Bachelor of Science in Engineering: Computer Security
June 2021

Assessing HTTP Security Header implementations

A study of Swedish government agencies' first line of defense against XSS and client-side supply chain attacks

Ludwig Johnson
Lukas Mårtensson

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Bachelor of Science in Engineering: Computer Security. The thesis is equivalent to 10 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author(s):
Ludwig Johnson
E-mail: lujh17@student.bth.se

Lukas Mårtensson
E-mail: lumb14@student.bth.se

University advisor:
University lecturer Nurul Momen
Department of Computer Science

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Background. Security on the web is a fundamental requirement as it becomes a bigger part of society and more information than ever is shared over it. However, as recent incidents have shown, even Swedish government agencies have had issues with their website security. One such example is when a client-side supply chain for several governmental websites was hacked and malicious javascript was subsequently found on several governmental websites. Hence this study is aimed at assessing the security of Swedish government agencies' first line of defense against attacks like XSS and client-side supply chain.
Objectives. The main objective of the thesis is to assess the first line of defense, namely HTTP security headers, of Swedish government agency websites. In addition, collecting statistics of what HTTP security headers are actually used by Swedish government agencies today were gathered for comparison with similar studies.
Methods. To fulfill the objectives of the thesis, a scan of all Swedish government agency websites, found on *Myndighetsregistret*, was completed and an algorithm was developed to assess the implementation of the security features. In order to facilitate tunable assessments for different types of websites, the algorithm has granular weights that can be assigned to each test to make the algorithm more generalized.
Results. The results show a low overall implementation rate of the various HTTP security headers among the Swedish government agency websites. However, when compared to similar studies, the adoption of all security features are higher among the Swedish government agency websites tested in this thesis.
Conclusions. Previous tools/studies mostly checked if a header was implemented or not. With our algorithm, the strength of the security header implementation is also assessed. According to our results, there is a significant difference between if a security header has been implemented, and if it has been implemented well, and provides adequate security. Therefore, traditional tools for testing HTTP security headers may be inefficient and misleading.

Keywords: http, security headers, web, csp, algorithm

Acknowledgments

We would like to thank our supervisor Nurul Momen for his guidance, support and ideas to help us with our thesis. We also want to thank Cris Staicu at CISPA – Helmholtz Center for Information Security for ideas and feedback for our thesis.

Contents

Chapter 1

Introduction

As our society is moving more and more towards the digital sphere, the importance of online security and privacy increases significantly. During the currently ongoing global pandemic, even more of our critical information is communicated over the internet from the likes of *Folkhälsomyndigheten* (FoHM) and *Myndigheten för Samhällsskydd och Beredskap* (MSB) in Sweden. These government agencies might be targets for adversaries that want access to sensitive/secret information or to manipulate the public information the websites provide. Therefore, the security of these websites is very important.

1.1 Motivation

Many websites often rely to some extent on external parties to provide additional functionality, like analytics, accessibility tools (such as text-to-speech and increase font size) and similar features. The functionality is often included by having the visitors web browsers load scripts from external third-party servers. This is a part of the client-side supply chain and the risks are high if the external party is breached. A recent example of a successful client-side supply chain attack was when a text-to-speech company was hacked and malicious code was injected into all websites using their tool [43]. Some government agencies were affected by this attack in Sweden, such as the Swedish police, the Swedish defense materiel administration and the Swedish consumer agency [35]. One way to defend against such client-side supply chain attacks (and many other types of attacks) is to use various HTTP (hypertext transfer protocol) security headers [1] [27]. By correctly configuring these headers, web browsers accessing the website is instructed on what is and is not allowed to run in the client browser. This in turn, can prevent a lot of security incidents where code is injected by a malicious entity in a way that is not desired.

HTTP headers are sent as part of the response to when a browser connects to the website. They contain information about the server but also instructions for the browser (this can include instructions about caching, compression algorithms, etc) on how to do some things. HTTP security headers are headers that instruct the browser of what it can/should and cannot/should not do. This includes from where it may load resources, to only load the website over TLS (HTTPS) or what browser APIs the website is allowed to access (like accessing a microphone/webcam) for example.

This means that if someone manages to inject code on a website (through a comment on a blog post for example), a security header, if configured correctly, can prevent the code from being executed by the browser and therefore protect the users

of said website. HTTP security headers should however act as a first line of defense and be a part of the defense-in-depth strategy by providing more measures to prevent code being injected/displayed to a user on the server side. This is because the various security headers have varying support in different browsers and all users may not get the same protection.

1.2 Aim and objectives

The aim of the study is to evaluate the security benefits of HTTP security headers and how well Swedish government websites implement them in light of recent supply chain attacks. The aims are therefore the following:

- Understand security headers and what they solve in terms of security on the web.
- How they can be used to prevent attacks and specifically XSS and client-side supply chain attacks.
- Develop a method for assessing the security of a website based on its HTTP security headers.
- What HTTP security headers have Swedish government agencies configured on their websites and what it means to their security.

To complete the aims of the study, a literature review of HTTP security headers and a technical survey of Swedish government agency websites will be performed. The aims have been divided into the following actionable objectives:

- Find all HTTP security headers and conduct a literary study to understand their use cases.
- Determine the grading, based on the literature study, of how and how many security headers have been implemented to be able to compare different web sites.
- Determine which Swedish government agency websites to scan.
- Build a tool to scan the HTTP headers and HTML code.
- Scan websites and compile the results.
- Analyze and present the results according to the methodology developed previously.

1.3 Problem statement

The main goal of the research is to find how Swedish government agency websites (SGAWs) make use of HTTP security headers. To better assess if these websites implementation of HTTP security headers are configured correctly, a literature review was performed to more accurately understand what these headers do and how they protect the website and its users.

1.3.1 Research questions

The main goal of the thesis is to answer these three questions:

RQ1: How can we assess the security measures of a website based on the HTTP headers used?

To answer this question, another question has to be answered first:

RQ2: What HTTP security headers exists, how they protect users and how they defend against XSS and client-side supply chain attacks?

Finally, to fully complete the aims of the study, the following question will be answered:

RQ3: What HTTP security headers Swedish government agency websites use to protect their users' security and privacy?

1.4 Scope

Similar research has been done as described in Section 2.2, but this paper has a focus on specifically Swedish government agency websites. While other work primarily look at just the HTTP headers, this paper also takes into account sub-resource integrity directly in the HTML code. The thesis also introduces a method of assessing how well the security of a website is (based on what HTTP Security Headers it has implemented and specifically, how well they have been implemented) and allowing comparisons to be made between different websites.

There are many supply chain aspects to account for, many of which are on the server side (like the web server itself, libraries to server-side code, etc.). This thesis focuses on the security and supply chain on the client-side. Therefore, we assume that the server hosting the website itself is trusted and secured and the thesis does not cover any potential issues on this server in regards to the supply chain. The thesis focuses on securing external and third-party inclusions (like javascript and stylesheets) that the client browser retrieves and how to secure those resources (which are out of control for the server hosting the website).

An issue with running a survey such as this one is that a website may respond differently to different types of clients (desktops, laptops, phones, tablets, etc.) and from where the request originates in the world. There are thousands of combinations of clients/browsers and hundreds of different countries to make requests from. Therefore, it was deemed impossible for the scope of our survey to see if there are any differences in this regard. Therefore, the tests were ran using a common desktop user-agent (Firefox and Chrome browsers) and from Swedish ISPs for our script to connect from and to try and represent a connection from a typical user of SGAWs. To avoid any potential issues with one particular network, two separate Swedish residential ISP networks and the network at the Blekinge Institute of Technology campus (SUNET) were used to run the tests.

1.5 Outline

This thesis is divided into six chapters. The first chapter is an introduction and contains the research questions as well as the scope of the thesis. The second chapter is a background and related works section and contains the results of a literary mapping study as well as technical background on HTTP security headers. The third chapter details the methodology of the technical survey as well as the details of how the algorithm for assessing websites works and the reasoning for why it works the way it does. The fourth chapter contains the results as well as the analysis of the results. Chapter five contains some discussion of the results. And lastly chapter six contains the conclusion and future works section.

1.6 Contributions

The thesis has been a collaborative effort between the two authors. The thesis was written using the online collaborative tool Overleaf for the LaTeX document. This meant that work could be done simultaneously while talking using a VoIP service. The code was developed using the version control software *git* and a shared *Github* repository to store the code in [14]. The code/development was also a collaborative effort to ensure the different components would fit together well. The code had different components that was developed separately but code review and tweaking was done by both authors across all of the code.

Chapter 2
Background and Related Work

To fully understand the context of this thesis, and to develop the algorithm for assessing the security of a website, technical background knowledge is needed. In this chapter, we briefly discuss the various HTTP security headers and how they work.

2.1 HTTP headers

HTTP headers are used to inform the browser connecting to a web server of information about the server, or instructions that the web server wants the browser to do/enforce [20]. This might be information about the server such as what software the server is running (Apache/Nginx for example), caching information (to instruct the browser about how long something should be stored locally in the browser to make a website load faster the second time the browser connects) and security information such as what the browser should allow the website to do. The latter is what is called HTTP security headers [20]. These security headers can be used to make a browser enforce certain rules for the website, like only running javascript code from a certain domain and not allow code loaded from anywhere else.

2.1.1 HTTP Strict Transport Security

HTTP Strict Transport Security (**HSTS**) is an HTTP header set by the web server to inform the browser that the website should only be accessed over an encrypted connection (HTTPS). This header therefore disables the possibility of going to the unencrypted version of the web site (HTTP) once the **HSTS** directive has been loaded by the browser. The header has an expiry time set (which is updated each time the browser visits the website with the header set and the expiry time is longer than the current time left) for when the header should no longer be used by the browser, once this expiry time is up, the browser will be able to connect over HTTP again [13].

There are two optional directives available as well; **preload** and **includeSubDomains**. The **includeSubDomains** directive tells the browser to not only enforce HTTPS usage on the main domain, like *example.com*, but to also use it for all subdomains, such as *www.example.com* and *login.example.com*. The **preload** directive is used to tell browser developers that the domain is allowed to be preloaded as an **HSTS** site [23]. Preloading means that the browser developers, such as Google, Mozilla, Apple and Microsoft, can include the site's **HSTS** header in the source code

for the browser and the **HSTS** header is therefore already in the browser before a user/browser ever having visited the site. Without the **preload** directive, the **HSTS** header is only known by the browser after the first visit to the site. If that first, and all subsequent requests to that site are intercepted, the header can be removed by an adversary and HTTPS is not enforced. However, if the header is already in the browser code, all requests (including the first one ever), will be done over HTTPS. The **HSTS** header can therefore protect a user from having their connection with the web server downgraded to an unencrypted (HTTP) version. To preload a website, an application needs to be sent to the browser developers, it is not enough to just set the header with the correct directive (the directive has to be set for the application to be approved though).

HSTS does not enforce what certificate is allowed though, only that HTTPS has to be used. This means that if someone is able to create a valid certificate for the website, they might still be able to intercept the traffic and still work with the **HSTS** header. There are other protections against this though, such as Certificate Transparency [12] (which forces certificates that are created to be reported to a public certificate log for them to work in modern browsers. If they are not in the logs, the browser will not allow the certificate. With the logs, it is possible for the owners of the domain to see if someone else creates a certificate for their domain.).

