harm than good. They have no idea about how their security protocols will be performing. Thus, attack your website under expert surveillance.

The attacks may include SQL injection, XSS, CSRF, DNS spoofing, sensitive data leak, broken authentication, etc.

**2. Invest in an SSL certificate**

We are living in 2022, and security is no afterthought today. If you want to play the long game, you have to learn something about security.

One of the paramount security protocols that you need is an SSL certificate. A Secure Socket Layer certificate is a security technology that encrypts your data and passes it over a protected network so that hackers cannot see or intercept it.

Since your users will be sending information like bank details, debit/credit card details, usernames, passwords, and addresses, SSL will help you hide all of that from cybercriminals.

Search engines like Google have already stated in their Page Experience Update rolling out for desktops that SSL is a must if you want Google to consider you for rankings.

The idea behind this is to ensure that users do not have to worry about the security of every website that they visit.

Also, suppose you want to facilitate online payment systems on your web application. In that case, SSL is a major implication that must be fulfilled to acquire the license by PCI or the Payment Card Industry.

Lastly, if you are worried about its cost, you can rest assured because SSL certificate cost hovers around $8.00/year.

**3. Read and educate**

In this information age, reading and education are of immense value. Educated hackers can be lethal, and educated staff is tough to fool through social engineering attacks.

There are plenty of blogs and videos that can help educate you about the current market trends. You can follow them and stay one step ahead of hackers.

Uneducated users often fall victim to non-targeted attacks where a hacker can easily compromise their outdated CMS software.

**4. Backup your data**

You never know when that day will come when all your security measures will be put to the test. Therefore, you must keep everything backed up, like countries preparing to fight against rogue states.

Data backups ensure that you don’t lose anything even if you lose the battle against the attacker.

They may be able to take your website down, but you can go live within hours of the crash if you have maintained proper data backups.

So, where to back up your data? We recommend you use cloud-based storage services that are comparatively safer and readily available than a hard drive which can be stolen or corrupted.

**5. Scan your website**

Though malware is made scanner-proof, but if you use quality scanners that can go off-pattern to detect threats then you can stop them from causing any harm.

They can quarantine the threats, preventing them from causing any further damage to your website.

It would be best to run a full-website scan once a month to keep clear of cyberattacks. However, scanning websites may not be enough; you must also scan your computer from time to time.

Therefore, scanning your website can remove potential vulnerabilities.

**6. Outsource your security**

If you are a layperson, checking all security aspects of your website can be a challenge. Therefore, you can invest in a well-versed and qualified person in website security.

Since your business is here to grow and prosper, a once-an-year investment in a security firm may not be a bad idea.

They have experienced professionals who know the current security challenges and guide you appropriately about website maintenance. Also, they have specialized scanners and security tools that can catch vulnerabilities invisible to spot for available scanners.

So, outsourcing your website security on a freelance or contract basis can be a great way to stay secure.

**7. Keep your software up to date.**

We already talked about how non-targeted attacks can compromise outdated software and hijack your website.

Therefore, software updates are essential to keep hackers at bay. In some CMS platforms, updates can automate, but you would have to take out time to update them in others.

Either way, you must take time to update them to protect your data and sensitive user information from the grasp of the bad guys.

**8. Manage your passwords**

We hate to change our passwords now and then. But passwords can become an easy gateway for hackers to creep into your website.

Therefore, educate your customers and staff about the importance of strict passwords. You can also integrate a two-factor authentication system to strengthen them.

2-FAs send a unique 4–6-digit code that a user is required to enter for authorization purposes. Failing to do so even after multiple attempts would automatically block the user’s IP address.

You can even apply constraints while users set up their passwords on your site. You can channel them to enter strict passwords that involve the use of numbers, words, and special symbols.

A strong 14-digit password is considered a good one as it is hard to guess for malicious bots during brute force attacks.

**Summary**

We all know how competitive online markets have become. It is tough to predict what means hackers can use to compromise your website.

