Analysis of HTTP Security Headers in Turkey

Koray Emre KISA, Emin İslam TATLI‡

Security Architecture, Türk Telekom, Ümraniye, İstanbul

‡Corresponding Author; Address: Çakmak Mah., Balkan Cad. No:49, Ümraniye-İstanbul, Tel: +90 212 460 1 500, e-mail: emre.kisa@turktelekom.com.tr; dr.emin.tatli@turktelekom.com.tr

Abstract- Web applications are targeted during cyber-attacks in order to get unauthorized access or manipulate sensitive data. Developers are expected to leverage secure coding best practices to protect their web applications. Over the last few years, browser vendors have integrated certain security header controls to support web application security. If these headers are enabled by developers, browsers check values of these header parameters and prevent certain attacks automatically. In this research, we analysed the existence of the common security headers within 8279 different URLs of 361 popular Turkish web portals from 18 different categories. The analysis results have shown that security headers are not utilized by most web developers and even critical web portals do not implement required security headers. This paper explains our contribution by providing the details of the HTTP Security headers, the attack types they can prevent, the analysis tool we have implemented and the analysis results.

Keywords- HTTP Security Headers, Web Security, Cyber Security Analysis, Large-scale Analysis.

1. Introduction

Application security is one of the most critical non-functional requirements for enterprises and organizations on the web. Several security controls including authentication, authorization, input validation, output encoding, etc. are expected to be implemented securely to protect web applications. Otherwise, even script-kiddies can use freely available hacking tools (e.g. sqlmap) and get unauthorized access to sensitive data easily. As an example, in February of 2015 the official website of Martin Schulz, the president of European Parliament, was hacked by unknown individuals who leaked the content of several internal databases using SQL Injection vulnerability [1].

It is therefore vital that web developers are trained for secure coding patterns and principles and utilize this knowledge during design and coding. In parallel, security experts try to integrate security solutions into development frameworks in order to support developers and simplify integration of security controls. In the recent years, browser vendors including Microsoft (IE), Firefox (Mozilla) and Google (Chrome) have followed this approach as well and integrated several security controls as HTTP response headers into their browsers. In order to activate these security controls, developers are required to append relevant security headers into HTTP responses. At this point, the following question raises: "Are web developers aware of these security headers and do they utilize them?"

In our research, we wanted to enlighten the answer of this question. Therefore, we performed an automated usage analysis of HTTP security headers of most popular web portals in Turkey and evaluated the statisticsal results to increase awareness of web developers and security experts. This paper extends our previous study [11] where we analysed only the main web pages of 361 web portals.

The paper is organized as follows: Section II explains the most common application security

risks and the same-origin-policy concept. The HTTP security headers are designed to prevent these common security risks. Section III gives a detailed explanation of all HTTP security headers. Our analysis results and the implemented tool are explained in Section IV. Section V concludes the paper and explains the future work.

2. Application Security Risks

Today there exist several types of attacks that target web applications. Organizations like OWASP (Open Web Application Security Project) Community and SANS Institute present the most critical application security risks as Top-10 [2] and Top-25 [3] lists respectively.

OWASP community aims to increase awareness on web application security. They created freely-available articles, methodologies, documentation, tools and technologies in the field of web application security. Its most popular project is OWASP Top 10 list which was first published in 2003 and is updated every three years. The goal of this project is to identify the most critical and common web application security risks. Figure 1 shows the current version (2013) of the Top 10 list.

The HTTP security headers that we explain in the following section in detail aim to prevent certain cyber-attacks that are included in the OWASP Top 10 risks list.

Fig. 1. OWASP Top 10 (2013).

Same-Origin policy concept is also vital to understand security mechanisms within browsers. This concept guarantees that a web browser permits scripts contained in one web page to access data in another web page, but only if both pages have the same origin. An origin is defined as a combination of URI scheme, hostname, and port number. This policy prevents a malicious script on one page from obtaining access to sensitive data on another web page through that page's Document Object Model (DOM). This concept is the core of web application security that extensively depend on HTTP Cookies to maintain authenticated user sessions. Web browsers that comply with Same-Origin policy must be able to strictly prevent sharing of data between pages on client-sides to prevent loss of data confidentiality and integrity. Figure 2 shows some examples of access restrictions based on different URLs.

