How security across HTTP can ensured

Can HTTP be made more safe?

Yes it's possible to do it only using HTTP. With HTTP security headers you can make it more safe even with only HTTP.
To know more about security headers you can look into owasp's cheat sheet for it.

You can also do something with the message alone like Obscuring or Integrity check. Although these are not alternatives to actual encrypting it does achieve a bit of security. stackoverflow encode string As the goal of the project is that everybody can connect to the platform and make some bots for it some security measures are not reasonable without compromising security in itself such as given the key to the actor. An example of that being the Fernet key.stackoverflow encode string

Is it feasible to ensure that everything comes across HTTPS?

Yes, using Let's encrypt you can generate free certificates. But a couple of downsides of it are that the certificates are only valid for 90 days, so it's possible that you will have a larger percentage of downtime when compared to paid certificates following SLL certificate StackExchange and Nutty about hosting.

With python the requests library has a parameter to include the certificate see the following

```
#explicitly verify
import requests

response = requests.get("https://api-management-example/run", verify=True)

#Self Signed Certificate
import requests

response = requests.get("https://api-management-example/run", verify="/path/to/local/certificate/file/")
```

Source: https://stackoverflow.com/questions/70052068/python-requests-library-how-to-ensure-https-requests

What is the breakdown of cost between HTTP and HTTPS?

HTTPS is secure HTTP with Let's encrypt you can have a free certificates so you can do HTTPS.

Can we be secure even if new actors are going to be made from users outside of the project?

This is debateable and depends on what the platform allows on which actor can access what. As the platform and FediVerse is mostly/completely open-source as such highly sensitive information can't be allowed to happen inside of it or it's not on the open-source side but privately. An example of a FediVerse platform is Mastodon as in the article of DEV.to says it depends on the server if it has a different privacy policy. Also only trusted individuals should have access to admin rights on the network/server, but because everybody can create their own server you need to see if you can trust the owner/entity.

As such I recommend to have at least a rule on that only the trusted individuals can let the bot be accepted on the platform/server with a code/key or such. After that regular users should be informed who the owner is and that questions with sensitive data can be compromised if they ask it on the platform with an owner that seems not completely trustworthy or as it is decentralized that the bot can be from another server but have a token from theirs.

An example of such key/code is the openAI key which allows developers to access OpenAI’s services and resources.

What needs to happen between communication of platform to actor?

It depends on the use case but likely there will be no need to have real-time data exchange and can wait for a response from the platform I recommend the HTTPS instead of WSS.

What needs to happen between communication of actor to actor?

First it depends on the use case but the most efficient way of communicating between users is WSS as it's a more secure connection of the websocket and although you can also use HTTPS for it as its not bidirectional communication over a single TCP connection, there will be delays so it doesn't allow real-time data exchange between clients and servers. HTTP.dev WSS HTTP.dev HTTPS

Other than that WSS means that it goes over HTTPS so it can only happen with a secure connection. WSS on HTTPS

other connection types

- **WS (WebSocket)**: A real-time communication protocol that enables bidirectional data exchange between a client and a server over a single, long-lived connection.
- **WSS (WebSocket Secure)**: The secure version of WebSocket, which uses encryption (usually over HTTPS) to protect data transmitted between client and server.
- **FTP (File Transfer Protocol)**: A standard network protocol used for transferring files from one host to another over a TCP-based network, typically used for uploading and downloading files.
- **SFTP (SSH File Transfer Protocol)**: An extension of SSH (Secure Shell) that provides secure file transfer capabilities, allowing users to transfer files securely over an encrypted channel.
- **SMTP (Simple Mail Transfer Protocol)**: The standard protocol for sending email messages between servers, used for sending outgoing emails.
- **IMAP (Internet Message Access Protocol)**: A protocol used by email clients to retrieve messages from a mail server, allowing users to manage their email messages remotely.
- **POP3 (Post Office Protocol version 3)**: Another email retrieval protocol that allows users to download email messages from a mail server to their local device.
- **LDAP (Lightweight Directory Access Protocol)**: A protocol for accessing and maintaining distributed directory information services, commonly used for querying and managing user accounts and other directory data.

