Evading Web XSS Filters through Word (Microsoft Office and Open Office) in Enterprise Web Applications

_____

Date: 11 March 2009

_____

Aditya K Sood, http://www.secniche.com | http://www.secniche.org | adi_ks [at] secniche.org

Abstract

This paper sheds light on the hyper linking issues observed during penetration testing of web based enterprise applications. This concept can be used to bypass standard XSS filters by creating a malicious Microsoft word document. The inline hyper linking with malicious code subverts the enterprise web application XSS filters (while conversion mechanism) when rendering is done in the context of browsers. As the code base is structured in a hierarchical manner, we will be laying stress on the vendor based applications. The web XSS filters present in the enterprise web application are not designed appropriately to trace the injection parameters. Numbers of vendors have been intimated against this XSS vector. This can also be used to launch other variants of XSS including Cross Site Request forging and Remote File Inclusion attacks. It works efficiently in most number of cases. The risk factor and feasibility depends on the ease of victim interacting with the enterprise web application. The attack vector works efficiently with Microsoft word and Open Office. But for discussion in detail we will be considering Microsoft Office Word.

Description:

The enterprise web applications allow uploading of Microsoft word document on the web server based on the requirements. It has been noticed that certain vendors allow the content of these Microsoft word documents to be rendered into the browser as a part of functionality in an application. For example: loading your resume in a word document on the web server. It can be opened in the application context to view the information. It has been detected that the transformation of content from the Microsoft document file is not properly filtered and malicious operations are allowed on the browser surface when the content is rendered. We will understand the behavior of Microsoft word document while handling hyper links. The behavior of Microsoft word is generic when a hyperlink is clicked directly. When a request is made through office, then a file is downloaded directly through browser (default internet explorer).

As we know, any content not mapped according to the structure defined, is undertaken as HTML by browser, It has been critically examined that Microsoft has applied proper security measures when a document is opened from website in the native Microsoft office software. But previewing the contents of Microsoft word document file directly in the browser is more of HTML. If an appropriate filter is not applied in the enterprise software, then malicious operations can be done with an ease.

The problems are identified and structured as:

The enterprise software does not use inline filtering technique to scrutinize the URL passed in the Microsoft word document for standard injection blacklists, when content is transformed. Malicious designed JavaScript which is passed as a hyperlink in the word document does the trick for the attackers.

Two specific cases arise in this:

1. When the content in the Microsoft word Document is rendered (conversion from doc to html) as HTML directly in the browsers.
2. When the content is rendered in the proprietary base software using a web browser.

The hyperlinks in Microsoft word are opened through web authoring protocol directly in the web browser, mainly in the read only format. It depends on the specification provided in the web server configuration and the type of request issued pointing to the required document. The SAVE-AS option allows you to save the file directly on the system base. As per the RFC 2616 HTTP/1.1 allows HTTP method OPTIONS which is used to set different options on the web resource. It defines the command and the type of request to be received by the server on the specific folder where the document or resource is placed. So opening a Microsoft word document in web browser requires certain steps to be covered by the Microsoft Office software through Office Discovery Protocol. The standard requirement and procedures are listed below as:

1.) The Web Authoring Protocol.

2.) Web Server issuing the request and the Authentication mechanism to be followed.

3.) The request methods in use and specification provided on the resource.

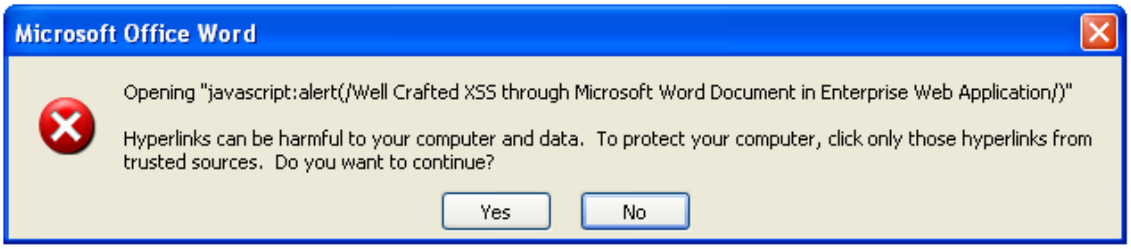
4.) Access methods deployed to use that resource.

5.) Time check covering the Expiration Time.

