Web Secure Software Lifecycle Model

D. R. Ingle, Dr. B.B. Meshram

Abstract-The traditional approach of developing software does not consider security as a prime factor for the development of any software. It is considered as one of the features of the developed system. Since in the recent years number of cyber crimes have increased, there is a need to redesign the software development process and introduce security at each and every phase of software development lifecycle. To induce security at every phase we need to identify the vulnerabilities at each phase. Hence this paper summarizes the survey conducted on vulnerabilities at different phases, the threats and tentative attacks due to the corresponding vulnerabilities. We have also conducted survey on patches available for these threats and hence we have proposed a secure software development lifecycle. We have considered web as the application to identify the vulnerabilities and we have also implemented the security patches on a case study.

Key words-*Vulnerability, security, requirement phase, design phase, implementation phase, testing phase .maintenance phase*

1. INTRODUCTION

Security is the prime concern of any project. Unfortunately in the early days software projects were developed with requirement as the primary concern[1]. The researchers identified the reason of failure of excellently coded software developed as per the basic requirement as the lack of security measures at the early stages of development of software[2]. Traditionally software project is developed using the waterfall lifecycle model which consists of the stages such as requirement engineering, design, coding, integration and deployment and maintenance. Implementing security at the requirement engineering phase as well as at the other phases also would always improve the integrity of the software [3].

- *D.R. Ingle is currently pursuing PhD program in computer engineering in Amravati University, India, PH-09702777927. E-mail: dringleus@yahoo.com*
- *Dr. B.B. Meshram is currently working as professor and HOD of Computer technology in University of Mumbai, India PH-01123456789. E-mail: bbmeshram@vjti.org.in*

Security is one of the properties of the software. It is not the feature of any software[4]. Application security always differs from the software security[5]. The user authentication, pin verification, intrusion detection system, firewalls are the application security which is implemented after the software is developed and deployed[6]. The application security is breached intentionally by an intruder violating the security measures[7]. Software security is always breached due vulnerabilities[8] in software development. Weakness or faults in a system or protection mechanism that expose information to attack or damage are known as vulnerabilities[9]. They can range from flaw in software package, to an unprotected system port, or an unlocked door. Vulnerabilities have been examined, documented and published are referred to as well known vulnerabilities[10]. As an instance the standard port for SMTP is port no.8080[11]. If SMTP is designed to work on port 8080 all the mails transferred via. this mail server can be accessed easily[12]. The need of software security is to identify the vulnerabilities and develop the flawless software[13].

Different researches in the field of software security are based on either analysis of security at the phases of software development lifecycle or implementing security at particular stage14]. This is overcome in SOSDLC where analysis of security is done only after it is implemented at each phase.

The rest of the paper is organized as follows. Section 2 deals with background work Section 3 deals with survey on vulnerabilities at different phases of SDLC and hence implementing security. Section 4 deals with implementation of SOSDLC Section 5 gives the conclusion.

2. BACKGROUND WORK

Developing a secure application is an important job of a software developer. In paper [15] three basic motivating points are considered as need of implementing security. These points emphasize on creating awareness about security, need for security team and also need for different methods and tools for implementing security. According to[16], software engineers generally do not use security failure data, particularly attack data, to improve the security and survivability of the systems that they develop. Providing up-to-date information about security problems to developers
And informing them about practices known to reduce security problems are important steps toward securing software. [17] Discuss about security teams investigate vulnerabilities and different security issues and also identify methods and tools to be deployed in software development process to improve security. Thee points do not take care about implementing security at each phase which is considered in SOSDLC.

To develop the framework of our proposed method we identified the following points to be considered.

2.1 Secure Software Development Team

This team identifies the common mistakes done by the developers and hence identify the flaw in the application. The team can be split into five groups to remove vulnerabilities at five different phases and hence implement different secure tools and methods . They also should work on analysis of security at different phases.

2.2 Identifying Secure Tools and Methods

The identification of vulnerabilities at different phases will help in identifying secure tools and methods. There are different ways of identifying vulnerabilities but this should be related to the application which is being developed. Rather than using normal UML diagrams to understand the design of system, different tools like UMLsec, ASMLses or STATLetc should be used

2.3 Testing integration of secure phases

Many researches are going on in the field of security in SDLC. Most of them recommend implementing security at particular phase. Though security was implemented at any stage different test cases can only give idea about the correctness of the security. The security implemented at particular phase might be correct but after integrating different phases the overall purpose of software development should not be changed and also the overall security of the application should not be violated

3. Survey on Vulnerabilities At Different Phases Of SDLC And Hence Implementing Security

Vulnerabilities and their patches

3.1Requirement Phase: We have identified the vulnerabilities at requirement phase based on [18].

3.1.1 Vulnerabilities

1. Non identification and non verification of users and client applications.
2. Unmanaged authorization of access to data and services for users and client applications.
3. No availability of mechanisms for detecting intrusions by unauthorized persons and client applications.
4. No mechanisms to ensure that unauthorized malicious programs (e.g., viruses) do not infect the application or component.
5. Non availability of mechanism to avoid intentional corruption of communications and data .
6. Non availability of mechanism for repudiation of interactions of parties with the application or component later.
7. Non availability of mechanism to ensure that confidential communications and maintain privacy of data.
8. Non availability of security personnel to audit the status and usage of the security mechanisms.
9. Non availability of recovery methods
10. Non availability of physical security of centres and their components and personnel protection against destruction, damage, theft, or surreptitious replacement (e.g., due to vandalism, sabotage, or terrorism).
11. Non availability of mechanism to avoid unintentionally disrupts of the security mechanisms of application, component, or centre due to maintenance.

3.1.2 Patches

The above mentioned vulnerabilities can be classified as the following points mentioned in [10]. Considering these points at the requirement phase of software development will help in developing secure software.

• **Identification Requirements** An identification requirement is any security requirement that specifies the extent to which a business, application, component, or centre shall identify its externals (e.g., human actors and external applications) before interacting with them.

• **Authentication Requirements**
An authentication requirement is any security requirement that specifies the extent to which a business, application, component, or center shall verify the identity of its externals (e.g., human actors and external applications) before interacting with them.

• **Authorization Requirements**
An authorization requirement is any security requirement that specifies the access and usage privileges of authenticated users and client applications.

• **Immunity Requirements** An immunity requirement is any security requirement that specifies the extent to which an application or component shall protect itself from infection by unauthorized undesirable programs (e.g., computer viruses, worms, and Trojan horses).

• **Integrity Requirements** An integrity requirement is any security requirement that specifies the extent to which an application or component shall ensure that its data and communications are not intentionally corrupted via unauthorized creation, modification, or deletion.

• **Intrusion Detection Requirements** An intrusion detection requirement is any security requirement that specifies the extent to which an application or component shall detect and record attempted access or modification by unauthorized individuals.

•**Nonrepudiation Requirements** A nonrepudiation requirement is any security requirement that specifies the extent to which a business, application, or component shall prevent a party to one of its interactions (e.g., message, transaction) from denying having participated in all or part of
the interaction.

• **Privacy Requirements** A privacy requirement is any security requirement that specifies the extent to which a business, application, component, or centre shall keep its sensitive data and communications private from unauthorized individuals and programs.

• **Security Auditing Requirements** A security auditing requirement is any security requirement that specifies the extent to which a business, application, component, or centre shall enable security personnel to audit the status and use of its security mechanisms.

• **Survivability Requirements** A survivability requirement is any security requirement that specifies the extent to which an application or centre shall survive the intentional loss or destruction of a component.

• **Physical Protection Requirements** A physical protection requirement is any security requirement that specifies the extent to which an application or centre shall protect itself from physical assault.

• **System Maintenance Security Requirements** A system maintenance security requirement is any security requirement that specifies the extent to which an application, component, or centre shall prevent *authorized* modifications (e.g., defect fixes, enhancements, updates) from accidentally defeating its security mechanisms.

3.1.3 Tools: Different UML diagrams can be used to model the security at the requirement phase. A framework for representing security requirements has been designed by Charles B. Haley, Robin Laney etal using different UML tools such as class diagram and activity diagrams. Hence using different UML diagrams we can model the security at requirement phase.

Case Study: We have implemented these patches on a case study on online paper conference system. The Requirement analysis of this case study focuses on the different categories of users, hereafter roles, which can interact with the application.

The main roles involved in this application are:

- *Authors* submit papers and browse all relevant information on their papers.
- *PC members* submit reviews, browse all papers, and discuss paper acceptance.
- *Conference chair* assigns papers to PC members and defines the program.

The use case diagram, misuse case and attack tree
Have been implemented in the below figures.

Use Case Diagram:

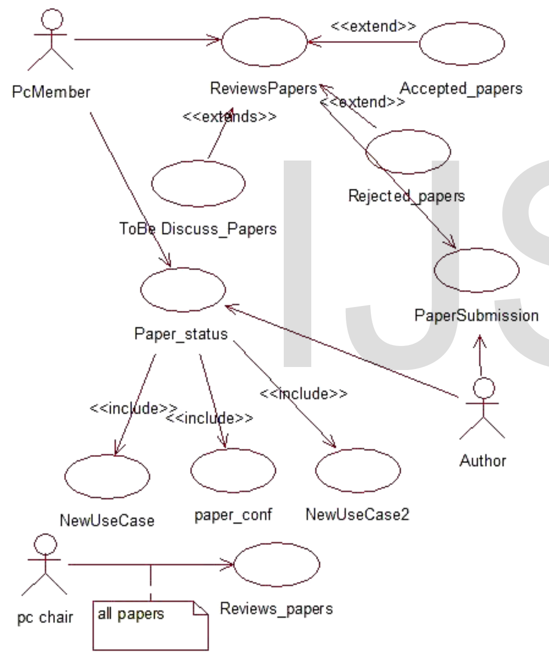


Figure 5 Use Case diagram of Online Paper Conference

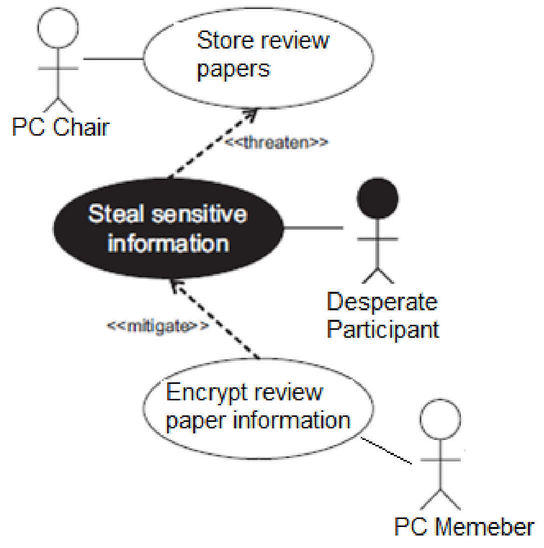


Figure 6 MisUse Case diagram of Online Paper Conference

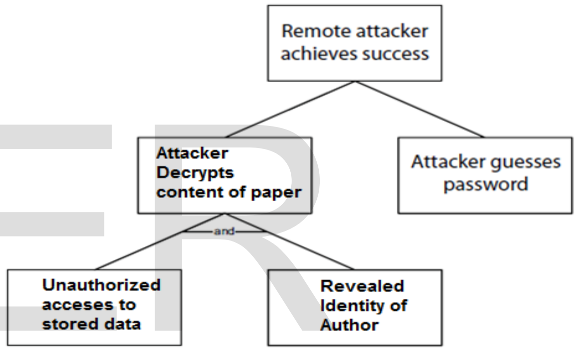

Remote attacker achieves success
Attacker Decrypts content of paper
Attacker guesses password
and
Unauthorized acceses to stored data
Revealed Identity of Author

Figure 7 Attcak Tree1

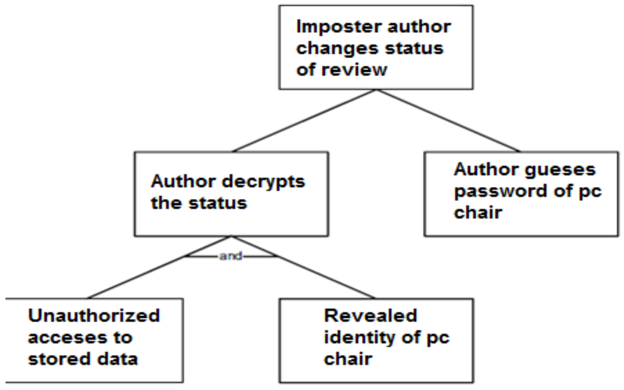

Imposter author changes status of review
Author decrypts the status
Author gueses password of pc chair
and
Unauthorized acceses to stored data
Revealed identity of pc chair

Figure 8 Attcak Tree2

To implement security at this phase avoiding the above vulnerabilities we have considered four levels of security at this phase with the help of [19].
1. User level security 2.Application level security 3.Funtional level security 4.Security for business continuity. These levels are implemented using block diagram as shown in figure 1,2,3,4.

1. User level security: The user level security considers the security at the user levels. Hence as shown in the below diagram under identification requirement the personal details and professional details of the user are verified. The authorization requirement identifies whether the user is authorized edit, add and delete. Based on the result obtained from identification requirement and authorization requirement authentication requirement identifies whether the user has author authentication PC chair authentication or PC Member authentication.

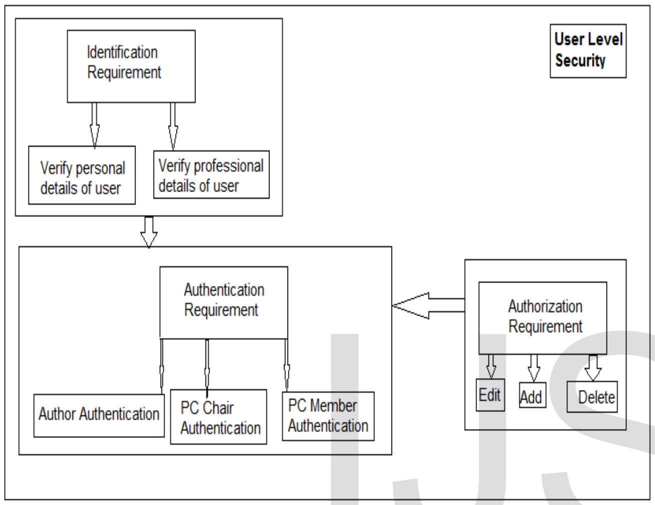

User Level Security
Identification Requirement
Verify personal details of user
Verify professional details of user
Authentication Requirement
Author Authentication
PC Chair Authentication
PC Member Authentication
Authorization Requirement
Edit
Add
Delete

Figure 9 User Level Security at Requirement Phase

2. Application Level Security: Application level security identifies the security of the web application. As shown in the diagram below the immunity requirement identifies the viruses, Trojans, worms which may attack the web application, the intrusion detection requirement identifies the failed authentication, failed identification and failed authorization and the integrity requirement identifies unauthorized creation unauthorized modification and unauthorized deletion.

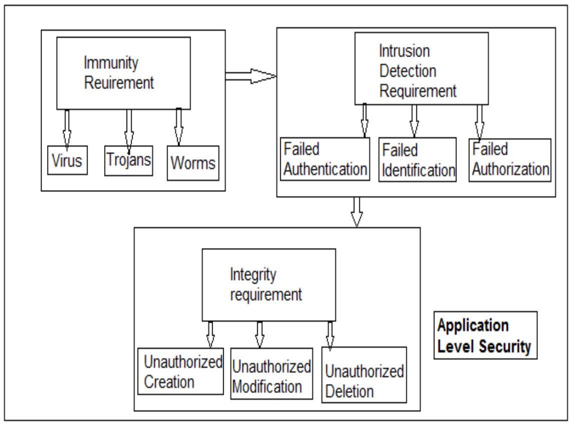


Figure 10 Application Level Security at Requirement Phase

3 Functional Level Security: To implement the security at functional level three major factors are considered as shown in the diagram below. They are No repudiation requirement which considers the date and time of received paper, date and time of sent acknowledgement and identity of the author. The Privacy requirement which considers the anonymity i.e. the application shall not store any personal information about the authors; the data storage privacy i.e. the application shall not allow unauthorized individuals or programs access
to any stored data; the communications privacy i.e. the application shall not allow unauthorized individuals or programs access to any communications. The Security Auditing Requirement gives the summary of status of immunity, status of integrity and status of intrusion detection.

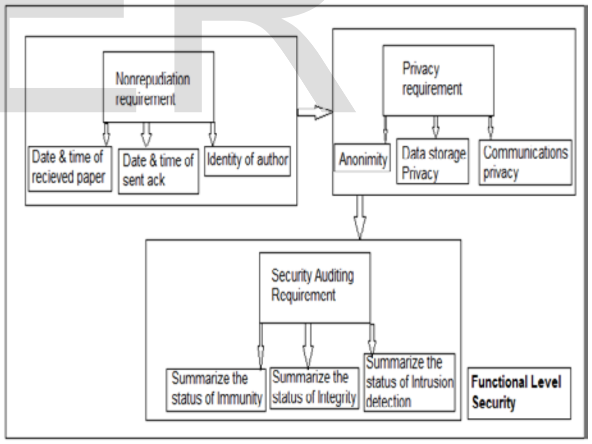

Nonrepudiation requirement
Date & time of recieved paper
Date & time of sent ack
Identity of author
Privacy requirement
Anonimity
Data storage Privacy
Communications privacy
Security Auditing Requirement
Summarize the status of Immunity
Summarize the status of Integrity
Summarize the status of Intrusion detection
Functional Level Security

Figure 11 Functional Level Security at Requirement Phase

4 Security For Buisness Continuity: The security at this level considers three basic factors survivability requirement, physical protection requirement and system maitenance security requirement. Survivability highlights the chances of component failure and database server failure, physical protection requirement identifies the security of the harware and the personnel and system maitenance security requirement identifies the precautions to be taken while upgrading hardware or software and while replacing hardware or software.

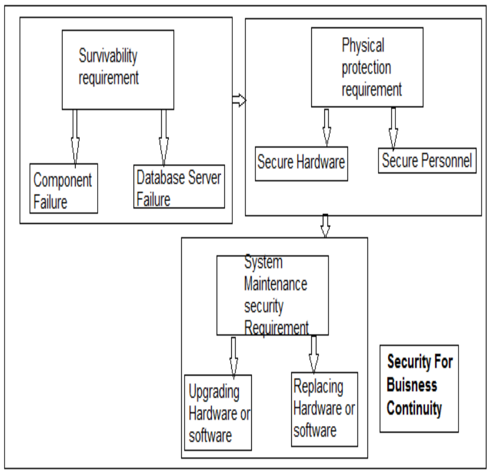

Survivability requirement
Component Failure
Database Server Failure
Physical protection requirement
Secure Hardware
Secure Personnel
System Maintenance security Requirement
Upgrading Hardware or software
Replacing Hardware or software
Security For Buisness Continuity

Figure 12 Security For Buisness Continuity at Requirement Phase

3.2Design phase: Software design represents the static structure and dynamic behaviour of software. It is necessary to make design decisions that are secure and do not introduce any security vulnerabilities in the completed software. Designing for security in software is futile unless it is planned to act on the design and incorporate necessary secure controls during the development stage of software development lifecycle.

3.2.1 Vulnerabilities at design phase:

We have identified the vulnerabilities at design phase by classifying the design phase as: data structure design, algorithm design, graphical interface user design, security design, hypertext design, authoring system design and access design. Vulnerability at each of the classification is explained below:

a. Data Structure Design: The data structure design can be vulnerable if there is no secure storing of sensitive data, especially those with high levels of confidentiality and integrity. The structure should follow the following flow chart as mentioned in [20]. The weightage given to the data should be maintained other the probability of vulnerabilities in data structure design increases.

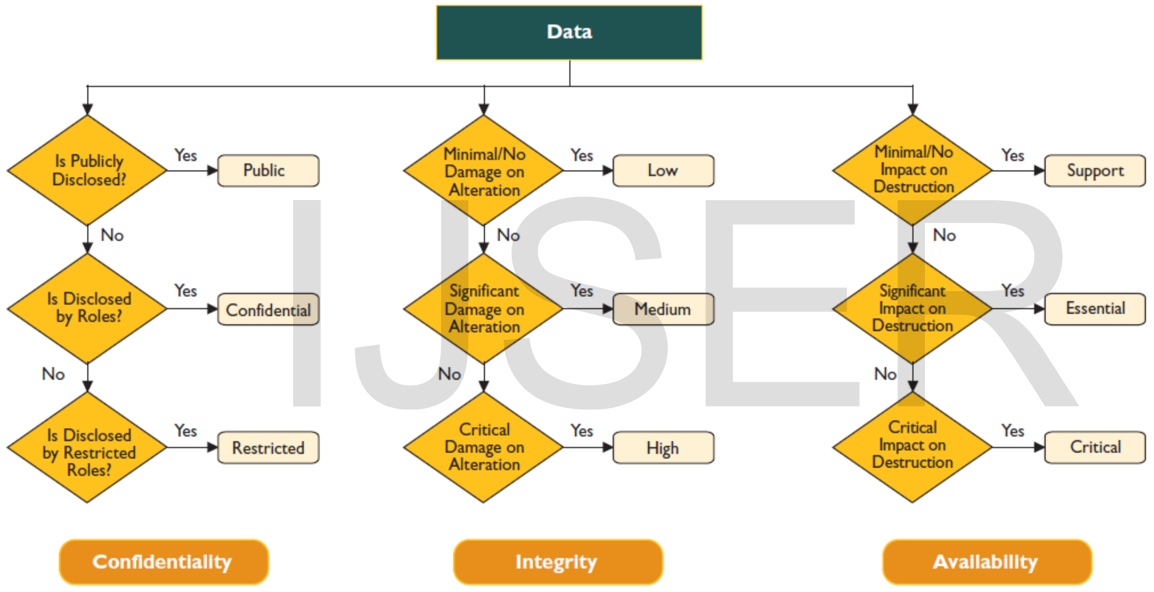

Data
Is Publicly Disclosed?
Yes
Public
No
Is Disclosed by Roles?
Yes
Confidential
No
Is Disclosed by Restricted Roles?
Yes
Restricted
Minimal/No Damage on Alteration
Yes
Low
No
Significant Damage on Alteration
Yes
Medium
No
Critical Damage on Alteration
Yes
High
Minimal/No Impact on Destruction
Yes
Support
No
Significant Impact on Destruction
Yes
Essential
No
Critical Impact on Destruction
Yes
Critical
Confidentiality
Integrity
Availability

Figure 13 Algorithm Design

b. Algorithm Design: Algorithm design deals with choosing a proper programming language in regards to program type, requirements and expected functionality. It is an important design time decision that can mitigate much possible vulnerability. For instance use of improper locking mechanism on shared resources may lead to vulnerability. Non usage of library calls instead relying on external/ system calls may also lead to vulnerable code.

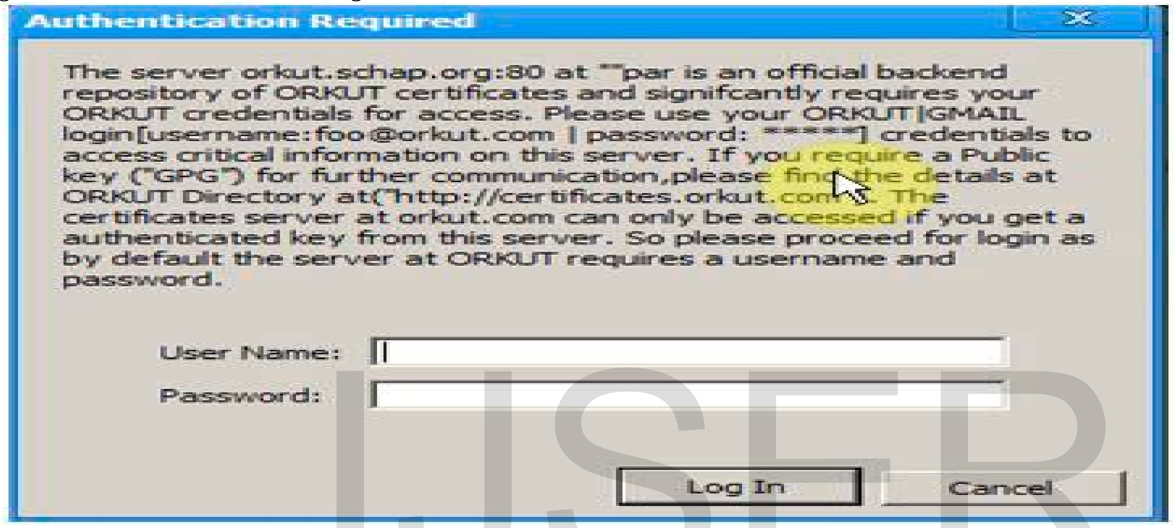
"when analyzing the running time of algorithms, a common technique is to differentiate best-case, common-case, and worst-cast performance. For example, an unbalanced binary tree will be expected to consume *O*(*n*log*n*) time to insert *n* elements, but if the elements happen to be sorted beforehand, then the tree would degenerate to a linked list, and it would take *O*(*n*2) time to insert all *n* elements. Similarly, a hash table would be expected to consume *O*(*n*) time to insert *n* elements. However, if each element hashes to the same bucket, the hash table will also degenerate to a linked list, and it will take *O*(*n*2) time to insert *n* elements. While balanced tree algorithms, such as red-black trees , AVL trees , and treaps can avoid predictable input which causes worst-case behavior, and universal hash functions can be used to make hash functions that are not predictable by an attacker, many common applications use simpler algorithms. If an attacker can control and predict the inputs being used by these algorithms, then the attacker may be able to induce the worst-case execution time, effectively causing a denial-of-service (DoS) attack."

c. Graphical User Interface: The user interacts with the web application using graphical user interface (GUI). There is much vulnerability in GUI design which may lead to different attacks. As discussed in [21] the design flaws in browser as related to GUI are as follows:

d.**HTTP Authentication Dialog Spoofing**

If a resource is protected, the server sends a particular HTTP response to the browser based on which the browser initiates a dialog authentication process. It is one of the main characteristics of
browsers to handle HTTP authentication. Every single HTTP authentication process has a realm value associated with it. In general, the realm value is a string that shows the domain name on which resource is protected. The real value also provides a user supplied string for identity purposes. A user can check the domain name and provide his credentials to gain access to the server. However, recent vulnerabilities have shown the fact that it is possible to manipulate the authentication dialog box.

A dialog box may look real and authentic but it can be spoofed. This type of flaw in browsers results in the stealing of user credentials without users being aware of the reality. For example: Internet Explorer and Google Chrome inherit this design flaw. A serious design flaw in Google [20] is that an authentication dialog can be completely spoofed and users are not able to distinguish the difference. A spoofed authentication dialog box is presented as in Figure 13.



Authentication Required

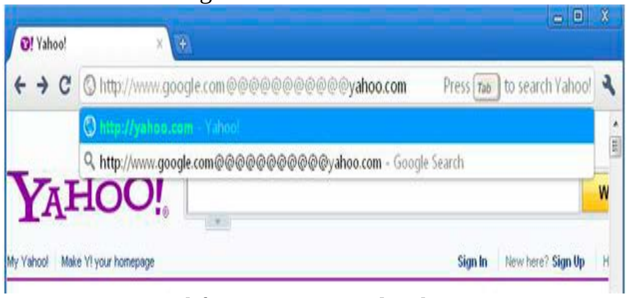
The server orkut.schap.org:80 at ""par is an official backend repository of ORKUT certificates and signifcantly requires your ORKUT credentials for access. Please use your ORKUT|GMAIL login[username:foo@orkut.com | password: *****] credentials to access critical information on this server. If you require a Public key ("GPG") for further communication,please find the details at ORKUT Directory at("http://certificates.orkut.com"). The certificates server at orkut.com can only be accessed if you get a authenticated key from this server. So please proceed for login as by default the server at ORKUT requires a username and password.

Figure 13: Spoofed authentication dialog box in Google Chrome

The spoofed authentication dialog box bedazzles the user. However, it has been noticed that a number of users fall into this trap and provide their authentication credentials as per the realm value shown in the dialog box. This design flaw persists because browsers are not able to handle the realm value passed as a parameter to the authenticated HTTP response header and render it directly in the dialog box. Most browsers do not handle the realm value in an appropriate manner, allowing spoofing attacks.

e. URL Obfuscation Flaws URL obfuscation is a trick that plays around the designing of URLs with certain meta characters in order to confuse browsers as well as users so that they can be redirected to malicious domain. This is a browser design flaw because browsers are not able to render the URLs appropriately thereby resulting in unauthorized redirection. As a result, the browser can be redirected to a malicious domain that is ready to serve malware.

In general, good practice requires that browsers should raise a warning about the obfuscation in a URL and should be smart enough to present a user with an appropriate choice. Primarily, the user thinks that a destination website is Google.com, but in reality, the user is redirected towards yahoo.com. An obfuscated URL is shown in Figure 2.

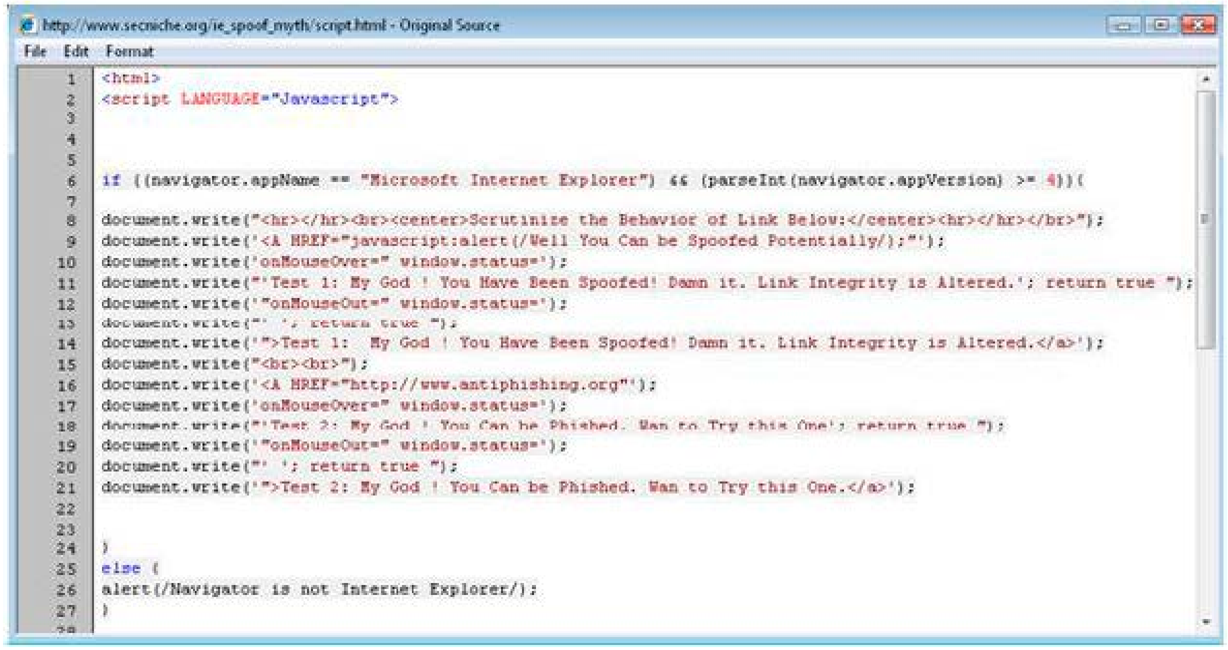


Yahoo!
http://www.google.com@@@@@@@@@@@yahoo.com
Press Tab to search Yahoo!
http://yahoo.com - Yahoo!
http://www.google.com@@@@@@@@@@@yahoo.com - Google Search
YAHOO!
My Yahoo!
Make Y! your homepage
Sign In
New here? Sign Up

Figure 14: URL Obfuscation in Google Chrome

Manipulating Browser Status Bars Browser status bars are used to present the active state of links when a user clicks a hyperlink on a webpage. In general, status bars represent the status of hyperlinks. The mindset behind the design of the status bar is that a user can see the authenticity of domain names and hyperlink. Basically, a user believes that the status bar displays the domain name in the form of a URL and the browser redirects to that page upon clicking. Attackers have exploited this design flaw by spoofing the status bar with JavaScript calls such as window .location or window. href to fool users.

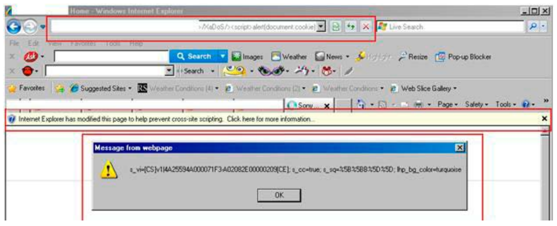
Figure shows code that is used to spoof the status bar in Internet Explorer.


http://www.secniche.org/ie_spoof_myth/script.html - Original Source

```
<html>
<script LANGUAGE="Javascript">



if ((navigator.appName == "Microsoft Internet Explorer") && (parseInt(navigator.appVersion) >= 4)){

document.write("<hr></hr><br><center>Scrutinize the Behavior of Link Below:</center><hr></hr></br>");
document.write('<A HREF="javascript:alert(/Well You Can be Spoofed Potentially/);"');
document.write('onMouseOver=" window.status=');
document.write("'Test 1: My God ! You Have Been Spoofed! Damn it. Link Integrity is Altered.'; return true ");
document.write('"onMouseOut=" window.status=');
document.write("' '; return true ");
document.write('">Test 1:  My God ! You Have Been Spoofed! Damn it. Link Integrity is Altered.</a>');
document.write("<br><br>");
document.write('<A HREF="http://www.antiphishing.org"');
document.write('onMouseOver=" window.status=');
document.write("'Test 2: My God ! You Can be Phished. Wan to Try this One'; return true ");
document.write('"onMouseOut=" window.status=');
document.write("' '; return true ");
document.write('">Test 2: My God ! You Can be Phished. Wan to Try this One.</a>');


}
else {
alert(/Navigator is not Internet Explorer/);
}
```

Figure 15: Custom HTML Code to Spoof Internet Explorer's Status Bar

f. Cross Site Scripting Attack Notification Bars – Bypassing Filters
With the advent of new browser security protection mechanisms, reflective Cross Site Scripting (XSS) filters have become a part of the browser architecture. The XSS filters in browsers are not well developed and can be bypassed easily to execute successful XSS attacks. For example, Internet Explorer released a built-in XSS filter with Internet Explorer 8, but it can be bypassed easily and no notification alert is raised. However, Internet Explorer's XSS filter raised a notification warning but was not able to sanitize the XSS attacks appropriately. This type of behaviour shows the inherent weakness in client-side XSS filters. Moreover, NoScript is considered a very good extension of Mozilla that prevents reflective XSS attacks.



Internet Explorer has modified this page to help prevent cross-site scripting. Click here for more information.

s_vi=[CS]v1|4A25594A000071F3-A02082E00000209[CE]; s_cc=true; s_sq=%5B%5BB%5D%5D; lhp_bg_color=turquoise

Figure 16: Successful bypass even after XSS notification

g. Download Dialog Box Spoofing Browsers use a download dialog box in order to download a file from a server. This process acts as a notification to the user about the characteristics of the file. The download dialog box is displayed when a user clicks a hyperlink to download a specific file. It is a type of GUI displayed to the user for raising an alert. Attackers are spoofing download dialog boxes to trick users into downloading malicious files instead of authorized files. This attack is triggered on a wide scale to infect user machines with malware. This attack is implemented in order to force a user to interact with the rogue pop-up window. In other words, it is a design bug in Internet Explorer that fails to differentiate between the download dialog box and a rogue pop-up window. Figure 16 shows the spoofed download dialog box in Internet Explorer 8.
In the Figure 6 screenshot, a fake End User License Agreement (EULA) pop up window overlaps the authorized download dialog box. This fake EULA window is embedded with malicious links and it locks the download dialog box completely. This attack forces the user to interact with a EULA window prior to downloading the file. In general, users are not aware of these design problems and spoofing tricks which help an attacker to launch attacks successfully. The figure clearly shows one of the serious design bugs in graphical user components in browsers.
h. Clickjacking Browser Interface The aim of this attack is to steal sensitive data and extract information about a user's activities in a stealthy manner. Primarily, this attack uses two major UI components in a browser–frames and buttons. The term click jacking itself points to hijacking mouse clicks in a browser window. In general terms, an attacker designs a transparent UI component such as a button and makes it hidden. When a legitimate user performs a mouse click in a browser window, the hidden button is clicked and it executes the backend command designed by the attacker to perform rogue functions. This attack is considered one of the most sophisticated attacks.

i. Security Design: Depending on the application being designed, the types of issues that must be addressed vary. For example, when you design a secure Web application, it is important that you follow guidelines to ensure effective user authentication and authorization, to protect sensitive data as it is transmitted over public networks, and to prevent attacks such as session hijacking.

Some of the important Web application issues that must be addressed with secure design practices are shown in Figure 17.

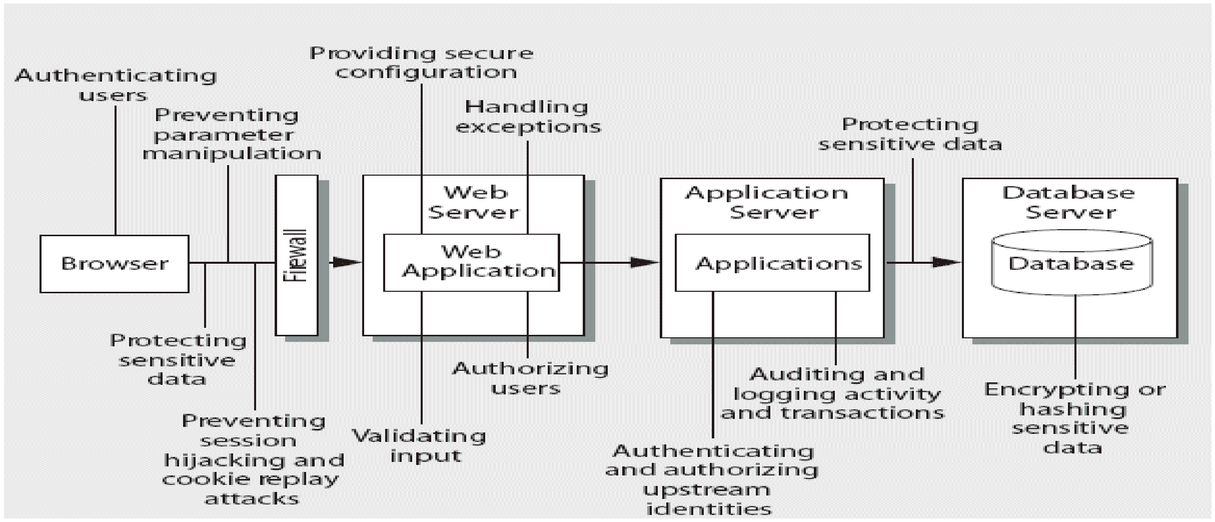

Authenticating users
Providing secure configuration
Preventing parameter manipulation
Handling exceptions
Protecting sensitive data
Browser
Firewall
Web Server
Web Application
Application Server
Applications
Database Server
Database
Protecting sensitive data
Authorizing users
Auditing and logging activity and transactions
Encrypting or hashing sensitive data
Preventing session hijacking and cookie replay attacks
Validating input
Authenticating and authorizing upstream identities

Figure 17 Secure Design practices

j. Hypertext Design: As mentioned in part of Hyper Text Transfer Protocol in the HTTP 1.1 has following vulnerabilities:

1.1 Personal Information HTTP clients are often privy to large amounts of personal information (e.g. the user's name, location, mail address, passwords, encryption keys, etc.), and SHOULD be very careful to prevent unintentional leakage of this information via the HTTP protocol to other sources.

1.1.1 Abuse of Server Log Information

A server is in the position to save personal data about a user's requests which might identify their reading patterns or subjects of interest. This information is clearly confidential in nature and its handling can be constrained by law in certain countries. People using the HTTP protocol to provide data are responsible for ensuring that such material is not distributed without the permission of any individuals that are identifiable by the published results.

1.1.2 Transfer of Sensitive Information

Like any generic data transfer protocol, HTTP cannot regulate the content of the data that is transferred, nor is there any a priori method of determining the sensitivity of any particular piece of information within the context of any given request. Therefore, applications SHOULD supply as much control over this information as possible to the provider of that information. Four header fields are worth special mention in this context: Server, Via, Referer and From.

1.1.3 Encoding Sensitive Information in URI's

Because the source of a link might be private information or might reveal an otherwise private information source, it is strongly recommended that the user be able to select whether or not the Referer field is sent. For example, a browser client could have a toggle switch for browsing openly/anonymously, which would respectively enable/disable the sending of Referer and From information.

Authors of services which use the HTTP protocol SHOULD NOT use GET based forms for the submission of sensitive data, because this will cause this data to be encoded in the Request-URI. Many existing servers, proxies, and user agents will log the request URI in some place where it might be visible to third parties. Servers can use POST-based form submission instead.

1.1.4 Privacy Issues Connected to Accept Headers

Accept request-headers can reveal information about the user to all servers which are accessed. The Accept-Language header in particular can reveal information the user would consider to be of a private nature, because the understanding of particular languages is often strongly correlated to the membership of a particular ethnic group. User agents who offer the option to configure the contents of an Accept-Language header to be sent in every request are strongly encouraged to let the configuration process include a message which makes the user aware of the loss of privacy involved.

1.2 Attacks Based On File and Path Names

Implementations of HTTP origin servers SHOULD be careful to restrict the documents returned by HTTP requests to be only those that were intended by the server administrators. If an HTTP server translates HTTP URIs directly into file system calls, the server MUST take special care not to serve files that were not intended to be delivered to HTTP clients. For example, UNIX, Microsoft Windows, and other operating systems use ".." as a path component to indicate a directory level above the current one. On such a system, an HTTP server MUST disallow any such construct in the Request-URI if it would otherwise allow access to a resource outside those intended to be accessible via the HTTP server. Similarly, files intended for reference only internally to the server (such as access control files, configuration files, and script code) MUST be protected from inappropriate retrieval, since they might contain sensitive information. Experience has shown that minor bugs in such HTTP server implementations have turned into security risks.

1.3 DNS Spoofing

Clients using HTTP rely heavily on the Domain Name Service, and are thus generally prone to security attacks based on the deliberate mis-association of IP addresses and DNS names. Clients need to be cautious in assuming the continuing validity of an IP number/DNS name association.

1.4 Location Headers and Spoofing

If a single server supports multiple organizations that do not trust one another, then it MUST check the values of Location and Content- Location headers in responses that are generated under control of said organizations to make sure that they do not attempt to invalidate resources over which they have no authority.

1.5 Content-Disposition Issues

RFC 1806 from which the often implemented Content-Disposition header in HTTP is derived, has a number of very serious security considerations. Content-Disposition is not part of the HTTP standard, but since it is widely implemented, we are documenting its use and risks for implementers.

1.6 Authentication Credentials and Idle Clients

Existing HTTP clients and user agents typically retain authentication information indefinitely. HTTP/1.1. Does not provide a method for a server to direct clients to discard these cached credentials. This is a significant defect that requires further extensions to HTTP. Circumstances under which credential caching can interfere with the application's security model include but are not limited to:

- Clients which have been idle for an extended period following which the server might wish to cause the client to re prompt the user for credentials.
- Applications which include a session termination indication (such as a `logout' or `commit' button on a page) after which the server side of the application `knows' that there is no further reason for the client to retain the credentials.

1.7 Proxies and Caching

By their very nature, HTTP proxies are men-in-the-middle, and represent an opportunity for man-in-the-middle attacks. Compromise of the systems on which the proxies run can result in serious security and privacy problems. Proxies have access to security-related information, personal information about individual users and organizations, and proprietary information belonging to users and content providers. A compromised proxy, or a proxy implemented or configured without regard to security and privacy considerations, might be used in the commission of a wide range of potential attacks.

1.7.1 Denial of Service Attacks on Proxies

They exist. They are hard to defend against. Research continues. Beware.

Authorization System Design: Authorization determines what the authenticated identity can do and the resources that can be accessed. Improper or weak authorization leads to information disclosure and data tampering. Defence in depth is the key security principle to apply to your application's authorization strategy.
The following practices improve your Web application's authorization:

- **Use multiple gatekeepers**.
- **Restrict user access to system-level resources**.
- **Consider authorization granularity**.

Use Multiple Gatekeepers
On the server side, you can use IP Security Protocol (IPSec) policies to provide host restrictions to restrict server-to-server communication. For example, an IPSec policy might restrict any host apart from a nominated Web server from connecting to a database server. IIS provides Web permissions and Internet Protocol/ Domain Name System (IP/DNS) restrictions. IIS Web permissions apply to all resources requested over HTTP regardless of the user. They do not provide protection if an attacker manages to log on to the server. For this, NTFS permissions allow you to specify per user access control lists. Finally, ASP.NET provides URL authorization and File authorization together with principal permission demands. By combining these gatekeepers you can develop an effective authorization strategy.
Restrict User Access to System Level Resources
System level resources include files, folders, registry keys, Active Directory objects, database objects, event logs, and so on. Use Windows Access Control Lists (ACLs) to restrict which users can access what resources and the types of operations that they can perform. Pay particular attention to anonymous Internet user accounts; lock these down with ACLs on resources that explicitly deny access to anonymous users.

For more information about locking down anonymous Internet user accounts with Windows ACLs, see Chapter 16, "Securing Your Web Server."

Consider Authorization Granularity

There are three common authorization models, each with varying degrees of granularity and scalability.

The most granular approach relies on impersonation. Resource access occurs using the security context of the caller. Windows ACLs on the secured resources (typically files or tables, or both) determine whether the caller is allowed to access the resource. If your application provides access primarily to user specific resources, this approach may be valid. It has the added advantage that operating system level auditing can be performed across the tiers of your application, because the original caller's security context flows at the operating system level and is used for resource access. However, the approach suffers from poor application scalability because effective connection pooling for database access is not possible. As a result, this approach is most frequently found in limited scale intranet-based applications. The impersonation model is shown in Figure 4.5.

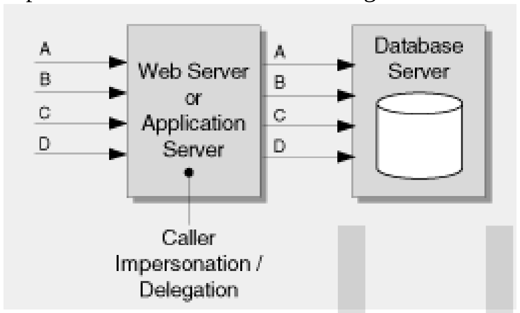

A
B
C
D
Web Server or Application Server
A
B
C
D
Database Server
Caller Impersonation / Delegation

Figure 18

Impersonation model providing per end user authorization granularity

The least granular but most scalable approach uses the application's process identity for resource access. This approach supports database connection pooling but it means that the permissions granted to the application's identity in the database are common, irrespective of the identity of the original caller. The primary authorization is performed in the application's logical middle tier using roles, which group together users who share the same privileges in the application. Access to classes and methods is restricted based on the role membership of the caller. To support the retrieval of per user data, a common approach is to include an identity column in the database tables and use query parameters to restrict the retrieved data. For example, you may pass the original caller's identity to the database at the application (not operating system) level through stored procedure parameters, and write queries similar to the following:

SELECT field1, field2, field3 FROM Table1 WHERE {some search criteria} AND UserName = @originalCallerUserName

This model is referred to as the trusted subsystem or sometimes as the trusted server model. It is shown in Figure 4.6.

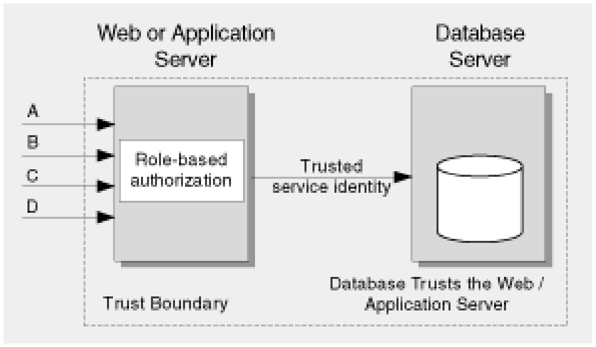


Figure 19

Trusted subsystem model that supports database connection pooling

The third option is to use a limited set of identities for resource access based on the role membership of the caller. This is really a hybrid of the two models described earlier. Callers are mapped to roles in the application's logical middle tier, and access to classes and methods is restricted based on role membership. Downstream resource access is performed using a restricted set of identities determined by the current caller's role membership. The advantage of this approach is that permissions can be assigned to separate logins in the database, and connection pooling is still effective with multiple pools of connections. The downside is that creating multiple thread access tokens used to establish different security contexts for downstream resource access using Windows authentication is a privileged operation that requires privileged process accounts. This is counter to the principle of least privilege. The hybrid model using multiple trusted service identities for downstream resource access is shown in Figure 4.7.

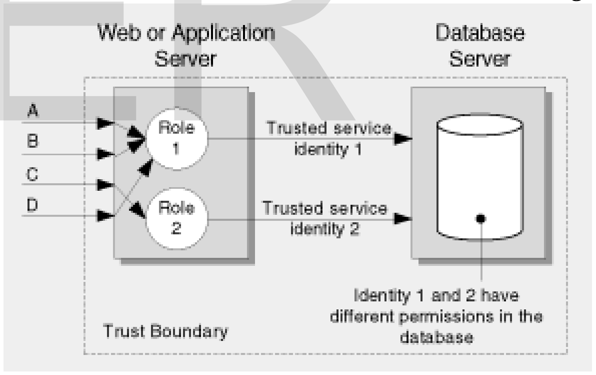
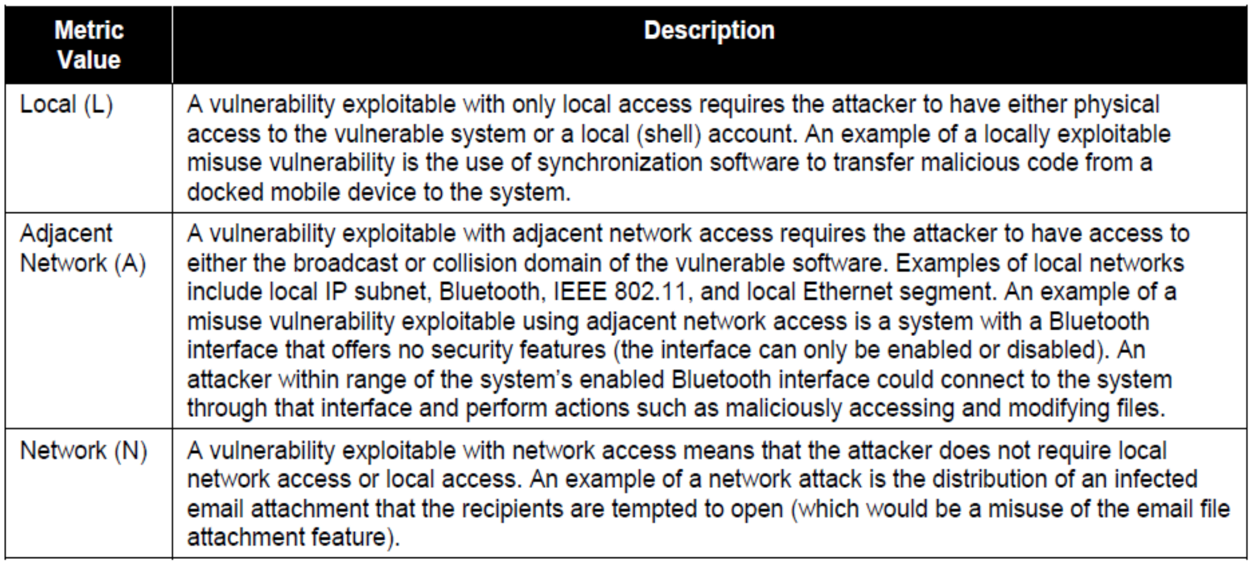

Web or Application Server
Database Server
A
B
C
D
Role 1
Role 2
Trusted service identity 1
Trusted service identity 2
Identity 1 and 2 have different permissions in the database
Trust Boundary

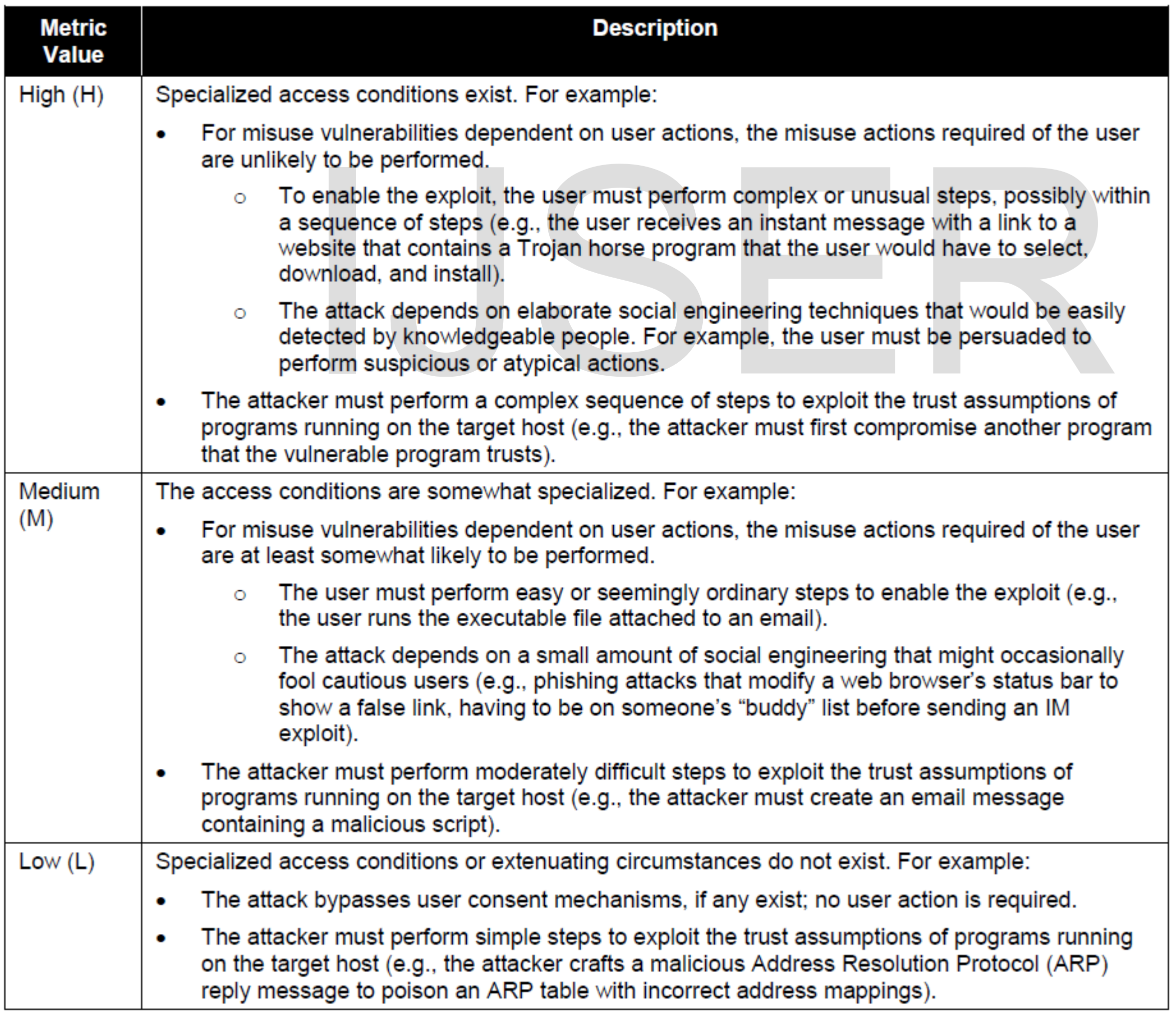
Figure 20

Hybrid model

g. Access Design: With the help of [11] we identify that to identify vulnerability of access design we need to consider what access to the system the attacker must possess in order to exploit the software feature.



Metric Value	Description
Local (L)	A vulnerability exploitable with only local access requires the attacker to have either physical access to the vulnerable system or a local (shell) account. An example of a locally exploitable misuse vulnerability is the use of synchronization software to transfer malicious code from a docked mobile device to the system.
Adjacent Network (A)	A vulnerability exploitable with adjacent network access requires the attacker to have access to either the broadcast or collision domain of the vulnerable software. Examples of local networks include local IP subnet, Bluetooth, IEEE 802.11, and local Ethernet segment. An example of a misuse vulnerability exploitable using adjacent network access is a system with a Bluetooth interface that offers no security features (the interface can only be enabled or disabled). An attacker within range of the system's enabled Bluetooth interface could connect to the system through that interface and perform actions such as maliciously accessing and modifying files.
Network (N)	A vulnerability exploitable with network access means that the attacker does not require local network access or local access. An example of a network attack is the distribution of an infected email attachment that the recipients are tempted to open (which would be a misuse of the email file attachment feature).



Metric Value	Description
High (H)	Specialized access conditions exist. For example: • For misuse vulnerabilities dependent on user actions, the misuse actions required of the user are unlikely to be performed. ○ To enable the exploit, the user must perform complex or unusual steps, possibly within a sequence of steps (e.g., the user receives an instant message with a link to a website that contains a Trojan horse program that the user would have to select, download, and install). ○ The attack depends on elaborate social engineering techniques that would be easily detected by knowledgeable people. For example, the user must be persuaded to perform suspicious or atypical actions. • The attacker must perform a complex sequence of steps to exploit the trust assumptions of programs running on the target host (e.g., the attacker must first compromise another program that the vulnerable program trusts).
Medium (M)	The access conditions are somewhat specialized. For example: • For misuse vulnerabilities dependent on user actions, the misuse actions required of the user are at least somewhat likely to be performed. ○ The user must perform easy or seemingly ordinary steps to enable the exploit (e.g., the user runs the executable file attached to an email). ○ The attack depends on a small amount of social engineering that might occasionally fool cautious users (e.g., phishing attacks that modify a web browser's status bar to show a false link, having to be on someone's "buddy" list before sending an IM exploit). • The attacker must perform moderately difficult steps to exploit the trust assumptions of programs running on the target host (e.g., the attacker must create an email message containing a malicious script).
Low (L)	Specialized access conditions or extenuating circumstances do not exist. For example: • The attack bypasses user consent mechanisms, if any exist; no user action is required. • The attacker must perform simple steps to exploit the trust assumptions of programs running on the target host (e.g., the attacker crafts a malicious Address Resolution Protocol (ARP) reply message to poison an ARP table with incorrect address mappings).

3.2.2 Patches: Security design principles are a specific type of guidelines and practices. They are proven rules for improving the security posture of an application, and in order to be useful, the principles must be applied to specific problems.
The security design principles in Table 1 are built
upon the idea of simplicity and restriction.
Based on the authors [20] SINTEF have identified the security design reviews and etal 2006 has defined threat modelling using STRIDE, an acronym for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. which is listed in the table below The table shows a simplified version of a checklist focused on Web application security.

Table 1. security design Principles

Element	Threat	Patch
Input validation	Spoofing, Denial of Service, Tampering, Information Disclosure	All entry points and trust boundaries are identified by the design.
		Input validation is applied whenever input is received from outside the current trust boundary.
		The design addresses potential SQL injection issues.
		The design addresses potential cross-site scripting issues.
		The design does not rely on client-side validation.
Authentication	Spoofing	The design partitions the Web site into public and restricted areas.
		Account management policies are taken into consideration by the design.
		The design ensures that minimum error information is returned in the event of authentication failure.
		The design adopts a policy of using least-privileged accounts.
		The identity that is used to authenticate with the database is identified by the design.
Authorization	Elevation of Privilege	The role design offers sufficient separation of privileges (the design considers authorization granularity).
		The design identifies code access security requirements. Privileged resources and privileged operations are identified.
		All identities that are used by the application are identified and the resources accessed by each identity are known.
Sensitive data	Denial of Service, Tampering, Information Disclosure, Repudiation	The design identifies the methodology to store secrets securely.
		The design identifies protection mechanisms for sensitive data that is sent over the network.
		Secrets are not stored unless necessary.

Cryptography	Denial of Service, Tampering, Information Disclosure, Repudiation, Spoofing	The methodology to secure the encryption keys is identified.
		Platform-level cryptography is used and it has no custom implementations.
		The design identifies the key recycle policy for the application.
Exceptions	Repudiation	The design outlines a standardized approach to structured exception handling across the application.
		The design identifies generic error messages that are returned to the client.
Auditing and logging	Tampering, Repudiation,	The design identifies the level of auditing and logging necessary for the application and identifies the key parameters to be logged and audited.
		The design identifies the storage, security, and analysis of the application log files.

Table 2. **Checklist for security review**

3.3Implementation phase:

3.3.1 Vulnerabilities at implementation phase: The vulnerabilities of implementation phase are listed [8]:

1. Environment variables: Variables that encapsulate information that does not change across executions of a program. On UNIX systems, the PATH environment variable lists the directories to be searched for a named executable. Regardless of how many different executables are searched for, the PATH variable's value does not change.

2. *Buffer Overflows*: Overflowing a memory stack so that the program will execute the data after the last address in the stack, usually an executable program that establishes a root or command line shell giving the attacker full control of the system. Others are heap overflows that contain code that the program can branch to via function pointers, and data overflows to alter variable values in conjunction with executing code contained in environment variables.

3. *Data as Instructions* or *Script Injections*: Using scripting languages to include information with executable code which the system executes due to
Improper input checking.

4. *Numeric Overflows*: Giving a larger or smaller value than expected. This assumes that a particular value stays within established bounds. The concept is to look for numbers that can be more than 2^32 or greater, or the maximum integer.

5. *Race Conditions*: Sending a string of data before another is executed. The most common type is the "Time of Check to Time of Use" flaw. Another is masquerading or "Man-In-The-Middle" attacks.

6. *Network Exposures*: Assuming that clients will check messages sent to a server adequately. Remote commands and executables provide the majority of examples of this type of exploit ("r" protocols like rsh, rlogin, and especially rexd).

7. *Information Exposure*: Exposing sensitive information to unauthorized users that can be used to compromise data or systems. For example: 1) non-secure transmission of sensitive information such as human resource data that can be used for social engineering; 2) Use of clear text user ID's and passwords; and 3) weak encryption schemes for access.

8. *Operational Misuse*: Operating a system in a non-secure mode. Using standard accounts with blank passwords, or providing open shares giving everyone access. Anonymous file transfer is common where users are given read/write access to a set of directories or files.

9. *Default Settings*: Default software settings may present a risk if they require user intervention to secure them. For example, Root or Administrator accounts that do not require an initial strong password also present risks if they are not set when installed such as Windows NT and 2000. Also, applications using open ports that neither the system nor application check for authentication, present potential risks. Known examples are: SunOS's use of "+" in the default /etc/hosts.equiv file; or leaving the uudecode alias in the mail alias file.

10. *Programmer Backdoors*: Unauthorized access paths left by developers of the software for easy access. If web services are included, this list greatly changes and expands as shown by Jaquith. The evaluation of security flaws in 45 commercial applications, found security design flaws in 70 percent of the defects observed, with nearly half of these classified as serious.

3.3.2 Patches

To perform a secure implementation different languages have their own constraints. The coding of the above case study is being implemented in PhP 5.3.6. We have worked on how to eliminate little vulnerability in C/C++ which are in common with PhP 5.3.6. The following steps should be followed [12]: ***Of strings:*** Weaknesses in string representation, string management, and string manipulation have caused a broad range of software vulnerabilities and exploits. Unbounded string copies, null-termination errors, and string truncation errors have led to numerous vulnerabilities in C and C++ programs, including the ubiquitous buffer overflow. However, help is either here or on the way. C++ programmers can use the standard std::string class defined in ISO/IEC standard 14882.1 The std::string class is the char instantiation of the std::basic_string template class, and it uses a dynamic approach to strings in that memory is allocated as required—meaning that in all cases, size() <= capacity().

Of integers:An inherent problem in computing is that digital representations of integers are always limited in the range of values they can represent. As a result, operations on these integers can result in integer overflow, truncation, and sign errors. Attackers often exploit integers used as array indices, loop counters, or lengths to create buffer overflows and execute arbitrary code. One solution for C++ users is to use the SafeInt template class, written by David LeBlanc.6 Before performing operations, most SafeInt functions evaluate operands to determine whether an error will occur. Because the class is declared as a template, you can use it with any integer type. It overrides nearly every relevant operator (except for the subscript operator) so that arithmetic
Operators can be used in normal inline expressions.

Methods, tools, and processes: Safe integer operations aren't necessarily the only solution to integer-overflow and other integer exception errors, but they do provide a safety net that is largely missing in C. Input validation and integer range checking are important mitigations against vulnerabilities in both C and C++. Safer, secure string libraries are available in both languages, although errors leading to vulnerabilities are still possible. As a result, software developers should still follow a policy of defence in depth and not rely on a single strategy.

3.4 Testing Phase:

3.4.1 Vulnerability: Vulnerability at testing phase is no test cases are developed to verify security of a system.

3.4.2 Patches: To avoid this vulnerability the developer should perform the following activities:

- Stress Testing (Abnormal activity, abnormal input) should be performed to validate design assumption.
- Test cases should be based on attack patterns.
- Software is able to limit the damage and rapidly recovers from attacks if succeeded.
- White box (Static/dynamic code analysis, fault, injection or propagation analysis) should be performed
- Verification of security standard should be confirmed.
- Test cases should comprise security concerns.

Importance of Stress Testing

Stress testing is considered to be important because of following reasons:

1. Almost 92% of the software/systems are developed with an assumption that they will be operating under normal scenario. And even if it is considered that the limit of normal operating conditions will be crossed, it is not considerably as high as it really could be.

2. The cost or effect of a very important considerable software, system and website failure under extreme conditions in real time can be huge (or may be catastrophic for the organization or entity owning the software/system).

3. It is always better to be prepared for extreme
Conditions rather than letting the system/software/
Web services crash, when the limit of normal, proper operation is crossed.

4. Testing carried out by the developer of the system /software/website may not be sufficient to help reveal conditions which will lead to crash of the system/software when it is actually submitted to
the operating environment.

5. It's not always possible to reveal possible problems or bugs in a system/software, unless it is subjected to such type of testing.

We have generated test cases for checking security at each phase and also the test cases for checking security after the integration of each phase. Very few researches have been done in this field. But we believe this phase should not be neglected when security of an application is concerned.

Since the above case study is being developed in PhP 5.3.6 where the cross site scripting can be avoided by Input Validation, we have developed test cases for the same.

Table 3: Test case for input validation

Test Case 1	
Test Objective: To check Input Validation of authors	
Preconditions: As defined in the requirement phase each author is given a unique Id. The communication of the authors with PC member is not visible to others.	
Test Steps	Expected Result
Step 1: Enter the qualification of the author as "high school"	"Qualification should be "graduate"
Step 2: Enter the employment status of the author as "unemployed"	"Author should be employed in an organisation or a student of an institute"
Step 3: More than three attempts to enter the username and password	"Your login attempt is expired"
Step 4: View review of other authors	"Enter login password to view review"
Test Result: Pass/Fail	

Table 4: Test case for input validation

Test Case 2	
Test Objective: To check Input Validation of PC members	
Preconditions: As defined in the requirement phase PC member and PC chair are given separate login ID. The author cannot change review status. The content of the paper is accessible only to PC member and PC chair	
Test Steps	Expected Result
Step 1: Enter the guessed user name and password	"User name and password does not match"
Step 2: Enter the login id of administrator and change the review status.	"Permission denied"
Step 3: Login using author id and view the content of different author.	"Access denied"
Test Result: Pass/Fail	

3.4.3 Tools: At testing phase the different test cases should be developed to observe security at this phase. These test cases can be used as tools to represent security at this level.

