EC 521 Project: Survey & Research of Security and Privacy-related Header Deployment in the Wild

By Taozhan Zhang, Juehao Lin, Zihao Diao

The Problem

During recent years, a variety of security-related headers have been proposed to mitigate potential security threats on the Web. Some of those security headers are de facto that are observed by most modern browsers while some step forward even more by becoming standardized by one or more request for comments (RFCs) from IETF or W3C.

In this project, we measured to which extent those headers are deployed in the wild? In those who deployed, we are going to investigate whether those headers are deployed correctly: is the syntax correct? Is the best practice followed?

To be more specific, we analyzed the following standardized and/or defacto header standard in the wild: HSTS, CSP, `X-Content-Type-Options` Header, `X-XSS-Protection` Header, `X-Frame-Options` Header.

The Approach

Crawling the Internet

The project first needs a list of headers in the wild to do analysis against. Analyzing against the top websites is both promising and feasible.

The Tranco list, a list of domains that is ranked by their popularity, is used in the project to guide which domains to be crawled. The Tranco list is security-related research conscious and provides an open standard of inclusion, exclusion and ranking. In this project, we are crawling the first 12000 domains in the Tranco list with permanent ID 6JX4X, collected in October 2022.

The Tranco list provides a list of popular registered domains for us to start with. However, when a user is actually visiting a website, they are not visiting a domain. Instead, their browser (user agent) will help them to convert a domain (e.g., www.bu.edu) to a URL (e.g., https://www.bu.edu/), which uniquely identifies a resource on the Web and send the request to the server via HTTP or HTTPS. Our crawler needs to add additional information to the domain, including the protocol used (http), the path (/).

Another problem is that while it is conventional to do redirection from a website on your *naked domain* (e.g., bu.edu) to the host on which you are actually serving HTTP(S) content (usually the www subdomain. For example, www.bu.edu), or vice versa if the site decides to use the naked domain to serve HTTP(S) content. Some domains, due to historical reasons, are serving the main web content on domains other than the naked domain.

To combat the problems, a heuristic approach is used. For each domain, the naked domain and the www subdomain are both visited with the HTTP and HTTPS protocol. Only the root path will be visited. The crawler will also follow the redirection from the server. For example, for domain bu.edu, the following 4 URLs are used as start point of crawling: 1) http://bu.edu/; 2) http://www.bu.edu/; 3) https://bu.edu/; 4) https://www.bu.edu/; If Any of the URLs has been visited before, the crawler will skip.

Another issue is that some of the domains in the Tranco list are not intended for serving Web content. Some of the domains do not resolve its www subdomain. Some do resolve but will refuse to serve meaningful content. In the case of HTTPS services, the connections will likely fail because the host failed to give the correct certificate when doing a TLS handshake. An example for the former case is `root-servers.net`, a domain that is used solely in domain name systems. The later case is common in many CDN services. A common practice among CDN services is to use one domain they own to do load balancing and direct users to the closest server with DNS. In this case, the domain they use is not intended to serve any content as its sole role is to act as a CNAME target for their client's domain. One example is `edgekey.net` owned by Akamai which ranked top 50 in the list and serves no actual content. In this case, a normal user is not expected to visit the site by any means.

The Tranco list does provide ways to filter which domains to be included in the list but the mechanism does not allow for filtering with the intended use (for Web service or not) of the domain. The problem then must be solved using a simple try and fail method. The list of URLs to be visited is generated through the heuristics method mentioned above. Each URL is tried. If an URL fails, because either the domain is not resolved, the connection failed or the TLS handshake failed, it is simply skipped.

Crawling is done on a machine on the network of RCN (AS 6079) in the Boston metropolitan area, starting from Nov 14, 2022 and lasted about 24 hours. In the process, the top 12000 domains are visited and 51205 responses are recorded in a SQLite database.

There are a few considerations in the crawling process. The first one is crawling on a residential broadband network (RCN in this case). While cloud computing resources are vastly available at a low price, the problem is that the IP address of almost all major cloud providers has been marked as data center IP by GeoIP databases and threat models. A common practice among CDN providers and website owners is to have stricter firewall rules for data center IPs. This will result in possible blocking of our visit via IP level or application level firewalls during the process especially when we are sending a relatively large volume of requests during a short period of time[1]. Being blocked will result in possibly inaccurate headers leading to skewed analysis. Our assumption behind this is that most sites will not have the same set of headers when blocking a request on the application level since they probably will block the request at a very early stage

[1] During the process, the DNS traffic on my home network recorded by my local Pi-Hole instance is 10 times larger than a normal day, which means the process sent out approximately 10 times more requests to the internet than a normal household with two adults. – Zihao Diao

of the request processing pipeline, for example, when doing load balancing, and the request will not reach the application server who is responsible for adding the complete security related headers.

While load balancing by accessing from multiple locations with different IPs is possible via proxy servers, the approach is not used. This is due to the fact that redirecting and GeoLocking by IP is widely used in modern web applications. Visiting from different IPs will reduce the chance of being blocked but the result will be influenced by in which location the URL is visited. This will complicate the analysis process and lead to possible data skew.

Another consideration is to let our requests look more like ones sent by a browser. Web servers serve different content with possibly different headers (and possibly block suspicious requests) for different user agents. Most applications do this by looking into the `User-Agent` header sent by the user agent. To be absolutely safe, the crawler is configured to use the full set of headers used by Chrome version 107 on an Intel macOS when fetching an HTML web page using GET method, including the `User-Agent`.

The DNS requests in the whole process is also sent to reputable recursive DNS servers[2] via DNS over HTTPS with no local filtering done to the result. DNSSEC validation is also done to the results. This eliminated the chance of the ISP DNS server blocking and poisoning the DNS responses.

Parsing and Analyzing

Parsing and analyzing the responses are done offline after all headers and bodies are collected.

Which URLs to be included in the final analysis? The crawling process records all the responses that it gets. The response with the `3**` HTTP status code means that the response is a redirection. Those response's body will not be parsed and rendered by modern browsers. So the threat of miss configured CSP headers has little influence on them. So for simplicity, all the redirection responses are skipped in the analysis. This did bring problems to the aspects of HSTS. We will not be able to detect some certain case of HSTS misconfigurations, including adding HSTS headers to HTTP responses (as in this case most HTTP response are simply redirecting the HTTPS), downgrading HTTPS website to HTTP and so on.