These are the certain methods which Office requires to complete the loading and viewing of the document in the browser. Once the hyperlink is clicked from a browser (Internet Explorer) the URLMON

finds a resource and downloads the content directly. The URLMON checks the HTTP header, Content-Type, CLSID to find the content type. Once it is done, an OLE object is created through IPersistMoniker. The new HLINK object is generated pointing to the resource through IPersistMoniker and the file is displayed to the user.

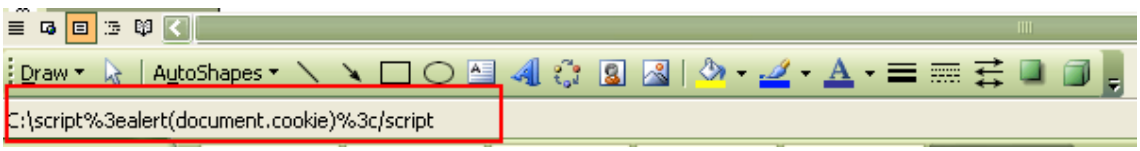
From the security perspective, the same policies and ingrained security mechanisms are applied as the base software.



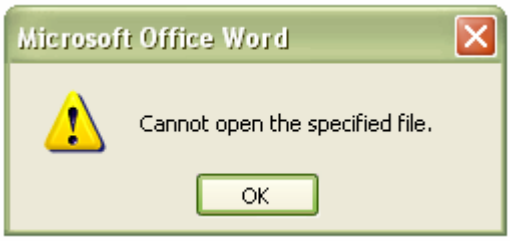
The Hlink.dll plays a critical role in executing a hyperlink that works for any Microsoft Word Document. To maintain the integrity, there is a secondary check introduced by Microsoft even if the hyperlink security warning is disabled in the registry. So that's how the actual functioning works in hyperlink execution. But our scope is wider than this. The problem actually starts from here. The functionality of editing, creating hyperlinks diversifies the attack pattern in enterprise web application when a word document is rendered directly in web browsers. This trick comes handy in manipulating the word functionality in order to hit the web application.

The problem is not with Microsoft Word document but using it as a carrier to bypass the XSS filters through malicious hyper linking by inserting XSS checks. This type of attack vector requires a third party carrier to trigger the attack efficiently. If we reverse the order, then problem is present in the Hyperlink Integrity Checkers applied in the web applications. We will discuss a real example for injecting XSS, CSRF etc. in an enterprise web application. The filters for checking of Meta characters and injection parameters work efficiently in those applications, but at the same time fail to instantiate the XSS injected into hyperlinks during content conversion directly into the web browser. It means the publicly available and enterprise web application conversion components trust all the Microsoft word documents as flawless and execute the action. This is not right as even the content needs to be traversed especially hyperlinks (XSS driven) whether the JavaScript call has been made through it or not.

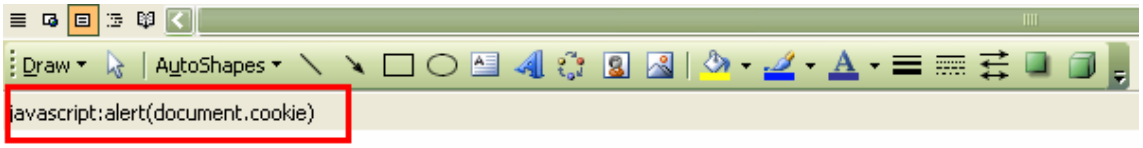
Microsoft provides a generic option of editing hyperlinks. It proves beneficial to the attackers for designing malicious hyperlinks with inline XSS parameters into it. Due to applied security, if an attacker passes parameter including Meta characters such as (<>), the hyperlink is appended to the URI which points to that file in a web server. As a result, the overall URL becomes the URI appended with hyperlink path which does not work and shows the sign of intrusion or injection. Let's see:



The lower pane in the Microsoft word document shows the hyperlink address that is interpreted by the software itself. It shows the behavior as discussed above. In order to inject the right parameter, we need to make injections inline so that the hyperlinks show only JavaScript code rather than the links. Generally, when this link is executed you will find the under mentioned error as a response:



So the hyper link does not work. By default, Microsoft word always tries to download the contents pointed by the hyperlink. In order to set the hyper linking for injections, one has to make it inline. Let see:



Now the parameters passed for injection are inline. There can be two checks that are produced by security mechanism in Microsoft word as:

1. If **DisableHyperlinkWarning** is not set, you will get a security check as:

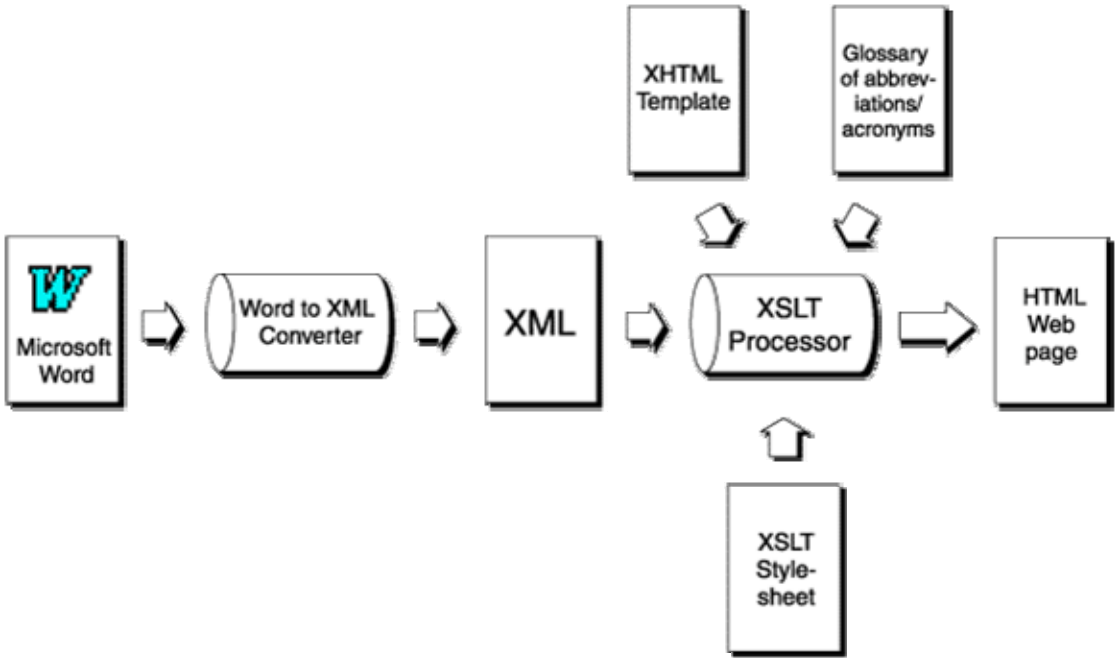
2. If **DisableHyperlinkWarning** is set, **the** Microsoft word tries to download the file directly or produces a trusted check as structured in HLINK.dll.

The hyperlink is working; the internet explorer tries to download the file as per the default behavior of Microsoft word. At this point of time, we are sure that our malicious link is injected properly in the Microsoft word document and is working fine.

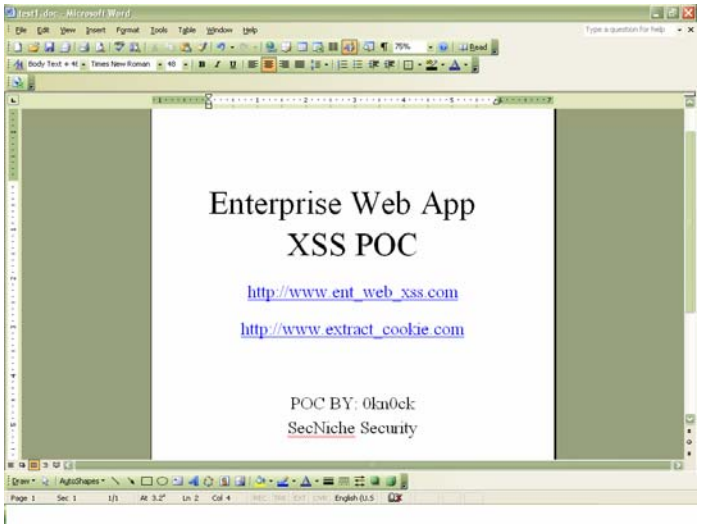
Microsoft released an advisory http://secunia.com/advisories/32138/ explaining the direct XSS occurring in the word through CDO URI parameter. This vulnerability is incessantly software driven. The XSS is produced when a document is directly downloaded from the web server in the browser.

