ISSN: 2229-6948(ONLINE) ICTACT JOURNAL ON COMMUNICATION TECHNOLOGY, MARCH 2017, VOLUME: 08, ISSUE: 01
DOI: 10.21917/ijct.2017.0217

AN INSECURE WILD WEB: A LARGE-SCALE STUDY OF EFFECTIVENESS OF WEB SECURITY MECHANISMS

Kailas Patil
Research and Development, Vishwakarma Institute of Information Technology, India

Abstract

This research work presents a large-scale study of the problems in real-world web applications and widely-used mobile browsers. Through a large-scale experiment, we find inconsistencies in Secure Socket Layer (SSL) warnings among popular mobile web browsers (over a billion users download). The majority of popular mobile browsers on the Google Play Store either provide incomplete information in SSL warnings shown to users or failed to provide SSL warnings in the presence of security certificate errors, thus making it a difficult task even for a security savvy user to make an informed decision. In addition, we find that 28% of websites are using mixed content. Mixed content means a secure website (https) loads a sub resource using insecure HTTP protocol. The mixed content weakens the security of entire website and vulnerable to man-in-the-middle (MITM) attacks. Furthermore, we inspected the default behavior of mobile web browsers and report that majority of mobile web browsers allow execution of mixed content in web applications, which implies billions of mobile browser users are vulnerable to eavesdropping and MITM attacks. Based on our findings, we make recommendations for website developers, users and browser vendors.

Keywords:

Web Security, Mixed Content, SSL Warnings, HSTS, CSP, X-Frame-Options, X-XSS-Protection

1. INTRODUCTION

The same-origin policy [21], a fundamental security mechanism within web browsers, which isolates one origin's resources from other origins. SOP protects one site's properties and documents from being accessed by the other websites opened in the web browser. However, attackers can subvert the SOP by using various attack vectors [6][7][9][19]. Also the SOP fails to isolate different directory paths within the URL. For example, URLs "www.examplesite.com/~ram" and "www.examplesite. com/~shyam" runs in the same origin, even though they belong to different users directory.

This research work seeks to study the efficacy of web security mechanism on real-world websites. Web browsers have adopted various web security solutions to protect users against various attack vectors such as cross-site scripting [14], clickjacking [19], etc. Web servers enforce web browsers to apply web security mechanism by using specific HTTP headers. However, web developers' at large technology companies may not have direct access to change the HTTP headers on web servers, making it difficult to enforce security mechanisms. This lack of direct access to HTTP headers is hindering the adoption of web security mechanisms by real-world web applications as shown by our results in Section 4. In this review work, we answer the following research questions.

- *RQ*1: Do web developers understood how to use web security mechanisms supported by web browsers?
- *RQ*2: How many websites enforce them incorrectly and nullify the defense of security mechanism?
- *RQ*3: Are all security mechanisms adopted in real-world websites?

The aim of this paper is to evaluate the efficacy of web security mechanisms adoption on real-world websites, consisting of 10,00,000 websites from Alexa's top websites list. Based on our findings, we make recommendations for website developers, users, and browser vendors.

The goals of this paper are to three-fold: 1) Investigate modern mobile web browsers against effectiveness to protect users in practice, 2) Perform analysis of real-world websites whether they protect users against various attack vectors, and 3) Identify answer to research questions such as, Do web developers understood how to use web security mechanisms supported by web browsers? And, how many websites enforce them incorrectly and nullify the defense of security mechanism? In this paper, we give a systematic measurement of the problems in websites and mobile web browsers.

In particular, this paper makes the following contributions:

- We performed a large-scale study on Alexa top 10,00,000 websites to find web security techniques adaptation in practice.
- We report the behavior of popular mobile web browsers in the presence of mixed content and SSL errors.
- We provide recommendations to website developers, users, and browser vendors.

Our experiments show a lack of web security mechanism implementations in real-world websites and the necessity for research works to enumerate web security best practices and help to promote adaptation.