After applying the inclusion rules, we have 17479 responses to do analysis on, out of the 51205 total responses recorded.

In additional to putting CSP-related header in the HTTP header section, the W3C standard allows for CSP headers to be included inside special `<meta>` tags in the HTML document with the `http-equiv` attribute set to "`Content-Security-Policy`" or "`Content-Security-Policy-Report-Only`". When the attribute is set, the user agent is

[2] CloudFlare DNS, Google Public DNS, Hurricane Electric Recursive DNS are used.

expected to retrieve the policy from the `content` attribute of the same tag. When multiple CSP headers present at the same time, in either HTTP headers or HTML meta tags, the policy is to be merged into one. To do this, a parser of CSP policies is written following the standard and all further analysis is done with the merged policy if the site provides multiple CSP headers.

The Result

Content Security Policy

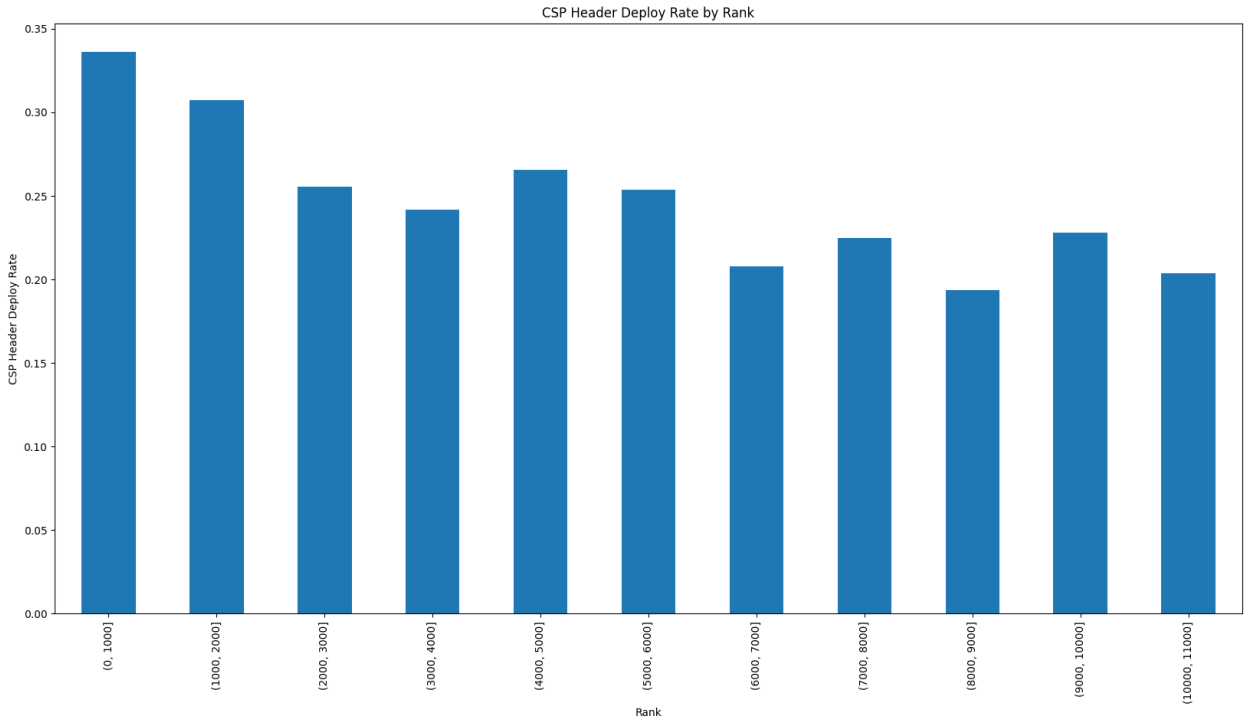
What is the Deploy Rate?

Out of 17479 valid responses we got from the top 12000 domains, we observed that 4174 responses contained valid CSP headers that can be parsed, resulting in a deploy rate of 23.8%.

There is another type of CSP header under the name `Content-Security-Policy-Report-Only` which tells the browser instead of rejecting requests according to the policy, the browser should report it to an endpoint set by the administrator. Notice that the report only header can be used in conjunction with the normal header and they can contain different sets of policies. In terms of the report only headers, we observed 521 responses containing them.

What factors will influence the deploy rate of CSP headers? We considered the problem in two dimensions.

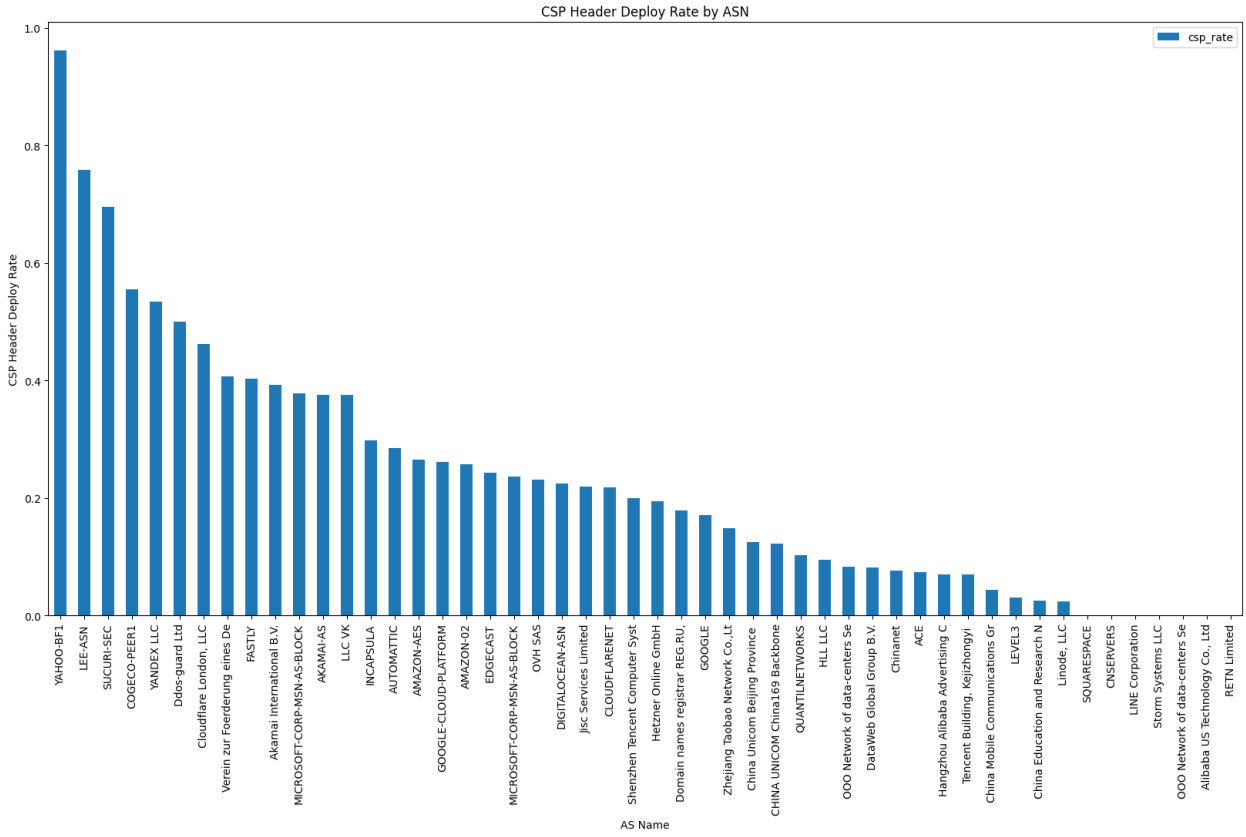
The first one is the ranking. Our assumption is that when the website is higher in ranking in the Tranco list, the site will receive more traffic which means both higher security threat to the users and website and more investment from the site owner which means possibly more investment in security. Both factors will lead to more deployment. The observation from the real world confirms with the assumption. The following figure shows the deploy rate of CSP headers in websites in different rank groups, grouped by every 1000 domains. We can observe a drop of deploy rate of 34.7% in the first 1000 domains to a deploy rate of 21.2% in the group of rank 10000 to 11000. However, more data on investment and traffic are needed to confirm the correlation between deploy rate, traffic and investment.