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://username:password**@www.example.com**/dir2/other.html	Success	Same protocol, host and port
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

Fig. 2. Same-Origin Policy restrictions

3. HTTP Security Headers

The HTTP Security Headers are mostly defined by IETF under different standards. IETF (Internet Engineering Task Force) is an open standards organization that voluntarily develops and promotes Internet Standards. All participants are volunteers, though their work is usually funded by their employers or sponsors such as Google, Mozilla and Microsoft.

The HTTP Security Headers whose details are explained below are as follows; *CSP (Content Security Policy), X-XSS-Protection, X-Frame-Options, HSTS (HTTP Strict Transport Security), X-Content-Type-Options, X-Download-Options, X-Permitted-Cross-Domain-Policies, X-Public-Key-Pins, Cookie Flags (httpOnly, Secure)*.

3.1.Content Security Policy Header (CSP)

Content Security Policy specification defines a mechanism by which web developers can control resources that a particular page can fetch or execute, as well as a number of security-relevant policy decisions. The policy language specified in the IETF specification consists of an extensible set of directives, each of which controls a specific resource type or policy decision.

This header has three different versions. The first version was released in November 2012 [4].

Web browser versions supporting the first CSP version are as follows;

- **Content-Security-Policy**: Chrome v25+, Firefox 23+, Opera 19+, Safari 7+, Microsoft Edge 12 build 10240+.
- **X-Content-Security-Policy:** Internet Explorer 10+, Firefox 4+
- **X-Webkit-CSP:** Chrome v14-v25, Safari 6+

CSP Version 1

CSP version 1 supports the following directives;

Default-src: This directive specifies the default source for other directives, unless they have explicitly a defined source.

Script-src: This directive restricts which scripts the protected resource can execute. This directive should be used with care because most web sites rely on third party scripts such as Google Analytics and Social Media Integrations to function properly. Unless explicitly defined, scripts from these sources will fail to execute, causing website malfunction.

The directive also has two sub-directives '*unsafe-inline*' and '*unsafe-eval*'.

If '*unsafe-inline*' sub-directive is not explicitly defined in script sources:

- Whenever the user agent is expected to execute an inline script (either from a script element or from an inline event handler), instead the user agent will not execute script.
- Whenever the user agent is expected to execute script contained in a javascript URI, instead the user agent will not execute the script.

If '*unsafe-eval*' sub-directive is not explicitly defined in script sources:

- Instead of evaluating their arguments, both operator eval and function eval will throw a security exception.
- When called as a constructor, the function Function will throw a security exception.
- When called with a first argument that is non-callable (e.g., not a function), the setTimeout function will return zero without creating a timer.
- When called with a first argument that is non-callable (e.g., not a function), the setInterval function will return zero without creating a timer.

Object-src: This directive restricts from where the protected resource can load plugins.

Style-src: This directive restricts which styles a user applies to protected resource.

If '*unsafe-inline'* sub-directive is not explicitly defined in style sources:

- Whenever the user agent would apply style from a style element, instead the user agent will ignore the style.
- Whenever the user agent would apply style from a style attribute, instead the user agent will ignore the style.

Img-src: This directive restricts from where the protected resource can load images.

Media-src: This directive restricts from where the protected resource can load video and audio.

Frame-src: This directive restricts from where the protected resource can embed frames.

Font-src: This directive restricts from where the protected resource can load fonts.

Connect-src: This directive restricts which URIs the protected resource can load using script interfaces.

- Processing the open() method of an XMLHttpRequest object.
- Processing the WebSocket constructor
- Processing the EventSource constructor

Sandbox: This directive specifies an HTML sandbox policy that the user agent applies to the protected resource.

Report URI: This directive specifies a URI to which the user agent sends reports about policy violation.

