# Post XSS Exploitation: Advanced Attacks and Remedies

**Nishtha Jatana[1], Adwiteeya Agrawal [2], Kritika Sobti [3]**

*[1]Assistant professor, Department of Computer Science and Engineering, Maharaja Surajmal Institute of Technology, New Delhi, India*
*nishtha.jatana@gmail.com*
*[2,3] Student, Department of Information Technology, Maharaja Surajmal Institute of Technology, New Delhi, India*
*adwiteeyaagrawal@gmail.com, kritikasobti92@gmail.com*

**Abstract** – *XSS (cross site scripting) is a web application vulnerability wherein an end point user can pass simple scripts as payloads in un-sanitized input variables. XSS vulnerability has been in existence since long but the current scenario deals with exploiting these vulnerabilities for further attacks, this concept is known as "Post XSS Exploitation" and is focused upon in this paper. This paper presents an in depth study of the dangers of XSS vulnerabilities and vulgarizes its exploitation, it also showcases the remedies of post XSS attacks that can be adopted as a safeguard. Further we exploit a vulnerability and develop a novel module for one of the popular tools of post XSS exploitation. This module can be used to make a SIP (Session Initiation Protocol) call. It has been developed with the intention of being included into the new release of the XSSF framework.*

**Keywords:** *XSS, Post- XSS, Attacks, Remedies.*

## 1. INTRODUCTION

With the advent of dynamic web application development technologies that are used today, the increasing use of the web services have also triggered the rate at which these web applications are becoming vulnerable. Cross Site Scripting (referred to as CSS or more commonly as XSS) is one of the most common code injection attacks. XSS is an injection based vulnerability found in web applications wherein malicious codes are injected as payloads in un-sanitized input variables. When legitimate users access an infected web application, the malicious code is echoed back to the user's browser. The injected code has the ability to read, alter and transmit any classified data accessible by the browser such as cookies, session tokens and so on.

XSS (Cross-site Scripting (XSS) - OWASP) is a vulnerability that is in existence since long. A detailed view of XSS can be obtained from (Shanmugam & Ponnavaikko, 2008). In this paper our prime focus is on Post XSS Exploitation, that is the attacks that can be executed after the XSS vulnerability is found or in conjunction with it. In the paper, we first present the basic features of some useful as well as popular XSS detection and post XSS exploitation tools in Section 2. In Section 3, we describe the categories of XSS which classify XSS attacks as Persistent, Non-persistent or DOM (Document Object Model) based. Section 4 provides description about some of the latest and most fascinating vulnerabilities found on the web and how they can be exploited. In Section 5, we have listed a few remedies which can be implemented on server side as well as on the client side to protect a webpage or application from being XSS vulnerable. In Section 6 we present the concept and the subsequent code of a new module developed for making VOIP (Voice over Internet Protocol) calls using XSS vulnerability. In the succeeding section we conclude our work and provide ideas for future work in this field.

## 2. RELATED WORK

Herein, we present a brief analysis of the various popular frameworks that exist for the detection of XSS vulnerabilities in web applications and their

exploitation (post XSS attacks). They work by injecting payloads and running scripts on the vulnerable page.

### 2.1 Xenotix

Xenotix (Abraham, 2012) is essentially a penetration testing tool used for post XSS exploitation. It has an in-built payload list of more than 450 XSS payloads including even those which can surpass the basic XSS filters that are employed by web developers for protection. It can make use of these payloads in manual as well as auto mode.
It can act as a key logger to capture the keystrokes made by a user when he visits the infected page. The attacker can also download a malevolent executable file on the user's system without him being aware of it. When the user then views the injected page, the java applet client.jar will access the command prompt of his system. With the help of echo command, it will write down some scripts to a Visual basic script file named winconfig.vbs in the temp directory(%temp%) and then the cmd.exe will start winconfig.vbs which will download the malicious executable specified by the attacker in the URL to temp directory and rename it as update.exe and finally it will execute update.exe. Another exploit offered by Xenotix is installation of a reverse shell (Hammer, 2006) at the user's system to gain access to his machine.