The rest of this paper is organized as follows: Section 2 summarizes related work. Section 3 presents background information about various security headers proposed and available for developers to strengthen security of their website. Section 4 presents our experimental evaluation and analysis. Section 5 describes mitigation techniques and we conclude the paper in section 6.

2. RELATED WORK

Yu-chi et al. [1] presented a survey on the security of certificate less signature schemes and show generalization of security models which consist of all cases of the adversaries. Majeed et al. [2] presented security challenges in Opportunistic networks. The authors propose the trust in Opportunistic networks is based on trust between people because OppNets strongly depend on human interaction. Neelam et al. [3] presented an

overview of major issues of information security in network environment. Researchers proposed various solutions to defend against script injection attacks, ranging from privilege separation [6], [8-10], [23-25], sanitization [12], filtering [11], confinement [13], to security policy enforcement mechanism [7].

As the browser environment evolves with increasing complexity, attackers have various ways in triggering malicious requests to the server. ClearRequest [25] validates web requests using dependencies in the browser environment. It extracts the dependency of web requests from the browser, representing it in a request dependency graph (RDG). RDG allows web servers to detect malicious requests through enforcing the request dependency integrity, which is applicable to a wide range of malicious-request-based attacks.

To control behaviors of content scripts injected by extensions in web pages, SessionGuard [28] isolates content scripts in an isolated environment, called the shadow DOM. With the shadow DOM, SessionGuard provides content scripts an encrypted view of web application data, and controls their access to the original DOM.

By exploiting layout and JavaScript features of a web page, attackers can create web page objects that hijack users' clicks. ClickGuard [24] is a solution to mitigate the problem of click event hijacking by inferring users' intentions. It ensures that the browser's behavior after a click matches the user's original intention.

Ronak et al. [23] evaluated effectiveness of browser security warnings. The authors reported inconsistency in SSL warnings among web browsers. This paper evaluated real-world websites on various mobile browsers and identified that in most of the browsers, security warnings are not emphasized, and browsers simply do not show warnings, or there are a number of ways to hide those warnings of malicious sites.

Content Security Policy (CSP) is a browser security mechanism that aims to protect websites from content injection attacks. To adopt CSP, website developers need to manually compile a list of allowed content sources. Patil et al. [4] performed measurements on a large corpus of web applications to provide a key insight on the amount of efforts web developers required to adapt to CSP. The authors also identified errors in CSP policies that are set by website developers on their websites. To address these issues and make adoption of CSP easier and error free, the authors proposed UserCSP [5] a tool as a Firefox extension. The UserCSP uses dynamic analysis to automatically infer CSP policies, facilitates testing, and gives savvy users the authority to enforce client-side policies on websites.

3. BACKGROUND

3.1 SSL CERTIFICATE WARNINGS

When a user visits a website using HTTPS protocol, the web browser uses SSL/TLS protocol to setup a secure connection between the web browser and the website's server. SSL/TLS protocol is used to ensure privacy and authenticity of web servers. The privacy means the communication between user's browser and website's server remains secret, and network attackers should not modify or read the information in transit. The authenticity of website's server means web browsers should be able to identify server identity and ensure that the web server is not lying about its identity. When web browsers are not able to ensure authenticity or secrecy of the network connection between browsers and website's server, they alert users with SSL security warnings. An ideal warning can empower users to make informed decision whereas ineffective SSL warnings can mislead users to fall victim of MITM or eavesdropping attacks. Section 4 reports the result of inconsistency to show SSL warnings by mobile browsers.

3.2 MIXED CONTENT

HTTPS protocol is designed to protect users from man-in-the-middle (MITM) and eavesdropping attacks by adding SSL/TLS security capabilities to HTTP protocol. The SSL/TLS security capabilities allow bidirectional encryption of communication channel between browsers and web servers, and enables authentication of a web server to browsers. In mixed content, HTTPS websites include some non-secure HTTP sub-resources. Thus, mixed content inclusion in HTTP websites provides network attackers an opportunity to exploit HTTPS protected websites.

