Scientific Research Publishing

Journal of Information Security, 2023, 14, 1-15
https://www.scirp.org/journal/jis
ISSN Online: 2153-1242
ISSN Print: 2153-1234

Secure Web Application Technologies Implementation through Hardening Security Headers Using Automated Threat Modelling Techniques

Maduhu Mshangi Mlyatu[1*], Camilius Sanga[2]

[1]NECTA, Dar es Salaam, Tanzania
[2]Sokoine University of Agriculture, Morogoro, Tanzania
Email: *mshangimaduhu@yahoo.com

How to cite this paper: Mlyatu, M.M. and Sanga, C. (2023) Secure Web Application Technologies Implementation through Hardening Security Headers Using Automated Threat Modelling Techniques. *Journal of Information Security*, **14**, 1-15.
https://doi.org/10.4236/jis.2023.141001

Received: September 25, 2022
Accepted: November 27, 2022
Published: November 30, 2022

Abstract

This paper investigates whether security headers are enforced to mitigate cyber-attacks in web-based systems in cyberspace. The security headers examined include X-Content-Type-Options, X-Frame-Options, Strict-Transport-Security, Referrer-Policy, Content-Security-Policy, and Permissions-Policy. The study employed a controlled experiment using a security header analysis tool. The web-based applications (websites) were analyzed to determine whether security headers have been correctly implemented. The experiment was iterated for 100 universities in Africa which are ranked high. The purposive sampling technique was employed to understand the status quo of the security headers implementations. The results revealed that 70% of the web-based applications in Africa have not enforced security headers in web-based applications. The study proposes a secure system architecture design for addressing web-based applications' misconfiguration and insecure design. It presents security techniques for securing web-based applications through hardening security headers using automated threat modelling techniques. Furthermore, it recommends adopting the security headers in web-based applications using the proposed secure system architecture design.

Keywords

Secure Web Applications, Security Headers, Systems Security, Secure Web Architecture Design

1. Introduction

Security headers adoption and implementation have raised attention on how they

are fused in web-based systems security design. Its enforcement in combination with security web applications technologies' best practices [1] during system design has not been given attention. Security threats and cyber attacks are accelerated by insecure design and security misconfiguration [2] of web-based applications. Security controls are missing or ineffective security controls [3] [4] [5] are defined during the system development lifecycle. Design flaws (vulnerabilities) are introduced [6] by insecure design which cannot be fixed by correct configurations or perfect implementations of the system [7]. The analysis of business risks and threats is not taken into account during system design and its systems development life cycle in general. Business profiling is overlooked; business profiling is not carried out, and security concern is an afterthought during the system development life cycle [4]. Perfect implementation and security configuration are not given proper attention during the systems development life cycle. One of the security designs and implementations which is overlooked is the correct configurations of security headers [8] [9] in web-based applications.

Security precautions are not defined in security header response parameters in web-based systems. This results in security holes/weaknesses or threats to web-based systems. These vulnerabilities can be eliminated, minimized, or corrected by implementing security headers [5] in web-based systems. This involves configuring and setting correct security header responses in web-based systems infrastructure. It provides another security tier layer [3] by helping to minimize or eliminate intrusions and security vulnerabilities and threats in web-based systems. Whenever a browser sends requests for a page from a web server, the server responds with the content along with HTTP security response headers [10]. Among the security headers in web-based applications are X-Content-Type-Options, X-Frame-Options, Strict-Transport-Security, Referrer-Policy, Content-Security-Policy, and Permissions-Policy. To enhance the security of web-based applications, you should correctly configure these security headers. This is achieved through defining parameters or handling communication web-based sessions over HTTPS, or setting and defining web-based content which is loaded by JavaScript. Correct configurations in these security headers are always forgotten or incorrectly configured. This results in a misconfigured information system and a vulnerable system to cyber-attacks. Security misconfiguration creates security vulnerabilities, dangerous gaps, or mistakes that open the systems to cyberattacks. According to the OWASP top 10 [2], this type of misconfiguration is among the top risks in the list of critical web application security risks. The misconfigured systems are vulnerable if are missing appropriate security hardening in any part of the information systems infrastructure (such as web-server, database level, operating system, network, etc.) or improperly or incorrectly configured permissions or privileges of services.