HTTP security headers

This section heavily relies on owasp's cheat sheet for HTTP security headers. I also use mozilla for further information.

X-Frame-Option

This sets if the browser should be allowed to render a pag in a `<frame>`,`<iframe>`,`<embed>` or `<object>`. This is useful to avoid clickjacking attacks, by ensuring that the websites content is not embedded into other sites.
Content Security Policy (CSP) frame-ancestors directive obsoletes X-Frame-Options for supporting browsers (source).

This header is also only useful when the HTTP response where it is included has something to interact with (e.g. links, buttons). If the HTTP response is a redirect or an API returning JSON data, does this not provide any security.

It's recommended to use Content Security Policy (CSP) frame-ancestors directive if possible. When not possible use `X-Frame-Options: DENY`.

X-XSS-Protection

The HTTP X-XSS-Protection response header was a feature of Internet Explorer, Chrome and Safari that stopped pages from loading when they detected reflected cross-site scripting (XSS) attacks. These protections are largely unnecessary in modern browsers when sites implement a strong Content-Security-Policy that disables the use of inline JavaScript ('unsafe-inline').

Even though this header can protect users of older web browsers that don't yet support CSP, in some cases, this header can create XSS vulnerabilities in otherwise safe websites. See the following example:

```
<script>
  var productionMode = true;
</script>
<!-- [...] -->
<script>
  if (!window.productionMode) {
    // Some vulnerable debug code
  }
</script>
```

This piece of code will be completely safe if the browser doesn't perform XSS filtering. However, if it does and the search query is `?something=%3Cscript%3Evar%20productionMode%20%3D%20true%3B%3C%2Fscript%3E`, the browser might execute the scripts in the page ignoring `<script>var productionMode = true;</script>` (thinking the server included it in the response because it was in the URI), causing window.productionMode to be evaluated to undefined and executing the unsafe debug code.

This means that if you do not need to support legacy browsers, it is recommended that you use Content-Security-Policy without allowing unsafe-inline scripts instead. Furthermore do not set this header or explicitly turn it off with `X-XSS-Protection: 0` or `X-XSS-Protection: 1; mode=block`.

X-Content-Type-Options

The X-Content-Type-Options response HTTP header is used by the server to indicate to the browsers that the MIME types advertised in the Content-Type headers should be followed and not guessed.

This header is used to block browsers' MIME type sniffing, which can transform non-executable MIME types into executable MIME types (MIME Confusion Attacks).

It's recommended to set the Content-Type header correctly throughout the site. Like: `X-Content-Type-Options: nosniff`.

Referrer-Policy

The Referrer-Policy HTTP header controls how much referrer information (sent via the Referer header) should be included with requests. Aside from the HTTP header, you can also set this policy in HTML.

Referrer policy has been supported by browsers since 2014. Today, the default behavior in modern browsers is to no longer send all referrer information (origin, path, and query string) to the same site but to only send the origin to other sites. However, since not all users may be using the latest browser its suggested that you force this behavior by sending this header on all responses. (E.g. `Referrer-Policy: strict-origin-when-cross-origin`)

Content-Type

The Content-Type representation header is used to indicate the original media type of the resource (before any content encoding is applied for sending). If not set correctly, the resource (e.g. an image) may be interpreted as HTML, making XSS vulnerabilities possible.

Although it is recommended to always set the Content-Type header correctly, it would constitute a vulnerability only if the content is intended to be rendered by the client and the resource is untrusted (provided or modified by a user).

Recommended is to set the `charset` attribute to prevent XSS in HTML pages. While `text/html` or other possible MIME types needs to be included to set the Content-Type. The following example shows how you can send a short text message `Content-Type: text/html; charset=UTF-8`.

Set-Cookie

The Set-Cookie HTTP response header is used to send a cookie from the server to the user agent, so the user agent can send it back to the server later. To send multiple cookies, multiple Set-Cookie headers should be sent in the same response.

