Systematically Breaking Online WYSIWYG Editors

Ashar Javed(✉) and Jörg Schwenk

Chair for Network and Data Security Horst Görtz Institute for IT-Security,
Ruhr-University Bochum, Bochum, Germany
{ashar.javed,joerg.schwenk}@rub.de

Abstract. Cross-Site Scripting (XSS) — around fourteen years old vulnerability is still on the rise and a continuous threat to the web applications. Only last year, 150505 defacements (*this is a least, an XSS can do*) have been reported and archived in Zone-H (a cybercrime archive) (http://www.zone-h.org/). The online **WYSIWYG** (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et) or rich-text editors are now a days an essential component of the web applications. They allow users of web applications to edit and enter HTML rich text (i.e., *formatted text*, *images*, *links* and *videos* etc.) inside the web browser window. The web applications use **WYSIWYG** editors as a part of comment functionality, private messaging among users of applications, blogs, notes, forums post, spellcheck as-you-type, ticketing feature, and other online services. The XSS in **WYSIWYG** editors is considered more dangerous and exploitable because the user-supplied rich-text contents (may be dangerous) are viewable by other users of web applications.

In this paper, we present a security analysis of twenty (20) popular **WYSIWYG** editors powering thousands of web sites. The analysis includes **WYSIWYG** editors like Enterprise TinyMCE, EditLive, Lithium, Jive, TinyMCE, PHP HTML Editor, markItUp! universal markup jQuery editor, FreeTextBox (popular ASP.NET editor), Froala Editor, elRTE, and CKEditor. At the same time, we also analyze rich-text editors available on very popular sites like Twitter, Yahoo Mail, Amazon, GitHub and Magento and many more. In order to analyze online **WYSIWYG** editors, this paper also present a systematic and **WYSIWYG** editors's specific XSS attack methodology. We apply the XSS attack methodology on online **WYSIWYG** editors and found XSS is all of them. We show XSS bypasses for old and modern browsers. We have responsibly reported our findings to the respective developers of editors and our suggestions have been added. In the end, we also point out some recommendations for the developers of web applications and **WYSIWYG** editors.

1 Introduction

Cross-Site Scripting (XSS) vulnerabilities in modern web applications are now "`an epidemic`". According to Google Vulnerability Reward Program (GVRP)

© Springer International Publishing Switzerland 2015
K.-H. Rhee and J.H. Yi (Eds.): WISA 2014, LNCS 8909, pp. 122–133, 2015.
DOI: 10.1007/978-3-319-15087-1_10

report of 2013, XSS is at number one as far as valid bug bounty submissions are concerned [1]. According to Google Trends, XSS is googled more often than SQL injection for the first time in history [2]. Recently, an XSS attack has been successfully used for the closure of a very popular web and mobile application i.e., TweetDeck [3]. The XSS issue in TweetDeck was able to affect more than 80,000 users within 96 min [13]. The XSS in **WYSIWYG** editors is considered more dangerous, effective and exploitable because the user-supplied rich-text contents (may be dangerous) are most of the time viewable by other users of the web applications e.g., we found an XSS in **WYSIWYG** editor of Twitter Translation center's forum[1] even in the presence of a Content Security Policy (CSP) [14] (see Sect. 3.1). The CSP is the W3C standard for the mitigation of an XSS attack. In case of an XSS in **WYSIWYG** editor, the attacker does not need to trick user to visit his page.

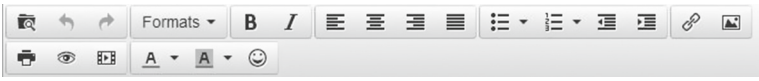

Fig. 1. A **WYSIWYG** Editor

The online **WYSIWYG** (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et) or rich-text editors (see Fig. 1) are main component of the modern web applications. **WYSIWYG** editors allow users of web applications to edit and enter HTML-based rich text (i.e., formatted text e.g., **bold**, *italic* and <u>underline</u>, *images*, *links* and *videos* etc.) inside the web browser. The modern web applications use **WYSIWYG** editors as a part of comment feature, private messaging among users of applications, blogs, wiki, notes, forums post, spellcheck as-you-type, ticketing feature, and other online services. The main purpose of rich-text editors is to provide users of web applications a better editing experience. The third-party **WYSIWYG** editors are normally available in the form of client-side JavaScript library, PHP or ASP based sever-side component and Rails gem.