Different companies and hosting service providers will have different security policies, and provide different levels of tools for adopting a new technology. This factor can heavily influence the deploy rate of CSP headers. To observe this, we match the server who sent the response's address with the anonymous system (AS) in which the IP address is in with the help of the MaxMind GeoIP database. In modern internet architecture, an AS is usually managed by an ISP, a company or an institution. With the information, we can have a rough idea of how different companies and hosting services will have different deployment rates.

To do this, we aggregated the URLs by which AS the server address is in. We then choose the top 50 AS with the largest response from and calculate their CSP deploy rate. The figure 50 is chosen at the cutting point: ASes smaller than this point will have too few responses or the deploy rate close to 0 thus is not interesting in this part.

The result is shown in the graph below. Deploy rate varies greatly from AS to AS. This confirmed our assumption. However, this does not take into account the scale of the ASes and the difference between companies, institutions and service providers.



Attack Surfaces to Current Deployment

CSP took the permissive approach in its design. This means that the browser can only send requests to the resources that are permitted by the CSP policy. Sending the correct CSP policy is hard for site administrators. Having too strict CSPs will lead to web pages failing to load in some cases. Having too loose CSPs will often mean no or little protection to the users. Compromise and craftsmanship is needed when setting pages with the correct CSP headers.

However, most administrators do not follow the best practices when setting the headers. They tend to provide loosen policies for the pages on the server. In this section we are analyzing what attack is still possible with the current deployment.

Getting what policy is too loose or unnecessary will require static analysis of the web page, the script and the style sheets on the webpage or a way to dynamically run the web page in a browser which is far beyond the scope of the project. We are simply parsing the CSP headers and see some obvious common mistakes.

Too Permissive Derivative

Many sites are left with attackers with too permissive derivatives that allow the attacker to do things that they are not supposed to be allowed to, including inserting a script to the DOM and running it which means XSS is still possible, running user input via `eval()` and so on.

In this project, the CSP from each website is parsed and analyzed statically to see if the attack can still carry on those common attacks.

- **Allowing unsafe inline scripting or style sheets via** `unsafe-inline`: There are still a large number of website allows this practice, including top websites including Facebook, Twitter, Instagram and AWS. 1711 out of 17479 URLs is allowing this practice;
- **Allowing unsafe use of `eval()` via** `unsafe-eval`: This practice is still common in a large number of websites including Facebook, Twitter and Apple. 1378 out of 17479 URLs is allowing this practice;
- **Allowing resources loaded from anywhere**: This includes those policies that includes wildcard (*), a single `data:` or `https:` as the resource.

The top websites that allow those practices are listed in the tables below.

Table 1: Top URLs that includes `unsafe-inline`

Rank	Domain	Actual URL	Ranking URL
4	facebook.com	https://www.facebook.com/	http://facebook.com/
8	twitter.com	https://twitter.com/	http://twitter.com/
9	instagram.com	https://www.instagram.com/	http://instagram.com/
10	amazonaws.com	https://aws.amazon.com/	http://amazonaws.com/
12	apple.com	https://www.apple.com/	http://apple.com/
13	linkedin.com	https://www.linkedin.com/	http://linkedin.com/
32	pinterest.com	https://www.pinterest.com/	http://pinterest.com/
33	github.com	https://github.com/	http://github.com/
36	mail.ru	https://mail.ru/	http://mail.ru/
39	zhihu.com	https://www.zhihu.com/signin?next=%2F	http://zhihu.com/
41	whatsapp.com	https://www.whatsapp.com/	http://whatsapp.com/
43	zoom.us	https://zoom.us/	http://zoom.us/
56	myfritz.net	https://www.myfritz.net/devices/	http://myfritz.net/
67	vk.com	https://vk.com/	http://vk.com/

Table 2: Top URLs that includes `unsafe-eval`

Rank	Domain	Actual URL	Ranking URL
4	facebook.com	https://www.facebook.com/	http://facebook.com/
9	instagram.com	https://www.instagram.com/	http://instagram.com/
12	apple.com	https://www.apple.com/	http://apple.com/
36	mail.ru	https://mail.ru/	http://mail.ru/
39	zhihu.com	https://www.zhihu.com/signin?next=%2F	http://zhihu.com/
41	whatsapp.com	https://www.whatsapp.com/	http://whatsapp.com/
67	vk.com	https://vk.com/	http://vk.com/
67	vk.com	https://vk.com/	http://www.vk.com/
71	mozilla.org	https://www.mozilla.org/en-US/	http://mozilla.org/
73	tiktok.com	https://www.tiktok.com/	http://tiktok.com/
79	msn.com	https://www.msn.com/	http://msn.com/
84	tumblr.com	https://www.tumblr.com/explore/trending?source=homepage_explore	http://tumblr.com/
87	spotify.com	https://open.spotify.com/__noul__?pfhp=2c2ccb58-8a92-4713-a1c0-8b43b3090b49	http://spotify.com/
88	nih.gov	https://www.nih.gov/	http://nih.gov/
89	paypal.com	https://www.paypal.com/us/home	http://paypal.com/
92	nytimes.com	https://www.nytimes.com/	http://nytimes.com/
93	xvideos.com	https://www.xvideos.com/	http://xvideos.com/
97	flickr.com	https://flickr.com/	http://flickr.com/