Therefore, you need to defend all fronts as an owner. It would help if you fortified vulnerable areas through staff education, SSL certificates, firewalls, and strong password policies so that hackers fail to find any opening for exploitation.

Technical knowledge is also the need of the hour. Since XSS, CSRF, DDoS, DNS spoofing, and SQL injection attacks are on the rise, you must know how these attacks are implemented and what repercussions your website will bear from them.

So, study all these above attacks and implement these 8 strategies to stay protected in the cyberworld.

**6 Web Application Security for the Everyday Software Engineer**

**6.1 Introduction**

With great power comes great responsibility

Today’s web platform allows developers to build products like push notifications, geolocation, and localStorage with technologies that were unimaginable just a few years ago.

These technologies come at a cost though. The spectrum of vulnerabilities is amplified, which means that there’s more we must know when developing for the web.

When iframes were introduced, everyone was quick to point out how great they were, as they easily allowed embedding content from different webpages. Few thought that the very same technology would serve as the basis for clickjacking, a vulnerability that is only possible thanks to iframes, an additional feature of the HTML standard.

As Wikipedia puts it:

Clickjacking is possible because [of] seemingly harmless features of HTML web pages.

Did you know that Cross-Site Request Forgery (CSRF) attacks are about to disappear? How you ask? It’s all thanks to browsers supporting SameSite cookies which will be discussed further on in the course.

**An everchanging web landscape**

The landscape surrounding the web is changing quickly. Having a good understanding of the platform and a keen eye on security is important for every developer. This is the goal of this course: to make sure we’ve raised our security awareness.

This course was written to demystify web security and make it easier for the everyday developer to understand important, security-related aspects of this universal platform.

**Code**

Some of the code provided will have to be run locally on your machine. It is advisable to clone this repo: https://github.com/odino/wasec, and then pull the latest code when running each example so you have the latest version of it.

**Terminologies**

HTTP: Hypertext Transfer Protocol

MDN: the Mozilla Developer Network, available at developer.mozilla.org/en-US/

OWASP: the Open Web Application Security Project, an online community that produces security-related documentation and recommendations at owasp.org

**6.2 & 6.3 Understanding the Browser**

Refer 🡪 Introduction to Browser (Understanding Browser & HTTP )

**6.4 Protection through HTTP Headers**

**Introduction**

As we’ve seen, servers can send HTTP headers to provide the client with additional metadata around the response. Besides sending the content that the client requested, servers are then allowed to specify how a particular resource should be read, cached or secured.

There’s a large spectrum of security-related headers that we should understand, as they have been implemented by browsers in order to make it harder for attackers to take advantage of vulnerabilities. The next paragraphs try to summarize each of them by explaining how they’re used, what kind of attacks they prevent, and a bit of history behind each header.

**HSTS**

Since late 2012, HTTPS-everywhere believers have found it easier to force a client to always use the secure version of the HTTP protocol, thanks to the HTTP Strict Transport Security. A simple **Strict-Transport-Security: max-age=3600** will tell the browser that for the next hour (3600 seconds) it should not interact with the applications with insecure protocols.

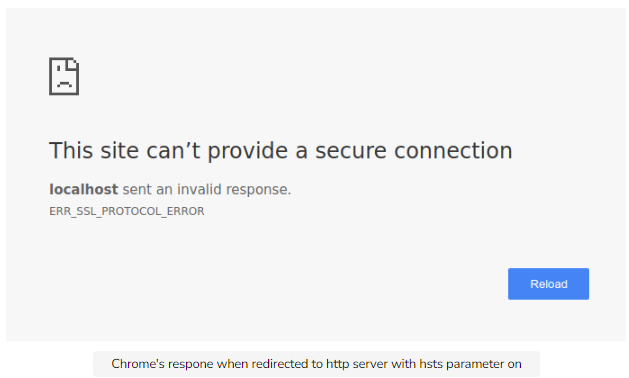
When a user tries to access an application secured by HSTS through HTTP, the browser will simply refuse to go ahead, automatically converting **http://** URLs to **https://**.

