Abstract

Cross-Site Scripting (XSS) attacks continue to be one of the most common and important web application security weaknesses with many high-profile attacks still occurring. Content Security Policy (CSP) is an HTTP header designed for combating this but is grossly underused. CSP improves the security of websites, providing defence in depth against XSS attacks. Adoption will be increased by creating suitable software that decreases the difficulty of deploying a CSP on established websites. The system will be designed so that it can be self-hosted, increasing the audience who can use it, and be able to automatically patch the CSP in place with the administrator only approving changes. This is possible by using Caddy web server which has an API for graceful configuration updates. Another implementation could have been using a crawler to generate the policy without users sending reports, avoiding issues with malicious reports and this was explored before settling on the final design. The system was tested by running it against off the shelf software and then comparing what it produced to policies specified before testing. It is successful at producing a strict policy based on the reports it received but requires a large enough sample of reports covering the whole of the website. The policy may not be safe if the site uses unsafe practices and requires further work by the administrator. Some improvements could be made, for example the requirement for the administrator to verify changes before making them is prone to mistakes. This can be improved by automatically crawling the page the report was generated from to verify the same report is made potentially marking non-reproducible reports as malicious.

Contents

1 Motivation & Aims

The objective of this project is to create a system that can be used by any individual who manages a website that does not yet use a Content Security Policy. The system will need to be simple to use so that it lowers the barrier of entry to deploying a Content Security Policy on an established website. The outcome of the project should be that more websites will be able to deploy a Content Security Policy to increase the security of the web and its users.

1.1 General problem

Websites are visited by users using web browsers (user agents) which fetch content from the website using the Hypertext Transport Protocol (HTTP). This content is usually in the Hypertext Markup Language (HTML) format which is interpreted by the browser using the annotations in the markup along with referenced styles and rendered to be displayed to the user. HTML is extensive and allows a single page to embed several types of media and even be transformed by embedded scripts after being loaded. Figure 1 shows a short example that demonstrates how various resource types can be used in an HTML document, showing a stylesheet, image, sub frame, scripts and an inline script.

```html
<html>
<head>
    <link href="/css/style.css" rel="stylesheet">
</head>
<body>
    <img src="/images/logo.png" />
    <iframe src="https://www.youtube.com/embed/aAbBcCdDeEf"></iframe>
    <script src="https://a-public-cdn.test/script.js"></script>
    <script src="/js/script.js"></script>
    <script>alert('Welcome to the example');</script>
</body>
</html>
```

Figure 1: Example HTML document

The browser rendering the content does not have any information about how the HTML has been constructed and simply renders what it is given. Unfortunately web applications are not always completely secure and could be susceptible to a Cross-Site Scripting (XSS) attack which is defined as a flaw when ‘an application includes untrusted data in a new web page without proper validation or escaping’ which ‘allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites' (The OWASP Foundation 2017*a*). Cross-Site Scripting attacks are one of the 10 'most common and most important web application security weaknesses' according to The OWASP Foundation (2017*b*), ranking at number 7. The fundamental solution is for web applications to sanitise untrusted data when used in a context that can be interpreted by browsers as anything other than plain text, but as noted by Stamm et al. (2010) the problem can be approached in a different way with the browser able to 'disable invading scripts as they attempt to run'.

They proposed a mechanism called Content Security Policy (CSP) which website administrators would use to specify a list of allowed scripts and other resources. Figure 2 shows a CSP in the `Content-Security-Policy` header which could be used for the HTML document above included in the response part of an HTTP connection to fetch the HTML.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 361
Content-Security-Policy: default-src 'none'; frame-src https://www.youtube.com;
↪ img-src 'self'; script-src 'self' https://a-public-cdn.test
↪ 'sha256-XwMKDpLB3O+EJ6I2rO3aDSrf8CPm2rNSlIiJhBpZy7E='; style-src 'self';
```

Figure 2: Example HTTP connection

For each different type of resource (scripts, styles, fonts, frames, images, etc) a separate list can be applied or inherited from a default list. As can be seen in the example policy shown in Figure 2, which by default blocks everything but allows sub frames for YouTube; scripts from the same origin, a third party site, an inline script with the specified hash and images and styles from the same origin.

Without visibility on what is being blocked by web browsers (user agents) it would be easy to miss a resource that you do want to allow. CSP has a mechanism for this, the `report-uri` directive (for CSP 3 this will be replaced with `report-to`) which specifies a Uniform Resource Identifier (URI) for an endpoint that can collect violation reports from user agents that generate and submit reports when resources are blocked by the CSP. A violation report is a structured JSON payload containing the important information about why the resource was blocked. For example the web page the user was on (document URI), the blocked resource (blocked URI), the specific part of the policy that blocked the resource (violated directive), the directive that controls that resource type (effective directive) and the full policy served to the user. This feedback allows website administrators to fix omissions from the policy and see attempted attacks that were caught by the policy. CSP can also be deployed in a non-disruptive mode called report only using the `Content-Security-Policy-Report-Only` header. In this mode user agents still evaluate the policy and send violation reports but do not block violations, so the functionality of the site is the same as without a policy. The CSP and CSPRO headers can be set on the same HTTP response allowing administrators to test a change to the policy without affecting the site's usability while keeping the protection provided by the original policy.

1.2 Significance of problem

It is not only important for a website to protect itself against hijacking of user sessions and exploitation of the facilities of the website but also for the public image of affected websites. In 2018 thousands of websites which used a third party script for accessibility were found to be causing end users to run cryptocurrency mining code (Williams 2018). BBC News (2018) highlighted that the incident affected the *Information Commissioner's Office* (ICO), with the incident being more damaging to their public image since the role of the ICO is to 'uphold information rights in the public interest' (Information Commissioner's Office 2019) and is the national data protection authority for the United Kingdom. Williams (2018) reported that Scott Helme recommended sites use Subresource Integrity (SRI) which instructs 'browsers to verify that resources they fetch are delivered without unexpected manipulation' (MDN Web Docs n.d.). This is achieved by specifying a hash of the expected contents of external resources in the tag that embeds them in the HTML and verifying that hash before execution. In this case CSP would not have been effective at preventing the malicious script from being inserted into the legitimate third party resource. However, a strict policy would have blocked outward requests to communicate the mined coins to the attacker. It may seem of little importance that the attacker could execute their own code but not profit from it but in other scenarios the data attempting to be sent by the malicious code could be credit card details as happened to *British Airways* (Cellan-Jones 2018), with that incident potentially linked to a breach affecting 380,000 customers. Neither SRI nor CSP should be seen as the only solution to use; both are vital tools in protecting users of websites from these threats in different ways.

In September 2019 only 45,031 (5%[1]) of websites in the top 1 million served a Content Security Policy header in their response to the six-monthly crawl conducted by Helme (2019) with even fewer serving the report only variant at 2,289 (0.26%[1]). Helme (2020*b*) notes, in his latest analysis in March 2020, that 'the use of Content Security Policy has seen some good growth' with the figure increased to 51,986 (6%[1]) and report only variant to 2,394 (0.28%[1]). The increase of implementation over the six months can be seen from the graph in Figure A.1

[1]These percentages are of the sites that responded to the crawl, not the percentage of the 1 million. In March 2020 13% of sites did not respond so are not included in the percentages.

with almost all groups of 4,000 sites appearing higher, meaning that in general the March 2020 line is higher than the September 2019 line. The breakdown of the 45,000 and 52,000 sites with a CSP header can also be seen with the higher ranked sites being more likely to use a CSP. Of the top 100,000 websites in March 2020 only 7,737 served a CSP and of sites ranked between 900,000 and 1,000,000 it was only 3,604, with this trend likely to continue to sites less popular than the top 1 million. This demonstrates that only the most popular sites have the resources to consider implementing a CSP, but even of those sites only 8.8%[1]do.

It is not a surprise that CSP has a high barrier of entry to implement as it requires knowledge of each use of embedded resources across the codebase of the website, including less common flows which are rarely interacted with. A website's codebase can be substantial and grown massively over time without notes on new use of resources. The process of creating a suitable CSP for a website that is being created from scratch is much simpler since the policy can be modified each time a new resource is used, accurately reflecting the entire codebase of the website as it is developed. However, for established websites to use a CSP, a solution is required to aid website administrators with the creation of a policy which can be used in the report only mode to be modified in iterations by reviewing violation reports that are collected for each iteration of the policy.

1.3 Prior work on problem

This problem has been previously worked on by Scott Helme, who operates the *Report URI* service which provides a dashboard and collects CSP reports using the `report-uri` directive (Report URI n.d.). The process to manually curate a suitable policy for a website requires refining the policy based on the reports received, however it is easy to be overwhelmed by the number of reports. Helme announced a new feature of the service called *CSP Wizard* which provides the website administrator with suggested policy changes which they can apply (Helme 2018).

Helme offers this service as part of his commercial interests and has not reported on any specific findings of the use of the wizard despite giving an in depth discussion of its features in the announcement on his blog.

1.4 Specific problem

So, the specific problem I intend to solve is the need for a general purpose solution for an established website to use for developing a suitable Content Security Policy that can be served to site visitors. The solution should be able to be self-hosted to work for internal/intranet web applications as well as public facing ones on the Internet. There are few broad solutions to just collecting reports with Helme's *Report URI* dominating the market for monitoring a deployed CSP. Although the solution is to be tailored for use in creating a CSP from scratch it would be ideal if it could also fit into the ongoing monitoring market.

2 Methodology

2.1 Intention

I am intending to build a piece of software to be used by a website administrator to help them create a Content Security Policy for their already established website. This system will be set up by the website administrator and hosted on their own infrastructure. Violation reports sent by visitors' web browsers when they access the site will be collected by the system. Reports will be attached to the policy that was served to the user agent that generated the report. It will present these reports along with suggested changes to the administrator to review. Each report will be shown with a suggested change that is dynamically generated based on the contents of the report. When the suggested change is applied then the new policy would permit the resource and so the same report will not be generated for the future revisions. The website administrator will be able to select changes they want to incorporate into the policy. Using this iterative workflow at the end of the process the website administrator will be able to see the changes they made at each stage and cross reference changes with reports for the previous policies.

2.2 Approach

Helme's *CSP Wizard* is a feature within his Report URI service. To use it, websites specify a report URI pointing to his service which means that a website's users connect to his servers to send violation reports. These visitors may not be aware that any web requests they make exposes their IP address and User Agent to the remote server and are even less likely to be aware that their browser is communicating to a third party which unknown to them. The websites themselves must evaluate the risk of their customer's data being sent to Helme who could maliciously use it. For example, with this data Helme can construct profiles of these visitors by using the document URI which is included in the reports and by performing user agent fingerprinting to link reports together from other sites using his service. Some sites may be uncomfortable with this potential privacy risk and would prefer to host the software themselves and this option is not provided by Helme. In addition, the main use of the Report URI service is for sites that already use CSP and need insight into the reports and so sites that do not use CSP yet may not be aware of it. Additionally, some websites hosted internally within organisations may include sensitive content in the URL which would be leaked outside the organisation when a violation report is sent. Since the *CSP Wizard* is a managed service offered by Helme it is not suitable for these scenarios. Another solution is required which can be self-hosted and managed by the website operator without relying on a third party.

Every time a change is accepted in the *CSP Wizard* the policy must be manually copied and updated in the web server configuration. Some web servers require restarting to use the updated configuration and the person manually updating it may make a mistake. In some environments access to the server configuration may be tightly controlled and the people with access will not want to be disturbed every time a modification is wanted by the person creating the CSP. So, it would be ideal for the system to patch the CSP header of the web server without additional manual effort when the changes to the policy are applied. Zero downtime updates to the configuration are essential as the system should not interfere with how the application behaves and should be a seamless experience. Caddy is a modern web server written in Go that is dynamically configurable with an HTTP API (Caddy Web Server n.d.) so fits these requirements perfectly.

The web application that a policy is being created for will likely already be configured and tuned for another web server. It would cause friction to require that configuration to be rewritten for another web server software, potentially requiring the website administrator to learn a new syntax and options. If the original web server is serving multiple hosts it would require moving other hosts to the new server as well because two different applications cannot both listen on the same port (e.g. port 80). Therefore it is ideal that the original web server's configuration can be adapted to proxy requests to the site to Caddy which can listen on a separate port bound to localhost which then proxies back to the other web server with the original configuration moved to a different port bound to localhost. The original web server only has a small configuration change and Caddy will only need to be configured to set the CSP header and proxy the traffic back to the other port the original server is listening on with the original configuration. This minimal setup is of benefit as the web administrator does not need to alter the Caddy configuration to work with the web application. Another bonus is that the reports generated can be down sampled by only setting the CSP header on a subset of responses, achieved by only proxying to Caddy on a subset of requests. For example, using the nginx web server you can proxy to an upstream which load balances randomly between Caddy and the original configuration using weights to alter the probability of the request getting the header set (nginx docs n.d.).

2.3 Design

The system will aid the website administrator to create their CSP by using an iterative approach. The administrator will be able to make small modifications to the CSP in each iteration, eventually reaching the ideal CSP. The general algorithm for this process is shown in Figure B.1. This means the administrator will not need to be constantly monitoring the reports being received and can check it twice daily, once per day or twice per week depending on the amount of traffic the website gets.

A relational database will store all the policy revisions and reports with a web management interface for the website administrator. Each report will have a relationship to the policy revision that it was generated for. An Entity Relationship Diagram is given in Figure C.3.

To serve the Content Security Policy header to users of the origin application the requests will be proxied to the Caddy web server. Caddy will proxy back to the original web server which is configured for the application already. The origin application will send a response which will be returned back through the chain and Caddy will set the CSP header as it passes back to the other web server. This flow can be seen in Figure 3 which is simplified part of the system diagram shown in Figure C.1.

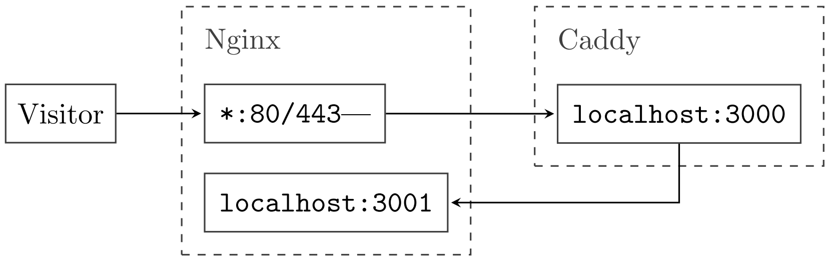


Figure 3: Host web server proxying

There will be an endpoint that collects violation reports and saves them to the database. The report will be validated to ensure it includes the required fields and only useful fields from the report will be saved to the database along with the user agent that sent the report and the time. Duplicate reports will be grouped by incrementing a count field. Each policy will have a Universally Unique Identifier (UUID) which will be used in the `report-uri` to facilitate linking collected reports to the correct policy revision. If sequential IDs were used for this a malicious party could enumerate these to send garbage reports.

The homepage of the management interface will show the latest reports received for the current policy revision. For each report a suggested change will be shown which ‘resolves’ the report as it would cause the report to not have been generated by the user agent if it was included in the policy served to it. A summary of these suggested changes will be shown below the list of reports, without duplicate suggestions.

The summary of suggested changes will be a web form in which the user will select the changes to incorporate into the policy. After the form is submitted, the user will be presented with the current policy and the policy they are about to update to (the current policy with the selected changes merged into it). In some instances the suggested change may be too specific and a broader value should be used instead, for example a site hosting user generated content may allow them to set their profile picture to an image hosted on a different site. The administrator will be able to use their judgement and knowledge of the website to make an informed decision to modify the change being made. For this reason, at the stage of confirming the update, the policy will be presented as a text input where the administrator can overwrite parts of the policy and then submit. Once confirmed, the revision will be saved to the database with a UUID generated. The domain for the report endpoint will be combined with the UUID in the path to generate the full `report-uri` which will be appended to the policy, giving the full CSP to serve to users. Using the Caddy API, the current report only header will be patched to change the value to the new CSP. The API can be used to fetch the value to confirm it was successfully updated and if not prompt the administrator to manually update the web server configuration, displaying the full CSP for them to copy. This allows for the system to be used without the dynamic updating provided by Caddy if the administrator does not wish to use it.

There will be a list of the various revisions of the policy for the administrator to audit. From this page the administrator will then be able to list reports that were saved for a particular policy revision. The suggested changes will be shown for this list just as for the reports for the current policy but there will be a notice informing the user that the changes may have already been made, encouraging them to verify the later policies before making the changes.

3 Results

3.1 Outcome

The system I have produced is a lightweight tool for curating a Content Security Policy by making changes based on reports received by users browsing the website. The system only requires interaction by the administrator through the web interface as it can automatically update the policy being served on the web server.

Appendix F contains screenshots of the system while in use. The interface is clean and simple with minimal distractions. CSPs are presented using syntax highlighting to help users with quickly scanning the policy. The screenshots are ordered and show a journey of an administrator using the system. Of note is the ability to browse reports for previous revisions and apply changes suggested for them which might not have been done at the time as shown in Figure F.10. Another feature to highlight is making manual changes to the policy before applying the patch. In Figure F.13 the administrator is shown several hosts about to be added to the permitted sources but they are aware that an image from any host can be embedded by users so decides to customise the policy to use a wildcard instead as shown in Figure F.14.

The system is made of several independent parts, Figure C.1 shows the overview of the whole system and how the parts interact. The domain names given are placeholders and can be replaced to the domain operated by the administrator.

3.2 Execution

At the start of the project I explored the various ideas that I had before committing to a design. One idea I was keen on was using *Headless Chrome* controlled by *Puppeteer* (Google Developers n.d.) to generate violation reports for a specific HTML document. The idea was to siphon off responses to real users of a website and queue a portion of the body of responses, for example using the random early drop queuing method. A worker that executed *Puppeteer* to render the given the HTML document and the Content Security Policy. This method would be transparent to the website visitor, the connection and speed would not be noticeably affected as all the processing would be done out of band of the connection to the origin application, see Figure C.2 which shows a diagram of this system.

However simply rendering the HTML document has the potential to miss huge parts of what the site should have in its CSP. Rendering the page does not submit forms, so you would have to come up with a solution to parse the HTML for the form action attributes for the `form-action` directive. It also does not click around or press buttons which may trigger JavaScript that loads other resources, for example another script, an image or even sending an HTTP request. This means this method would not work for many sites but also a whole class of websites, Single Page Apps where JavaScript is used to load all of the resources on the page and navigation to different pages on the site is done through JavaScript, simply rendering the HTML and JavaScript of the first page will miss content that loads after an interaction which is the entire rest of the website.

In the end this idea turned out to be an over complex reinvention of the principle of Content Security Policy in report only mode. It is much simpler to serve the CSP in report only mode so that the users who are visiting your website send you the violation reports. An advantage is that how the actual website users interact with the website is the most accurate representation of the real world use of the site, crawling and dumb rendering of isolated HTML documents cannot get close to that. However, the disadvantage of relying on end users to send reports is that malicious parties can take advantage by sending bogus reports or attempt to overload your report collection endpoint.

After this exploration I settled on a different system to solve the problem which is outlined in the Design section. I was keen for the system to be simple to set up and not require extensive reconfiguration of the origin application. The use of Caddy as the web server presented a problem as the site may use a different web server and not want to replace it since that would involve learning Caddy's configuration syntax. I was able to resolve this issue by only mandating minimal changes to the original web server configuration by passing traffic through from the original web server to Caddy and back to the original configuration. This proxying was beneficial as it only required a small change to make the original configuration listen on a localhost port and add a new configuration for the host on the original port to proxy the request to Caddy. This step is explained in the setup instructions and the resulting changes to the configuration file is only a few lines long. Furthermore, Caddy's configuration file for this purpose does not need any modification by the administrator irrespective of the website they are using the system for.

4 Evaluation

4.1 Testing

4.1.1 Approach

The approach I am taking to testing the solution is assessing its effectiveness when using it to create a policy for an off the shelf web application. The process will be to install it and generate a reasonable amount of content that it will use and display. I will then set up a fresh copy of the system and go through the process a website administrator would, to tune the policy served to visitors until there are no more changes suggested. To generate the violation reports, I imitated site visitors by browsing the website and interacting with features, for example submitting forms.

To measure the success of the system, after evaluating the final iteration of the policy, I will compare the computed policy against policies I specified before testing. This will include any policies recommended for the software in its documentation or another reliable source that are useful for evaluation against. I will also draft a policy before testing based on what I would expect to be allowed given knowledge of the content that is used in the application and brief inspection of the source. This draft policy will not be a perfect policy and is not exactly what I expect the system to produce; it will be used to verify that the content found by manual inspection is identified by the system.

To perform stress testing / benchmarking on the report endpoint, I required a load testing tool that could be used for both sending duplicate reports and unique reports. Several tools that were appealing could only send a single request body for all of the requests and so only met my first requirement while another allowed a request body to be specified per request but this meant computing and storing the thousands of request bodies upfront. It is important to use the same tool for both types of test so that the results from both could be comparable. I eventually settled on k6 (Load Impact n.d.) as it suited both requirements and allowed me to define the tests using a JavaScript program. It outputted the statistics I was looking for along with even more detailed breakdowns than other tools. The scripts used for these tests are given in Appendix D.

4.1.2 Results

Unfortunately, after researching various off the shelf open source software, I discovered that my measure of success based on recommended policies for software was too ambitious as open source projects did not provide a starter or recommended policy. It was disappointing to not find any documented policies for the software I researched because this is yet another limitation to widespread adoption of the Content Security Policy technology.

MyBB (forum software) MyBB did give a generic Content Security Policy in its documentation intended for ensuring embedded content is over HTTPS (MyBB Documentation n.d.) which is shown in Figure 4.