CSP Version 2

CSP version 2 is mostly compliant with the previous version. Version 2 adds support for a number of new directives and capabilities which are explained below **Error! Reference source not found.**.

a. Base-URI: This directive restricts the URLs that can be used to specify the HTML document's base URL. The base URL is used throughout the HTML document for relative URL addresses.

b. Child-src: This directive defines the valid sources for web workers and nested browsing contexts loaded using elements such as <frame> and <iframe>. This directive is preferred over the frame-src directive explained above which is deprecated in version two.

c. Form-action: This directives specifies valid endpoints for <form> submissions.

d. *Frame-ancestors:* This directive specifies valid parents that may embed a page using the <frame> and <iframe> elements.

e. *Plugin-types:* This directive specifies the valid plugins that the user agent may invoke.

The most important contribution of CSP version 2 is that individual inline scripts and stylesheets can be whitelisted via nonces.

CSP Version 3

CSP version 3 is currently under development. Some of the noteworthy upcoming draft changes are as follows;

- Frame-src directive which was deprecated in version is undepreciated. Worker-src directive has been added.
- Unsecure http URL's now match their secure https variants.
- Manifest-src has been added.
- Report-uri directive has been deprecated in favor of report-to directive.

3.2.X-XSS-Protection Header

This header enables XSS protection mechanism of the user agent [6]. The possible values of this header are given in Table 1.

Table 1. X-XSS-Protection Header Values

Value	Description
0	Filter disabled.
1	Filter enabled. If a cross-site scripting attack is detected, in order to stop the attack, the browser will sanitize the page.
1; mode=block	Filter enabled. Rather than sanitizing the page, when a XSS attack is detected, the browser will prevent rendering of the page.
1; report=http://[MY DOMAIN]/MY_URI	Filter enabled. The browser will sanitize the page and report the violation. This is a Chromium function utilizing CSP violation reports to send details to a URI of your choice.

3.3.X-Frame-Options Header

X-Frame-Options response header improves protection of web applications against clickjacking attacks. It declares a policy communicated from a host to a client browser on whether the browser must not display the transmitted content in frames of other web pages [7]. The possible values of this header are given in Table 2.

Table 2. X-Frame-Options Header Values

Value	Description
deny	No rendering within a frame.
sameorigin	No rendering if origin mismatch.
allow-from: DOMAIN	Allows rendering if framed by frame loaded from DOMAIN.

3.4.HTTP Strict Transport Security Header

HTTP Strict Transport Security (HSTS) is a web security policy mechanism which helps to protect websites against protocol downgrade attacks and cookie hijacking. It allows web servers to declare that web browsers (or other complying user agents) should only interact with it using secure HTTPS connections, and never via the insecure HTTP protocol. HSTS is an IETF standard track protocol and specified in RFC 6797 [8]. A server implements a HSTS policy by supplying a header (Strict-Transport-Security) over a HTTPS connection because HSTS headers over HTTP are

ignored. The possible values of this header are given in Table 3.

Table 3. HSTS Header Values

Value	Description
max-age=SECONDS	The time, in seconds, that the browser should remember that this site is only to be accessed using HTTPS.
includeSubDomains	If this optional parameter is specified, this rule applies to all of the site's subdomains as well.
preload	With this option enabled, URL of your website will be hardcoded into browser code such that it will only communicate over secure HTTPS channel.

If "*preload*" option explained above is set, the following requirements must be satisfied;

- Have a valid CA approved certificate,
- Redirect all HTTP traffic to HTTPS,
- Serve all your subdomains over HTTPS,
- Serve an HSTS header on the base domain for HTTPS requests
- The max-age must be at least eighteen weeks (10886400 seconds).
- The includeSubDomains directive must be specified.
- The preload directive must be specified.
- If you are serving an additional redirect from your HTTPS site, that redirect must still have the HSTS header (rather than the page it redirects to).

After meeting these requirements, you can submit your website URL to https://hstspreload.appspot.com/. Approved URLs will be hardcoded into next release of supported web browsers so that the browser will use secure channels only.

Table 4. HSTS Browser Support

IE	Edge	Firefox	Chrome	Safari	Opera	Android
11	13	46	50	9.1	37	50

3.5.X-Content-Type-Options Header

It provides protection from MIME content-sniffing attacks against Chrome and Internet Explorer. Some browsers try to guess the type of file even though it has a declared content type. This behaviour could be abused by hackers to upload malicious files. Setting the header to nosniff disables the content sniffing feature.

