Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Revisiting Security Vulnerabilities: Web Applications Perspective

R. Kumar
Department of Computer Science
Jamia Millia Islamia (Central University),
New Delhi-110 025, India

Abstract: The potential for damage and the diversity of vulnerabilities and threats is astounding, both from within and without. However, while many vulnerabilities and threats exist, the remedies can be crystallized into a tractable set of practices and procedures that can mitigate known vulnerabilities and threats and help to guard against the next unknown vulnerability and threat.

Keywords: Security Vulnerabilities, Web Application Security, Vulnerability Management.

I. Introduction

As business systems embrace new technologies like Internet to offer an easy and convenient interface to clients, partners and employees alike, applications are becoming increasingly complex and closely integrated with sensitive internal systems. Seeing that, a poorly built web application is no longer restricted to defacement and brand-damage of a business. It can also leads to expensive and embarrassing exposure of personal data, financial information or intellectual property. With the corresponding increase in legislation, corporate governance and schemes such as the card industry PCI requirements, the impact on business can be severe (www.PCI.com). A report based on the published vulnerability disclosures by Cenzic, states that certain vulnerabilities are responsible for almost 80 percent of all web-related flaws in web applications [1]. Even the products of well established brands including Adobe, SAP, Microsoft, Mozilla, Sun, Apache, and Oracle were affected by security vulnerabilities. Security is termed as a compromise in confidentially, integrity, and availability, which are globally accepted attributes of security. These attributes are largely compromised due to the weaknesses of the software, which is referred to as 'vulnerability'. Such software vulnerabilities are variously the causes of threats as well as risks. Though it is rightly advocated by experts that finding and fixing software vulnerability should be done as early as possible for building secure software, there is still left a lot desired. The rest of the paper is organized as follows: Section II presents a brief discussion on the web applications, whereas in Section III, web applications' security vulnerabilities are discussed and analyzed. In Section IV, some important conclusive findings are reported. Section V presents Conclusion and Future work.

II. Web Applications

As web is growing as an effective and inexpensive channel to communicate and exchange information with prospects and transactions with customers, millions of businesses embracing it rapidly. Technically, the web is a highly programmable environment, which allows the immediate deployment of a large and diverse range of applications. Most importantly, modern websites allow the capture, processing, storage and transmission of sensitive customer data such as personal details, credit card numbers, and social security information for immediate and recurrent use. Web applications are, therefore, computer programs allowing website visitors to submit and retrieve data to/from a database over the Internet using their preferred web browser. For the more technically oriented, Web applications query the content server (essentially a content repository database) and dynamically generate web documents to serve the client (people surfing the website). The documents are generated in a standard format to allow support by all browsers (e.g., HTML or XHTML). The web browser is the key that interprets and runs all scripts, while displaying the requested pages and content. Wikipedia brilliantly terms the web browser as the 'universal client for any web application'.

III. Web Applications' Security Vulnerabilities

This section discusses the most prevalent security vulnerabilities related to web applications reported and well supported by the three largely accepted repositories of security vulnerabilities i. e. OWASP, WASC (WASC threat classification version 2.00), and SANS/OWASP (SANS Top 20 attack vectors and MITRE's Common Weakness Enumeration) are given as follows. These vulnerabilities are also reported in the superset of commonly reported vulnerabilities namely CWE (CWE/SANS Top 25) [2, 3, 4, 5].

- ***Insufficient Authentication*** When a website allows an attacker to access the content and functionality without proper authentication, it is termed as insufficient authentication. It occurs when the software/web site doesn't prove or insufficiently prove that the claim made by an actor is correct [6]. These attackers' may attempt to steal the accounts information from others. Basically, they use the leaks or flows in the authentication or session management namely expose accounts, password, session id's to prove the false identity of users [7]. Hence, depending upon the web site resources, the web application should not be directly accessible to users without verifying the authentication.

