Memoirs of the Scientific Sections of the Romanian Academy
Tome XXXVIII, 2015

COMPUTER SCIENCE

GENERAL ASPECTS OF SOME CAUSES OF WEB APPLICATION VULNERABILITIES

MIRONELA PÎRNĂU

"Titu Maiorescu" University, Faculty of Informatics, 187 Văcăreşti Street, District 4, 004051, Bucharest, Romania, mironela.pirnau@utm.ro

Because web applications are complex software systems in constant evolution, they become real targets for hackers as they provide direct access to corporate or personal data. Web application security is supposed to represent an essential priority for organizations in order to protect sensitive customer data, or those of the employees of a company. Worldwide, there are many organizations that report the most common types of attacks on Web applications and methods for their prevention. While the paper is an overview, it puts forward several typical examples of web application vulnerabilities that are due to programming errors; these may be used by attackers to take unauthorized control over computers.

Keywords: Web attacks, Web-security, vulnerability.

1. INTRODUCTION

Web applications have known amazing development in the last years. Since the emergence of Web 2.0 and the moving to 3.0, the information sharing through social networks and the number of ways of doing business on the Internet have increased. Therefore, websites are often attacked directly. Nowadays, by developing cloud computing, a great part of Internet user activity has moved on Web [4]. The spectacular development of Web applications has been possible due to technological innovation in the field, which has transformed simple Web pages, displaying static information, into dynamic, interactive pages, which allow high interaction of user with the application. Web development has facilitated premises of occurring vulnerabilities specific to any technological product. The number of attacks is continuously increasing; many are based on vulnerabilities that have causes such as configuration errors, software bugs, developer experience, the lack of specialists in information security, and lack of information about information security.

In practice, there have been cases when the web programmer considered that a particular script can be known only by certain people, and he did not give enough attention to its security, thus vulnerabilities occurred even in very important and large information systems. There are cases when programmers have no knowledge about these vulnerabilities, especially when using a CMS-Content Management System for administering, managing and automating the content of a Website.

Many free Web platforms such as Word Press, Joomla, Magento, Drupal present vulnerabilities that are successfully exploited by attackers.

A new critical vulnerability was discovered in Drupal and Word Press by using a PHP's XML process that can allow an attacker even to disable a Website through a method of attack known as Denial of Service (DoS). Thus, the latest update to Word Press 3.9.2 is mainly a problem of XML vulnerability that could be exploited to cause a DoS (Denial of Service) attack. This vulnerability affects all previous versions of WordPress, and was reported by Nir Goldshlager of the Salesforce.com product security team [21]. Both Word Press and Drupal have released new versions to remove this problem.

To identify Web vulnerabilities, one can use specialized applications of vulnerabilities of scanner type. They have the role of a utility used to test the security of a system or a network in order to discover the weaknesses. These applications do not directly provide protection or security for a system, or a network, but gather and report information that other mechanisms, policies or applications may put into practice to provide protection against detected vulnerabilities. Many "security holes" may be where one really does not expect one to be and, generally, are found manually by the attacker, and not by means of other applications [2, 3]. No automated software can find a vulnerability as an attacker can, and for a good security of the application, it is strongly recommended to validate as many inputs as possible. Web security should take into account:

– *Client based on:* interaction with the user, personal data stored, asynchronous transfers; the existence of suspicious plug-ins/extensions; data in transit; network security (with/without wire); secure exchange of messages between different entities-repudiation of data.

– *Server that should take into consideration***:** server/Web server's security; security of applications and frameworks, data security.

– *Data in transit which refers to network security*; secure exchange of messages between various entities, non- repudiation of data.

There are a number of technical solutions for designing, construction and testing of secure Web applications, such as:

– Black Box testing tools, such as vulnerability scanners and penetration testing software,

– White Box testing tools such as static source code analyzers.

– Fuzzing – Tools used for programming testing technique.

– Web Application Firewalls (WAF) used to provide firewall protection at the web application level,

– Hack password, password strength testing instruments and implementation.

2. VULNERABILITIES AND WEB ATTACKS

Web attacks are techniques by which various sites vulnerabilities are exploited in order to extract information from databases, delete and upload files on exploited servers, to infect various victims in case of XSS worms, etc. There are many types of Web attacks that require different skills, starting with HTML and JavaScript up to SQL, Oracle, Linux and even programming languages on the server side. In terms of Web application security, there are many types of attacks. The Open Web Application Security Project (OWASP) is a community dedicated to fighting against the causes of software security issues. It is registered in the United States as a non-profit foundation whose global mission is to increase the visibility of information security issues, so that organizations and people make objective decisions on the security risks. Essentially OWASP should increase the visibility of IT security issues and lead to the improvement of software quality. Subsequently, we will discuss the following main types of attacks: SQL injection attacks (SQL Injections); session hijacking attacks (session hijacking); attacks by capturing packets transmitted over the network (Network Sniffing); attacks by injecting malicious scripts (Cross Site Scripting – XSS) [13]; gross attacks (Brute Force Attacks), Web 2.0 attacks, etc.

2.1. ATTACKS BY INJECTING MALICIOUS CODE INTO THE SQL STATEMENTS (SQL INJECTIONS)

Such an attack can exploit sensitive data from the database (using the functions insert, update and delete), can perform operations of database administration, or even execute commands on the operating system. This type of attack is also based on the data taken through forms where SQL instructions can be inserted. This attack can be a "spoof identity", which means stealing the identity of a user and execute commands with the newly acquired rights. SQL Injections is a common attack of PHP and ASP applications due to the importance of older functional interfaces. In antithesis, J2EE and ASP.NET applications are more difficult to attack because of the programmatic interfaces available [2].

Attacks by "SQL injection" add unexpected elements in SQL code leading to database manipulation several unanticipated ways. The basic modality to authenticate on a site is based on an HTML form from which user-name and password are retrieved. When a user tries to login, the user-name and password fields are sent to an ASP, JSP or PHP script [17], etc., and are available to that script through a Request Form collection. Verification and validation of the data are done through a SQL query that verifies entered data with the database in a corresponding table. SQL Injection is a vulnerability that occurs in cases data received from users is not processed correctly. As a result – potential perpetrator may change request of database, thereby stealing private data being possible.

The best strategy to defend against SQL Injections is based on implementation of powerful input data validation routines, so the attacker cannot enter data other than those required by the application. Among the specific measures that can be implemented at databases and applications level, there are the following: the use of well-defined variables and definitions of the columns in the database; assigning query results to a well-defined variable; limiting the length of data; avoid creating queries by concatenating the strings; applying separation of data and access based on role within the database.

Compared to other database, Oracle offers fewer opportunities of attack for SQL Injection and generally cope with SQL Injection attacks because there is no support for multiple SQL queries (SQL Server and PostgreSQL), no EXECUTE request (SQL Server), and no INTO OUTFILE function (MySQL) – all these being methods of exploiting vulnerabilities to SQL Injection. And even more, Oracle uses variables connected to Oracle environmental for performance reasons, so it offers the most effective protection against SQL Injection attacks [9]. It should be noted that SQL Injection vulnerabilities are very widespread.

In practice, SQL injection vulnerability can also occur when an intruder manages by injecting syntax, to modify the declaration logic so as to perform a different action. It should be noted that SQL Injection vulnerabilities are widespread and hackers can take advantage of it in order to find/extract information about the server, to get a shell performing the master.dbo.xp cmdshell procedure or perform DoS attacks. Select type applications are most vulnerable; there is the possibility that the villains try to inject UNION SELECT clauses because if successful they will get access to all system tables. Before injecting UNION SELECT the hacker will be interested to know the number of attributes for the SQL request, the type and name of each attribute and name of some system tables which is considered difficult to accomplish when no errors are displayed in the browser. To use a UNION SELECT request we must have the same number of attributes and their identification can be achieved with applications such as *http://127.0.0.1/test.php?id = -1'+UNION+SELECT+0/**.

Methods of defense against SQL injections must be based on the implementation of sub-programs for input data validation, so that the attacker may not enter data other than those required by the application [9].

2.2. CROSS SITE SCRIPTING ATTACK TYPE

Cross Site Scripting attack type (XSS) malicious scripts are injected into websites. XSS is an attack technique used in order to force a Web page to display a malicious code (usually written in HTML, JavaScript, ActiveX, Flash etc.), which is then executed in a user's browser. This type of attack does not target the web site server, this being only a host. Malware code is executed directly in the browser as the true target of the attack is the user. The hacker will use the site only to attack, and when he controls user's browser he can use it to steal different data: bank accounts, user's accounts, passwords, theft from browser's history registrations etc. A written malware in JavaScript may become resident on a Web page when it was loaded intentionally by the website/web page owner or/and it is injected in the public section of a site taking advantage of its certain vulnerabilities (permanent vulnerability) by an attacker [8, 14].

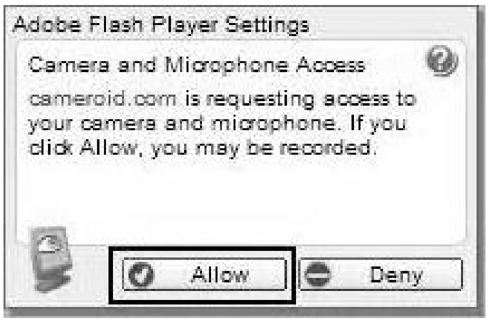
The victim may access a link especially prepared (sent by e-mail or by other methods), behind which a non-persistent XSS, or based on Document Object Model (DOM) is hidden. It is known that the XSS persistent attack or injection with HTML code do not need special links to be executed, the hacker should only add the XSS code in a side of the web page that has high potential to be accessed by users (comments on blogs, postings on forums, chats etc.). When the user accesses the bugged page, the code runs automatically and makes this type of attack more dangerous as there is no means of protection for the user, and even users who are familiar with this kind of vulnerability may be easily compromised. There are simple solutions to protect ourselves as clients, such as choosing a secure browser; use a virtual device; accessing only well-known links; personal data should not be disclosed.

Methods of preventing Cross-site scripting attacks (XSS)

Encoding input and output data has each its own pros and cons. The positive side of encoding input data offers one point of access, while encoding output data offers the possibility to cope with all uses of the text and it's positioning on the page. The negative side is that no input data encoding can stop a persistent XSS attack after being stored, and output data encoding cannot stop other types of attack such as SQL code injection, as it comes too late.

2.3. CLICK-JACKING ATTACK

Click-jacking attack is known as "UI redress attack" (i.e. GUI attack recovery) so that the attacker uses multiple transparent or opaque layers to mislead the user and make him select attacker's link. Generally this type of attack works as follows: the attacker publishes a website that contains a button with the message "You've won an iPhone" and over the button inserts an iframe with user email page and it aligns it exactly with the delete button for all messages *<iframe src = "delete.php" .../>*[15, 19]. The victim "tries" to win a phone and he deletes all his e-mails. The best known attack occurred against Adobe Flash settings page, so the attacker uploaded settings page in an invisible frame, thus misleading users and rendering web camera and microphone permission, as shown in the Figure 1, taken from *http://news.softpedia.com/news/Flash-Player-Clickjacking-Flaw-Allows-Hackers-to- Hijack-Your-Webcam-360980.shtml#sgal_0.*


Adobe Flash Player Settings

Camera and Microphone Access

cameroid.com is requesting access to your camera and microphone. If you click Allow, you may be recorded.

Fig. 1. Flash Player Click-jacking flaw allows hackers to hijack the web-cam.

To prevent **click-jacking** attacks it is recommended a defensive code using graphical interface so that the application be on the topmost level, any subsequent frame inserted occurs in the application as well as their browsers, by corresponding headers to prevent loading frames from other domains. In practice the most popular defense against click-jacking is to include a "frame-breaker" script (switch frames) in each page, which must not use frames. Furthermore the following example should not be used as such, *<script>if (top! = self) top.location.href = self. location.href</script>*, because the script is trying to prevent loading of the current page in a <frame> or <iframe> or forcing the parent window to load the current URL. Unfortunately this script is defeated by various methods. Some techniques of frames cancellation brows towards the correct page assigning a value to the parent location attribute. It works correctly if the victim's page is a single frame. However the attacker can load the page in a frame that is in another frame (Double Framing). The script for frame blocking, trying to modify parent location attribute, determines modern browsers to consider this as a security breach due to browsing policy.

2.4. BRUTE FORCE ATTACK

Brute Force Attack is a common attack of web applications which attempts to guess the password of an account. It attempts to find systematically the password using all possible letters, numbers and symbols until it succeeds. The positive fact is that, depending on the complexity and length of the password, the attack may take even years, as there are trillions of possible combinations. To get success as fast as possible, databases with most likely passwords (called dictionaries) are used. This attack puts at risk a user account and loads server traffic. Although these attacks are easy to spot they are not easy to prevent. Attackers can easily open a session of authentication to the application through the so-called "proxy server". As each query comes from another area of IP, these attacks cannot be blocked by restricting IP [10, 11].

All these attacks attempt to gain control over user accounts (deleting and/or changing) or on the server. The approach used here uses a combination of data filtering, encryption and other methods to secure the Web application and the hardware as much as possible [12].

2.5. SESSION HIJACKING ATTACK

Session hijacking attack consists in the exploitation of web session control mechanisms. Since HTTP communication uses multiple connection types, the application needs a way to recognize the connection of each user. The most used method is to create a "token" that a server sends to the browser if the authentication was successful. A token consists of a variable text string that can be used in several ways: in the URL address, in a cookie, in the header of application response, etc. Such an attack compromises the session token by stealing or guessing a valid token so as to gain access to the server. Pin modes that can compromise a session are different: intuition of a valid token, stealing a valid token of another user, client-browser attacks, etc.

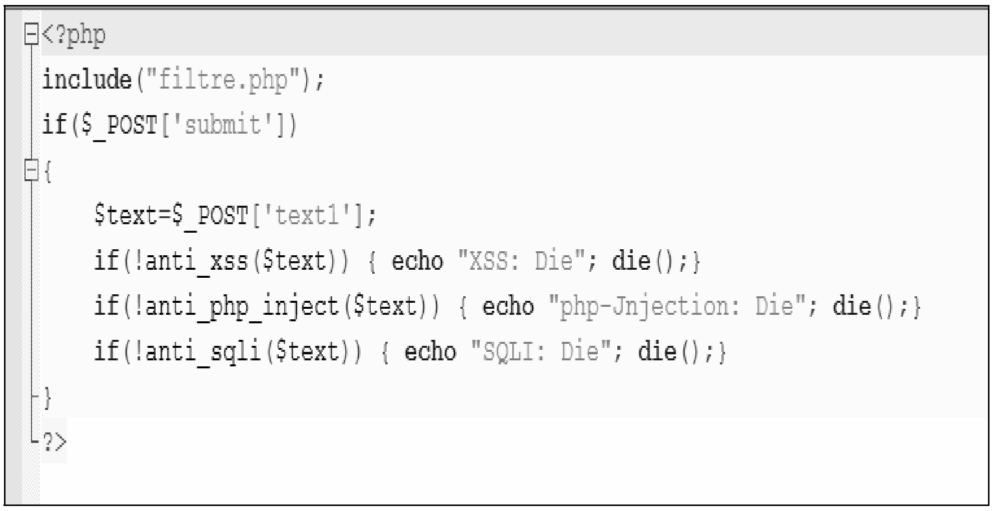
2.6. NETWORK SNIFFING ATTACK

Network Sniffing attack (Eavesdropping) is a network-level attack consisting of capturing packets of information sent from one computer to another and reading this data in search of sensitive data (passwords, email, token, etc.). The attack is done with special devices called “network sniffers”. They intercept data packets and, depending on their quality, analyze the information by means of protocol decoders and reassemble back the pack. This type of attack is passive and very difficult to detect [1].

3. EXAMPLES OF SOLUTIONS FOR PROTECTION

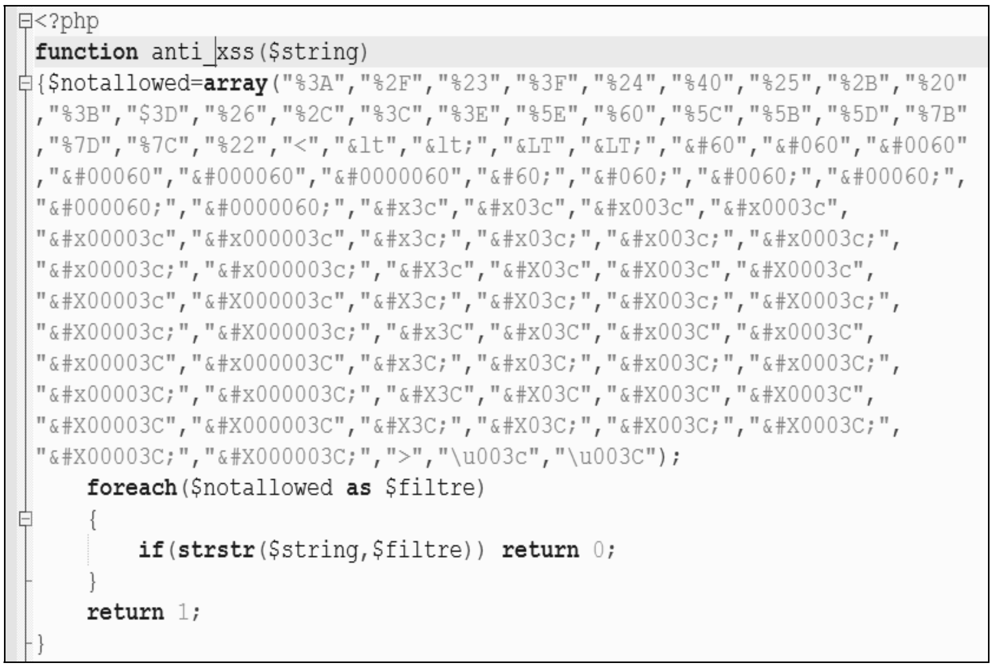
ENCODING AND VALIDATION OF INPUT DATA

Data filtering operation may produce unexpected results if ins and outs from the application are not carefully monitored. Without using other methods, the only use of filtering can create new risks by facilitating new types of attack [5, 7]. It is very important the order of the applied filters and to establish how they interact with each other (Fig. 2). If one takes data from a HTML form, filters can be applied to it so as to prevent a possible XSS attack (Fig. 3), PHP Injection, and for SQLI orders (Fig. 4).



```
<?php
include("filtre.php");
if($_POST['submit'])
{
    $text=$_POST['text1'];
    if(!anti_xss($text)) { echo "XSS: Die"; die();}
    if(!anti_php_inject($text)) { echo "php-Jnjection: Die"; die();}
    if(!anti_sqli($text)) { echo "SQLI: Die"; die();}
}
?>
```

Fig. 2. Input taking <form action = ‘test.php’ method = ‘post’>.



```
<?php
function anti_xss($string)
{$notallowed=array("%3A","%2F","%23","%3F","%24","%40","%25","%2B","%20"
,"%3B","$3D","%26","%2C","%3C","%3E","%5E","%60","%5C","%5B","%5D","%7B"
,"%7D","%7C","%22","<","&lt","&lt;","&LT","&LT;","&#60","&#060","&#0060"
,"&#00060","&#000060","&#0000060","&#60;","&#060;","&#0060;","&#00060;",
"&#000060;","&#0000060;","&#x3c","&#x03c","&#x003c","&#x0003c",
"&#x00003c","&#x000003c","&#x3c;","&#x03c;","&#x003c;","&#x0003c;",
"&#x00003c;","&#x000003c;","&#X3c","&#X03c","&#X003c","&#X0003c",
"&#X00003c","&#X000003c","&#X3c;","&#X03c;","&#X003c;","&#X0003c;",
"&#X00003c;","&#X000003c;","&#x3C","&#x03C","&#x003C","&#x0003C",
"&#x00003C","&#x000003C","&#x3C;","&#x03C;","&#x003C;","&#x0003C;",
"&#x00003C;","&#x000003C;","&#X3C","&#X03C","&#X003C","&#X0003C",
"&#X00003C","&#X000003C","&#X3C;","&#X03C;","&#X003C;","&#X0003C;",
"&#X00003C;","&#X000003C;",">","\u003c","\u003C");
    foreach($notallowed as $filtre)
    {
        if(strstr($string,$filtre)) return 0;
    }
    return 1;
}
```

Fig. 3. Filter for XSS.

For avoiding a malicious SQL injection attack one should not accept in the inputs strings like AUX, CLOCK$, COM1- COM8, CON, CONFIG$, LPT1 - LPT8, NUL and PRN. There should also be rejected certain special characters or SQL statements as we can see in function anti_sqli in Figure 5 [16]. If the user will use parameterized input with stored procedures she should notice that they may be susceptible to SQL injection if they use unfiltered input. Other methods of filtering inputs use queries that may identify procedures containing these statements. For instance checks for maximum four spaces after EXECUTE and EXEC [18]:

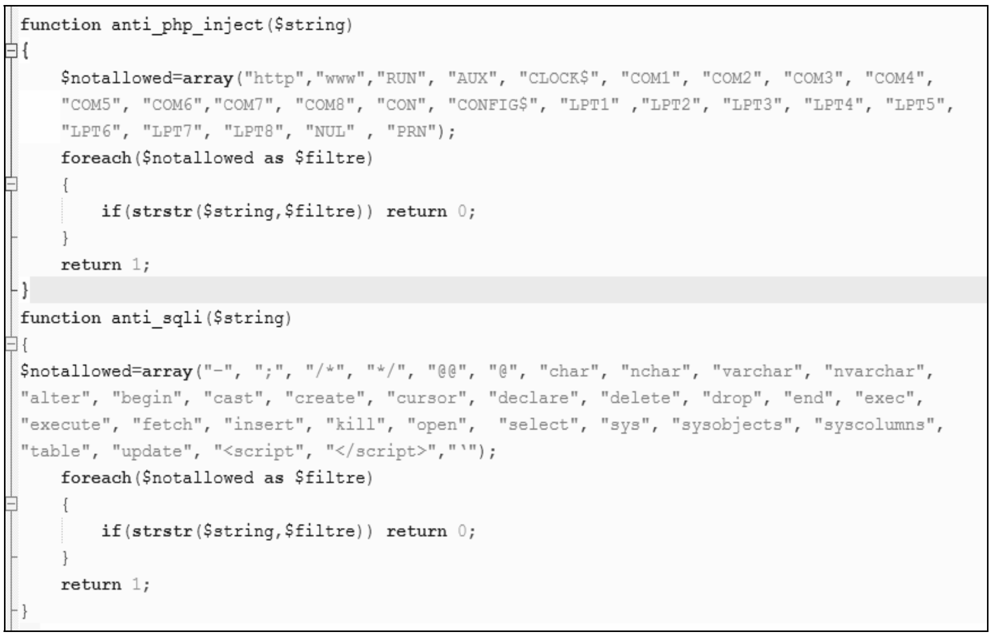
```
select object_name(id) from table_value where upper(text) like
'%execute (%'

or upper(text) like '%execute (%' or upper(text) like '%execute (%'

or upper(text) like '%execute (%' or upper(text) like '%exec (%'

or upper(text) like '%exec (%' or upper(text) like '%exec (%'

or upper(text) like '%exec (%' or upper(text) like
'%sp_executesql%'
```



```
function anti_php_inject($string)
{
    $notallowed=array("http","www","RUN", "AUX", "CLOCK$", "COM1", "COM2", "COM3", "COM4",
    "COM5", "COM6","COM7", "COM8", "CON", "CONFIG$", "LPT1" ,"LPT2", "LPT3", "LPT4", "LPT5",
    "LPT6", "LPT7", "LPT8", "NUL" , "PRN");
    foreach($notallowed as $filtre)
    {
        if(strstr($string,$filtre)) return 0;
    }
    return 1;
}
function anti_sqli($string)
{
$notallowed=array("-", ";", "/*", "*/", "@@", "@", "char", "nchar", "varchar", "nvarchar",
"alter", "begin", "cast", "create", "cursor", "declare", "delete", "drop", "end", "exec",
"execute", "fetch", "insert", "kill", "open",  "select", "sys", "sysobjects", "syscolumns",
"table", "update", "<script", "</script>","`");
    foreach($notallowed as $filtre)
    {
        if(strstr($string,$filtre)) return 0;
    }
    return 1;
}
```

Fig. 4. Filter for PHP-Injection and SQLI.

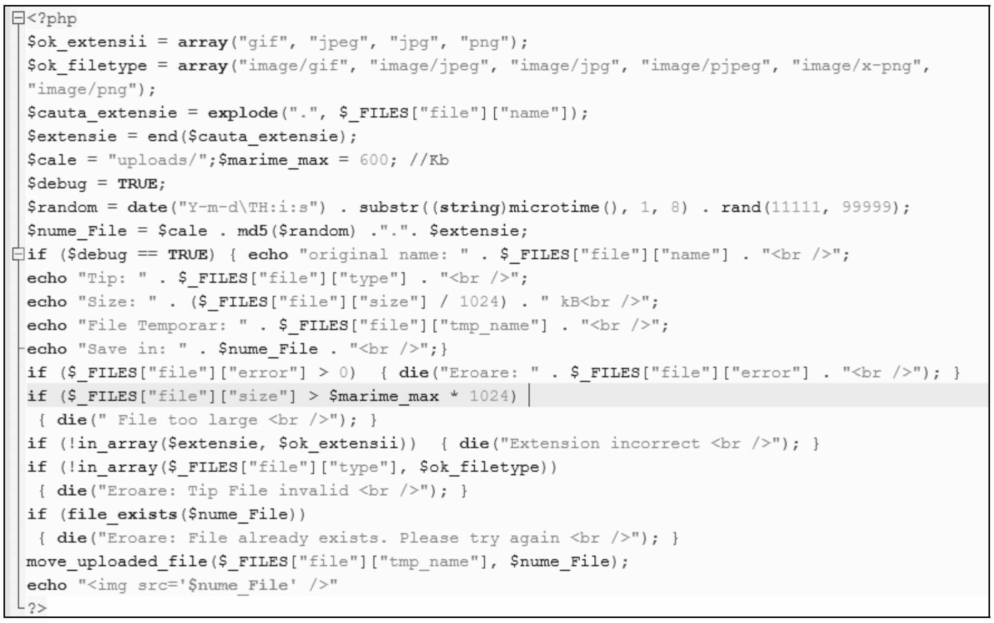
The filtering examples above point out that if a Web site uses heavily a dynamically built SQL, it is recommended to analyze and eliminate all input values that can be unsafe, but this may be not sufficient [1, 9].

SECURING VULNERABILITIES IN FILE UPLOAD SCRIPTS

These vulnerabilities occur because of the programmer or of the script manager. Prevention of this vulnerability requires checking the extension of uploaded files so as to not allow the upload of PHP files or files with PHP content. Also, the files must not exceed a maximum size. An example of script for filtering extensions is shown in Figure 5 and is in PHP. The form from which the file is uploaded may look as follows:

```
<form enctype = "multipart/form-data” action = "upload.php” method
= "POST” />
<input type = "hidden” name = "MAX_FILE_SIZE” value = "100000” />
Selectati documentul:
<input name = "uploadedfile” type = "file” accept = "image/*” />
<input type = "submit” value = "Upload File” />
</form>
```

Uploaded files represent a significant risk to applications and the essential role of many attacks is to get a possibility that via a file sent to the server, the system may be attacked. Developers of Web applications should not rely only on validation of browser but take into account, before uploading file on server, the list allowed extensions, file name length, limiting the maximum size of the file, eventuality to build an algorithm to determine the file-names (for instance a file name can be a MD5 [20] hash followed by the current date – see Fig. 5) etc.



```php
<?php
$ok_extensii = array("gif", "jpeg", "jpg", "png");
$ok_filetype = array("image/gif", "image/jpeg", "image/jpg", "image/pjpeg", "image/x-png",
"image/png");
$cauta_extensie = explode(".", $_FILES["file"]["name"]);
$extensie = end($cauta_extensie);
$cale = "uploads/";$marime_max = 600; //Kb
$debug = TRUE;
$random = date("Y-m-d\TH:i:s") . substr((string)microtime(), 1, 8) . rand(11111, 99999);
$nume_File = $cale . md5($random) .".". $extensie;
if ($debug == TRUE) { echo "original name: " . $_FILES["file"]["name"] . "<br />";
echo "Tip: " . $_FILES["file"]["type"] . "<br />";
echo "Size: " . ($_FILES["file"]["size"] / 1024) . " kB<br />";
echo "File Temporar: " . $_FILES["file"]["tmp_name"] . "<br />";
echo "Save in: " . $nume_File . "<br />";}
if ($_FILES["file"]["error"] > 0)  { die("Eroare: " . $_FILES["file"]["error"] . "<br />"); }
if ($_FILES["file"]["size"] > $marime_max * 1024)
 { die(" File too large <br />"); }
if (!in_array($extensie, $ok_extensii))  { die("Extension incorrect <br />"); }
if (!in_array($_FILES["file"]["type"], $ok_filetype))
 { die("Eroare: Tip File invalid <br />"); }
if (file_exists($nume_File))
 { die("Eroare: File already exists. Please try again <br />"); }
move_uploaded_file($_FILES["file"]["tmp_name"], $nume_File);
echo "<img src='$nume_File' />"
?>
```

Fig. 5. Validation for input type = "file".

ENCODING AND VALIDATION OF INPUT DATA

This technique can make a single input data point for all encodings. It may protect against XSS vulnerabilities, but also from injections with SQL code and order injections, which can be verified before storing information in data base. It cannot stop persistent XSS attacks after storing.

Upon registration of a user one must check all possible types of attack on the application code. The most important aspect of a system based on the accounts is how we protect the user's password (XSS attack). The database on the server can be broken, so we need to protect passwords through a protection system like the one known as "salted password hashing". Hash encrypting algorithms operate in only one direction. They take an amount of information and transform it into a fixed-length string, a process that cannot be reversed. Moreover, if the input data are altered even in a small way, the result will be completely different.

This type of protection is very good because we want to save passwords in the database in an encrypted form for which there is no decryption method, and which will help us to verify the password introduced by the user. However, the solution is not perfect. With brute force attacks and databases of possible passwords (dictionaries) one can quite easily get access to the user account. This can happen because the passwords are encrypted in the same way every time, resulting in the same password hash. If two users have the same password, the hash password will be the same. To prevent this, it is recommended to add a random text string to a password (jump) and only then we create the hash password. To validate the user we will have to save only hash password and random string (jump) in the data base [6]. Once the checks on the form are done, an encrypted jump string (SHA512) is created and added to the password introduced by the user, and then encrypted as hash. This hash password and random jump text are then saved in the database. However, for the security of information sent over the Internet to register or login, the form should be validated locally using JavaScript and the password should not be sent on the Internet. It is good that the input data to be retrieved using a secure browser updated daily and customized with security settings to reduce as much as possible other vulnerabilities.

4. CONCLUSIONS

In this paper we examined the multitude of situations where vulnerabilities of web applications occur and explained that it is easier to predict vulnerabilities than to handle them. To avoid the most common vulnerabilities, those who write the code should be well informed about these vulnerabilities and the ways to prevent them. Many vulnerabilities can be prevented by the algorithm that is to be written. The main specific measures that should be implemented in the Web application interacting with database take into account the use of certain variables; assigning query results only to well defined variables; limiting the length of data; avoiding query by concatenating the strings; applying separation of data and access based on roles within the database. There is not an application fully free of vulnerabilities, but there can be reduced flaws.

Attacks on Web applications are due to many causes, but the most common are: vulnerabilities that allow a remote cracker to control sensitive data on a system; mistakes in programming/implementation/configuration of applications or services; lack of patches or open mail relay, errors in the configuration of the SMTP mail server that allows any user to send mails (vulnerability often used by spammers); vulnerabilities in system accounts such as common passwords, blank passwords or default (root, admin). In order to have safe Web applications, they should be thoroughly tested to discover weaknesses and then rectify in time so as possible damage be minimum.

REFERENCES

1. BINBIN QU., *Design of automatic vulnerability detection system for Web application program*, 4th IEEE International Conference on Software Engineering and Service Science, Beijing, 2013, 89–92.
2. CLARKE J., *SQL Injection Attacks and Defense,* 2nd Edition**,** Syngress-Elsevier, Waltham, MA 02451, USA, 2012,
3. DUBOIS P., *MySQL Cookbook: Solutions for Database Developers and Administrators*, O'Reilly Media, USA, 2014.
4. GARFINKEL S., SPAFFORD G., *Web Security, Privacy & Commerce*, O'Reilly Media, USA, 2001**.**
5. GAUTHIER F., MERLO E., *Fast Detection of Access Control Vulnerabilities in PHP Applications*, 19th Working Conference on Reverse Engineering (WCRE), Canada, 2012, 247–256.
6. HUANG Y.-W., YU F., HANG C., TSAI C.-H., LEE D.T., KUO S.-Y., *Verifying web applications using bounded model checking*, Dependable Systems and Networks, Florence, Italy, 2004, 199–208.
7. HUANG Y-W, YU F., HANG C., TSAI C.-H., LEE D.-T., KUO S.-Y, *Securing web application code by static analysis and runtime protection*. Proceedings of the *13th* International Conference on World Wide Web, New York, NY, USA, 2004.
8. JOHNS M., *On JavaScript Malware and related threats*, Journal in Computer Virology, 2008, **4**(3), 161–178.
9. KIEZUN A., GUO PH., JAYARAMAN K., ERNST M.D., *Automatic creation of SQL Injection and cross-site scripting attacks***,** IEEE Computer Society Washington, DC, USA 2009.
10. LI QIAN, JIAHUA WAN, LU CHEN, XIUMING CHEN, *Complete Web Security Testing Methods and Recommendations*, International Conference on Computer Sciences and Applications (CSA), Wuhan – China, 2013, 86–89.
11. PFLEEGER C.P., PFLEEGER S.L., *Security in Computing,* 4th Edition, Prentice Hall, Boston – MA, USA, 2006.
12. TEODORESCU H.-N., IFTENE E.-F., *Efficiency of a Combined Protection Method against Correlation Power Analysis Side-Attacks on Microsystems*, Int J Comput Commun, February 2014, **9**(*1*), 79–84.
13. WASSERMANN G., SU Z., *Static detection of cross-site scripting vulnerabilities*, Proceedings of the 30th International Conference on Software Engineering, New York, NY, USA: ACM, 2008, 171–180.
14. XIE Y., AIKEN A., *Static Detection of Security Vulnerabilities in Scripting Languages*, 15th USENIX Security Symposium, CA, USA, 2006, **15**(13), 179–192.
15. ⁂ Clickjacking Defense Cheat Sheet – OWASP/ https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet/(Accessed 18 July 2014)
16. ⁂ How to configure URLScan 3.0 to mitigate SQL Injection Attacks/ http://blogs.msdn.com/b/webtopics/archive/2008/10/15/how-to-configure-urlscan-3-0-to-mitigate-sql-injection-attacks.aspx/ (Accessed 1 September 2014)
17. ⁂ PHP Security Consortium: PHP Security Guide/ https://www.owasp.org/index.php/Unrestricted_File_Upload/(Accessed 20 August 2014)
18. SQL Injection - TechNet – Microsoft/ http://technet.microsoft.com/en-us/library/ms161953%28v = sql.105%29.aspx / (Accessed 15 August 2014)
19. ⁂ The OWASP Foundation OWASP Busting Frame Busting / www.owasp.org/images/0/0e/OWASP_AppSec_Research_2010_Busting_Frame_Busting_by_Rydstedt.pdf/(Accessed 11 July 2014)
20. ⁂ Unrestricted File Upload – OWASP / https://www.owasp.org/index.php/Unrestricted_File_Upload/(Accessed 20 August 2014)
21. ⁂WordPress › WordPress 3.9.2 Security Release/ https://wordpress.org/news/2014/08/wordpress-3-9-2/ (Accessed 2 October 2014)

Received October 29, 2014
Revised December 22, 2014; May 31, 2015