3.6.X-Download-Options Header

This security header is specific to Internet Explorer 8+. This header should be used as a defence-in-depth measure. When you deal with user created content, an attacker exploiting a vulnerability on your website might be able to run scripts on users' browser. By setting this header to noopen, browsers are forced to download a file rather than executing its content.

3.7.X-Permitted-Cross-Domain-Policies Header

This header is used to limit which resources Adobe Flash and PDF documents can access on your domain. If you don't want to share any content with others, you should have no crossdomain.xml file on your server and send the X-Permitted-Cross-Domain-Policies "none" header with each response.

3.8.X-Public-Key-Pins

Certificate pinning is the process of associating a host with their expected certificate or public key. The HTTPS web server serves a list of public key hashes. Web browser connecting to that server expect server to use one or more of those public keys in its certificate chain. Successful implementation of this security measure greatly reduces the risk of a man-in-the-attack **Error! Reference source not found.**

Table 5. X-Public-Key-Pins Header Values

Value	Description
pin-sha256="<sha256>"	The quoted string is the Base64 encoded Subject Public Key Information (SPKI) fingerprint. It is possible to specify multiple pins for different public keys. Some browsers might allow other hashing algorithms than SHA-256 in future.
max-age=SECONDS	The time, in seconds, that the browser should remember that this site is only to be accessed using one of the pinned keys.
includeSubDomains	If this optional parameter is specified,

	this rule applies to all of the site's subdomains as well.
report-uri="<URL>"	If this optional parameter is specified, pin validation failures are reported to the given URL.

3.9.Cookie Flags

Although this header is not a security header itself, it is worth mentioning set-cookie security flags. Web applications extensively depend on cookies to maintain authenticated user sessions. If one can successfully steal a user's cookie, they can act on their behalf. Since cookies play such an important role in session management, specific controls have been implemented by web browsers to secure it.

Table 6. Cookie Values and Descriptions

Value	**Description**
Set-cookie $RANDOM	Used to create a cookie on user's web browser with given random value.
Set-cookie $RANDOM; HttpOnly	If HttpOnly flag is set, scripts on a web page is forbidden to access the cookie by the web browser.
Set-cookie $RANDOM; HttpOnly; Secure;	If Secure flag is set, web browser will only send the cookie information over secure HTTPS channel denying any other attempt.

As of 29th of March, Google Chrome has started to support a new cookie flag named SameSite to prevent CSRF.

Value	**Description**
Set-Cookie: CookieName=CookieValue; SameSite=Strict;	As the name suggests, this is the option in which the Same-Site rule is applied strictly. When the SameSite attribute is set as Strict, the cookie will not be sent along with requests initiated by third party websites.
Set-Cookie: CookieName=CookieValue; SameSite=Lax;	When SameSite attribute is set to Lax, the cookie will be sent along with the GET request initiated by third party website. So it must cause a top level navigation and a url change on the address bar to be able to submit the Cookie.

4. The Analysis of Security Headers in Turkey

We aimed to evaluate usage statistics of the aforementioned security headers in Turkey and performed an automated analysis. We picked up Alexa Top 500 Turkey websites and eliminated global web portals (e.g. google.com). In the final list, 361 web portals remained for the analysis. This list includes many commercial, governmental and news websites. We also categorized their URLs and results similar to Alexa's categorizations such as Shopping, Internet, Press etc. To extend our previous work [11], we also identified any links on the given home page by extracting subdomains of the given target URL. Extracted URL list included 8279 different URL's for the relevant 361 home pages. 8279 URL was pointing to 1274 different subdomains in total. We merged the results of subdomains under each main domain.

For our analysis, we developed a tool in python namely SecurityHeaderChecker which can be accessed freely on Github [10]. This tool is given a set of URLs as input file. It then visits each URL and checks existence of the relevant security headers and cookie flags. It is important to note that the tool neither attempts to login to the given URLs nor finds the subdomains itself.