3.3 HTTP STRICT - TRANSPORT-SECURITY (HSTS)

It is a web security HTTP header based mechanism designed to protect web applications against cookie hijacking and mixed content. HTTP connection is susceptible to various attack vectors. The presence of this header in the web server response ensures that the connection to a web server from a web client cannot be established through an insecure HTTP channel. It enforces web browsers to interact with web servers only using secure HTTPS connections. HSTS is an Internet Engineering Task Force (IETF) standards protocol specified in RFC 6797 [17].

3.4 CONTENT SECURITY POLICY (CSP)

CSP is a web browser security mechanism proposed by Mozilla. The aim of the CSP mechanism is to protect websites from content injection attacks. The web browser security model is rooted in the same-origin policy (SOP), which isolates one origin's resources from other origins. However, attackers can subvert the SOP by injecting malicious content into a vulnerable website through attacks such as Cross-Site Scripting (XSS) [14]. According to the OWASP vulnerability assessment in 2013, XSS attacks are among top five vulnerabilities [15]. The root cause of code injection problem on websites is that browsers are unable to distinguish between legitimate and maliciously injected content in a web application. To mitigate threats of XSS attacks, Mozilla proposed Content Security Policy (CSP) [16] a defense-in-depth.

CSP aims to solve this problem by providing a declarative content restriction policy in an HTTP header that the browser can enforce. CSP defines directives associated with various types of content that allow developers to create whitelists of content sources and instruct client browsers to only load, execute, or render content from those trusted sources. However, writing an effective and comprehensive CSP policy for websites is laborious. A policy can break website functionality if a legitimate content is overlooked during policy generation. A few websites adapted to CSP as shown by our results in Section 3 indicates that web developers prefer rich user interface instead of security.

3.5 X-FRAME OPTIONS

The header field "X-Frame-Options" aims to protect websites against clickjacking [19] attacks. In the clickjacking attack, an attacker uses opaque or transparent layers on top of webpage to trick a user into clicking on a button or link on another page from server B whereas users intentions to click on the page from server A visible to them. Thus, using hidden overlays on top of web age from server A, the attacker is able to "hijack" clicks meant for page A and sending them to server B. This is an attack on both the user and on server B. The X-Frame-Options is an IETF standards protocol specified in RFC-7034 [18].

3.6 X-XSS-PROTECTION

The x-xss-protection header aims to protect a website against reflected cross-site scripting (XSS) attacks. The presence of this header in a web response from a web server instructs web browsers to enforce the reflected cross-site scripting filter inbuilt in the web browsers.

3.7 SUBORIGIN RESOURCE INTEGRITY (SRI)

SRI is a W3C recommendation supported by most widely-used web browsers such as Google Chrome and Mozilla Firefox. SRI is based on an idea of signature based whitelisting proposed in research solutions such as BEEP [7] and Noncespaces [6]. It allows web publishers to specify a cryptographic hash of the JavaScript code text as a signature. Web publishers calculate a hash of embedded JavaScript code in web pages and include it as the integrity attribute in a *<script>* tag. At the client-side, web browsers calculate a hash of the JavaScript code in the script tag and compare it with the hash send by the web publisher. If it matches then browsers allow that JavaScript to execute otherwise browsers prevents it from execution.

4. EXPERIMENTAL EVALUATION AND ANALYSIS

We conducted empirical measurements to obtain the data for evaluating web security usage in wild. Our measurements are mainly carried out on a HP desktop computer running Ubuntu 12.04 64bit, with 4-core 2.67GHz CPUs and 8GB RAM. This section describes the testbed setup and results of our large-scale measurement experiment to answer to three research questions namely *RQ*1, *RQ*2, and *RQ*3.

4.1 QUANTITATIVE ANALYSIS GOALS

Our large-scale analysis experiments aim to measure the following:

Goal_1. ***Investigate the SSL warnings inconsistency in mobile web browsers.***

Through experiments, we find inconsistency in SSL errors reporting to users by mobile web browsers.

Goal_2. ***Study the behavior of popular mobile browser in the presence of mixed content.***

Using a large-scale measurement experiment of Alexa top 10,00,000 websites, we identify the behavior of popular mobile browsers on websites containing the mixed content.

