A survey on web application vulnerabilities and countermeasures

Atefeh Torkaman
ICT Research Center
12 PUBLICATIONS **88** CITATIONS
SEE PROFILE

Marjan Bahrololum
ICT Research Institute
11 PUBLICATIONS **198** CITATIONS
SEE PROFILE

Mohammad Hesam Tadayon
ICT Research Institute
38 PUBLICATIONS **530** CITATIONS
SEE PROFILE

A Survey on Web Application Vulnerabilities and Countermeasures

Hasty Atashzar
h_atashzar@vu.iust.ac.ir
ICT Research Institute
(ITRC)

Atefeh Torkaman
torkaman@itrc.ac.ir
ICT Research Institute
(ITRC)

Marjan Bahrololum
bahrololum@itrc.ac.ir
ICT Research Institute
(ITRC)

Mohammad H. Tadayon
tadayon@itrc.ac.ir
ICT Research Institute
(ITRC)

Abstract- Security vulnerabilities continue to infect web applications, allowing attackers to access sensitive data and exploiting legitimate web sites as a hosting ground for malware. Consequently, researchers have focused on various approaches to detect and prevent critical classes of security vulnerabilities in web applications, including anomaly-based and misuse-based detection mechanisms, static and dynamic server-side and client-side web application security policy enforcement.

This paper present a survey on web application security aspects includes critical vulnerabilities, hacking tools and also approaches to improve web application and websites security level.

Keywords; Web Application Security, Web Application Vulnerabilities, Fundamental and Mitigating Solution

INTRODUCTION

Various web sites provide a variety of services on the Internet. According to "Communications Usage Trend Survey"[1], businesses grow increasingly dependent upon Web applications and social interaction through the web sites is expected to keep growing. Most organizations equip their Web sites with firewalls, Secure Sockets Layer (SSL) and network and host security devices. However the majority of attacks are on applications themselves and these technologies cannot prevent them. According to IPA (Information-Technology Promotion Agency), more than 75 Percent attack occurs to Web Application [3].

There are various approaches to increase the web applications security such as validation of input and output data, avoiding storing data that you do not need on the website and its database(s).

The Open Web Application Security Project (OWASP) provides the Top Ten project which is a list of the 10 most dangerous current Web application security flaws, along with effective methods of dealing with those flaws [1].

OWASP is an organization that provides unbiased and practical, cost-effective information about computer and Internet applications. Project members include a variety of security experts from around the world who share their knowledge of vulnerabilities, threats, attacks and countermeasures. It is an Open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted. OWASP contains:

• Application security tools and standards
• Complete books on application security testing, secure
Code development and security code review
• Standard security controls and libraries
• Local chapters worldwide
• Cutting edge research
• Extensive conferences worldwide
• Mailing lists
• And more

This paper mainly focuses on OWASP Top Ten risks and its countermeasures released in 2010. The rest of the paper is organized as follows. Section one presents the most critical vulnerabilities of the web applications. Section 2 reviews the fundamental solutions against the mentioned vulnerabilities. Web Application hacking tools and mitigating solutions is presented in section three, and finally section four reviews the paper and draws a conclusion.

I. WEB APPLICATION VULNERABILITIES

A. *Injection*

Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data [1-2, 4- 5].

B. *Cross Site Scripting (XSS)*

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute script in the victim's browser which can hijack user sessions,

[1] Communication Usage Trend Survey, Ministry of Internal Affairs and Communications (MIC)
http://www.soumu.go.jp

deface web sites, or redirect the user to malicious sites [1, 8-9].

C. Broken Authentication and Session Management

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit implementation flaws to assume other users' identities [1].

D. Insecure Direct Object References

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data [1].

E. Cross Site Request Forgery (CSRF)

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim [1, 6-7].

F. Security Misconfiguration

Security depends on having a secure configuration defined for the application, framework, web server, application server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults [1].

G. Insecure Cryptographic Storage

Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes [1].

H. Failure to Restrict URL Access

Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks each time these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway [1].

I. Insufficient Transport Layer Protection

Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly [1].

J. Unvalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages [1].

II. How To Protect Youeself

There are various mitigating solutions for Web Application vulnerabilities. Figure 1 shows many points within a system that might require protection [14]. The rest of this section presets the fundamental solution against the mentioned Web Application vulnerabilities.

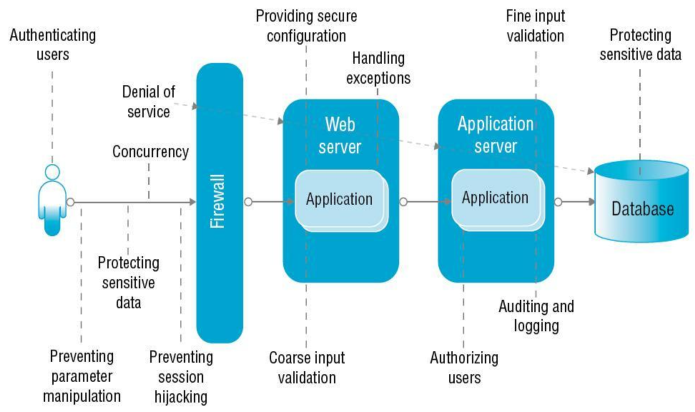

Fig 1. Mitigating Solution against Web Application Vulnerabilities [15]

A. Injection

The simplest way to protect against injection is to avoid accessing external interpreters wherever possible. For many shell commands and some system calls, there are language specific libraries that perform the same functions. Using such libraries does not involve the operating system shell interpreter, and therefore avoids a large number of problems with shell commands [1-2, 4-5].

For those calls that you must still employ, such as calls to backend databases, you must carefully validate the data provided to ensure that it does not contain any malicious content. You can also structure many requests in a manner that ensures that all supplied parameters are treated as data, rather than potentially executable content. The use of stored procedures or prepared statements will provide significant protection, ensuring that supplied input is treated as data. These measures will reduce, but not completely eliminate the risk involved in these external calls. You still must always validate such input to make sure it meets the expectations of the application in question.

Another strong protection against command injection is to ensure that the web application runs with only the privileges it absolutely needs to perform its function. So you should not run the web server as root or access a database as DBADMIN, otherwise an attacker can abuse these administrative privileges granted to the web application. Some of the J2EE environments allow the use of the Java sandbox, which can prevent the execution of system commands.

If an external command must be used, any user information that is being inserted into the command should be rigorously checked. Mechanisms should be put in place to handle any possible errors, timeouts, or blockages during the call.

All output, return codes and error codes from the call should be checked to ensure that the expected processing actually occurred. At a minimum, this will allow you to determine that something has gone wrong. Otherwise, the attack may occur and never be detected.

The OWASP Filters project is producing reusable components in several languages to help prevent many forms of injection. OWASP has also released Code Seeker, an application level firewall.

B. Cross Site Scripting (XSS)

The best way to protect a web application from XSS attacks is ensure that your application performs validation of all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed. The validation should not attempt to identify active content and remove, filter, or sanitize it. There are too many types of active content and too many ways of encoding it to get around filters for such content. We strongly recommend a 'positive' security policy that specifies what is allowed. 'Negative' or attack signature based policies are difficult to maintain and are likely to be incomplete [1, 8-9].

Encoding user supplied output can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from JavaScript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:

From:	To:
<	&lt;
>	&gt;
(	&#40;
)	&#41;
#	&#35;
&	&#38;

The OWASP Filters project is producing reusable components in several languages to help prevent many forms of parameter tampering, including the injection of XSS attacks. OWASP has also released Code Seeker, an application level firewall. In addition, the OWASP Web Goat training program has lessons on Cross Site Scripting and data encoding.