Table 7. Security Headers Usage Statistics in Turkey

Header	**Used in # of subdomains / Total # of subdomains**	**Percentage**
No security related headers found	465 / 1274	**36%**
HTTP Only Cookies	446 / 1274 *	**35%**
Secure Cookies	73 / 287 **	**25%**
Strict-Transport-Security	61 / 287 ***	**21%**
X-Frame-Options	153 / 1274	**12%**
X-Content-Type-Options	86 / 1274	**7%**
X-XSS-Protection	69 / 361	**19%**
Content Security Policy	21 / 361	**6%**

* 1274 of 8279 domains used cookies on their home pages.
** 287 of 8279 domains used both HTTPS and Cookies on their main pages.
*** 287 of 8279 websites used HTTPS on their main pages

Header	Used in # of subdomains / Total # of subdomains	Percentage
X-Download-Options	8 / 361	**2%**
Public-Key-Pins	0 / 287	**0%**

Table 7 shows the results of our analysis. Our previous analysis showed us that if we only consider main home pages, more than half of the analysed websites do not use any of these security headers. However, the new results show that if we examine also subdomains the usage ratio drops to 36%. So we can say that developers are tend to use security headers on subdomains rather than home pages. Secure and httpOnly security flags are used the most. Developers seem to think that cookie flags will keep them secured. Our previous research showed that HttpOnly flag usage was calculated to be 60% on homepageonly analysis. However, here we can say that developers are tend to use this flag on their home pages rather than on the subdomains because the ratio of the flag usage dropped to 35% since we included the subdomains into calculation.

We found that 21% of all HTTPS websites implemented the Strict-Transport-Security header, however only 12 subdomains has made it into the preload list.

Humans are the weakest link in the security chain and clickjacking attacks targeting humans can be prevented by using X-Frame-Options header. However, we see that only 12% implemented it. We believe that weakest links should have been better protected.

About the X-XSS-Protection header, unless it was explicitly disabled this feature is expected to be enabled by default. We believe this may explain the low ratio of 19% only. We see that 90% of the time X-XSS-Protection was used, it was used in block mode. However, one of the websites used the header to explicitly disable the protection filter by setting it to 0!

Even though only 6% of subdomains have implemented Content Security Policy it is a great improvement over the %2, which was the ratio of homepageonly analysis. Hopefully, this paper will increase awareness to implement a Content Security Policy on web pages.

Even worse is that no website ever integrated Public Key Pins header.

To further improve our research we included categories of these websites into our work. This allowed us to see which sectors make use of these security headers the most, giving us the idea about that sector's security maturity level.

We developed a rating system to be able to compare sector maturities. If a website subdomain has any of the security headers explained above, it gets +10 point for each header and then the sum is divided by number of unique subdomains found in that URL. For example; let's say www.google.com.tr has three different headers; X-XSS-Protection, X-Frame-Options and HttpOnly cookies. 3 different header equals to +30 points for google.com.tr. If it has 4 different subdomains like; account.google.com.tr, cdn.google.com.tr, www.google.com.tr and test.google.com.tr than the final result will be 30 / 4 = 7.5 points.
Below are the results for each category sorted by points in a descending order.

Table 8. Points Based on Categories

Category	Websites Count	Subdomains Count	Total Pts.	Avg. Pts.
Forums	6	9	60,0	6,67
Finance	14	43	216,7	5,04
Real Estate	3	7	26,7	3,81
Games & Bets	18	49	183,5	3,74
Internet	69	242	897,6	3,71
Commerce	29	76	253,3	3,33
Retail	4	11	36,7	3,33
Logistics	4	11	31,7	2,88
Videos, Movies	37	89	245,6	2,76
Sources	13	47	128,2	2,73
Telecommunication	9	37	77,7	2,10
Aviation	2	10	15,6	1,56
Job Search	5	15	21,3	1,42
Government	22	141	165,6	1,17
Service	3	12	12,2	1,02
Press	108	346	274,5	0,79
Fun & Life	12	122	51,1	0,42
Others	3	3	0,0	0,00

Internet sector seems to carry the flag with 897.6 points however it's average is equal to 3.71 points

only. It means that Internet sector got so many points because it has 242 subdomains total.

Another great example to show that sheer numbers will not help you is that; 346 subdomains in comparison is in Press sector, however its security is inefficient. Having an average of 0.67 points only shows that most of the websites in Press sector has no security header ever. Since many of websites is in press sector in Alexa Turkey Top list, we can conclude that press sector puts end users into more risk than any other.