```
default-src https: data: 'unsafe-inline' 'unsafe-eval'; frame-ancestors 'none';
↪ base-uri 'self'; upgrade-insecure-requests
```

Figure 4: Suggested CSP for MyBB

It is useful for comparison but should not be taken to be the CSP the system should arrive at since it is very general and its purpose is not for strictly limiting embedded content.

Based on my knowledge of the content that can be used in MyBB and inspection of some of the source code I devised the predicted CSP shown in Figure 5, which can be used for comparison.

```
default-src 'none'; connect-src 'self'; form-action 'self'; frame-src *;
↪ img-src *; script-src 'self' 'unsafe-inline'; style-src 'self'
↪ 'unsafe-inline'
```

Figure 5: Predicted CSP for MyBB

Figure E.1 shows all the policy revisions made during testing using MyBB as the example software. After ten revisions to the policy when interacting with MyBB no more reports were generated. Figure 6 shows the final revision to the CSP for the MyBB software.

```
default-src 'none'; connect-src 'self'; font-src https://fonts.gstatic.com;
↪ form-action 'self'; frame-ancestors 'self'; frame-src 'self'
↪ https://www.youtube.com; img-src * data:; script-src 'self' 'unsafe-inline'
↪ 'unsafe-eval'; style-src 'self' 'unsafe-inline' https://fonts.googleapis.com
```

Figure 6: Final policy revision for MyBB

It can be seen that `'unsafe-inline'` and `'unsafe-eval'` were both correctly added to the policy but in the more specific directives (e.g. `'unsafe-inline'` was specified for script and style sources). Although the suggested policy denied frame ancestors, one page was served using a frame which would have been denied so the suggested policy or application should be revised to fix this. Only a YouTube video was encountered during the testing but other video platforms can be embedded in forum posts so it would be up to the administrator to alter the change made to either a list including the URLs for the various platforms or a more permissive wildcard. Ignoring the elements specific to HTTPS (and taking `https:` to be `*`), the suggested policy allows all the content that is allowed by the stricter final policy. The final policy also covers what was specified in the expected CSP with some additions not accounted for.

WordPress (blog software) During my research I could not find a recent suggested Content Security Policy for WordPress that would be suitable for comparison so there is only the policy I draft to use as a measure for the success of the WordPress test. From my understanding of the sort of content that can be used on WordPress and looking at the source I devised the predicted CSP shown in Figure 7, which can be used for comparison.

```
default-src 'none'; connect-src 'self'; form-action 'self'; frame-src *;
↪ img-src *; script-src 'self' 'unsafe-inline'; style-src 'self'
↪ 'unsafe-inline'
```

Figure 7: Predicted CSP for WordPress

Figure E.2 shows all the policy revisions made during testing using WordPress as the example software. After seven revisions to the policy when interacting with WordPress no more reports were generated. Figure 8 shows the final revision to the CSP for the WordPress software.