2.1.2 Content-Security-Policy

The latest standardized version of the **Content-Security-Policy** (CSP) is version 2 with version 3 being in the working draft phase and many browser have implemented version 3 already. A CSP can be deployed in two ways, either via an HTTP header or inline with the HTML code as a meta tag in the head of the document. The CSP is used to restrict how and from/to where data can be loaded/sent. It can be done in a fairly granular way or by blocking everything with a default directive. The main goal of the CSP is to block Cross Site Scripting (XSS) attacks.

XSS is when an attacker is able to inject code on a website that runs in the browsers of everyone visiting the website. This can be done in two main ways, stored XSS and reflected XSS. Stored XSS is if the code is stored by the website in a database (like if the attacker can comment on a blog post, the comment is stored in the database and the code is loaded for all users viewing the comments of the blog post). Reflected XSS keeps the code in the URL via a parameter in a GET request, and the attacker has to trick the user into clicking on the link for the code to execute. For this to work, the website has to be setup in such a way that a GET parameter on some page is then printed on the page for example. An example could be example.com/welcome?name=Mike, where the welcome page is printing a personalized message with the users name, in this case, Mike. But an attacker could replace Mike with javascript and instead of displaying a name, the javascript code runs.

New websites are generally suggested to be built with a strong CSP from the start since it provides a lot of security benefits and it is easier to do it from the start rather than trying to add it later. To help existing websites to transition to using a CSP, the Content-Security-Policy-Report-Only header can be used instead. The header tells the browser to only report the errors/violations it finds to the administrator but

not block them. This header does not offer any security benefits but it can help with testing a CSP since any violations are reported and edge cases can be found before blocking them completely with a real CSP header. A CSP can also use the **report-to** (the older variant of this is **report-uri**) directive in the CSP to find potential errors but also attempts at finding/exploiting XSS vulnerabilities.

The most common type of CSP directive is the *-src type. They all share a function in that they define the source for the specific function they restrict. For example script-src defines where "<script>" elements should be allowed from and style-src where "<style>"/"<link>" elements should be allowed from. The syntax is ***-src** *<source>;* where source can be a host (IP address or a URL) or a scheme (http:, https:, ...). The source can also be set to *'self'* which allows scripts etc. to be loaded from the URL that hosts the document. For example, if the website is example.com, with a *'self'* value for the script-src directive, scripts will only be allowed to be loaded from the example.com domain, like example.com/someScript.js. There are three unsafe values *'unsafe-eval'*, *'unsafe-hashes'* and *'unsafe-inline'*. They are considered unsafe because they explicitly disables the benefit of CSP for certain content (eval() functions, inline event handlers and inline resources such as "<script>" elements in the HTML code) [40]. The *'none'* value blocks everything and is recommended in the **default** directive.

It is possible to create lists of allowed code by using cryptographic hashes and/or nonces (Number used once), using either the *'<hash-algorithm>-<base64-value>'* and/or *'nonce-<base64-value>'* values. Using a hash, the hash of the code must match one of the hashes in the CSP. This can also help protect against client-side supply chain attacks by only allowing certain hashes and any modifications on an external server will produce a different hash and therefore not load in the browser. If a nonce is used, the nonce must be added to the HTML code for each resource which would be unknown by an attacker and thus will not allow an attacker to inject code. If for example a script is whitelisted in **script-src** by a nonce or a hash and the script loads other scripts itself, those child-scripts might be blocked. To allow any child-scripts from the parent-script, the *'strict-dynamic'* directive can be added after the hash or nonce in the **script-src** directive. This will make child-scripts inherit the permission of the parent-script. This will of course decrease the protection somewhat since the child-script is not verified and is simply given implicit trust which may introduce malicious code to run.

The *'report-sample'* tells the browser to include a sample of the violating code in the violation report. The **default-src** directive is the fallback in case any of the other directives are missing. This is why it is recommended to keep the default directive as restrictive as possible and only open specific directives that are actually needed. Similar to how a firewall should block all incoming requests except the ones specifically needed.

As described by researchers from Google [38], there can be issues with using only a whitelist for external resources like scripts if the whitelisted domain contains code with callback functions/endpoints. A callback function can be used to bypass a CSP by writing the code in the URL to the callback script's URL and then the code is allowed because it originates from a whitelisted URL. The recommendation from the researchers is therefore to use hashes and nonces to only allow specific scripts and not allow everything from a certain domain.

There are many CSP directives, some that provide slightly different functionality than above are upgrade-insecure-requests, frame-ancestors, form-action and base-uri. The **upgrade-insecure-requests** instructs the browser to treat all insecure URL:s (served over http) as secure (served over https). The **frame-ancestors** sets the URL:s that are allowed to frame the current page in a "<frame>", "<iframe>" or similar elements. It can have allowed URL:s, scheme, *'self'* or *'none'* as values. The **form-action** directive restricts which URL:s can be a target of a form submission. It has the same values as the *-src directives. The **base-uri** directive is used to set what is allowed as the base URI for relative links/destinations on an HTML page. For a complete list of CSP directives, the CSP version 2 specification [41] and CSP version 3 draft [40] can be used.

2.1.3 X-Frame-Options

The **X-Frame-Options** security header's main functionality is to improve security against clickjacking [31]. Clickjacking is an attack where the attacker creates one or more invisible layers on the website to trick users into clicking on them. An example would be a website has a button that does something useful but an attacker has loaded a transparent button over it so if the user thinks it clicks on the useful button, they are actually clicking on the attackers button. In essence the attacker is hijacking the users click. The header tells the browser to restrict content loaded through frames. Frames are a way for a browser to display content and it is possible to use frames to load content from remote sources. The header can have one of two possible values *deny* which completely denies content rendered within a frame and *sameorigin* which only allows content that originates from the same page.

2.1.4 X-XSS-Protection

This header instructs whether the browser should enable the cross-site scripting (XSS) filter or not [26]. For an explanation of what XSS is, see 2.1.2 Content Security Policy. The *0* value will disable the filter. The *1* value will enable the filter and if an attack is detected the browser will try to sanitize the page. The *1; mode=block* will enable the filter but if an attack is detected the browser will not render the page but block it completely instead. The header also has a report attribute *1; report=<URI>* where any violations will be sent to the specified URI. This header is deprecated in most modern browsers and a CSP should be used instead [26].

2.1.5 Set-Cookie

The **Set-Cookie** header is used to send HTTP cookies to the browser which the browser sends back in future requests. An HTTP cookie is a small piece of data stored on the users computer by the browser. It is used to remember actions taken during browsing (like saving a shopping cart) or to save a session token (to keep a user logged in without having to enter a username and password for each request). In essence, cookies serve as memory for a browser. The base configuration of this header is: **Set-Cookie**: *<cookie-name>=<cookie-value>*. **Set-Cookie** also has several optional attributes. The first optional attribute is **Expires**=*<date>* and this attribute

sets the lifetime of the cookie, if this attribute is not set, the cookie will become a session cookie and will be removed when the browser is closed (browser session). **Max-Age**=<*number*> sets the number of seconds until the cookie expires, if both the **Max-Age** and **Expires** attributes are set, **Max-Age** takes precedence. **Domain**=<*domain-value*> sets to which domain the cookie should be sent to in future requests. If it's missing the domain will default to the current URL. **Path**=<*path-value*> sets a path that must exist within the URL for the browser to send the cookie. **Secure** makes it so the browser will only send the cookie to the server if the websites uses HTTPS [2]. **HttpOnly** prevents javascript from accessing the cookie, which can help mitigate against XSS attacks stealing information from the cookies (which may contain session tokens for the user). **SameSite**=<*samesite-value*> determines if a cookie should be sent with cross-origin requests, this offers some protection against cross-site requests forgery (CSRF) attacks [42].

Cross-site requests forgery (CSRF) is an attack where a user's browser is used to send requests as the user (forgery) to another website (cross origin). Essentially, the user is logged into website X and then goes to website Y. Website Y contains javascript code that does a request to website X in the browser. Since the browser has a cookie with session information for website X, the action is performed as the user of the browser on website X. This action could be to change email/password, send money to the attacker, etc.

The **SameSite** attribute has three modes, *Strict* specifies that cookies are only to be sent on same-site requests (if the request is coming via website Y to website X, the browser will not send the cookie), *Lax* is less strict than *Strict* in that it will allow cookies to be sent in a limited amount of circumstances to increase convenience for users (in "safe" HTTP requests such as GET/HEAD requests that should not change information, while requests like POST will not include the cookie when it is a cross origin request). The last mode is *none* which, as expected, offers no protection [42]. The feature is supported by most modern browsers and can effectively stop all CSRF attacks if implemented correctly (if SameSite is set as either *Lax*/*Strict* and the website is working according to standards and do not alter state on GET requests), but traditional CSRF protections are still recommended for older browsers that may not support the attribute (and to have a defense-in-depth approach in general). Some modern browsers are now defaulting to use a **SameSite** policy of *Lax* if it is not explicitly set as something else [22].

2.1.6 X-Content-Type-Options

When loading resources into a website, the browser requests the files from a web server and then it tries to use them. But to know what to do with a file, it relies on MIME types (Multipurpose Internet Mail Extensions) sent with the file from the web server as **Content-Type** [21]. Sometimes, this MIME type is not setup or is setup incorrectly which forces the browser to try and figure out how to use the file (run it as script, display as image, render HTML page, etc) correctly. However, this can introduce security issues when the browser has to try and figure out what to do with a file. If a website allows for images to be uploaded, an attacker might upload a javascript file instead and once the image is downloaded by a victims browser for viewing, the browser might recognize that it is not actually an image but a script

and instead run the script.

The **X-Content-Type-Options** header has a single directive: **nosniff**. What this header and its directive does is tell the browser to not try to MIME sniff and guess what type of file it is, but follow the content type as set by the web server and not change that or try to guess if it is missing [25].

2.1.7 Referrer-Policy

The **Referrer-Policy** tells the browser what to include in the **referer** (misspelling intentional) header when making requests [11]. The **referer** header includes the URL of the page the user came from, for example if a user uses google and presses a link, the request will include google as the referrer. The **Referrer-Policy** must include one of several values. The *no-referrer* tells the browser to completely omit the **referer** header in requests. The *no-referrer-when-downgrade* value tells the browser to include the origin when the protocol security level stays the same (e.g HTTPS->HTTPS, HTTP->HTTP) or when the security level increases (HTTP->HTTPS), but not when it diminishes (HTTPS->HTTP). The *Origin* value tells the browser to only send the document (www.example.com/) of the origin instead of the whole origin (www.example.com/page.html). The *origin-when-cross-origin* is similar to *Origin* except that a full origin URL will be sent during same-origin requests. For *same-origin* a referrer will be sent only for same-site origins, cross-origin requests will have no referrer information. A *strict-origin* value will make the browser only send a referrer when the protocol security level stays the same. The *strict-origin-when-cross-origin* is similar to *strict-origin* except that it will send full referrer information for same-origin requests, this is the the default behavior on most modern browsers if **Referrer-Policy** is not specified. The last value is *unsafe-url* and it will always send full referrer information, this can in some circumstances leak private information to insecure origins [11]. This header can protect against information leakage for users following links to other websites. If a user is on a sensitive website with a specific path, and then follows a link to another website, the other website might find out about this sensitive information (the actual information may not be disclosed but the meta data, i.e. domain/path can still disclose sensitive information). The referrer header is sometimes also used for CSRF protection since requests originating from the same domain is unlikely to be a CSRF attack.

2.1.8 Permissions-Policy

The **Permissions-Policy** header exists to selectively enable and disable specific browser features and API:s, like camera, battery or microphone [19]. The full syntax is **Permissions-Policy**: *<directive> <allowlist>* where the directive is the feature (microphone, camera, geolocation, etc) and the allowlist is a list of origins that are allowed to use the feature (some examples, *'none'*, *'self'*). This can protect users from an adversary that is able to run javascript code via an XSS vulnerability from accessing browser APIs of the user's browser (like the accessing a camera or getting access to geolocation data, etc).

