The Conundrum of Declarative Security HTTP Response Headers: Lessons Learned

Aditya K Sood, Richard J. Enbody
Department of Computer Science and Engineering
Michigan State University, East Lansing, MI 48824-1226, USA
soodadit@cse.msu.edu, enbody@cse.msu.edu

Abstract

The stringency of attacks has grown simultaneously with the development of the web. To combat some of the new attacks, declarative security has been proposed in the form of HTTP response headers from the server side. The declarative model provides an extensible set of security parameters in form of HTTP responses. In this, browsers can respond with a requested security mechanism. This paper explores the state of HTTP declarative security and how it is being applied today.

1 Introduction

With the advent of new web attacks, protective strategies have had to adapt. The latest attacks such as Clickjacking [1], XSS Filter bypassing, MIME sniffing [9] have dramatically affected the World Wide Web community. No doubt XSS has been the most exploited bug in applications, but the new generation attacks such as Cross Site Request Forging (CSRF), Clickjacking and so on have transformed the nature of exploitation. The new attack patterns require new type of security mechanisms because of the vector of origination. New security solutions, named Declarative Security have been proposed and are applied at a low level in the HTTP. The main idea is to specify security in an HTTP parameter that is set by the server and is sent along to the web browser as a part of the in-line response. A browser renders the content of that web page by scrutinizing the HTTP headers and tries to invoke the specified security module in order to head off the attacks.

As the name implies, the protection parameters for a specific set of attack are *declared* by the developer as a part of the web server or application running on the server. In this way, declarative security provides both portable and flexible security defense. Most of these declarative security protection parameters are not the part of HTTP 1.1 specification, but are considered as vendor specific or customized security solutions related to a specific product. Usually, the declarative security in HTTP parameters is understood as the "X" factor protection. Most of the HTTP headers start with "X" in order to differentiate between standard HTTP 1.1 and normalized ones. Some of the headers that define the declarative security are *X-XSS-Protection, X-Frame-Options, X-Content-Type-Options, X-Download-Options, X-Content-Security-Policy*, etc. Microsoft and Mozilla have adopted this type of security. We will describe declarative security and the corresponding attacks in the next section in more detail.

This paper aims to determine the state of declarative security by testing the Alexa [11] top 1000 websites to find the penetration of this type of security. In addition, we will discuss some fallacies and problems with the declarative security model and the lessons learned during this experiment. For our analysis, we will be covering a set of declarative protection headers which have been deployed to prevent the new attack trends on the web.

2 Applied Security Model

According to the Microsoft's applied security model [12,13], implementation of security can be either imperative or declarative. The security model is deployed based on the two factors as:

- A security model is considered as *declarative* [12], if the security features are implemented in the application as a parameter which is used at the run time.

- A security model is considered as *imperative* [13] , if the security features such as permissions are allowed to generate or invoke a new object based on the requirements of the code.

The major difference between these two security models is that imperative security is used to perform operations based on the demands and overrides, whereas declarative security issues requests. Declarative security has the advantage that the 'request' model allows easier integration within existing implementations as well as the flexibility to adapt to new attacks.

3 HTTP Declarative Security

Recent web attacks have proven to be difficult to handle in the real time. Examples include Clickjacking, MIME sniffing, manipulating file downloads, CSRF, and XSS variants. In response, the declarative security model has been proposed. It uses an HTTP header that has to be declared on the server side by the developer or administrator. The HTTP header triggers a response in the browser to deploy requested security on the client side.

Browsers play a critical role in determining the success of these types of attacks, so it is required that servers and client browsers coordinate to fend off the attacks. To facilitate coordination, browsers are modified to recognize declarative security headers in HTTP headers sent by servers. Once a declarative security HTTP header is detected, the browser will execute the requested security mechanism. Of course, if the server does not serve up HTTP headers with declarative security parameters,no protection can be implemented. Similarly, if browsers cannot recognize the headers, no protection can be deployed.

3.1 Types of Threats

The HTTP header parameters are specific to particular types of attacks. Here we examine a set of attacks (and protections) that can be specified in the headers.

3.1.1 Clickjacking Attacks: The clickjacking attacks [1] are a new class of attacks that use differential techniques to hide or obscure an element for trapping the user to visit destinations that he did not intend to visit. They use differential techniques such as compressed frames as hidden or transparent, trapping mouse events [2], or UI Redressing [3] primarily to trap the user. A number of protection mechanisms have been implemented including frame bursting codes [4], or restricting the frame execution. Microsoft introduced a protection feature in HTTP headers as a part of declarative security. The *X-FRAME-OPTIONS* [5] sets a restriction on the framing of a web page for a particular domain. It uses the value *DENY* and *SAMEORIGIN* for rendering the contents into a child frame. It is possible to stop the rendering completely in a child frame using *DENY* as a parameter. The *SAMEORIGIN* parameter declares that the content can only come from the parent site and that no third party content rendering is allowed.