1.1. Problem Statement

Web-based applications are developed without incorporating security headers

during system design; leaving the system vulnerable to attacks. Injection flaws (such as cross-site scripting, and SQL injections) are introduced during system design; cannot be fixed during systems implementations. The security controls that are not defined during system design leave the system vulnerable to cyber-attacks. The insecure design is contributed by ineffective or lack of business profiling using automatic threat modelling tools during the systems development lifecycle. This led to the failure to determine the optimal level of system security design required to withstand various cyber-attacks. The insecure design gap is widened by misconfigurations of the secure web applications technologies' best practices such as hardening security headers in information systems. It introduces security holes, dangerous gaps, or mistakes that leave the system open to cyber attacks.

1.2. Objective of Study

This study aims to investigate the security headers implementations landscape in web-based information systems in Africa's top 100 Universities/Colleges, and proposes a secure web-based system architecture design using threat modelling techniques.

2. Related Work

Security headers introduce another additional layer of security for preventing cyber-attacks such as cross-site scripting [6], click-jacking; session hi-jacking. Among these securities, headers are X-XSS security header, Content-Security-Policy, X-Frame-Options, and Strict Transport Security header (HSTS). Various research is still going on how to adopt security headers as an additional security layer against cyber attacks [10] [11]. The debate has been on how to configure them for maximum security. Previous studies have found that some security headers should be disabled (removed) [2]. Other researchers are arguing that disabling or removing some security headers and reverting to old configurations introduces risks. Thus, some browser developers do not have a plan for implementing them [11] [12]. One of the security headers that have raised debates is X-XSS protection [12].

X-XSS protection security header is responsible for protecting against cyber-attacks such as Cross-Site Scripting attacks [11] [13]. The optimal configuration for X-XSS is the protection header has changed from blocking (X-XSS auditor has been removed in modern browsers like Chrome, and Edge); other browsers like Firefox have not implemented X-XSS protection. There is a contradictory idea of whether to implement them or not; Chrome is reverting X-XSS protection to filter mode. Researchers are arguing that setting the browser to the default filter mode introduces cyber-attack risks [12]. X-XSS-Protection: 0; Condition 0 will disable the XSS filter. X-XSS-Protection: 1; Condition 1 enables the filter when an XSS attack is positive. X-XSS-Protection: 1; mode = block. Condition 1 is used with block mode; which blocks accessing the page with XSS mali-

cious codes [11]. The optimal alternative to X-XSS-Protection is to implement a Content-Security-Policy security header.

The Content Security Policy (CSP) header implements an additional layer to prevent web-based attacks [1]; among these attacks are Cross-Site Scripting (XSS) and code injection attacks [13]. This is achieved by employing a whitelisting method that informs the browser where to fetch the images, scripts, and, CSS. The value of CSP can be set with the following directives: default-src, script-src, media-src, and img-src. These directives, define the sources where the browser should load various types of resources [10]. [11] argued that for ensuring the security of web-based applications, CSP should be defined as default-src "none"; connect-src "self"; script-src "self"; img-src "self"; style-src "self"; frame-ancestors "self"; form-action "self". How full adoption of CSP is still questionable; web applications are vulnerable to cyber-attacks due to failure to implement CSP and other security headers (such as X-Frame-Options).

The X-Frame-Options HTTP response header is used to indicate if a browser is permitted to execute a page [8] in a "frame", "iframe" or "object" HTML tag. Sites and applications can use this to dodge clickjacking attacks, by ensuring their content cannot be embedded into other sites [10]. A clickjacking attack is an interactive interface-based user tricking attack. The user's minds set are manipulated to click on the hidden web application (links, action icons, or contents) using embedding techniques such as iframe. They are logically forced to transact in snare-based web applications rather than the required ones. Clickjacking attacks are an increasing threat to applications due to the failure to implement and correctly configure the X-Frame-Options security header. Clickjacking can lead to loss of confidentiality and disclosure of users "private sensitive data [1] [3] [14]. It can compromise users' e-mail, webcam cameras, and users' devices" audio speakers and; in turn, steal sensitive information [2] [14]. The X-Frame-Options should be set to deny or same-origin. This allows only the resources that are in the set of the same-origin policy to the frame of the original resources to be accessible by users of the web application [11]. The directive when set to the value of denying; rejects any resources hosted locally or in a remote location for creating snare environments for tricking the users [8] [10]. The directives for X-Frame-options for some parts of the application can be set to allow contents from a specific URL; this permits only the specified URL for framing the given page [2]. Thus, to ensure security against cyber-attacks X-Frame-Options and other security headers (such as the X-XSS-Protection header, and X-Content-Type-Options header) should be correctly configured.