Example: There are many web applications which have been designed with administrative functionality located directly off the root directory (/admin/). Although this directory is never linked from anywhere on the web site under normal conditions, but it can be accessed by using a standard web browser. The user/developer doesn't accept anyone to view this page because they assume that it is not linked. This results in not enforcing the authentication in many times. By the way, if an attacker visits this page, s/he can get the complete administrative access to the website.

- ***Insufficient Authorization*** If the web applications' users are allowed to perform a function or access data without going through an adequate or proper authorization checks, leads to insufficient authorization vulnerability. Attacker's can exercise unauthorized functionality or gain access to protected data by exploiting improper authorization security policy [8]. Weak cryptographic key's generation, not rotating keys, weak algorithm usage and insecure storage are the common practices leading to insufficient authorization.

  Example: Many applications expose underlying data identifiers in a URL. For example, when accessing a medical record on a system one might have a URL such as:
  http://example.com/RecordView?id=12345
  If the application does not check that the authenticated user ID has *read rights*, then it could display data to the user that the user should not see.

- ***Integer Overflow*** An integer overflow or wraparound occurs, when the resulting value of an arithmetic operation exceeds than the original value. User-supplied inputs in an operation can also trigger an integer overflow [9]. If the result of an operation affected by integer overflow is used to control looping, make a security decision, or determine the offset or size in memory allocation, copying, concatenation etc becomes security critical.

  Example: Attackers can create serious security impacts by exploiting the vulnerability, referred to as integer overflow operations. A shift from a positive value to a negative value for calculating a purchase order total can transfer additional money to a customer's account on completion of the transactions. An attacker can transfer the money into her account from a customer's bank account by forcing a back-end system to cast a very large positive number into a negative number (signed integer).

- ***Insufficient Transport Layer Protection*** Insufficient Transport Layer Protection occurs when a web application allows the transmission or storage of sensitive or critical information in an unencrypted form [10, 11]. Lack of proper data encryption does not guarantee confidentiality, integrity, and accountability. This leads to a compromise of web application and/or steal sensitive information by exposing the communication to untrusted third parties or an attacker.

  Example: A site simply doesn't use SSL for all pages that require authentication. Attacker simply monitors network traffic (like an open wireless or their neighborhood cable modem network), and observes an authenticated victim's session cookie. Attacker then replays this cookie and takes over the user's session.

- ***Remote File Inclusion*** Remote file inclusion (RFI) is a technique used to attack web applications from a remote computer. This is a technique used combine common code into separate files, which are referenced by main web application. In this technique, a web application becomes vulnerable to remote file inclusion (RFI) attack, if the included remote file contains malicious code [12]. The application vulnerability leading to RFI is a result of insufficient validation on user input. This happens by passing the user input such as URL, parameter value, etc. into the file include commands from the HTTP request. Attackers can run any program or code through included file in the context of web application server by influencing the location of the program, which can lead to complete system compromise.

  Example: Typically, RFI attacks are performed by setting the value of a request parameter to a URL that refers to a malicious file. Consider the following PHP code:
  ```
  $incfile = $_REQUEST[‚file‛];
  include($incfile.‛.php‛);
  ```
  The first line of code extracts the value of the file parameter from the HTTP request. The second line of code dynamically sets the file name to be included, using the extracted value. If the web application does not properly sanitize the value of the file parameter (for example, by checking against a white list) this code can be exploited.
  Consider the following URL:
  http://www.target.com/vuln_page.php?file=http://www.attacker.com/malicous
  In this case, the included file name will resolve to: http://www.attacker.com/malicous.php
  Thus, the remote file will be included and any code in it will be run by the server.