The online cross-browser **WYSIWYG** are very popular e.g.:

- **Jive** — very popular editor and in use on sites like Amazon, T-Mobile and Thomson-Reuters etc. [5].
- **TinyMCE** — Javascript HTML **WYSIWYG** editor and in use on sites like XBox, Apple, Open Source CMS Joomla and Oracle etc. [6].
- **Lithium** — another popular rich-text editor and in use on sites like Paypal, Skype and Sephora etc. [8].
- **Froala** — jQuery **WYSIWYG** text editor and has been downloaded around 6000 times within two and half months of its launch [9].
- **EditLive** — an advanced **WYSIWYG** editor and 1500 organizations like Verizon, The New York Times and Nissan etc. are using it [11].

[1] https://translate.twitter.com.

- **CKEditor** — it has been downloaded 9472723 times and in use in sites like MailChimp, IBM and Terapad etc. [4]. It is formally known as FCKEditor.
- **Markdown** — another popular rich-text editor and in use on sites like Twitter, GitHub and Gitter (a private chat service for GitHub) [12].

This paper presents a study of analyzing 20 popular **WYSIWYG** editors. In order to evaluate **WYSIWYG** editors, we also present a systematic attack methodology (see Sect. 2.2). For testing purpose, we use the demo pages available by **WYSIWYG** editor. All the testing is carried out on the demo pages so that it will not harm any real user of the respective editor (see Sect. 2.1). Further, we also study home-grown **WYSIWYG** editors available on top sites like Yahoo Mail, Twitter and Magento Commerce. During evaluation of our attack methodology, we were able to break all **WYSIWYG** editors (see Sect. 3). We found XSS bypasses for old and modern browsers. We have responsibly reported our findings to the respective projects and our suggestions have been added in **WYSIWYG** editors like TinyMCE, Lithium, Jive and Froala. At the same time, we were awarded bug bounties by companies like Magento Commerce, GitHub and Paypal for finding bugs in their **WYSIWYG** editors and acknowledged by Twitter, Paypal and GitHub on their security hall of fame pages. To the best of our knowledge, this is the first study of analyzing XSS attacks in **WYSIWYG** editors. In the end, we also recommend best practices that **WYSIWYG** editors and web applications may adopt for the mitigation of an XSS attack (see Sect. 4).

This paper makes the following contributions:

- A security analysis of 20 popular **WYSIWYG** editors. The complete list of **WYSIWYG** editors is avaiable in the appendix (see Sect. A in appendix). Further, we also analyze **WYSIWYG** editors of top sites like Paypal, Yahoo, Amazon, Twitter and Magento.
- A systematic and step-wise attack methodology for evaluating **WYSIWYG** editors.
- Our suggestions have been added in top **WYSIWYG** editors like Lithium, Jive, TinyMCE and Froala.
- We also point out best practices that **WYSIWYG** editors and web applications may use in order to minimize the affect of XSS.

2 Methodology

In this section, we describe the testing and attack methodology.

2.1 Testing Methodology

In this section, we briefly describe the testing process. In order to test **WYSIWYG** editors, we use the demo pages made available by the respective developers of **WYSIWYG** editors. The main advantage of testing on demo pages is that it will not harm any user. In case, we found an XSS during testing process,

we act responsibly and filled the bug(s) on GitHub or directly report via email. During testing of **WYSIWYG** editors, we identify common injection points (almost all **WYSIWYG** editors support these injection points) that are of an attacker interest e.g., link creation, image and video insertion, description of images, class or id names and styling of contents. In the next section, we will present a respective XSS attack methodology for these injection points.

2.2 Attack Methodology

In this section, we describe the XSS attack methodology for the common injection points identified in previous section. The attack methodology for every injection point is systematic in nature.

