Contents

List of Figures

List of Tables

Attachments

1 Introduction

Rust is a programming language that focusses on performance, security an reliability. The *Rust Project* is open source and dual-licensed under the Apache 2.0 and MIT license (Hoare and et al. 2019). Rust 1.0 the first stable version was announced in May 2015 (The Rust Core Team 2015). This release also marked the start of the *commitment to stability* which promises stability on future Rust stable releases (Turon and Matsakis 2014). This new commitment also introduced a 6-week release cycle as well as development channels for language users and early adapters (Turon and Matsakis 2014). The latest stable compiler version 1.51.0 has been released on 25 of March 2021 (The Rust Core Team 2021). Developers and teams within the project put high effort into open communication. This focussed is formalized in the official *Code of conduct* (The Rust Team 2021b). The language with its connected tools has attracted over 5900 individual contributes as of writing this (The Rust Team 2021a).

The Rust project develops several tools besides the compiler itself. These tools are seen as a vital part in automating parts of the development process and collaboration among teams. *Clippy* is the official linter for Rust and is being developed in the *rust-clippy* repository. The linter contains over 450 lints which span from complexity and style lints over to restriction lints which might be required by certificates (The Rust Clippy Developers 2021). Clippy is written in Rust itself and interfaces with the compiler directly. This direct connection enables the use of the existing lexer, parser and connected diagnostic tools and ensures that the project stays up to date with the latest compiler changes. Since 2018 Clippy is distributed as a component of the Rust installation itself (Lusby 2018).

1.1 Problem

Clippy, like the compiler, puts great effort into giving helpful diagnostic messages. These messages consist out of a message, a direct quotation of the suboptimal code fragment and a direct suggestion how the code quality can be improved. The first message of a lint additionally includes a note why which lint is enabled in this location and a reference to the lint documentation in Clippy's lint list. This makes Clippy's lint list the second point of contact for new users with the project itself. The initial load time of the lint list is noticeably slower than most other websites in the Rust eco system.

Another pain point of the website are some technical deficiencies. The development documentation of the Rust project includes some technical guidelines that the current lint list doesn't fully fulfill.

1.2 Research question

The described problem in 1.1 leads to the following research question: *How can the internet presentation of the lint list for the rust-clippy project be improved?*

1.3 Overall goal

The primary goal of this paper is to review the previously mentioned problems and to find solutions for them.

1.4 Approach

The start of this paper will collect the formal and informal specifications that Clippy's lint list should optimally fulfill. The next part will analyze the current state of the website. This chapter will conclude with a list of aspects that should be improved as well as a selection which this research will focus on.

The previous preparation leads to the solution finding process. This process will start with a technical explanation of the aspect to determine why this specification is important. The next step tries to find a solution for the found issues under the given circumstances. Additionally, this section will include a practical test if the suggested change would improved the focussed problem.

This paper will end with a summary of the reviewed aspects and a conclusion as well as a suggestion what further work can be done on this topic.

2 Requirements

The goal of this work is to improve the impression of Clippy's lint list. This section of the document will set a list of requirements to focus on in further research for this paper. Requirements are usually split up into the following two groups (Sommerville 2010, p. 83ff):

- *Functional requirements* describe direct functionality and behavior that a system should provide. They can also be defined as negations, stating that a certain behavior should not happen. These requirements are usually documented in an abstract way to enable system users to understand them.
- *Non-functional requirements* are focussed on the characteristics of the system itself, an example might be the requirement to have a reliable and maintainable system. These requirements can include constraints that the system might have to take care of.

2.1 Implemented functionality

The topic of this assignment focusses on the perception and impression of Clippy's lint list as an entire system. The research question therefor puts a focus on non-functional requirements. The website itself is based on functional requirements and these should remain fulfilled even after the suggested changes. It is therefor important to note them in some way or form while not taking focus of the key point of this paper. All requirements that have been implemented previously will therefor be summarized in the following functional requirement: *The functionality of the website should not be impacted by the implementation of new measures to improve the impression or usage.* This requirement covers functionality like the search feature, filter options and theming. The implementation of all of these has been completed at the point of writing this. A more extensive specification of the underlying requirements can be retrieved from the rust-clippy issue tracker.