2.1.9 Subresource Integrity

Although a CSP can limit scripts/stylesheets from running based on if the contents match the computed cryptographic hash value in their respective CSP directive, another option can be used to enforce that the contents of the external scripts/stylesheets have not been altered; Subresource integrity [24]. This is done via an integrity attribute in the HTML code where the script is loaded (SRI is not actually an HTTP header). This option is more limited than a CSP since it does not protect against if an adversary finds a XSS vulnerability in the website. However, it does protect users of the website from loading code from an external server that has not been validated by the website administrators. This means, if a website uses an external javascript provided by a third party, the administrators can verify the contents of the script and then compute a cryptographic hash of this script and insert it into the HTML code. When the users browsers load the web page, the contents of the third party javascript code will be hashed and compared with what is in the HTML code. If the hashes match, the code will run. If the code has been changed however, the hashes will not match and the browser will not run the code. Its usefulness is limited but it would help to prevent some client-side supply chain attacks as described in Section 2.1.2 Content Security Policy. Therefore, if a CSP cannot be implemented immediately, subresource integrity checks in the HTML code can be a good first step towards protecting against client-side supply chain attacks. An example could look like [24]:

```
<script src="https://example.com/example-framework.js"
integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HN
QlGYl1kPzQho1wx4JwY8wC" crossorigin="anonymous"></script>
```

2.2 Related Work

2.2.1 Literature Mapping Study

In this section, the findings of the literature mapping study is discussed. As per the topic and goals of the thesis, the three most relevant databases to search in is Springer, ACM Digital Library and IEEE Xplore. The first step was to search for generic queries regarding the topic such as "http security headers". From the given results, the first review process was conducted by reading titles and abstracts for what seemed relevant. After rejecting many articles, the remaining articles were reviewed based on their results, discussion and conclusions [15]. In total, the process yielded 16 articles. From the 16 articles, snowballing and reverse snowballing returned another 9 relevant articles, for a total of 25 articles. Each step and the result is illustrated in Figure 2.1. The most relevant articles to our thesis are described in Section 2.2.2.

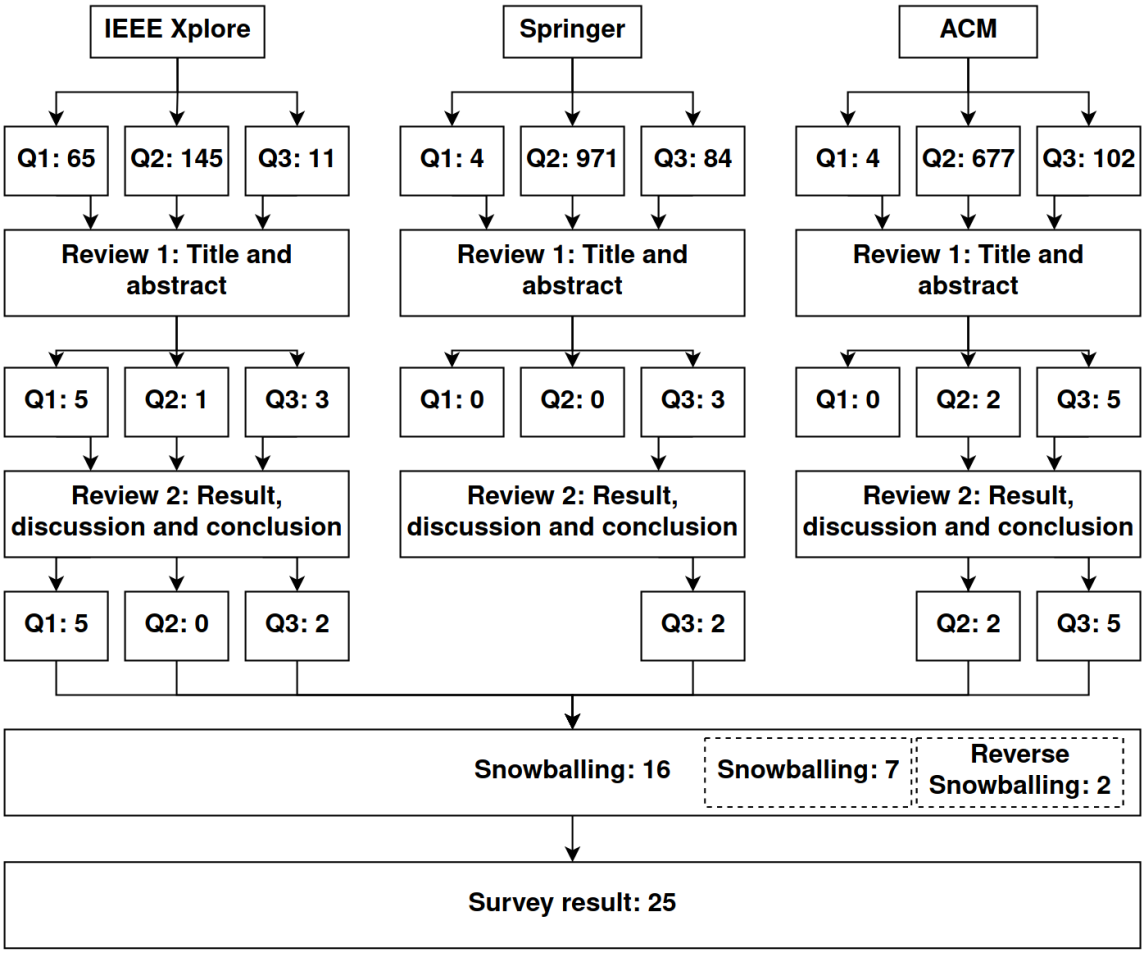

Figure 2.1: Overview of the literature mapping study procedure used to find the most relevant articles from the IEEE Xplore, Springer and ACM databases.

Table 2.1 show the queries made to find articles and what filters were used to sort for the most relevant papers. These articles were then accepted or rejected based on title, abstract and year.

Query	**Filter (IEEE Xplore)**	**Filter (ACM)**	**Filter (Springer)**
Q1: http security headers	Conferences, Early Access Articles, Journals, Books, Standards	NOT VirtualContent: true, ACM Content: DL	-
Q2: http headers	Conferences, Early Access Articles, Journals, Books, Standards, Magazines	NOT VirtualContent: true, ACM Content: DL	-
Q3: content security policy	-	NOT VirtualContent: true, ACM Content: DL	-

Table 2.1: Search queries used during the literature mapping study and the filters used.

Further, table 2.2 describes what inclusion criteria was used to accept and reject the different papers in the second review of the literature mapping study were the contents were looked at more thoroughly.

ICs	**Description**
IC1	Usage statistics of security header(s)
IC2	Discussing strengths and weaknesses of one or more security header(s)
IC3	Discussing XSS and/or client-side supply chain issues and attacks
IC4	Analyses some other related web security aspect or attack

Table 2.2: The inclusion criteria (IC) of the literature mapping study used to accept and reject different papers.

Tables 2.3, 2.4 and 2.5 indicates with an **X** what criteria were met for them to be considered.

Paper	Year	IC1	IC2	IC3	IC4
Lavrenovs & Melón [16]	2018	X	X		
Lavrenovs & Podins [28]	2018	X	X	X	X
Dolnák & Litvik [10]	2017		X	X	
Dolnák [9]	2017		X		
Köpsell et al. [37]	2017	X	X	X	X
Dolnák [8]	2017		X	X	
Pathan & Yusof [44]	2016		X	X	

Table 2.3: The papers found during the literature mapping study from the IEEE Xplore database and their inclusion criteria (IC) fulfilment.

Paper	Year	IC1	IC2	IC3	IC4
Xurong et al. [17]	2018	X	X		X
Weissbacher et al. [39]	2014	X	X	X	

Table 2.4: The papers found during the literature mapping study from the Springer database and their inclusion criteria (IC) fulfilment.

Paper	Year	IC1	IC2	IC3	IC4
Barth et al [3]	2008				X
Stamm et al [34]	2010		X	X	
Van Acker et al [36]	2016	X	X		X
Weichselbaum et al [38]	2016	X	X		
Calzavara et al [6]	2016	X	X		
Some et al [33]	2017	X	X		
Calzavara et al [5]	2018	X	X		

Table 2.5: The papers found during the literature mapping study from the ACM database and their inclusion criteria (IC) fulfilment.

2.2.2 Literature summary

Multiple studies have been conducted on the usage of HTTP security headers for the Alexa top 1 million most visited websites. Their findings are fairly similar in that the adoption of these security headers are not very widespread and many implementations of headers such as the CSP header contains flaws that can be used to bypass the protection.

Extensive research into the various HTTP security headers and the adoption of them among the top 1 million according to Alexa was conducted by Lavrenovs and Melón [16]. Their research showed that websites that only use HTTP (and no encryption via HTTPS) were far less likely to implement security headers. But

generally, the adoption was quite low even among the most popular websites in the world. The HSTS header for example was only implemented in 38% of the top 1 thousand websites and in total only about 17.5% for all 1 million scanned websites.

The research by Buchanan et al. [4] suggests a grading system based on their scans of the top 1 million websites (according to Alexa). However, this grading system includes the now deprecated HPKP header (that could be used to enforce only one certificate to be allowed for encryption) and also did not consider the contents of the headers, just that they were implemented. If a header was implemented, the website got the "points" for the header, even if it was inherently insecure. Our grading algorithm should therefore be better at determining actual security benefit of the implemented security headers rather than just showing if they have been set explicitly (an explicit security header set to something insecure is often the same or worse than just using the browser default).

Calzavara et al [6] conducted a study on the adoption rate of CSP in the Alexa top 1 million websites. They also checked browser support for CSP, if CSP was correctly configured and how the adoption rate was maintained over 14 weeks. They found that less than 1% of the websites uses CSP and of those 40% are in enforcement mode. They concluded that 92.4% of the websites using CSP in enforcement mode are still vulnerable due to using 'unsafe-inline' in the script-src or default-src directive. While we plan to look at CSP adoption in our study, we also want to look at other http headers.

Weissbacher et al [39] performed a longterm study on CSP adoption in the Alexa top 1 million to determine challenges in CSP deployment. They found that only 1% deployed CSP in enforcement mode. They performed weekly crawls over a period of 16 months. They found that a majority of the site using CSP uses a configuration that drastically reduces the benefits of CSP. They also found that most of the CSP enabled websites were installations of phpMyAdmin which comes with a weak default CSP policy. They conducted an experiment using CSP's report-URI directive, which generates a report each time a CSP violation occurs, on four websites with different configurations with the aim to identify a good policy configuration. They found that many of the reports came from browser extensions trying to load in resources from non-whitelisted sources. They also tried using a semi-automated approach at generating policy configuration with mixed results. The methods proposed required too much manual fine-tuning. Their research were more focused on generating good CSP configurations while we want to focus on a method to detect whether the configurations in use are good or not.

Researchers from Google investigated the security of all websites indexed by the Google crawler bot (that creates the Google search index) which is a dataset of 6.5PB (Petabytes) of data [38]. Of all the unique CSPs they found, 94.72% of them were bypassable. One of the biggest reasons for this was the usage of insecure directives such as "unsafe-inline" or whitelisting CDNs that allow user uploads or has callback functions that can be used by an adversary to inject code. The solutions they found was to use either a nonce (number used once - for each response, give a new nonce for scripts that an attacker cannot predict) or hashing (sub-resource integrity) to limit what resources could be loaded and not just limit it to a whitelist. This research is similar in the way it analyzes CSPs and contains a lot of research about the security aspects of CSPs. Our research into CSPs is not quite as thorough since we are doing

a general overview of all security headers. However, the paper was very useful in the design of our algorithm for scanning CSPs.