Despite being a simple tool, it carries a few concerns with it. The key logging feature is not persistent as it can capture only those strokes made inside the infected page. The drive-through download can run only 16 bit supported executable files. Also the user is prompted every time a drive by download or reverse shell attack is made as the software uses self signed applets to execute the attack.

### 2.2 XSSF

The XSSF Project as elucidated in (Tomes, 2011) (htt1) and (xssf - Cross-Site Scripting Framework - Google Project Hosting) aims to exhibit the potential dangers associated with XSS vulnerabilities. Its basic working includes creation of a communication channel (known as an XSSF tunnel) with the target browser (which has an XSS vulnerability) to perform various attacks. The attacker can execute various attacks, each existing as a separate module. A large number of modules such as file stealer, iphone Skype call, network scanning and many others exist that can be implemented to exploit the vulnerable web applications.

Basically XSSF works by creating a tunnel that involves listing of all the victim id's when a victim arrives on an XSS vulnerable webpage. The attacker then checks the user's browser, searches a suitable exploit, executes it and sends a session to the user. It might then gain access to the user's system. The advanced post XSS attacks that can be performed include creation of an XSSF tunnel that can provide access of the local host of the remote machine to the attacker and allows him to gain its functionality. Also using XSSF which has been integrated with the Metasploit console (Offensive Security Ltd., 2012) one can run any browser based exploit on an XSS vulnerable site to get its meterpreter session thus gaining access to the system. Another feature of this tool is XSSF auto attack wherein various exploits may be added in a queue, each with its own job id and can be executed automatically once the victim visits the vulnerable link provided by the attacker.

While on one hand XSSF offers great many features as a successful tool for Post XSS attacks it does not provide a large number of means for detection of XSS vulnerabilities. Also working with the XSSF framework mandates prior knowledge of Metasploit.

### 2.3 BeEF

Beef (Home · beefproject/beef Wiki · GitHub, 2006) is an acronym for Browser Exploitation Framework. It is a powerful penetration testing tool for the web browser. It uses various client side vectors to assess the actual security postures of a target environment. The framework comprises of various command modules that employ BeEF's simple and powerful API contributing to its effectiveness and efficiency. It allows quick and easy development of user modules.

BeEF hooks one or more web browsers as beachheads for launching of directed command modules and further attacks against the system from within the browser context. The different browsers are likely to lie within different security contexts, and each context may have a set of unique attack vectors. The framework allows the penetration tester to select specific modules (in real-time) to target each browser, and therefore each context.

BeEF framework is a powerful tool that can use XSS vulnerabilities to launch various attacks such as browser fingerprinting (gathering information about the browser), persistence, network fingerprinting, DNS enumeration, Port scanning, and IRC NAT Pinning to name a few.

## 3. TYPES OF XSS

There exist three types of XSS attacks namely

- Non-Persistent or Reflected Vulnerability
- Stored or Persistent vulnerability
- DOM based or Local XSS.

The vulnerability that exists on various web pages or web applications can be classified as any of the above three. Each one of these is explained below with the help of a sequence diagram describing the process of detection and exploitation (post XSS attack) for each type.