Attacking Link Creation Feature: All **WYSIWYG** editors support "`create link`" feature. The "`create link`" functionality corresponds to HTML's anchor tag i.e., `<a>` and its "`href`" attribute. The user-supplied input as a part of "`create link`" in **WYSIWYG** editor lands as a value of "`href`" attribute. The attacker can abuse this functionality with the help of following steps (see Fig. 2) and can execute arbitrary JavaScript in the context of a web application. The step ❶ makes use of JavaScript URI e.g., `javascript:alert(1)` in order to execute JavaScript e.g., we found XSS via JavaScript URI in Froala, EditLive, CNET's **WYSIWYG** editor and Twitter etc. The attacker can also use different types of encoding in JavaScript URI e.g., "`javascript:alert%28 1%29`" (URL encoded parenthesis) and "`jav	ascr	ipt:alert(1)`" (HTML5 entity encoding). In case, **WYSIWYG** editor filters the word "`javascript`", then attacker can use DATA URI based JavaScript execution in step ❷ e.g., "`data:text/html; base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+`". We found XSS via DATA URI in Jive because Jive does not allow JavaScript based URI. In step ❸, the attacker can also leverage VbScript based code execution but it is limited to Internet Explorer browser. In last step i.e., step ❹, the attacker can make use of valid URL but as a part of query parameter's value, he uses the following attack string i.e., `"onmouseover="alert(1)` in order to break the URL context. The step ❹ is very useful in case if **WYSIWYG** editors only accept URLs start with "`http(s)`". We found XSS in Amazon's **WYSIWYG** editor with the help of step ❹.

Fig. 2. Attack Methodology for Link Creation Feature

Attacking Image Insertion Feature: Another common functionality that all **WYSIWYG** editors support is "`Insert/Edit Image`". The "`Insert/Edit`

`Image`" feature corresponds to HTML's <`img`> and its "`src`" attribute. The user-supplied input as a part of "`Insert/Edit Image`" in **WYSIWYG** editor lands as a value of "`src`" attribute. The attacker may use the following XSS attack methodology (see Fig. 3) in order to abuse this feature. The step ❶ consists of a valid "`jpg`" image URL ends in `?` and after the question mark `"onmouseover="alert(1)`. In URL, the question mark symbol is legally valid and all browsers respect it while at the same time for the **WYSIWYG** editors, it is also a legit input at this point because their implementations expect input to be a valid URL but then we used hard-coded `"` symbol and the sole purpose is to break or jump out of the context and execute JavaScript via eventhandler e.g., onmouseover. We found XSS in Amazon's **WYSIWYG** editor with the help of first step. The step ❷ consists of a valid SVG image hosted on free domain for demo purpose. The step ❷ serves two purpose:

1. JavaScript execution via SVG image. We found XSS in GitHub's rich-text markup feature with the help of an SVG image and we were awarded bounty for that. In favor of space restrictions, we refer to the work by Heiderich *et al.* in [15] and it shows how an attacker can leverage SVG images for arbitrary JavaScript code execution.
2. If **WYSIWYG** editors are doing explicit decoding on the server side then JavaScript can be executed because decoding will convert the %22 into hard-coded `"`, which in turns break the context. We found XSS in Alexa's rich-text tool bar creation feature with the help of this technique.

(1) http://www.ieee-security.org/images/new-web/Trojan_Horse.jpg?"onmouseover="alert(1)

(2) http://xssplayground.net23.net/xss%22onmouseover=%22alert(1);%20imagefile.svg

Fig. 3. Attack Methodology for Image Insertion Feature

Attacking "`alt`", "`id`" and "`class`" attributes: Another common injection points that we found in almost all **WYSIWYG** editors are "`alt`" attribute of an <`img`> tag. In **WYSIWYG** editors, user can specify the image description as a value of an "`alt`" attribute. In a similar manner, we found attributes like "`id`" and "`class`" are common across all **WYSIWYG** editors. The attacker can abuse these injection points with the help of following XSS attack methodology (see Fig. 4). The step ❶ consists of attack vector "`anytext"onmouseover="alert(1)`". It is clear from the attack string that if **WYSIWYG** editors fail to properly sanitize/filter `"`, then the attack string will jump out from the attribute context and attacker can execute JavaScript. We found XSS in Yahoo Mail's **WYSIWYG** editor with the help of this attack vector. The step ❷ is related to

innerHTML based XSS and specific to old Internet Explorer (IE) browser. The old IE browser treats back-tick i.e., (``) as a valid separator for attribute and its value. The back-tick based XSS attack string is very useful in cases where **WYSIWYG** editors properly filter double quotes (") and do not allow to break the context e.g., the following XSS attack vectors would result in an innerHTML based XSS in IE8 browser. `<div class="``onmouseover=alert(1)">`div layer ` `, `<a href="#" id="``onmouseover=alert(1)">`click`` and `<img src="x" alt="``onmouseover=alert(1)">` etc. Almost all **WYSIWYG** editors are vulnerable to innerHTML based XSS including Lithium, TinyMCE, Froala and GitHub's **WYSIWYG** editor. For details about innerHTML based XSS, we refer to the recent work by Heiderich *et al.* in [16].