The research conducted by Vumo et al [37] is the closest in terms of scope and goal to ours. They performed an assessment on 240 mozambique based websites to see the implemented security mechanisms. Like us they looked at the HTTP security headers. They did look at certificate information aswell, which we did not but we however looked at subresource integrity which they did not. Their data collection method also differs from ours, they made use of pentesting tools while we used passive python scrapers. We used our methods because of the ethical concerns of using pentesting tools on websites without proper approval. Their results show an extremely low adoption rate of HTTP security headers to the point of being none existent.

Most of these articles include a much bigger data set than our research used and their results are more generalizable but our focus is on the Swedish government (of which we scan all government agency websites) and creating an algorithm for grading and comparing different websites. The results are still very interesting to compare against and the articles contain a lot of useful information.

Chapter 3

Method

The methodology used for answering research questions **RQ1** and **RQ3** is explained in this chapter. The main sections are data collection, analysis and statistics. To answer **RQ2**, a literature review was performed and the results are found in Chapter 2. Figure 3.1 shows the iterative work done with both research and development to design and create the algorithm.

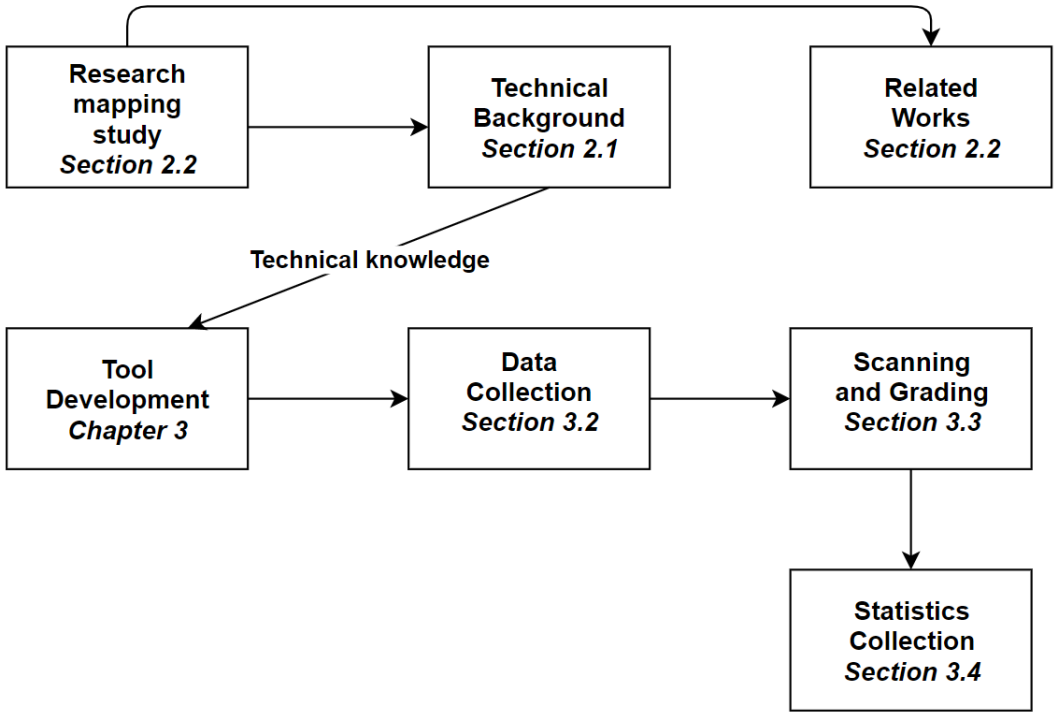

Figure 3.1: An overview of the different phases of the study. This shows how the research phase ties together with the development and survey part of the thesis.

3.1 Research method

3.1.1 Technical survey

To answer the main research question a technical survey was performed. This method was chosen over other methods due to it being the most efficient. The other method that was considered was an interview based survey but interviewing hundreds of

system administrators would be very difficult and time consuming provided they accept the interview. A traditional survey may lack in number of responses and might yield biased data were only the agencies with higher budgets and potentially better security will respond. A technical survey is also very reliable because using an automated method to collect data directly from the website will yield the objective truth on the implemented security features without misrepresentation/bias.

3.2 Website scanning

The scope of the research is limited to analyzing Swedish government agency websites (SGAWs). A list of all of these websites are provided by SCB through *Myndighetsregistret* [32]. To accomplish the goals of the research questions, data about the websites had to be fetched for later analysis. This was done through a python script and the *requests* [30] library for handling the HTTP requests. All of the data (such as headers and cookies) was stored in a database. To ensure the validity and reliability of the results, redirections and status codes were stored so that any server errors or similar could be filtered out during analysis.

Some information, while not HTTP headers, was collected and stored as well as it is relevant for the analysis. This included parsing the HTML code of each website to determine the number of external resources (javascript and stylesheets) loaded by the website and how many of those external resources used an integrity check. A parser for the CSP header was written to separate the directives and store them individually in the database for simpler analysis.

An issue with running a survey such as this one is that a website may respond differently to different types of clients (desktops, laptops, phones, tablets, etc.) and from where the request originates in the world. There are thousands of combinations of clients/browsers and hundreds of different countries to make requests from. Therefore, it was deemed impossible for the scope of our survey to see if there are any differences in this regard. Therefore, the tests were ran from Swedish ISPs and using a common Desktop user-agent (Firefox and Chrome browsers) for our script to connect from to try and represent a connection from typical a user of SGAWs. To avoid any potential issues with one particular network, two separate Swedish residential ISP networks and the network at the Blekinge Institute of Technology campus (SUNET) were used to run the tests.

3.3 Quality analysis

To assess the security measures of the SGAWs, an algorithm was created. This algorithm grades the website based on the implemented security headers (from 0 to 100 percent). The algorithm also takes into account a weight that decides how much the particular test will affect the overall score. The weights can be changed to give more/less focus to a particular header/feature/property. This was deemed necessary since not all headers/features/properties are needed for all websites.

An example of this could be an online retailer with a shopping cart feature. This functionality might work by using a cookie to store what items are in the shopping

cart. To update the cart, javascript on the client might be used to minimize the requests to the server. Having the security feature *HttpOnly* on all cookies for a website like this may not be strictly necessary since the contents of the cookie is unlikely to contain sensitive information. On the other hand, a bank that uses cookies to store a session token for their logged in users are likely going to want to use the *HttpOnly* security feature. This is to ensure that if malicious javascript runs in a users browser (through XSS vulnerabilities for example), the javascript will be unable to steal the session token for the bank account.

This algorithm works by receiving a JSON object as input for the different weights and tests. To make it easier to use, a website was created that creates a graphical user interface (GUI) to scan a website (Figure A.1), set the weights (Figure A.2) and then receive the results (Figure A.3) without having to interact with code.

To be able to compare different websites, the weights are configurable and can be used to adapt the algorithm to suit different kinds of websites with different security needs. It is unlikely that a "one-size-fits-all" algorithm is possible that does not require some user intervention to get fair and accurate results for all types of websites. This is a drawback of the algorithm since it requires some knowledge about the different security headers/features and the many properties that are configurable to utilize it properly. Someone that has this knowledge may not necessarily need a tool to be able to determine their websites' security posture. However, it can be used to effectively compare many websites even for someone with a lot of knowledge. To try and solve the issue of the algorithm being hard to use, the GUI was created that contains guidance and can also supply the user with preconfigured profiles to choose from as starting points for the weighting. This should be simpler for most than to directly interface with the algorithm using code/JSON.

The HTTP security headers and features that are analyzed were decided on because they are a part of the web standard [20] and were used by more than one browser/browser engine. The features/headers were also picked based on related studies discussed in Section 2.2 that looked for if the header was implemented or not.

The algorithm works by performing multiple tests on the website and for each test, the website receives a score of either 0, 0.5 or 1. This score is then multiplied with the weight assigned to the test and all of the scores are then summed into one final score. The algorithm reports what the score was for each test but also the total score as a JSON object. In some cases it is possible that a website implemented a header more than one time, in those cases we decided that the value of the last instance of the header is the valid one based on the documentation, for example for the Referrer-Policy header [11]. Figure 3.2 describes the overview of how the algorithm works.

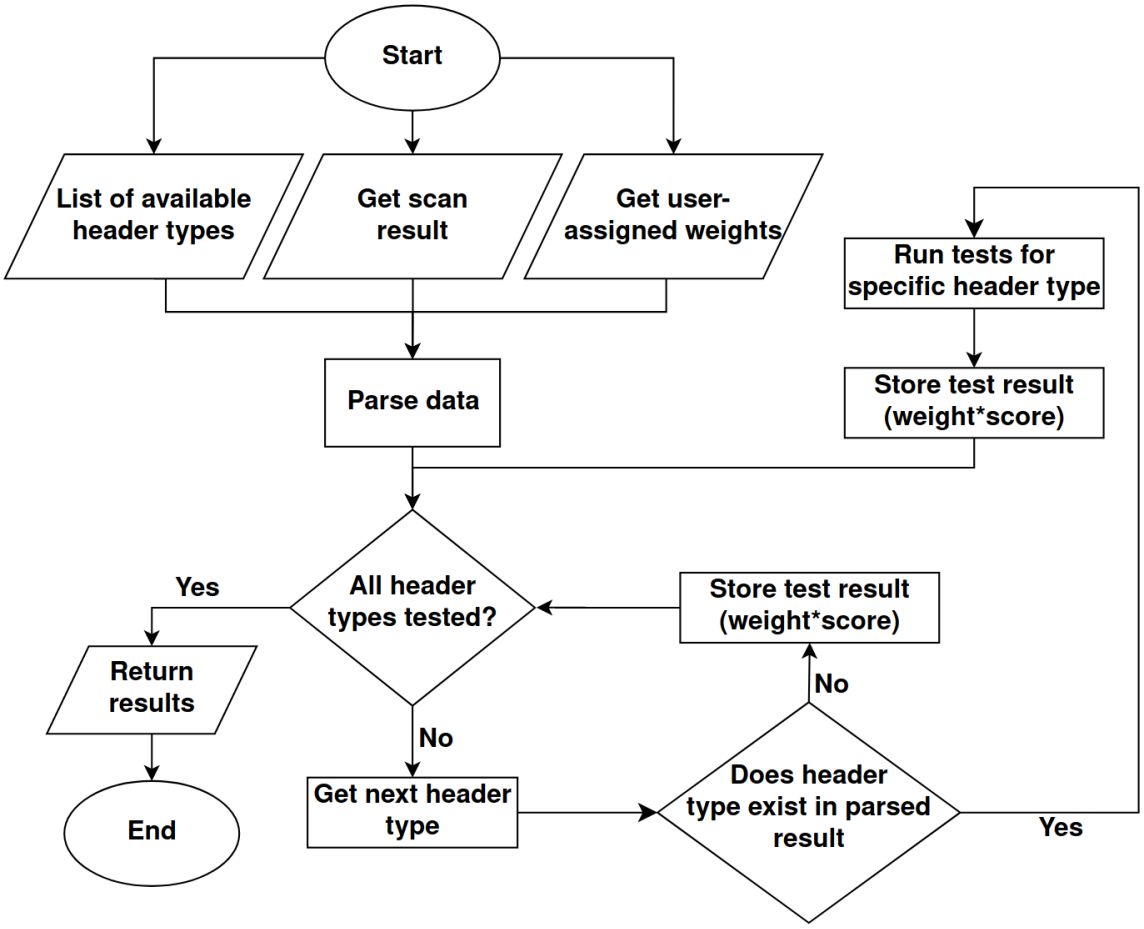

Figure 3.2: A flowchart of the algorithm used to score a website's implementation of HTTP security headers. This shows an overview of the algorithms functionality.