The X-Content-Type-Options header provides an option for disabling MIME-type sniffing in web-based applications. MIME-type sniffing is the functionality of a web browser to inspect web contents to determine the file type format for uploaded files from users. This functionality can be misused to execute XSS attacks [13] on the given web application. This type of sniffing attack is speculating what content type the server response will be; instead of trusting what the header's content type value states. The browsers can be tricked to execute malicious code such

as XSS resulting in controlling the web-based applications depending on the access of the actual user of the web-based application. The MIME sniffing attacker facilitates an attack to gain access to the system and performs functions that can be performed by a real user of the given compromised system. To prevent, a content sniffing attack; the value of X-Content-Type-Options header should be configured to "nosniff" [15]. The browser will no longer "sniff" the content of the file received; instead it will use the value from the Content-Type header. This security header should be configured correctly in conjunction with other security headers such as the Strict Transport Security header.

A Strict Transport Security header (HSTS) forces browsers only to be accessed through HTTPS protocol instead of HTTP [15] [16]. If the web-based application allows connection through HTTP before redirecting to HTTPS it can result in man-in-the-middle attacks. This enables the attackers to intercept the communications between the user browser and the server. The requests for the user or responses from the server can be eavesdropped on by the attacker resulting in loss of confidentiality (viewing unencrypted traffics) or integrity (modifying the traffics). When the Strict-Transport-Security response header is configured; the web application is only forced to use the HTTPS protocol [16]. HSTS can be configured as Strict-Transport-Security: max-age = "expire-time"; max-age = "expire-time"; includeSubDomains; max-age = "expire-time"; preload. The "max-age" value is high enough to keep the website cached for the entire duration. For example for two years, it can be set as Strict-Transport-Security: max-age = 63,072,000; includeSubDomains; preload [17].

A study by [8] argued that the increase in the number of threats requires effective enforcement of security policies from the server to the client side. Previous studies [10] [13] [18] revealed that security breaches in web applications are the result of misconfiguration in web application infrastructure. Among these misconfigurations are incorrect configurations and usage of security headers in web application servers. One of the misconfigurations is in Content Security Policy (CSP) header. The misconfiguration in CSP can make web applications vulnerable to cross-sitting scripting. This can result in the loss of integrity and confidentiality of data and information systems. The previous studies by [10] found that CSP is used in less than 0.2% of the sites, and oftentimes incorrectly. They also investigated other relevant security-related headers. In particular, they found that X-XSS-Protection, X-Frame-Options, and Strict-Transport-Security headers were implemented, respectively, in about 4.4%, 4.1%, and 1% of the websites they analyzed. Despite the low adoption rate of HTTP security-related headers found by [8] [10], their results show a noticeable failure in integrating security headers during systems design. This results in the insecure design of web-based applications.

3. Research Methods and Materials

The study employed a controlled experiment using an automated security header analysis tool. The web-based applications (websites) were analyzed for wheth-

er security headers have been correctly implemented. The experiment was iterated for 100 universities in Africa [19] which are ranked high. The sample size consisted of 100 universities that have been ranked high.

3.1. Sampling Techniques

The study sample was selected using purposive sampling. The purposive sampling technique was employed to get a depth understanding [20] of the status quo of the security headers implementations. The sample frame comprised 100 universities in Africa universities ranked high including universities from Tanzania which are accredited by TCU. The purposive sample frame comprised two strata; one stratum comprised 70 top-ranked African universities and 30 from Tanzania accredited by TCU [21]. The two strata were combined to form a sample frame size of 100 ranked high universities in Africa.

3.2. Data Analysis Techniques

The data analysis was done using a controlled experiment that involved the analysis of web-based applications/websites for 100 top-ranked universities in Africa. The analyzed HTTP response headers and rated web-based systems were rated into F, E, D, C, B, A and A^+; where F is the lowest; and A^+ is the highest. The score rating of F means that an institution has not implemented security headers in the given system; E to B means that an institution has implemented some of the security headers in the given web-based application/website. Score A or A^+ means that institution has implemented most of the required security headers. A security rating score of D to F implies that a given website/web-based application is open to cyber-attacks such as cross-site scripting, SQL injections, and session hijacking.

3.3. Experiment for Evaluation Security of Web-Based Applications

3.3.1. Preparation of Experiment

The following materials were prepared for conducting the controlled experiment for:

1) List of web-based applications/websites for institutions in Africa;

2) Laptop with Intel Core i7 processor, 2.3 GHz, 16 GB RAM, 64-bit processor, Windows 10 Pro. with Internet connectivity;

3) Security headers analysis tool [22].