```
default-src 'none'; connect-src 'self'; font-src 'self' data:
↪ https://fonts.gstatic.com; form-action 'self'; frame-ancestors 'self';
↪ frame-src 'self' https://www.youtube.com; img-src * data:; style-src 'self'
↪ 'unsafe-inline' https://fonts.googleapis.com; script-src 'self'
↪ 'unsafe-inline' 'unsafe-eval'
```

Figure 8: Final policy revision for WordPress

Similarly to the test with MyBB, a YouTube embedded video was encountered and so allowed in the policy but other sites can be used in WordPress and the site administrator needs to use that knowledge to widen the scope of the directive as they update it. The final policy covers all the sources specified in the expected CSP with more additions like the MyBB test. These include `data:` for the font source list which based on the report and further investigation appears to be used as a backup source for the icons font used by WordPress.

Summary of usage test results The system was effective in both tests at identifying all the sources given in the predicted CSPs. Font sources (and style sources for those fonts) were not included in the predicted CSPs but in both tests the *Google Fonts* sources were added. Additional sources that needed to be permitted were picked up by the system which were not found by the manual inspection. This shows the success of the system in aiding an administrator with creating a CSP and how it is not effective to rely solely on manual inspection of software to create a CSP.

Stress testing / benchmarking Figure 9 shows the throughput of reports sent to the report endpoint for the stress testing. This is with the system running on an Ubuntu 18.04 virtual machine using VirtualBox with 2GB of RAM and one virtual CPU. PHP-FPM and nginx were not tuned for performance and used the default configuration for worker processes but OPcache was enabled in the `php.ini` configuration.

Report type	Requests	Requests / s	Request duration (ms)				
			min	max	avg	p(90)	p(95)
Duplicate	10000	160.67	6.96	619.3	62	90.84	118.03
	50000	162.08	6.39	624.58	61.47	92.09	126
Unique	10000	143.86	16.47	470.3	69.17	95.73	116.14
	50000	145.33	15.59	628.09	68.5	95.49	115.36

Figure 9: Table of benchmarking statistics

The results in Figure 9 show that the report endpoint can sustain a decent throughput of reports even when under load on this hardware. Users whose user agents are submitting the reports will not notice any effects of slow reports even if there were long request durations as they are handled asynchronously by the browser. Similarly, if requests were to time out this would not impact users browsing the website. If the reporting endpoint does become overloaded the technique described in the Approach section for down sampling the number of reports can be used to ensure the reporting endpoint is stable.

It is interesting to note that for unique reports, the requests per second, average duration and top percentiles' duration are extremely close for both test runs of 10k and 50k reports, whereas for duplicate reports the top percentiles' durations are both greater for the 50k test run than they are in the 10k one. I would hypothesise that this is because unique reports can be dealt with completely asynchronously as they are a simple insert into the database with no record locking. Whereas for duplicate reports a counter is incremented on the corresponding row in the database, all those incoming duplicate reports have to wait for the record to become unlocked before performing their update and finishing. With many requests being sent there is a greater amount of time for the number of requests waiting to increase which would increase the worst-case response times as the more requests waiting the longer it takes to complete them.

During the stress testing I discovered one instance where the count field was one less than expected. By analysing and debugging the code I realised there was a possibility that the first reports could pass the check for a similar report existing in the database and move to the insertion part of the code. A duplicate row could not actually be inserted into the database due to the primary key constraint but an error would have been caused and the report thrown away.

Due to an unrelated issue where PHP was configured to display errors the HTTP response code that was returned was `200 OK` so, from the perspective of the testing tool, all requests appeared to have succeeded. The fix was to try checking for an existing row once more if the insertion fails to then be able to increment the count.

4.1.3 Assessment

The system is successful at generating a strict policy for the parts of the website that are visited during the period it is being created. The system is limited in its power as the CSP it produces is very specific to only the reports it was based on. It is up to the website administrator to use their knowledge of the website to make sensible decisions about including more broad sources in the CSP. The system is not effective for creating a safe policy, which is a policy that does not include any of the unsafe tokens. This can happen when the website uses unsafe practices like inline scripts, especially those which are dynamically generated as they will not have a fixed hash which can be used in the CSP to allow the script. When the system produces an unsafe policy, it will be up to the website administrator to improve the policy by replacing the unsafe tokens with a nonce token and updating the website's source code to include the nonce value with the inline scripts. The nonce is a random number generated per request and only used once which has enough entropy to not be predictable. When using the nonce method for permitting scripts the nonce is included in the CSP and specified in a `nonce` attribute on the HTML element. Although no CSP is worse than an unsafe one, it can be harmful to present an unsafe CSP as a good one as that leads to complacency. It should be the website administrator's goal to improve upon the final CSP the system produces. This CSP is a better starting place than no CSP to produce one that covers all of the content on the website but is strict enough to fulfil the goal of Content Security Policy described earlier which is to be effective at preventing Cross-Site Scripting and other related attacks.

4.2 Reflection

Another time it would be beneficial to store policies as a tree, the root node would have children which are directives and directives would have values. This structure would be easier to modify when adding new values for a directive and new directives. While in a way I did use the tree structure for the form that allows the selection of changes to apply, these were inserted into the string by searching for the substring of the directive name to insert new values in the right place. When a directive has a value of `'none'` this must be removed when adding another value. This is very complex and brittle. If the administrator customises the policy and slightly changes some syntax the automatic insertion would be completely wrong and the policy would not conform to the correct grammar, whereas a pre-order traversal of the tree would give the policy as a string and give the benefit of easily removing and adding nodes without potentially disrupting the rest of the policy.

If policies were stored in the tree structure in the database a relation could be added to the value records corresponding to the report that was used to generate the suggestion. This would massively help with auditing the final policy produced to give insight into why a value is used in the policy. Without this, in the event a site was complex and had many values and reports it would be hard to justify why a value was added in the first place.