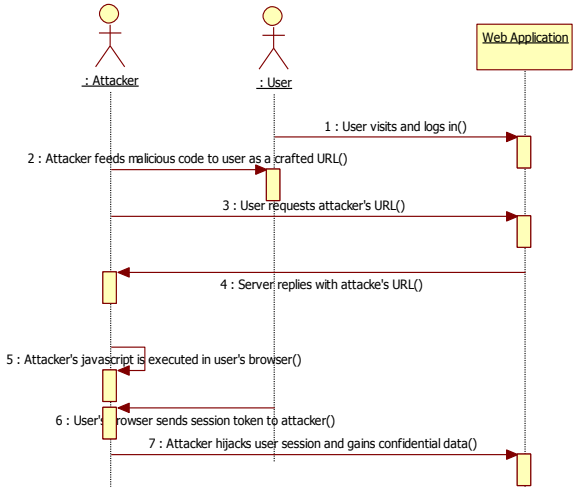


**Fig. 1: Typical scenario of Reflected or non-persistent XXS**

Non-persistent or Reflected attacks (Fig 1) are carried out when data provided by a web client is used right away by server-side scripts to generate a page of results for that user. If user-supplied data is invalidated and is included in the resulting page without HTML encoding, this enables client-side code to be injected into the dynamic page. The injected code can now be reflected off the web server, like in a search result, an error message, or any such response messages that includes part of the input sent to the server as part of the request. Reflected attacks are delivered to users via another route, like in an e-mail message, or may be on some other web server. When a user is tricked into clicking on a malicious link or submitting a specially crafted form, the injected code travels to the vulnerable web server, which reflects the attack back to the victim's browser. The browser then executes the code because it came from a reliable server.

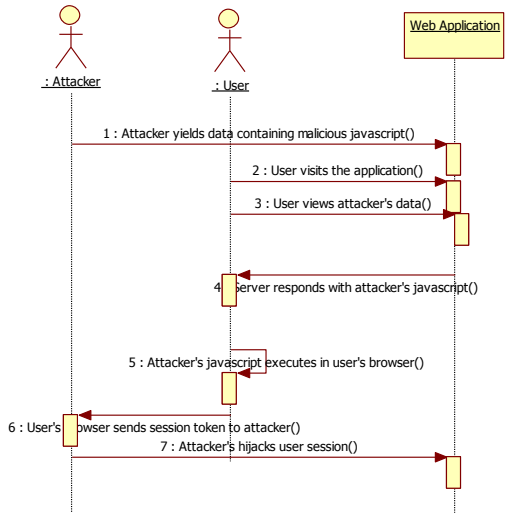


**Fig.2: Typical Scenario of Stored or persistent XXS**

Stored or persistent vulnerability (Fig 2) allows most potent kind of attacks wherein the malicious code is submitted to a website where it is stored for certain duration (in a database, file system, or other location) and subsequently displayed to users in a web page without being encoded using HTML entities. An example of such a situation is with online message boards, where users are permitted to post HTML formatted messages for other users to read.

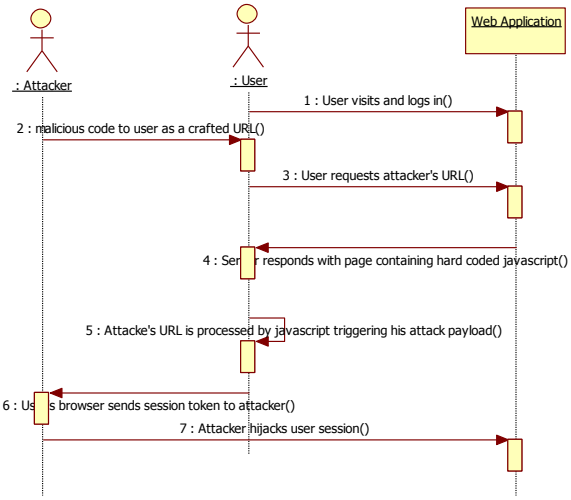


**Fig. 3: Typical Scenario of DOM based XXS**

For DOM (Document Object Model) (Fig 3) -based or Local XSS, the attacker abuses the runtime embedding of attacker data in the client side, from within a page served from the web server.

For example, if a piece of JavaScript accesses a URL request parameter and writes some HTML to its own page, using this information which is not encoded using HTML entities, an XSS hole will possibly be present, since this written data will be re-interpreted by browsers as HTML which could include additional client-side script.