Our aim is to make the XSS persistent through malicious hyper linking in the Microsoft word document. The file remains on the web server and when an owner renders it in a browser to view the content, the injection occurs. The next test is to upload the malicious word file into the enterprise web application. We will discuss this in next section.

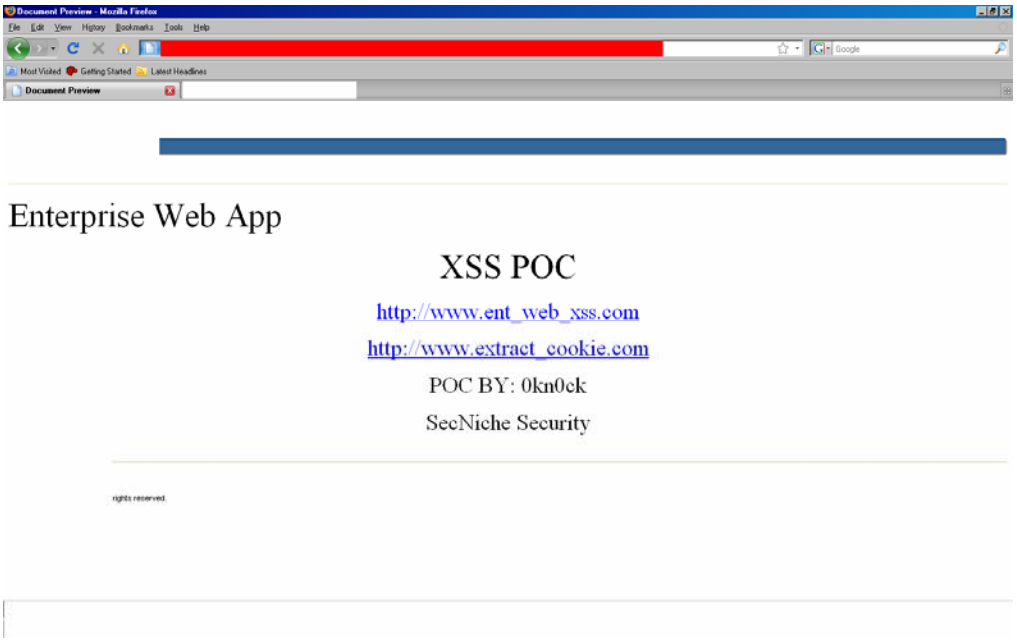
Now, we are ready with our malicious word document. The document will be uploaded into the enterprise web application on the server. We have taken a real world example, but due to legal and responsible disclosure concerns, we will not be pointing out the details of the web application. Let's have a look at the normal conversion mechanism from doc to html.



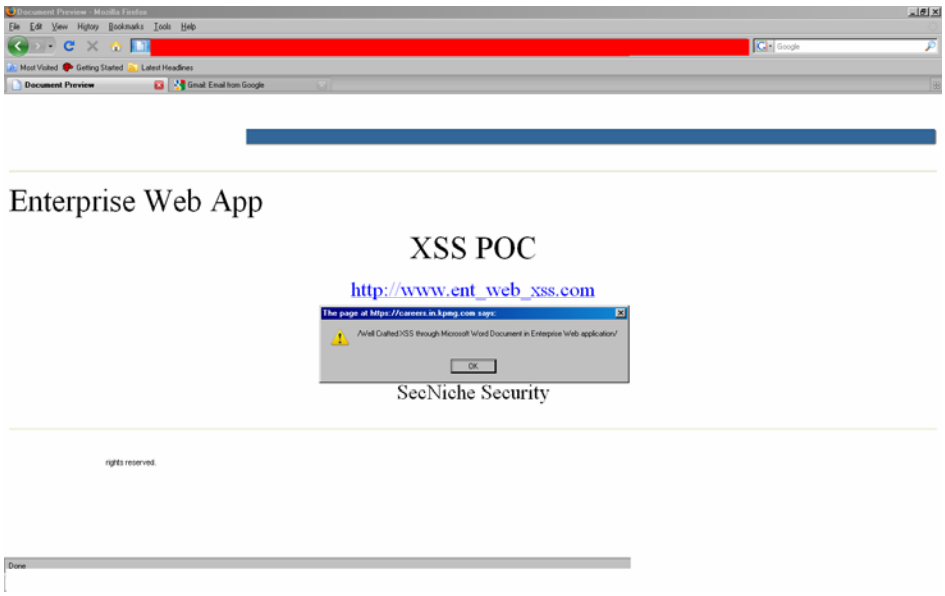
The transformation occurs as structured above. Let's see how the transformation occurs from malicious word document to html. The word document was designed as:



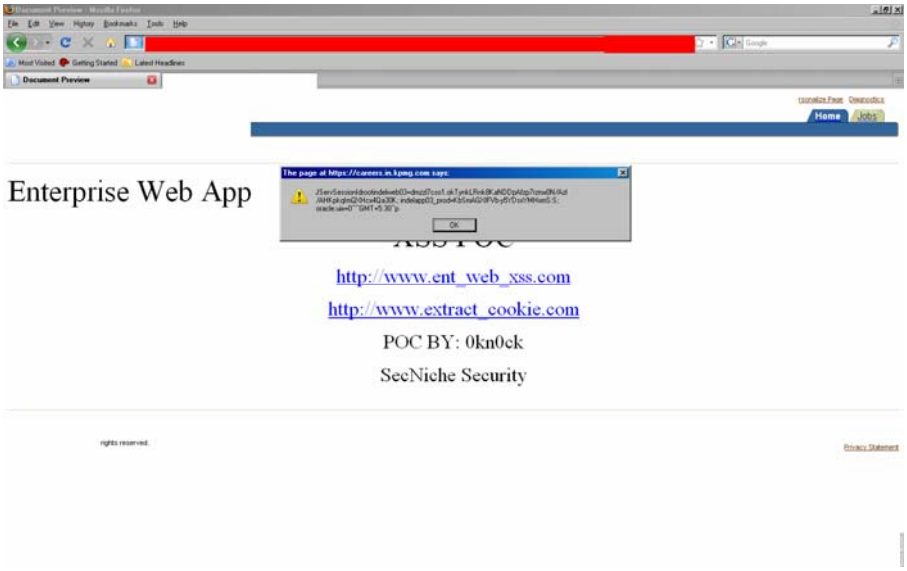
When the document was converted to html and previewed in the context of enterprise web application it looked as:



The malicious hyperlinks were rendered in the application. When we clicked the above specified link in the enterprise web application, we successfully produced XSS.

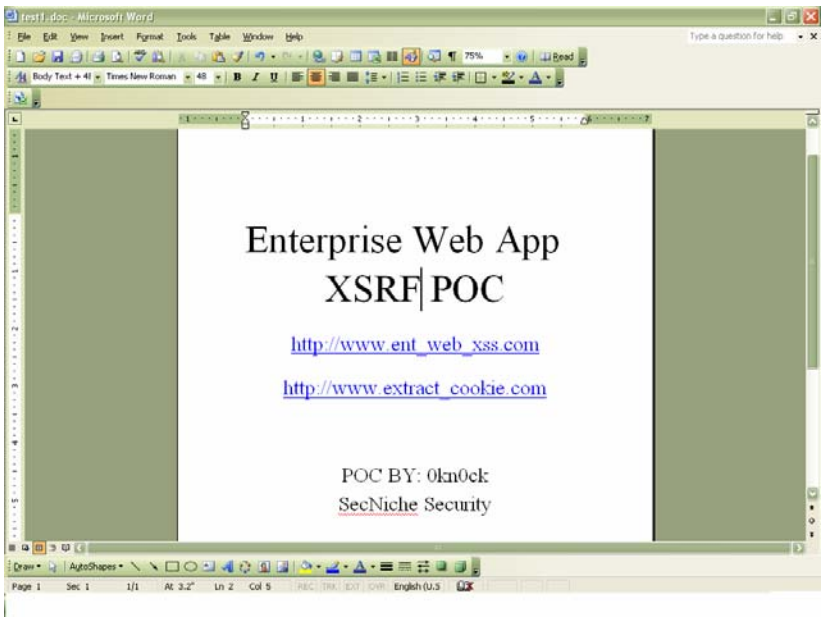


The alert box was displayed as a positive outcome of the injected link.

