Security on the web

Websites contain several different types of information. Some of it is non-sensitive, for example the copy shown on the public pages. Some of it is sensitive, for example customer usernames, passwords, and banking information, or internal algorithms and private product information.

Sensitive information needs to be protected, and that is the focus of web security. If that information fell into the wrong hands, it could be used to:

- Put companies at a competitive disadvantage by sharing their information with competitors.
- Disable or hijack their services, again causing serious problems with their operation.
- Put their customer's privacy at risk, making them vulnerable to profiling, targeting, loss of data, identity theft, or even financial loss.

Modern browsers already have several features to protect users' security on the web, but developers also need to use best practices and code carefully to ensure that their websites are secure. Even simple bugs in your code can result in vulnerabilities that bad actors can exploit to steal data and gain unauthorized control over services.

This article provides an introduction to web security, including conceptual information to help you understand website vulnerabilities and practical guides on how to secure them.

Relationship between security and privacy

Security and privacy are distinct yet closely related topics. It is worth knowing the differences between the two and how they relate.

- **Security** is the act of keeping private data and systems protected against unauthorized access. This includes both company (internal) data and user and partner (external) data.
- **Privacy** refers to the act of giving users control over how their data is collected, stored, and used, while also ensuring that it is not used irresponsibly. For example, you should let your users know what data you are collecting from them, the parties with whom it will be shared, and how it will be used. Users must be given a chance to consent to your privacy policy, have access to their data you store, and delete it if they choose to.

Good security is essential for good privacy. You could follow all the advice listed in our Privacy on the web guide, but acting with integrity and having a robust privacy policy are futile if your site is not secure and attackers can just steal data anyway.

Security features provided by browsers

Web browsers follow a strict security model that enforces strong security for content, connections between the browser and the server, and data transportation. This section looks at the features that underpin this model.

Same-origin policy and CORS

Same-origin policy is a fundamental security mechanism of the web that restricts how a document or a script loaded from one origin can interact with a resource from another origin. It helps isolate potentially malicious documents, reducing possible attack vectors.

In general, documents from one origin cannot make requests to other origins. This makes sense because you don't want sites to be able to interfere with one another and access unauthorized data.

However, you might want to relax this restriction in some circumstances; for example, if you have multiple websites that interact with each other, you may allow them to request resources from one another using `fetch()`. This can be permitted using Cross-Origin Resource Sharing (CORS), an HTTP-header-

based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources.

HTTP model for communication

The HTTP protocol is used by web browsers and servers to communicate with one another, request resources, provide responses (for example, providing a requested resource or detailing why a request failed), and provide security features for that communication.

Transport Layer Security (TLS) provides security and privacy by encrypting data during transport over the network and is the technology behind the HTTPS protocol. TLS is good for privacy because it stops third parties from being able to intercept transmitted data and use it maliciously.

All browsers are moving towards requiring HTTPS by default; this is practically the case already because you can't do much on the web without this protocol.

Related topics:

Transport layer security (TLS)

> The TLS protocol is the standard for enabling two networked applications or devices to exchange information privately and robustly. Applications that use TLS can choose their security parameters, which can have a substantial impact on the security and reliability of data.

HTTP Strict-Transport-Security

> The `Strict-Transport-Security` HTTP header lets a website specify that it may only be accessed using HTTPS.

Certificate Transparency

> Certificate Transparency (CT) is an open framework designed to protect against and monitor for certificate misissuance. Newly issued certificates are 'logged' to publicly run, often independent CT logs. These provide append-

only, cryptographically assured records of issued TLS certificates.

Mixed content

An HTTPS page that includes content fetched using cleartext HTTP is called a **mixed content** page. Pages like this are only partially encrypted, leaving the unencrypted content accessible to sniffers and man-in-the-middle attackers.

Weak signature algorithms

The strength of the hash algorithm used in signing a digital certificate is a critical element of the security of the certificate. Some signature algorithms are known to be weak, and should be avoided when appropriate.

Secure contexts and feature permissions

Browsers control the usage of "powerful features" in different ways. These "powerful features" include generating system notifications on a website, using a user's webcam to get access to a media stream, manipulating the system GPU, and using web payments. If a site could just use the APIs that control such features without restriction, malicious developers could attempt to do the following:

- Annoy users with unneeded notifications and other UI features.
- Turn their webcam on without warning to spy on them.
- Clog up their browser/system to create Denial of Service (DoS) attacks.
- Steal data or money.