Goal_3. ***Measure security headers adaptation by websites to protect users.***

We performed analysis of Alexa 10,00,000 websites to measure security headers used by real-world websites in practice to protect users from various attack vectors.

4.2 QUANTITATIVE ANALYSIS ON ALEXA TOP 1,00,000 WEBSITES AND 25 MOBILE WEB BROWSERS

In our experiments, we used Scrapy framework to crawl Alexa's top 10,00,000 websites for a 3-month period, starting 8th May 2016. Scrapy [22] is an open source framework that allows extraction of data from websites using python in a fast, simple and extensible way. We used python to write a spider to crawl a site and extract data. We scanned in total 1,00,000 Alexa top desktop websites and 25 mobile web browsers and checked response for security headers such as CSP headers such as X-ContentSecurity-Policy, X-WebKit-CSP, Content-SecurityPolicy, X-Frame-Options, HSTS, and X-XSS-Proteciton. Web applications can detect user agents and send appropriate CSP header in the response; therefore, we scanned all websites multiple times by using separate user agent strings [29]. The user agent strings we used are listed in the Table.1.

Table.1. A list of user agent strings

Browser	Version	User Agent String
Firefox	40.1	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:40.0) Gecko/20100101 Firefox/40.1
Google Chrome	41.0.2228.0	Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36
Internet Explorer	11	Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; AS; rv:11.0) like Gecko

Next, we explain how we measure metrics for above mentioned three goals and present their results.

Goal_1. ***Investigate the SSL warnings inconsistency in mobile web browsers.***

To investigate the SSL warnings inconsistency in mobile web browsers, we used 25 popular mobile web browsers based on number of downloads of the mobile web browsers and market share. To establish a secure HTTPS connection between servers and browser the SSL/TLS protocol is used. In the presence of SSL/TLS certificate errors such as a wrong domain name, self-signed certificates or certificates signed by untrusted CAs, web browsers display an SSL warning to users. This is the last line of defense used to prevent MITM attack on HTTPS connections. However, according to our quantitate analysis; mobile browsers behave differently in the presence of SSL certificate errors. The Table.2

summarizes inconsistency of mobile browsers in handling SSL certificate errors. We captured SSL warnings and address bar warnings on most popular 25 Android mobile browsers.

Table.2. Summary of SSL Warnings Inconsistency in Mobile Browsers (✓: Yes, ×: No)

Mobile Browser Name (version number)	Maximum Number of Installs on Google Play	W3C Recommendations	SSL Warnings	URL bar Warnings
Aesir (9.1)	500,000	×	✓	✓
Apus (1.5)	10,000,000	×	✓	✓
Best (1.5)	500,000	×	✓	×
Boat (8.7.4)	10,000,000	×	✓	✓
Browser (1.1.7)	5,000,000	×	×	×
Chrome (52.0)	5,000,000,000	✓	✓	✓
CM (5.20)	50,000,000	×	×	×
Free Cool (0.1)	100,000	×	×	×
Dolphin (11.5.8)	100,000,000	×	✓	✓
Dolphin Zero (1.3)	1,000,000	×	✓	✓
DU (6.4)	50,000,000	×	✓	✓
Firefox (4.0.3)	500,000,000	✓	✓	✓
FlashFox (44.0)	5,000,000	×	✓	×
Javelin Incognito (2.0.6)	100,000	×	✓	✓
Maxthon (4.5.9)	50,000,000	×	✓	×
Mini (2.33)	50,000	×	✓	×
NC (3.1.0)	5,000	×	✓	×
Next (2.15)	10,000,000	×	✓	×
Opera Mini (18.0)	500,000,000	✓	✓	✓
Opera (37.0)	500,000,000	✓	✓	✓
Puffin (4.8)	50,000,000	×	✓	×
UC (10.10)	500,000,000	✓	✓	✓
Web Explorer (10.1)	5,000,000	×	✓	×
Web Browser (2.0)	500,000	×	×	×
Yolo (4.1)	500,000	×	✓	×

***Lack of Meet W3C Recommendations*:** Total 20 out of 25 Android mobile browsers do not meet the guidelines provided by the W3C consortium. These browsers are namely, Aesir, Apus, Best, Boat, Browser, CM, Free Cool, Dolphin, Dolphin Zero, DU, Javelin, Maxthone, Mini browser, NC, Next, Puffin, Web browser, and Yolo. These browsers in total have more than a billion users on Google Play.