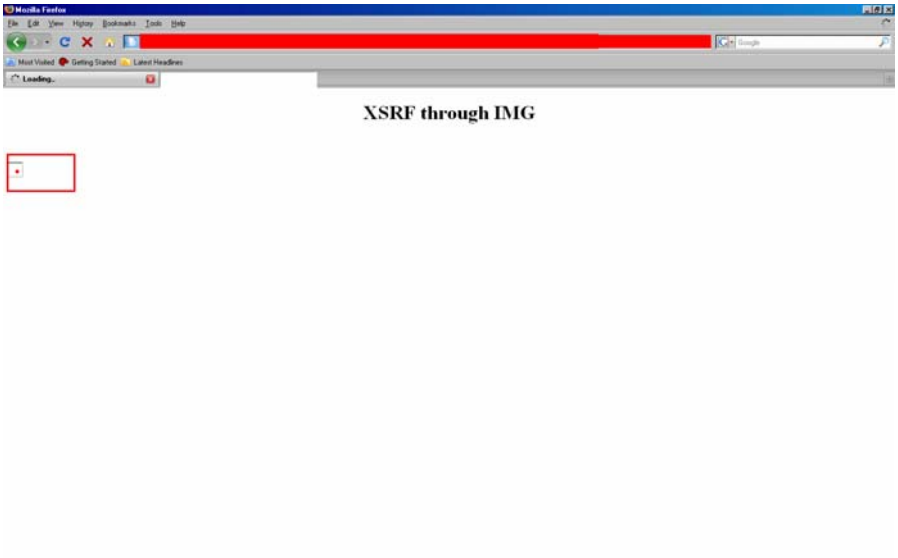


The cookie was extracted. We conducted another test to check the cross site request forging. This worked even better in the context of running enterprise web application.

Another malicious document was created with a forged XSRF request.



We encountered the following structured output which clearly represented the cross site request forging attack.



This injection occurred through document.writeln JavaScript call to execute the forged request. So we have noticed the impact of this kind of attack on the web applications. No doubt it's an obscure way of producing XSS but it results in real attack and stealing of information.

Conclusion:

The XSS contains a randomized vector of attack, as we have seen in the past about different variants of cross site scripting attacks. These types of attacks help the penetration testers to scrutinize the designed filters against the XSS. In addition, it also checks the vulnerability persisting due to insecure coding and inappropriate web application design in the application. These types of attacks require a kind of carrier (in this case Microsoft Word) to pass the injection parameters to the target. The attack vector is successful in triggering this type of behavior in a number of web enterprise web applications. This helps the tester to go through every single component of web application to trace the hidden vectors of XSS exploitation.

Thanks:

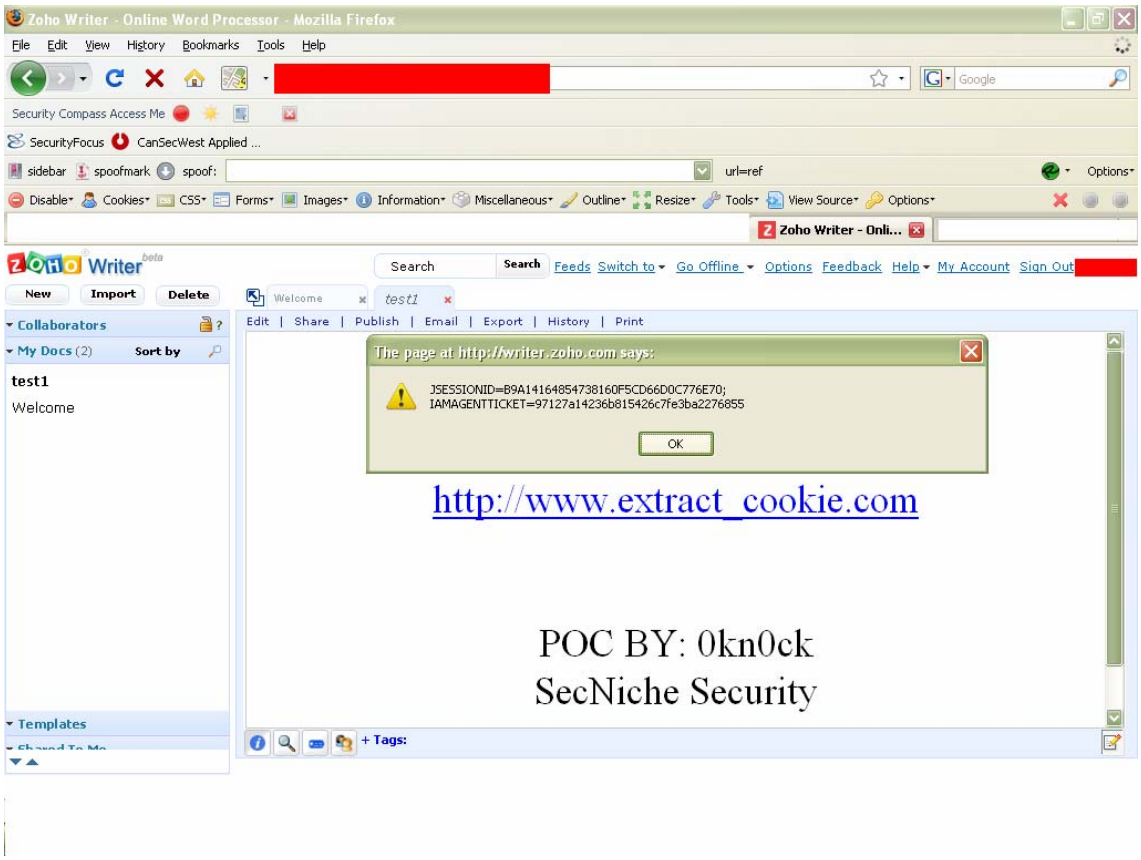
I would like to thank Mr. Jeremiah Grossman (WhitehatSec) and Johnny Long (iHackStuff) for the discussions and feedback in polishing this paper.