For each header type, there are one or more tests ran to check the header's implemented security benefit. When analyzing a website, the following tests are performed:

HTTPS

- Check if server supports HTTPS (1 point).
- Check if server automatically redirects HTTP traffic to HTTPS (1 point).

The tool only checks if the server supports HTTPS and whether the server redirects clients from HTTP to HTTPS. There are other types of data that could be checked for, like expiry date, algorithms supported, version of SSL/TLS and many others. However, the two tests were decided on because the main goal of the study is HTTP security headers, not HTTPS. HTTPS is scanned for to know more about the website itself and if the headers it sets are valid (HSTS requires HTTPS for example).

The tests for the HTTPS feature are binary. They are either enabled or not. Therefore, the score is either 1, if implemented, or 0 if not implemented. The weights,

which are configurable, determines the importance of the test in relation to all the other tests.

HSTS

- Check if max-age is set (0 points if set to less than 6 months, 0.5 points if between 6 months and 2 years and 1 point if set to 2 years or more).
- Check if includesubdomains is set (1 point).
- Check if preload is set. (1 point if includesubdomains is set and max-age is at least 1 year).

Strict transport security has three attributes, one that is required and two optional ones. Max-age is required and specifies the amount of time the browser should remember to only access the website using HTTPS. Based on recommendations from the browser vendors [29], the end goal is to have a max-age of at least 2 years, which result in 1 point. The includesubdomains flag tells the browser that the sites subdomains should also be loaded using HTTPS and will therefore ensure that no subdomain is missed like when not including the flag. To get points for setting the *preload* flag a few conditions must be met based on the preload list requirements [29].

CSP

- Check the values of multiple *-src* directives according to their security level:

  Bad (0 points)

  - * - Wildcard, allows everything.
  - unsafe-inline / unsafe-eval / unsafe-hashes - Allows inline scripts which allows for XSS vulnerabilities.
  - data: - Allows data: URIs which can be used to inject data.
  - http: scheme - http scheme allows basically all sources without enforcing encryption.
  - http:// whitelist - http URLs are unencrypted and can be intercepted by an attacker.

  Moderate (0.5 points)

  - https:// whitelist - Allows only encrypted (HTTPS) communication with whitelisted domains.
  - https: scheme - Allows connections to all websites with HTTPS enabled.
  - strict-dynamic - Allows scripts/stylesheets to inherit permissions to run by the scripts that included them.

  Good (1 point)

  - self - Only allow resources to be loaded from the current domain (not inline).

  - hash / nonce - Only allow resources loaded with an accompanying hash or number used once (nonce).
  - none - Do not allow anything.

- Undefined directives inherit the *default-src* directive if available (As according to the CSP standard. Some directives do not inherit the *default-src* directive.).
- Separate checks for special directives such as
  - frame-ancestors - Defines valid parents that can frame the page (1 point if https whitelist, none or self. 0.5 points if https scheme. 0 points if *, http scheme or http whitelist).
  - report-to / report-uri - Defines how to report violations to administrators (1 point).
  - sandbox - What capabilities are allowed (like pop-ups, scripts, forms, etc.) (1 point).
  - upgrade-insecure-requests - Resources are forced to be loaded over HTTPS over HTTP (1 point).

The content security policy header is special since it contains a lot of directives and values. The directives chosen are the ones in the standard that are not experimental and not deprecated [18]. Many of the directives follow the same format however and support the same values (directives ending in *-src*). This helps with analyzing the header. The *-src* directives have three possible scores they can receive. Either 0, 0.5 or 1.

The 0 (or **Bad**) score is used for values that allow everything (such as wildcards, unsafe-inline (and similar) and the data: scheme which allows data to be stored and used for XSS. Allowing these values essentially removes any defense against XSS the CSP might protect against. The http: scheme is also considered a zero-score since it allows all URLs and the included data is not forced to be sent encrypted. This means that even legitimate external resources (if a whitelist with URLs using the http: scheme is used) could be intercepted and have their data changed and is therefore not considered secure.

The 0.5 (or **Moderate**) score is used for the https: scheme and whitelists with only URLs using the https: scheme. The https: scheme itself is not a great protection since anyone could create a website with legitimate SSL/TLS certificates. However, it does protect against traffic being intercepted and changed. The whitelist with only https URLs is better but issues brought up by a paper from Google [38] highlights an important issue where the CSP can be bypassed if the allowed URLs support JSONP endpoints (callbacks). Whitelisting without such endpoints still suffer the risk of being replaced with something malicious which is an issue with the supply chain. The *strict-dynamic* directive simplifies deployments of CSPs since it allows scripts/stylesheets to load their own scripts/stylesheets without being explicitly allowed by the CSP. However, this does introduce a risk to the supply chain since those sub-resources are not validated.

The 1 (or **Good**) score is used for if nothing is allowed. The score can also be given if the website itself loads resources from its own domain/server (there is a

potential issue with this value if the website hosts its own JSONP endpoint. But this is considered out of scope since the server the website is running on is considered secure). Another value which gives the maximum score is to use hashes and nonces. A hash for example, allows the browser to determine a scripts cryptographic hash, and by comparing it with the CSP before allowing it to execute, the website is safe from both XSS and client-side supply chain attacks (since both inline/self-hosted and external resources has to match the hash).

Similar to how browser work, undefined *-src* directives will inherit the default directive if it is defined in the grading algorithm. In this case, it will inherit its score but will use its own weight for each respective directive.

The frame-ancestors directive should define a set of hosts (https URLs) that it allows to frame it (or none). If it does, the score is 1. If the value is defined to just the https scheme, the directive will only get a moderate score (0.5) since while it does force it to load over an encrypted connection, all encrypted connections are allowed. The directive gets a score of 0 if the value is set to load over http or it uses a wildcard. This is because there are no restrictions at all with these values and the directive is essentially not needed. The report-to/report-uri, sandbox and upgrade-insecure-requests directive receives a score of 1 if implemented and 0 if not implemented.

set-cookie

- Check if the secure attribute is set (1 point).
- Check if the httponly attribute is set (1 point).
- Check is the samesite is set to Lax or Strict (1 point).

If more than one cookie exists, the site will only get points for the lowest scoring one. The three attributes offer some form of protection. The *secure* attribute will make the site more resistant against man-in-the-middle attacks by forcing it to only load over encrypted HTTPS connections. *HttpOnly* will not allow the cookie to be accessed by javascript, which can help to prevent cookies from being stolen in XSS attacks. The *SameSite* attribute, with either lax or strict, can help to protect against Cross-Site-Request-Forgeries (CSRF) attacks. They are all binary attributes, they are either implemented or not, which is why they all receive a score of 1 if implemented correctly.

X-Content-Type-Options

- Check if the nosniff attribute is set (1 point).

This header only has one attribute *nosniff*, which disables the browsers MIME sniff capability. One point is awarded if this header is present.

X-XSS-Protection

- Check if the XSS filter is enabled (1 point).

- Check if the filter mode is set to block (1 point).

The tests for this header checks whether the filter is enabled and if the website instructs the browser to block the rendering of the website if an XSS attack is detected. These tests are binary and is therefore either a 1 for the header value being implemented or 0 if the value is not implemented.

X-Frame-Options

- Check if set to deny or sameorigin (1 point).

This header can only have one of two values, deny or sameorigin. Both offer protection against click-jacking attacks and implementing it correctly is therefore enough to get 1 point.

Referrer-Policy

- Check if referrer policy is set to anything other than unsafe-url (1 point).
- Check if referrer policy is not set because modern browser sets a good default policy (0.5 points).

Most modern browsers have a good default referrer policy, so the only way to receive 0 points is to actively set a bad one. The algorithm gives half a point to website that does not have a referrer policy because Safari on MacOS (as of version 14) and iOS (as of version 14.5) does not have a default policy [7].

SRI

- Checks ratio of external resources to external resources with implemented SRI (Ratio points).

SRI is another feature the algorithm checks against that is not a security header. This is checked against since, while a CSP can use hashes to allow/deny external resources, it is not necessary from a supply chain aspect. An SRI for each external script can be used to only allow scripts with a certain hash and if they are replaced, they will not load. The score for SRI is simply the ratio of external resources to external resources with SRI implemented.

Generalizable

The algorithm should be generalizable since it is built/designed after the HTTP standards and that all websites/web browsers follow. The code should be possible to extend with new functionality easily to accommodate future security features/headers that are introduced. Given the configurability of the algorithm, it should also be possible to use in almost all scenarios by changing the weight parameters.

3.4 Collecting statistics

To collect data on how many of the websites implemented the security features, a python script was made. The script fetches all the unique domains in the database and runs them through the grading algorithm above. The script then counts the number of times a header was implemented as well as the number of times the header was well/moderately implemented. The script outputs a JSON object with the statistics. The results of this is discussed in Chapter 4.

Chapter 4

Results and Analysis

The results of the thesis were collected through a technical survey as described in chapter 3 and are presented in this chapter. The data was collected on 2021-03-19 for all SGAWs. All websites were scanned multiple times during a two week time span (between 2021-03-08 and 2021-03-19) to ensure that no website happened to be offline during a specific scan. Of course, administrators may change these headers at any time so our results only show a snapshot of reality on this particular day. However, the tool for scanning and analyzing is available [14] so that the study can be replicated in the future, even if the results may not be exactly the same. The methodology is also available so that the survey can be replicated without the use of the exact same code.

4.1 General

A total of 318 websites were scanned but only 311 responded with a successful status code. The 7 websites that responded with an unsuccessful status code responded with the status code of 404 (page not found error). These 7 web pages are remnants from older/smaller agencies that no longer exist but have not been removed from *Myndighetsregistret* [32] and were hosted by another SGAW that is already in the dataset. Out of the 311 websites, 205 of them were unique. This was caused by domains/sub-domains being redirected to the same website in a lot of cases but also pages hosted on the same domain with a different path.

Websites	**Websites (status code 200)**	**Unique (status code 200)**
318	311	205

Table 4.1: Total number of scanned websites, websites that successfully responded and the total number of unique websites that responded successfully.

4.2 Headers

Out of the 205 unique websites, most supported HTTPS. With exception for **X-Frame-Options** and **HTTPS**, all security features had a total usage of less than 50% on all websites (**Set-cookie** is not strictly considered a security feature but its attributes, *secure*, *httponly* and *samesite*, are, which is why it is included).

Security feature	Total
HTTPS	200
HSTS	62
CSP	23
Set-cookie	146
X-Content-Type-Options	60
X-XSS-Protection	53
X-Frame-Options	109
Referrer-Policy	39
Subresource integrity	16

Table 4.2: Total number of websites with the implemented security feature.