Table 3: Top URLs that includes `*`, empty `data:`, empty `http(s):`

rank	domain	url	start_url
4	facebook.com	https://www.facebook.com/	http://facebook.com/
8	twitter.com	https://twitter.com/	http://twitter.com/
8	twitter.com	https://twitter.com/	http://www.twitter.com/
9	instagram.com	https://www.instagram.com/	http://instagram.com/

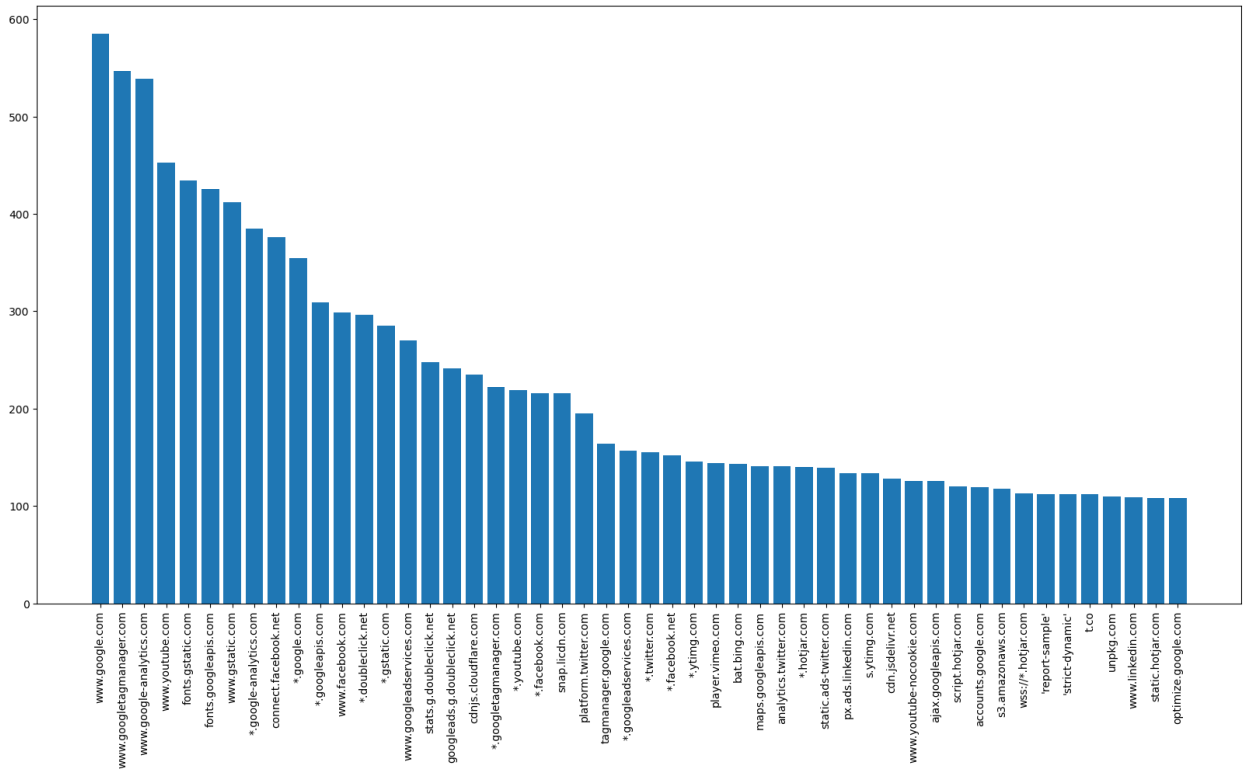
10	amazonaws.com	https://aws.amazon.com/	http://amazonaws.com/
10	amazonaws.com	https://aws.amazon.com/	http://www.amazonaws.com/
12	apple.com	https://www.apple.com/	http://apple.com/
13	linkedin.com	https://www.linkedin.com/	http://linkedin.com/
32	pinterest.com	https://www.pinterest.com/	http://pinterest.com/
33	github.com	https://github.com/	http://github.com/
33	github.com	https://github.com/	http://www.github.com/
36	mail.ru	https://mail.ru/	http://mail.ru/
36	mail.ru	https://mail.ru/	http://www.mail.ru/
36	mail.ru	https://mail.ru/	https://www.mail.ru/
39	zhihu.com	https://www.zhihu.com/signin?next=%2F	http://zhihu.com/
41	whatsapp.com	https://www.whatsapp.com/	http://whatsapp.com/
43	zoom.us	https://zoom.us/	http://zoom.us/
43	zoom.us	https://www.zoom.us/	http://www.zoom.us/
56	myfritz.net	https://www.myfritz.net/devices/	http://myfritz.net/
67	vk.com	https://vk.com/	http://vk.com/

Common Resources

What will happen if one site is seized by the attacker and almost all sites have some kind of permissive policy in the CSP? Then the attacker will probably gain the ability of inserting unwanted script, images and so on to the page.

The internet has become more centralized in the past few decades. How is this reflected in the CSP policy? What is the implication of it on the security of websites?

We collected the top 50 most common domains listed in the CSP policies of websites. Breaking into any of them, despite CSP being obeyed, will result in hundreds of websites vulnerable in some ways. The result is summarized below.