## 4. POST XSS Attacks

### 4.1 Android Data Stealing Vulnerability

The vulnerability explained herein (Cannon, 2013) exists in the Android 2.2 framework. It can be exploited to gain access of the files stored in the SDcard of that android machine. The Android browser doesn't prompt the user while downloading a file, for example a file such as"payload.html" is automatically downloaded to /sdcard/download/ payload.html. A JavaScript can be used to open this file "payload" automatically which causes the browser to render the local file and allows the script to gain access to the local context of the SDcard and hence the files stored within. It can then post the contents of the accessed files back to the infected website. Being a simple exploit involving JavaScript and redirects, it can be used on multiple handsets and various versions of android. But it also has a few limitations such as the name and path of the file to be accessed has to be known beforehand. Since it's not a root exploit it cannot access all the files but only those stored on the SDcard.

### 4.2 Skype's improper URI scheme and embeddable Webkit browser on IOS

This vulnerability as explained in (Kumar, 2011) and (Purviance, 2011) and (iPhones Make Automatic Skype Calls | Security Generation, 2010) exits in the iOS framework. It can be exploited by an attacker to gain access to the user's SQLlite Address Book database and also to place direct calls using Skype. The Skype application developed for iOS uses a locally stored HTML file to display chat messages from other Skype users, but it fails to properly encode the incoming user's "Full Name", allowing an attacker to craft malicious JavaScript code that runs when the victim views the message.

The problem is made more exploitable by the use of embeddable Webkit browser. Also the Skype developers have set the URI scheme for the embedded browser to "*file://*" which allows an attacker to access the file system and read any file that can be read by the iOS application sandbox. Further the lack of need for permission for the third party apps to perform the action defined by URL as well as the URI schemes allow websites to embed an invisible iframe that forces Skype to open (if installed) and call a particular number. The JavaScript for the same being<*iframe src="skype:///1900expensivepremium number?call"*> <*/iframe*>.

### 4.3 HTML5 API for cross domain calls

This vulnerability can be exploited only for Windows systems (Kuppan, 2010). HTML5 has two APIs for making cross domain calls - Cross Origin Requests and WebSockets. By using these, JavaScript can make connections to any IP and to any port (apart from blocked ports), making them an ideal candidate for port scanning. These API's can be exploited to determine if the port being connected to is open, closed or filtered. It does so by the help of two properties: 'ready state' property that indicates the status of the connection at a given time and the 'time duration' for which a specific 'ready State' value lasts.

Thus by observing the difference in behaviour we can determine the nature of the port. Being an application-level scan its success also depends on the nature of the application running on the target ports. When a request is sent to certain type of applications they read the request and remain silent keeping the socket open, probably expecting more input or input in a particular format. If the target is running such an application then its status cannot be determined. Since even closed ports can be identified we can extend this technique to perform network scanning as well as internal IP detection.

### 4.4 HTML5 implementation of AJAX history

This vulnerability is stated in (Kotowicz, 2010). HTML5 has a feature that allows the users to access various web pages and links within a site without changing the URL. It is done with the help of window.history.pushState() function. It was created for AJAX websites for easy modification in the window location bar and manipulating history. It's a great and convenient feature for developers - for example, AJAX apps can now easily support back & forward buttons without resorting to URI fragment identifier (#) tricks. But it can also be exploited for an XSS vulnerable website as it allows the attacker to redirect the user to any link without changing the URL in the address bar.

**4.5 Access to the WScript ActiveX control in Internet Explorer**

The security settings in Internet Explorer grants access to the WScript ActiveX control through scripting languages such as JavaScript and VBScript. The sample application shows how to use the "WScript.shell" ActiveX object in order to interact with the client machine and some more can be obtained from (Spitzen, 2008). With this control one can execute commands similar to a shell prompt without notifying the user. Using Shell one can also create, delete and modify text files through WScript.FileSystemObject. IE7 has put in a new security control called "Access data sources across domain", which now by default is set to prompt the user if they want to allow your script to talk to other 'domains' (it considers file system as a seperate domain) but one can write a script file directly to disk and then execute it, getting around the extra IE7 permissions.

**4.6 File API in HTML5**

This vulnerability is currently being implemented in Webkit (latest Google Chrome) and can be exploited to convert the Google chrome browser into a file server. The File API in HTML5 allows JavaScript to access the file once it is chosen by the user (i.e. before uploading it). Apart from delivering better file uploading experience, it can also be used maliciously to steal your files in XSS attack. With clever styling you can hide *input type=file* control so that the user is unaware that he's going to upload the file. In this case the file chosen by the user in 'Open File' dialog box is the only one that can be accessed. However *input type=file directory* is a splendid feature which allows the user to upload contents of a chosen directory thus giving access of the whole directory to the attacker.

**4.7 XSS MAP**

Google while collecting data for the Google Street View had also collected data of the wireless networks in the vicinity and the MAC address of those routers and then mapped them to the GPS co-ordinates. In this, as elaborated in (Higgins, 2010), an XSS exploit can be use to map the location of a user. The post XSS exploit can retrieve the MAC address of the target's router and then uses Google Maps to GPS co-ordinates. A malicious page you're visiting might perform an XSS exploit and recover your GPS coordinates from the Google Maps. The router and web browser themselves don't contain any geo location/GPS data and neither its IP based Geo location. It works via Router XSS which obtains the MAC address of the router via AJAX. The MAC address is then sent to the attacker who forwards it to Google's Location Based Services which can map the location (approximate GPS co-ordinates) of a user based on his MAC address.

**4.8 NAT PINNING - IRC Over HTTP**

Samy Kamkar explains about this vulnerability (Kamkar, 2010). In this post XSS attack, a web page forces the user's router or firewall, unbeknownst to them, to port forward any port number back to the user's machine. When the victim clicks on an XSS vulnerable URL that has a hidden form connecting to http://attacker.com:6667 (IRC port), he submits the form without knowing. An HTTP connection is created to the (fake) IRC server run by the attacker that simply listens. The victim's router sees an "IRC connection" (even though its client is speaking in HTTP) and an attempt at a 'DCC chat'. Direct Client-to-Client (DCC) is an IRC-related sub-protocol that allows exchange of files and performs non-relayed chats by enabling peers to interconnect with each other using an IRC server for handshaking. DCC chats require opening of a local port on the client to which the remote chatter to connect back. Since the router is blocking all inbound connections, it decides to forward any traffic to the port in the DCC chat back to the victim to allow NAT traversal for the friendly attacker to connect back and chat with him. However the attacker has to specify the port. For example, port 21 (FTP), the router port forwards 21 back to the victim's internal system. The attacker now has a clear route to connect to the victim on port 21 and launch an attack.

**4.9 Browser Exploits**

In these one can exploit the browser application stack and maliciously execute a shell code or open a meterpreter session by using a memory corruption exploit involving XSS. Also the other exploits can return a meterpreter session that does not attack the application stack directly. For example the java self signed applet may be used to maliciously download and execute an exe file.

## 5. NOVEL MODULE FOR POST XSS

In this section we introduce a new module which can be entrenched with cross site scripting framework (XSSF). The module, its working and utility are described below.

### 5.1 The module

Herein we explain the concept behind the creation of a new module for XSSF that exploits an XSS vulnerability to place a VoIP call.

The Elastix PBX (Elastix Freedom to communicate, 2006) is a VoIP based PBX server available for free download and use from *http://www.elastix.org /index.php/en/downloads.html*. The software includes a web interface for configuring the entire PBX functionality and can be accessed from any machine in the network by simply pointing the browser to *https://IP/* (where IP is the address of the server). The version of Elastix PBX under concern for our module is 2.2. On 20th March 2012, an XSS vulnerability was reported by "Martin Tschirsich" on the web interface of the Elastix PBX server.
As per his disclosure the vulnerable file is www/html/recordings/ misc/callme_page.php. This vulnerability allows an SIP client to launch a call to the specified extension. We have developed a module for the cross site scripting framework (XSSF) in Ruby language to exploit this vulnerability and call an extension. When our module is executed the victim is prompted with a call on his screen.

When a user visits an XSS vulnerable link that has already been compromised by the attacker, he gets listed on the victim panel in the XSSF framework. If the victim is a registered SIP client with the Elastix PBX server, the attacker can launch this module and perform the attack.

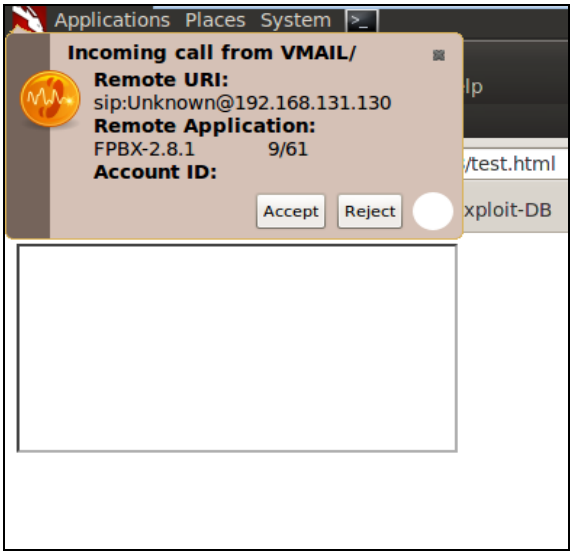


**Fig.4 : Screenshot after module execution**

### 5.2 The module code

#### 5.2.1 The module code: Module initialization (Fig 4)

```
def initialize(info = {})
    super(update_info(info,
        'Name'        => 'Elastix PBX VoIP Call',
        'Description' => 'This Module Uses the XSS
        vulnerability in the Web Interface of Elastix
        PBX server version 2.2.0 to launch a call.
        The call is made only by an authenticated SIP
        client.','Author' => 'Adwiteeya Agrawal
        <adwiteeyaagrawal[at]gmail[dot]com>'
        ))

    register_options(
        [
        OptAddress.new('address',[true, 'IP adress',
        'localhost']),OptInt.new('extension',
        [true, 'Extension', 1000]),
        ], self.class
        )
    end
```

**Fig 4: Module Initialization**

#### 5.2.2 The module code: Script sent to the victim (Fig 5)

```
def on_request_uri(cli, req)

code = %Q{
    document.body.innerHTML = "<iframe
    src=https://#{datastore['address']}
    /recordings/misc/callme_page.php?action=
    c&callmenum=#{datastore['extension']}
    @from-internal/h></iframe>";

    XSSF_POST("Phone call launched",'#{self.name}');
    }

    send_response(cli, code)
end
```

**Fig.5: Script to be sent to the victim**

### 5.3 Steps to reproduce the attack

For this attack it is required that the Elastix 2.2.0 PBX server be running inside the victim's subnet. The Victim must also have an SIP client registered with the PBX server. The attacker is then required to install the XSSF inside the Metasploit framework on his system. This attack basically aims at making an SIP user visit the following link from his browser:

*https://IP_address_of_Elastix/recordings/misc/callme_page.php?action=c&callmenum=Extension@from-internal/h*

This is done to launch a call to the "Extension". The following steps are then performed:

- The attacker sends (directly or indirectly) a link of the vulnerable website to the target with the script to connect back to the XSSF server.
- The target gets listed as the victim in xssf victim list after visiting the link.
- Then "Elastix_PBX_voip_call" module is loaded by the attacker with the command:
  *Use auxiliary/xssf/public/misc/Elastix_PBX_voip_call*
- The attacker enters the IP address of the VoIP server in the "address" option and the extension of the client in "extension" using the following commands:
  *set address IP*
  *set extension EXT*

- The module is launched and on the victim's browser an iframe, whose source is the link to make the call, is created and the victim receives a call from an unknown number.