You can test this locally with the code at **github.com/odino/wasec/tree/master/hsts**. You will need to follow the instructions in the README (that involves installing a trusted SSL certificate for **localhost** on your machine, through the mkcert tool, and then try opening [**https://localhost:7889**](https://localhost:7889).

**HTTPS**

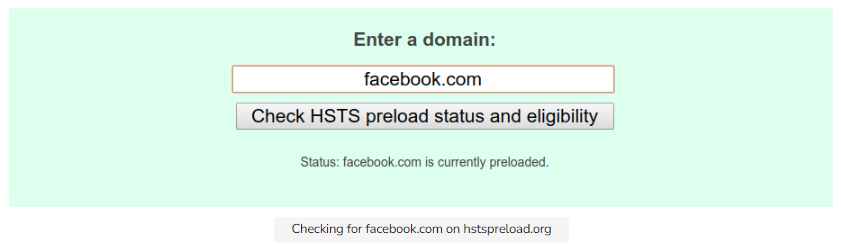
There are two servers in this example, an HTTPS one listening on **7889**, and an HTTP one on port 7888. When you access the HTTPS server, it will always redirect you to the HTTP version, which will work since there is no HSTS policy on the HTTPS server.

If you instead add the **hsts=on** parameter in your URL, the browser will forcefully convert the link in the redirect to its https:// version. Since the server at 7888 is http-only, you will see a page that looks more or less like this:



You might be wondering what happens the first time a user visits your website, as there is no HSTS policy defined beforehand. Attackers could potentially trick the user to the **http://** version of your website and perpetrate their attack there, so there’s still room for problems. That’s a valid concern, as HSTS is a trust on first use mechanism. It tries to make sure that once you’ve visited a website, the browser knows that subsequent interactions must use HTTPS.

A way around this shortcoming would be to maintain a huge database of websites that enforce HSTS, something that Chrome does through **hstspreload.org**. You must set your policy then visit the website to check whether it’s eligible to be added to the database. For example, we can see Facebook made the list.



By submitting your website to this list, you can tell browsers in advance that your site uses HSTS so that even the first interaction between clients and your server will be over a secure channel. This comes at a cost though, you really need to commit to HSTS. It’s not an easy task for browser vendors to remove your website from the list.

**Note**:

Be aware that inclusion in the preload list cannot easily be undone.Domains can be removed, but it takes months for a change to reach users with a Chrome update and we cannot make guarantees about other browsers. Don’t request inclusion unless you’re sure that you can support HTTPS for your entire site and all its subdomains for the long term. hstspreload.org

This happens because the vendor cannot guarantee that all users will be on the latest version of their browser, with your site removed from the list. Think carefully, and decide based on your degree of confidence in HSTS and your ability to support it in the long run.

**6.5 HTTP Public Key Pinning**

**Why HTTP Public Key Pinning?**

HTTP Public Key Pinning (abbr. HPKP) is a mechanism that allows us to advertise which SSL certificates to expect when a browser connects to our servers. It is a trust on first use header, just like HSTS, meaning that, once the client connects to our server, it will store the certificate’s info for subsequent interactions.

If at any point in time the client detects that another certificate is being used by the server, it will politely refuse to connect, rendering man in the middle (MITM) attacks very hard to pull off.

This is what an HPKP policy looks like:

Public-Key-Pins:

pin-sha256="9yw7rfw9f4hu9eho4fhh4uifh4ifhiu=";

pin-sha256="cwi87y89f4fh4fihi9fhi4hvhuh3du3=";

max-age=3600; includeSubDomains;

report-uri=<https://pkpviolations.example.org/collect>

**HPKP is dangerous**

The header advertises what certificates the server will use (in this case it’s two of them) using a hash of the certificates. It includes additional information like the time-to-live of this directive (**max-age=3600**), and a few other details. Sadly, there’s no point in digging deeper to understand what we can do with public key pinning, as this feature is being deprecated by Chrome, a signal that its adoption is destined to plummet very soon.