C. Broken Authentication and Session Management

Careful and proper use of custom or off the shelf authentication and session management mechanisms should significantly reduce the likelihood of a problem in this area. Defining and documenting your site's policy with respect to securely managing users is a good first step. Ensuring that your implementation consistently enforces this policy is key to having a secure and robust account and session management mechanism. Some critical area includes:

- Password Change Controls
- Password strength
- Password Storage
- Protecting Credentials in Transit
- Session ID Protection
- Account Lists
- Browser Caching
- Trust Relationship
- Backend Authentication

D. Insecure Direct Object References

The best protection is to avoid exposing direct object references to users by using an index, indirect reference map, or other indirect method that is easy to validate. If a direct object reference must be used, ensure that the user is authorized before using it. Establishing a standard way of referring to application objects is important:

- Avoid exposing private object references to users whenever possible, such as primary keys or filenames
- Validate any private object references extensively with an "accept known good" approach
- Verify authorization to all referenced objects

E. Cross Site Request Forgery (CSRF)

Applications must ensure that they are not relying on credentials or tokens that are automatically submitted by browsers. The only solution is to use a custom token that the browser will not 'remember' and then automatically include with a CSRF attack. The following strategies should be inherent in all web applications [1, 6, 8]:

- Ensure that there are no XSS vulnerabilities in your application (see Cross Site Scripting)
- Insert custom random tokens into every form and URL that will not be automatically submitted by the browser.
- For sensitive data or value transactions, re-authenticate or use transaction signing to ensure that the request is genuine. Set up external mechanisms such as e-mail or phone contact in order to verify requests or notify the user of the request
- Do not use GET requests (URLs) for sensitive data or to perform value transactions. Use only POST methods when processing sensitive data from the user. However, the URL may contain the random token as this creates a unique URL, which makes CSRF almost impossible to perform.
- POST alone is insufficient a protection.

F. Security Misconfiguration

The first step is to create a hardening guideline for your particular web server and application server configuration.

This configuration should be used on all hosts running the application and in the development environment as well. We recommend starting with any existing guidance you can find from your vendor or those available from the various existing security organizations such as OWASP, CERT, and SANS and then tailoring them for your particular needs. The hardening guideline should include the following topics [1]:

- Configuring all security mechanisms
- Turning off all unused services
- Setting up roles, permissions, and accounts, including disabling all default accounts or changing their passwords
- Logging and alerts

Once your guideline has been established, use it to configure and maintain your servers. If you have a large number of servers to configure, consider semi-automating or completely automating the configuration process. Use an existing configuration tool or develop your own. A number of such tools already exist. You can also use disk replication tools such as Ghost to take an image of an existing hardened server, and then replicate that image to new servers. Such a process may or may not work for you given your particular environment.

Keeping the server configuration secure requires vigilance. You should be sure that the responsibility for keeping the server configuration up to date is assigned to an individual or team. The maintenance process should include:

- Monitoring the latest security vulnerabilities published
- Applying the latest security patches
- Updating the security configuration guideline
- Regular vulnerability scanning from both internal and external perspectives
- Regular internal reviews of the server's security configuration as compared to your configuration guide
- Regular status reports to upper management documenting overall security posture

G. Failure to Restrict URL Access

Taking the time to plan authorization by creating a matrix to map the roles and functions of the application is a key step in achieving protection against unrestricted URL access. Web applications must enforce access control on every URL and business function. It is not sufficient to put access control into the presentation layer and leave the business logic unprotected. It is also not sufficient to check once during the process to ensure the user is authorized, and then not check again on subsequent steps. Otherwise, an attacker can simply skip the step where authorization is checked, and forge the parameter values necessary to continue on at the next step [1].

- Enabling URL access control takes some careful planning. Among the most important considerations are:
- Ensure the access control matrix is part of the business, architecture, and design of the application
- Ensure that all URLs and business functions are protected by an effective access control mechanism that verifies the user's role and entitlements prior to any processing taking place. Make sure this is done during every step of the way, not just once towards the beginning of any multi-step process
- Perform a penetration test prior to deployment or code delivery to ensure that the application cannot be misused by a motivated skilled attacker
- Pay close attention to include/library files, especially if they have an executable extension such as .php. Where feasible, they should be kept outside of the web root. They should verify that they are not being directly accessed, e.g. by checking for a constant that can only be created by the library's caller
- Do not assume that users will be unaware of special or hidden URLs or APIs. Always ensure that administrative and high privilege actions are protected
- Block access to all file types that your application should never serve. Ideally, this filter would follow the "accept known good" approach and only allow file types that you intend to serve, e.g., .html, .pdf, .php. This would then block any attempts to access log files, xml files, etc. that you never intend to serve directly.
- Keep up to date with virus protection and patches to components such as XML processors, word processors, image processors, etc., which handle user supplied data