References:

1. www.scanit.be/**uploads**/php-file-**upload**.pdf
2. https://www.owasp.org/index.php/Unrestricted_File_Upload
3. http://technet.microsoft.com
4. http://ha.ckers.org/blog/20061215/csrf-with-word-part-ii/
5. http://michaeldaw.org/md-hacks/csrf-with-msword/

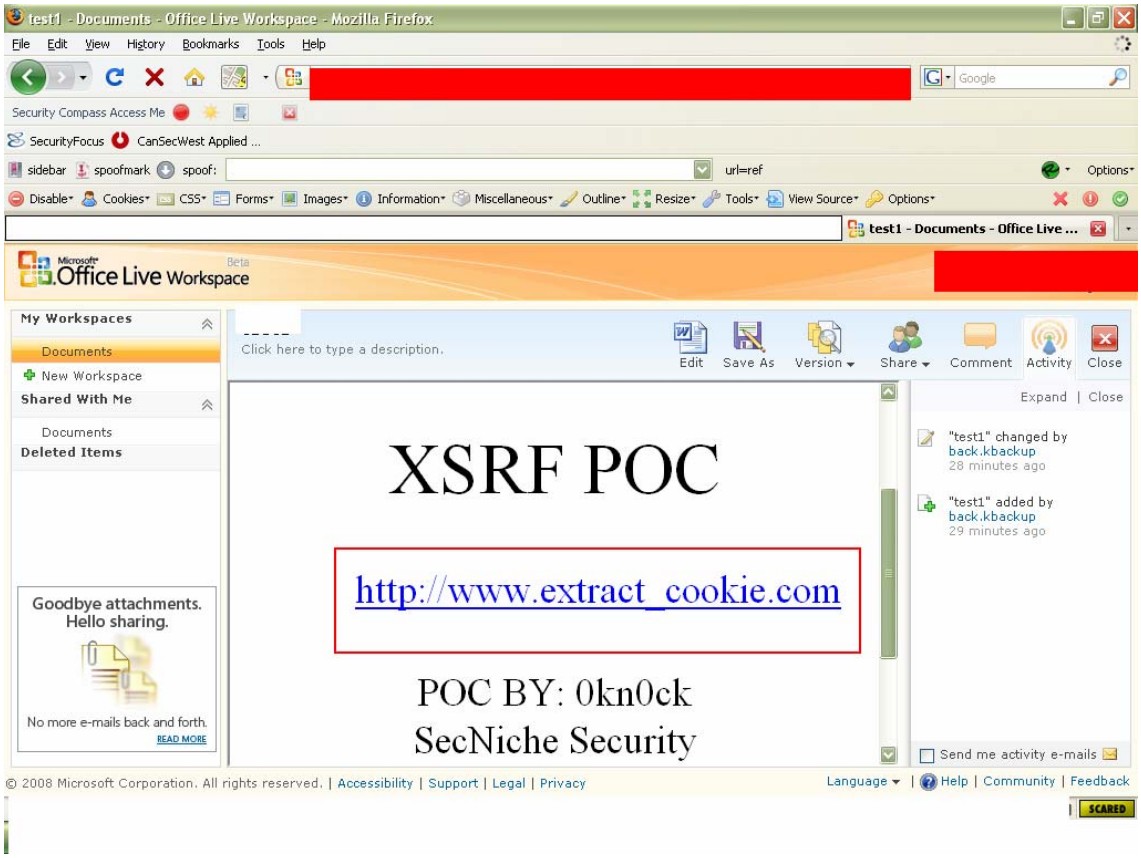
We have tested this functionality on certain number of Web based editors and found different behaviors as explained in the appendix.