3.3.2. The Objective of the Experiment

The experiment aimed to evaluate the security of web-based applications/websites using a security header security analysis tool; a case study of higher learning institutions. The top 100 ranked universities in Africa were considered for this study.

3.3.3. Condition of the Experiment

The study analyzed security headers for a given website/web application to deter-

mine whether security headers have been implemented or not. The security headers responses for Strict-Transport-Security Referrer-Policy, X-Frame-Options, Content-Security-Policy, Permissions-Policy, and X-Content-Type-Options security headers were recorded.

3.3.4. Conducting the Experiment

The given website or web application was analyzed to determine the security level and readiness implementations of security headers. The experiment was executed in circular fashion iterations for 100 universities. The results are presented in table form for each security header employed in the experiment. The results are presented in section 4.

4. Results and Discussions

This section presents results findings for analysis of security in web-based applications for the top 100 universities, colleges, and higher learning institutions in Africa. The results and their discussions are presented for 6 security headers; namely: X-Content-Type-Options, X-Frame-Options, Strict-Transport-Security, Referrer-Policy, Content-Security-Policy, and Permissions-Policy as follows.

4.1. X-Content-Type-Options

The results for security header analysis of misconfigurations of security header: X-Content-Type-Options were carried out for 100 top-level universities in Africa. The results revealed that 70% of the web-based applications in Africa have not been correctly configured. The content-Type-Options security header is shown in Table 1. Its correct configuration helps to prevent sniffing of the MIME type for web-based applications through the browser. MIME stands for Multi-purpose Internet Mail Extensions. MIME types are means of determining file types in web-based applications used by web browsers while communicating with the server hosting the application.

MIME sniffing is used by some web to examine the content of a particular asset. This can cause security risks by allowing attackers to send an XSS (Cross-Site Scripting) attack. To stop this vulnerability set X-Content-Type-Options: nosniff in the web-server configuration file. This will force the browser to disable MIME sniffing and stop analyzing MIME type) and use the correct MIME type which is sent by the server hosting the application.

4.2. X-Frame-Options

The results revealed that 70% of the web-based applications in Africa have not

Table 1. Compliance for -content-type-options security header.

X-Content-Type-Options	%
No	70
Yes	30

correctly configured the X-Frame Options security header as shown in Table 2.

4.3. Strict-Transport-Security

The results revealed that 90% of the web-based applications in Africa (Tanzania inclusive) have not correctly configured the X-Frame Options security header as shown in Table 3. Thus, 10% of web-based applications have been configured with the Strict-Transport-Security option. The risks involve intercepting encrypted unencrypted traffic requests during the http redirect to HTTPS by prohibiting loading http requests. It forces all requests to HTTPS eliminating the chance of intercepting or redirecting the original web request to malicious web applications. When it is correctly set; it mitigates man-in-the-middle cyber-attacks (such as downgrade cyber-attacks; cookie hijacking attacks). The setting checked is Strict-Transport-Security: max_age = "expire time in seconds". This configuration is done on the web server; for example, Strict-Transport-Security "max-age = 93,084,000; includeSubdomains".

4.4. Referrer-Policy

The results revealed that 96% of the web-based applications in Africa (Tanzania inclusive) have not correctly configured the X-Frame Options security header as shown in Table 4. Thus, 4% of web-based applications have been configured with Referred Policy security header option.

4.5. Content-Security-Policy

The results revealed that 96% of the web-based applications in Africa (Tanzania inclusive) have not correctly configured the content security policy security header as shown in Table 5. Thus, 4% of web-based applications have been

Table 2. Compliance for X-frame options.

X-Frame-Options	%
No	70
Yes	30

Table 3. Compliance with strict-transport-security.

Strict-Transport-Security	%
No	90
Yes	10

Table 4. Compliance with referred policy.

A Referrer Policy	%
No	96
Yes	4

Table 5. Compliance with content security policy.

Content-Security-Policy	%
No	96
Yes	4

Table 6. Compliance with permission policy.

Permissions Policy	%
No	100
Yes	0

configured with a content security policy security header.

4.6. Permissions-Policy

The results revealed that 100% of the web-based applications in Africa (Tanzania inclusive) have not correctly configured the content security policy security header as shown in Table 6. Thus, this depicts the permissions policy that has not been adopted in Africa (Tanzania inclusive) as one security header for improving the security of data and information in web-based applications.