2.2 Non-functional requirements (Maybe: Requirements by the Infra team)

The Rust Infrastructure team has created a set of guidelines that static websites affiliated with the Rust project should fulfill. Clippy is an official Rust project and the website itself presents static content, the guidelines therefor apply to the website.

These guidelines contain a *formal specification* that states: *"The website must reach an A+ grade on the*

Mozilla Observatory." (The Rust Infrastructure team 2020). This specification is based on the requirement of security. The A+ grade ensures that all important headers have been set and that enhanced users privacy features should be enabled by the browser (The Rust Infrastructure team 2020). A secure website contributes to a trustful relation ship between the user and Clippy's lint list. It additionally improves the ranking in most search engines and therefor helps users to faster find the documentation they require.

The guidelines from the infrastructure team additionally contain some functional requirements that are not directly connected to the research question, they are also already fulfilled by the current setup. These are therefor included in the defined functional requirement.

2.3 Performance

The second major non-functional requirement this assignment will look at is performance.

- Search performance
- Load performance

3 Fulfillment of requirements

This chapter will measure the current fulfillment for the previously in 2 defined requirements. These results will then be used to identify the key aspects that need improvement. Additionally they will be used for comparison when testing suggestions.

3.1 Mozilla Observatory

Mozilla Observatory is a collection of tools that can analyze a website to determine which available security measures have been utilized by it (King and et al. 2018b). These security is focussed on values that can be set in the HTTP header to indicate that opt-in security options should be enabled by the browser (**TODO**). The Rust development documentation links to a free online interface[1] for the Mozilla Observatory that is provided by the Mozilla Foundation free of charge.

3.1.1 Scoring

The result of the analyzes is summarized in a single score with a corresponding grade. The score is calculated using a baseline each checked criteria can add bonus points or subtracted penalty. This implementation is used to give different weight to specific configurations. The significance of these modifiers are based on how important the analyzed aspect for security. Scores can range from a minimum of 0 to a maximum of 135, the score of 100 already indicates that the website is configured correctly a higher score can be archived by archiving bonuses. A score of 100 and above corresponds to the grade $A+$ (King and et al. 2018a).

The observatory documentation notes that all websites are graded equally, this means certain graded configurations might be unimportant for the specific use case (King and et al. 2020).

[1] `<https://observatory.mozilla.org/>`

3.1.2 Measurement

Scanning Clippy's lint list results in an overall grade of *C* with a score of 55/100. It is to note that the analysis cut of the path to the lint list and graded the host itself. The results are therefor for *rust-lang.github.io* in general. The score was calculated using the baseline of 100 points and subtracting a penalty of 45 points. This sanction is the result of three failed tests that are shown in table 1.

No.	Score	Reason
1.	−20	HTTP Strict Transport Security (HSTS) header not implemented
2.	−20	X-Frame-Options (XFO) header not implemented
3.	−5	X-Content-Type-Options header not implemented

Table 1: Mozilla Observatory analysis penalties for *rust-lang.github.io* from 2021-04-24

The original scan output with all test results is included in attachment number 1.

3.2 Initial page load time

What is exactly being measured?

- From opening the lint list
- To seeing the list (Loading... is not valid)

3.2.1 Note problems

Influenced by almost everything

Currently interested in significant speedups this means that small deviations are not detrimental.

3.2.2 Setup

Describe setup maybe using debugger, tool or log statements. Take average of X times

3.2.3 Measurement

Add a nice table with an average and so on

3.3 Summary of benchmarks

Select * from above where x.interesting is true

4 Analysis of benchmark results

This chapter will inspect each identified technical problem from section 3. The inspection will first explain the technical background behind the problem and then identify the optimal configuration.

The observatory scan focuses on HTTP header which are set by the server behind the domain. The scan was therefor conducted for the domain `rust-lang.github.io`. Clippy's lint list is indirectly included in this result as well as documentation from repositories by the *Rust Organization*. Further investigation will continue to focus on the context of Clippy's lint list however improvements to the server would directly improve other websites.