Fig. 4. Attack Methodology for Attributes

Attacking Video Insertion Feature: **WYSIWYG** editors (not all) support "Insert Video" feature. As a part of this feature, **WYSIWYG** editors only allow HTML's `<object>` and/or `<embed>` tag. The attacker can abuse this feature with the help of following example code snippet (see Fig. 5). The code snippet is taken from the Youtube's video sharing via embedded code feature. In the perfectly legit code snippet, we have added "`onmouseover=alert(1)`" in order to fool **WYSIWYG** editors. We found XSS in froala with the help of this trick.

```
<object width="560" height="315" onmouseover=confirm(1)><param name="movie" value="//www.youtube-nocookie.com
/v/KoA7Cpujrus?version=3&amp;hl=en_US"></param><param name="allowFullScreen" value="true"></param><param
name="allowscriptaccess" value="always"></param><embed src="//www.youtube-nocookie.com/v/KoA7Cpujrus?version=3&
amp;hl=en_US" type="application/x-shockwave-flash" width="560" height="315" allowscriptaccess="always"
allowfullscreen="true"></embed></object>
```

Fig. 5. Attack Methodology for Video Insertion Feature

Attacking "`Styles`": All **WYSIWYG** editors support "`styling`" of the rich-text contents e.g., user can specify the height, width, color and font properties of the contents. The attacker can easily abuse this feature with the help of CSS expressions [17]. The old versions of Internet Explorer (IE) browsers support JavaScript execution via CSS expressions. The XSS attack vectors may use for this purpose are: "`width:expression(alert(1))`" or "`x:expr/**/ession (alert(1))`". We found XSS in Ebay, Magento, Amazon, TinyMCE and CKEditor's **WYSIWYG** editors with the help of "`styles`".

3 Evaluation of Attack Methodology

In this section, we discuss the results of evaluating attack methodology on popular **WYSIWYG** editors. We were able to break all **WYSIWYG** editors in one or other common injection points discussed in the previous section. In favor of space restrictions, here we discuss three examples that we consider worth sharing as a part of evaluation.

3.1 XSS in Twitter Translation Forum's WYSIWYG editor

Twitter[2] (Alexa rank `#9`), a popular social networking web site. Twitter is one the handful of web sites that are using CSP for the mitigation of XSS attacks. Twitter's CSP is available at the following URL http://i.imgur.com/ESkQG9O.jpg. The CSP explicitly tells the browser about trusted resources for images, script, media, and styles etc. Twitter Translation[3] is one of the Twitter's service where community can help in translating Twitter related stuff in different languages. On Twitter Translation forum, we found that it supports rich-text markup feature. The following Fig. 6 shows Twitter's markdown cheat sheet. We found one of the way of specifying links in the forum post is: `[Twitter]` `(https://twitter.com)` (see area marked in red in Fig. 6). As discussed in previous section (see Sect. 2.2), the attacker can abuse the link creation feature with the help of JavaScript, Data and VbScript URI. By keeping in mind attack methodology related to "`create link`", we input the following: `[Twitter]` `(javascript:alert(1))`. Twitter internally treats the above input in the following manner: `<a href="javascript:alert(1">`Twitter`</a>`. The JavaScript does not execute because of the missing closing parenthesis in "`alert(1`". The reason we found is: Twitter's **WYSIWYG** editor's syntax is causing problem because internally it treats the closing parenthesis of "`alert(1)`" as "URL ends here" and did not look for the last parenthesis. As a part of next step, we convert the small parenthesis into the respective URL encoded form i.e., (becomes %28 and) becomes %29. The next attack string looks like: `[Twitter]` `(javascript:alert%28 1%29)` and this time it works as expected and internally it looks like: `<a href="javascript:alert%28 1%29">`Twitter`</a>`. The following Fig. 7 shows JavaScript execution in Twitter's **WYSIWYG** editor.