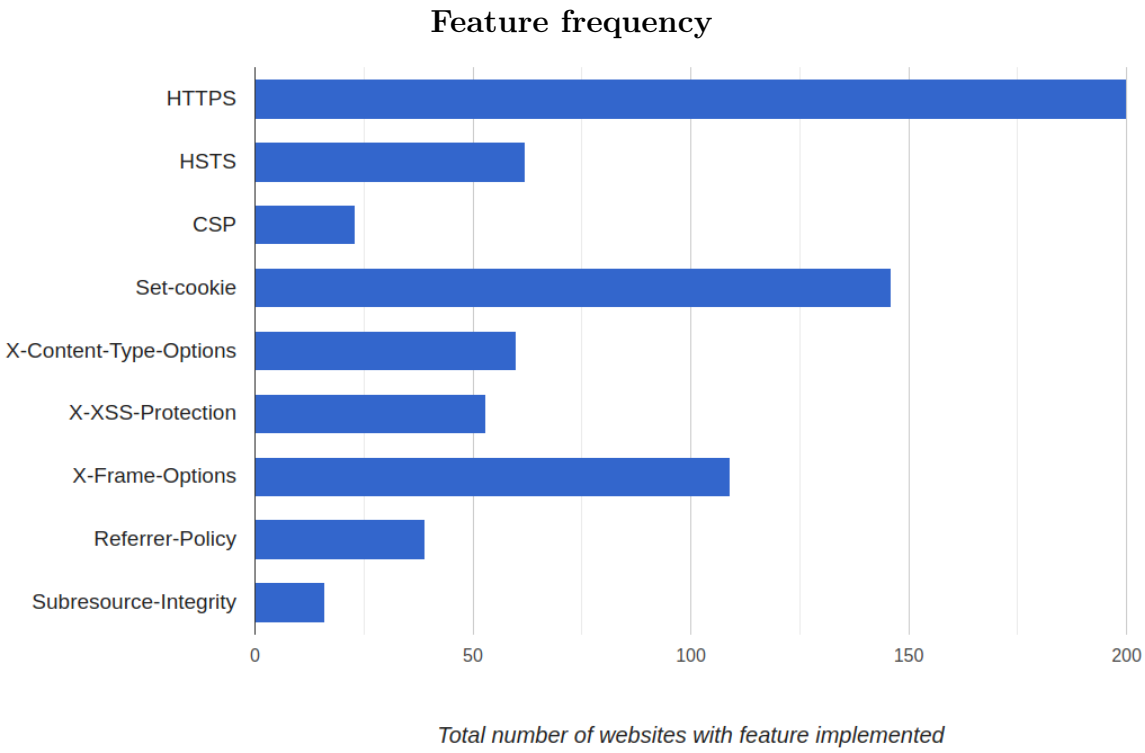

Figure 4.1: Number of websites with the implemented security features.

4.2.1 HTTPS

HTTPS was supported on 200 out of the 205 unique websites scanned. 196 automatically redirect to HTTPS from HTTP. Meaning that 5 SGAWs do not support HTTPS at all and out of the 200 that do, 4 do not redirect to the encrypted version of their website.

Information	**Total**
HTTPS exists	200
Redirects to HTTPS	196

Table 4.3: Information about usage of HTTPS on SGAWs.

4.2.2 Strict-Transport-Security

HSTS was supported on about 31% of the 200 websites that supported **HTTPS**. Only 62 websites had implemented **HSTS**, 7 of them had a *max-age* attribute set to the recommended time of 2 years or more and 46 had between 6 months and 2 years. 23 websites included the *includesubdomains* attribute and 8 had the *preload* attribute in conjuncture with at least 1 year max-age and *includesubdomains* active (which is necessary for the *preload* attribute to be accepted by the browser vendors).

Attributes	**Total**
max-age (>= 2 years)	7
max-age (>= 6 months and < 2 years)	46
includesubdomains	23
preload	8

Table 4.4: Statistics of the attributes for the HSTS header out of the 62 website with the header.

4.2.3 Content-Security-Policy

The **Content-Security-Policy** was only implemented on 23 out of 205 unique websites scanned. Out of the 23 websites, 9 had a well implemented *default-src* and none had a moderately implemented one. Some of the more important attributes to look at are *script-src*, *style-src*, *frame-ancestors* and *upgrade-insecure-requests*. Only 2 websites had a well implemented *script-src* while 1 had a moderately implemented one. 1 website had a well implemented *style-src* and none had a moderately implemented one. For the *frame-ancestor* attribute 10 websites had a good implementation and there were no moderately implemented ones. Only 1 website made use of *upgrade-insecure-requests*.

Attributes	**Total**
default-src	9
child-src	8
connect-src	2
font-src	5
frame-src	0
img-src	1
manifest-src	8
media-src	6
object-src	9
script-src	2
style-src	1
worker-src	9
report-to/report-uri	5
base-uri	5
form-action	4
frame-ancestors	10
sandbox	0
upgrade-insecure-requests	1

Table 4.5: Number of *well*-implemented attributes among the 23 websites with a CSP.

Attributes	**Total**
default-src	0
child-src	1
connect-src	7
font-src	4
frame-src	9
img-src	3
manifest-src	0
media-src	4
object-src	1
script-src	1
style-src	0
worker-src	1
report-to/report-uri	-
base-uri	0
form-action	1
frame-ancestors	0
sandbox	-
upgrade-insecure-requests	-

Table 4.6: Number of *moderately*-implemented attributes among the 23 websites with a CSP.

The best CSP

The best CSP found in one of the SGAWs has its distribution of directive implementation shown in Figure 4.2. The CSP for the website used *'none'* for the *default-src* which is important for any directives that are not explicitly implemented since no directive risks being missed. The two most important directives, *script-src* and *style-src* only used the *'self'* value and made use of hashing to allow certain scripts/stylesheets. This means, that unless a JSONP endpoint (as described by researchers from Google [38] could be used as an XSS bypass) is hosted by the website itself, XSS should be practically impossible (without another type of CSP bypass). The moderately graded

directives used external URLs for *child-src* and *frame-src* which should mostly be fine. The badly graded directives were either not implemented (and were directives that does not inherit the *default-src* directive, such as *report-to/report-uri* directive) or used the *data:* value which is not recommended since it can be used as a data source for XSS.

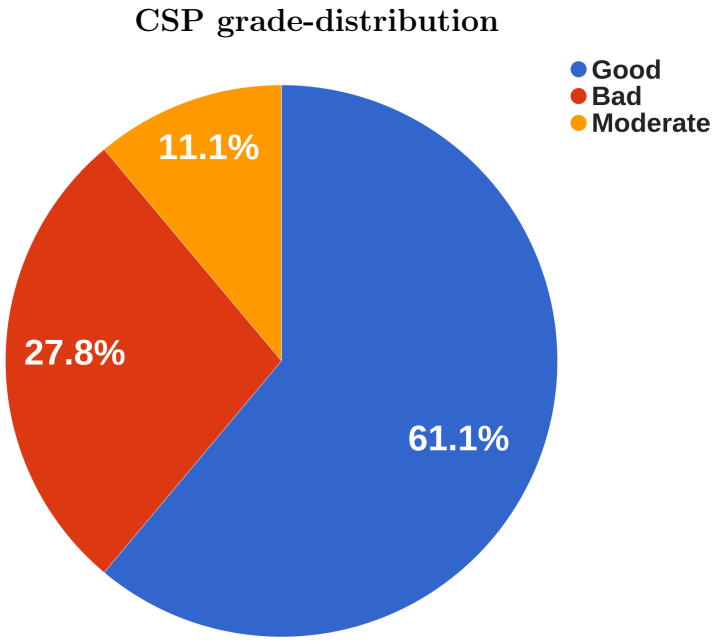

Figure 4.2: Distribution of good (61.1%), moderate (11.1%) and bad (27.8%) CSP directives for the website with the best CSP among the SGAWs.

4.2.4 X-XSS-Protection

Among the 53 websites that implemented **X-XSS-Protection**, all 53 had enabled the filter which instructs the browser to try and sanitize the malicious code. 47 of the 53 websites had set the *mode* to *block* which blocks the rendering of the website completely instead of trying to sanitize it.

Attributes	Total
Filter enabled	53
mode=block	47

Table 4.7: The number of websites with the X-XSS-Protection filter enabled and the number of websites with mode set to block.

4.2.5 X-Content-Type-Options

There were 60 websites that implemented the **X-Content-Type-Options** header, all of them used the *nosniff* attribute. This header only has one attribute so a 100% adoption rate of the particular attribute is expected. The header forces the browser

to interpret files from the server according to their MIME-type and that it should not guess it.

Attribute	Total
nosniff	60

Table 4.8: The number of websites with the X-Content-Type-Options header set with the only available attribute *nosniff* set.

4.2.6 X-Frame-Options

X-Frame-Options was found on 109 out of the 205 scanned websites and 106 websites used the *DENY* or *SAMEORIGIN* attribute. The 3 remaining websites used the deprecated *Allow-From* attribute. The header dictates what parent is allowed to load the current website in a frame.

Attribute	Total
deny/sameorigin	106

Table 4.9: The number of websites with the X-Frame-Options header set to either *deny* or *sameorigin*.

4.2.7 Set-Cookie

The **Set-cookie** header was found on 145 websites out of the 205 unique websites scanned. *HttpOnly* was present in 112 of them, *SameSite* with a value of *strict* or *lax* in 46 and lastly *secure* in 71. 7 websites has cookies with all three attributes. 37 websites has cookies with *HttpOnly* and *SameSite*. 54 has cookies with *HttpOnly* and *Secure*. And lastly only one website has cookies with *SameSite* and *Secure*. Note that the counted cookies is each site's most insecure cookie, not the total amount of cookies.

Attributes	Total
HttpOnly	112
SameSite lax/strict	46
Secure	71

Table 4.10: Total number of attributes found in the cookie with the least amount of attributes per website.

Cookies with ...	Total
All three attributes	7
HttpOnly and SameSite	37
HttpOnly and Secure	54
SameSite and Secure	1

Table 4.11: Total number of combinations of all cookie attributes found in the cookie with the least amount of attributes per website.

4.2.8 Referrer-Policy

Out of the 205 scanned websites, 39 had a **Referrer-Policy**. All of them used a policy that was not *unsafe-url*, which is the only "bad" policy. Not having a **Referrer-policy** is mostly fine since most modern browser will use a decent default policy if the header is missing.

Attribute	**Total**
Not *unsafe-url* policy	39

Table 4.12: The number of Referrer-Policy headers with an *unsafe-url* policy.

4.2.9 SRI

Subresource integrity is only found in 16 out of the 158 websites with external resources (such as javascript/stylesheets loaded from another domain than the website itself). Among the 158 websites with external resources, a total of 420 external resources was found but only 20 of those resources used SRI to ensure that the correct code was being ran. Using SRI for external resources ensures that the code will only run if it matches the cryptographic hash set in the code and if it changes, the browser will not load the external resource.

Websites	**Total**
Websites with external tags	158
Websites with SRI	16

Table 4.13: Number of websites with external resources and number of websites using SRI to some extent.

Tags	**Total**
Number of external tags	420
Number of SRIs	20

Table 4.14: Total number external tags on all websites and total number of tags with the SRI attribute.

Chapter 5

Discussion

The results of the technical survey answers **RQ3** and shows that many SGAWs are lacking in the number of security headers that have been implemented. The results also show that while some websites have implemented certain security headers, many of them are still lacking in terms of security benefit due to the bypassability of the implementations. We answered **RQ2** with the literature review and the result are described in the background chapter. The result of the literature review gave us the technical knowledge to then answer **RQ1**. The answer for **RQ1** is the method/ tool we developed as described in Chapter 3. The method is different from similar studies/tools that only look at the implementation rate of the headers rather than also looking at the implementation details of the headers.