In Helme's *CSP Wizard* each suggestion can either be allowed or blocked. Allowed suggestions are added to the policy and blocked suggestions are stored. Any future suggestion that appears in the blocked list are suppressed from being shown to the user again. I did not implement this but I consider it to be particularly helpful as reports can be generated when users have plugins in their web browser as some insert scripts and other elements into the page which would be rightfully blocked by the policy. In addition, a malicious party may flood you with reports for their scripts in order to get you to permit them in the policy. This could lead to the administrator accidentally adding them without properly checking whether they should be permitted. It may get to the point that the administrator is so fed up of seeing the reports/suggestions for these scripts and allow them just to make the reports go away, as Helme (2020*a*) speculated may be the cause of a suspicious policy used by *Zoom*. This could be prevented by using Headless Chrome in a similar way to that described in the Execution section except instead of running it on HTML taken from a proxy between the user and application it could be directed to the `document-uri` of incoming reports. The worker could then validate that the same report is generated by Headless Chrome and mark it as trusted. A report being trusted could be shown next to the suggestion to indicate to the administrator the report has been verified. Any suggestions that did not have this marker would need further investigation by the administrator before being applied. This technique should significantly reduce the risk of accidentally including a malicious source in the CSP and lower the number of reports the administrator must investigate before applying the suggested change.

The admin interface does not have any user access control. It is left to the user of the software to secure the interface so that only those who need access can. This can be done by having the admin site listen on a localhost port and use port forwarding so only those with SSH access can use the admin interface. Alternatively, it can be done using Transport Layer Security (TLS) Client Authentication with an existing Public Key Infrastructure (PKI) to manage access. Or more simply, setting up HTTP Basic Authentication on the web server (as well as serving it using HTTPS only). It would be useful to extend the system with user accounts to manage access control along with supporting multiple servers so that several different websites can be managed. Users could be assigned to the different sites within the system and only have access to the reports for the site they manage. This would be particularly helpful in a large enterprise where different people are responsible for different websites within the organisation. The main infrastructure department could set up the system and delegate to site owners to use the system to create a CSP for their own sites.

5 Conclusion

While the system is simple to use by building the CSP with an iterative approach, its success is limited to the reports that are sent to it. Without good coverage of the website it is possible to miss resources that should be permitted. Due to the system's 'dumb' nature of basic analysis of reports it is heavily reliant on the knowledge and competence of the website administrator using it. The system does not know the type of content possible on the website and cannot extrapolate out to a more permissive change using a wildcard.

The system produced in this project is not to be used as the final or only part of curating a CSP for a website; it is one step in an ongoing process. It will not be able to produce perfect results and additional work will need to be carried out by the administrator after using it, not only for updating the CSP when the websites source code changes but also for narrowing the permissiveness to a safer policy that does not use the unsafe values. For example, using a nonce based solution for permitting inline scripts.

To conclude, the system successfully assists administrators with creating a CSP for a website they operate and so reduces the difficulty of producing a CSP for an established website, therefore allowing increased adoption of the Content Security Policy header.

References

BBC News (2018), ‘Digital cash hack hits government websites’, *BBC News* . [online] Available at: https://www.bbc.com/news/technology-43025788 [Accessed 20 Feb. 2020].

Caddy Web Server (n.d.), ‘Caddy 2 - The Ultimate Server with Automatic HTTPS’. [online] Available at: https://caddyserver.com [Accessed 24 Mar. 2020].

Cellan-Jones, R. (2018), ‘Suspect code behind BA hack ‘found”, *BBC News* . [online] Available at: https://www.bbc.com/news/technology-45481976 [Accessed 22 Mar. 2020].

Google Developers (n.d.), ‘Puppeteer | Tools for Web Developers’. [online] Available at: https://developers.google.com/web/tools/puppeteer [Accessed 27 Mar. 2020].

Helme, S. (2018), ‘Introducing the CSP Wizard on Report URI’. [online] Available at: https://scotthelme.co.uk/report-uri-csp-wizard/ [Accessed 29 Oct. 2019].

Helme, S. (2019), ‘Top 1 Million Analysis - September 2019’. [online] Available at: https://scotthelme.co.uk/top-1-million-analysis-september-2019/ [Accessed 28 Feb. 2020].

Helme, S. (2020*a*), ‘My best guess is that they were receiving reports for these sources due to compromised clients and the best way to stop those reports is to allow those sources in the CSP.’, *Twitter* . [online] Available at: https://twitter.com/Scott_Helme/status/1242548621595291648 [Accessed 26 Mar. 2020].

Helme, S. (2020*b*), ‘Top 1 Million Analysis - March 2020’. [online] Available at: https://scotthelme.co.uk/top-1-million-analysis-march-2020/ [Accessed 24 Mar. 2020].

Information Commissioner’s Office (2019), ‘What we do’. [online] Available at: https://ico.org.uk/about-the-ico/what-we-do/ [Accessed 20 Feb. 2020].

Load Impact (n.d.), ‘loadimpact/k6’, *GitHub* . [online] Available at: https://github.com/loadimpact/k6 [Accessed 30 Mar. 2020].

MDN Web Docs (n.d.), ‘Subresource Integrity’. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity [Accessed 22 Mar. 2020].

MyBB Documentation (n.d.), ‘Setting up HTTPS’. [online] Available at: https://docs.mybb.com/1.8/administration/security/https/#security-headers [Accessed 27 Mar. 2020].

nginx docs (n.d.), ‘Module ngx_http_upstream_module’. [online] Available at: http://nginx.org/en/docs/http/ngx_http_upstream_module.html [Accessed 24 Mar. 2020].

Report URI (n.d.), ‘Products - Content Security Policy’. [online] Available at: https://report-uri.com/products/content_security_policy [Accessed 17 Mar. 2020].

Stamm, S., Sterne, B. & Markham, G. (2010), Reining in the Web with Content Security Policy, *in* ‘Proceedings of the 19th International Conference on World Wide Web’, ACM, pp. 921–930.

The OWASP Foundation (2017*a*), ‘OWASP Top 10 - 2017’, p. 7. [online] Available at: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf.pdf [Accessed 30 Nov. 2019].

The OWASP Foundation (2017*b*), ‘OWASP Top 10 - 2017’. [online] Available at: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf.pdf [Accessed 30 Nov. 2019].

Williams, C. (2018), ‘UK ICO, USCourts.gov... Thousands of websites hijacked by hidden crypto-mining code after popular plugin pwned’, *The Register* . [online] Available at: https://www.theregister.co.uk/2018/02/11/browsealoud_compromised_coinhive/ [Accessed 30 Nov. 2019].

Appendices

A CSP Usage Breakdown

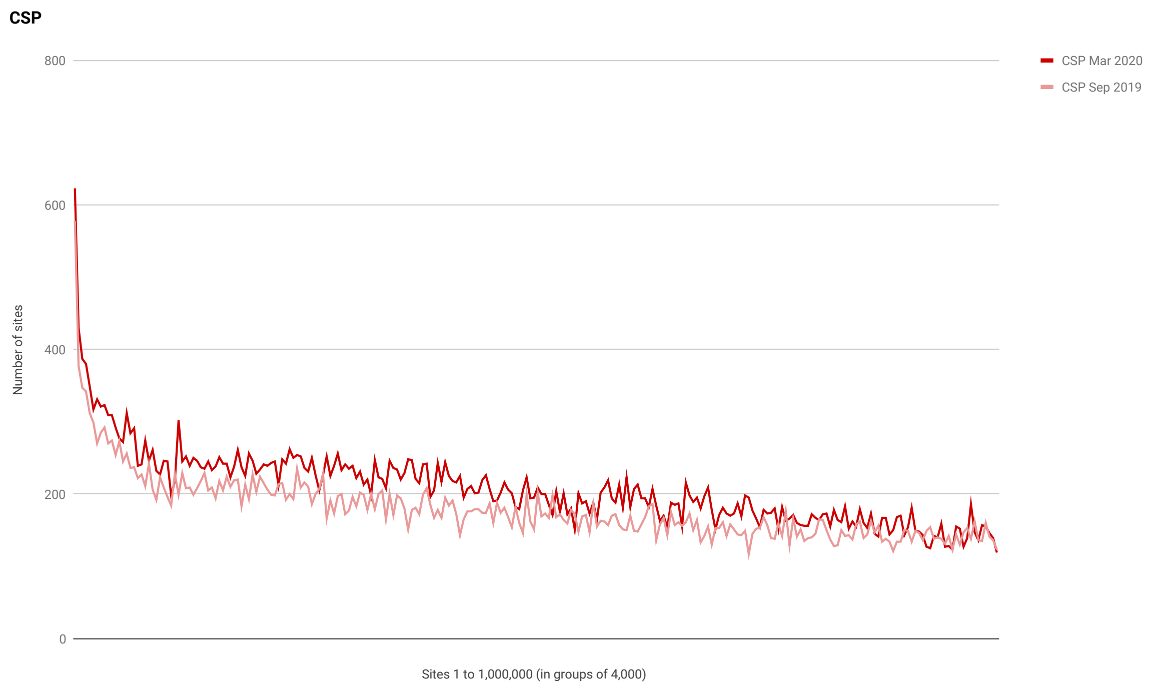


Figure A.1: Comparison of CSP usage between September 2019 and March 2020 (Helme 2020*b*)

B Algorithm for CSP Refinement

0. CSP set to `default-src 'none'; form-action 'none'; frame-ancestors 'none'`
1. Visitor navigates to site and the user agent generates and sends violation reports
2. Reports saved to database by report collection endpoint
3. Administrator uses dashboard to modify the CSP
    1. Suggested changes are shown where a suggested change is a modification to the CSP which would have prevented the report
    2. Administrator selects suggested changes to apply
    3. CSP is updated to include the changes and set on web server
4. GOTO 1

Figure B.1: General algorithm for refining CSP in iterations

C System Diagrams

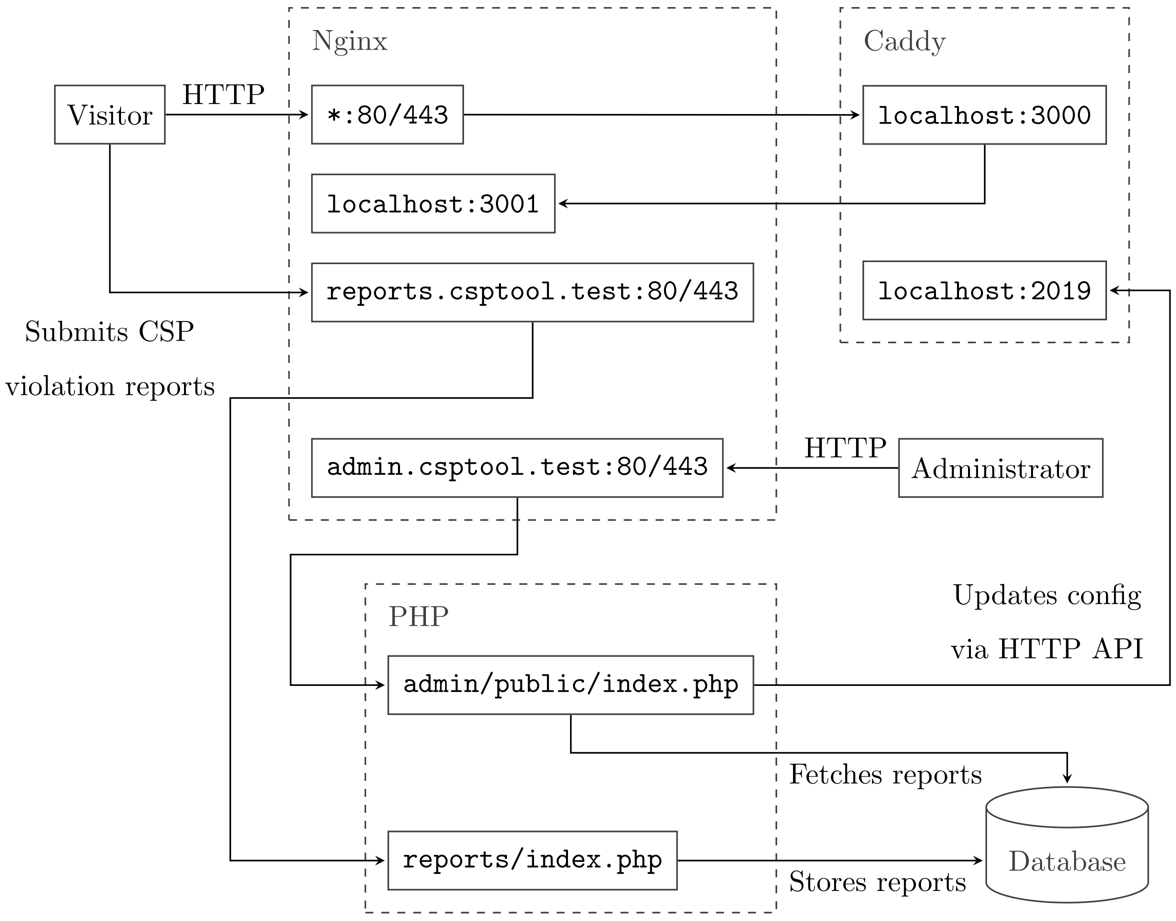


Figure C.1: Whole system diagram

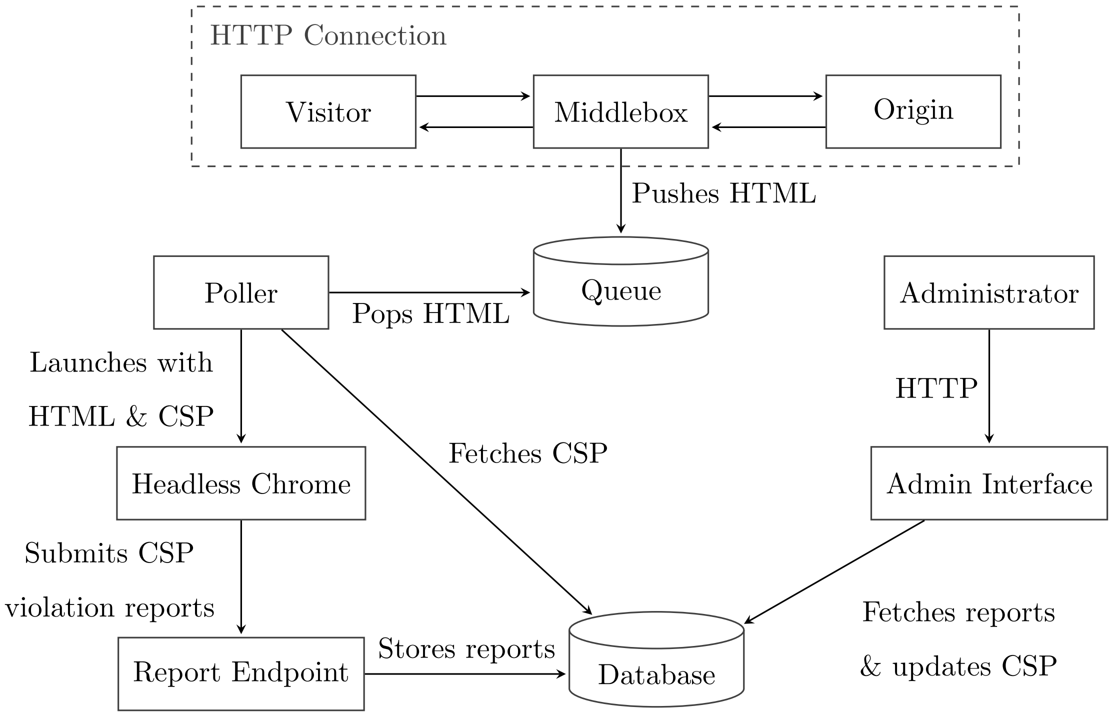


Figure C.2: Headless Chrome system diagram

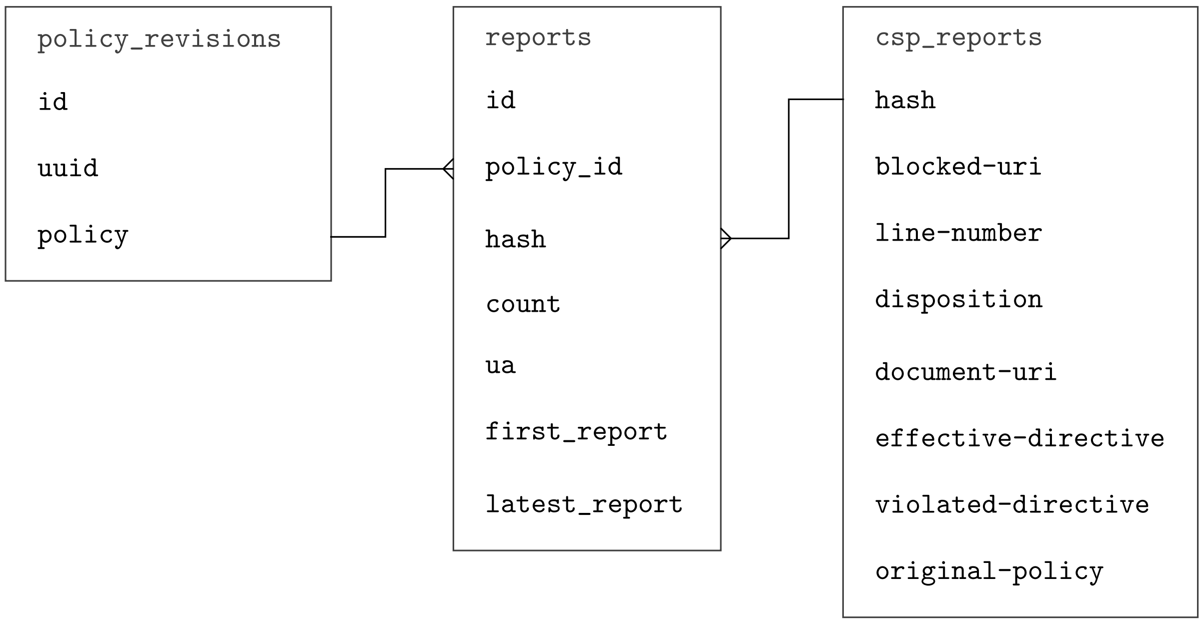


Figure C.3: Entity Relationship Diagram

D Load Test Scripts