- TODO xFrednet 2021-04-29: Move section about clippy hosting to specification
- TODO xFrednet 2021-04-29: HTTP explanation/into header stuff

4.1 HTTP Strict-Transport-Security (HSTS)

HTTP Strict Transport Security (HSTS) is a optional HTTP header field that requests the client accessing the HTTP API to only use encrypted connection for further requests. The request to use and encrypted connection extends to all resources that are referenced by the requested result. It is therefor necessary that these resources also provide the option to connect via HTTPS (Hodges and et al. 2012, p. 6ff).

4.1.1 Risks

The specification references three threads that can be prevented using this header (Hodges and et al. 2012, p. 6ff):

1. Using an unencryped connection allows attackers to eavesdrop on the exchanged data. This is a *passive network attack* and can be used to collect personal information, passwords or browsing habits.
2. A HTTPS connection requires a certificate that has to be signed by a certification authority. This certificate intern lists the owner or organization. This can be used to validate that the displayed content really originates from the expected source and with that prevent attackers from creating a fake website copy to steal otherwise secure information.
3. Forcing the use of HTTPS additionally ensures that mistakes like referencing ressources via HTTP links will be corrected by the requesting client

4.1.2 Importance for Clippy

Clippy's lint lint only displays publicly available information about lints in a easy accessible and searchable way. A passive network attack could therefor not collect any secret of personal information about the user. Except the fact that they visited the domain at all. However, this would however also be possible with the header as the connected IP is not effected by it. This also extends to the third thread of accidentally not requesting unencrypted resources, this can currently still happen but would not be detrimental.

The second thread of modification of the website is the relevant thread in this case. An attacker could for instance inject a donation button as several developers have expressed interest to donate to the Rust Foundation itself. This button would then forward the user to another page of the attacker to donate.

With all of this being said it has to be noted that all references to the website already include `https` at the start and a user has to deliberately enter the domain with http in front. Most browsers will then still recommend to use the encrypted connection or at least add a *not encrypted* notice next to the URL. All of this results in a very low risk. The header should still be set if the hosting provider provides a simple setting for this. Also due to the fact that the targeted A+ rating would require this field.

4.2 X-Frame-Options (XFO)

This header was initially implemented by browsers as a non-standard HTTP header field as a new security measure to prevent the thread clickjacking. In 2013 the header was formalized by the *Internet Engineering*

Task Force (*IETF*) in RFC7014. Clickjacking describes is the act of hijacking clicks of the user, this can be done by embedding a website that should be hijacks as a frame and than getting the user to unknowingly interact with that site. The XFO header field allows a host to specify that delivered content must not be displayed in a frame (Ross and Gondrom 2013, p. 3).

4.2.1 Variations

The option can be set to three mutually exclusive values (Ross and Gondrom 2013, p. 4):

- *DENY*: Indicates that the content should not be displayed in any frame.
- *SAMEORIGIN*: Allows the display of the content inside a frame as long as it originated from the same origin as the frame.
- *ALLOW-FROM*: This prohibits the display of the content with the exception of the origins that are defined after the "ALLOW-FROM" value.

4.2.2 Importance for Clippy's lint list

Clickhijacking is used to make a victim interacts with a different website to use the privileges or data that the user has saved on that site. Clippy's lint list provides the same data to everyone and the only user specific data is the selected color theme. An attacker has therefor nothing to gain with this attack. Adding the header would actually reduce flexibility from external users to embed the lint list in their own interface, even if the project at this point doesn't know of website doing so.

However, Clippy's lint list is just one site that's hosted under the domain, it should be investigated if other sites contain sensitive data that would require the header. This paper will still look into setting the header as it is required so receive a A+ grade by Mozilla Observatory. The goal will therefor be to set the header to *DENY* this can later be expanded to *SAMEORIGIN* or *ALLOW-FROM* if required.