3.5. Maintenance Phase:

3.5.1 Vulnerability: Security at this phase comes into account only after the application is deployed. Hence the vulnerability at phase plays an important role when corrective and preventive maintenance is considered. Some malicious code threats during the software's operation include attacks intended to implant new malicious code or to execute a vulnerability or malicious code embedded in the targeted software. Examples of these attacks include zero-day attacks; viruses (macro, polymorphic, stealth viruses); worms; logic and time bombs; Trojan horses; network attacks; exploitation of trapdoors and rootkits; cross-site scripting attacks; SQL, XML, and other command injection attacks; exploitation of buffer overflows; format-string attacks; insider attacks; malicious mobile code attacks and reconnaissance attacks such as connection or password sniffing. Malicious code vulnerabilities are introduced during maintenance of the software, just as when the software was constructed. Downloading recommended security patches is considered to be a best practice; however certain patches may have adverse effects. For example, some patches may be compromised, or have inadequate testing and validation. Patches may also conflict with environment components as they are configured in the operational environment. Obtaining patches or updates from multiple sources may increase the vulnerability of the software; this is especially true for systems that also contain OSS or legacy components. Network stability and regression issues may occur, or patches may interfere adversely with the previously existing software.

3.5.2 Patches: To have a secure maintenance phase the change control should also be performed securely. Monitoring the activities of the application is normally neglected which is vulnerable. Different logins, log time, files, performance of the application should be monitored regularly 2007. Because software patches and updates are so important to the software, implementation of a patch management process is one way to strengthen the software's security. There are various guidelines available that can be tailored to an individual environment, but generally they include some or all of the following:
• A policy or strategy should be tailored to the system's unique environment. 10 September 2007 *Guidance for Addressing Malicious Code Risk* 45
• A team of qualified and trained individuals should be responsible for overseeing and implementing the process.
• Appropriate testing should be done to ensure that the patch is applied appropriately to the
intended environment. As different patches and updates are released, it is important to maintain control over which ones are implemented through a version control process.
• A process for removing patches and updates should also be developed. In the event that a patch has an adverse effect, procedures can facilitate the removal of such software.
• Vulnerabilities should be patched in a timely manner. Version control and configuration control should be used in the event of an issue with the patch or update.
• Patches should be deployed on the least sensitive or critical software first, in the event that the software fails.10
• Continuous monitoring of the software should be done on a regular basis. Patch maintenance should be an ongoing process, with regular reports and logs.11
• Patches should be examined for the presence of malicious code in a manner equivalent to the level at which the software being patched was examined.

4. SOSDLC

From the survey conducted on indentifying, analysing and implementing security at different phases of software development lifecycle we understand that to develop secure software each and every phase of development of the software should be secure. Current systems either induce security after the development of software or at particular phase of the software. When security is concerned many researchers discuss about access control or input validation. Few talk about vulnerable coding and very few about implementing security at testing or maintenance phase. After the software is deployed maintenance plays an important role. If the change control is not handled properly or monitoring is not done regularly the software becomes vulnerable for the attackers. Hence we propose a system which incurs security methods at all the phases of software development lifecycle to develop secure software. We have called this system as "Security Oriented Software Development Lifecycle" (SOSDLC) which we have implemented on requirement and design phase of a case study and have surveyed different methods for implementing security in implementation, testing, and maintenance phase .We have designed the following f

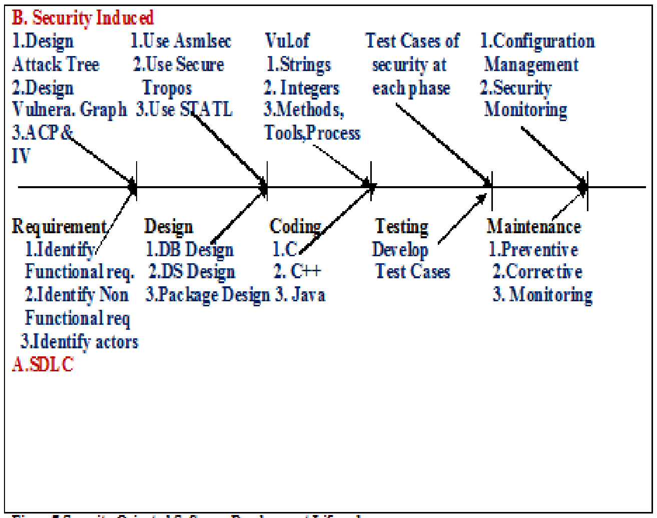


Figure7 Security Oriented Software Development Lifecycle

ramework for SOSDLC

5. Conclusion and future Scope:

Software security has become a prime necessity now days with the increased attackers and increased hackers. Many methods and tools have been proposed by many researchers. Some tools modified to obtain the maximum security index.

emphasize on security at requirement phase some at coding or some at design, implementation and maintenance also. Some researchers have worked upon overall development of secure software. Few have worked on analysing the errors at each phase while others have implemented the tools like UML state charts, Secure UML and many such tools. There are few SSDLC's which are readily available among which MS SDL and CLASP are very popular. Studies related to SSDLC indicate that standard methods or tools cannot be implemented for different types of software development. Hence in this paper using different research papers we have tried to understand different methods to secure the basic phases of secure software development that is requirement, design and coding. Based on this study I have proposed that a secure software development cycle should be based on implementing security at each phase.

As the future scope of this method I would propose that this method should be implemented for development of new software. To check the correctness of the proposed method it should be implemented on reengineered software and hence the result should be compared with the security in actual software. The security at each phase should be analyzed and quantized and hence based the security index obtained the imparted security in a software should be measured and

References

[1] P. Moore, R. J. Ellison, and R. C. Linger. Attack modelling for information security and survivability. In Dependable Systems and Networks Conference, Gothenburg, Sweden, 2001.

[2] Aditya K. Sood 2011. " Browser User Interface Design Flaws Exploiting User Ignorance", Michigan State University Richard J. Enbody, Ph.D., Michigan State University, CrossTalk.

[3] Shawn Hernan and Scott Lambert and Tomasz Ostwald and Adam Shostack. 2006 "Threat Modeling Uncover Security Design Flaws Using The STRIDE Approach" MSDN Magazine, November 2006

[4] Charles B. Haley, Robin Laney, Jonathan D. Moffett," Security Requirements Engineering:A Framework for Representation and Analysis" Member, IEEE, and

[5] Bashar Nuseibeh, Member, IEEE Computer Society

[6] Control Vulnerabilities in Web Applications Computer Systems Laboratory CSAIL Stanford University MIT {mwdalton, kozyraki}@stanford.edu nickolai@csail.mit.edu 2008

[7] Dave Hoover **"**Guidance For Addressing Malicious Code Risk" National Security Agency 9800 Savage Road, Suite 6755fort Meade, Md 20755-6755 Nsa-Guidance@Missi.Ncsc.Mil 10 September 2007

[8] David P. Gilliam, Thomas L. Wolfe, Josef S. Sherif Jet "Software Security Checklist for the Software Life Cycle" Propulsion Laboratory, California Institute of Technology david.p.gilliam@jpl.nasa.gov, thomas.l.wolfe@jpl.nasa.gov, josef.s.sherif@jpl.nasa.gov Matt Bishop University of California at Davis bishop@cs.ucdavis.edu in Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03) 1080-1383/03 $17.00 © 2003 IEEE

[9] Dianxiang Xu, Vivek Goel, and Kendall Nygard 2006 "An Aspect-Oriented Approach to Security Requirements Analysis" Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06) 0-7695-2655-1/06.

[10] D. G. Firesmith, "Engineering Security Requirements", Journal of Object Technology, Vol. 2, No. 1, January-February 2003.

[11] Elizabeth Van Ruitenbeek Karen Scarfon. The Common Misuse Scoring System (CMSS): Metrics for Software Feature Misuse Vulnerabilities (DRAFT) NIST Interagency Report 7517(Draft)

[12] J.D. Meier, Alex Mackman, Blaine Wastell, Prashant Bansode, Jason Taylor, Rudolph Araujo Security Engineering Explained - Chapter 3 - Security Design Guidelines -

[13] Khalid Sultan, Abdeslam En-Nouaary, Abdelwahab Hamou-Lhadj Department of Electrical and Computer Engineering Concordia University, Montreal, Canada "Catalog of Metrics for Assessing Security

[14] Mano Paul. The Ten Best Practices for Secure Software Development , CSSLP, CISSP, AMBCI, MCAD, MCSD, Network+, ECSA www.isc2.org 2010

[15] M. Howard. 2004. Building more secure software with improved development process. IEEE Security & Privacy, 2(6):63–65

[16] Michael Dalton, Christos Kozyrakis, and Nickolai Zeldovich. Nemesis: Preventing Authentication & Access

[17] Muhammad Umair Ahmed Khan and Mohammed Zulkernine 2009. "On Selecting Appropriate Development Processes and Requirements

Engineering Methods for Secure Software" 0730-3157/09 $25.00 © 2009 IEEE DOI 10.1109 /COMPSA

[18] Nancy R. Mead. , 2008. "SQUARE Process" Software Engineering Institute [vita] Copyright © 2006, 2008 Carnegie Mellon University2006-01-30; Updated 2008-09-17

[19] Per Håkon Meland and Jostein Jensen "Secure Software Design in Practice" SINTEF Information and Communication Technology Department of Security, Safety and System Development {Per.H.Meland, Jostein.Jensen}@sintef.no The Third International Conference on Availability, Reliability and Security

[20] S. B. Lipner. 2004. The trustwothy computing security development lifecycle. In Procedeeings of the 20th Annual Computer Security Applications Conference, pages 2–13, Tucson,AZ, USA.

[21] Scott A. Crosby Dan S. Wallach."Denial of Service via Algorithmic Complexity Attacks" scrosby@cs.rice.edu dwallach@cs.rice.edu Department of Computer Science, Rice