H. Unvalidated Redirects and Forwards

The best way to find out if an application has any unvalidated redirects or forwards is to [1]:

- Review the code for all uses of redirect or forward (called a transfer in .NET). For each use, identify if the target URL is included in any parameter values. If so, verify the parameter(s) are validated to contain only an allowed destination, or element of a destination.
- Also, spider the site to see if it generates any redirects (HTTP response codes 300-307, typically 302). Look at the parameters supplied prior to the redirect to see if they appear to be a target URL or a piece of such a URL. If so, change the URL target and observe whether the site redirects to the new target.
- If code is unavailable, check all parameters to see if they look like part of a redirect or forward URL destination and test those that do.

  Safe use of redirects and forwards can be done in a number of ways:

- Simply avoid using redirects and forwards.
- If used, don't involve user parameters in calculating the destination. This can usually be done.
- If destination parameters can't be avoided, ensure that the supplied value is valid, and authorized for the user. It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of

the URL, and that server side code translate this mapping to the target URL.Applications can use ESAPI to override the send Redirect()method to make sure all redirect destinations are safe. Avoiding such flaws is extremely important as they are a favorite target of phisherstrying to gain the user's trust.

I. *Insecure Cryptographic Storage*

The most important aspect is to ensure that everything that should be encrypted is actually encrypted. Then you must ensure that the cryptography is implemented properly. As there are so many ways of using cryptography improperly, the following recommendations should be taken as part of your testing regime to help ensure secure cryptographic materials handling [1]:

- Do not create cryptographic algorithms. Only use approved public algorithms such as AES, RSA public key cryptography, and SHA-256 or better for hashing.
- Do not use weak algorithms, such as MD5 / SHA1. Favor safer alternatives, such as SHA-256 or better
- Generate keys offline and store private keys with extreme care. Never transmit private keys over insecure channels
- Ensure that infrastructure credentials such as database credentials or MQ queue access details are properly secured (via tight file system permissions and controls), or securely encrypted and not easily decrypted by local or remote users
- Ensure that encrypted data stored on disk is not easy to decrypt. For example, database encryption is worthless if the database connection pool provides unencrypted access.
- Under PCI Data Security Standard requirement 3, you must protect cardholder data. PCI DSS compliance is mandatory by 2008 for merchants and anyone else dealing with credit cards. Good practice is to never store unnecessary data, such as the magnetic stripe information or the primary account number (PAN, otherwise known as the credit card number). If you store the PAN, the DSS compliance requirements are significant. For example, you are NEVER allowed to store the CVV number (the three digit number on the rear of the card) under any circumstances. For more information, please see the PCI DSS Guidelines and implement controls as necessary.

J. *Insufficient Transport Layer Protection*

The best way to find out if an application has sufficient transport layer protection is to verify that [1]:

- SSL is used to protect all authentication related traffic.
- SSL is used for all resources on all private pages and services. This protects all data and session tokens that are exchanged. Mixed SSL on a page should be avoided since it causes user warnings in the browser, and may expose the user's session ID.
- Only strong algorithms are supported.
- All session cookies have their 'secure' flag set so the browser never transmits them in the clear.
- .The server certificate is legitimate and properly configured for that server. This includes being issued by an authorized issuer, not expired, has not been revoked, and it matches all domains the site uses.

Providing proper transport layer protection can affect the site design. It's easiest to require SSL for the entire site. For performance reasons, some sites use SSL only on private pages. Others use SSL only on 'critical' pages, but this can expose session IDs and other sensitive data. At a minimum, do all of the following:

- Require SSL for all sensitive pages. Non-SSL requests to these pages should be redirected to the SSL page.
- Set the 'secure' flag on all sensitive cookies.
- Configure your SSL provider to only support strong (e.g., FIPS 140-2 compliant) algorithms.
- Ensure your certificate is valid, not expired, not revoked, and matches all domains used by the site.
- Backend and other connections should also use SSL or other encryption technologies.

