Secure web applications?

Mario Konecki, Željko Hutinski, Tihomir Orehovački
Faculty of Organization and Informatics
University of Zagreb
Address: Pavlinska 2, 42 000 Varaždin, Croatia
Phone: +385 42 390 800 Fax: +385 42 213 413
E-mail: mario.konecki@foi.hr, zeljko.hutinski@foi.hr, tihomir.orehovacki@foi.hr

Abstract - Today we live in a society that we call the information society where information has become a resource of great value. Along with this kind of society web applications that provide information through various services have appeared. Web applications have been driven to the point of very high visual appearance and services quality level but the part that is still in the second plan is the question of web applications security. In this paper we will point to this problem, we will show the most common problems in web applications security today and we will give suggestions that can help in solving these problems. We will also present our research data about knowledge and common practice of companies in the matter of solving these problems.

I. INTRODUCTION

Today, there is an increasing number of web applications in all aspects of business and education. Also, web applications are becoming more and more important part of any system. The systems are connecting between each other and web applications are substituting standard desktop information systems. Also, it has become obvious that there are many flaws in web security [7] which result in malicious activities in business, education and other areas. Because of this, web applications security becomes one of the main topics. Because of constant increase of companies that are developing web applications, there is a need to find out the current situation regarding security problems that are mostly occurring as a first step to find a proper solution and educate companies about these solutions.

II. IDENTIFYING THE MOST COMMON SECURITY PROBLEMS

It is practically impossible to determine the main security problems in web applications because they depend on many factors such as particular organization of web site, specific technologies and configurations, etc. But, even so, when concentrating on a whole group of technologies and applications, 10 main security problems that occur in web applications today have been identified [2]. These problems, along with some suggestions on how to solve them are listed below. The words that could be used here instead of problem are also attack, flaw or vulnerability.

A. Unvalidated Input

Web application reactions are based upon input from HTTP request (or sometimes files) [2]. Attacker can attempt to tamper with any part of this request. Many web applications today don't even have client-side input validation. But even those applications that have this kind of validation are not secure enough. Client-side validation is effective and useful for users but it provides no real protection if there is a lack of server-side input validation. There are simple tools, even telnet, which attacker can use to intercept HTTP request and modify it or to create his own HTTP request. This kind of problem can be solved only by detailed server-side validation mechanisms. This includes proper validation of all types of input that are part of HTTP request, including URLs, forms, cookies, query strings, hidden fields, and parameters.

Validation algorithms should check [2]:

- Data type (number, string, …)
- Allowed length
- Allowed set of characters
- Parameter necessity
- Whether null is allowed or not
- Allowed parameter values
- etc.

B. Broken Access Control

The first thing that is important here is to know which types of users have certain privileges and rights. There has to be a clear access control policy [6] and documentation about this matter. A tool that can be used here to describe users and their roles is a role matrix [1;163]. Many applications today have very superficial solutions to this problem. The authorization mechanisms [1;162] are often developed ad hoc as needed and are scattered in many places. This kind of approach leaves plenty of room for mistakes and flaws. This mechanisms should as well as input validation algorithms be centralized [1;262]. Problem that deserves to be mentioned here is also remote administration. A detailed testing has to be performed regarding connections security and means that are used to ensure administrators identity.

Some things that are to be taken in consideration are especially [2]:

- Web Caching – there are tools that copy entire web sites with all of their references on users local hard drive. Developers should use HTTP headers, Meta Tags, and other means to ensure that sensitive pages are not cached.
- File access restriction – all sensitive data should be always kept on backend servers. But, there is always some sensitive data that has to be kept locally on web servers and this data should not be accessible to public. Examples of such data are configuration files, default files, some scripts, …
- Using unvalidated input to get files - using relative addresses via unvalidated input to get files that would not be accessible if required directly.
- Skipping access control checks – simply skipping the page with the security check and going directly to desired web content.