```
import http from 'k6/http';
export default function() {
  http.post(
    'http://reports.csptool.test/7a0111c7-5143-40db-be82-0b7cf5eb411a',
    `{"csp-report": {"document-uri": "http://www.csptool.test/", "referrer":
    ↪ "", "violated-directive": "script-src-elem", "effective-directive":
    ↪ "script-src-elem", "original-policy": "default-src 'none'; form-action
    ↪ 'none'; frame-ancestors 'none'; report-uri
    ↪ http://reports.csptool.test/7a0111c7-5143-40db-be82-0b7cf5eb411a",
    ↪ "disposition": "report", "blocked-uri":
    ↪ "http://www.csptool.test/loadtest.js?k6", "status-code": 200,
    ↪ "script-sample": ""}}`
  );
}
```

Figure D.1: Script for running the duplicate report load test

```
import http from 'k6/http';
export default function() {
  http.post(
    'http://reports.csptool.test/7a0111c7-5143-40db-be82-0b7cf5eb411a',
    `{"csp-report": {"document-uri": "http://www.csptool.test/", "referrer":
    ↪ "", "violated-directive": "script-src-elem", "effective-directive":
    ↪ "script-src-elem", "original-policy": "default-src 'none'; form-action
    ↪ 'none'; frame-ancestors 'none'; report-uri
    ↪ http://reports.csptool.test/7a0111c7-5143-40db-be82-0b7cf5eb411a",
    ↪ "disposition": "report", "blocked-uri":
    ↪ "http://www.csptool.test/loadtest.js?v=${__VU}&i=${__ITER}",
    ↪ "status-code": 200, "script-sample": ""}}`
  );
}
```

Figure D.2: Script for running the unique report load test

E Test Results

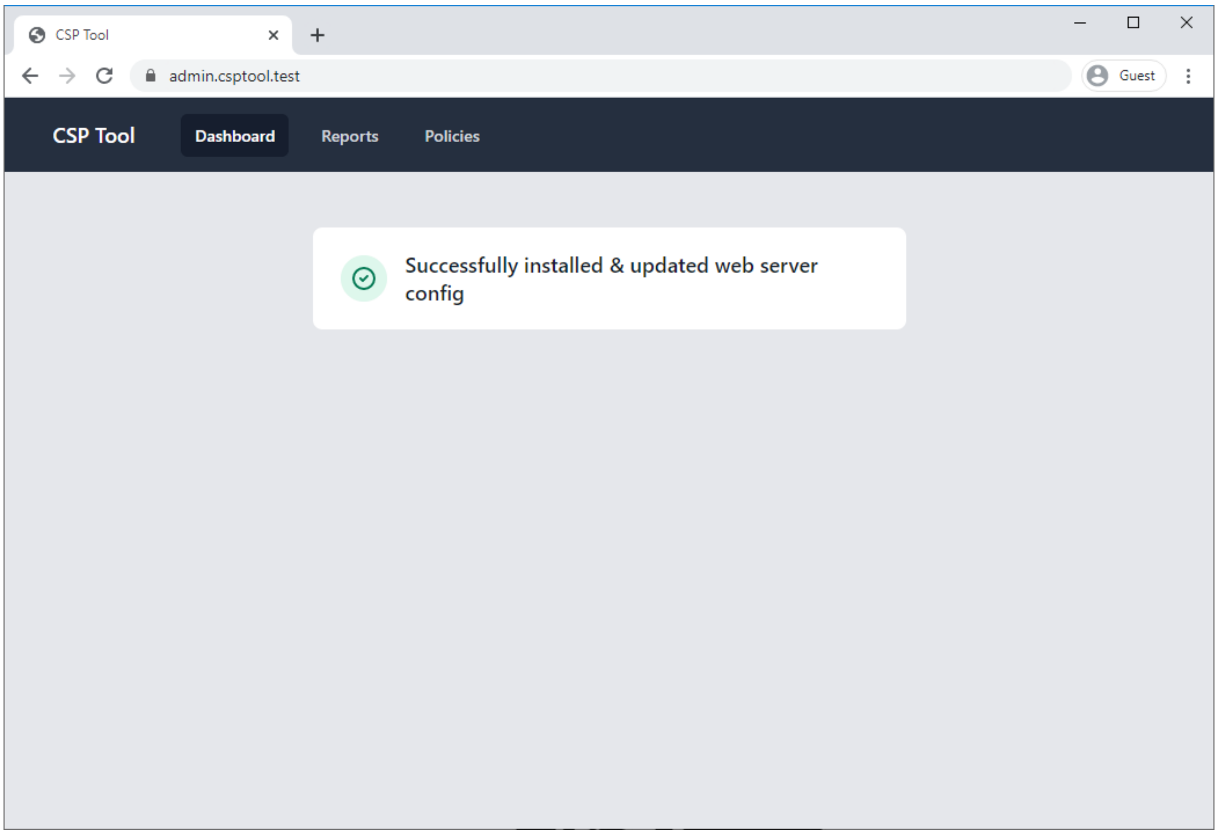
Revision	Policy
#1	`default-src 'none'; form-action 'none'; frame-ancestors 'none'`
#2	`default-src 'none'; form-action 'none'; frame-ancestors 'none';` ↪ `style-src 'unsafe-inline' 'self' https://fonts.googleapis.com;` ↪ `img-src 'self'; script-src 'self'`
#3	`default-src 'none'; form-action 'none'; frame-ancestors 'none';` ↪ `style-src 'unsafe-inline' 'self' https://fonts.googleapis.com;` ↪ `img-src 'self'; script-src 'unsafe-inline' 'self'; font-src` ↪ `https://fonts.gstatic.com`
#4	`default-src 'none'; form-action 'none'; frame-ancestors 'none';` ↪ `style-src 'unsafe-inline' 'self' https://fonts.googleapis.com;` ↪ `img-src https://images.unsplash.com 'self'; script-src` ↪ `'unsafe-inline' 'self'; font-src https://fonts.gstatic.com`
#5	`default-src 'none'; form-action 'none'; frame-ancestors 'none';` ↪ `style-src 'unsafe-inline' 'self' https://fonts.googleapis.com;` ↪ `img-src * 'self'; script-src 'unsafe-inline' 'self'; font-src` ↪ `https://fonts.gstatic.com`
#6	`default-src 'none'; form-action 'none'; frame-ancestors 'none';` ↪ `style-src 'unsafe-inline' 'self' https://fonts.googleapis.com;` ↪ `img-src * 'self'; script-src 'unsafe-inline' 'self'; font-src` ↪ `https://fonts.gstatic.com; frame-src https://www.youtube.com`
#7	`default-src 'none'; form-action 'self'; frame-ancestors 'none';` ↪ `style-src 'unsafe-inline' 'self' https://fonts.googleapis.com;` ↪ `img-src * 'self'; script-src 'unsafe-inline' 'self'; font-src` ↪ `https://fonts.gstatic.com; frame-src https://www.youtube.com`
#8	`default-src 'none'; form-action 'self'; frame-ancestors 'none';` ↪ `style-src 'unsafe-inline' 'self' https://fonts.googleapis.com;` ↪ `img-src * 'self'; script-src 'unsafe-inline' 'self'; font-src` ↪ `https://fonts.gstatic.com; frame-src https://www.youtube.com;` ↪ `connect-src 'self'`
#9	`default-src 'none'; form-action 'self'; frame-ancestors 'self';` ↪ `style-src 'unsafe-inline' 'self' https://fonts.googleapis.com;` ↪ `img-src data: * 'self'; script-src 'unsafe-inline' 'self';` ↪ `font-src https://fonts.gstatic.com; frame-src 'self'` ↪ `https://www.youtube.com; connect-src 'self'`
#10	`default-src 'none'; form-action 'self'; frame-ancestors 'self';` ↪ `style-src 'unsafe-inline' 'self' https://fonts.googleapis.com;` ↪ `img-src data: * 'self'; script-src 'unsafe-eval' 'unsafe-inline'` ↪ `'self'; font-src https://fonts.gstatic.com; frame-src 'self'` ↪ `https://www.youtube.com; connect-src 'self'`

Figure E.1: Policy revisions during testing using MyBB

Revision	Policy
#1	`default-src 'none'; form-action 'none'; frame-ancestors 'none'`
#2	`default-src 'none'; form-action 'none'; frame-ancestors 'none';` `↪ style-src 'self' 'unsafe-inline'; script-src 'unsafe-inline'` `↪ 'self'; font-src data: 'self'; img-src 'self'`
#3	`default-src 'none'; form-action 'self'; frame-ancestors 'none';` `↪ style-src 'self' 'unsafe-inline'; script-src 'unsafe-eval'` `↪ 'unsafe-inline' 'self'; font-src data: 'self'; img-src` `↪ *.gravatar.com 'self'; connect-src 'self'`
#4	`default-src 'none'; form-action 'self'; frame-ancestors 'none';` `↪ style-src https://fonts.googleapis.com 'self' 'unsafe-inline';` `↪ script-src 'unsafe-eval' 'unsafe-inline' 'self'; font-src` `↪ https://fonts.gstatic.com data: 'self'; img-src data:` `↪ *.gravatar.com 'self'; connect-src 'self'`
#5	`default-src 'none'; form-action 'self'; frame-ancestors 'self';` `↪ style-src https://fonts.googleapis.com 'self' 'unsafe-inline';` `↪ script-src 'unsafe-eval' 'unsafe-inline' 'self'; font-src` `↪ https://fonts.gstatic.com data: 'self'; img-src data:` `↪ *.gravatar.com 'self'; connect-src 'self'; frame-src 'self'`
#6	`default-src 'none'; form-action 'self'; frame-ancestors 'self';` `↪ style-src https://fonts.googleapis.com 'self' 'unsafe-inline';` `↪ script-src 'unsafe-eval' 'unsafe-inline' 'self'; font-src` `↪ https://fonts.gstatic.com data: 'self'; img-src * data:` `↪ *.gravatar.com 'self'; connect-src 'self'; frame-src 'self'`
#7	`default-src 'none'; form-action 'self'; frame-ancestors 'self';` `↪ style-src https://fonts.googleapis.com 'self' 'unsafe-inline';` `↪ script-src 'unsafe-eval' 'unsafe-inline' 'self'; font-src` `↪ https://fonts.gstatic.com data: 'self'; img-src * data:` `↪ *.gravatar.com 'self'; connect-src 'self'; frame-src` `↪ https://www.youtube.com 'self'`

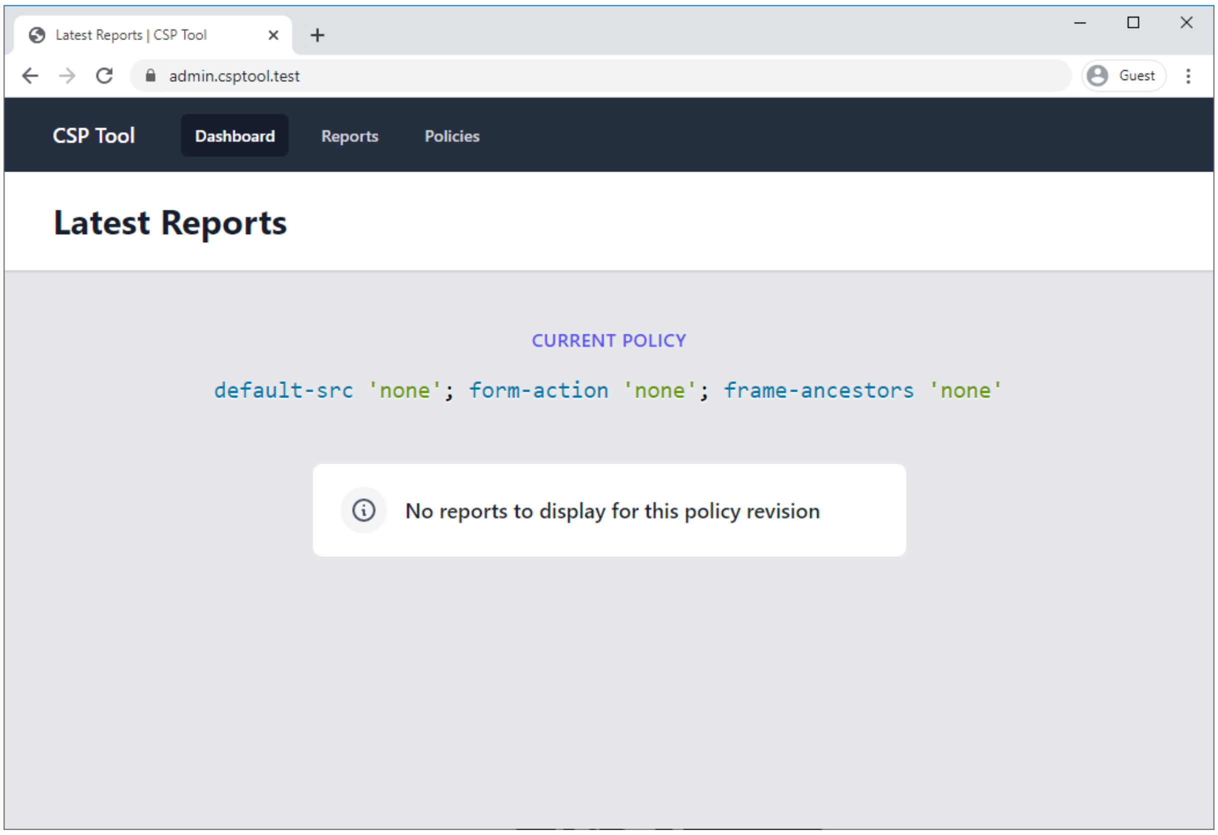
Figure E.2: Policy revisions during testing using WordPress

F Screenshots



Successfully installed & updated web server config

Figure F.1: Install screen shown when first loading the admin interface



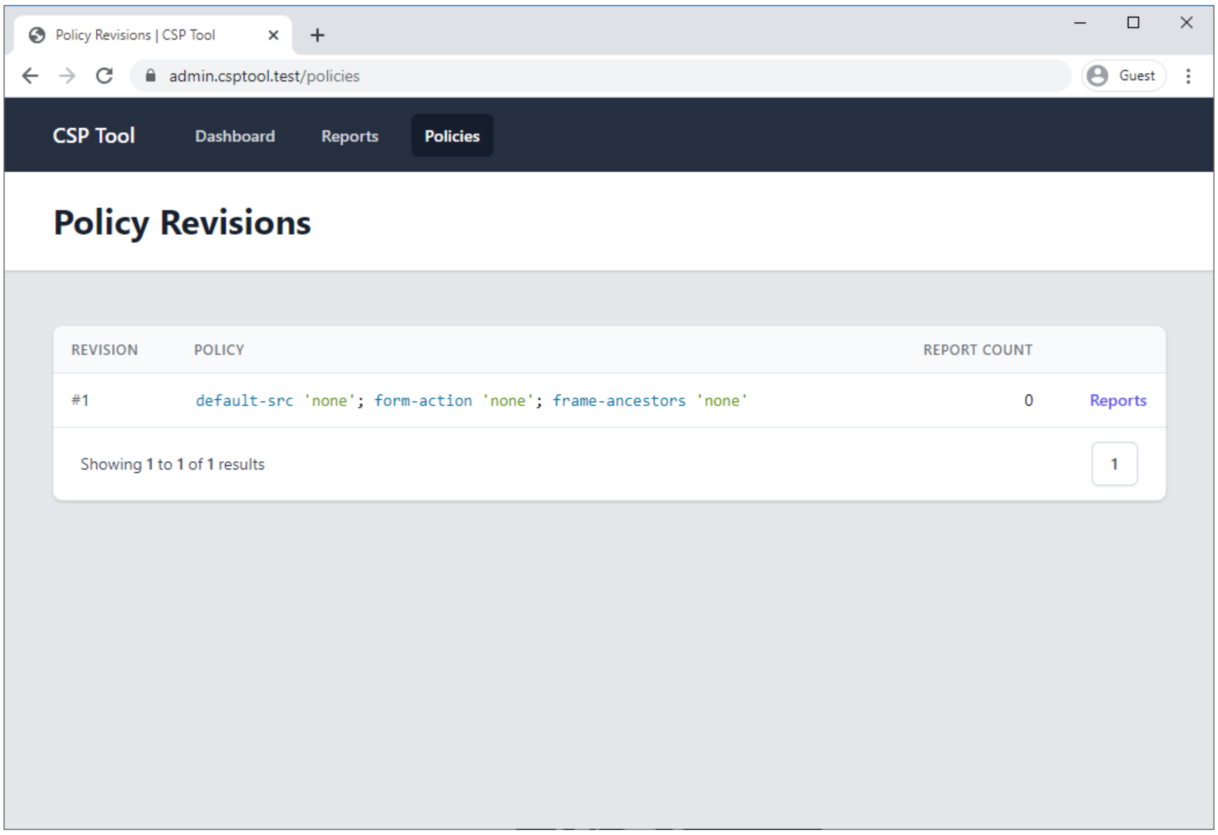
Latest Reports

CURRENT POLICY