Chrome’s decision is not irrational, it is simply a consequence of the risks associated with public key pinning. If you lose your certificate, or simply make a mistake while testing, your website is gone for the duration of the **max-age** directive, which is typically weeks or months. As a result of these potentially catastrophic consequences, adoption of HPKP has been extremely low, and there have been incidents where big-time websites have been unavailable because of a misconfiguration.

All things considered, Chrome decided users were better off without the protection offered by HPKP, and security researchers aren’t entirely against this decision.

**Note:**

**HPKP gone wrong**

Smashing Magazine, a leading website in the field of web design, documented its disastrous experience with HPKP in a blog post in late 2016.

Long story short, the website was unavailable, due to a misconfiguration in their Public-Key-Pins header. When their SSL certificate expired, they had no way to issue a new certificate that would not violate their previously set HPKP policy. As a result, most of their users could not access the website for four days.

Moral of the story? HPKP is dangerous and even the best make mistakes.

While HPKP has been deprecated, a new header stepped in to prevent fraudulent SSL certificates from being served to clients, **Expect-CT**. Let’s study it in the next lesson.

**6.6 Expect-CT**

**Why Expect-CT?**

The goal of Expect-CT is to inform the browser that it should perform additional background checks to ensure the certificate is genuine. When a server uses the Expect-CT header, it is requesting the client to verify that the certificates being used are present in public Certificate Transparency (CT) logs.

**NOTE**:

The Certificate Transparency initiative is an effort led by Google in order to:

[provide] an open framework for monitoring and auditing SSL certificates in nearly real time.

Specifically, Certificate Transparency makes it possible to detect SSL certificates that have been mistakenly issued by a certificate authority or maliciously acquired from an otherwise unimpeachable certificate authority. It also makes it possible to identify certificate authorities that have gone rogue and are maliciously issuing certificates.

certificate-transparency.org

Note that a rogue server wouldn’t set the expect-ct header, putting themselves on the line. A genuine server can ask clients to opt-in with all subsequent requests to be validated with CT moving forward. If the client gets tricked into connecting to a malicious server, the attack will never work as the SSL certificate won’t pass the CT validation.

**Sample header**

The header takes this form:

Expect-CT: max-age=3600, enforce, report-uri="https://ct.example.com/report"

In this example, the server is asking the browser to:

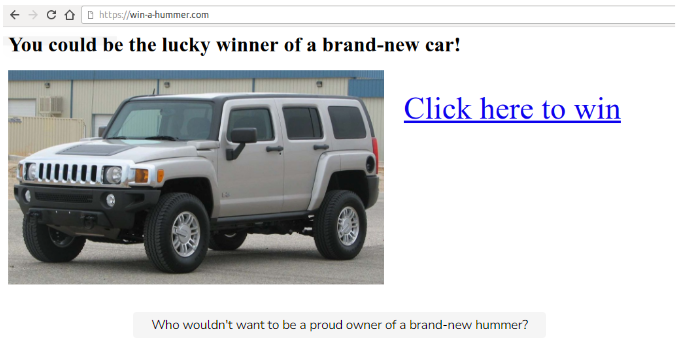
* enable CT verification for the current app for a period of one hour (3600 seconds)
* **enforce** this policy and prevent access to the app if a violation occurs
* send a report to the given URL if a violation occurs

The Certificate Transparency initiative’s goal is to detect erroneously issued or malicious certificates (including rogue Certificate Authorities) earlier, faster, and more precisely than any other method before. By opting-in using the **Expect-CT** header, you can take advantage of this initiative to improve your app’s security posture.

**6.7 X-Frame-Options**

**What is clickjacking?**

Imagine seeing a web page like this on your screen:



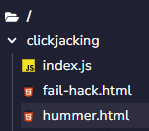
As soon as you click on the link, you realize that all the money in your bank account is gone. What happened?