C. Broken Authentication and Session Management

Authentication means to establish the rights of an authorized user. This aspect of secure web application is very important but it is not enough. If user's credentials can be compromised due to insufficient session security then attacker can easily steal user's session and gain his identity. This then can lead to attacker's ability to view other user's data and to perform malicious activities under some other identity than his own. In web application security proper authentication and session management are crucial.

To ensure this kind of protection these aspects should be implemented and used [2]:

- Strong passwords – system should demand more complex passwords (longer, certain number of certain types of characters, etc.)
- Number of login check – system should number and log unsuccessful login attempts, block logging after certain number of attempts, perform analysis and permanently lock login from certain IP address after certain conditions are fulfilled.
- Password change control – user should be asked to enter both his old and new password when changing the password. This will prevent one to steal someone's session and to change the password without knowing the old one.
- Secure password storage – all passwords that are stored should be encrypted or hashed (can't be reversed). Passwords mustn't be hard coded.
- Credentials in transit and session protection – some sort of protection should be used, such as SSL, to protect entire session or login transaction. Just hashing the password isn't sufficient because attacker can intercept this form of password and send it again and gain access to the system never knowing the plaintext password.
- Browser caching – developer should never design application to sent authentication or session information via GET, rather via POST method [1;165]. Also all authentication pages should prevent caching so that the user can't simply load cached page in order to get into web application.
- Authentication for each application part – each application component should authenticate itself to other components unless there isn't a strong reason for authentication to be implicit. Attacker could find a weakness in one component of web application and then gain access to other components if authentication is implicit in all of web components.

D. Cross Site Scripting - XSS

Cross Site Scripting (XSS) [3] occurs when attacker uses web application to send malicious code (usually JavaScript) to a different end user. The end user's browser then executes this script because it has come from a trusted source.

There are many things that can be result of this kind of attack [2]:

- Accessing the cookie
- Accessing and stealing session
- Rewriting the content of a HTML page
- Attacking the end user's computer
- etc.

XSS attacks can be divided into 2 categories [2]:

- Stored attacks
- Reflected attacks

Stored attacks – in this case malicious code is permanently stored on target server (in a database, web page, log, etc.). Some user sends request for some of that information and malicious code is executed because it came from a trusted source and because user has requested it himself.

Reflected attacks – in this case injected code is reflected off the web server (in an error message, search result, or any response that includes some or all of the input sent to the server as a part of the request). This kind of attack is brought to user via email, forum messages, etc. User is in this case tricked into clicking on a suspicious link and then malicious code travels to a vulnerable web server which reflects it back to end user's browser which executes the code because it has come from a trusted server. An example of this kind of attacks is clicking on a suspicious link in a web forum or on a link that is a part of e-mail that is spoofed [3]. Then malicious code is executed as if it came from a trusted source and if there are some active sessions there is a possibility of stealing user credentials, or harming user's computer.

One mean of preventing this kind of attacks is to perform detailed control on all headers, queries, forms filed, etc. in

the sense of controlling that it has values that are expected rather that trying to filter malicious code (there are too many encodings, and types of active content to do this properly without flaws). Another thing that can be done here is to encode all characters (<, >, &, ;, etc. [3]) that are used in JavaScript using HTML encoding.

E. Buffer Overflow

Buffer overflow is the most common flaw/attack. It is not easy to find this kind of flaw and even if it is found, it is very hard to exploit it. The most common attack is to crash down web application or cause application to produce incorrect results. Typical attack of this kind is to send input to an application that is then stored in a stack buffer that is too small. This result is that call stack is overwritten along with the function's return pointer [2]. Then the pointer is set so that it points to the malicious code that is sent by attacker. Code that is most vulnerable to this type of attack is code that uses external data, services or properties.