- ***Format String*** A web application is vulnerable to format string attack when it uses user supplied or externally-controlled data directly as input to format strings for functions such as fprintf, printf, sprint, setproctitle, syslog, etc [13]. Attacker can alter the flow of an application, may execute arbitrary code on the server, read values off the stack, or cause segmentation faults/software crashes by using string formatting features. Buffer overflow and integer overflow attacks can also be resulted from this attack.
  Example: Let's assume that a web application has a parameter emailAddress, dictated by the user. The application prints the value of this variable by using the printf function:
  printf (emailAddress);
  If the value sent to the emailAddress parameter contains conversion characters, printf will parse the conversion characters and use the additionally supplied corresponding arguments. If no such arguments actually exist, data from the stack will be used in accordance with the order expected by the printf function. In that case, if the output stream of the printf function is presented back to the attacker, s/he may read values on the stack by sending the conversion character "%x" (one or more times).
- ***Buffer Overflow*** Copying the untrusted input to a block of memory, or buffer, beyond its boundaries, without checking the size of that input, leads to buffer overflow [14]. As a result, an attacker may be able to modify target process' address space, which can be further used to control the process execution, process crash, modification of internal variables, putting the process in an infinite loop etc.
  Example: The following code asks the user to enter their last name and then attempts to store the value entered in the last_name array.
  char last_name[20];
  printf ("Enter your last name: ");
  scanf ("%s", last_name);
  The problem with the code above is that it does not check the size of the name entered by the user. If the user enters "Very_very_long_last_name" which is 24 characters long, then a buffer overflow will occur since the array can only hold 20 characters total.
- ***Cross-site Scripting*** Cross-site scripting is an attack technique in which an attacker echoes its malicious code into a user's browser instance [15]. It occurs when a web application sends a page containing user supplied data to the browser without validation, filtering, or escaping [16]. Web technology like ASP, JSP, PHP, CGI, Perl, ASP.NET, VB.NET and C# may suffer with this vulnerability. Non-persistent or reflected, persistent or stored, and DOM based XSS are three variants of this attack. This attack can lead to hijacking the user account, delivery of fraudulent content into user's web browser etc.
  Example: The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:
  (String) page += "<input name='creditcard' type='TEXT'value='" + request.getParameter("CC") + "'>";
  The attacker modifies the 'CC' parameter in their browser to:
  '><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'.
  This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.
- ***Cross-site Request Forgery*** An attack in which a web application user is forced to send a malicious HTTP request, crafted by an attacker using web pages, email, etc. links to a targeted destination without the knowledge of a user [17]. The underlying cause is persistence and implicit trust placed in user session cookies by web application.
  Example: The application allows a user to submit a state changing request that does not include anything secret in the following manner:
  http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243
  So, the attacker constructs a request that will transfer money from the victim's account to their account, and then embeds this attack in an image request or frame stored on various sites under the attacker's control in the following way:
  <imgsrc="http://example.com/app/transferFunds?amount=1500&destinationAccount=attackersAcct#"width="0" height="0" />
  If the victim visits any of these sites while already authenticated to example.com, any forged requests will include the user's session info, inadvertently authorizing the request.
- ***Denial of Service*** When a web application does not appropriately limit the size or amount of resources that are requested or influenced by an actor, which can be used to consume more resources than anticipated Denial of Service attack occurs, resulted in the deterrence of normal user activities [18]. This attack targets to consume all or some resources of independent components such as database server, authentication server, and web server or application server of web application environment. Application-level DoS attacks much harder to detect as they emulate the same request syntax and network-level traffic characteristics as that of the legitimate clients.

Example: A web application, for a Health-Care system responsible to maintain database of patient's medical history, generates reports with medical history. For each report request, the web application queries the database to fetch all the records matching a single social security number. As the web application consists of hundreds of thousands of records stored in the database, the user normally requires to wait for three minutes to get their medical history report. During these three minutes of time, the database server's CPU reaches 60% utilization while searching for matching records. A common application layer DoS attack will send 10 simultaneous requests asking to generate a medical history report. These requests will most likely put the web site under a DoS-condition as the database server's CPU will reach 100% utilization. At this point the system will likely become inaccessible to normal user activity. In another example, an intruder will repeatedly attempt to login to a web application as user, purposely doing so with an invalid password. This process will eventually lock out the user.