5.1 Comparing the results

Most of the related articles recorded a CSP adoption rate of 0-2% while the findings of this study showed a CSP adoption rate of 11.2%. A reason for this could be that CSP adoption rate have increased in recent years compared to when the related articles were published. Another reason could be that the sample of this study are government agencies which may have more demands toward security. Yet another reason could be that the websites observed by the other studies could be larger and more complex websites where implementing a CSP could be difficult due to the number of included resources, while the SGAWs are generally small websites that does not include many resources, resulting in an easier implementation of a CSP that is also easier to verify. However as observed by other researchers [38], our results also show that a majority of the CSP implementations are easily bypassable due to usage of *'unsafe-'* (and similar) values.

When comparing our results with similar studies in Table 5.1, our results show a significant increase in adoption of all headers for the SGAWs compared to the top 1 million websites studies in the two articles by Lavrenovs and Melón (2018) and Buchanan et al. (2017). This can be affected by many things however. For example, the related studies were performed in 2017 and 2018. A new scan of the top 1 million websites in 2021 may reveal vastly different results. The sample size for our study is also much smaller, which means that outliers will affect the total percentage much more than the other studies with a sample size of 1 million.

Study	HTTPS	HSTS	CSP	XXP	XCTO	XFO	RP
This study	97.5%	30.2%	11.2%	25.85%	29.26%	53.17%	19.02%
Lavrenovs [16]	47.7%	7.0%	1.6%	8.91%	11.88%	12.27%	0.18%
Buchanan [4]	24.78%	5.40%	1.57%	8.31%	10.57%	11.11%	-%

Table 5.1: A comparison between studies investigating the usage of HTTP security headers. This study is completed in 2021, Lavrenovs in 2018 and Buchanan in 2017.

The two studies by Lavrenovs and Melón (2018) and Buchanan et al. (2017) show very similar results for the Content-Security-Policy (CSP), X-XSS-Protection (XXP), X-Content-Type-Options (XCTO) and X-Frame-Options (XFO) headers. This is not entirely unexpected since the time between the scans is only about one year and the CSP header can be hard to implement on existing websites. The others are largely unsupported by modern browsers which is why adoption may be lacking. The adoption of HTTPS and Strict-Transport-Security (HSTS) show a bigger increase however and is more to be expected since it is both easier to implement and offers a lot of security for the website's users. Comparatively, our study shows much bigger adoption rates overall, even the headers that are not supported by modern browsers. This could be due to accessibility reasons and that support for older browsers such as Internet Explorer is a bigger concern (where the older headers are supported and the newer ones are not) for SGAWs than the typical top 1 million website.

Most of the website we scanned did not have a well implemented set of security features. On the one hand, it is important for government agencies to secure their websites as they may host a lot of important information as well as being a more attractive target for attackers to spy on users or spread misinformation. On the other hand, in terms of XSS vulnerabilities, a largely static government website without user interaction/input may be less likely to be vulnerable to stored XSS vulnerabilities, or for them to even exist on the website. Therefore, the added security from a CSP for example may not do much for such a static website (for XSS attacks). However, there are many other aspects, such as the client-side supply chain that is very important to protect as that is not only a big issue, it is a real issue, as seen in 2018 with the supply chain hack that affected many government websites in Sweden [35].

5.2 Limitations

A limitation with the methodology is that everything is scripted with Python libraries and it is not a real web browser handling the requests. The scripts respond to the server in similar/identical ways as most browsers (in terms of the HTTP protocol) would, but since no javascript code is ran, there may be redirections missed because of that for example. Sometimes, implementations slightly differ from the standard and could therefore show slightly different results. This is true for all browser though and is therefore hard to test for since, as previously discussed, there are thousands of combinations of clients/web browsers. An improvement of the methodology could be to use something like Puppeteer which is a headless Chrome instance that is

scriptable to get similar results but with the actual implementation of a real web browser. While this is better, there can still be some discrepancies between the different browsers (Chrome, Firefox, Internet Explorer, Safari and variations of them for mobile or similar).

Another limitation with the methodology is that currently, everything is tested with a limited subset of all the various devices/browsers/ISPs/countries. A website may respond differently depending on these factors and the results could therefore be different depending on from where the research is conducted and on what devices.

The usage of customizable weights to increase and decrease the importance of certain tests adds complexity to the usability of the algorithm and comparing results between different websites is highly dependant on using the same weights. The ability to customize these weights are important though for the tool to be generalizable and usable for many different types of websites. To increase the tools usability, a graphical user interface with profiles to start from is an option to help with the usability. To accurately compare different websites, it is also very important that the weights used are provided. This may cause some confusion compared to other, similar, tools that do not offer the same customization options.

An unexpected finding during our data collection phase was that in some cases, there were multiple instances of the same header for one website. This could be an issue if the two headers are contradictory and therefore adds confusion for browsers on what header to use. According to the documentation for most headers (like the *referrer-policy* header [11]), there are instructions on which header to choose but it is unclear if all web browsers handles duplicate headers the same way. For our algorithm, we followed the documentation, but in reality, some browsers may not handle the headers in the same way making the result for the website to potentially be misleading.

5.3 Credibility of sources

Many sources are peer-reviewed and where found in trusted databases such as ACM, IEEE Xplore and Springer. To increase the credibility of the papers claims even more, we used multiple different papers/sources to confirm the claims. We also used the documentation of the protocols published by the creators, such as the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C). Furthermore, documentation created by the browser vendors such as Mozilla and Google was also used and is deemed credible since they are the ones implementing the standards in the browsers.

Chapter 6

Conclusions and Future Work

Our results show that many SGAWs are lacking in terms of implemented security headers and most importantly, security headers that are actually effective and not trivially bypassable. However, compared with previous research done in 2017 and 2018, the SGAWs have implemented these features at a higher rate than the average top 1 million websites on the internet had at that time.

The Swedish police, and the security incident where an external provider to them was hacked (referenced in Chapter 1), is now one of the best SGAW in terms of what, and how well, security headers are implemented on their website. Their security headers today would not allow for a similar attack to occur again. Unfortunately, many SGAWs are still vulnerable to these types of client-side supply chain attacks since for example, 158 of all 205 websites scanned, use external resources. These external resource may get breached in the future, but only 16 websites use an SRI (for some of their external resources) that could protect against external resources being changed.

The algorithm we developed shows that traditional website scanning tools that works by looking at whether or not an HTTP security header is implemented or not is unlikely to give a fair representation of a website's security. This is because many headers can be, and frequently are, implemented in an insecure way that can be bypassed. A Content-Security-Policy header for example, was in our study found on 23 out of the 205 scanned websites. However, only 9 of them (about 39% of the 23 websites with a CSP) had a properly secured default policy. This means that instead of the 11.2% figure (which would be given by previous tools/studies) of how many websites with a CSP existed, only about 4.4% will actually protect the website according to our algorithm (based solely on the default policy). It is therefore very important to acknowledge the implementation details when assessing the security measures taken by a website and not only looking at if the header is implemented or not.

Future work could be to expand the tool to not only test the home page of a website, but all pages/subdomains within its domain. The algorithm is built so that adding new tests is possible. Adding more tests to include upcoming headers would therefore be possible to, in the future, see if the SGAW have improved. Another avenue worth exploring could be to add a cumulative performance score through an impact assessment (essentially an average of all scores) that could be used for grading a website instead of a sum of all test scores. Another interesting approach could be to run similar surveys on bigger data sets, like all government agency websites in the EU. This could then be used to compare different countries and the general

security posture of their websites. Recreating the algorithm using a real browser as the backend instead of a Python script (using a tool like Puppeteer) could allow for some currently unsupported browser features to work (like a Javascript engine which could detect Javascript-based redirects) and make the results even more accurate.

Bibliography

[1] J. M. A. Averay, "Owasp secure headers project," OWASP, 2017, (Accessed: 01 February 2021). [Online]. Available: https://owasp.org/www-project-secure-headers/

[2] A. Barth, "HTTP State Management Mechanism," Internet Requests for Comments, U.C. Berkeley, RFC 6265, April 2011, (Accessed: 25 February 2021). [Online]. Available: https://tools.ietf.org/html/rfc6265

[3] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, ser. CCS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 75–88. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/1455770.1455782

[4] W. J. Buchanan, S. Helme, and A. Woodward, "Analysis of the adoption of security headers in http," *IET Information Security*, vol. 12, no. 2, pp. 118–126, 2018. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-ifs.2016.0621

[5] S. Calzavara, A. Rabitti, and M. Bugliesi, "Semantics-based analysis of content security policy deployment," *ACM Trans. Web*, vol. 12, no. 2, Jan. 2018. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/3149408

[6] S. Calzavara, A. Rabitti, and M. Bugliesi, "Content security problems? evaluating the effectiveness of content security policy in the wild," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1365–1375. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/2976749.2978338

[7] A. Deveria, "Can i use..." Caniuse, 2021, (Accessed: 31 March 2021). [Online]. Available: https://caniuse.com/mdn-http_headers_referrer-policy_default_strict-origin-when-cross-origin

[8] I. Dolnák, "Content security policy (csp) as countermeasure to cross site scripting (xss) attacks," in *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2017, pp. 1–4.

[9] I. Dolnák, "Implementation of referrer policy in order to control http referer header privacy," in *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2017, pp. 1–4.

[10] I. Dolnák and J. Litvik, "Introduction to http security headers and implementation of http strict transport security (hsts) header for https enforcing," in

2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA), 2017, pp. 1–4.

[11] D. Farolino, J. Eisinger, and E. Stark, "Referrer policy," W3C, 2021, (Accessed: 22 February 2021). [Online]. Available: https://w3c.github.io/webappsec-referrer-policy/

[12] Google, "How ct works," Certificate Transparency, 2021, (Accessed: 31 March 2021). [Online]. Available: https://certificate.transparency.dev/howctworks/#stepby

[13] J. Hodges, C. Jackson, and A. Barth, "HTTP Strict Transport Security (HSTS)," Internet Requests for Comments, RFC 6797, November 2012. [Online]. Available: https://tools.ietf.org/html/rfc6797

[14] L. Johnson and L. Mårtensson, "Headerscannertool," Github, 2021, (Accessed: 7 May 2021). [Online]. Available: https://github.com/headerscanner/headerscannertool

[15] S. Keshav, "How to read a paper," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 3, p. 83–84, Jul. 2007. [Online]. Available: https://doi.org/10.1145/1273445.1273458

[16] A. Lavrenovs and F. J. R. Melón, "Http security headers analysis of top one million websites," in *2018 10th International Conference on Cyber Conflict (CyCon)*, 2018, pp. 345–370.

[17] X. Li, C. Wu, S. Ji, Q. Gu, and R. Beyah, "Hsts measurement and an enhanced stripping attack against https," in *Security and Privacy in Communication Networks*. Cham: Springer International Publishing, 2018, pp. 489–509.

[18] Mozilla, "Content security policy (csp)," MDN, March 2021, (Accessed: 22 March 2021). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#browser_compatibility

[19] Mozilla, "Feature-policy," MDN, February 2021, (Accessed: 22 February 2021). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Feature-Policy

[20] Mozilla, "Http headers," MDN, April 2021, (Accessed: 22 April 2021). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

[21] Mozilla, "Mime types (iana media types)," MDN, March 2021, (Accessed: 22 February 2021). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

[22] Mozilla, "Samesite cookies," MDN, March 2021, (Accessed: 22 March 2021). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite

[23] Mozilla, "Strict-transport-security," MDN, February 2021, (Accessed: 22 February 2021). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

[24] Mozilla, "Subresource integrity," MDN, April 2021, (Accessed: 22 April 2021). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity

[25] Mozilla, "X-content-type-options," MDN, March 2021, (Accessed: 22 February 2021). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options

[26] Mozilla, "X-xss-protection," MDN, February 2021, (Accessed: 22 February 2021). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection

[27] L. Petkova, "Http security headers," *Knowledge International Journal*, vol. 30, no. 3, pp. 701–706, 2019.