This is not a security header per se, but its security attributes are crucial.

The recommendation for this header is to read the Session Management Cheat Sheet for a detailed explanation on cookie configuration options.

Strict-Transport-Security (HSTS)

The HTTP Strict-Transport-Security response header (often abbreviated as HSTS) lets a website tell browsers that it should only be accessed using HTTPS, instead of using HTTP.

You should note that you need to read carefully how this header works before using it. If the HSTS header is misconfigured or if there is a problem with the SSL/TLS certificate being used, legitimate users might be unable to access the website. For example, if the HSTS header is set to a very long duration and the SSL/TLS certificate expires or is revoked, legitimate users might be unable to access the website until the HSTS header duration has expired.

An example of how the header can be set is `Strict-Transport-Security: max-age=63072000; includeSubDomains; preload`.
For more information about this you can go to the following cheat sheet: HTTP Strict Transport Security Cheat Sheet

Expect-CT

The Expect-CT header lets sites opt-in to reporting of Certificate Transparency (CT) requirements. Given that mainstream clients now require CT qualification, the only remaining value is reporting such occurrences to the nominated report-uri value in the header. The header is now less about enforcement and more about detection/reporting.

Not recommend to use. Mozilla even recommends removing it.

Content-Security-Policy (CSP)

Content Security Policy (CSP) is a security feature that is used to specify the origin of content that is allowed to be loaded on a website or in a web applications. It is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement to distribution of malware.

It should be noted that this header is relevant to be applied in pages which can load and interpret scripts and code, but might be meaningless in the response of a REST API that returns content that is not going to be rendered.

Content Security Policy is complex to configure and maintain. That is why it's recommend that you look through the following cheat sheet: Content Security Policy Cheat Sheet.

Access-Control-Allow-Origin

If you don't use this header, your site is protected by default by the Same Origin Policy (SOP). What this header does is relax this control in specified circumstances.

The Access-Control-Allow-Origin is a CORS (cross-origin resource sharing) header. This header indicates whether the response it is related to can be shared with requesting code from the given origin. In other words, if siteA requests a resource from siteB, siteB should indicate in its Access-Control-Allow-Origin header that siteA is allowed to fetch that resource, if not, the access is blocked due to Same Origin Policy (SOP).

If you use it, set specific origins instead of *. Checkout Access-Control-Allow-Origin for details.
E.g. `Access-Control-Allow-Origin: https://yoursite.com`

Cross-Origin-Opener-Policy (COOP)

The HTTP Cross-Origin-Opener-Policy (COOP) response header allows you to ensure a top-level document does not share a browsing context group with cross-origin documents.

This header works together with Cross-Origin-Embedder-Policy (COEP) and Cross-Origin-Resource-Policy (CORP) explained below.

This mechanism protects against attacks like Spectre which can cross the security boundary established by Same Origin Policy (SOP) for resources in the same browsing context group.

As this headers are very related to browsers, it may not make sense to be applied to REST APIs or clients that are not browsers.

It's Recommended to Isolate the browsing context exclusively to same-origin documents.
E.g. `HTTP Cross-Origin-Opener-Policy: same-origin`

Cross-Origin-Embedder-Policy (COEP)

he HTTP Cross-Origin-Embedder-Policy (COEP) response header prevents a document from loading any cross-origin resources that don't explicitly grant the document permission (using CORP or CORS).

NOTE: Enabling this will block cross-origin resources not configured correctly from loading

Recommendation for this is a document can only load resources from the same origin, or resources explicitly marked as loadable from another origin.

E.g. `Cross-Origin-Embedder-Policy: require-corp`

NOTE: you can bypass it for specific resources by adding the crossorigin attribute: `<img src="https://thirdparty.com/img.png" crossorigin>`

Cross-Origin-Resource-Policy (CORP)

The Cross-Origin-Resource-Policy (CORP) header allows you to control the set of origins that are empowered to include a resource. It is a robust defense against attacks like Spectre, as it allows browsers to block a given response before it enters an attacker's process.