These "powerful features" are controlled in the following ways:

- Usage of such features is permitted only in secure contexts. A secure context is a `window` or a `worker` for which there is reasonable confidence that the content has been delivered securely (via HTTPS/TLS). In a secure context, the potential for communication with contexts that are **not** secure is limited. Secure contexts also help to prevent man-in-the-middle

attackers from accessing powerful features. To see a list of web platform features available only in secure contexts, see Features restricted to secure contexts.

- The usage of these features is gated behind a system of user permissions: users have to explicitly opt-in to providing access to such features, meaning that they can't be used automatically. User permission requests happen automatically, and you can query the state of an API permission by using the Permissions API.
- Several other browser features can be used only in response to a user action such as clicking a button, meaning that they need to be invoked from inside an appropriate event handler. This is called **transient activation**. See Features gated by user activation for more information.

High-level security considerations

There are many aspects of web security that need to be thought about on the server- and client-side. This section focuses mainly on client-side security considerations. You can find a useful summary of security from a server-side perspective, which also includes descriptions of common attacks to watch out for, at Website security (part of our Server-side website programming learning module).

Store client-side data responsibly

Handling data responsibly is largely concerned with cutting down on third-party cookie usage and being careful about the data you store and share with them. Traditionally, web developers have used cookies to store all kinds of data, and it has been easy for attackers to exploit this tendency. As a result, browsers have started to limit what you can do with cross-site cookies, with the aim of removing access to them altogether in the future.

You should prepare for the removal of cross-site cookies by limiting the amount of tracking activities you rely on and/or by implementing the persistence of the desired information in other ways. See Transitioning from third-party cookies and Replacing third-party cookies for more information.

Protect user identity and manage logins

When implementing a secure solution that involves data collection, particularly if the data is sensitive such as log-in credentials, it makes sense to use a reputable solution. For example, any respectable server-side framework will have built-in features to protect against common vulnerabilities. You could also consider using a specialized product for your purpose, for example an identity provider solution or a secure online survey provider.

If you want to roll your own solution for collecting user data, make sure you understand all aspects and requirements. Hire an experienced server-side developer and/or security engineer to implement the system, and ensure it is tested thoroughly. Use multi-factor authentication (MFA) to provide better protection. Consider using a dedicated API such as Web Authentication or Federated Credential Management to streamline the client-side of the app.

Here are some other tips for providing secure logins:

- When collecting user login information, enforce strong passwords so that your user's account details cannot be easily guessed. Weak passwords are one of the main causes of security breaches. In addition, encourage your users to use a password manager so that they can use more complex passwords, don't need to worry about remembering them, and won't create a security risk by writing them down. See also our article on Insecure passwords.
- You should also educate your users about **phishing**. Phishing is the act of sending a message to a user (for example, an email or an SMS) containing a link to a site that looks like a site they use every day but isn't. The link is accompanied by a message designed to trick users into entering their username and password on the site so it can be stolen and then used by an attacker for malicious purposes.

> **Note:** Some phishing sites can be very sophisticated and hard to distinguish from a real website. You should therefore educate your

> users to not trust random links in emails and SMS messages. If they receive a message along the lines of "Urgent, you need to log in now to resolve an issue", they should go to the site directly in a new tab and try logging in directly rather than clicking the link in the message. Or they could phone or email you to discuss the message they received.

- Protect against brute force attacks on login pages with rate limiting, account lockouts after a certain number of unsuccessful attempts, and CAPTCHA challenges .
- Manage user login sessions with unique session IDs , and automatically log out users after periods of inactivity.

Don't include sensitive data in URL query strings

As a general rule, you shouldn't include sensitive data in URL query strings because if a third party intercepts the URL (for example, via the `Referer` HTTP header), they could steal that information. Even more serious is the fact that these URLs can be indexed by public web crawlers, HTTP proxies, and archiving tools such as the internet archive , meaning that your sensitive data could persist on publicly accessible resources.

Use `POST` requests rather than `GET` requests to avoid these issues. Our article Referer header policy: Privacy and security concerns describes in more detail the privacy and security risks associated with the `Referer` header, and offers advice on mitigating those risks.

> **Note:** Steering away from transmitting sensitive data in URLs via `GET` requests can also help protect against cross-site request forgery and replay attacks .