5. Proposed Secure Web-Based Architecture Design

The study proposes a secure web-based architecture design; it comprises of secure web browser (for the clients), a secure web server, and threat modelling automatic system integration as shown in Figure 1. The secure web-based architecture design uses a structured business process to identify security requirements. It analyzes vulnerabilities and threats through the business process to establish security requirements [23]. It uses a process flow diagram and process map to visualize users' systems interactions by portraying the web-based applications' use cases systems interactions such as how the user is identified and authenticated securely. It includes various technical controls that make up the use case: forms, cookies, sessions, or other coding elements/controls building up the given use case for the particular web-based applications and their communications boundaries. The web server is hardened by correctly configuring the security headers. Among these securities, headers are X-XSS protection, permission policy, referrer policy, x-content-type option, strict transport security, and x-frame options security header.

The descriptions of components of secure web-based architecture design are as follows:

1) Security threat modelling

This assists to highlight design faults that leave the systems vulnerable to cyber-attacks. Among these security threat modelling frameworks/models are STRIDE, PASTA, TRIKE, VAST, DRED, and OTAVE. The steps in security threats modelling for secure web-based design are: step 1-Diagram the application:

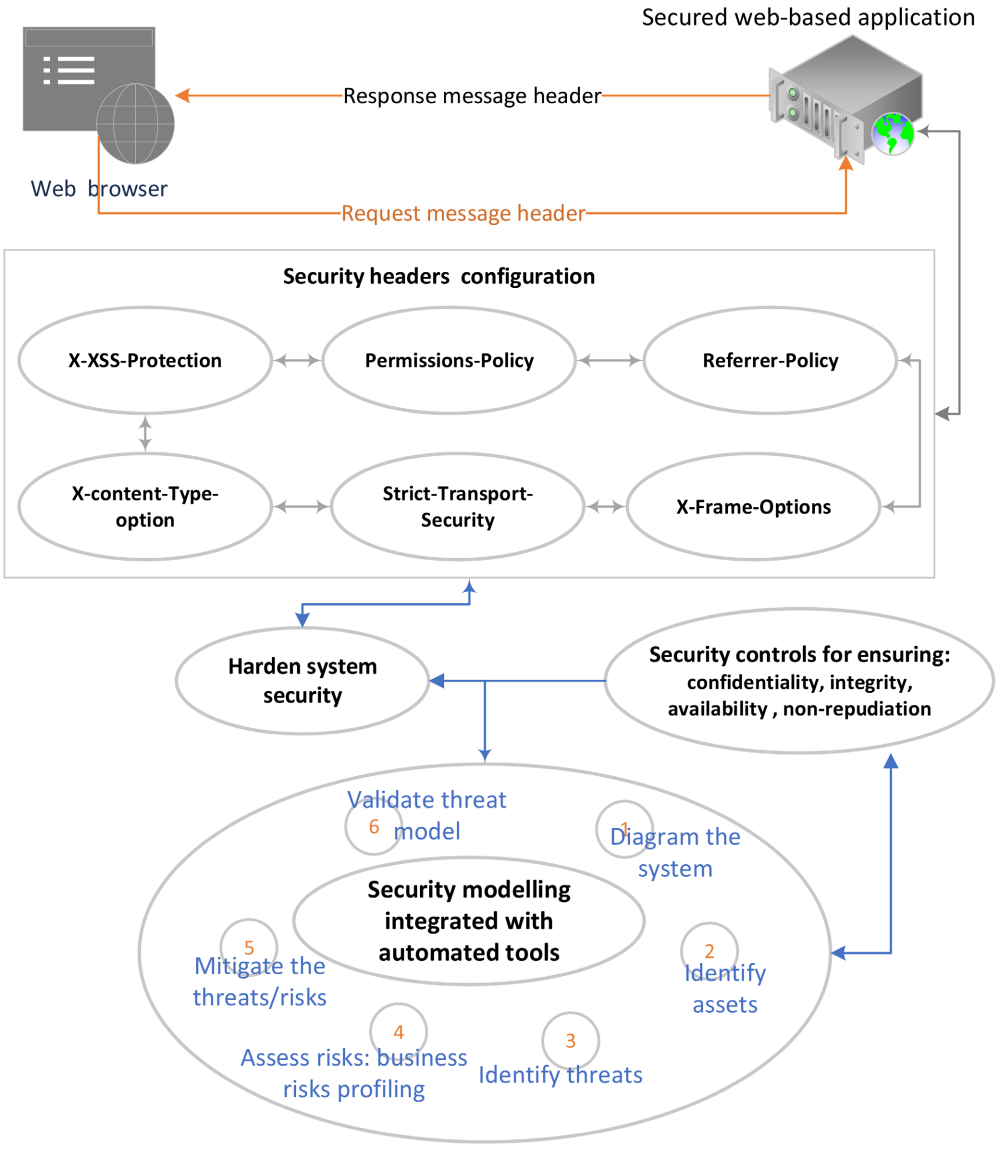

Figure 1. Secure web architecture design.