***No SSL Warnings*:** CM browser version 5.0, Free Cool browser version 0.1, Browser (1.1.7) and Web browser version 2.0 used in our experiments do not perform validity checks for security certificates presented in HTTPS connections. In addition, CM web browser does not show SSL warnings to users when a fraudulent certificate is presented for HTTPS connection. Nevertheless, the CM web browser always displays the "*Green Shield*" in browser's URL (Uniform Resource Locator) bar for all secure HTTPS connections, despite the presence of certificate errors.

***No SSL Warnings in the URL bar*:** As Table.1 shows, total 13 mobile browsers failed to provide SSL certificate error warnings in the browser's URL bar. The goal of SSL security certificate warnings is to prompt users that the server's certificate is not trusted, and secure connection cannot be guaranteed with the servers. Six mobile browsers (i.e. Boat, Dolhin, Javelin Incognito, Maxthon, Next, and Web Explorer) show pop-up of SSL warnings to alert the users.

Goal_2. ***Study the behavior of popular mobile browser in the presence of mixed content.***

We study Alexa's top 10,00,000 websites and find that 32,216 websites supporting HTTPS protocol. From the analysis of 32,216 HTTPS websites, we find that 9,021 (28%) websites have mixed content in form of subresource (CSS, Flash, iframe, image, JavaScript) inclusion using HTTP protocol. We observed that Google Chrome, Firefox, UC, Opera mobile web browsers have a mixed content blocker whereas other mobile browsers lack to provide mixed content blockers. However, Chrome browser shows an SSL certificate warning when attempting to view HTTP subresource over HTTPS connection and allow users to load blocked content by overriding the default behavior of browser as Fig.1 shows.

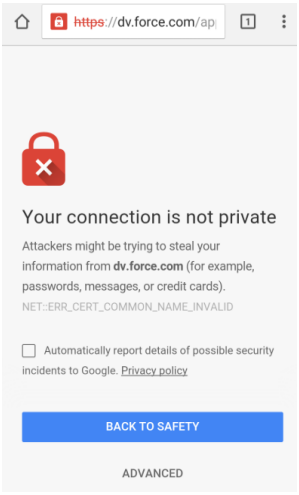

Fig.1. Certificate Warning in Google Chrome for Mixed content

Goal_3. ***Measure security headers adaptation by websites to protect users.***

Many existing defense solutions and best practices in web security are adopted in real-world systems such as Strict-Transport-Security (HSTS), Content Security Policy (CSP), X-Frame-Options, and X-XSS-Protection. The Fig.2 presents real-world adoption of various security headers.

***Strict-Transport-Security (HSTS)*:** The strict transport security headers were used on 11,308 websites our of Alexa's 10,00,000 websites. Most importantly, out of the 1 million websites we analyzed, 32,216 of them are using HTTPs and redirecting users to HTTPS version of the website. This implies 20,908 HTTPS websites lack to enforce Strict-Transport-Security header. Thus, these websites are vulnerable to protocol downgrade attacks and cookie hijacking.