```
default-src 'none'; form-action 'none'; frame-ancestors 'none'
```

No reports to display for this policy revision

Figure F.2: Latest Reports screen before any reports have been received

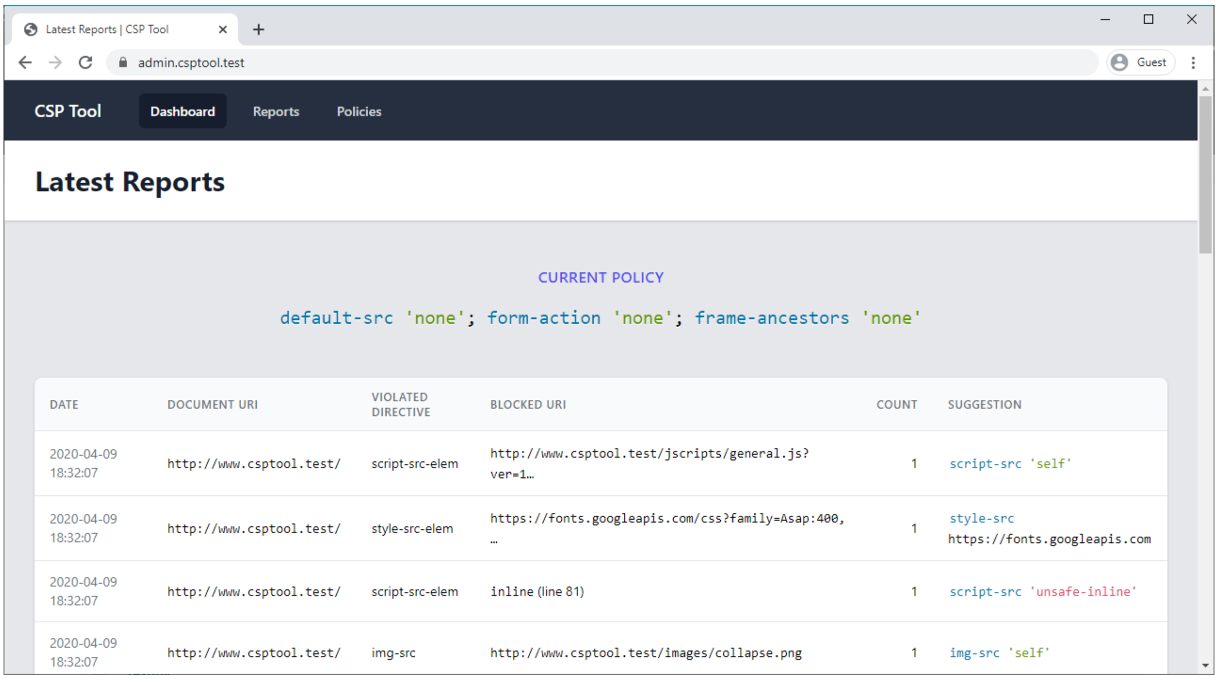


Policy Revisions

REVISION	POLICY	REPORT COUNT	
#1	`default-src 'none'; form-action 'none'; frame-ancestors 'none'`	0	Reports

Showing 1 to 1 of 1 results

Figure F.3: Policy Revisions screen before any changes have been made



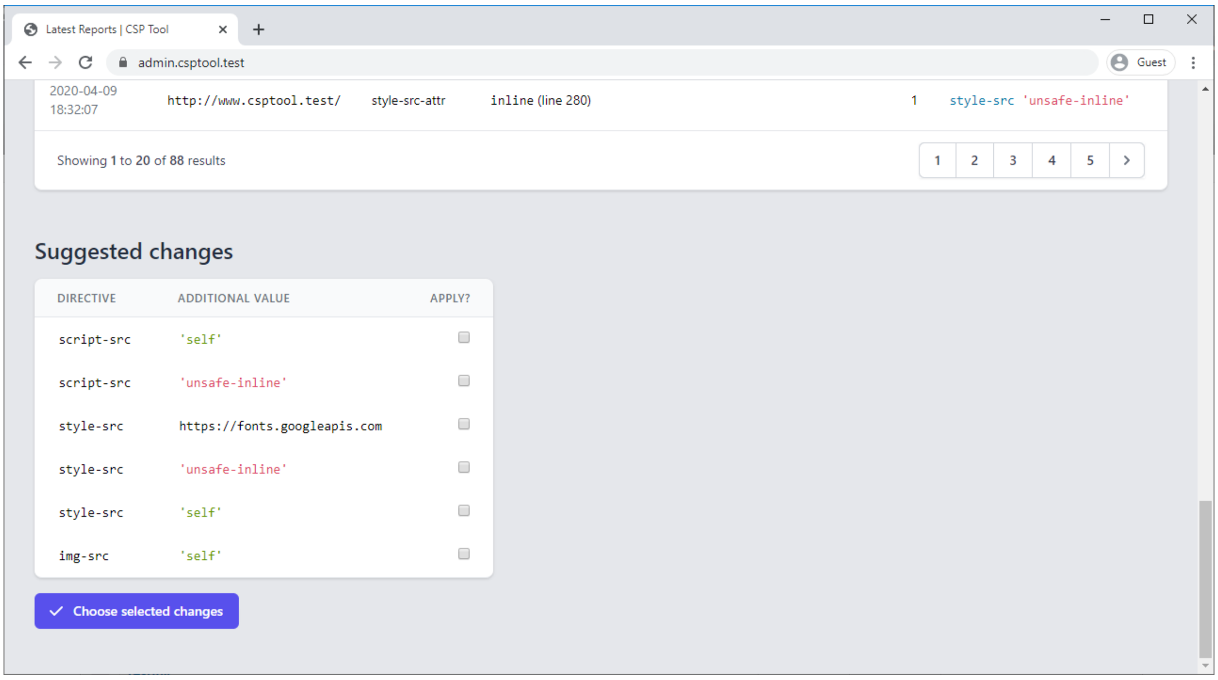
Latest Reports

CURRENT POLICY

`default-src 'none'; form-action 'none'; frame-ancestors 'none'`

DATE	DOCUMENT URI	VIOLATED DIRECTIVE	BLOCKED URI	COUNT	SUGGESTION
2020-04-09 18:32:07	`http://www.csptool.test/`	script-src-elem	`http://www.csptool.test/jscripts/general.js?ver=1…`	1	`script-src 'self'`
2020-04-09 18:32:07	`http://www.csptool.test/`	style-src-elem	`https://fonts.googleapis.com/css?family=Asap:400,…`	1	`style-src https://fonts.googleapis.com`
2020-04-09 18:32:07	`http://www.csptool.test/`	script-src-elem	inline (line 81)	1	`script-src 'unsafe-inline'`
2020-04-09 18:32:07	`http://www.csptool.test/`	img-src	`http://www.csptool.test/images/collapse.png`	1	`img-src 'self'`

Figure F.4: Latest Reports after a user has browsed the website and reports have been received



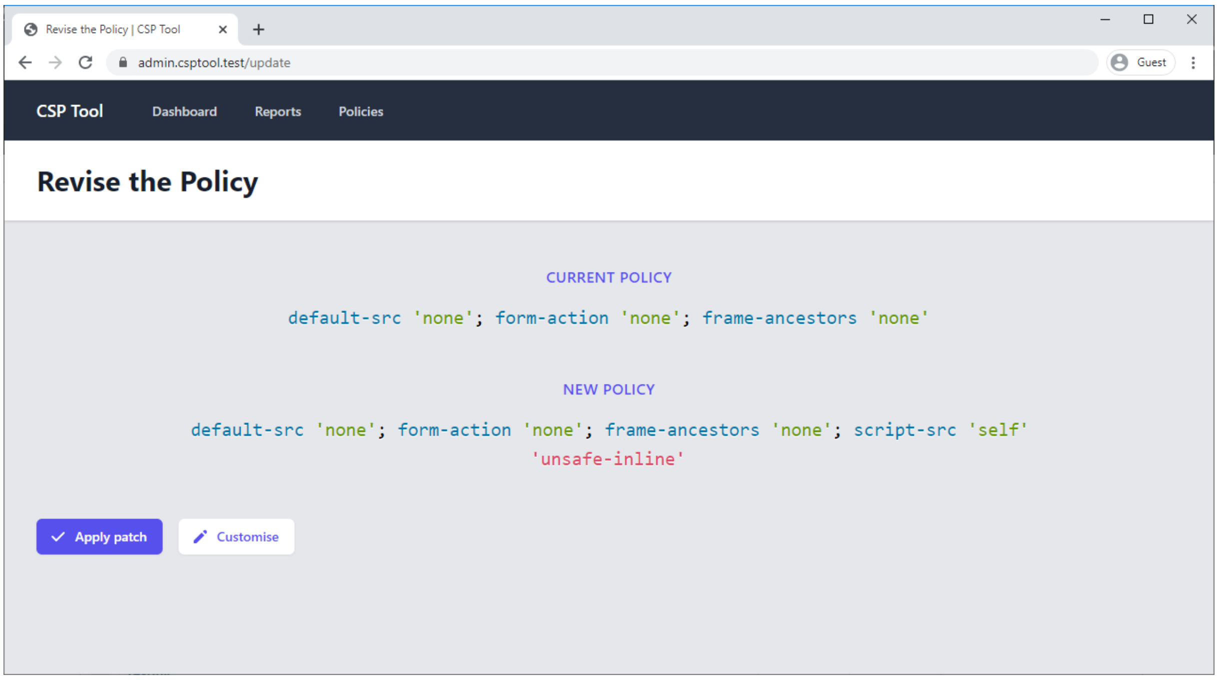
2020-04-09 18:32:07	`http://www.csptool.test/`	style-src-attr	`inline` (line 280)	1	`style-src 'unsafe-inline'`

Showing 1 to 20 of 88 results

Suggested changes

DIRECTIVE	ADDITIONAL VALUE	APPLY?
`script-src`	`'self'`	☐
`script-src`	`'unsafe-inline'`	☐
`style-src`	`https://fonts.googleapis.com`	☐
`style-src`	`'unsafe-inline'`	☐
`style-src`	`'self'`	☐
`img-src`	`'self'`	☐

Figure F.5: Suggested changes for the reports at the bottom of screen shown in previous figure



Revise the Policy

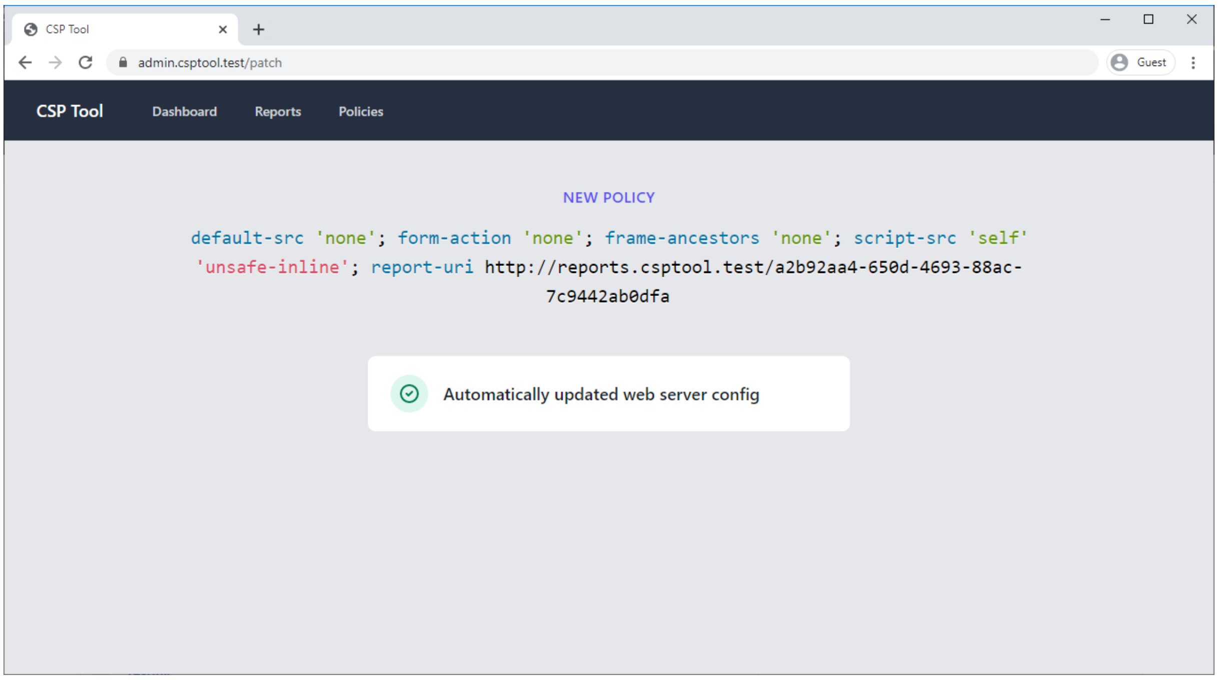
CURRENT POLICY

```
default-src 'none'; form-action 'none'; frame-ancestors 'none'
```

NEW POLICY

```
default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self'
'unsafe-inline'
```

Figure F.6: Revise policy screen after selecting the script-src suggestions

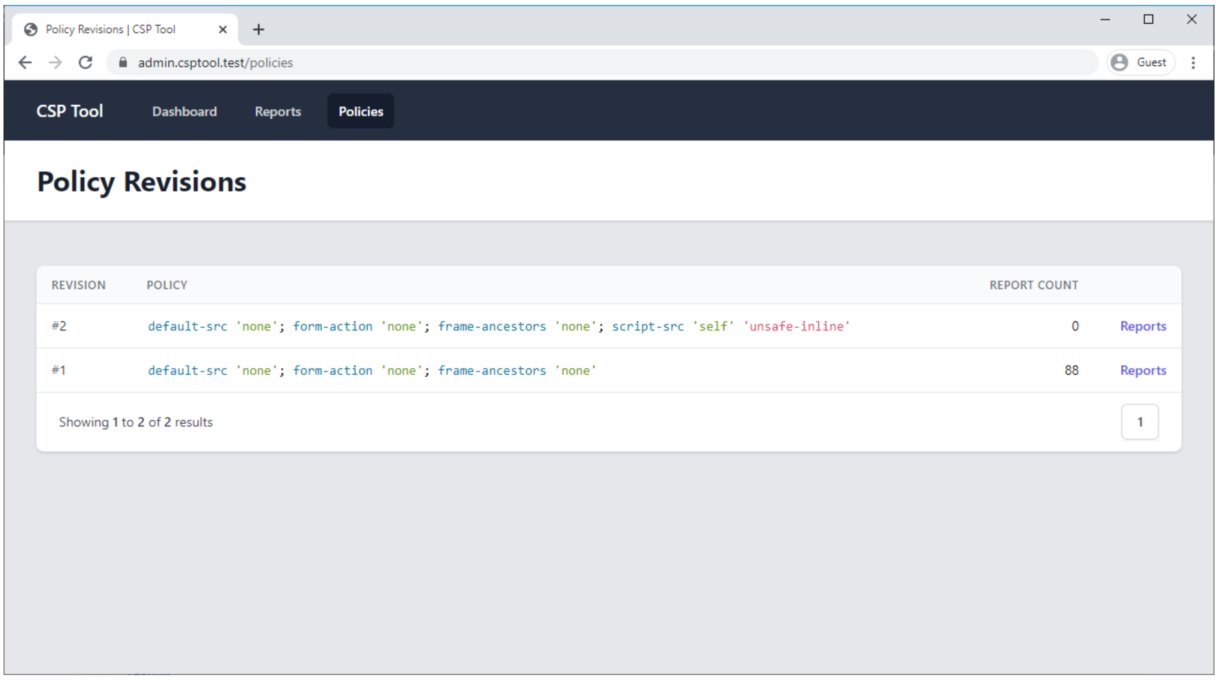


NEW POLICY