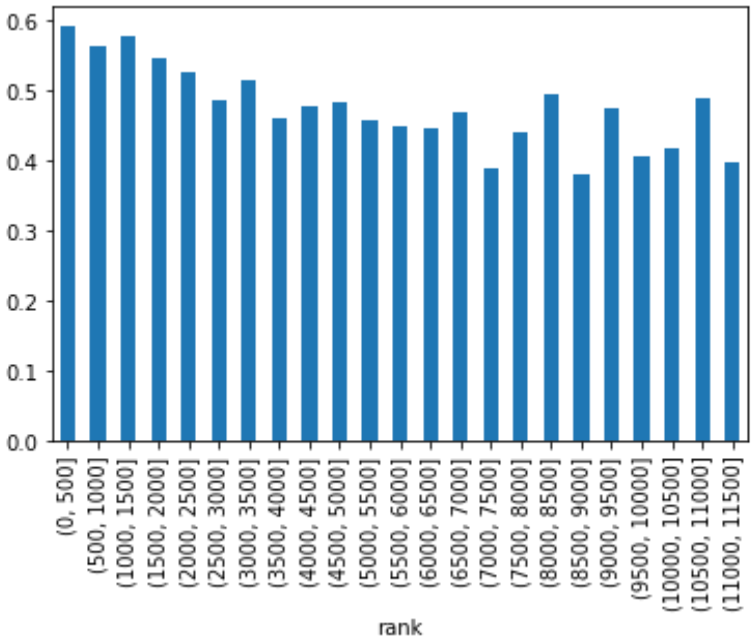
HTTP Strict Transport Security (HSTS)

General concepts:
The HSTS (HTTP Strict-Transport-Security) is a policy mechanism that allows web servers to declare that web browsers (or other complying user agents) should automatically interact with it using only HTTPS connections, which provide Transport Layer Security (TLS/SSL).
The HTTP Strict Transport Security header informs the browser that it should never load a site using HTTP and should automatically convert all attempts to access the site using HTTP to HTTPS requests instead. Therefore, helps to protect websites against man-in-the-middle attacks such as protocol downgrade attack and cookie hijacking.
The first time a user's site is accessed using HTTPS and it will return the HSTS header, the browser records this information, so that future attempts to load the site using HTTP will automatically use HTTPS instead. When the expiration time specified by the Strict-Transport-Security header elapses, the next attempt to load the site via HTTP will proceed as normal instead of automatically using HTTPS.

Directives:
`max-age=<expire-time>`;
The time, in seconds, that the browser should remember that a site is only to be accessed using HTTPS.
`includeSubDomains`;
If this optional parameter is specified, this rule applies to all of the site's subdomains as well.
`Preload`;

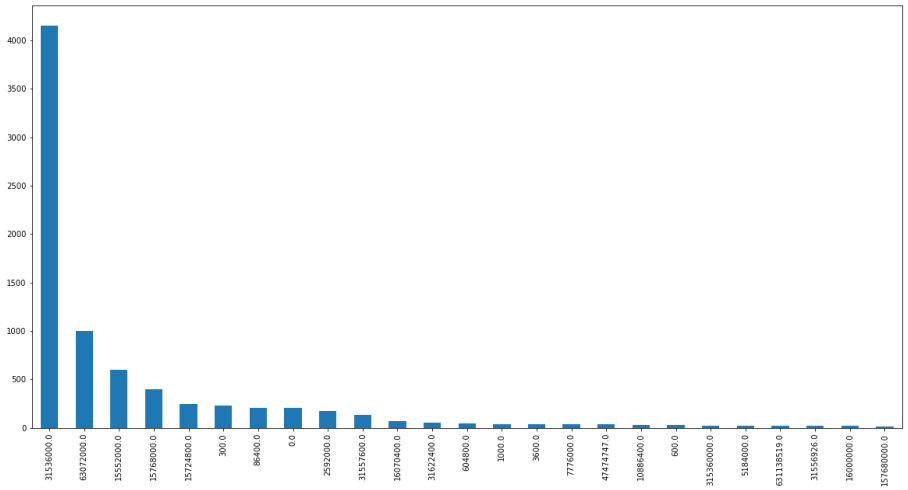
Make the browser to preload a global list of hosts and enforce the use of HTTPS ONLY on their site, so the browser does not need to depend on the issuing of the HSTS response headers to enforce the policy, which makes it a header independent browser that supported this service.

HSTS Deploy percentage sorted based on rank (range 500 per group)



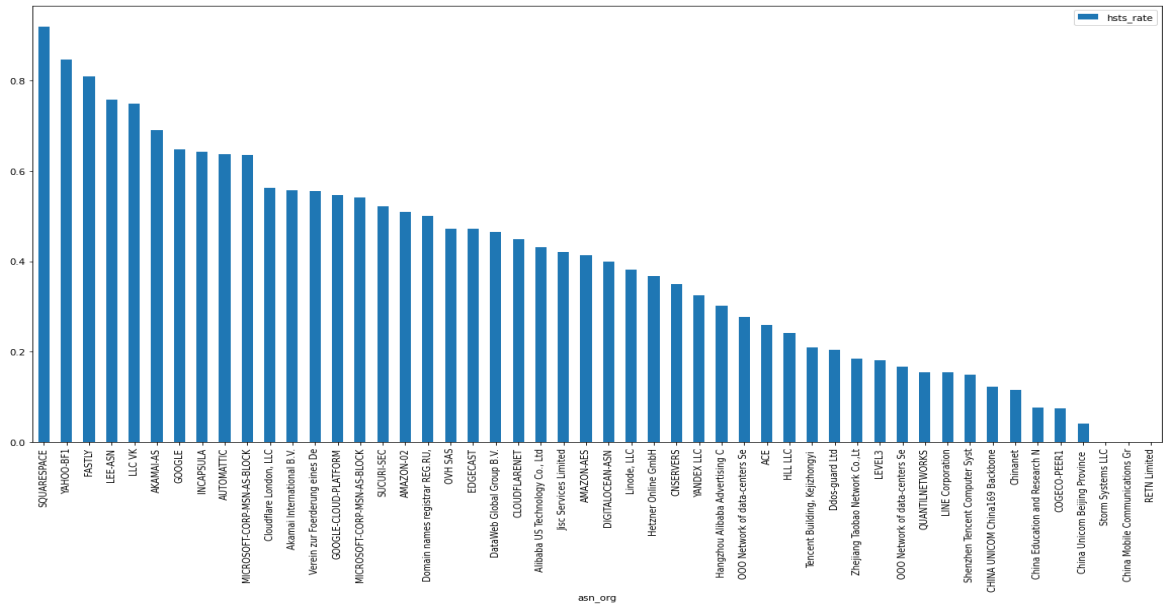
We analyzed the data and plotted the deployment rate of HSTS based on the popularity of the website. As usual, there is a correlation between the popularity and the deploy rate of the HSTS header. The more popular the website is, the more likely it is getting attacked and more likely it will spend the resources to set up defenses. The deploy percentage of the top 500 websites is around 60%, while the website after rank 5000 mostly have a deploy percentage of below 50%.

HSTS: `Max-age` distribution



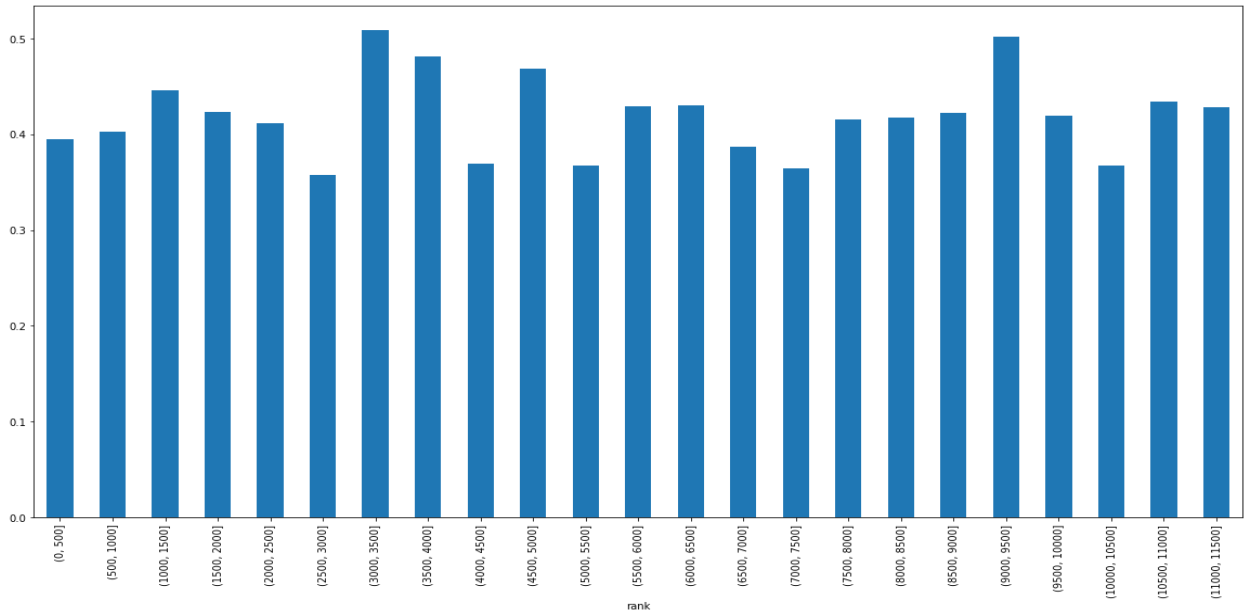
From the figure above, we can see that most of the Max-age is set to be 31536000. And we found out that such value in second can be converted to 31536000/(3600*24) = 365 days = 1 year. Which is the minimum requirement for preload service, as many organizations prefer to use it to make the loading process HSTS independent.
Another interesting thing is Max-age = 0.0. It seems to make little sense to set an instantly expired HSTS header, but there are practical uses of this value. Such value signals the UA to cease regarding the host as a known HSTS Host and use HTTP connection instead, which will include all the subdomain of the website if the includeSubDomains directive was enabled.

HSTS deploy rate per ASN



We also analyzed the deploy rate of HSTS header in different anonymous system (AS) in which the IP address is in with the help of the MaxMind GeoIP database. As we can see from the figure above, the deployment rate differs from different AS organizations, likely due to some AS will automatically apply a security header when users use their services. As a result, the deploy rate of HSTS header varies from 90% and down to ~0% depending on the AS.

HSTS `includeSubDomains` distribution



In addition, we analyzed the deployment rate of HSTS header. The deploy rating is mostly uniformly distributed, it is likely due to that there are many different implementations to cover subdomain and alias for security headers.

HSTS conclusion

In conclusion, our survey found that only ~60% of the top 500 websites have HSTS policy, which is in contrast to HTTPS being a common security practice in modern internet and only ~40% of the top 500 websites included sub-domain.
In addition to that, we also found a few cases where the HSTS header issued on the first user access, HSTS header was transmitted on an unsecured HTTP interface. Resulting in a vulnerability that attackers may be able to erase stored headers to remake a request, hence resulting in an unexpected data leak, or even an man-in-the-middle attack and redirected to a malicious link.

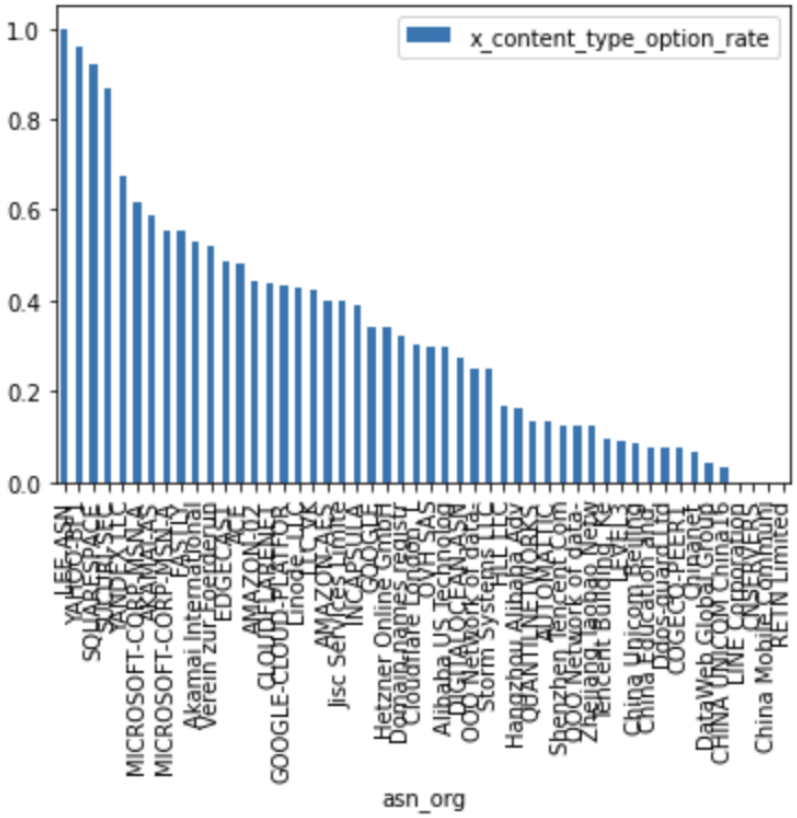
De Facto Headers

The de facto headers are designed to maximize the compatibility with the older version of the browser, meanwhile maintaining the security and blocking out the potential known vulnerabilities. Within the evolution of HTML standards and the integration of similar functionalities under CSP headers, these http security response headers are not as widely deployed or configured as before,

X-Content-Type-Options:
X-Content-Type-Options is one of the security response headers.The only defined value, "`nosniff`", prevents Internet Explorer from MIME-sniffing a response away from the declared content-type. This also applies to Google Chrome, when downloading extensions. It acts as a

marker used by the server to indicate that the MIME types advertised in the Content-Type headers should be followed and not be changed. The header allows users, for both the webmaster and the customer, to opt out of MIME type sniffing by saying that the MIME types are deliberately configured.

X-Content-Type-Options deployment rate per ASN:



The deploy rate is similar to the HSTS and CSP's circumstances as mentioned above, the deployment rate differs from different AS organizations, the relative larger AS group like Yahoo and squarespace have the highest deployment rates closing to 100%, whereas the other entities like Google, Microsoft are relatively lower from 50% to 70%. Noticeably, some entities do not deploy this feature or have the lowest deployment rates overall, like the China Mobile, due to the concern of overall HTML standards stability and security.

X-Frame-Options:
X-Frame-Options is one of the security response headers. Clickjacking protection: - no rendering within a frame, sameorigin - no rendering if origin mismatch, allow-from - allow from specified location, allowall - non-standard, allow from any location. It can be used to indicate whether or not a browser should be allowed to render a page in a `<frame>`, `<iframe>`, `<embed>` or `<object>`. Sites can use this to avoid click-jacking attacks, by ensuring that their content is not embedded into other sites.

Syntax:

```
X-Frame-Options: DENY
X-Frame-Options: SAMEORIGIN
```

Directives:

`DENY`

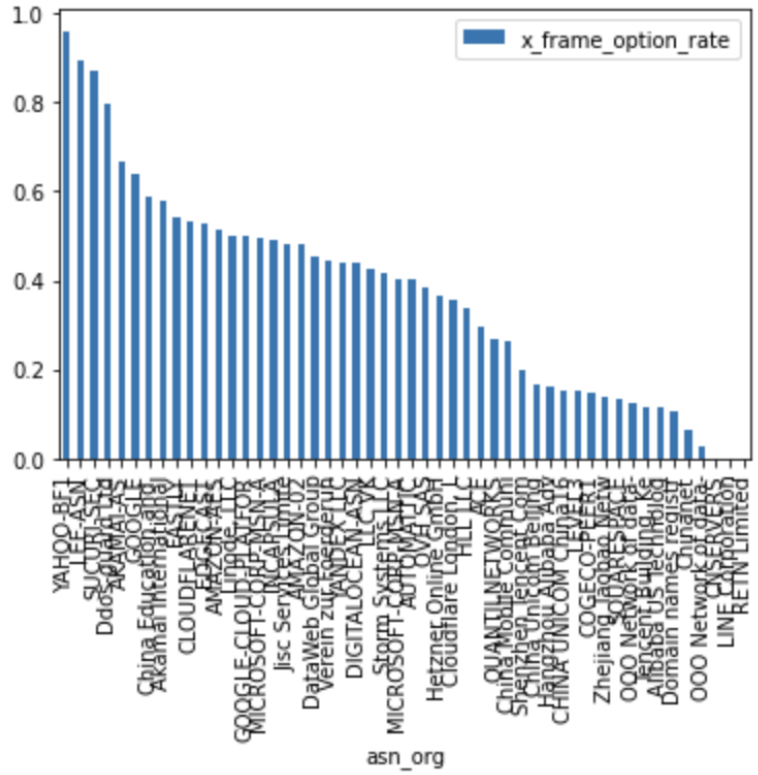
The page cannot be displayed in a frame, regardless of the site attempting to do so.

`SAMEORIGIN`

The page can only be displayed if all ancestor frames are the same origin to the page itself.

There are rare special cases where this header value is set to `ALLOW-FROM=url`, which is an obsolete directive that no longer works in modern browsers, except Firefox for android. The Content-Security-Policy:frame-ancestors have exactly the same functionality as this directive.

X-Frame-Options deployment rate per ASN:



The deployment rate is similar to the circumstances of X-Content-Type-Options, the deployment rate differs from different AS organizations, the relatively larger AS group like Yahoo and squarespace have the highest deployment rates closing to 100%, whereas the other entities like Google, Microsoft are relatively lower from 50% to 70%. Noticeably, some entities do not deploy this feature or have the lowest deployment rates overall, like the China Mobile, due to the concern of overall HTML standards stability and security.

Syntax merge error cases:

There are few deploy cases that both have `DENY` and `SAMEORIGIN` value at the same time. Presumably happened during the HTML standards update, as all of the three HTML security response headers constructed as key-value dictionary pairs, the error on the merge of configuration didn't overwrite the value and just simply appended, separated by comma. Similarly, the X-Content-Type-Options syntax has the exact same weird cases, but usually runs as `nosniff`, so there's really no concern as the browser is actually only running one single policy.

	rank	domain	url	start_url	remote_addr	http_header_x_content_type_options	http_header_x_frame_options	h
4034	2936	dns.id	https://pandi.id//	http://dns.id/	45.126.59.79	nosniff, nosniff	DENY, sameorigin	
4035	2936	dns.id	https://pandi.id//	http://www.dns.id/	45.126.59.79	nosniff, nosniff	DENY, sameorigin	

X-XSS-Protection:

X-XSS-Protection is one of the security response headers acting as Cross-site scripting (XSS) filter. Currently only Safari browser fully supports this feature, the other modern browser may still leave this configuration option on the run, but not set it to open source to developers, so it can be treated as a functionality loss feature. Initially, It is a feature of Internet Explorer, Chrome and Safari that stops pages from loading when they detect reflected 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'`) as mentioned in previous sections about CSP.

Syntax:

```
X-XSS-Protection: 0
X-XSS-Protection: 1
X-XSS-Protection: 1; mode=block
X-XSS-Protection: 1; report=<reporting-uri>
```

Directives:

`0`

Disables XSS filtering.

1

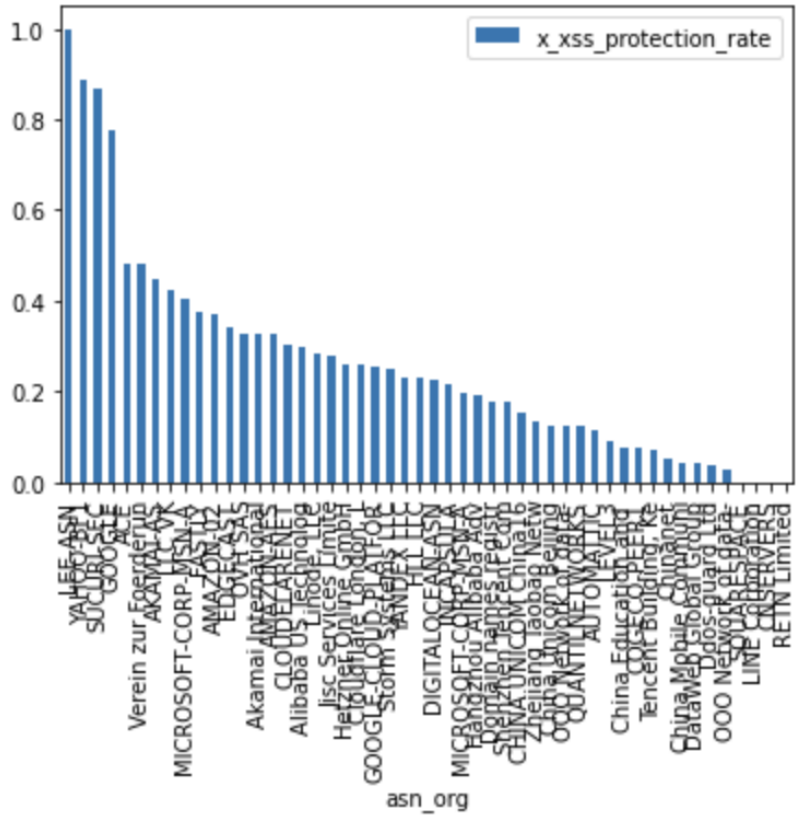
Enables XSS filtering (usually default in browsers). If a cross-site scripting attack is detected, the browser will sanitize the page (remove the unsafe parts).

`1; mode=block`

Enables XSS filtering. Rather than sanitizing the page, the browser will prevent rendering of the page if an attack is detected.

`1; report=<reporting-URI>` (Chromium only)
Enables XSS filtering. If a cross-site scripting attack is detected, the browser will sanitize the page and report the violation. This uses the functionality of the CSP report-uri directive to send a report.

X-XSS-Protection deployment rate per ASN:



The deployment rate has some differences compare to the circumstances of X-Content-Type-Options and X-frame-options, though Yahoo still have highest deployment rate, other entities has significantly lower deployment rate there, presumably due to the integration of CSP headers, the same functionality can be achieved via the configuration in a more effective method.

De Facto headers conclusion

3		0, 1; mode=block
4		0, 1; mode=block
5		0, 1; mode=block
6		0, 1; mode=block
7		0, 1; mode=block
8		0, 1; mode=block
9		0, 1; mode=block
10		0, 1; mode=block
11		0, 1; mode=block
12		0, 1; mode=block
13		0, 1; mode=block
14		0, 1; mode=block
15		0, 1; mode=block
16		0, 1; mode=block
17		0, 1; mode=block
18		0, 1; mode=block
19		0; report=/xss-report?...
20		0; report=/xss-report?...

Syntax merging error is significant, especially for X-XSS-Protection as the options expanded to 4. Some sites configure this header value to be a real contradiction, where the multiple syntax has totally opposite functionality shown up at the same time, disable XSS filtering and enable XSS filtering simultaneously. Unfortunately, the xss-protection header relevant manual file is not open source which makes it almost impossible to know the details of implementation under such a situation. The mechanism for combining multiple headers with the same field, which means multiple syntax under one single http header, into one field-name: field-value pair, or key-value dictionary pair structure as previous mentioned, appending each subsequent field value to the combined field value in order, separated by a comma, without changing semantics of the message. Still, this may leave vulnerable spaces for malicious users as these syntax are actually identified by the browser as an undefined behavior, so the browser can either act on all directives, where contradictions arise and result in lethal configuration fault, or do nothing at all, which is exact the same as no configuration had been implemented at all.

The Limitations

The analysis in this project report in no way means to be complete. The project is just doing static analysis on the patterns that the headers are deployed.

Additionally, some of the headers mentioned in the project may interact with each other. For example, how the `upgrade-insecure-request` directive in CSP interacts with the HSTS? How does the CSP interact with defacto headers mentioned in the last part? In both cases, the attack surface is narrowed. But is it possible for misconfigurations to interact with each other and lead to more severe security problems? This is still an open problem that is not solved by the project.

The Team

- Zihao Diao is responsible for crawling the HTTP response for the whole project, deciding the criteria of inclusion of requests, parsing the response and analyzing the CSP headers.
- Juehao Lin, focuses on hsts header data analysis and parsing.
- Taozhan Zhang, focus on the de facto headers data analysis and parsing.

References

- Tranco A Research-Oriented Top Sites Ranking Hardened Against Manipulation: https://tranco-list.eu/
- Content Security Policy Level 3, W3C Working Draft, 1 December 2022 https://www.w3.org/TR/CSP3/
- RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, Internet Engineering Task Force (IETF), June 2014 https://www.rfc-editor.org/rfc/rfc7230
- Weichselbaum, Lukas, et al. "CSP is dead, long live CSP! On the insecurity of whitelists and the future of content security policy." Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016.
- X-Content-Type-Options - HTTP | MDN Documentation: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options
- X-XSS-Protection - HTTP | MDN Documentation: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection
- X-Frame-Options - HTTP | MDN Documentation: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
- "'Xss': Can I Use... Support Tables for HTML5, CSS3, Etc." "Xss" | Can I Use... Support Tables for HTML5, CSS3, Etc, https://caniuse.com/?search=xss.
- List of HTTP header fields: https://en.wikipedia.org/wiki/List_of_HTTP_header_fields#cite_note-68
- Combining header fields RFC Documentation: https://www.rfc-editor.org/rfc/rfc7230#section-3.2.2
- HTTP Strict Transport Security (HSTS) RFC Documentation: https://www.rfc-editor.org/rfc/rfc6797
- HSTS Mozilla Documentation: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security