Appendix 1 – ZOHO Writer



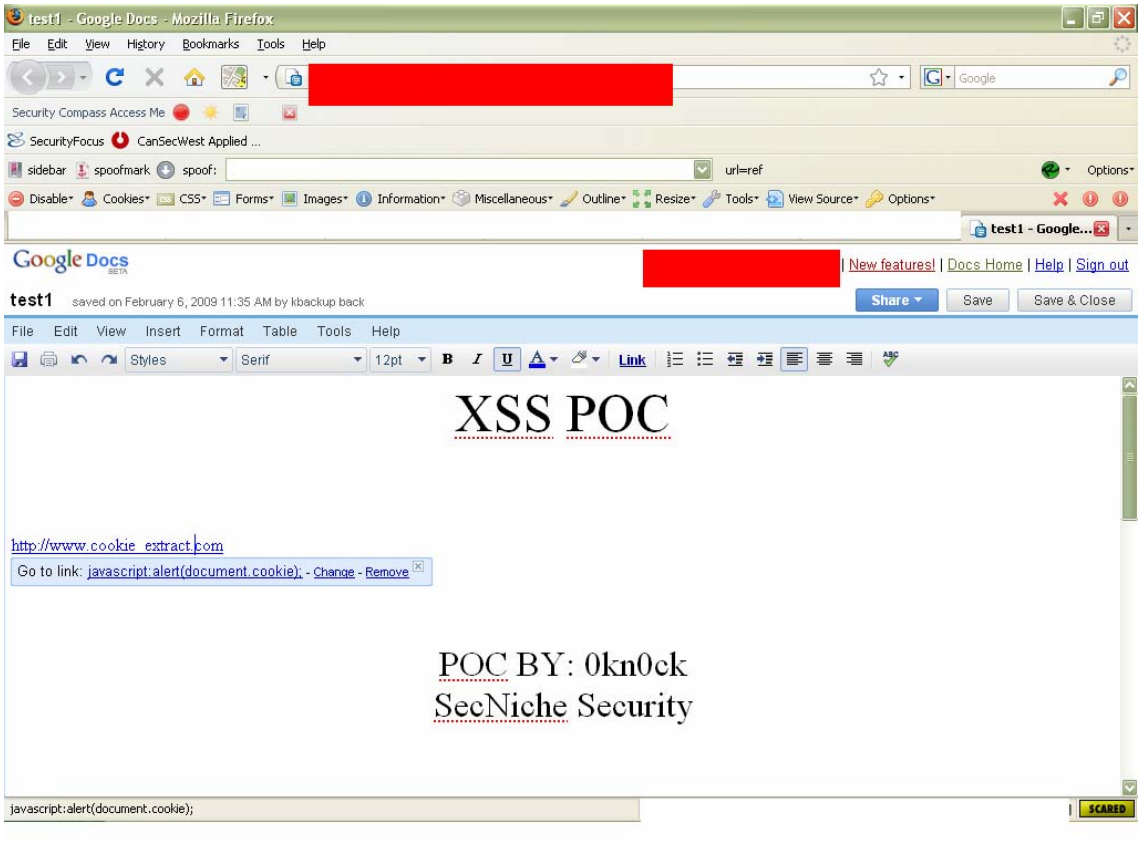
It does not filter the hyperlinks and injection is possible.

Appendix 2: Microsoft Office Live Workspace



The Live Office does not allow the link to be processed directly from the browser.

Appendix 3: Google Docs



Google docs add an extra check(Go to Link) while conversion of document.