Enforce usage policies

Consider using web platform features like Content Security Policy (CSP) and

Permissions Policy to enforce a set of feature and resource usage rules on your website that make it harder to introduce vulnerabilities.

CSP allows you to add a layer of security by, for example, allowing images or scripts to be loaded only from specific trusted origins. This helps to detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks. These attacks involve a range of malicious activities, including data theft, site defacement, and distribution of malware.

Permissions policy works in a similar way, except that it is more concerned with allowing or blocking access to specific "powerful features" (as mentioned earlier).

> **Note:** Such policies are very useful to help keep sites secure, especially when you are using a lot of third-party code on your site. However, keep in mind that if you block usage of a feature that a third-party script relies on to work, you may end up breaking your site's functionality.

Maintain data integrity

Following on from the previous section, when you allow feature and resource usage on your site, you should try to ensure that resources have not been tampered with.

Related topics:

Subresource integrity

> **Subresource Integrity** (SRI) is a security feature that enables browsers to verify that resources they fetch (for example, from a CDN) are delivered without unexpected manipulation. It works by allowing you to provide a cryptographic hash that a fetched resource must match.

HTTP Access-Control-Allow-Origin

The **`Access-Control-Allow-Origin`** response header indicates whether the response can be shared with requesting code from the given origin.

HTTP X-Content-Type-Options

The **`X-Content-Type-Options`** response header is a marker used by the server to indicate that the MIME types advertised in the `Content-Type` headers should not be changed and must be followed. This header is a way to opt out of MIME type sniffing, or, in other words, to specify that the MIME types are deliberately configured.

Sanitize form input

As a general rule, don't trust anything that users enter into forms. Filling out forms online is complicated and tedious, and it is easy for users to enter incorrect data or data in the wrong format. In addition, malicious folks are skilled in the art of entering specific strings of executable code into form fields (for example, SQL or JavaScript). If you're not careful about handling such inputs, they could either execute harmful code on your site or delete your databases. See SQL injection for a good example of how this could happen.

To protect against this, you should thoroughly sanitize data entered into your forms:

- You should implement client-side validation to inform users when they have entered data in the wrong format. You can do this using built-in HTML form validation features, or you can write your own validation code. See Client-side form validation for more information.
- You should use output encoding when displaying user input in an application UI to safely display data exactly as a user typed it in and avoid it being executed as code. See Output encoding for more information.

You can't rely on client-side validation alone for security — it should be combined with server-side validation. Client-side validation enhances the user experience by providing instant validation feedback without having to wait for a round trip to the server. However, client-side validation is easy for a malicious

party to bypass (for example, by turning off JavaScript in the browser to bypass JavaScript-based validation).

Any reputable server-side framework will provide functionality for validating form submissions. In addition, a common best practice is to escape any special characters that form part of executable syntax, thereby making any entered code no longer executable and treated as plain text.

Protect against clickjacking

In a clickjacking attack, a user is fooled into clicking a UI element that performs an action different from what the user expects, often resulting in the user's confidential information being passed to a malicious third party. This risk is inherent in embedded third-party content, so make sure you trust what is being embedded into your site. Additionally, be aware that clickjacking can be combined with phishing techniques. You can read about phishing in the previous section Protect user identity and manage logins.

The following features can help guard against clickjacking:

HTTP X-Frame-Options

The **`X-Frame-Options`** HTTP response header can be used to indicate whether a browser should be allowed to render a page in a `<frame>`, `<iframe>`, `<embed>` or `<object>`. Sites can use this to avoid clickjacking attacks, by ensuring that their content is not embedded into other sites.

CSP: frame-ancestors

The HTTP `Content-Security-Policy` (CSP) **`frame-ancestors`** directive specifies valid parents that may embed a page using `<frame>`, `<iframe>`, `<object>`, or `<embed>`.

Practical security implementation guides

To get comprehensive instructions for implementing security features effectively on websites and to ensure you're following best practices, see our set of Practical security implementation guides.

Some of these guides are directly related to the HTTP Observatory tool. Observatory performs security audits on a website and provides a grade and score along with recommendations for fixing the security issues it finds. These guides explain how to resolve issues surfaced by the MDN Observatory tests: the tool links to the relevant guide for each issue, helping guide you towards an effective resolution. Interestingly, Mozilla's internal developer teams use this guidance when implementing websites to ensure that security best practices are applied.

See also

- Privacy on the web
- Learn: Website security
- Mozilla Security Blog
- OWASP Cheat Sheet series