***Content Security Policy (CSP)*:** Our empirical study results show that 1,456 websites are using CSP policy to protect their users against cross-site scripting attacks. In addition, 201 websites are using report only mode of Content-Security-Policy header. We also measured the inconsistency in enforcing CSP policies in real-world websites. In our experiment we observed,

- One website *http://start.funmoods.com/* used all three headers X-WebKit-CSP, X-Content-Security-Policy, and Content-Security-Policy.
- Three websites are used both X-Content-Security-Policy and Content-Security-Policy headers, namely:
  - http://mega.co.nz/,
  - https://github.com/EllisLab/CodeIgniter/wiki, and
  - http://lastpass.com/
- Four websites are serving both X-Content-Security-Policy and X-WebKit-CSP, namely:
  - http://blog.twitter.com/,
  - business.twitter.com,
  - demo.phpmyadmin.net,
  - http://papa.me/

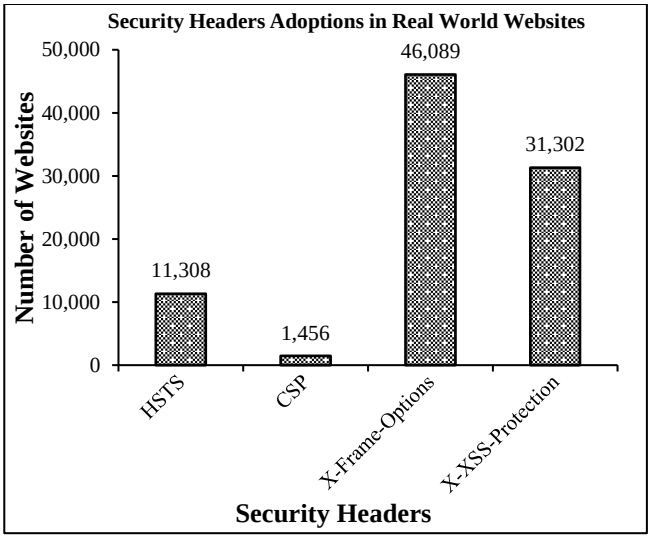

Fig.2. Web Security headers usage on real-world websites

The empirical study results show that 78 websites are using CSP policy to protect their home pages rather than enforcing it on all internal web pages. We observed websites are setting CSP policy incorrectly and thus keeping open doors for content injections. For example, http://www.metro-partner.ru/ website sets following CSP policy:

X-Content-Security-Policy: allow 'self '; script-src *; img-src *; options eval-script inline-script;

The http://www.metro-partner.ru/ website has defined CSP policy in an incorrect way and thus made it non-effective to protect users from content injection attacks. It allows scripts to be executed from any domain and images to be loaded from any arbitrary domains. Furthermore, it also allows execution of inline scripts and eval() usage.

The above results reveal that web developers have limited knowledge about CSP mechanism and its correct usage. As a result of errors made by web developers in setting CSP policies nullifies the protection provided by CSP and provides attackers an opportunity to exploit content injection vulnerabilities in websites.

***X-Frame-Options*:** Total 46,089 websites adapted to X-Frame-Option on their home pages to protect against clickjacking attacks. That is, only 4.6% of websites out of the Alexa 1 million websites are proactively protecting their users from clickajking attacks whereas remaining websites have not yet adapted to X-Frame-Options header.

***X-XSS-Protection*:** Total 31,302 websites adopt this header out of Alexa 1 million website domains. This result shows that majority of websites lack to adopt the protection inbuilt in web browsers to defend against reflected cross-site scripting injection attacks.

Suborigin Resource Integrity (SRI): Our results show that out of 10,00,000 Alexa top websites only 69 unique websites have adopted SRI mechanism to protect code injection attacks on their websites.

4.3 SUMMARY

The analysis shows that website developers have not yet completely adapted to all security solutions available to protect their users from various attack vectors. In addition, web developers' at large technology companies may not have direct access to change the website header on web servers, making it difficult to enforce web security mechanisms. Furthermore, web developers are not completely aware of all security solutions and how to use them correctly [4]. This leads to security errors from web developers such as enforcing CSP only on home pages whereas leaving internal webpage unprotected, incorrectly using security headers, etc.

This infers that web developers are unwilling to sacrifice functionality over security and limited knowledge of inbuilt defense mechanisms in modern web browsers among web developers resulted in the incorrect enforcement of web security mechanism.