```
default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self'
'unsafe-inline'; report-uri http://reports.csptool.test/a2b92aa4-650d-4693-88ac-
7c9442ab0dfa
```

Automatically updated web server config

Figure F.7: Policy updated confirmation screen



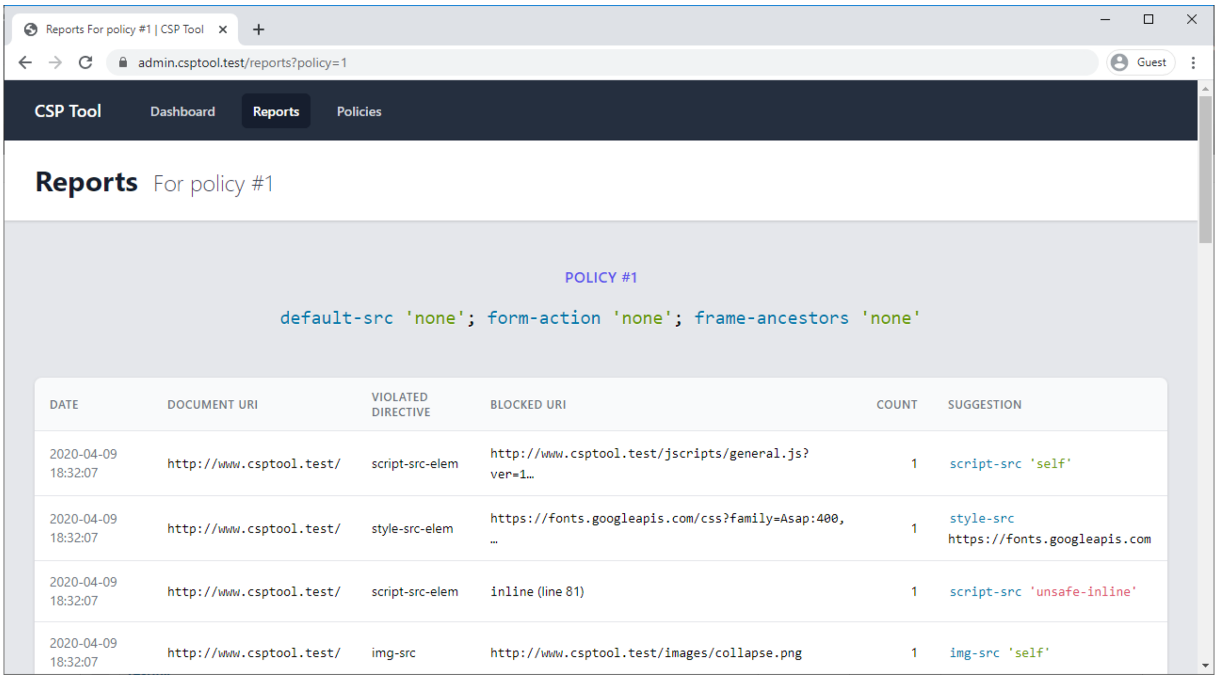
Policy Revisions

REVISION	POLICY	REPORT COUNT	
#2	`default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self' 'unsafe-inline'`	0	Reports
#1	`default-src 'none'; form-action 'none'; frame-ancestors 'none'`	88	Reports

Showing 1 to 2 of 2 results

1

Figure F.8: Policy revisions screen with the new revision



Reports For policy #1 | CSP Tool
admin.csptool.test/reports?policy=1
Guest
CSP Tool
Dashboard
Reports
Policies

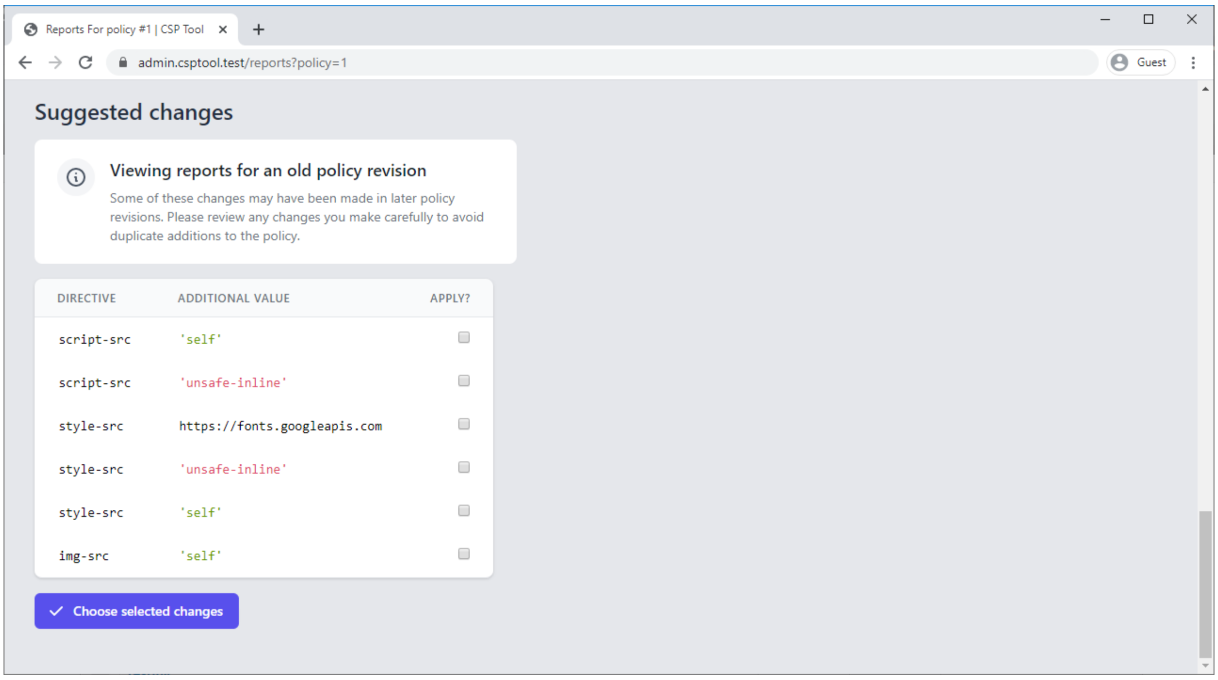
Reports For policy #1

POLICY #1

`default-src 'none'; form-action 'none'; frame-ancestors 'none'`

DATE	DOCUMENT URI	VIOLATED DIRECTIVE	BLOCKED URI	COUNT	SUGGESTION
2020-04-09 18:32:07	http://www.csptool.test/	script-src-elem	http://www.csptool.test/jscripts/general.js?ver=1…	1	script-src 'self'
2020-04-09 18:32:07	http://www.csptool.test/	style-src-elem	https://fonts.googleapis.com/css?family=Asap:400, …	1	style-src https://fonts.googleapis.com
2020-04-09 18:32:07	http://www.csptool.test/	script-src-elem	inline (line 81)	1	script-src 'unsafe-inline'
2020-04-09 18:32:07	http://www.csptool.test/	img-src	http://www.csptool.test/images/collapse.png	1	img-src 'self'

Figure F.9: Viewing reports for previous policy revision



Reports For policy #1 | CSP Tool
admin.csptool.test/reports?policy=1
Guest

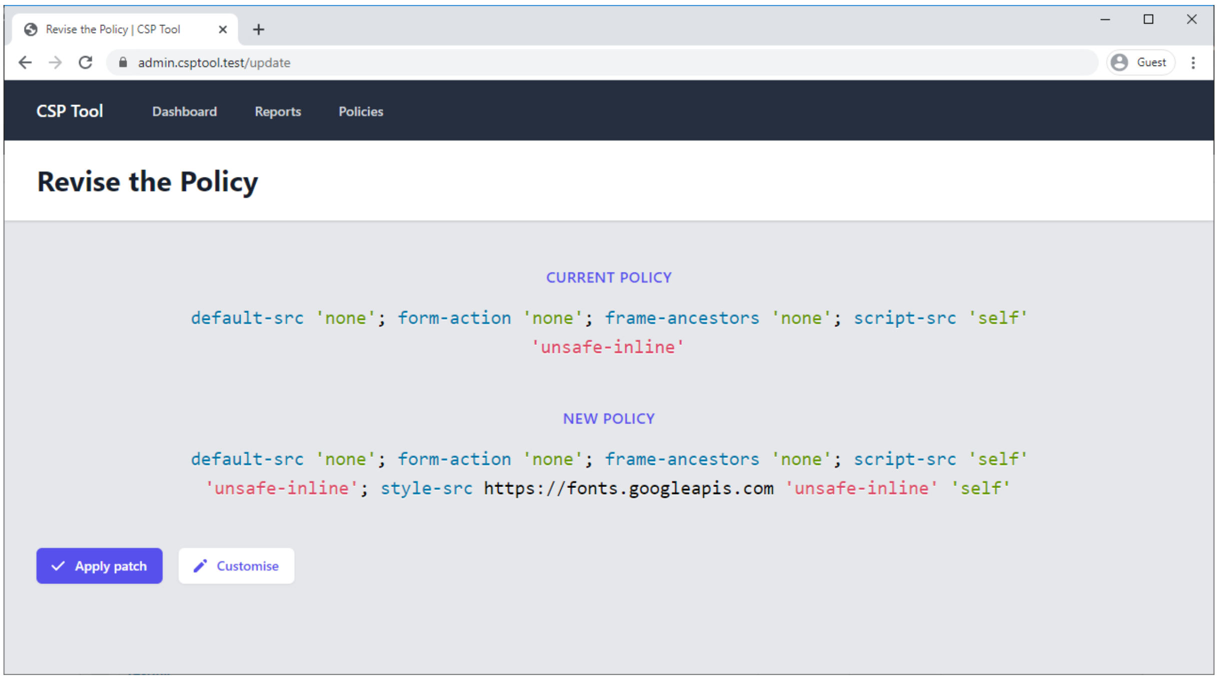
Suggested changes

Viewing reports for an old policy revision

Some of these changes may have been made in later policy revisions. Please review any changes you make carefully to avoid duplicate additions to the policy.

DIRECTIVE	ADDITIONAL VALUE	APPLY?
script-src	'self'	☐
script-src	'unsafe-inline'	☐
style-src	https://fonts.googleapis.com	☐
style-src	'unsafe-inline'	☐
style-src	'self'	☐
img-src	'self'	☐

Figure F.10: Viewing suggestions for the previous policy revision



Revise the Policy

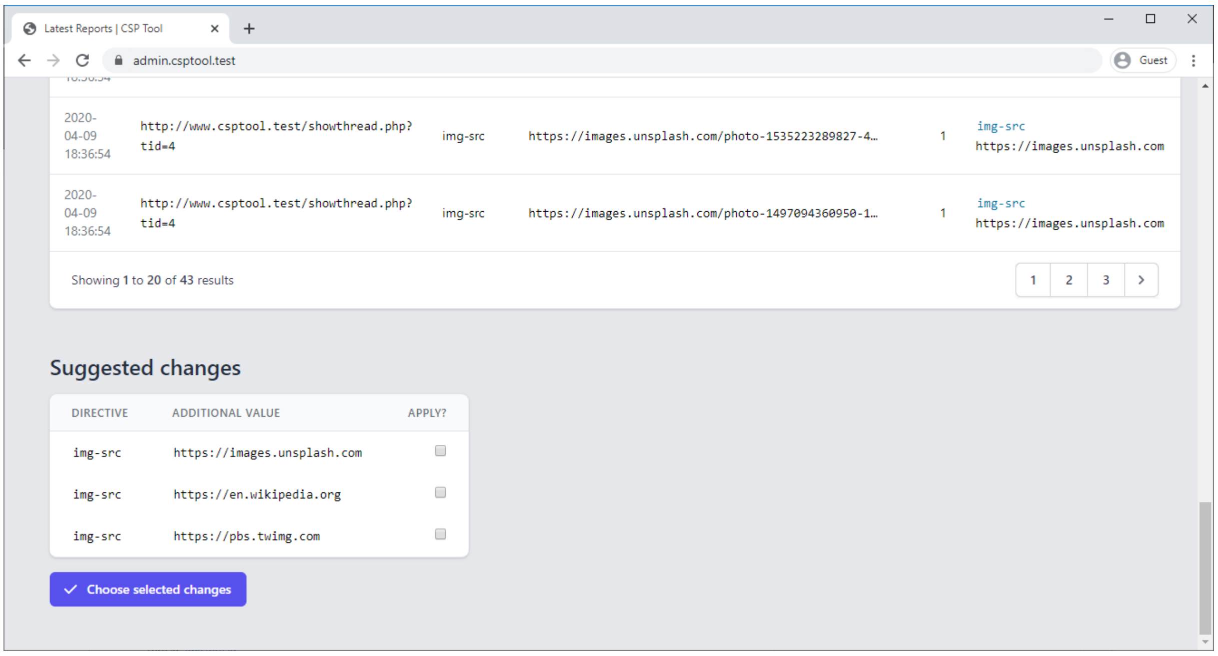
CURRENT POLICY