F. Injection Flaws

This attack refers to injection of malicious code to web application. Web application then executes this code on database (sql queries), operating system (system calls), external programs (shell commands or command injection attack [5]), etc. Every usage of interpreter of any kind is a potential danger to web application security. Malicious code is usually sent to web application via HTTP request, embedded in parameters that are sent. A simple example of this kind of flaw is a simple guestbook where user can write some own comments. If one writes some JavaScript in this comment field it will execute in a browser when the comment is viewed. A most common injection is sql injection where attacker seeks parameters that are sent and form sql query and then insert a part of sql that will execute and reveal some information or do some damage to stored data. A way to protect web application against this kind of attack is to carefully validate all inputs, to validate input and form sql queries in stored procedures [4] or functions rather that trough parameters directly [1;218], to avoid shell commands (using libraries that provide similar functionality instead), etc.

G. Improper Error Handling

There are many typical errors than can occur while working with web applications. Improper error handling results in messages that occur in browser and reveal technical and implementation details that can help potential attacker to find flaws in the system. There should be a well organized policy for error handling. All errors have to be processed and a proper message must be presented to users. Even if this is the case, some messages can again reveal too much. For example if user tries to access a file that is restricted a message that occurs should be "File does not exist" rather than "Access to this file is denied" because the message reveals that the file really exist and that is something a web application user should not now [2].

H. Insecure Storage

When storing information many web applications use some sort of cryptographic algorithm. But there are a lot of mistakes that developer do here. For example: usage of a weak cryptographic algorithm, similar random patterns, attempt to use a new algorithm, etc [2]. Some of algorithms that are used today can be very easily decoded. For example .htaccess protection that is still used for administrators and identification on many web pages uses base64 [1;134] algorithm that can be decoded using a variety of tools available on the web. So if attacker can catch an encrypted password he can easily find out the plaintext password and enter the site. The best protection is to store a minimum of information as possible and to use well tested and proven algorithms there where it is absolutely necessary.

I. Application Denial of Service

This kind of flaw refers to attack that consumes all of web application resources [1;92] such as database connections, bandwidth, disk storage, CPU, etc. This causes the application to crash if it doesn't handle this kind of error properly or application locks and becomes unavailable to other users. There is also possibility for attacker to steal legitimate user's username and then to send invalid passwords until the user's account is blocked or he could also cause the user to get a huge number of email messages with his password because attacker has requested it again and again using legitimate user's username. This kind of flaw is very hard to overcome. It is not possible to identify user just upon IP address because it is not possible to distinguish whether repeated IP address means that there is an attacker or the legitimate user is just reloading the page [2]. The best way to protect against this kind of attack is to limit resources for authenticated users to a minimum and to avoid unauthenticated users access to any resources that are not absolutely necessary.

J. Insecure Configuration Management

Web application, as well as database, are located on a web server and web application uses services and resources that server provides (disk storage, messaging, etc.). If this server is not configured properly the whole effort to produce a well designed and secure web application is futile.

There are a number of lacks that can occur here [2]:

- Old version of software without new security patches
- Default credentials
- Default accounts
- Improper permissions policy
- Unnecessary services active
- Accessible administrators functions

Using simple scanning tools attacker is able to find these flaws. To prevent this, a well described configuration policy has to be established and a regular maintenance

(applying new patches, reading security reports, etc.) has to be performed.

III. RELATIONS BETWEEN SECURITY PROBLEMS

Some of these problems are connected and they affect each other. A table that shows these dependencies is shown in Table 1.

TABLE I
Dependencies between security problems

Security flaw	Main dependencies
Unvalidated Input	- Cross Site Scripting (XSS) - Buffer Overflow - Injection Flaws
Broken Acces Control	- Unvalidated Input - Broken Authentication and - Session Management - Cross Site Scripting (XSS) - Buffer Overflow - Injection Flaws - Improper Error Handling - Insecure Storage - Application Denial of - Service - Insecure Configuration - Management
Broken Authentication and Session Management	- Unvalidated Input - Broken Authentication and Session Management - Cross Site Scripting (XSS) - Buffer Overflow - Injection Flaws - Improper Error Handling - Insecure Storage - Application Denial of Service - Insecure Configuration Management
Cross Site Scripting (XSS)	- Buffer Overflow - Injection Flaws - Improper Error Handling - Insecure Storage - Application Denial of Service
Injection Flaws	- Broken Access Control - Broken Authentication and Session Management - Insecure Storage - Improper Error Handling - Application Denial of Service
Improper Error Handling	- Insecure Application
Insecure Storage	- Broken Access Control - Broken Authentication and Session Management
Application Denial of Service	- Insecure Application
Insecure Configuration Management	- Insecure Storage - Application Denial of Service

IV. PRELIMINARY RESEARCH OF THE CURRENT SECURITY CONDITION

In order to find out the current situation regarding web applications security a preliminary research has been conducted. Participants in this research were small and medium-size companies whose main field of interest is programming and web applications development. The list of companies that were included in this research was formed using several sources.

Sources that were used are:

- CD Business Croatia Fall 2006
- Republic of Croatia - Central Bureau of Statistics
- Croatian Chamber of Economy
- Yellow pages
- Google search results

First we got the list of all companies that are classified under Primary Business Activity Code (according to CBS) K 72 (Computer and related activities). Then we visited and examined web pages of all companies from the list and formed a new list of those companies whose primary or secondary business was web development. This procedure was necessary because detailed classification of K 72 group didn't give very good results as many companies which develop web applications didn't register under right Primary Business Activity Code.

Research data has been collected using web questionnaire. Questions were formed in such a way to give information about developer's knowledge about main security problems, their experience in dealing with main security problems and their awareness about importance of web applications security. The questionnaire was active for 2 weeks. When questionnaire was activated, a notification email was sent to all companies. After one week another email with was sent in which we expressed gratitude to all that have filled up the questionnaire and we asked all those that didn't fill up the questionnaire to do so by the end of the week. After 2 weeks the questionnaire was deactivated. After analysis of collected data several conclusions were formed. Programmers are aware that the question of web application security and users privacy is of great importance and that the lack of security policy implementation can lead to serious consequences (Figure 2). Also, programmers are convinced that they are taking every step that is necessary during their web application development to protect the users (Figure 3). But, as can be seen in Figure 1, programmers have not been dealing with the most common security threats many times and they don't have much experience in solving these problems. Based upon this, it can be said that programmers still don't take care of web application security as well as they should and that they rarely take proper preventive precautions in order to ensure web applications security and web applications users privacy. Further research will be focused on detailed analysis of current state in Croatia regarding web application security as well as on finding out which actions are needed in development and administration in order to increase security and quality of web applications.

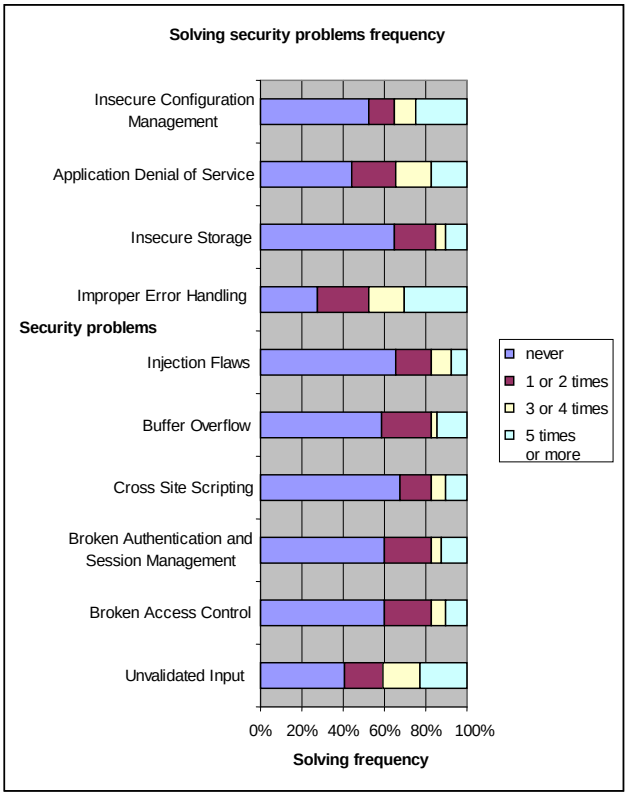

Figure 1. Solving security problems frequency

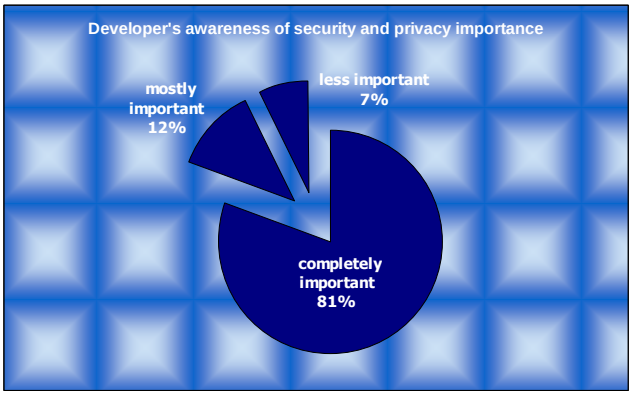

Figure 2. Developer's awareness of security and privacy importance

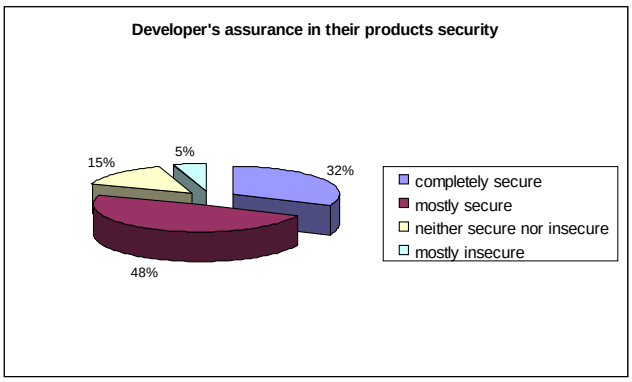

Figure 3. Developer's assurance in their products security

IV. CONCLUSION

In this paper we have pointed out to security problems that are mostly occurring. We have also presented preliminary research results about the current condition regarding web application security in Croatia. From these data we can conclude that although the question of web applications security is obviously of great importance, it is greatly undermined by developers. Developers are aware of this question but have little practical knowledge and experience in solving these problems. In our further work we will conduct a more detailed research and try to give solutions for these security issues.

REFERENCES

[1] Scambray, J., Shema, M., Sima, C., Hacking exposed web applications, 2nd Edn., McGraw-Hill/Osborne Media, 2006.

[2] OWASP - Open Web Application Security Project - the open application security community, Top Ten Project, http://www.owasp.org/index.php/OWASP_Top_Ten_Project

[3] Morgan, D., Web Injection Attacks, *Network* Security, vol.2006, no.3, March 2006, pp. 8-10., Elsevier, UK.

[4] Morgan, D., Web Injection Attacks, *Network* Security, vol.2006, no.4, April 2006, pp. 4-5., Elsevier, UK.

[5] Zhendong, S., Wassermann, G., The essence of command injection attacks in Web applications, SIGPLAN Notices, vol.41, no.1, Jan. 2006, pp. 372-382., ACM, USA.

[6] Thuraisingham B., Clifton C., Gupta A., Bertino E., Ferrari E., Directions for Web and e-commerce applications security., Proceedings Tenth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. WET ICE 2001 . IEEE Comput. Soc. 2001, pp. 200-204. Los Alamitos, CA, USA.

[7] Scott D., Sharp R., Specifying and enforcing application-level Web security policies, IEEE Transactions on Knowledge & Data Engineering. 15(4):771-783, 2003 Jul-Aug.