5. MITIGATION TECHNIQUES

Since developers control CSP adoption, users lack control over their own security. Users do not have a mechanism to apply

Content Security Policies on the websites that they visit and cannot protect themselves from Cross-Site Scripting and Clickjacking attacks. We recommend the security savvy users to install UserCSP [5] extension available for Mozilla Firefox to protect themselves with custom policies that they can create and modify. In addition, UserCSP can help web developers to break down the challenges involved in adopting Content Security Policy with UserCSP feature to automatically infer policies.

To provide comprehensive security to its user, a website using HTTPS protocol can use Strict-Transport-Protocol (HSTS) and Content Security Policy headers. By enforcing the HSTS header, a website is protected from SSL-stripping attack [20] and CSP header protects the website from XSS [14][26][27] and Clickjacking [19] attacks. In addition, when untrusted content is embedded in web applications then they should be executed in an <iframe>. Because <iframe> provides separate JavaScript execution context and DOM elements, thus, isolates untrusted content from website's content.

Web browser vendors should provide effective SSL warnings to its users. An effective SSL warning means a warning shown to users should be comprehensive, that is, a user should be able to understand the source of data threat and the data at the risk.

6. CONCLUSION

In this paper, we did a large-scale study of SSL Warnings, security headers adoption in real-world and inferred difficulties in the web security mechanisms adoption. Web security mechanisms are not adopted in wild because limited knowledge of web security mechanisms available in web browsers among web developers. Using the extensive evaluation of real-world websites and mobile browser, this paper makes users aware about lack of proper security mechanisms adaptation by web developers and browser vendors. Our analysis reveals that web developers have limited knowledge about web security mechanisms supported by modern browsers and correct usage of them. This lack of security awareness in web developers put web users on high security risk. Since adoption is controlled by developers, users lack control over their own security. Users do not have a mechanism to apply security mechanisms to the websites that they visit and cannot protect themselves from various attack vectors. Tools such as UserCSP helps users to apply their only security policy to protect themselves against injection attacks using custom policies that they can create and modify.

ACKNOWLEDGMENTS

This work is in part supported by an AWS Education Research Grant Award, USA. The author would like to acknowledge the valuable comments and suggestions of the reviewers, which have improved the quality of this paper.

REFERENCES

[1] Yu-Chi Chen and Raylin Tso, "A Survey on Security of Certificateless Signature Schemes", *IETE Technical Review*, Vol. 33, No. 2, pp. 115-121, 2016.

[2] Majeed Alajeely, Robin Doss and Asmaa Ahmad, "Security and Trust in Opportunistic Networks-A Survey", *IETE Technical Review*, Vol. 33, No. 3, pp. 256-268, 2016.

[3] Neelam Bhalla, "Information Security: A Technical Review", *IETE Technical Review*, Vol. 19, No. 2, pp. 47-59, 2002.

[4] Kailas Patil and Braun Frederik, "A Measurement Study of the Content Security Policy on Real-World Applications", *International Journal of Network Security*, Vol. 18, No. 2, pp. 383-392, 2016.

[5] Kailas Patil, T. Vyas, F. Braun, M. Goodwin, and Z. Liang, "Poster: User CSP-User Specified Content Security Policies", *Proceedings of Symposium on Usable Privacy and Security*, pp. 1-2, 2013.

[6] Matthew Van Gundy and Hao Chen, "Noncespaces: using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks", *Proceedings of 16th Network and Distributed System Security Symposium*, pp. 1-13, 2009.

[7] T. Jim, N. Swamy, and M. Hicks, "Defeating Script Injection Attacks with Browser-Enforced Embedded Policies", *Proceedings of 16th International Conference on World Wide Web*, pp. 601-610, 2007.

[8] D. Akhawe, P. Saxena and D. Song, "Privilege Separation in Html5 Applications", *Proceedings of 21st Conference on Security Symposium*, pp. 23, 2012.

[9] E. Budianto, Y. Jia, X. Dong, P. Saxena, and Z. Liang, "You can't be me: Enabling trusted paths and user sub-origins in web browsers", *Proceedings of International Workshop on Recent Advances in Intrusion Detection*, pp. 150-171, 2014.