[28] K. Podins and A. Lavrenovs, "Security implications of using third-party resources in the world wide web," in *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, 2018, pp. 1–6.

[29] T. C. Project, "Hsts preload list submission," Hstspreload, 2016, (Accessed: 31 March 2021). [Online]. Available: https://hstspreload.org/

[30] K. Reitz, "Python http for humans." Pypi, 2020, (Accessed: 22 March 2021). [Online]. Available: https://pypi.org/project/requests/

[31] D. Ross and T. Gondrom, "HTTP Header Field X-Frame-Options," Internet Requests for Comments, RFC 7034, October 2013. [Online]. Available: https://tools.ietf.org/html/rfc7034

[32] SCB, "Myndighetsregistret," Myndighetsregistret, 2021, (Accessed: 01 February 2021). [Online]. Available: http://www.myndighetsregistret.scb.se/

[33] D. F. Some, N. Bielova, and T. Rezk, "On the content security policy violations due to the same-origin policy," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017, p. 877–886. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/3038912.3052634

[34] S. Stamm, B. Sterne, and G. Markham, "Reining in the web with content security policy," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 921–930. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/1772690.1772784

[35] TT, "Svenska sajter kan ha utsatts för hackerattack," Expressen, 2018, (Accessed: 26 January 2021). [Online]. Available: https://www.expressen.se/nyheter/svenska-sajter-kan-ha-utsatts-for-hackerattack/

[36] S. Van Acker, D. Hausknecht, and A. Sabelfeld, "Data exfiltration in the face of csp," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 853–864. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/2897845.2897899

[37] A. P. Vumo, J. Spillner, and S. Köpsell, "Analysis of mozambican websites: How do they protect their users?" in *2017 Information Security for South Africa (ISSA)*, 2017, pp. 90–97.

[38] L. Weichselbaum, M. Spagnuolo, S. Lekies, and A. Janc, "Csp is dead,

long live csp! on the insecurity of whitelists and the future of content security policy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1376–1387. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/2976749.2978363

[39] M. Weissbacher, T. Lauinger, and W. Robertson, "Why is csp failing? trends and challenges in csp adoption," in *Research in Attacks, Intrusions and Defenses*, A. Stavrou, H. Bos, and G. Portokalidis, Eds. Cham: Springer International Publishing, 2014, pp. 212–233.

[40] M. West, "Content security policy level 3," W3C, 2021, (Accessed: 23 February 2021). [Online]. Available: https://w3c.github.io/webappsec-csp/

[41] M. West, A. Barth, and D. Veditz, "Content security policy level 2," W3C, 2016, (Accessed: 23 February 2021). [Online]. Available: https://www.w3.org/TR/CSP2/

[42] M. West and J. Wilander, "Cookies: HTTP State Management Mechanism draft-ietf-httpbis-rfc6265bis-07," Internet Requests for Comments, RFC, December 2020, (Accessed: 25 February 2021). [Online]. Available: https://tools.ietf.org/html/draft-ietf-httpbis-rfc6265bis-07

[43] C. Williams, "Uk ico, uscourts.gov... thousands of websites hijacked by hidden crypto-mining code after popular plugin pwned," The Register, 2018, (Accessed: 26 January 2021). [Online]. Available: https://www.theregister.com/2018/02/11/browsealoud_compromised_coinhive/

[44] I. Yusof and A.-S. K. Pathan, "Mitigating cross-site scripting attacks with a content security policy," *Computer*, vol. 49, no. 3, pp. 56–63, 2016.

Appendix A

Supplemental Information

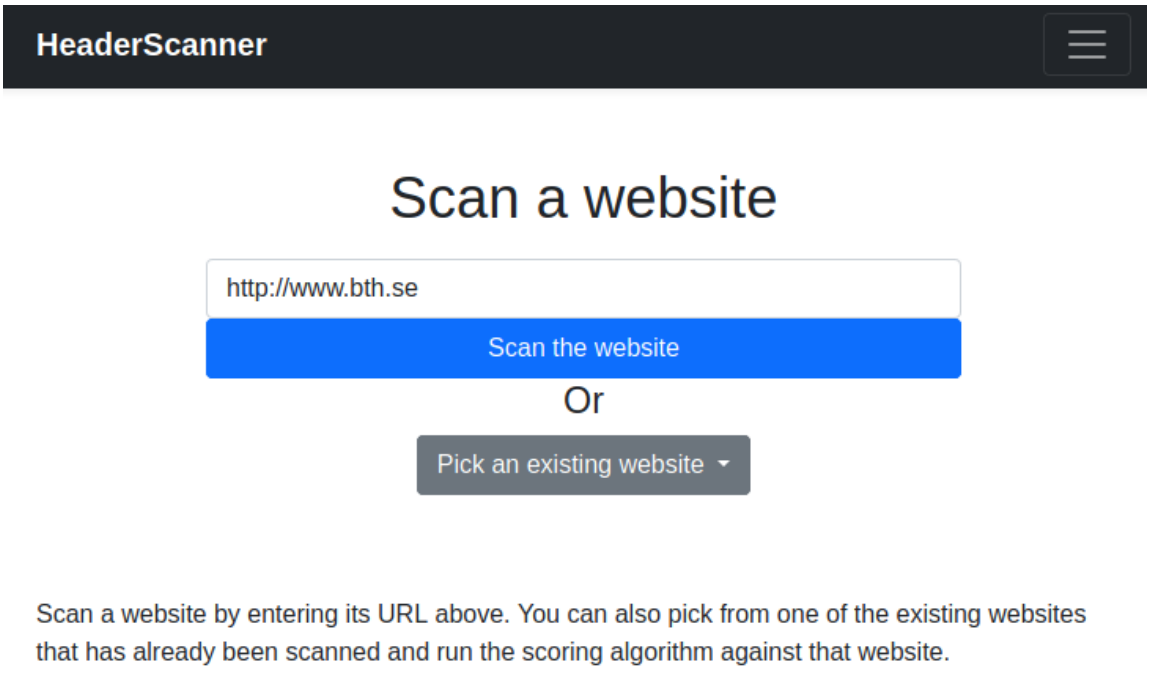

Figure A.1: Interface for starting a new scan or selecting an existing one in the GUI.

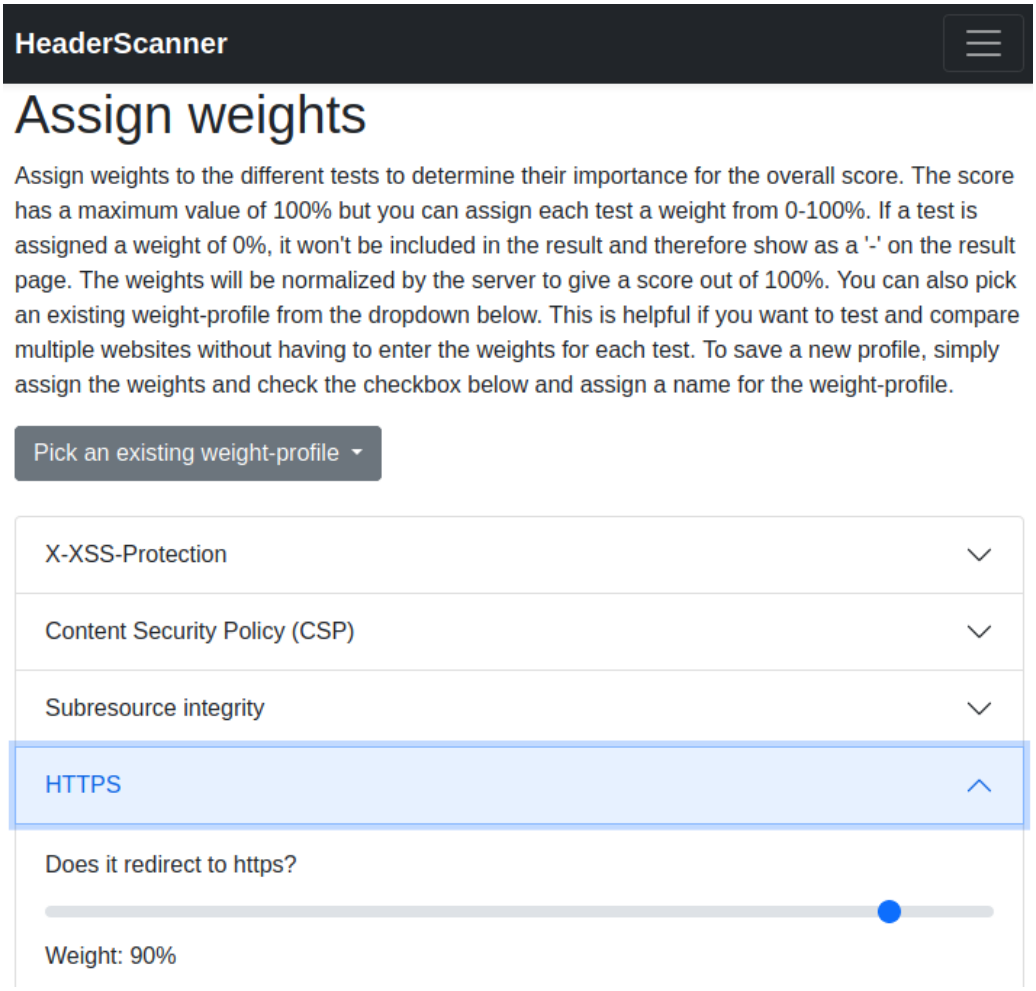

Figure A.2: Interface for setting weights for the different tests in the scan. Profile for weights can be loaded/saved.

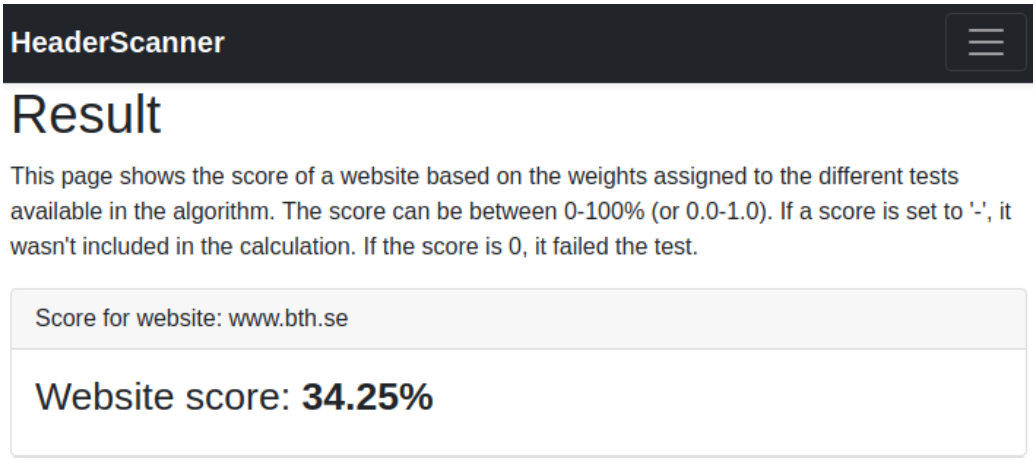

Figure A.3: Interface for the result of a scan in the GUI.

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden