I. J. Computer Network and Information Security, 2012, 1, 31-37
Published Online February 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2012.01.04

CSRF Vulnerabilities and Defensive Techniques

Rupali D. Kombade, Dr. B.B. Meshram,
Veermata Jijabai Technological Institute, Matunga, Mumbai.
rupalikombade@gmail.com, bbmeshram@vjti.ac.in

***Abstract*- Web applications are now part of day to day life due to their user friendly environment as well as advancement of technology to provide internet facilities, but these web applications brought lot of threats with them and these threats are continuously growing, one of the these threat is Cross Site Request Forgery(CSRF). CSRF attack is immerged as serious threat to web applications which based on the vulnerabilities present in the normal request response pattern of HTTP protocol. It is difficult to detect and hence it is present in most of the existing web applications. CSRF attack occurs when a malicious web site causes a user's web browser to perform an unwanted action on a trusted site. It is listed in OWASP's top ten Web Application attacks list. In this survey paper we will study CSRF attack, CSRF vulnerabilities and its defensive measures. We have compared various defense mechanisms to analyse the best defense mechanism. This study will help us to build strong and robust CSRF protection mechanism.**

***Index Terms*- Web Application, Vulnerability, Attacks, Defensive measures, Cross-Site Request forgery.**

I. INTRODUCTION

Use of internet tremendously increasing with technology, it is now used for each possible function that can perform online, web applications playing important role to provide these functions. Web applications are become part of life of human beings. Some of these are reducing their efforts like (reservation systems, online banking etc...) and some are entertaining and connecting them socially (facebook, myspace etc...). But with all these facilities they have also bring some problems i.e. web application attacks. Web application attacks create insecure environment for web application's users. It can be result in huge loss. Web applications are a major target for hackers. According to the study, websites experience an average of 27 attacks per hour or about once every two minutes. However, 27 attacks per hour is only an average. When sites come under automated attack, the target can experience up to 25,000 attacks per hour or 7 per second. [1] OWASP (open source web application security project) has listed the top ten web application attacks of 2010 as below. [2]

- Injection
- Cross site scripting
- Broken authentication and session management
- Insecure direct object reference
- Cross site request forgery
- Security misconfiguration
- Failure to restrict URL access
- Unvalidated redirects and forwards
- Insecure cryptographic storage
- Insufficient transport layer protection

In this paper we are concentrating on Cross Site Request Forgery Attack (CSRF). This attack is less known to developers, some considers it same as XSS and some considers that XSS mitigation techniques will work for this attack. But it is different from XSS and its mitigation techniques need something extra efforts than XSS defensive measures. CSRF attacks have been known as "sleeping giant" of web-based vulnerabilities [3], because many sites on the Internet fail to protect against them and they have been largely ignored by the web development and security communities. Cross-Site Request Forgery Attacks are also known as Cross-Site Reference Forgery, XSRF, session Riding and Confused Deputy attacks. [4]

This survey paper is divided into following sections. Section II describes how CSRF attack is carried out. Section III describes various CSRF vulnerabilities present in web applications; Section IV contains available CSRF Mitigation techniques, section V compares the CSRF mitigation techniques discussed in section IV, section VI concludes this survey paper, and then referenced material and author's introductions are listed.

II. OVERVIEW OF CSRF ATTACK

CSRF is an attack which forces an end user to execute unwanted actions on a web application, in which he/she is currently authenticated [2]. It takes the advantage of HTTP protocols functionality to send

session cookie for each request to server once user authenticate successfully, which helps server to confirm that the request is coming from authenticated user. CSRF attacker first study the request pattern i.e. type of request (GET request or POST request), parameters names, type of parameters values etc.. Once studied the request's URL pattern deeply, he embed this URL in html tags of web pages or emails. Then attacker forces the authenticated user to execute this request. As user is authenticated browser automatically sends session cookie value with this request, server accepts this request and execute it. Figure 1 shows the scenario of CSRF attack.

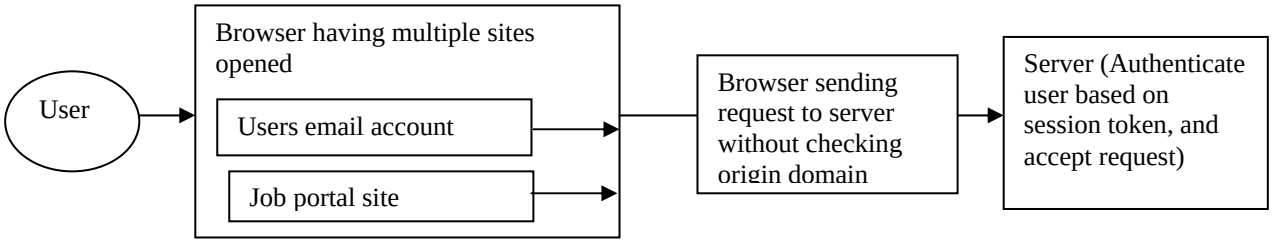

Figure 1 CSRF Attack scenario

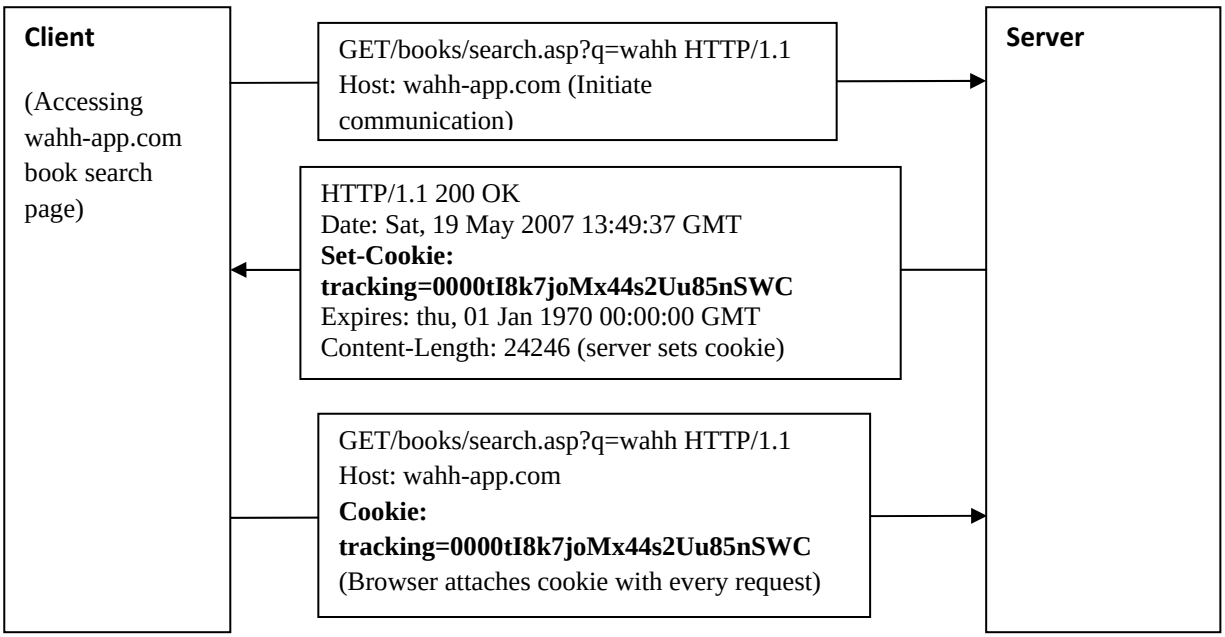

Figure 2 Working of Http request and response

Following example shows how CSRF attacker uses 'img' or 'script' tag to send request to server without knowledge of user. Consider, the user logged in to his email account and found email saying that 'check jobs matching to your profile', If user open this email and follow the link given there, in next tab jobs website page will get open. If attacker also has email account in same site as user and he know how 'change password' functionality works, he can put the corresponding action URL in some HTML tag. Consider attacker has added this URL in image tag as given below.

<img src=www.examplemail.com/changepass. php?newpass=somevalue>

This will not affect the GUI of jobs website page and URL embed in 'src' tag will get called as page will be loaded. As user is already logged in, browser will automatically send session id to server while sending this request. Hence server will accept the request as valid request. This way without knowledge of user the CSRF attack is carried out.

Effects of CSRF attacks may differ based on the vulnerabilities exploited and privilege of the user exploited. A successful CSRF exploit can compromise end user data and operations when it targets a normal user, for example transfer of amount from user account to attacker account. If the targeted end user is the administrator account, a CSRF attack can compromise the entire web application. It's not just your public Web applications that are at risk, CSRF tactics can be used to attack servers behind corporate firewalls. Following image tag shows such example. [14]

<img
src="http://intranet/admin/purgedatabase?rowslike=%2A
&confirm=yes">

If the attacker knows enough to make a URL and can get you to open a message, that's all it takes. Such effects of CSRF can result in huge loss of web application's user as well as owner; that's why it is very important to stop this attack. CSRF can affects web devices same as web sites. For Example in January 2008 attackers sent out e-mail having request embedded in image tag having URI 192.168.1.1 which is the default IP address of web enabled Linux based router, if web interface is vulnerable to CSRF and authentication is also vulnerable then on opening email image tag get loaded and shell commands can be execute on router of email account holder. [9] Following URL shows how shell command can be executed on router.

http://192.168.1.1/cgi-bin/;reboot

CSRF vulnerabilities are present in so many existing websites; some of these are described in [6]. Hence in this paper we are studying CSRF vulnerabilities as well as mitigation techniques which help us to build strong and robust protection mechanism against CSRF.

III. CSRF VULNERABILITIES

Attackers are not required to do extra efforts to carry out attack because the way web handles the web application traffic between client and server allowing attacker to carry out attacks. So many flaws are there which helps attackers and make their job easy to satisfy their requirement. In this section we will take review of such vulnerabilities presents in web applications.

a. HTTP session handling mechanism

Number of website required user authentication while accessing it, which is most important requirement to carry out user specific tasks as well as to provide privacy to user's data and information. To simplify this requirement HTTP protocol provides facility of session and cookie, which allow web server to differentiate the request coming from different users. Once user gets authenticated, this session cookie information gets passed in every request from server to client and vice versa. Following code shows the format of HTTP request and response; also figure 2 shows how web request and response are carried out.

HTTP request

GET /books/search.asp?q=wahh HTTP/1.1
Accept: image/gif, image/xxbitmap, image/jpeg, image/pjpeg, application/xshockwaveflash, application/vnd.msexcel, application/vnd.mspowerpoint, application/msword, */*
Referer: http://wahh-app.com/books/default.asp
Accept-Language: en-gb,en-us;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: wahh-app.com
Cookie:
JSESSIONID=0000tI8rk7joMx44S2Uu85nSWc_:vsnlc502

HTTP response

HTTP/1.1 200 OK
Date: Sat, 19 May 2007 13:49:37 GMT
Server: IBM_HTTP_SERVER/1.3.26.2 Apache/1.3.26 (Unix)
Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Language: en-US
Content-Length: 24246
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
...

So whenever server gets request having valid session information it executes that request without bothering the origin of the request. Hence when CSRF attacker sends request to sever through browser by embedding it in exploited site, it executes on server successfully and no one can detect that request has come from other domain and it is invalid.

b. HTML tags

CSRF attackers embed the request they want to execute in HTML tags due to which attack become invisible and while loading particular page (with page, it loads the all elements present on page), request gets executed. Also sometime it is embedded into the tags where it will get execute only if user click on that tag's user interface like 'href tag'. In this case attacker forces the user to click on such tags by showing text which attracts user e.g. "50% discount on jwelleries" etc. There are so many tags present in HTML which can send request to server, but each and every tag is made for particular type of request like for image file, JavaScript file etc.. HTML does not check the tag source property contains the valid URL or not, and CSRF attackers take advantage of this vulnerability. Table1shows the list of HTML tags that can be used by attacker to carry out CSRF. We have already seen example of exploiting HTML tag to carry out CSRF attack in section II.

Table1- Various html tags used to carry out CSRF attack

HTML tag	Exploited format of HTML tag
body	<body { background: url('attack_request')}> <body onload="attack_request">
img	
input	<input type = "image" src = " attack_request" alt = "Submit" />
link	<link rel = "stylesheet" type = "text/css" href = "attack_request" />
script	<script type = "text/javascript" src ="attack_request" > </script>
table	<table background = "attack_request" >
td	<td background = "attack_request">
th	<th background = "attack_request">
iframe	<iframe src="attack_request">

This problem might be even worse, since in integrated mail/browser environments simply displaying an email message containing the image would result in the execution of the request to the web application with the associated browser cookie. [2] And in case if anyone put check on the 'src' field of tag to contain valid URL specific to particular tag then this may be obfuscated further, by referencing seemingly valid image URLs such as

<img src="h ttps:// [attackers url]/picture.gif" width="0" height="0">

Where [attacker] is a site controlled by the attacker and by utilizing a redirect mechanism on http://[attacker]/picture.gif to http://[third party]/action.

c. Browser's view Source option

There are various different ways by which attacker get knowledge of functionality used by web application, which helps attacker to generate valid request. Attacker can himself log on the website and check the whole functionality, also information about working of forms on the web pages can be easily available by facility provided by web browser using option 'View Source', Which shows all the information of the fields present on forms, validation for each field can be accessed by using JavaScript files and much more information attacker can collect. If web application using extra session variable on each request to protect application from CSRF and if that session information is saved in hidden field, using view source option attacker can get the logic used to generate this session field unless until it is not strongly generated random token.

d. GET and POST method of form submission

Information in the form fields sends to the server by using two methods GET and POST, where GET method generate a request which contain all the information itself in request and it is also visible to the user, so attacker can make use of this easily available information to generate valid request. It was suggested that to use POST instead of GET method to stop this vulnerability**.** But POST method is also not helped to protect web applications from CSRF attack. Once attacker get all form fields he can embed these fields into his web page, which he is going to force the victim to open and can put the JavaScript function which allow form to submit on onload event. Following example describe this scenario.

Where {element} = HTML element

<{element}
onload=javascript:document.myform.submitO >
<form name="myform" method="POST" action="{vulnerable site}">
<input name="variablel" value="attackl" >
<input name="variable2" value="attack2" >
<input name="variable3" value="attack3" >
</form>

Here we can see that form is submitting directly on onload event, without knowledge of user.

e. Input Validation Error

CSRF can be divided into two forms stored and reflected [4], Stored CSRF is when the attacker gets the CSRF to be executed within the domain of the targeted websites, while the reflected CSRF is when the attack is triggered from a different domain. In case of Stored CSRF we can give example of social networking site where user can add a post which contain malicious request which can perform some malicious action on that site. In this case attacker uses the vulnerabilities present in input validation functionality. While processing input data submitted by user, its format should be well specified and well checked. If this validation functionality is weak, it can allow attackers malicious content to get into the system, which will help them to carry out CSRF.

f. Handling of data through Javascript

Javascript is also used to transfer the data in application using AJAX. AJAX uses XMLhttpRequest to communicate to action to server and server returns lightweight response containing data in JSON format. This returned response is then processed and used by client. Such use of Javascript make possible for malicious website to exploit same origin policy handling of javascript and gain access to data generated by other website. Malicious site can perform this using two ways, overriding the default array constructor and by implementing a suitable callback function. [10]

Number of times the data return by XMLhttprequest contains a serialized array, malicious website can override the default constructor for this array or object to gain access to the data. This can be done by retrieving script tag's target and executing it. Such vulnerability was discovered within GMAIL functionality by Jeremiah Grossman in 2006. Sometime javascript does not only return data but also invokes callback function on returned data. For example:

```
showContacts(
    [
    [ 'Jeff', '1741024918', 'ginger@microsoft.com' ],
    [ 'C Gillingham', '3885193114',
    'c2004@symantec.com' ],
    [ 'Mike Kemp', '8041148671',
    'fkwitt@layerone.com' ],
    [ 'Wade A', '5078782513',
    kingofbeef@ngssoftware.com' ]]);
```

This can be exploited by simply implementing the showcontacts function and include the target script. [13] For example:

```
<script>
function showContacts(a) {
alert(a);
}
</script>
<script
src="http://wahhapp.com/private/contacts.json?callb
ack=showContacts"></script>
```

IV. CSRF DEFENSIVE MECHANISM

As CSRF become popular various defensive measures against it were suggested, but none of these is able to defence against CSRF completely. But these helps to minimised the risk of CSRF up to certain extent. In this section we are going to review such defensive measures which will help us to build more robust technique to mitigate CSRF.

a. Checking Referer Header

HTTP request contain different parameters, one of these parameters contain the URL of site from which request originates, that parameter name is 'Referer'. This parameter can be used by browser to check requests domain on client side before forwarding request to server. So web developers check Referer header to protect applications from CSRF. This can be applied in case of critical operation like password change, amount transfer, purchasing items and changing user privileges etc. This will allow only same domain request to execute.

b. Custom Header

Custom headers, those prefixed with X-, are sent to the client together with the default HTTP header. One important property of these headers is that they cannot be sent cross domain [9]. With the help of custom headers we can identify that the request has come from same domain, as browser prevent to send custom header from one site to another. To use this mechanism web application must issue all state modifying requests using XMLHttpRequest and attach the custom header. The state modifying request having no custom header will considered as invalid request. [10] For example request with custom header will look like as below:

```
GET /auth/update_profile.cgi?email=victim@social.site
HTTP/1.1
Host: social.site
X-CSRF: 1
```

Here X-CSRF represents that this request consist of custom header and it also confirm that the request has come from same domain. Browser should not forward custom header between domains. But vulnerability arise due to exception to security rules, in this case Plug-ins like Flash or Silverlight might allow request to include any number or type of headers regardless of the origin and destination of the request.[9] This vulnerability could expose users to CSRF even with application of custom header.

c. Client side tool with white listing

As we know there are so many websites which need cross domain operation to perform, in such cases 'Referer' mechanism cannot be used. Hence it was suggested to use a tool which maintains white listing of websites having cross domain operations. This implement a client-side defence measure that previews the HTML code before each page load and detects potential CSRF attack. The detector would first find all form tags and check the "action" attribute of the "form" tags for deep linking. If such forms are found, the CSRF detector will prompt the user 'if they want to add the pairing of the URL of the website the code is located on and the URL of the form action to a white list'. IF user will add that URL to whitelist then whitelist get updated

and this updated whitelist will be used further. This tool can be installed as an extension to browser. [5]

d. Limiting the lifetime of authentication cookies:

CSRF attacks can be minimised by limiting the lifetime of cookies to a short period of time. If user will open the other website and started surfing on it, it will cause cookies of previous site to expire and after a short period of time and user have to login again for any action he want to perform. If the attacker will try to send any HTTP request, he will not successful as server rejects the request, because it will not get session information due to cookies expiration. [6]

e. Anti CSRF

It is a library developed in C# for ASP.NET developers to guard themselves from CSRF attacks. It is HTTP module which can be added to web application to protect application against CSRF. This module itself takes care of token generation and checking it on every page of web site, assuming it inherits from System.Web.Page and contains ASP.NET form [11].

This library need to be added as a reference to web application and related settings has to be done in web configuration file. Normal way of adding CSRF token to the ASP.NET application is to use ViewState in combination with ViewStateUserKey. This requires ViewState to be enabled and as well as session to be enabled because sessionid will be used as a unique key to identify user. AntiCSRF module works without these requirement and hence provide more independent environment. AntiCSRF requires Cookies to be enabled on Users browser and cookies used on browser get cleared when browser will get closed. It uses hidden field to carry out CSRF token. [12]

f. CSRF detector

Cross Site Request Forgery (CSRF) can be carried out using XSS attacks and maximum protection mechanism suggested against CSRF are depend on cross origin policies and that also not completely protects web applications from CSRF. CSRF detector detects CSRF attacks with the notion of visibility and content checking of suspected requests. The idea is to intercept a suspected request containing parameters and values and relate them with one of the visible forms present in an open window. If there is an exact match, the suspected request is modified to make it benign, then it is launched to the remote website to identify the content type, this content type is then matched with the expected content type. Any mismatch between request attribute values or content type results in a warning. [7] This approach does not rely on cross-origin policy or server side program states. Moreover, it does not require storing URLs or tokens to be matched at a later stage for attack detection. This can be implemented as a Firefox plug-in. Once it detected the CSRF attack we can stop that request or blacklist that particular site if it is cross site request. Hence this detector will be useful to prevent CSRF attack.

V. COMPARISION

As we have discussed various defensive mechanisms against CSRF in previous section, we will see which is more useful. Very first we have seen is checking Referer header, this method will help in very few cases because most of the sites don't use Referer header for security purpose. This mechanism is useful for reflected CSRF only i.e. CSRF carried out from other domains; this drawback is applicable to white listing defensive measure also. Custom header is also used to detect only reflected CSRF as well as its required to use XMLHTTPrequest each time we need to protect system from CSRF, which make protection mechanism dependent on particular technology. Whereas CSRF detector allows us to detect both reflected and stored CSRF attacks. Limiting the lifetime of Authenticated cookies is needed to be implemented on server side, i.e. application developer can implement this mechanism into their application. This method can minimize the CSRF attacks but cannot provide complete protection. CSRF Guard is technique provided by OWASP which need to implement with application code and it can well protect the system. AntiCSRF working same as CSRF Guard and it is specifically used for ASP.NET applications. Both these implementations are vulnerable to session hijacking attacks and social engineering through different ways to capture the session token [8] and both of these are technology specific. In case of CSRF detector, it checks content type of response with expected content type of request to decide suspiciousness of request. This may produce wrong result as some servers may return incorrect content type or some may have not sending content type at all.

VI. CONCLUSION

In this survey paper we discussed CSRF vulnerabilities which will help to understand CSRF attack scenario and causes behind it. Also we discussed various CSRF defensive techniques suggested yet. In section V we compared all the techniques we discussed as per their ability to protect web application against CSRF attacks. As per analysis it is found that CSRF guard and CSRF detector are most powerful techniques but still cannot provide full protection, they can only minimise the CSRF attacks. Hence complete protection against CSRF is not available and our discussed techniques need more improvement so that they can completely protect the application. Robust and strong protection mechanism against CSRF is needed to protect the web applications.

REFERENCES

[1]. Imperva's Web application Attack Report July 2011 Edition #1, www.imperva.com

[2]. OWASP. https://www.owasp.org/index.php/CSRF, Cross-Site Request Forgery, Testing for CSRF (OWASP-SM-005)

[3]. Grossman, "Cross Site Request Forgery 'The Sleeping Giant of Website Vulnerabilities'", in *RSA Conference, San Francisco,* April 2008.

[4]. Xiaoli Lin, Pavol Zavarsky, Ron Ruhl, Dale Lindskog, "Threat Modeling for CSRF Attacks", International Conference on Computational Science and Engineering, 2009

[5]. Tatiana Alexenko, Mark Jenne, Suman Deb Roy, Wenjun Zeng , "Cross-Site Request Forgery: Attack and Defense", IEEE CCNC 2010

[6]. Mohd. Shadab Siddiqui, Deepanker Verma, "Cross Site Request Forgery: A common web application weakness", 2011 IEEE

[7]. Hossain Shahriar and Mohammad Zulkernine "**Client-Side Detection of Cross-Site Request Forgery Attacks",** 21st International Symposium on Software Reliability Engineering, **2010** IEEE

[8]. Boyan Chen, Pavol Zavarsky, Ron Ruhl and Dale Lindskog, "**A Study of the Effectiveness of CSRF Guard", 2011** IEEE

[9]. "Seven Deadliest Web Application Attacks", Mike Sharma.

[10]. Adam Barth, Collin Jackson, John C. Mitchell "Robust Defenses for Cross-Site Request Forgery", CCS'08, October 27–31, 2008, Alexandria, Virginia, USA.

[11]. http://anticsrf.codeplex.com/ AntiCSRF - A Cross Site Request Forgery (CSRF) module for ASP.NET

[12]. http://netappsec.blogspot.in/2011/05/anti-csrf-viewstate.html "Anti-CSRF & ViewState"

[13]. "The Web Application Hackers Handbook", discovering and exploiting security flaw", Dafydd Stuttard, Marcus Pinto

[14]. Peter W "Cross Site Request Forgeries" – http://www.tux.org/~peterw/csrf.txt

Dr. B. B. Meshram is working as Professor in Computer Technology Dept., VJTI, Matunga, Mumbai, INDIA. He is Ph.D. in Computer Engineering and has published international journal is 25, National journal is 1, international conference is 70 and national conference 39 papers to his credit. He has taught various subjects such as Object Oriented Software Engg., Network Security, Advanced Databases, Advanced Computer Network (TCP/IP), Data warehouse and Data mining, etc at Post Graduate Level. He has guided several projects at graduate and post graduate level. He is the life member of CSI and Institute of Engineers, etc.

Rupali Kombade received her B.E. Degree in Computer Engineering from RTM Nagpur university. She has worked as PHP and .NET Web developer. She is pursuing M.Tech degree in Network Infrastructure Management System from VJTI, Matunga, Mumbai, INDIA. Her research interest includes web development, web security, web application attacks and its defense.