Forums got the leadership because they have little subdomains other than their homepages. This reduces the attack surface and in turn results in higher points.

Even though Finance could not make it to the top of the chart, it has a good average score despite of having many subdomains. It means that Finance sector is more secure compared to others.

Especially, Government, Telecommunication and Aviation sectors are worth mentioning because with all the investment power they have, we had expected them to be scored much better.

Additionally, most commonly used security headers in each sector are given in the following table in detail:

Table 9. Sector by Sector Headers Count

Category	Found Header	Count of Found Header
Internet	Set-Cookie : HTTPOnly	465
	No Security	404
	X-Frame-Options	272
	X-Content-Type-Options	249
	X-XSS-Protection : 1	161
	Set-Cookie : Secure	145
	Strict-Transport-Security	142
	Set-Cookie : None	136
	X-Download-Options	119
	Content-Security-Policy	22
	X-XSS-Protection : 0	9
Internet Total		**2.124**
Videos & Movies	Set-Cookie : HTTPOnly	1.885
	Set-Cookie : None	151
	No Security	51
	X-Frame-Options	5
	X-Content-Type-Options	4
	X-XSS-Protection : 1	1
	Set-Cookie : Secure	1
Videos & Movies Total		**2.098**
Press	No Security	1.229
	Set-Cookie : None	392
	Set-Cookie : HTTPOnly	320
	X-Frame-Options	20
	Set-Cookie : Secure	15
	X-Content-Type-Options	2
Press Total		**1.978**
Commerce	Set-Cookie : HTTPOnly	181
	Set-Cookie : Secure	91
	X-Frame-Options	85
	No Security	77
	Set-Cookie : None	20
	X-XSS-Protection : 1	8
	X-Content-Type-Options	6
Commerce Total		**468**
Government	Set-Cookie : HTTPOnly	115
	X-Frame-Options	44
	No Security	41
	Set-Cookie : None	30
	X-Content-Type-Options	28
	Set-Cookie : Secure	11
	X-XSS-Protection : 1	9
	Strict-Transport-Security	9
Government Total		**287**
Finance	Set-Cookie : None	70
	Set-Cookie : HTTPOnly	47
	Set-Cookie : Secure	32
	X-Frame-Options	25
	X-Content-Type-Options	6
	No Security	4
Finance Total		**184**
Fun & Life	Set-Cookie : None	99
	No Security	45
	Set-Cookie : HTTPOnly	13
	Set-Cookie : Secure	2
	X-XSS-Protection : 1	1
Fun & Life Total		**160**
Games	Set-Cookie : HTTPOnly	82
	X-Frame-Options	22
	No Security	20
	Set-Cookie : None	8

	X-XSS-Protection : 1	4
	Set-Cookie : Secure	4
	X-Content-Type-Options	3
	Content-Security-Policy	3
	Strict-Transport-Security	2
Games Total		**148**
Sources	Set-Cookie : HTTPOnly	48
	Strict-Transport-Security	26
	Set-Cookie : Secure	23
	No Security	13
	Set-Cookie : None	8
	X-XSS-Protection : 1	6
	X-Frame-Options	5
	X-Content-Type-Options	5
Sources Total		**134**
Telecommunications	Set-Cookie : HTTPOnly	35
	Set-Cookie : Secure	28
	No Security	20
	Set-Cookie : None	18
	X-Frame-Options	4
	Content-Security-Policy	1
	X-Download-Options	1
	X-Content-Type-Options	1
	X-XSS-Protection : 1	1
	Strict-Transport-Security	1
Telecommunications Total		**110**
Real Estate	X-XSS-Protection : 1	43
	Set-Cookie : None	41
	Set-Cookie : HTTPOnly	4
	No Security	1
Real Estate Total		**89**
Job Search	No Security	18
	Set-Cookie : HTTPOnly	9
	Set-Cookie : None	3
	X-Frame-Options	1
Job Search Total		**31**
Aviation	No Security	7
	Set-Cookie : HTTPOnly	4
	X-XSS-Protection : 1	3
	Set-Cookie : None	3
	X-Frame-Options	1
	X-Content-Type-Options	1
	Strict-Transport-Security	1
Aviation Total		**20**
Service	Set-Cookie : HTTPOnly	9
	No Security	3
	X-XSS-Protection : 1	3
	Set-Cookie : None	3
	Set-Cookie : Secure	2
Service Total		**20**
Logistics	Set-Cookie : HTTPOnly	7
	Strict-Transport-Security	3
	Set-Cookie : None	3
	No Security	2
	X-Frame-Options	1
Logistics Total		**16**
Retail	No Security	5
	Set-Cookie : HTTPOnly	4
	Set-Cookie : None	3
	Set-Cookie : Secure	1
	X-Frame-Options	1
	Strict-Transport-Security	1
Retail Total		**15**
Forum	Set-Cookie : HTTPOnly	8
	No Security	2
	X-Frame-Options	1
	X-XSS-Protection : 0	1
	X-Content-Type-Options	1
Forum Total		**13**
Others	No Security	3
Others Total		**3**