break down the system into components and visualize using process flows showing interactions, actors, and protocols involved. Step 2: identify the assets to be protected. Step 3: identify the various threats affecting each identified asset based on system components. Step 4: assess risks: business risks profiling; risks are defined based on the threats affecting assets for the information system. Step 5: mitigate the threats/risks: the risks are mitigated throughout the system development lifecycle based on business risk profiling; this results in a secure system design architecture. Step 6: validate the threat model: by using automated tools [24] [25]; the system is validated if the identified threats/risks have been mitigated to an acceptable optimal level. If not, steps one to five is repeated and iterated until a secure system design is achieved based on security requirements.

2) Security requirements and security controls

Define security controls [3] for ensuring confidentiality, integrity, availability,

and non-repudiation security requirements based on the threat modelling analysis. These security controls are subjected to threat modelling by mitigating the identified risks/threats by hardening the given web-based information system.

3) Harden system security

This involves hardening the information system based on the defined security controls through threat modelling. Security headers and other security controls are correctly configured according to security requirements in an iterative fashion until the secure system's optimal level is reached.

4) Security headers configuration

The configurations of security headers are based on security requirements for the given web-based application; its description is as follows.

a) X-content-Type-option

{X-Content-Type-Options: nosniff}

The X-Content-Type-Options should be set to "nosniff to mitigate MIME sniffing in the client web browser. This mitigates the exploitation of data in response to requests from the web server while clients access resources from the web server. Its exact syntax slightly varies from one operating system's distribution to another.

b) X-Frame-Options

The X-Frame-Options security header is used to indicate whether or not a web-based application accessed via the browser by the client should be allowed to render a web page in a "frame" or when "iframe" is employed. Web applications use this security header to protect against Clickjacking attacks [18]; it ensures that the contents are not embedded into other web pages. X-Frame-Options security header has three options values which can be set: DENY, SAMEORIGIN, or ALLOW-FROM URI. DENY: stops any web-based domain application from enclosing the content; this is recommended setting for mitigating cyber-attacks such as clickjacking attacks [18] [26]. SAMEORIGIN allows only the current web application to frame its content. ALLOW-FROM URL: permits the specified "URL" to frame the defined URL. Syntax: {X-Frame-Options: DENY; X-Frame-Options: SAMEORIGIN}

c) Strict-Transport-Security

HTTP Strict transport security header (HSTS) forces the web-based application to be accessible only through HTTPS connections [16]. HSTS addresses passive attackers, active attackers in the network, and imperfect web developers (such minimal requirements gathered; minimal system architecture design as users of the system do not know what they want initially until developing a prototype for them to see). The passive attacks addressed by HSTS include network traffic analysis (using tools such as Wireshark), eavesdropping attacks (exploiting Wi-Fi networks), footprint attacks (collecting information about the network, and systems using tools such as search engines, google hacking, social engineering techniques). HSTS attempts to address man-in-the-middle attacks through invalid or expired certificates. HSTS does prevent phishing and malware attacks which can be achieved through creating awareness among employees

about the danger; limiting sensitive information; disable directory listing.

Syntax:

Strict-Transport-Security: max-age=<expire-time>

Strict-Transport-Security: max-age=<expire-time>; includeSubDomains

Strict-Transport-Security: max-age=<expire-time>; preload

For example:

{Header set Strict-Transport-Security: max-age = 31,536,000; includeSubDomains; preload}

max-age = "*expire-time*": The time (in seconds) the browser should remember the web-based application is accessible through HTTPS only.

includeSubdomains: this is the option; when is set it implies that this rule applies to all the domain and their subdomains for a web-based application.

Preload (*option*): This directive enables the owner of the web-based application to directly preload their domains and their subdomains. It leads to privacy violations of cookie values; login cookies are leaked. Preloading has some usability limitations: removing subdomains added in error takes time as you should wait for removal to complete its propagation; changing the max-age directive is possible after the original specified maximum age time expires; some systems use only http; when all sub-domains are forced to HTTPS, it can cause usability challenges.

d) Referrer-Policy: {Keywords: *, none, self, hosts}

The Referrer-Policy security header controls the amount of referrer information which are sent through the Referrer-Policy security header with each request in web-based applications.