3.1.2 CSRF and XSS Attacks: The advanced level of Cross-Site Scripting (XSS) attacks in collaboration with the Cross Site Request Forging (CSRF) has had a dramatic impact on the World Wide Web. In order to protect against this attack, companies such as Mozilla and Microsoft have introduced a declarative security parameter. Mozilla has introduced Content Security Policy (CSP) [6]. The CSP provides the HTTP header *X-Content-Security-Policy* which is defined by a particular site in order to define the characteristics of the content to be rendered. In particular, the header specifies what content can be trusted. To prevent reflective cross site scripting attacks, Microsoft implemented a protection feature in the form of HTTP header *X-XSS-Protection*[8]. If a website specifies this HTTP header, the browser will trigger the XSS filter and stop the rendering of the content in Internet Explorer.

3.1.3 MIME Scripting Attacks: An unrestricted content type included from a third party can disrupt the robustness of a web application resulting in security problems. There are a number of attacks that utilize the Content-Type header to replace specific content elements with malicious code to be downloaded and rendered in the browser. The malicious code gets executed in the context of the domain which can result in stealing of data. Microsoft has introduced a protection feature in the form of HTTP header as *X-Content-Type-Options* [9] for subverting the MIME sniffing attacks. The *nosniff* parameter stops the rendering of image as a MIME object and transforms it into plain text format to avoid the execution of scripts.

3.1.4 File downloading Scripting Attacks: These types of attacks are commonly used for exploitation. It is based on the fact that some browsers such as Internet Explorer use inbuilt functionality to open the files directly from the domain while downloading . As a result, any malicious file that opens directly from the domain results in untamed output which is a result of the scripts present in it. Microsoft has introduced a new protection feature as a part of its declarative security as *X-Download-Options* [9] which stops the opening of the files directly from the domain. The browser removes the file-opening control from the download box when it encounters a *noopen* parameter in the *X-Download-Options* as a part of the HTTP response.

4 Experiment

What is the state of declarative security? Are servers making these declarations? Are clients responding? As pointed out earlier, action is needed from both the server and client sides in order for this technique to work.

4.1 Objectives

Here is a list of objectives that what we want to achieve:

1. *To determine how widely adopted declarative security is for new attacks.*
2. *To determine whether server softwares from different vendors are using the solutions by default.*
3. *To examine the effectiveness of the combined security solution between the client side and the server side.*
4. *To estimate the risks to the business in the online world.*

4.2 Testbed

For our experiment, we used Alexa [11] for a list of the top 1000 websites across a number of different domains. Since their ranking is based on web traffic we will be examining sites that are not only popular, but also attractive to malicious users. The number of users associated with a specific domain indicates the severity of risk and exploitation if an attack happens. The protection factor can be determined only based on the deployed security solutions. In addition, we have tested some popular websites that are not on the list.

We wrote a small PERL tool named *x_enum* to analyze websites. Since this experiment is entirely based on the protection features implemented in HTTP headers, we resorted to low level HTTP debugging to analyze the responses. The tool is designed in two modules. The first module traps the responses and performs pattern matching to detect the presence of a particular security parameter. The second module examines the behavior of HTTP headers. Based on the output, we are able to determine the type of attack that is sanitized by the domain through these headers.

The experiment is strictly based on the security that is declared in the HTTP headers and does not cover the standard security modules used on the client side. It is possible to have a number of countermeasures for a particular attack, but this analysis only covers the security solutions implemented at a low level in the HTTP protocol.

4.3 Results

The results indicate that very few websites out of the Alexa [11] top 1000 websites are using any of the declarative security headers when a request is initiated for the default web page and other linked pages at the time of testing. Out of the 1000 websites containing over 15,000 pages, that were put to test, we have not detected any website that is using *all* the parameters. We also tried some intensive tests on the sub-domains and mail service providers on the popular websites such as Google, Yahoo, Ebay, etc.

Table 2 shows the acceptance level of declarative security in HTTP response headers in the top 1000 websites. A "Yes" indicates that at least one type of declarative security header was used. Only one site out of the top 25 uses any declarative security headers and only 7 of the top 25 use headers. Overall, only 8 of the top 1000 use any declarative security headers. At the bottom of the table we have included a few security-oriented sites to indicate that usage does exist outside the top 25.