You should limit current resource loading to the site and sub-domains only.
This can be done like the following: `Cross-Origin-Resource-Policy: same-site`

Permissions-Policy (formerly Feature-Policy)

Permissions-Policy allows you to control which origins can use which browser features, both in the top-level page and in embedded frames. For every feature controlled by Feature Policy, the feature is only enabled in the current document or frame if its origin matches the allowed list of origins. This means that you can configure your site to never allow the camera or microphone to be activated. This prevents that an injection, for example an XSS, enables the camera, the microphone, or other browser feature.

More information can be found at Permissions-Policy

The recommendation for this header is to set it and disable all the features that your site does not need or allow them only to the authorized domains: `Permissions-Policy: geolocation=(), camera=(), microphone=()`
NOTE: This example is disabling geolocation, camera, and microphone for all domains.

An even bigger example is: `accelerometer=(),ambient-light-sensor=(),autoplay=(),battery=(),camera=(),display-capture=(),document-domain=(),encrypted-media=(),fullscreen=(),gamepad=(),geolocation=(),gyroscope=(),layout-animations=(self),legacy-image-formats=(self),magnetometer=(),microphone=(),midi=(),oversized-images=(self),payment=(),picture-in-picture=(),publickey-credentials-get=(),speaker-selection=(),sync-xhr=(self),unoptimized-images=(self),unsized-media=(self),usb=(),screen-wake-lock=(),web-share=(),xr-spatial-tracking=()`

FLoC (Federated Learning of Cohorts)

FLoC is a method proposed by Google in 2021 to deliver interest-based advertisements to groups of users ("cohorts"). The Electronic Frontier Foundation, Mozilla, and others believe FLoC does not do enough to protect users' privacy.

Although not a direct recommendation if you want your site not be included in the user's list of sites for cohort calculation. Then by sending this HTTP header: `Permissions-Policy: interest-cohort=()`, you will not be included.

If you are interested in partaking into this you can check out the blog from google.

Server

The Server header describes the software used by the origin server that handled the request — that is, the server that generated the response.

This is not a security header, but how it is used is relevant for security.

Recommended is to remove this header or set non-informative values. An example of non-informative values: `Server: webserver`. An example of an actual is `Server: Apache/2.4.1 (Unix)` this means that the server is running Apache version 2.4.1 on a Unix system.

Bad actors can use this to exploit known security holes.
NOTE: Remember that attackers have other means of fingerprinting the server technology.

X-Powered-By

The X-Powered-By header describes the technologies used by the webserver. This information exposes the server to attackers. Using the information in this header, attackers can find vulnerabilities easier.

Recommended to remove all X-Powered-By headers.
NOTE: Remember that attackers have other means of fingerprinting your tech stack.

X-AspNet-Version

Provides information about the .NET version.

As it is not important for the platform or other components that are not actors. There will be no recommendation if you want to see it you can go to owasp HTTP Headers Cheat Sheet

X-AspNetMvc-Version

Provides information about the .NET version. But then when it's probally a MVC project.

As it is not important for the platform or other components that are not actors. There will be no recommendation if you want to see it you can go to owasp HTTP Headers Cheat Sheet

X-DNS-Prefetch-Control

The X-DNS-Prefetch-Control HTTP response header controls DNS prefetching, a feature by which browsers proactively perform domain name resolution on both links that the user may choose to follow as well as URLs for items referenced by the document, including images, CSS, JavaScript, and so forth.

The default behavior of browsers is to perform DNS caching which is good for most websites. But if you do not control links on your website, you might want to set off as a value to disable DNS prefetch to avoid leaking information to those domains.

This can be done like the following: `X-DNS-Prefetch-Control: off`
NOTE: Do not rely in this functionality for anything production sensitive: it is not standard or fully supported and implementation may vary among browsers.

Public-Key-Pins (HPKP)

The HTTP Public-Key-Pins response header is used to associate a specific cryptographic public key with a certain web server to decrease the risk of MITM attacks with forged certificates.