Referrer-Policy: *strict-origin-when-cross-origin*

e) X-XSS-Protection: {Header set X-XSS-Protection "1; mode = block"}

X-XSS protection security header is protecting web-based applications against cyber-attacks such as Cross-Site Scripting attacks. The optimal configuration for the X-XSS protection header has changed from blocking (X-XSS auditor has been removed in modern browsers like Chrome, and Edge); other browsers like Firefox have not implemented X-XSS protection.

Syntax: {

X-XSS-Protection: 0; X-XSS-Protection: 1; X-XSS-Protection: 1; mode=block;

X-XSS-Protection: 1; report=<reporting-uri>}

f) Permissions-Policy

The referrer (or "referrer") header is sent to a server when you visit a website and were previously on another website. The target site can use that header to see where you came from.

Syntax:

{Referrer-Policy: no-referrer

Referrer-Policy: no-referrer-when-downgrade

Referrer-Policy: origin

Referrer-Policy: origin-when-cross-origin

Referrer-Policy: same-origin

Referrer-Policy: strict-origin
Referrer-Policy: strict-origin-when-cross-origin
Referrer-Policy: unsafe-URL
}

no-referrer: The Referer header is excluded: sent requests do not include any referrer information.

no-referrer-when-downgrade: Send the origin, path, and query string in Referer when the protocol security level stays the same or improves (http→hhtp; http→https; https→https). Referer headers are not sent to the less secure endpoint for web-based requests (https→http; https→file).

The descriptions of these descriptive:

origin: Send only the origin in the Referer header. For example, a document at https://example.com/page.html will send the referrer to https://example.com/.

origin-when-cross-origin: When performing a same-origin request to the same protocol level (HTTP → HTTP, HTTPS → HTTPS), send the origin, path, and query string. Send only the origin for cross-origin requests and requests to less secure destinations (HTTPS → HTTP).

same-origin: Send the origin, path, and query string for same-origin requests. Don't send the Referer header for cross-origin requests.

strict-origin: Send only the origin when the protocol security level stays the same (HTTPS → HTTPS). Don't send the Referer header to less secure destinations (HTTPS → HTTP).

strict-origin-when-cross-origin (*default*): Send the origin, path, and query string when performing a same-origin request. For cross-origin requests send the origin (only) when the protocol security level stays the same (HTTPS → HTTPS). Don't send the Referer header to less secure destinations (HTTPS → HTTP).

For enhancing security, the recommended optimal configuration for Referer header-policy security header is set as *Referrer-Policy*: *strict-origin-when-cross-origin.* It tells the referrer header to not send referrer information to any user who visits another server.

6. Conclusion and Recommendations

The findings reveal that security headers in web-based systems are not taken into consideration during the systems development life cycle. This has been shown in the analysis survey of web-based applications in the top 100 websites of universities in Africa (Tanzania inclusive). The security headers included in this survey analysis using a web-based security header analysis tool are X-Content-Type-Options, X-Frame-Options, Strict-Transport-Security, Referrer-Policy, Content-Security-Policy, and Permissions-Policy. The results revealed that most of the web-based systems in cyberspace have not been correctly configured with the required security header configurations. The study found that 70% to 100% of the web-based applications in Africa have not been correctly configured with the required secu-

rity headers. This has accelerated cyber-attacks (such as cross-site scripting and SQL injection attacks) on information systems. For enhancing security in web-based systems, the study recommends the adoption and correct configuration of the security headers in web-based systems. Web-based protocols should be enhanced to automatically check the minimum enforcement of these security headers.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] SANS (2022) Securing Web Application Technologies [SWAT] Checklist. https://www.sans.org/cloud-security/securing-web-application-technologies

[2] OWASP (2021) OWASP Secure Headers Project. https://owasp.org/www-project-secure-headers

[3] Mshangi, M., Nfuka, E.N. and Sanga, C. (2017) An Innovative Soft Design Science Methodology for Improving Development of a Secure Information System in Tanzania Using Multi-Layered Approach. *Journal of Information Security*, **8**, 141-165. https://doi.org/10.4236/jis.2017.83010

[4] Mshangi, M., Sanga, C. and Ngemera Nfuka, E. (2016) Designing Secure Web and Mobile-Based Information System for Dissemination of Students' Examination Results: The Suitability of Soft Design Science Methodology. *International Journal of Computing and ICT Research*, **10**, 10-40. https://www.researchgate.net/publication/313469379

[5] CISA (2022) Weak Security Controls and Practices Routinely Exploited for Initial Access. https://www.cisa.gov/uscert/ncas/current-activity/2022/05/17/weak-security-controls-and-practices-routinely-exploited-initial

[6] Weamie, S.J.Y. (2022) Cross-Site Scripting Attacks and Defensive Techniques: A Comprehensive Survey. *International Journal of Communications, Network and System Sciences*, **15**, 126-148. https://doi.org/10.4236/ijcns.2022.158010

[7] NIST (2020) National Institute of Standards and Technology Special Publication 800-53, Revision 5: Security and Privacy Controls for Information Systems and Organizations. NIST Special Publication, NIST-800-5 (Revision 5), 1-465.