Table 1 shows the applied declarative security by different web servers that are mostly used in top 1000 domains ranked by Alexa. We have looked at the type of web servers that are used extensively. There is a possibility of existence of different web servers apart

Web Server	Headers
GSE	Yes
Apache/1.3.41.fb2	None
Apache	Yes
YTS	None
MS-IIS	None
Sun-JS-Web-Server	None
sffe	Yes
Apache-Coyote	None
Server	None
GFE 2.0	Yes
BWS/1.0	None
Resin/2.1.17	None
Ning HTTP Server 2.0	None
nginx	None
IBM_HTTP_Server	None
KONICHIWA/1.0	None
Cafe	Yes
AmazonS3	Yes

Table 1: Types of Web servers used in the top 1000 Websites and their declarative security header usage.

from the web servers listed in the Table 1. The *Google's GSE* server is using most of HTTP header protection parameters during monetary transactions. This has been noticed in Gmail and Google checkout sub-domain. The response is evaluated as

HTTP/1.1 200 OK
Cache-Control: no-cache, no-store, max-age=0
Pragma: no-cache
Expires: Fri, 01 Jan 1990 00:00:00 GMT
Date: Wed, 14 Apr 2010 19:46:54 GMT
Content-Type: text/html; charset=UTF-8
Set-Cookie: [Cookie value is truncated]
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Server: GSE

The *Yahoo Traffic Server (YTS)*, used for Yahoo mail and other service websites, does not provide any signs of usage of declarative security. The *Rediff Apache Server* and *Microsoft IIS 6.0/7.0* used for Rediffmail and Hotmail respectively, are completely ignoring the HTTP header protection parameters. The *s0.2mdn.net* and **.ytimg.com* hosts images and flash files that are used in major websites such as Youtube, Amazon, Hotmail etc. for advertisement purposes. It uses *X-Content-Type-Options:nosniff* in its responses for preventing MIME sniffing. The Youtube's *Apache* server, Google Advertisement *Cafe* server and

Rank	Website	Server	Headers
1	google.com	GSE 2.0	Yes
2	facebook.com	Apache/1.3.41.fb2	None
3	youtube.com	Apache	Yes
4	yahoo.com	YTS/1.18.4	None
5	live.com	MS-IIS/7.0	None
6	wikipedia.org	Sun-JS-Web-Server/7.0	None
7	baidu.com	BWS/1.0	None
8	blogger.com	sffe	Yes
9	msn.com	Microsoft-IIS/6.0	None
10	qq.com	nginx/0.6.39	None
11	twitter.com	Apache, hi	None
12	yahoo.co.jp	YTS/1.16.4	None
13	google.co.in	GWS	Yes
14	taobao.com	Apache	None
15	google.de	GWS	Yes
16	google.com.hk	GWS	Yes
17	wordpress.com	nginx	None
18	sina.com.cn	Apache/2.0.54 (Unix)	None
19	google.co.uk	GWS	None
20	amazon.com	Server	None
21	myspace.com	MS-IIS/7.5	None
22	microsoft.com	MS-IIS/7.5	None
23	google.fr	GWS	Yes
24	bing.com	Microsoft-IIS/6.0	None
25	ebay.com	Apache-Coyote/1.1	None
...			
62	orkut.com	GFE 2.0	Yes
...			
1000+	adsense.google.com	Cafe	Yes
1000+	gmail.com	GSE 2.0	Yes
1000+	ha.ckers.org	Apache	Yes
1000+	noscript.net	Apache	Yes
1000+	flashgot.net	Apache	Yes
1000+	eyeviewdigital.com	AmazonS3	Yes

Table 2: Applied declarative security as HTTP response headers in top 1000 websites.

Blogger's *sffe* server also implement this option.

The Facebook *Apache/1.3.41.fb2* , Twitter's *Apache and Server hi* , MySpace's *Microsoft IIS 7.5* servers do not indicate usage of declarative parameters, but Orkut's *GFE 2.0* server uses *X-XSS-Protection:0* as one of the response header. Further, Orkut uses *GSE* server for collaborative work. Popular websites such as *NetFlix, Flickr, Ning, Photobucket, Linkedin, Internet Movie Database (IMDb), Paypal, Skype* are not using declarative security specific HTTP response headers.

Further, we looked at a number of websites having a high page rank apart from the top 1000 list from Alexa

[11]. We noticed that some of the websites that are popular in the security community provide us with some good signs. The *ha.ckers.org, noscript.net, flashgot.net* shows the use of declarative security as:

HTTP/1.1 200 OK
Date: Thu, 15 Apr 2010 17:45:28 GMT
Server: Apache
Content-Script-Type: text/javascript
X-FRAME-OPTIONS: SAMEORIGIN
X-XSS-Protection: 0
Vary: Accept-Encoding,User-Agent
Connection: close
Content-Type: text/html; charset=UTF-8

In summary, we have not seen much variations as few commercial websites include any HTTP headers that contain the specific set of declarative security parameters. We noticed that even some of the most popular and recognized security companies, banks, and organizations have not adopted this solution. It appears that declarative security is not widely implemented on the server side.