4.3 X-Content-Type-Options

In 2008 the *X-Content-Type-Options* HTTP header was initially implemented by Microsoft in Internet Explorer 8 to prevent attacks that abuse *MIME-sniffing* for attacks (Lawrence 2008b). HTTP includes a content-type header that indicate the type of content that is being delivered, these types are called *MIME types*. Most browsers have a mechanic called *MIME-sniffing* to determine what MIME type the received resource is in. This functionality is used for backwards compatibility with for example legacy servers that serve all content with the `text/plain` content type. MIME-Sniffing can determine that received data is in a different data type than specified and display it in the determined way. This would for instance render a HTML document that is send with the `text/plain` content-type if the text contains HTML elements (Lawrence 2008a).

The feature has however introduced some security concerns for content hosts. Attackers could create content like images that contain HTML text with scrips. The sniffing functionality could then falsely determine during the inspection that the received resource is a HTML document and then execute the contained script instead of showing an image (Lawrence 2008a). This lead to the introduction of the X-Content-Type-Options field that can be used to prevent such content sniffing (Lawrence 2008b).

The header can only be set to `nosniff` which disables the sniffing feature. It is supported by all major

browsers (Bengtsson and et al. 2021).

4.3.1 Importance for Clippy's lint list

This field can actually be of high importance to the project. Clippy like all Rust projects has a review policy that only allows the merge of changes if they have been reviewed by a project member. This type of attack especially focusses on hiding the malicious code inside an image, this could therefor also easily be overlooked during the review process. Additionally due to the fact that the project maintainers mainly focus on Rust and not the website.

This header requires that the `content-type` header is set correctly for content that is being delivered by the host. GitHub pages doesn't support the manual specification of the content type it instead uses a open source database to determine the correct MIME type based on the file extension (GitHub Docs 2021a). Clippy's lint list is composed out of a *html* and a *json* which both are delivered withe the correct content type as can be seen in attachment 2 and 3. The nosniff option can therefor be enabled without side effects.

4.4 Slow loading times

Some form of art

5 Solutions for unfulfilled specifications

The analysis has shown that the aspects to improve can be split into two parts. First the addition of HTTP headers for security and secondly the optimization of the website content for faster loading times. These two will be investigated individually.

5.1 HTTP header fields

The analysis has shown that three HTTP headers have not been set and determined that they should be configured as defined in Table 2.

HTTP header field	Value	Reference
`Strict-Transport-Security`	`max-age=63072000`	See 4.1
`X-Frame-Options`	`DENY`	See 4.2
`X-Content-Type-Options`	`nosniff`	See 4.3

Table 2: Determined values for the investigated HTTP header fields

5.1.1 GitHub Pages configuration

The GitHub Pages documentation does not contain any information if and how HTTP header can be set. There has been requests to support user defined HTTP headers in several places by the GitHub community. All of them have concluded with the answer that this is currently not possible (trante and et al. 2013, Laukenstein and Balter 2017, yawnoc and et al. 2021).

Searching in the documentation for the header functionality reveals that GutHub Pages provides an option called "Enforce HTTPS" (GitHub Docs 2021b). This option can be enabled for each hosted site, under the condition that the original `github.io` domain is used (GitHub Docs 2021b). Putting this setting to

the test under a personal fork of the rust-clippy project reveals that the effect is limited. Requesting the project domain over HTTP results in a *301 Moved Permanently* responds that forwards the browser to the same domain using HTTPS. The Strict-Transport-Security header which could enforce this behavior by the client is not set. The responds for the test page is included in attachment 4. This forward message only works for the root project url, other resources and direct HTML pages can still be loaded without an encrypted connection. Clippy uses paths to display version specific documentation. This setting is therefor not helpful in enforcing HTTPS security for Clippy.

The GutHub Pages documentation currently does not contain any information regarding the other header options.

5.1.2 HTML meta tag

A discussion on the topic of setting HTTP header fields in GutHub Pages included suggestions to use a meta tag inside the main html file header (trante and et al. 2013). The meta tag is part of the living HTML standard defined by the *Web Hypertext Application Technology Working Group* (*WHATWG*). It can be used to add supplementary information for the client. The tag can contain a `http-equiv` attribute with a linked `content` attribute that can define values that would usually be set in the HTTP response header. The standard currently defines a set of fields that can be set with the meta attribute. The in focussed fields defined in Table 2 are not listed in the living standard (WHATWG 2021). However, clients can still deviate from this standard or support additional functionality that has not yet been specified.