- ***Brute Force*** In web applications some assets like identity, passwords, information, encryption keys, database lookup keys etc., are protected by a finite secret value. Brute Force attack is a method in which an attacker attempts to determine an unknown secret value to gain access to a protected asset by using an automated process usually trial-and-error [19]. Entropy of the value is smaller than perceived is the main fact about this attack. Log-in credentials brute forcing in web application is possible, as users usually select 'easy to memorize' words or phrases as passwords. Such an attack attempting to log-in to a system using a large list of words and phrases as potential passwords is often called a 'word list attack' or a 'dictionary attack'.

  Example: Shopping online with stolen credit cards usually requires information in addition to the credit card number, most often the CVV/SCS and/or expiration date. A fraudster may hold a stolen credit card number without the additional information. For example the CVV/CSC is not imprinted on the card or stored on the magnetic strip so it cannot be collected by mechanical or magnetic credit card swiping devices. In order to fill in the missing information the hacker can guess the missing information using a brute force technique, trying all possible values. Guessing CVV/CSC requires only 1000 or 10000 attempts as the number is only 3 or 4 digits, depending on the card type. Guessing an expiration date requires only several dozen attempts.
- ***Information Leakage*** Information Leakage is a web application weakness where a web application reveals sensitive data, such as technical details of the web application, environment, server configuration or user-specific data such as personally identifiable information, authentication credentials [20]. Attackers can use sensitive data to exploit the target web application, its hosting network, or its users. Therefore, any sensitive data or information not necessary to the functionality should be removed, limited or prevented whenever possible. Information Leakage, in its most common form, is the result of one or more of the following conditions [21]: A failure to scrub out HTML/Script comments contains sensitive information, improper application or server configurations, or differences in page responses for valid versus invalid data.

  Example: Developer inline comments within the HTML and/or script code to help facilitate the debugging or integration process during the pre-production phase can result in the leak of sensitive, contextual, information such as server directory structure, SQL query structure, and internal network information. Improper server configurations such as Software version numbers and verbose error messages can be useful to an attacker to have information about framework, languages, or pre-built functions being utilized by a web application. Pages that provide different responses based on the validity of the data can lead to Information Leakage; such as account numbers, user identifiers (Drivers license number, Passport number, Social Security Numbers, etc.) and user-specific information (passwords, sessions, addresses).
- ***Credential/Session Prediction*** Credential/Session Prediction is a technique of hijacking or impersonating a web application user [22]. When web application generates knowable values in a context requiring unpredictability, it may be achievable for an attacker to assume or guess the subsequently generated value that identifies a particular session or user, and apply this guessed value to impersonate another user or access sensitive information.

  Example: Many web applications are designed to authenticate and track a user when communication is first established. Usually a web application identifies the users by a username/password (credentials) mixture. Rather than passing these confidential credentials back and forth with each transaction/session, web application generates a unique "session ID" to identify the user session as authenticated. Subsequent communication between the user and the web application is tagged with the session ID stored in a cookie, hidden form-field, or URL as "proof" of the authenticated session. These session IDs are generated using proprietary algorithms, such as by simply incrementing static numbers or factoring in time and other computer specific variables. If an attacker can determine the algorithm used to generate the session ID, an attack can be mounted as by connecting to the web application acquiring the current session ID or by calculating or Brute Forces the next session ID or by switching the current value in the cookie/hidden form-field/URL and assumes the identity of the next user.
- ***SQL Injection*** Databases are very important support to web applications. SQL (Structured Query Language) is a specialized programming language used to communicate with the database. Applications often use user-supplied data to create SQL statements. Reprehensible creation of SQL statements by an application can lead to SQL Injection attack, which in turn make achievable for an attacker to modify the statement structure and execute

unintended and potentially hostile commands [23]. When such commands are executed, they do so under the context of the user specified by the application executing the statement. This capability allows attackers to gain control of all database resources accessible by that user, up to and including the ability to execute or modify commands or queries to steal, corrupt, or otherwise change underlying data on the hosting system [24], 25, 26].

Example: A web based authentication form might build a SQL command string using the following method:

```
SQLCommand = ‚SELECT Username FROM Users WHERE Username = ‘‛
SQLCommand = SQLComand & strUsername
SQLCommand = SQLComand & ‚’ AND Password = ‘‛
SQLCommand = SQLComand & strPassword
SQLCommand = SQLComand & ‚’‛
strAuthCheck = GetQueryResult(SQLQuery)
```

In this code, the developer combines the input from the user, strUserName and strPassword, with the logic of the SQL query. Suppose an attacker submits a login and password that looks like the following:

```
Username: foo
Password: bar’ OR ‘’=’
```

The SQL command string built from this input would be as follows:

```
SELECT Username FROM Users WHERE Username = ‘foo’
AND Password = ‘bar’ OR ‘’=’’
```

This query will return all rows from the user’s database, regardless of whether “foo” is a real user name or “bar” is a legitimate password. This is due to the OR statement appended to the WHERE clause. The comparison ‘’=’’ will always return a “true” result, making the overall WHERE clause evaluate to true for all rows in the table. If this is used for authentication purposes, the attacker will often be logged in as the first or last user in the Users table.

- ***Session Fixation*** In session fixation, an attacker have the opportunity to steal authenticated sessions, if a web application authenticates a user without first invalidating the existing session, or otherwise establishing a new user session [27]. In addition, an attacker is able to force a known session identifier on a user so that, once the user authenticates, the attacker has access to the authenticated session. In the generic exploit of session fixation vulnerabilities, an attacker creates a new session on a web application and records the associated session identifier [28]. The attacker then causes the victim to associate, and possibly authenticate, against the server using that session identifier, giving the attacker access to the user's account through the active session.

  Example: The following example shows a snippet of code from a J2EE web application where the application authenticates users with a direct post to the <code>j_security_check</code>, which typically does not invalidate the existing session before processing the login request.

  HTML *Bad Code*

  ```
  <form method="POST" action="j_security_check">
  <input type="text" name="j_username">
  <input type="text" name="j_password">
  </form>
  ```

- ***Insufficient Session Expiration*** When a web application permits an attacker to reuse old session credentials or session IDs for authorization, insufficient session expiration occurs. Since HTTP is a stateless protocol, Web applications commonly use cookies to store session IDs that uniquely identify a user from request to request. Multiple users can access the same account if, each session ID confidentiality is not maintained. Session expiration is comprised of two timeout types: inactivity and absolute. An absolute timeout is defined by the total amount of time a session can be valid without re-authentication and an inactivity timeout is the amount of idle time allowed before the session is invalidated. The lack of proper session expiration may increase the likelihood of success of certain attacks such as to steal or reuse users session identifiers that further can be used to view other users account or perform a fraudulent transaction.

  Example: A user logs onto his banks Web site to transfer money from her checking account to her savings account. Once user completes her transaction she gets distracted, forgets to sign off from her banks Web site. A second user now uses the same computer as first. Instead of using the browser to navigate to a new site, second user simply explores the browser history to return to the previous URL where first user account information was displayed. Because first user session is still active, second user can now transfer money, open new accounts, order additional credit cards, or perform any other actions banks Web site.

IV. Concrete Conclusive Findings

Based on this critical analysis of existing security vulnerabilities related to web applications, some decisive findings are obtained, which can also serve as a significant and useful effort for the development of more secure web applications. These are given as follows:

- Web applications are the portals to many corporate secrets. Whether they sit on the edge of the lawless Internet frontier or safeguard the corporate payroll, these applications are a popular target for all sorts of mischief.
- Authentication and access control parameters (submitted by a user or from application to application) very often are incorrectly used, managed and analyzed. This can create a risk of identity theft and allow access to illegitimate functions or data, also creating a risk for data confidentiality and integrity breaches.
- If the internal functions and components of an application are not sufficiently layered, or if the application displays non-secure data through error messages and comments, sensitive data may be leaked, such as client usernames and accounts, application components types and versions, SQL queries, session information, input data, cookies and even application information. This may lead to a risk of loss of confidentiality and can also provide an attacker with information that would facilitate later attacks like SQL injection.
- The most common web application security weakness is the failure to properly validate input from the user, client or environment. This weakness leads to almost all of the major vulnerabilities in applications, such as interpreter injection, locale/Unicode attacks and buffer overflows. Code Injection technique exploits vulnerabilities in input validation to run arbitrary commands in the database.
- Injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allows the complete disclosure of data on the system, destroys the data or makes it otherwise unavailable, and become unauthorized administrator of the database server.
- A major cause of web application attack is reporting mechanism of errors. A detailed error message from the web application is returned as feedback in order to understand where the problem is. Configuration errors, bugs in the code or malicious input can create some exceptional conditions. A malicious user can use these error messages to guess sensitive information related to the location and nature of the data along with valuable connection details.
- A missing understanding of the technology is in particular obvious when it comes to less common attacks like header response splitting. The web server and the HTTP protocol are frequently not sufficiently understood to realize the scope of these attacks.
- Security considerations and issues must be addressed with application requirements, design, development, deployment, and maintenance in view, not during any one of these phases in isolation. It cleverly walks through a process, prescribing actions and making suggestions along the way.
- Stakeholders are still not very cautious about security; therefore do not employ mitigations, even for existing vulnerabilities.
- A single weakness may cause much vulnerability in a web application.
- Most stakeholders agree that a better approach is needed to understand vulnerabilities and threats to develop sound practices, and use solid research practices to provide layers of defense.
- The potential for damage and the variety of vulnerabilities and threats is staggering, both from within and without. However, while many vulnerabilities and threats exist, the remedies can be crystallized into a tractable set of practices and procedures that can mitigate known vulnerabilities and threats and help to guard against the next unknown vulnerability and threat.

V. Conclusion And Future Work

Existing security vulnerabilities related to web applications were discussed. Some concrete conclusive findings are also compiled and raised. The major need is to devise a strategy to develop web applications immune to the vulnerabilities. One technique may be development of security requirements based on the vulnerability analysis and assessment. Requirements are considered as foundation stone on which the entire software is to be built and the requirements phase is the foremost opportunity for the product team to consider 'how security will be integrated into a development process'. This research work may help to provide effective and efficient ways to incorporate security *'in inception itself'* in the development life cycle enabling secured web applications.

References

[1] Web Application Security Trends Report Q3-Q4, (2008). (866) 4-CENZIC (423-6942). Retrieved February 21, 2009, from www.cenzic.com

[2] Open Web Application Security Project. (2010) The ten most critical Web application Security vulnerabilities. Retrieved March 10, 2010, from http://umn.dl.sourceforge.net/sourceforge/owasp/OWASPTopTen2010.pdf, 2010.

[3] WASC threat classification version 2.00. (2010). Retrieved January 10, 2010, from web application security consortium. http://www.webappsec.org

[4] SANS Top 20 attack vectors (http://www.sans.org/top20/) and MITRE's Common Weakness Enumeration (CWE) (http://cwe.mitre.org/).

[5] CWE/SANS Top 25 Most Dangerous Programming errors and Common Weakness Enumeration (CWE). Retrieved April 16, 2009, from http://cwe.mitre.org/top25

[6] Security focus (2008). BilboBlog admin/index.php Authentication Bypass Vulnerability. Retrieved December 2, 2009, from http://www.securityfocus.com/bid/30225.

[7] Dalton, M., Zeldovich, N., and Kozyrakis, C. (2009). Nemesis: Preventing authentication & access control vulnerabilities in web applications. In Proceedings of the *18th Annual USENIX Security Symposium*.

[8] Security focus (2008). WordPress Cookie Integrity Protection Unauthorized Access Vulnerability. Retrieved December 2, 2009, from http://www.securityfocus.com/bid/28935.

[9] Ahmad, D. (2003). The rising threat of vulnerabilities due to integer errors. IEEE Security & Privacy Magazine. 77–82.

[10] Dierks, T. and E. Rescorla, E. (2008, August). RFC 5246: The transport layer security (TLS) protocol version 1.2. IETF, Retrieved October 23, 2009, from http: //tools.ietf.org/html/rfc5246.