## 6. Remedies for XSS vulnerabilities

In today's world web applications are gaining widespread importance to provide various online services. But at the same time application vulnerabilities are being discovered and disclosed at an alarmingly fast pace. In the world where web security can be easily compromised, it becomes mandatory to safeguard oneself from such attacks. Various measures can be adopted to avoid being a victim of XSS. These mechanisms (XSS (Cross Site Scripting) Prevention Cheat Sheet - OWASP, 2012) can be implemented either on the server side or client side.

### 6.1 Server Side protection

In order to protect from XSS vulnerabilities, the following measures may be taken by the developer at the server side. The basic concept used here is,
not to trust the input supplied (including the cookies) by the user. The user needs to be verified and validated before allowing access to it. Cookie security as explained by Robert Hafner (Hafner, 2009) protection can be implemented by limiting the domain and path for accepting cookies, setting them as HttpOnly, using SSL and never storing confidential data in cookies. Another safe way out could be to disable the use of client site scripts. Content-Security-Policy headers can also be used to provide security against post XSS exploits. Also, appropriate encoding of HTML control characters, JavaScript, CSS, and URLs should be done to make them harmless before they are displayed in a browser. Some of the filters that can be used to sanitize the user inputs are *filter_sanitize_encoded* (for URL encoding), *htmlentities* (for HTML filtering), *filter_sanitize_magic_quotes* (to apply addslashes()). These filters keep a watch on the user inputs and checks for javascript or HTTP POST in the input and then stop these scripts from being executed. Apart from these measures there are a number of security libraries available for encoding user input such as OWASP Encoding Project available at Google Code, the HTML Purifier or Htmlawed for PHP Anti-XSS Class for .net applications AntiSamy API for .Net or XSS-HTML-Filter for Java.

### 6.2 Endpoint Protection

End users can take steps to prevent becoming a victim of cross-site scripting by installing various browser add-ons. These add-ons keep a watch on various input fields (forms, URLs etc), if a JavaScript or HTTP POST is encountered, it then uses XSS filters to stop those scripts from execution. Examples of such add-ons include NoScript for FireFox; NotScripts for Chrome and Opera whereas Internet Explorer 8 has them as an in-built feature.

### 7. Conclusion and Future Work

In the current era web applications have become an integral part of our lifestyle. But these websites are often vulnerable to certain malicious attacks. This paper has explored one of the rifle vulnerability that exists and its post exploitation. XSS is a pre-dominant code injection attack that can form the basis of many powerful exploits. It can often be combined with other vulnerabilities to execute further critical attacks. In this paper, we have discussed a few such attacks that are prevalent. We have listed a few tools for XSS detection and post XSS exploitation along with their key features. Further we have included a few latest as well as fascinating post XSS attacks and explained the concept behind them. Finally we have developed a new module for post XSS attack which can be embedded with XSSF to make VOIP calls. The coding for the module, along with its basic idea and working has also been included in the paper. In conclusion we have also listed a few protection mechanisms that can be implemented either on the client side or server side to safeguard ourselves from the XSS attacks. As new applications and functions continue to be developed

so would newer vulnerabilities and attacks. We propose to further advance our work by combining such critical vulnerabilities with XSS and creating modules that can be used by other software and frameworks.