[8] Buchanan, W.J., Helme, S. and Woodward, A. (2018) Analysis of the Adoption of Security Headers in HTTP. *IET Information Security*, **12**, 118-126. https://doi.org/10.1049/iet-ifs.2016.0621

[9] Petkova, L. and Technologies, I. (2019) HTTP Security Headers. *Knowledge—International Journal*, **30**, 701-706. https://doi.org/10.35120/kij3003701p

[10] Lavrenovs, A. and Melón, F.J.R. (2018) HTTP Security Headers Analysis of Top One Million Websites. *International Conference on Cyber Conflict, CYCON*, Tallinn, 29 May-1 June 2018, 345-370. https://doi.org/10.23919/CYCON.2018.8405025

[11] Mozilla (2021) Content Security Policy (CSP). https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

[12] Braun, F. (2019) Chrome Switching the XSSAuditor to Filter Mode Re-Enables the Old Attack. https://frederik-braun.com/xssauditor-bad.html

[13] Dolnak, I. (2017) Content Security Policy (CSP) as Countermeasure to Cross-Site

Scripting (XSS) Attacks. *ICETA* 2017—15*th IEEE International Conference on Emerging eLearning Technologies and Applications*, *Proceedings*, Stary Smokovec, 26-27 October 2017, 1-4. https://doi.org/10.1109/ICETA.2017.8102476

[14] Wu, L., Brandt, B., Du, X. and Ji, B. (2017) Analysis of Clickjacking Attacks and an Effective Defense Scheme for Android Devices. 2016 *IEEE Conference on Communications and Network Security*, *CNS* 2016, Philadelphia, 17-19 October 2016, 55-63. https://doi.org/10.1109/CNS.2016.7860470

[15] MDN (2022) X-Content-Type-Options. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options

[16] Jackson, C. and Barth, A. (2012) HTTP Strict Transport Security (HSTS). Internet Engineering Task Force (IETF), 1-46. https://www.rfc-editor.org/rfc/pdfrfc/rfc6797.txt.pdf

[17] Albeniz, Z., Morgenroth, S. and Yildirimkaya, U. (2018) X-Frame-Options Content-Security-Policy (CSP) HTTP Strict Transport Security (HSTS) Public Key Pinning (PKP). The Most Commonly Used HTTP Security Headers and How They Work. Netsparker.

[18] Kalim, A., Jha, C.K., Tomar, D.S. and Sahu, D.R. (2021) Novel Detection Technique for Framejacking Vulnerabilities in Web Applications. 2021 2*nd International Conference on Computation*, *Automation and Knowledge Management* (*ICCAKM*), Dubai, 19-21 January 2021, 1-6. https://doi.org/10.1109/ICCAKM50778.2021.9357764

[19] WEBOMETRICS (2022) Ranking Web of Universities. https://www.webometrics.info/en/Africa

[20] Cohen, L., Manion, L. and Morrison, K. (2018) Research Methods in Education. 8th Edition, Routledge, Taylor & Francis Group, London.

[21] TCU (2022) University Institutions Operating in Tanzania. https://www.tcu.go.tz/sites/default/files/Bachelor-Degree-Admission-Guidebook-Direct-Entry_06.06.2022.pdf

[22] Securityheaders.com (2022) Security Headers. https://securityheaders.com

[23] Conklin, L. (2022) Threat Modeling Process. https://owasp.org/www-community/Threat_Modeling_Process

[24] Goodwin, M. (2020) OWASP Threat Dragon. https://owasp.org/www-project-threat-dragon

[25] MICROSOFT (2016) Microsoft Threat Modeling Tool. https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling

[26] Rao, K.S., Jain, N., Limaje, N., Gupta, A., Jain, M. and Menezes, B. (2016) Two for the Price of One: A Combined Browser Defense against XSS and Clickjacking. 2016 *International Conference on Computing*, *Networking and Communications*, *ICNC*, Kauai, 15-18 February 2016, 1-6. https://doi.org/10.1109/ICCNC.2016.7440629