This header is deprecated and should not be used anymore.

WebSocket Secure (WSS)

WebSocket Secure indicates that traffic over that connection is to be protected via TLS. rfc6455 WSS
You can relate it to HTTP and HTTPS where it also goes over TLS with HTTPS. CloudFlare

Websocket's in general is to enable bi-directional communication without abusing the HTTPS for instant messaging and gaming applications. Which resulted in problems regarding where: rfc6455

- The wire protocol has a high overhead, with each client-to-server message having an HTTP header.
- The client-side script is forced to maintain a mapping from the outgoing connections to the incoming connection to track replies.

The WebSocket Protocol is designed on the principle that there should be minimal framing (the only framing that exists is to make the protocol frame-based instead of stream-based and to support a distinction between Unicode text and binary frames). rfc6455
It is expected that metadata would be layered on top of WebSocket by the application layer, in the same way that metadata is layered on top of TCP by the application layer (e.g., HTTP). rfc6455

The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. rfc6455

The wss scheme is not only more secure but more reliable because that some proxy servers do not recognize the WebSocket-specific scheme. Consequently, an intermediate may drop the request or terminate the connection. However, because wss is encrypted using SSL/TLS, there is no visibility as it passes through intermediaries, and because they can't read the data, it is passed through without further inspection. HTTP.dev WSS

Security issues around WSS

You should consider that WebSockets are not restrained by the same-origin policy, an attacker can thus easily initiate a WebSocket request to the WSS endpoint URL. Due to the fact that this request is a regular https request, browsers send the cookies and HTTP-Authentication headers along, even cross-site. christian-schneider

This attack can be extended from a write-only CSRF attack to a full read/write communication with a WebSocket service by physically establishing a new WebSocket connection with the service under the same authentication data as the victim. You can also call it a Cross-Site WebSocket Hijacking (CSWSH). christian-schneider

Effectively, this allows the attacker to read for example the victim's stock portfolio updates pushed via the WebSocket connection and update the portfolio by issuing write requests via the WebSocket connection. This is possible due to the fact that the server's WebSocket code relies on the session authentication data (cookies or HTTP-Authentication) sent along from the browser during the WebSocket handshake/upgrade phase. christian-schneider

You can secure it via the following methods: christian-schneider

- Check the Origin header of the WebSocket handshake request on the server, since that header was designed to protect the server against attacker-initiated cross-site connections of victim browsers!
- Use session-individual random tokens (like CSRF-Tokens) on the handshake request and verify them on the server.

For further security considerations you can look into the rfc6455 security considerations.

Example of using WSS

As an example of using WSS in Javascript you can see the following block.

```
const connection = new WebSocket('wss://files.example.re/');
```

For an example in python you can look into the site of CodingPointer for a tutorial. Or you can see from the github repository of Aymeric Augustin, websocket example on how it can be done.

LDAP

This section is here because it is a set of open protocols used to access centrally stored information over a network. LDAP is commonly used for allowing users to easily access contact information for other users. Which you can see as a virtual phone directory. redhat ldap

You possibly want to use this to consolidate information into a central repository. LDAP also supports a number of back-end databases in which to store directories. This allows administrators the flexibility to deploy the database best suited for the type of information the server is to disseminate. redhat ldap

You can use it for SSO (single sign-on), where an existing account in a directory is used to authenticate a user to an application or service. OAuth2 and SAML are the more well known versions of SSO. builtin
In addition to authentication, we can use LDAP for informational purposes to query the directory for user attributes like title or department information, group membership, employee ID, access control lists and so on. builtin
Depending on the level of access a person has to the LDAP directory, it's also possible to perform updates to the directory. We can use LDAP to add, remove or modify directory entries. builtin

Some example of use cases can be:

- Centralized Authentication
- Email Services
- Network Services and Devices (to help manage network devices such as routers, switches, and firewalls.)
- Certificate Management
- Cloud Services Integration (also user authentication)

Sources

(only visible on markdown)