I see many websites that are behind a web application firewall (WAF) that don't have the security settings in their HTTP header configured. Solely relying on the WAF is a false sense of security because of human error. All it takes is a bad update or a misconfiguration to leave your website vulnerable to cross-site scripting, cross-frame scripting, MIME sniffing, HSTS, and data leakage through referrers. Ideally, you'll also want to configure a content security policy, but using the previously mentioned settings are good enough for most circumstances.

These settings are called response headers which are sent after the GET request from the browser. Response headers are from the web server sent to the browser instructing the browser how to handle the data that it's going to receive.

You can change your HTTP headers if you're using WordPress by installing a plugin called HTTPS Headers that is user-friendly. If using a plugin isn't an option you can find your HTTP header setting in the ".htaccess" file in the "public_html" folder in File Manager. The settings will be placed at the end of the ".htaccess" file.

Apache example:

```
# BEGIN HttpHeaders
<IfModule mod_headers.c>
  Header always set X-Content-Type-Options "nosniff"
  <FilesMatch "\.(php|html)$">
    Header set X-Frame-Options "SAMEORIGIN"
    Header set X-XSS-Protection "1; mode=block"
    Header set X-DNS-Prefetch-Control "on"
    Header set Connection "keep-alive"
    Header set Cache-Control "must-revalidate, no-cache, public, max-age=604800"
    Header set Strict-Transport-Security "max-age=2592000; preload"
    Header set Referrer-Policy "no-referrer-when-downgrade"
    Header append Vary "Accept-Encoding"
  </FilesMatch>
</IfModule>
# END HttpHeaders
```

Let's go through the meaning of the five settings and the configuration options.

- X-XSS-Protection: this setting strips out (escapes) tags or prevents the page from rendering for Javascript and HTML from user input in the search box and forms. Without the appropriate tags, the web server see's the input as plain text and execution doesn't happen.
    - Options
        - 0: Disables filtering
        - 1: Enables filtering; removes tags

        - 1;mode=block: Enables filtering; prevents the page from rendering
        - 1;report=<reporting-URI>: Only works with Chrome and must have the report-uri directive from a content security policy
    - Examples
        - PHP: header("X-XSS-Protection: 1; mode=block");
        - Apache
            - <IfModule mod_headers.c>
              Header set X-XSS-Protection "1; mode=block"
              </IfModule>
        - Nginx: add_header "X-XSS-Protection" "1; mode=block";
    - Compatibility
        - Desktop: Chrome, Internet Explorer, Edge, Opera, Safari (no support by Firefox)
        - Mobile
            - Android: Android webview, Chrome, Edge, Opera, Safari
            - Apple: Chrome, Edge, Opera, Safari
            - No support in Firefox for Android nor Apple
- X-Frame-Options: this setting prohibits content from one page being rendered on another page in <frame>, <iframe>, and <object> tags. This prevents clickjacking where links are hidden to cause a user to perform an action on another page, e.g., clicking an ad to earn revenue for another site owner which is fraudulent.
    - Options
        - Deny: The page can't be displayed in a frame
        - SAMEORIGIN: The page can be displayed in a frame only if it's on the origin server as the page itself
        - ALLOW-FROM: The page can be displayed in a frame on the origin listed in the specified URI
    - Examples
        - Apache
            - Header always set X-Frame-Options SAMEORIGIN
            - Header set X-Frame-Options DENY
            - Header set X-Frame-Options "ALLOW-FROM https://example.com/"
        - nginx
            - add_header X-Frame-Options SAMEORIGIN;
        - IIS (web.config)
            - <system.webServer>
              ...<httpProtocol>
              <customHeaders>
              <add name="X-Frame-Options"

```
value="SAMEORIGIN" />
</customHeaders>
</httpProtocol>...
</system.webServer>
```