## REFRENCES


(n.d.). Retrieved from http://santoshdudhade.blogspot.in/2012/07/xssf-v22-cross-site-scripting-framework.html

Abraham, A. (2012). *Detecting and Exploiting XSS with Xenotix XSS Exploit Framework*. Retrieved from Club Hack 2012: http://www.clubhack.com/2012/event/technical-briefings/detecting-and-exploiting-xss-with-xenotix-xss-exploit-framework/

Cannon, T. (2013, november 23). *Android Data Stealing Vulnerability | thomascannon.net*. Retrieved 2013, from thomascannon.net: http://thomascannon.net/blog/2010/11/android-data-stealing-vulnerability/

*Cross-site Scripting (XSS) - OWASP*. (n.d.). Retrieved February 2013, from www.owasp.org: https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29

*Elastix Freedom to communicate*. (2006). Retrieved from http://www.elastix.org/index.php/en/downloads.html.

Hafner, R. (2009, August). *How to create totally secure cookies.* Retrieved from teamtreehouse.com: http://blog.teamtreehouse.com/how-to-create-totally-secure-cookies

Hammer, R. (2006). *Inside-Out Vulnerabilities, Reverse Shells.* Retrieved from www.sans.org: http://www.sans.org/reading_room/whitepapers/covert/inside-out-vulnerabilities-reverse-shells_1663

Higgins, K. J. (2010). *Hack Pinpoints Victim's Physical Location*. Retrieved from http://www.darkreading.com/security/news/222200541/hack-pinpoints-victim-s-physical-location.html

*Home · beefproject/beef Wiki · GitHub*. (2006). Retrieved from github.com: https://github.com/beefproject/beef/wiki

*iPhones Make Automatic Skype Calls | Security Generation.* (2010, November 10). Retrieved 2013, from www.securitygeneration.com: http://www.securitygeneration.com/tech/iphones-make-automatic-skype-calls/

Kamkar, S. (2010, January). *NAT Pinning: Penetrating routers and firewalls from a web page (forcing router to port forward)*. Retrieved from http://samy.pl/natpin/

Kotowicz, K. (2010, November). *THE WORLD. ACCORDING TO KOTO*. Retrieved from blog.kotowigz.net: http://blog.kotowicz.net/2010/11/xss-track-got-ninja-stealth-skills.html

Kumar, M. (2011, September 20). *iPhone Skype XSS Vulnerability Lets Hackers Steal Phonebook [Video] - Hacking News.* Retrieved 2013, from thehackernews.com: thehackernews.com/2011/09/iphone-skype-xss-vulnerability-lets.html

Kuppan, L. (2010). *Port Scanning with HTML5 and JS-Recon.* Retrieved from http://blog.andlabs.org: http://blog.andlabs.org/2010/12/port-scanning-with-html5-and-js-recon.html

Offensive Security Ltd. (2012). *Introduction - Metasploit Unleashed*. Retrieved from www.offensive-security.com: http://www.offensive-security.com/metasploit-unleashed/Introduction

Purviance, P. (2011, September 19). *XSS in Skype for iOS « Superevr.* Retrieved from superevr.com: https://superevr.com/blog/2011/xss-in-skype-for-ios/

Shanmugam, J., & Ponnavaikko, D. M. (2008). Cross Site Scripting-Latest developments and solutions: A survey. *International Journal of Computational Mathematics, 1*(2). Retrieved from http://www.emis.de/journals/IJOPCM/files/IJOPCM(Vol.1.2.2.S.08).pdf

Spitzen, I. (2008). *Using WScript.shell to interact with the Client machine*. Retrieved from http://www.visualwebgui.com/Developers/KB/tabid/654/article/using_wscript_shell_to_interact_with_the_client_machine_by_mark_reed/Default.aspx

Tomes, T. (2011, July ). *PaulDotcom*. Retrieved from http://pauldotcom.com/2011/07/xssf-expanding-the-attack-surf.html

*XSS (Cross Site Scripting) Prevention Cheat Sheet - OWASP*. (2012). Retrieved February 2013, from www.owasp.org: https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

*xssf - Cross-Site Scripting Framework - Google Project Hosting*. (n.d.). Retrieved February 2013, from code.google.com: http://code.google.com/p/xssf/