You were a victim of a clickjacking attack! An attacker directed you to their website, which displays an attractive link to click. Unfortunately, they also embedded an iframe from your-bank.com/transfer?amount=10000000&to=attacker@example.com in the page but hid it by setting its opacity to 0%. Then, instead of clicking on the original page and winning a brand-new hummer, the browser captured a click on the iframe, a dangerous click that confirmed the transfer of money. Most banking systems require you to specify a one-time PIN code to confirm transactions, but your bank hasn’t caught up with the times, and all of your money is gone.

The example is pretty extreme but should help you understand the possible consequences of a [clickjacking attack](https://www.troyhunt.com/clickjack-attack-hidden-threat-right-in/). The user intends to click on a particular link while the browser will trigger a click on the invisible page that’s been embedded as an iframe.

**A runnable example**

I have included an example of this vulnerability in the code below. Run it and see the website. If you run the example and try clicking on the appealing link, you will see the actual click is intercepted by the iframe, which increases its opacity so that’s easier for you to spot the problem.



//Index.js

var qs = require('querystring')

var url = require('url')

var fs = require('fs')

require('http').createServer((req, res) => {

let path = url.parse(req.url).pathname === '/' ? '/hummer.html' : url.parse(req.url).pathname

let query = qs.parse(url.parse(req.url).query)

let headers = {}

if (query.xfo === "on") {

headers['X-Frame-Options'] = "DENY"

}

if (query.csp === "on") {

headers['Content-Security-Policy'] = "frame-ancestors 'none'"

}

res.writeHead(200, headers)

let content = ""

try {

content = fs.readFileSync(\_\_dirname + path)

if (path.endsWith('.html')) {

content = content.toString().replace('\_\_QUERY\_\_', req.url.replace('/', ''))

}

} catch (err) {}

res.end(content)

}).listen(7888)

//fail-hack.html

<html>

<body>

<div>

<h1>

My attack didn't go through :'-(

</h1>

</div>

</body>

</html>

//hammer.html

<html>

<head>

<link rel="icon" href="data:;base64,iVBORw0KGgo=">

<style>

a {

font-size: 3em;

}

#content {

display: flex;

flex-direction: row;

width: 100%;

}

#content div {

width: 50%;

}

iframe {

width: 95%;

height: 95%;

position: absolute;

opacity: 0;

overflow:hidden;

}

</style>

</head>

<body>

<iframe src="/bank.html\_\_QUERY\_\_"></iframe

<div>

<h1>

You could be the lucky winner of a brand-new car!

</h1>

</div>

<div id="content">

<div>

<img src="/hummer.png" width="600" />

</div>

<div>

<a href="/fail-hack">Click here to win!</a>

</div>

</div>

<script type="text/javascript">

var eventMethod = window.addEventListener

? "addEventListener"

: "attachEvent";

var eventer = window[eventMethod];

var messageEvent = eventMethod === "attachEvent"

? "onmessage"

: "message";

eventer(messageEvent, function (e) {

if (e.data === "hijacked" || e.message === "hijacked") {

alert(`You just thought of buying a hummer but instead clicked on a banking website. To let you understand the mechanics of this attack, let me tweak te opacity of the banking page, which is embedded through an iframe`)

console.log(document.querySelector('iframe').style.opacity)

document.querySelector('iframe').style.opacity = 0.1 + document.querySelector('iframe').style.opacity \* 2;

}

});

</script>

</body>

</html>

//bank.html

<html>

<head>

<link rel="icon" href="data:;base64,iVBORw0KGgo=">

<style>

\* {

background: white;

}

a {

font-size: 3em;

}

#content {

display: flex;

flex-direction: row;

width: 100%;

}

#content div {

width: 50%;

}

</style>

</head>

<body>

<div>

<h1>

This is a banking website

</h1>

</div>

<div id="content">

<div>

<img src="/scrooge.jpeg" width="600" />

</div>

<div>

<a onclick="javascript:alert('Money!')">Click here to donate all your money to a random person!</a>