It can be concluded from the table that most common security control is HTTP Only cookies. Hopefully, in future other security headers will be utilized more as well.

5. Conclusion

Browsers integrate several security headers to improve web application security. Developers are expected to enable these security headers during design and coding. We developed a tool that can visit web pages and check existence of the security headers. By using this tool, we performed an analysis of 361 most popular web portals in Turkey including any subdomains linked on their home pages. The analysis methodology and the performed analysis with the detailed results are the main contributions of our research.

The analysis results have shown that developers are not aware of the security headers since they are not integrated at all. Comparing different sectors we can conclude that although Finance web portals utilize more security headers than others, overall security maturity in Turkish web landscape is still way lower than expected. Lack of controls mentioned in this article shows us only the tip of the iceberg. But looking at it, we gain insights on Turkish application security maturity level in general. We suggest enterprises, the government and all other web site owners here to invest in secure software development lifecycle programs to satisfy these and any further security requirement. For the interested researchers, we suggest OWASP SAMM (Security Assurance Maturity Model) which is an open framework to help organizations formulate and implement a strategy for software security that is tailored to the specific risks facing the organization.

Acknowledgements

A preliminary version of this work was presented at the ISCTurkey 2016 Conference [11]. Also, we would like to thank Özkan Boztaş for his precious support, reviewing the paper.

References

[1] "Website of European Parliament President Hacked", http://news.softpedia.com/news/Website-of-European-Parliament-President-Hacked-472575.shtml, Latest Access Time for the website is 29.09.2016

[2] OWASP Top 10 Project, https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, Latest Access Time for the website is 29.09.2016

[3] SANS Top 25 Security Errors, https://www.sans.org/top25-software-errors/, Latest Access Time for the website is 29.09.2016

[4] W3C, "Content Security Policy 1.0", https://www.w3.org/TR/2012/CR-CSP-20121115/, Latest Access Time for the website is 29.09.2016

[5] Mozilla Foundation, "CSP Policy Directives" "https://developer.mozilla.org/en-US/docs/Web/Security/CSP/CSP_policy_directives", Latest Access Time for the website is 29.09.2016

[6] OWASP, "OWASP Security headers project X-XSS-Protection header", https://www.owasp.org/index.php/OWASP_Secure_Headers_Project#X-XSS-Protection, Latest Access Time for the website is 29.09.2016

[7] OWASP Security headers project, X-Frame-Options header, https://www.owasp.org/index.php/OWASP_Secure_Headers_Project#X-Frame-Options, Latest Access Time for the website is 29.09.2016

[8] IETF, RFC 6797 HTTP Strict Transport Security (HSTS), https://tools.ietf.org/html/rfc6797, Latest Access Time for the website is 29.09.2016

[9] "OWASP Security Headers Project Public Key Pinning", https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning, Latest Access Time for the website is 29.09.2016

[10] Our implemented SecurityHeaderChecker tool, https://github.com/ttemrekisa/securityheaderchecker, Latest Access Time for the website is 29.09.2016

[11] K.E. Kısa, E.İ. Tatlı, "Analysis of HTTP Security Headers in Turkey", Procedings of 9th International Conference on Information Security and Cryptology (ISCTurkey 2016), pp.39-46, Ankara, Turkey, October 25-26, 2016.