    - Compatibility
        - Desktop
            - SAMEORIGIN: Chrome, Edge, Firefox, Internet Explorer, Opera, Safari
            - ALLOW-FROM: Firefox, Internet Explorer, Safari (no Chrome support; Edge and Opera are unknown)
            - DENY: Chrome, Edge, Firefox, Internet Explorer, Opera, Safari
        - Mobile
            - DENY: Chrome, Edge, Firefox, Internet Explorer, Opera, Safari (Android and Apple)
            - SAMEORIGIN
                - Android: Android webview, Chrome, Opera (Edge, Firefox, Safari unknown)
                - Apple: Chrome (all others unknown)
            - ALLOW-FROM: unknown for all browsers
- X-Content-Type-Options: prohibits text and audio from being changed from a non-executable file to an executable file that can be used to run malicious code. Images aren't protected by this setting in any browser.
    - Option and Example
        - X-Content-Type-Options: nosniff
- HTTP Strict-Transport-Security (HSTS): tells browsers that the website can be accessed only by HTTPS. If your website has HTTP to HTTPS redirection there may be some communication that is unencrypted before the redirection which creates a man-in-the-middle opportunity that would allow an attacker to injection a header that redirects the visitor to a malicious website.
    - Options
        - max-age=<expire-time>: time in seconds that the browser should cache that the site is to be accessed only by HTTPS
        - includeSubDomains: this is an optional option that applies this setting to the site's subdomains
        - preload: another optional option that is facilitated by Google. Submitting your domain to Google's preload services assures that browsers will never connect to your website over HTTP. More information about preload can be found here.
    - Examples
        - Strict-Transport-Security: max-age=<expire-time>

        - Strict-Transport-Security: max-age=<expire-time>; includeSubDomains
        - Strict-Transport-Security: max-age=<expire-time>; preload
- Referrer-Policy: determines which referrer information (URL) should be included with requests.
    - Options
        - no-referrer: the referrer header will be left out
        - no-referrer-when-downgrade: this is the default behavior; the origin is sent as the referrer when it's HTTPS to HTTPS, but not HTTPS to HTTP
        - origin: only send the origin as the referrer in all cases (https://domain.com/page.html will send https://domain.com)
        - origin-when-cross-origin: sends a full URL when a same-origin request is made but sends only the origin for other requests
        - same-origin: referrer will be sent for same-site origin requests; cross-origin requests will have no referrer information
        - strict-origin: sends only the origin URL as the referrer when HTTPS to HTTPS, but not HTTPS to HTTPS
        - strict-origin-when-cross-origin: sends a full URL when a same-origin request is made and send only the origin when HTTPS to HTTPS, but no header when HTTPS to HTTP
        - unsafe-url: send the full URL when same-origin and cross-origin requests are made
    - Syntax
        - Referrer-Policy: no-referrer
        - Referrer-Policy: no-referrer-when-downgrade
        - Referrer-Policy: origin
        - Referrer-Policy: origin-when-cross-origin
        - Referrer-Policy: same-origin
        - Referrer-Policy: strict-origin
        - Referrer-Policy: strict-origin-when-cross-origin
        - Referrer-Policy: unsafe-url
    - Examples
        - no-referrer: domain.com/page.html to any domain or path - no referrer
        - no-referrer-when-downgrade
            - https://domain.com/page.html to https://domain.com/otherpage.html - https://domain.com/page.html is sent

        - https://domain.com/page.html to **http**://domain.com - no referrer
    - origin: https://domain.com/page.html to any domain or path - https://domain.com/page.html is sent
    - origin-when-cross-origin
        - https://domain.com/page.html to https://domain.com/otherpage.html - https://domain.com/page.html is sent
        - https://domain.com/page.html to domain.net - https://domain.com is sent
        - https://domain.com/page.html to **http**://domain.net - https://domain.com is sent
    - same-origin
        - https://domain.com/page.html to https://domain.com/otherpage.html – https://domain.com/page.html is sent
        - https://domain.com/page.html to https://domain.net – no referrer is sent
    - strict-origin
        - https://domain.com/page.html to https://domain.net - https://domain.com is sent
        - https://domain.com/page.html to **http**://domain.net – no referrer is sent
        - http://domain.com/page.html to any domain or path – http://domain.com is sent
    - strict-origin-when-cross-origin
        - https://domain.com/page.html to https://domain.com/otherpage.html - https://domain.com/page.html is sent
        - https://domain.com/page.html to http://domain.net – no referrer is sent
        - https://domain.com/page.html to https://domain.net – https://domain.com is sent
    - unsafe-url
        - https://domain.com/page.html?id=1 to any domain or path – https://domain.com/page.html?id=1 is sent

Lastly, make sure you add the "HttpHeadersCookieSecurity" setting that prohibits cookies from being transmitted until an encrypted connection is established.

Example:

```
# BEGIN HttpHeadersCookieSecurity
php_flag session.cookie_secure on
# END HttpHeadersCookieSecurity
```