Putting the meta tag to the test reveals that both Firefox and Chromium accept values for Strict-Transport-Security and X-Content-Type-Options. Assigning a value to X-Frame-Options produces a warning in the Chromium console with the message that this option is not supported and should be set as a HTTP header. After setting the meta tags both browsers take care to enforce HTTPS connections for all requested resources. Accessing a HTTP connection produces in both cases an error message indicating that mixed content is not allowed and the request has been blocked.

The meta tag can therefor be used to define Strict-Transport-Security and X-Content-Type-Options fields for individual websites and with that increase security. The max-time defined in the Strict-Transport-Security header also ensures that future accesses to the website will use HTTPS. This solution still has four drawbacks:

- The header to enforce HTTPS is only set during the loading of the page. An attacker could therefor still modify the page and remove the tag if they catch the initial request where HTTPS is not yet enforced.
- The meta tag to set these headers is not yet fully specified and can still change. The fact that Chromium and Firefox both accept these headers is an additional functionality.
- The X-Frame-Options header can not be set via a meta tag. This header is as discussed in 4.2 the least important for now but still relevant when it comes to the Mozilla Observatory rating.
- These meta tags are defined in HTML files, it will therefor not increase the Mozilla Observatory scoring and they have to be added to each project in each html file to ensure that they are enforces in the project.

5.1.3 Access over Proxy

- Maybe cloud front (But GH still links the environments directly)
- Possible, but extra service that has to be maintained.
- GH directly links to the github.io page therefor not all solved
- Old references will continue to link to io domain

5.2 Slow loading times

Well optimize with the use of static content

6 Conclusion

6.1 Summary

6.2 Steps moving forward

6.2.1 HTTP header fields

This can be done using cloudfront but the importance is as discussing in ref1, ref2, ref3 not vital for the current lint list and future plans. Therefor analyze if other projects contain sensitive information and if not... Don't touch a running system ;)

6.2.2 Slow loading website

Discussion of moving to MD book, mentioned in (Metadata collection PR ref) (Darkmode issue ref) and (Clippy book PR ref). And progress will be tracked by (Clippy book PR and me)

References

Bengtsson, Peter and et al. (2021-03). *X-Content-Type-Options*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options (visited on 2021-04-26).

GitHub Docs (2021a). *About GitHub Pages*. URL: https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages#mime-types-on-github-pages (visited on 2021-04-30).

– (2021b). *Securing your GitHub Pages site with HTTPS*. URL: https://docs.github.com/en/pages/getting-started-with-github-pages/securing-your-github-pages-site-with-https (visited on 2021-04-30).

Hoare, Graydon and et al. (2019-01). *COPYRIGHT*. URL: https://github.com/rust-lang/rust/blob/master/COPYRIGHT (visited on 2021-04-20).

Hodges, Jeff and et al. (2012-11). *RFC6797: HTTP Strict Transport Security (HSTS)*. Tech. rep. Internet Engineering Task Force (IETF). URL: https://tools.ietf.org/html/rfc6797 (visited on 2021-04-25).

King, April and et al. (2018a-01). *HTTP Observatory Scoring Methodology*. URL: https://github.com/mozilla/http-observatory/blob/fa38ab4/httpobs/docs/scoring.md (visited on 2021-04-24).

– (2018b-03). *Mozilla HTTP Observatory*. URL: https://github.com/mozilla/http-observatory/blob/1bb1566/README.md (visited on 2021-04-24).

– (2020-10). *Frequently Asked Questions*. URL: https://observatory.mozilla.org/faq/ (visited on 2021-04-24).

Laukenstein, Binyamin and Ben Balter (2017-02). [*Github Pages*] *Modify headers, respect _config.yml webrick headers*. URL: https://github.com/github/pages-gem/issues/415 (visited on 2021-04-30).