```
default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self'
'unsafe-inline'
```

NEW POLICY

```
default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self'
'unsafe-inline'; style-src https://fonts.googleapis.com 'unsafe-inline' 'self'
```

Figure F.11: Revising the policy using the style-src suggestions from the previous revision



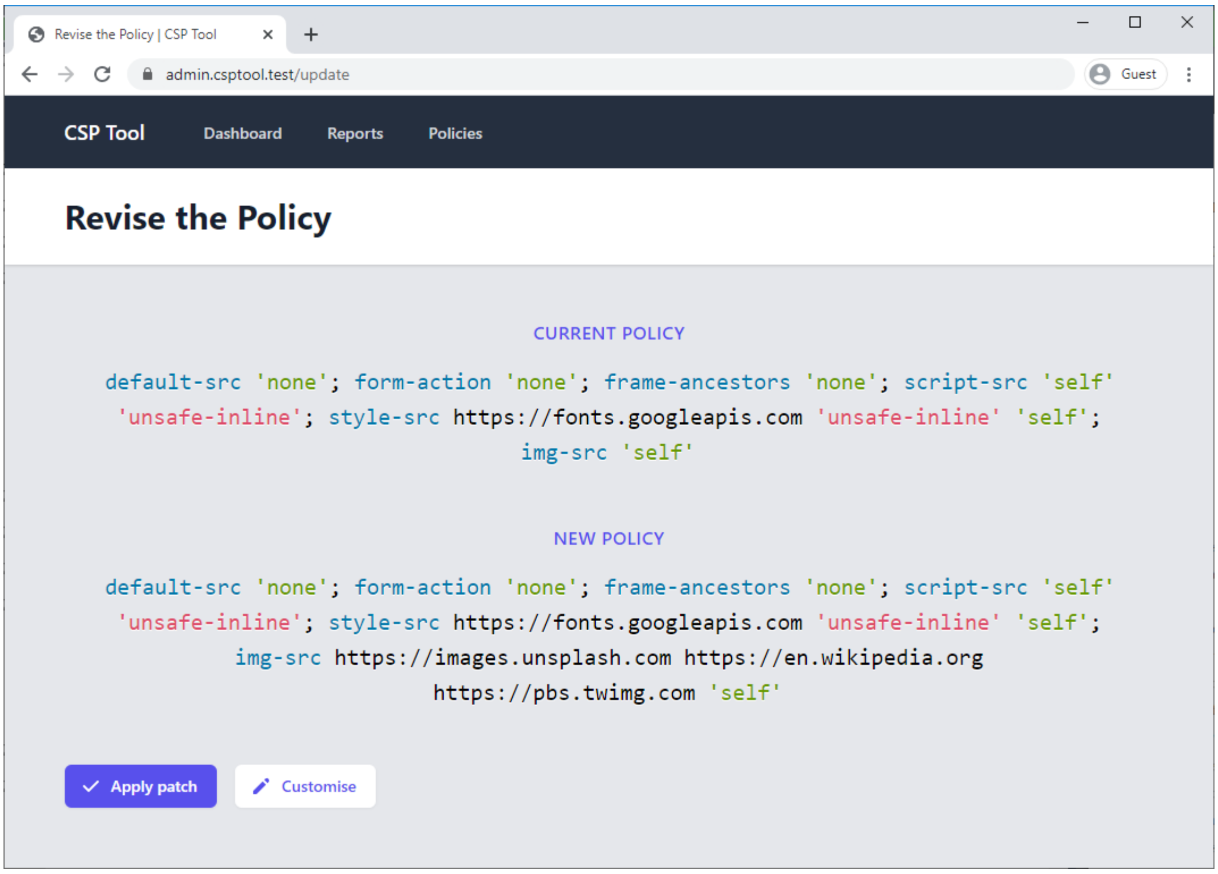
2020-04-09 18:36:54	http://www.csptool.test/showthread.php?tid=4	img-src	https://images.unsplash.com/photo-1535223289827-4…	1	img-src https://images.unsplash.com
2020-04-09 18:36:54	http://www.csptool.test/showthread.php?tid=4	img-src	https://images.unsplash.com/photo-1497094360950-1…	1	img-src https://images.unsplash.com

Showing 1 to 20 of 43 results

Suggested changes

DIRECTIVE	ADDITIONAL VALUE	APPLY?
img-src	https://images.unsplash.com	☐
img-src	https://en.wikipedia.org	☐
img-src	https://pbs.twimg.com	☐

Figure F.12: Latest Reports screen showing multiple suggested values for img-src after additional browsing



Revise the Policy

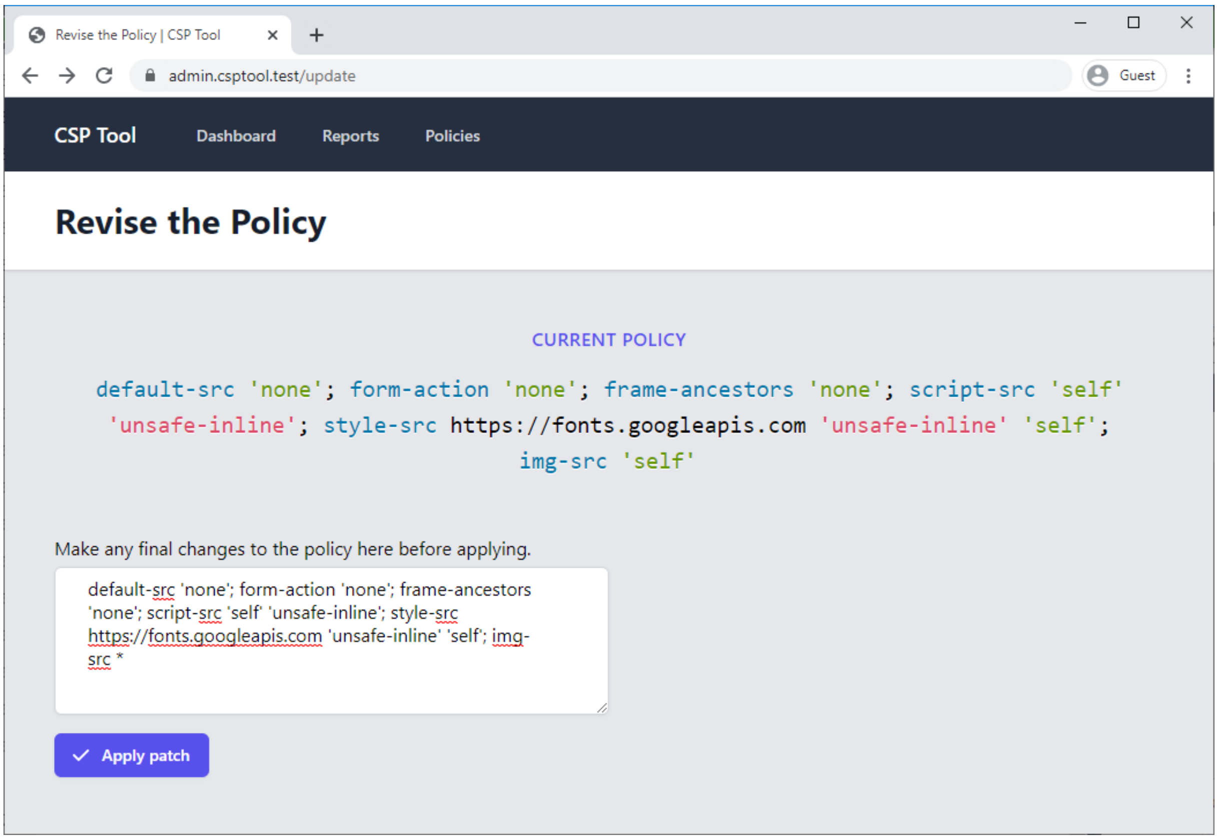
CURRENT POLICY

```
default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self'
'unsafe-inline'; style-src https://fonts.googleapis.com 'unsafe-inline' 'self';
img-src 'self'
```

NEW POLICY

```
default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self'
'unsafe-inline'; style-src https://fonts.googleapis.com 'unsafe-inline' 'self';
img-src https://images.unsplash.com https://en.wikipedia.org
https://pbs.twimg.com 'self'
```

Figure F.13: Revise policy screen after selecting the img-src suggestions



Revise the Policy

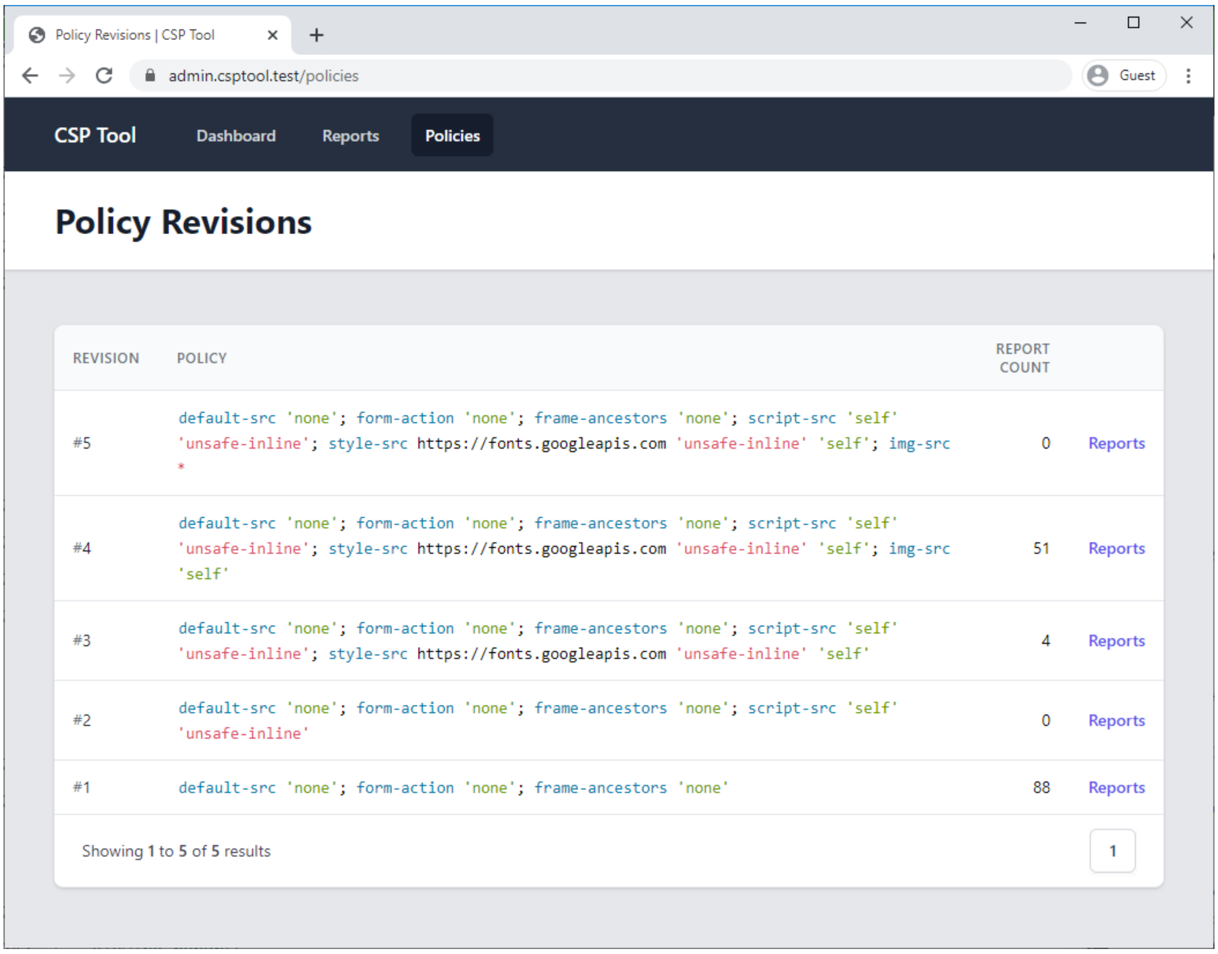
CURRENT POLICY

```
default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self'
'unsafe-inline'; style-src https://fonts.googleapis.com 'unsafe-inline' 'self';
img-src 'self'
```

Make any final changes to the policy here before applying.

```
default-src 'none'; form-action 'none'; frame-ancestors
'none'; script-src 'self' 'unsafe-inline'; style-src
https://fonts.googleapis.com 'unsafe-inline' 'self'; img-
src *
```

Figure F.14: Revise policy screen after clicking customise button and changing the image sources to just a wildcard (*)



Policy Revisions

REVISION	POLICY	REPORT COUNT	
#5	`default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self' 'unsafe-inline'; style-src https://fonts.googleapis.com 'unsafe-inline' 'self'; img-src *`	0	Reports
#4	`default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self' 'unsafe-inline'; style-src https://fonts.googleapis.com 'unsafe-inline' 'self'; img-src 'self'`	51	Reports
#3	`default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self' 'unsafe-inline'; style-src https://fonts.googleapis.com 'unsafe-inline' 'self'`	4	Reports
#2	`default-src 'none'; form-action 'none'; frame-ancestors 'none'; script-src 'self' 'unsafe-inline'`	0	Reports
#1	`default-src 'none'; form-action 'none'; frame-ancestors 'none'`	88	Reports

Showing 1 to 5 of 5 results

1

Figure F.15: Policy Revisions screen after that revision