[11] Salowey, J., Zhou, H., Tschofenig H., and Eronen, P. (2008, January).RFC 5077: Transport layer security (TLS) session resumption without server-side state. Re-quest for comments, IETF. Retrieved January 02, 2009, from http://tools.ietf.org/html/rfc5077.

[12] Katz., O. (2009, May). Detecting Remote File Inclusion Attack. Retrieved January 05, 2010, from http://library.back2hack.cc/books/Hacking/Detecting_Remote_File_Inclusion_Attack_(Or_Katz)_en.pdf

[13] Pietraszek, T., and Berghe, C., (2005, September). Defending against injection attacks through context-sensitive string evaluation. In Proceedings of *the Recent Advances in Intrusion Detection*.

[14] Grossman, J. (2006). Myth-busting web application buffer overflows. Retrieved July 24, 2008 from http://www.whitehatsec.com/articles/mythbusting_buffer_overflow.shtml

[15] Cross-Site Scripting. Retrieved March 21, 2009,from http://projects.webappsec.org/Cross-Site-Scripting

[16] Zalewski, M. (2006, January). Cross Site Cooking. Whitepaper, Retrieved April 11, 2007, from http://www.securiteam.com/securityreviews/5EP0L2KHFG.html.

[17] Barth, A., Jackson, C., and Mitchell, C., J. (2008). Robust Defenses for Cross-Site Request Forgery. In the proceedings of *15th ACM Conference on Computer and Communications Security (CCS)*. (pp. 75-88).

[18] Srivatsa, M., Iyengar, A., Yin, J., and Liu, L. (2008, July). Mitigating application-level denial of service attacks on Web servers: A client-transparent approach. ACM Trans. Web 2(3), Article 15, (pages 1-49).

[19] Endler, D. (2001) Brute-force exploitation of web application session IDs. iDEFENSE, 2001. Retrieved March 02, 2008, from http://www.cgisecurity.com/lib/SessionIDs.pdf

[20] Tsipenyuk, K., Chess, B., and McGraw, G. (2005, November). Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors. In *Proc. NIST Workshop on Software Security Assurance Tools,Techniques, and Metrics (SSATTM)*,US Nat'l Inst. Standards and Technology. 81-84.

[21] Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. (2009). Hey, You, Get Off of My Cloud! Exploring Information Leakage in Third-Party Compute Clouds. ACM CCS.

[22] EUROSEC (2007). Web application session management. Retrieved April 08, 2008, from http://www.secologic.org/downloads/web/070212_SecologicSessionManagementSecurity.pdf

[23] Merlo, E., Letarte, D., Antoniol, G. (2007, 21-23 March). Automated Protection of PHP Applications Against SQL-injection Attacks, *IEEE CSMR 07; 11th European Conference on*. (pp. 191-202).

[24] Antunes, N., Laranjeiro, N., Vieira, M., Madeira, H. (2009, 21-25 September). Effective Detection of SQL/XPath Injection Vulnerabilities in Web Services. In proceedings of *IEEE International Conference on Services Computing SCC '09*, (pp. 260–267).

[25] Alonso, J.M., Bordon, R., Beltran, M., Guzman, A. (2008). Informatica, Mostoles; LDAP injection techniques. In proceedings of *11th IEEE Singapore International Conference on Communication Systems. ICCS 2008*, (pp. 980-986).Guangzhou.

[26] Merlo, E., Letarte, D. & Antoniol, G. (2006, October). Insider and Outsider Threat-Sensitive SQL Injection Vulnerability Analysis in PHP. IEEE WCRE 2006 13th Working Conference on. (pp. 147-156).

[27] Kolšek, M. (2002, December). Session Fixation Vulnerability in Web- Based Applications. Acros Security. Retrieved January 05, 2010, from www. acrossecurity.com/papers/session_fi xation.pdf.

[28] Zhong, W. (2008). Session Fixation. Retrieved January 08, 2009, from http://www.owasp.org/index.php/Session_Fixation.