3.2 XSSes in TinyMCE's WYSIWYG editor

TinyMCE [6] is one of the most popular **WYSIWYG** editor. We found three different XSSes in TinyMCE: one in "`create link`" feature (see Sect. 2.2), one in "`styling`" feature (see Sect. 2.2) and one innerHTML based XSS in "`alt`" attribute (see Sect. 2.2). We filled three different bugs[4] in TinyMCE's bug tracker and now all XSSes have been fixed [7]. The following list items summarizes our findings:

[2] https://twitter.com/.
[3] https://translate.twitter.com.
[4] http://www.tinymce.com/develop/bugtracker_view.php?id=6855|6851|6858.

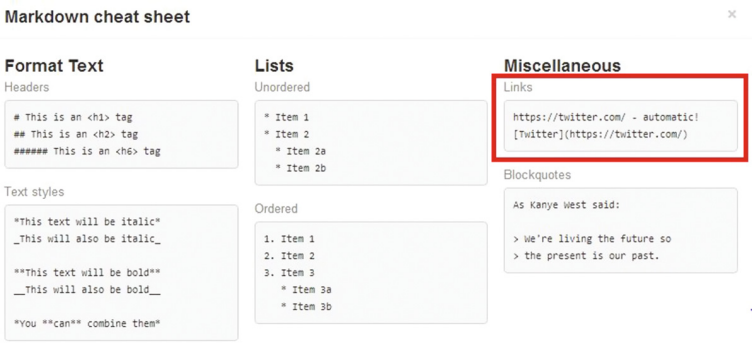

Fig. 6. Markdown Cheat Sheet

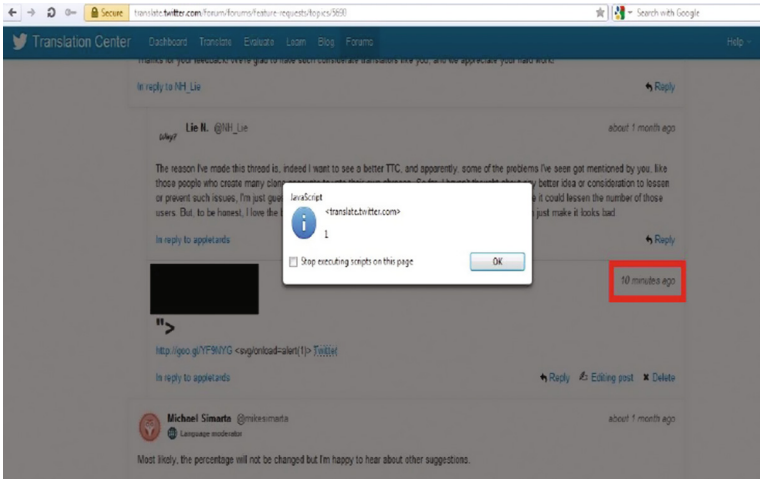

Fig. 7. XSS in Twitter Translation

1. TinyMCE was vulnerable to an XSS in "`create link`" feature with the help of DATA URI i.e., "`data:text/html; base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+`".
2. TinyMCE was vulnerable to an XSS in "`style`" functionality. In order to execute XSS, we used CSS expressions i.e., "`x:expr/**/ession(alert(1))`". TinyMCE's implementation does not allow the word "`expression`" as a part of styles and that's why we used "`expr/**/ession`" i.e., use of multi-lin comments in "`expression`" word and old IE browsers simply ignores it.
3. TinyMCE was also vulnerable to an innerHTML based XSS and the attack vector used for this purpose was: ``onmouseover=alert(1).

3.3 XSSes in Froala's WYSIWYG editor

Froala — jQuery **WYSIWYG** text editor is also one of the popular and latest rich-text editor [10]. We found XSSes in almost all common injection points identified in Sect. 2.2. A bug[5] has been filled on GitHub and all reported XSS issues have been fixed in the upcoming version. The following list items summarizes our findings:

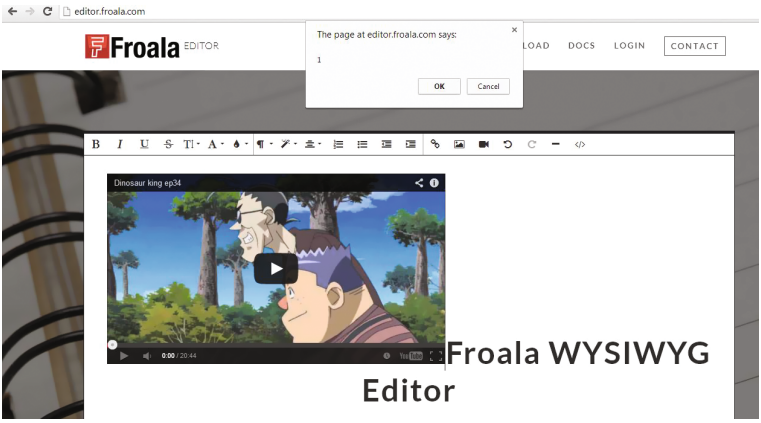

Fig. 8. XSS in Froala

1. "`Create Link`" feature was vulnerable to an XSS e.g., "`javascript:alert(1)`" and "`data:text/html; base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+` worked.
2. "`Insert Image`" feature was vulnerable to a trick discussed in previous section (see Sect. 2.2) i.e., attacker can execute JavaScript with the help of following: `http://www.ieee-security.org/images/new-web/Trojan_Horse.jpg?"onmouseover="alert(1)`.
3. "`Insert Video`" feature was vulnerable to the attack method discussed earlier (see Sect. 2.2). The Fig. 8 shows XSS in Froala via "`Insert Video`" functionality.
4. "`Image Title`" feature was vulnerable to an innerHTML based XSS attack method discussed earlier (see Sect. 2.2).

4 Practical and Low Cost Countermeasures

Web applications normally integrate third-party **WYSIWYG** editor(s) in order to give customers of web application a better and rich editing experience. In this section, we discuss low cost, practical and easily deployable countermeasures that web applications may adopt in order to minimize the affect of an XSS in **WYSIWYG** editor(s). Further, we also recommend some suggestions to the developers of **WYSIWYG** editors.

[5] https://github.com/froala/wysiwyg-editor/issues/33.

4.1 HttpOnly Cookies

Web applications use cookies in order to maintain a session state between authenticated client and server because of the stateless nature of an HTTP protocol. If a flag "`HttpOnly`" is set on a cookie then JavaScript can not read the value of this cookie and all modern browsers respect this. We recommend web applications' developers to use "`HttpOnly`" cookie especially if **WYSIWYG** editor is in use. In case of an XSS in **WYSIWYG** editor, the attacker can not read the session cookie of the victim with the help of JavaScript. We found an XSS in Magento's (an Ebay company) **WYSIWYG** editor and found "`PHPSESSID`" cookie was not "`HttpOnly`" and at the same time site only allows authenticated users to post on forum. With the help of this XSS in **WYSIWYG** editor, attacker may steal the session cookie of the forum administrator and hack the forum. The XSS in Magento's **WYSIWYG** editor is now fixed and the details are available in a post here[6]. Magento acknowledged our findings and we were awarded thousand US dollar in the form of bug bounty.

4.2 Iframe's "`sandbox`"

We recommend developers of the web applications to use <`iframe sandbox`> in order to integrate third-party **WYSIWYG** editor. With the help of "`sandbox`" attribute, the developers of the web applications can restrict the capabilities of third-party **WYSIWYG** editor. In case of an XSS in **WYSIWYG** editor, if "`sandbox`" attribute is used then attacker can not access the DOM contents of the main web page or locally stored data on the client side. All mordern browsers support iframe's "`sandbox`". For details about <`iframe sandbox`>, we refer to [18] for interested readers.

4.3 Content Security Policy

We recommend developers of web applications to retrofit their applications for CSP [14]. The CSP policy is now a W3C standard for the mitigation of an XSS attacks. The CSP is based on directives for images, script, media, styles and iframes etc. The developers of web applications can explicity tell the browser about the trusted resources. By default, CSP prohibits inlining scripting. In case of an XSS in **WYSIWYG** editor, if CSP is defined then first browser will not allow injected, inline script to execute (unless "`unsafe-inline`" directive is specified) and second CSP helps in minimizing the affect of an XSS because attacker can not ex-filtrate sensitive data to his domain.

4.4 Guidelines for Developers of WYSIWYG editors

In this section, we briefly describe some guidelines for **WYSIWYG** editors' developers.

[6] http://www.scribd.com/doc/226925089/Stylish-XSS-in-Magento-When-Style-helps-you.