III. Web Application Hacking Tools and Mitigating Solutions

The common Website hacking tools are: [10-14]

- IISxploit
- ASP Trojan or CMD.ASP
- Metasploit (Open Source)
  - 30 Persent Exploit
- Immunity Framework
  - 60 Percent Exploit
- Core Impact
  - 90 Percent Exploit
- Web Inspect
- Shadow Security Scanner

The common Web Application hacking tools are:

- Instant Source
  View Web Site Source Carefully
- Wget
  Download website and visit offline
- Web Sleuth
  Scans site for XSS bug
- Window Bomb
  Tests website incoming query
- Burp
  Java base application for web Security
- CURL
  Scans URL for
  Format and Unicode
  Characters and structure
- Black Window
  Gets all website files up to the root directory

It's like illegal directory traversal

- Acunetix
  Best web scanner
  It's very fast and extra feature

Best References for Bug Track are:

- www.exploit-db.com
- www.security-focus.com
- www.nesuss.org
- www.bugtrack.net
- www.bug-tracking.info

The common ways for increasing website or server security are:

- Using Application Firewall
- Administration Account Renaming
- Disable Default Website
- Removal of unused application mapping
- Disable Directory Browsing
- Legal Notices
- Always update security patch
  - Service Pack
  - Hotfix
- Disable Remote Administration
- Check for Malicious input form

IV. Conclusion

In the Open System Interconnection (OSI) reference model, every message travels through seven network protocol layers. The application layer at the top includes HTTP and other protocols that transport messages with content, including HTML, XML, Simple Object Access Protocol (SOAP) and Web services.

Many hackers know how to make HTTP requests look benign at the network level, but the data within them is potentially harmful. Web Application attacks can allow unrestricted access to databases, execute arbitrary system commands and even alter Web site content.

Due to the importance of Web Application security, this paper overviewed the most critical Web Application vulnerabilities and its fundamental and mitigating solutions, and also presented the common website and web application hacking tools. By recognizing the web application vulnerabilities and solutions, servers could perfume better against the attackers and malicious activities.

References

[1] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[2] G. Wassermann, Z. Su "Sound and precise analysis of web applications for injection vulnerabilities", Proceedings of the conference on Programming language design and implementation (ACM SIGPLAN),Vol. 42 (6), 2007.

[3] http://www.ipa.go.jp/index-e.html

[4] Justin Clarke, SQL injection attacks and defense, 2009.

[5] Chip Andrews, David Litchfield, Bill Grindlay, SQL server security, McGraw-Hill Prof Med/Tech, 2003.

[6] G. Lawton "Web 2.0 Creates Security Challenges" IEEE Computer Society , Vol.40 (10) pp. 13 – 16, 2007.

[7] A. Barth, C. Jackson, J. C. Mitchell Robust defenses for cross-site request forgery, Proceedings of the 15th ACM conference on Computer and communications security (CCS '08), 2008.

[8] E. Kirda, C. Kruegel, G. Vigna, N. Jovanovic "Noxes: a client-side solution for mitigating cross-site scripting attacks" Proceedings of the 2006 ACM symposium on Applied computing(SAC '06), 2006.

[9] S. Kals, E. Kirda, C. Kruegel , N. Jovanovic, "SecuBat: a web vulnerability scanner" Proceedings of the 15th international conference on World Wide Web (WWW '06), 2006.

[10] S. McClure, S. Shah, S. Shah "Web hacking: attacks and defense" Addison-Wesley Professional, 2003.

[11] J. Mirkovic, S. Dietrich, P. Reiher "Internet denial of service: attack and defense mechanisms", Prentice Hall Professional Technical Reference, 2005.

[12] Mark Kleiman, When brute force fails: how to have less crime and less punishment, Princeton University Press, 2009.

[13] Lech Janczewski, Cyber warfare and cyber terrorism, Idea Group Inc (IGI), 2008.

[14] J. Grossman. Preventing Cross-Site Request Forgeries (CSRF) using ModSecurity [Internet]. Version 3. Knol. 2008 Aug 9. Available from: http://knol.google.com/k/jeremiah-grossman/preventing-cross-site-request-forgeries/bn6vj9pl000/17.

[15] http://www-01.ibm.com/software/tivoli/governance/security/application-security.html