</div>

</div>

<script>

document.querySelector('a').onclick = function () {

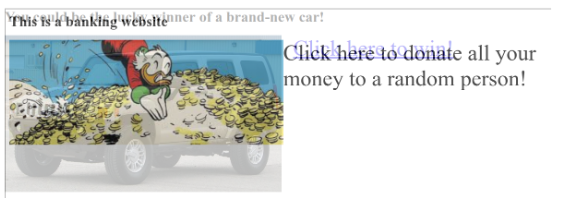
parent.postMessage("hijacked", "\*")

};

</script>

</body>

</html>



Luckily, browsers have come up with a simple solution to the problem, X-Frame-Options (abbr. XFO). XFO lets you decide whether your app can be embedded as an iframe on external websites. Popularized by Internet Explorer 8, XFO was first introduced in 2009 and is still supported by all major browsers. When a browser sees an iframe, it loads it and verifies that its XFO allows its inclusion in the current page before rendering it.

**The supported values are:**

* **DENY**: This web page cannot be embedded anywhere. This is the highest level of protection as it doesn’t allow anyone to embed our content.
* **SAMEORIGIN**: Only pages from the same domain as the current one can embed this page. This means that example.com/embedder can load example.com/embedded so long as its policy is set to SAMEORIGIN. This is a more relaxed policy that allows owners of a particular website to embed their own pages across their application.
* **ALLOW-FROM uri**: Embedding is allowed from the specified URI. We could, for example, let an external, authorized website embed our content by using ALLOW-FROM https://external.com. This is generally used when you intend to allow a third party to embed your content through an iframe.

The supported values are:

DENY: This web page cannot be embedded anywhere. This is the highest level of protection as it doesn’t allow anyone to embed our content.

SAMEORIGIN: Only pages from the same domain as the current one can embed this page. This means that example.com/embedder can load example.com/embedded so long as its policy is set to SAMEORIGIN. This is a more relaxed policy that allows owners of a particular website to embed their own pages across their application.

ALLOW-FROM uri: Embedding is allowed from the specified URI. We could, for example, let an external, authorized website embed our content by using ALLOW-FROM https://external.com. This is generally used when you intend to allow a third party to embed your content through an iframe.

An example HTTP response that includes the strictest XFO policy possible looks like this:

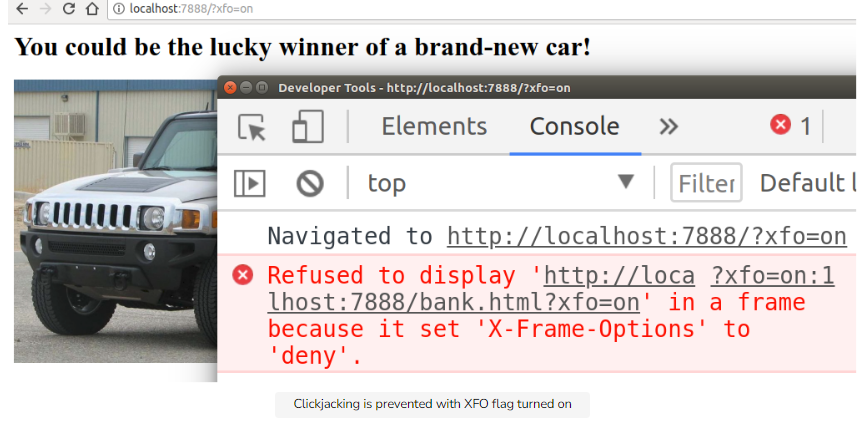
HTTP/1.1 200 OK

Content-Type: application/json

X-Frame-Options: DENY

...

In order to showcase how browsers behave when XFO is enabled, we can simply change the URL of our example to **http://localhost:7888/?xfo=on**. The **xfo=on** parameter tells the server to include **X-Frame-Options: deny** in the response, and we can see how the browser restricts access to the iframe.



XFO was considered the best way to prevent frame-based clickjacking attacks until another header came into play years later, the Content Security Policy.