- Should force users to input URL or "`create link`" that starts with an "`http://`" or "`https://`".
- Should not allow SVG images. With the help of an SVG image, attacker may execute JavaScript.
- Should properly encode potentially dangerous characters in attributes e.g., double quotes and back-tick because these characters can help attacker to break the attribute context and execute JavaScript.
- Should not allow CSS expressions in "`styling`" of the contents.
- Should not allow Flash-based movies because attacker can execute JavaScript code via Flash file.
- In case, **WYSIWYG** editor allows to upload a file then developers should validate the file type.

5 Conclusion

In this paper, we analyzed twenty popular **WYSIWYG** editors and found XSS in all of them. We had presented a systematic XSS attack methodology for common injection points in **WYSIWYG** editors. We hope that this paper will raise awareness about the XSS issue in modern feature of an HTML5-based web applications i.e., rich-text editors.

A List of WYSIWYG Editors

1. Mercury Editor: The Rails HTML5 WYSIWYG editor (http://jejacks0n.github.com/mercury)
2. bootstrap-wysihtml5: Simple, beautiful wysiwyg editor (https://github.com/jhollingworth/bootstrap-wysihtml5)
3. KindEditor (http://kindeditor.org/)
4. PHP HTML Editor (http://phphtmleditor.com/demo/)
5. elRTE — an open-source WYSIWYG HTML-editor (http://elrte.org/)
6. medium-editor (https://github.com/daviferreira/medium-editor)
7. TinyMCE (http://www.tinymce.com/)
8. Lithium (http://www.lithium.com/)
9. Jive (http://www.jivesoftware.com/)
10. Froala (http://editor.froala.com/)
11. CKEditor (http://ckeditor.com/)
12. EditLive (http://ephox.com/editlive)
13. jquery.qeditor (https://github.com/huacnlee/jquery.qeditor)
14. mooeditable (http://cheeaun.github.io/mooeditable/)
15. HTML5 WYSIWYG Editor (https://github.com/bordeux/HTML-5-WYSIWYG-Editor)
16. markItUp! universal markup jQuery editor (http://markitup.jaysalvat.com/home/)
17. FreeTextBox HTML Editor (http://www.freetextbox.com/)
18. Markdown (http://daringfireball.net/projects/markdown/)
19. CLEditor (http://premiumsoftware.net/CLEditor/SimpleDemo)
20. Bootstrap Wysihtml5 with Custom Image Insert (https://github.com/rcode5/image-wysiwyg-sample)

References

1. Google Vulnerability Reward Program Report for year 2013.: https://www.youtube.com/watch?v=oAYjZy1Nuyg
2. Google Trends.: http://www.google.com/trends/explore#q=XSS%2C%20SQL%20Injection&date=today%2012-m&cmpt=q
3. TweetDeck ShutDown.: https://twitter.com/TweetDeck/status/476770732987252736
4. CKEditor.: http://ckeditor.com/about/who-is-using-ckeditor
5. Jive.: http://www.jivesoftware.com/why-jive/customers/#view=list
6. TinyMCE.: http://www.tinymce.com/enterprise/using.php
7. TinyMCE Tracker.: http://www.tinymce.com/develop/bugtracker.php
8. Lithium.: http://www.lithium.com/why-lithium/customer-success/
9. Froala.: https://github.com/stefanneculai/froala-wysiwyg/issues/33#issuecomment-41170451
10. Froala Editor.: http://editor.froala.com/
11. Edit Live.: http://ephox.com/customers
12. Markdown.: http://daringfireball.net/projects/markdown/
13. From "I wonder..." to Exploitable Worm in 96 Minutes.: https://storify.com/pacohope/from-i-wonder-to-exploitable-worm
14. Content Security Policy 1.1.: http://www.w3.org/TR/CSP11/
15. Heiderich, M., Frosch, T., Jensen, M., Thorsten, H.: Security risks of scalable vectors graphics. In: CCS, Crouching Tiger - Hidden Payload (2011)
16. Heiderich, M., Schwenk, J., Frosch, T., Magazinius, J., Yang, E.Z.: mXSS attacks: attacking well-secured web-applications by using innerHTML mutations.. In: CCS (2013)
17. About Dynamic Properties.: http://msdn.microsoft.com/en-us/library/ie/ms537634(v=vs.85).aspx
18. Play safely in sandboxed IFrames.: http://www.html5rocks.com/en/tutorials/security/sandboxed-iframes/