5 Lessons Learned

During the course of this testing, we did not encounter any serious problems, but there were certain issues that made the testing a time consuming process. The handling and generation of responses depend a lot on the nature and design of web application including configuration of the web server. During our research, websites using HTTP to HTTPS redirection produced long delays in the responses and even experienced hanging of processes. We found that different web servers handle the request differently and hence showed differential behavior in their response. A lot of variance has been noticed during the course of this experiment. In the end, with the appropriate manual effort and automated solutions, the tests have been completed successfully. We conclude:

5.1 The main commercial web servers, such as *IIS* and *IBM_HTTP_SERVER*, have not implemented attack-specific HTTP headers in their default responses.

5.2 Our analysis shows that websites running open and customized web servers such as *Apache, GWE, GWS, Cafe* are the only ones deploying the security parameters in HTTP headers. Declarative security appears to be *"Security by Choice"*. On the other hand, many open source web servers have not implemented these security solutions by default.

5.3 Our analysis has shown that awareness may be a roadblock in the effective development of an optimal security solution. Looking at the results, we speculate that there are three main issues that impact the awareness. The first one may be that the developers are not well educated about the existence of the new types of web attacks. In the absence of awareness about the attacks, it is hard to expect an appropriate and robust solution to be deployed. Second, changes are required in a huge number of web applications and the deployed hosting infrastructure to strengthen the security. Third, a problem may lie in the acceptance of these solutions as being robust. For example, most of the HTTP header based solutions are vulnerable to classic *HTTP Response Splitting* [10] attacks, but these can be circumvented by applying additional security measures.

5.4 A declarative solution requires participation of both the server and the client. If either is not participating, no security action is taken. From our results, it is apparent that there is little participation on the server side so the risk of these web attacks remains high.

5.5 While we found that the declarative security solutions are not widely implemented, we have noticed some evidence of acceptance of declarative security in security-oriented sites.

6 Conclusion and Future Work

We have taken a look at one of the issue of getting an optional security mechanism embedded in HTTP response headers: adoption. A declarative solution requires participation of both the server and the client. There are pros and cons of such a protocol. An advantage is that if few major browsers implement the client side, the burden then falls on the server side. However, the downside is the same: the burden is now on the server side, and that is where we find ourselves.

Briefly, some browsers have it, but few servers are serving it. In this case, half a solution is no solution, but

it is early in the game.

Once both servers and clients are fully compliant, the open question becomes: how effective will declarative headers be to circumvent this class of attack? There is hope that the flexibility of this approach may allow fast response to any weaknesses.

Finally, if this approach proves to be successful, what other classes of attack will this be appropriate for?

Acknowledgment: A sincere thanks to SecNiche Security [14] and Armorize [15] for supporting and giving me enough time to pursue work of this kind.

7 References

[1] Clickjacking - Robert R Hansen and Jeremiah Grossman. http://sectheory.com/clickjacking.htm, 2009

[2] Blog Article - More towards Clickjacking, Aditya K Sood. http://zeroknock.blogspot.com/2009/02/more-towards-clickjacking-simulating.html, 2009

[3] Browser Security Handbook- UI Redressing. http://code.google.com/p/browsersec/wiki/Part2

[4] Frame Bursting Technique. Source Wikipedia - http://en.wikipedia.org/wiki/Framekiller

[5] Blog Article - ClickJacking Protection Feature. http://blogs.msdn.com/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx

[6] Complete Details - Content Security Policy. http://people.mozilla.org/~bsterne/content-security-policy/details.html

[7] Specification Details - Content Security Policy. https://wiki.mozilla.org/Security/CSP/Spec

[8] XSS Filter Details - Reflected XSS Protection. http://blogs.msdn.com/ie/archive/2008/07/01/ie8-security-part-iv-the-xss-filter.aspx

[9] IE Blog - Comprehensive Protection Parameters http://blogs.msdn.com/ie/archive/2008/07/02/ie8-security-part-v-comprehensive-protection.aspx

[10] Details about HTTP Response Splitting Attacks. http://www.securiteam.com/securityreviews/5WP0E2KFGK.html

[11] Alexa Top Sites. http://www.alexa.com/topsites

[12] MSDN - Microsoft Declarative Security Model. http://msdn.microsoft.com/enus/library/kaacwy28.aspx

[13] MSDN - Microsoft Imperative Security Model. http://msdn.microsoft.com/enus/library/0xkh23z7.aspx

[14] SecNiche Security - http://www.secniche.org

[15] Armorize Technologies - http://www.armorize.com