Lawrence, Eric (2008a-02). *IE8 Security Part V: Comprehensive Protection*. URL: https://docs.microsoft.com/en-us/archive/blogs/ie/ie8-security-part-v-comprehensive-protection (visited on 2021-04-29).

– (2008b-02). *IE8 Security Part VI: Beta 2 Update*. URL: https://docs.microsoft.com/en-us/archive/blogs/ie/ie8-security-part-vi-beta-2-update (visited on 2021-04-29).

Lusby, Jane (2018-07). *Add clippy to the tools list #1461*. URL: https://github.com/rust-lang/rustup/pull/1461 (visited on 2021-04-17).

Ross, David and Tobias Gondrom (2013-10). *RFC7034: HTTP Header Field X-Frame-Options*. Tech. rep. Internet Engineering Task Force (IETF). URL: https://tools.ietf.org/html/rfc7034 (visited on 2021-04-25).

Sommerville, Ian (2010). *Software Engineering*. 9th edition. 501 Boylston Street, Suite 900, Boston, Massachusetts 02116: Pearson Education, Inc. ISBN: 978-0-13-703515-1.

The Rust Clippy Developers (2021-03). *Clippy*. URL: https://github.com/rust-lang/rust-clippy/blob/7fcd1/README.md (visited on 2021-04-17).

The Rust Core Team (2015-05). *Announcing Rust 1.0*. URL: https://blog.rust-lang.org/2015/05/15/Rust-1.0.html (visited on 2021-04-17).

The Rust Core Team (2021-03). *Announcing Rust 1.51.0*. URL: https://blog.rust-lang.org/2021/03/25/Rust-1.51.0.html (visited on 2021-04-17).

The Rust Infrastructure team (2020-03). *Rust Infrastructure hosting for static websites*. URL: https://forge.rust-lang.org/infra/guidelines/static-websites.html (visited on 2021-04-24).

The Rust Team (2021a-04). *All-time Contributors*. URL: https://thanks.rust-lang.org/rust/all-time/ (visited on 2021-04-20).

– (2021b). *Code of conduct*. URL: https://www.rust-lang.org/policies/code-of-conduct (visited on 2021-04-20).

trante and et al. (2013-02). *Github pages, HTTP headers*. Last time updated on 2019-06-18. URL: https://stackoverflow.com/questions/14798589/github-pages-http-headers (visited on 2021-04-30).

Turon, Aaron and Niko Matsakis (2014-10). *Stability as a Deliverable*. URL: https://blog.rust-lang.org/2014/10/30/Stability.html (visited on 2021-04-20).

WHATWG (2021-04). *HTML Living Standard*. Tech. rep. Web Hypertext Application Technology Working Group (WHATWG). URL: https://html.spec.whatwg.org/multipage/semantics.html (visited on 2021-05-01).

yawnoc and et al. (2021-04). [*Feature request*] *Set HTTP header to opt out of FLoC in GitHub Pages*. URL: https://github.community/t/feature-request-set-http-header-to-opt-out-of-floc-in-github-pages/174978 (visited on 2021-04-30).

7 Attachments


```
{
    "content-security-policy": {
        "expectation": "csp-implemented-with-no-unsafe",
        "name": "content-security-policy",
        "output": {
            "data": {
                "connect-src": [
                    "'self'"
                ],
                "default-src": [
                    "'none'"
                ],
                "img-src": [
                    "data:"
                ],
                "style-src": [
                    "'unsafe-inline'"
                ]
            },
            "http": true,
            "meta": true,
            "policy": {
                "antiClickjacking": false,
                "defaultNone": true,
                "insecureBaseUri": true,
                "insecureFormAction": true,
                "insecureSchemeActive": false,
                "insecureSchemePassive": false,
                "strictDynamic": false,
                "unsafeEval": false,
                "unsafeInline": false,
                "unsafeInlineStyle": true,
                "unsafeObjects": false
            }
        },
        "pass": true,
        "result": "csp-implemented-with-unsafe-inline-in-style-src-only",
        "score_description": "Content Security Policy (CSP) implemented with unsafe sources inside style-src. This includes 'unsafe-inline', data: or overly broad sources such as
            ↪ https:.",
        "score_modifier": 0
    },
    "contribute": {
        "expectation": "contribute-json-only-required-on-mozilla-properties",
        "name": "contribute",
        "output": {
            "data": null
        },
        "pass": true,
        "result": "contribute-json-only-required-on-mozilla-properties",
        "score_description": "Contribute.json isn't required on websites that don't belong to Mozilla",
        "score_modifier": 0
    },
    "cookies": {
        "expectation": "cookies-secure-with-httponly-sessions",
        "name": "cookies",
        "output": {
            "data": null,
            "sameSite": null
        },
        "pass": true,
        "result": "cookies-not-found",
        "score_description": "No cookies detected",
        "score_modifier": 0
    },
    "cross-origin-resource-sharing": {
        "expectation": "cross-origin-resource-sharing-not-implemented",
        "name": "cross-origin-resource-sharing",
        "output": {
            "data": {
                "acao": "*",
                "clientaccesspolicy": null,
                "crossdomain": null
            }
        },
        "pass": true,
        "result": "cross-origin-resource-sharing-implemented-with-public-access",
        "score_description": "Public content is visible via cross-origin resource sharing (CORS) Access-Control-Allow-Origin header",
        "score_modifier": 0
    },
    "public-key-pinning": {
        "expectation": "hpkp-not-implemented",
```

```
        "name": "public-key-pinning",
        "output": {
            "data": null,
            "includeSubDomains": false,
            "max-age": null,
            "numPins": null,
            "preloaded": false
        },
        "pass": true,
        "result": "hpkp-not-implemented",
        "score_description": "HTTP Public Key Pinning (HPKP) header not implemented",
        "score_modifier": 0
    },
    "redirection": {
        "expectation": "redirection-to-https",
        "name": "redirection",
        "output": {
            "destination": null,
            "redirects": true,
            "route": [
                "http://rust-lang.github.io/",
                "https://rust-lang.github.io/"
            ],
            "status_code": null
        },
        "pass": true,
        "result": "redirection-to-https",
        "score_description": "Initial redirection is to HTTPS on same host, final destination is HTTPS",
        "score_modifier": 0
    },
    "referrer-policy": {
        "expectation": "referrer-policy-private",
        "name": "referrer-policy",
        "output": {
            "data": null,
            "http": false,
            "meta": false
        },
        "pass": true,
        "result": "referrer-policy-not-implemented",
        "score_description": "Referrer-Policy header not implemented",
        "score_modifier": 0
    },
    "strict-transport-security": {
        "expectation": "hsts-implemented-max-age-at-least-six-months",
        "name": "strict-transport-security",
        "output": {
            "data": null,
            "includeSubDomains": false,
            "max-age": null,
            "preload": false,
            "preloaded": false
        },
        "pass": false,
        "result": "hsts-not-implemented",
        "score_description": "HTTP Strict Transport Security (HSTS) header not implemented",
        "score_modifier": -20
    },
    "subresource-integrity": {
        "expectation": "sri-implemented-and-external-scripts-loaded-securely",
        "name": "subresource-integrity",
        "output": {
            "data": {}
        },
        "pass": true,
        "result": "sri-not-implemented-but-no-scripts-loaded",
        "score_description": "Subresource Integrity (SRI) is not needed since site contains no script tags",
        "score_modifier": 0
    },
    "x-content-type-options": {
        "expectation": "x-content-type-options-nosniff",
        "name": "x-content-type-options",
        "output": {
            "data": null
        },
        "pass": false,
        "result": "x-content-type-options-not-implemented",
        "score_description": "X-Content-Type-Options header not implemented",
        "score_modifier": -5
    },
    "x-frame-options": {
        "expectation": "x-frame-options-sameorigin-or-deny",
        "name": "x-frame-options",
```

```
            "output": {
                "data": null
            },
            "pass": false,
            "result": "x-frame-options-not-implemented",
            "score_description": "X-Frame-Options (XFO) header not implemented",
            "score_modifier": -20
        },
        "x-xss-protection": {
            "expectation": "x-xss-protection-1-mode-block",
            "name": "x-xss-protection",
            "output": {
                "data": null
            },
            "pass": true,
            "result": "x-xss-protection-not-needed-due-to-csp",
            "score_description": "X-XSS-Protection header not needed due to strong Content Security Policy (CSP) header",
            "score_modifier": 0
        }
    }
```


Attachment 1: Mozilla Observatory output for rust-lang.github.io from 2021-04-24

```
1  HTTP/1.1 200 OK
2  Date: Fri, 30 Apr 2021 19:08:24 GMT
3  Via: 1.1 varnish
4  Cache-Control: max-age=600
5  Expires: Fri, 30 Apr 2021 19:18:25 GMT
6  Age: 0
7  X-Served-By: cache-fra19138-FRA
8  X-Cache: MISS
9  X-Cache-Hits: 0
10 X-Timer: S1619809705.954395,VS0,VE90
11 Vary: Accept-Encoding
12 X-Fastly-Request-ID: 941e117097d3830dfbc28eda40096862241b1bcc
13 Server: GitHub.com
14 Content-Type: text/html; charset=utf-8
15 permissions-policy: interest-cohort=()
16 Last-Modified: Mon, 26 Apr 2021 21:40:11 GMT
17 Access-Control-Allow-Origin: *
18 ETag: W/"6087333b-4a05"
19 Content-Encoding: gzip
20 x-proxy-cache: MISS
21 X-GitHub-Request-Id: CB52:4B29:333A3A:3A24C8:608C3FC5
22 Content-Length: 4722
23 Accept-Ranges: bytes
```

Attachment 2: The HTTP response header for `https://rust-lang.github.io/rust-clippy/master/index.html` from 2021-04-30

```
1  HTTP/1.1 200 OK
2  Date: Fri, 30 Apr 2021 19:08:25 GMT
3  Via: 1.1 varnish
4  Cache-Control: max-age=600
5  Expires: Fri, 30 Apr 2021 19:18:25 GMT
6  Age: 0
7  X-Served-By: cache-fra19138-FRA
8  X-Cache: MISS
9  X-Cache-Hits: 0
10 X-Timer: S1619809705.316870,VS0,VE94
11 Vary: Accept-Encoding
12 X-Fastly-Request-ID: 30deedad3daffa933d4f55f7174a69c2fd13ffa3
13 Server: GitHub.com
14 Content-Type: application/json; charset=utf-8
15 permissions-policy: interest-cohort=()
16 Last-Modified: Mon, 26 Apr 2021 21:40:11 GMT
17 Access-Control-Allow-Origin: *
18 ETag: W/"6087333b-4ec5f"
19 Content-Encoding: gzip
20 x-proxy-cache: MISS
21 X-GitHub-Request-Id: 98EA:29DE:BE987C:C3D1A8:608C3FC5
22 Content-Length: 80925
23 Accept-Ranges: bytes
```

Attachment 3: The HTTP response header for `https://rust-lang.github.io/rust-clippy/master/lints.json` from 2021-04-30

```
1  HTTP/1.1 301 Moved Permanently
2  Server: GitHub.com
3  Content-Type: text/html
4  permissions-policy: interest-cohort=()
5  Location: https://xfrednet.github.io/rust-clippy/
6  X-GitHub-Request-Id: 3620:3A01:104182F:110536D:608C3D70
7  Content-Length: 162
8  Accept-Ranges: bytes
9  Date: Fri, 30 Apr 2021 17:25:04 GMT
10 Via: 1.1 varnish
11 Age: 0
12 X-Served-By: cache-fra19120-FRA
13 X-Cache: MISS
14 X-Cache-Hits: 0
15 X-Timer: S1619803505.769013,VS0,VE87
16 Vary: Accept-Encoding
17 X-Fastly-Request-ID: b837829c5922d053b087ff1e129d92f5b470a120
```

Attachment 4: The HTTP response for the test page with *enfore HTTPS* enabled from 2021-04-30