[10] Kailas Patil, Xinshu Dong, Xiaolei Li, Zhenkai Liang and Xuxian Jiang, "Towards Fine-Grained Access Control in JavaScript Contexts", *Proceedings of 31st International Conference on Distributed Computing Systems*, pp. 720-729, 2011.

[11] X. Dong, K. Patil, J. Mao, and Z. Liang, "A Comprehensive Client-Side Behavior Model for Diagnosing Attacks in Ajax Applications", *Proceedings of 18th* International *Conference in Engineering of Complex Computer Systems*, pp. 177-187, 2013.

[12] Joel Weinberger, Prateek Saxena, Devdatta Akhawe, Matthew Finifter, Richard Shin and Dawn Song, "A Systematic Analysis of XSS Sanitization in Web Application Frameworks", *Proceedings of European Symposium on Research in Computer Security*, pp. 150-171, 2011.

[13] X. Dong, Z. Chen, H. Siadati, S. Tople, P. Saxena, and Z. Liang, "Protecting Sensitive web Content from Client-Side Vulnerabilities with Cryptons", *Proceedings on ACM conference on Computer and Communications Security*, pp. 1311-1324, 2013.

[14] Amit Klein, "Cross Site Scripting Explained. Sanctum Security Group", Available at: https://crypto.stanford.edu/cs155/papers/CSS.pdf.

[15] Web Application Security Assessment Report, Available at: http://www.cstl.com/CST/Penetration-Test/CST-Web-Application-Testing-Report.pdf.

[16] S. Stamm, B. Sterne and G. Markham, "Reining in the Web with Content Security Policy", *Proceedings of 19th*

International Conference on World Wide Web, pp. 921-930, 2010.

[17] HTTP Strict Transport Security (HSTS), Available at: https://tools.ietf.org/html/rfc6797.

[19] Lin-Shung Huang, Alex Moshchuk, Helen J. Wang, Stuart Schechter and Collin Jackson, "Clickjacking: Attacks and Defenses", *Proceedings of 21st USENIX Security Symposium*, pp. 413-428, 2012.

[20] M. Marlinspike, "New Tricks for Defeating SSL in Practice", Available at: https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf

[21] Hossein Saiedian and Dan S. Broyles, "Security Vulnerabilities in the Same-Origin Policy: Implications and Alternatives", *Computer*, Vol. 44, No. 9, pp. 29-26, 2011.

[22] Scrapy Framework, Available at: https://scrapy.org/, Accessed on 2015.

[23] Ronak Shah and Kailas Patil. "Evaluating Effectiveness of Mobile Browser Security Warnings", *ICTACT Journal of Communication Technology*, Vol. 7, No. 3, pp. 1373-1378, 2016.

[24] Kailas Patil, "Preventing Click Event Hijacking by User Intention Inference", *ICTACT Journal of Communication Technology*, Vol. 7, No. 4, pp. 1408-1416, 2016.

[18] HTTP Header Field X-Frame-Options, Available at: https://tools.ietf.org/html/rfc7034.

[25] Kailas Patil, "Request Dependency Integrity: Validating Web Requests using Dependencies in the Browser Environment", *International Journal of Information Privacy, Security and Integrity*, Vol. 2, No. 4, pp. 281-306, 2016.

[26] Dnyaneshwar K Patil and Kailas Patil, "Automated Client Side Sanitizer for Code Injection Attacks", *International Journal of Information Technology and Computer Science*, Vol. 8, No. 4, pp. 86-95, 2016.

[27] Dnyaneshwar K. Patil and Kailas Patil, "Client-Side Automated Sanitizer for Cross-Site Scripting Vulnerabilities", *International Journal of Computer Applications*, Vol. 121, No. 20, pp. 1-7, 2015.

[28] Kailas Patil, "Isolating Malicious Content Scripts of Browser Extensions", *International Journal of Information Privacy, Security and Integrity*, 2017.

[29] User Agent String Explained, Available at: http://www.useragentstring.com/, Accessed on 2013.