One approach to the testing of security of proposed database application software

SINIŠA S. ILIĆ, LJUBOMIR LAZIĆ, PETAR SPALEVIĆ
Department of computer sciences
University of Priština
Knjaza Miloša 7, 38220 Kosovska Mitrovica
SERBIA
sinisa.ilic@pr.ac.rs, http://www.ftn.pr.ac.rs

Abstract: - This paper presents the concept of database configuration and development considering security issues especially when connected to internet. Regardless of precautions on security voulnerabilities implemented on other levels of database environment, such as: network, operating system, client application, it is important to protect database itself by avoiding well known database security issues. In order to prove that proposed configuration has a high level of security protection, security testing has to be performed. The overall goal of security testing is to reduce vulnerabilities within a software system and we have proposed testing methodology including code review and vulnerability assessment that represent the most widespread of best practices for software security assurance.

Key-Words: - database configuration, security vulnerabilities, testing of security

1 Introduction

In order to deliver functionally reliable software systems of higher quality within the time and budget, potential vulnerabilities in security are mostly neglected. Software developers try to design database and application according to the functional specifications without security considerations, because the primary goal in software development is to meet functional requirements. Since security is a non-functional requirement, it is not the utmost concern for system developers. In the software development process there is not a "recipe" how to treat potential issues related to security. Often it is very hard even to foresee them. As the programming of procedures for handling of most frequent security risks is very time consuming compared to the time needed for programming the functional requirements, it becomes clear why lack of security occur.

Security flaws depend also on architecture of the software deployment. If the software is reliable in some environment, the same might be un-reliable in another environment. That is the reason why a lot of security threats must be handled in a software development even if sometimes it looks needless. In the multi-level environment one can count on high possibility that attacker will not pass security barriers of the levels that are between an attacker and software system. But it also might happen that security vulnerability of one level leads to jeopardizing of security of other levels.

Often the validation of input data must be performed both: in client application (by using validation scripts or functions) and in database by using constraints, triggers or stored procedures. By checking data on both levels, the chance that data with wrong values will be inserted in database is minimal.

Designing a database that will achieve all security requirements is very difficult, since a database system processes large amount of data in complex ways. The result is that most conventional database systems have leaks that attacker can use to penetrate the database.

Before any software system is going to be deployed, testing must be performed. Unfortunately, most cases in test scenarios of database systems check if functional requirements are met, security tests are performed poorly. It is almost practice that testing in general is performed under immense time pressure. Often tight-schedule testers are restricted to test software functionality only, and lack other quality aspects. On the other side, functional testing certifies whether or not the system behaves as intended, and security testing aims at uncovering unspecified behavior within the system. The task of security testing is very hard considering that security threats are rapidly growing and it is difficult to simulate these conditions.

Fortunately, there are plenty of free tools that can be used in testing of security vulnerabilities. Some

of them are part of OS (like ssh, netcat, wireshark, etc.); others are parts of web browser (Firefox, Firebug add-on, View Source Chart add-on, Tamper Data add-on, etc.) and others. There are also some free (HP Scrawlr, SQLiX) as well as commercial (HP WebInspect, IBM Rational AppScan) tools that can automatically test for security flaws.

Testing of security in these applications poses grave challenges to the engineers.

2 Related Work

Security is a process of maintaining an acceptable level of risk. So, security is a process, not a final state, i.e. security is not the final product.

When system of data security is considered, it requires a systematic approach that must include: employees, physical and technical security, confidentiality of business procedures, network security (LAN and / or WAN), responsibility and sanctioning [1]. This is a very general approach to the security, as most people treat security issues related to configuration of firewalls. But, in order to protect collected, stored and transferred data from: physical hazards, hardware failures, operational errors, software defects, theft of media and other various manipulations, every decent company has to obtain data security policy. The job of managers who are responsible for security at all levels, is to analyze: the potential risks, their effects, the impacts if the risk happened, probability of risk appearance, handling methods to be taken to ensure that risk will not happen and who is responsible to take appropriate measures.

In order to achieve high-rated secure systems, a flawless penetration testing must be performed [2]. Penetration testing cannot prove or even demonstrate that a system is flawless. It can place a reasonable bound on the knowledge and work factor required for a penetrator to succeed. It is not recommended to perform this type of testing on production systems because of possible system halts, dumps or crashes. Unfortunately, results of penetration tests show that most conventional database systems have leaks that attacker can use to penetrate the database.

There are several ways proposed in papers to handle security risks. We can group them into following classes: Intrusion detection techniques, database protection techniques and automation of security testing.

An intrusion can be defined as “any set of actions that attempts to compromise the integrity, confidentiality, or availability of a resource. In intrusion detection process, some activity can be treated as suspicious according to the predefined criteria (malicious behavior that deviates from established normal patterns). The problem with current state-of-the-art is to reduce false negative and false positive rate. At the same time, a real-time intrusion detection system should be considered. It is difficult to achieve both.

Malicious activities can be detected by Support Vector Machines (SVM) - one of the most successful classification algorithms in the data mining area [3]. The limits of SVM use is its long training time and authors presented a study for enhancing the training time of SVM, specifically when dealing with large data sets, using hierarchical clustering analysis.

As “malicious activities” need not to come from an attacker, there must be ensured enough time for security engineer(s) to investigate if activity is really dangerous or not. During that time, a potential attacker must be convinced that his/her transaction is successful. The method for solving this problem is to build as many “copies” of database as many suspicious users are connected to database [4]. If a “suspicious” user is proven to be an attacker, it is blocked, and if it is not, there is built Merging Algorithm that replaces its trustworthy version (values) with its suspicious version (values), and then removes the suspicious version.

Many software systems are designed to be Web-based and available to the public via the Internet. In this way they become exposed to a variety of Web-based attacks. Up to 78% of recently reported vulnerabilities affected Web applications [5]. Two classes of attacks are particularly common and damaging. In SQL injection, the attacker executes malicious database statements by exploiting inadequate validation of data flowing from the user to the database. In cross-site scripting, the attacker executes malicious code on the victim’s machine by exploiting inadequate validation of data flowing to statements that output HTML.

In order to protect data integrity and confidentiality different validation techniques are proposed. Each data item that user sends from client machine to server is checked and filtered before it comes to database. This process is called sanitizing.

The input data can be parsed into segments according to the key words of SQL syntax and compared with expected structure [6]. If expected structure is different than actual one (parsed from input field), the input data is not transferred to database.

Another approach is to filter input data by using regular expression search tool [7]. The regular expression search tool enables finding of templates

ISBN: 978-1-61804-019-0

within the text string (by: key words, text in brackets, tags, templates of numbers etc.). The key words of SQL statements, tautologies, URL encoding and other encoding characters can be put at regular expression search strings. These malicious characters/words can be removed from input fields and not transferred to database. The similar approach can be achieved with java function HashMap that can be used to convert encoded characters (that attacker can use instead of normal characters in order to skip some standard validation techniques) to standard ones in minimal time [8].

As stated in introduction, the security of database systems often depends of security of other layers between an attacker and database. The better security is applied to database itself the less likely to increase the risk of intrusion into the database.

Data in database might be encrypted and even in the case of penetration, an attacker could not use such data [9]. Data can be divided as public, classified and private. Classified data can be accessed by authorized users (for example accountant can see the payroll data for all employees) and private data can be accessed only by users who inserted it (each user has own private data). Data is protected by symmetrical keys (generated by system) and distributed in certified envelopes by using public/private key pairs to users. Many alternative encryption techniques might be used as well by considering confidentiality and performance [10].

The protection of database can be also embedded within the database. One can use embedding policies into the database itself and enable these policies to block every attempt to compromise the state of the database, or to alter its configuration in a way that contradicts what has been established and fed into the policy by the system owner [11]. It is achieved by using user access to database and by using stored procedures even for changing parameters of database by database administrators (not the database owner). When a power user or a hacker initiates an attempt to change security configurations (database parameters), the request goes through a process of verification before it can be processed. This step is carried out by database stored procedures that have built-in logic for checking the request against the policies. If the request complies with the set policies that govern its scope of applicability, then the request is applied. Then, the database system tables/views are updated to reflect the change and an audit trail is recorded. Otherwise, the request is rejected and the system owner is alerted, the user notified, and an audit trail is recorded.

By using modified MAC (Mandatory Access Control), RBAC (Role-Based Access Control) and DAC (Discretionary Access Control) models it is possible to design a database security system that can individually control user access to data groups of various sizes and is suitable for the situation where the user's access privilege to arbitrary data is changed frequently [12]. In these models user can access any data that has lower or equal security levels, and that is accessible by the roles to which the user is assigned.

It is not enough to build modules for detection of intrusions and protection of database. The built models should be tested on security risks. Often it is very time consuming to build test scenarios and to perform them.

There are two approaches in testing on security vulnerabilities: by analyzing source code statically and dynamically and by treating the system to be tested as black box.

Source code of an application can be analyzed in a way to find a data set with which a program execution can reach a specific node, referred to as the target node in the Control Flow Graph (CFG), which represents the objective of the test analysis. This is the chaining approach [13]. The security chaining approach would result in the generation of only those event sequences that are needed to be executed for security vulnerabilities to be detected. If the approach cannot find a solution to traverse a tree, then this tree is considered impossible, and hence no vulnerability is detected or reported, eliminating all false positives. By knowing the source code, the tree is built and a set of data is automatically generated in order to find vulnerable nodes.

The automatic testing tool for checking the SQL injection vulnerability – Ardilla [14] tracks the flow of generated tainted data through the database. When tainted data is stored in the database, the taint information is stored with it. When the data is later retrieved from the database, it is marked with the stored taint. This precision makes Ardilla able to accurately detect second order Cross-Site Scripting attacks. Ardilla uses also SQL Injection attack patterns developed by security professionals and detects attacks by looking for differences in the way the program behaves when run on two inputs: one innocuous and the other potentially malicious.

3 Problem formulation

According to the listed papers, there are a lot of security vulnerabilities that can harm running database systems. An attacker will try to get

ISBN: 978-1-61804-019-0

administrative rights on database. Our goal is to try to protect database as much as we can by self-protecting at database level.

If presume that an attacker will try to execute malicious code from client application logging as guest, he/she will probably try to perform the following attacks.

3.1 SQL Injection attacks

SQL Injection vulnerability results from the application's use of user input in constructing database statements. If user input will be assigned to variables *input1* and *input2* and if the contents of that variable will be used to dynamically create SQL statement *query* like:

query = "SELECT UserId FROM Users WHERE uname ='" +*input1*+"' and passw= '"+*input2*+""""

it is obvious that if *input1*=admin, and *input2*=trust, *query* will be:

query = "SELECT UserId FROM Users WHERE uname='admin' and passw='trust'"

and query will be sent to database to be executed. If user is allowed to put anything to appropriate web form field (our variable input), he/she might try to put the following [15]:

1) *input1*=anyuser, *input2*=pass' or '1'='1
2) *input1*= anyuser' or 1=1 LIMIT 1;#, *input2*=pass
3) *input1*=anyuser, *input2*=pass' AND 1=0 UNION SELECT (case+when+(USER()='root@localhost')+then+ 2+else+1+end) AND '1'='1

In the first case the *query* will consists of SQL command that will return UserId of all users to an attacker (because of tautology '1'='1'). In the second case SQL command will return first user (that is almost always admin) because the key word LIMIT will filter the first of all users, and # (hash sign) will comment all remaining characters in *query* string. In the third case SQL command will consist of two sub-commands, the first will return nothing (because of 1=0 that is false) and the second one (after the key words UNION ALL) will return 2 if the database user is root@localhost and 1 if the database user is not root@localhost (the particular case is taken for MySQL Server, but it is very similar for other DBMS).

One can conclude by himself about similar SQL commands creation by using of different values for *input* variable, especially by knowing that many DBMS offer to user functionalities of reading data from files in OS and writing data to files.

3.2 Cross-Site Scripting (XSS) attacks

It is a type of security vulnerability found in web applications that enables attackers to inject client-side script into web pages viewed by other users. Let assume that users can submit to the database some inputs in text form. If the text submitted is saved in database, it can be viewed by other users that have permission to look at it.

An attacker might use the chance to put malicious web code into that field and submit it to database. When other user opens web page with that content, the script will be executed at the user's browser and potentially harm user's computer or compromise some user's private data.

3.3 Eavesdropping and password theft

Sometimes an attacker will try to listen to the network traffic between the server and client that tries to log to the system by using system credentials (username, password). If the credentials are not encrypted, an attacker will try to log to the web application by using stolen credentials. But if credentials are encrypted, an attacker will try to log by repeating stolen network packets – the same that authorized user sent to the database for authentication.

4 Proposal for Problem Solution

As it was mentioned, the security of database cannot rely on security of other levels of software system. Database must be as much self-protected as possible. We have created the database with the following specifications:

1) users that have permissions and privileges to access to database application are not controlled through User table, but they are registered as regular database users without any administrative roles,
2) there is no one regular user (except the db owner) that can have any (select, insert, update, delete) permission on any table,
3) the only way of viewing, inserting, modifying or deleting data is through the stored procedures, where all users have permission of executing procedures,
4) in stored procedures there is no dynamically created queries that can be executed through execute_sql(string) commands; queries are built with stored procedures parameters (see Figure 1)

ISBN: 978-1-61804-019-0

5) stored procedures have built-in logic for checking the user rights and permissions on different business functionalities (through the tables with encrypted attributes),
6) stored procedures have built-in logic for checking the properties of parameters sent to them (width, black list words, black list encoded characters, etc.),
7) calls to stored procedures with suspicious parameters' values are logged to special tables,
8) local database administrators may change user rights and permissions only through stored procedures that identify if request comes from LAN.

In addition to the database specifications, for the web-based application it is very important that credentials should be transferred from user to server by using Secure Sockets Layer (SSL) certified by trusted party.

Let's explain proposed database specifications which are not difficult to implement.

Usually application is connected to database through the super power database user, and application users are then controlled through table "Users" indirectly, which can be easily accessed from application. By implementing specification number 1) connection between application and database is established for the particular database user. Through the built in DBMS security any regular database user will have only explicit rights to execute stored procedure (specification 3) and will have no access directly to database tables (specification 2). It means that no one user can create any kind of query – neither static neither dynamic. The business logic is implemented through SQL commands stored in procedures and the only communication between application and database is in calling of stored procedures and submitting the values of stored procedure's parameters (specification number 4).

In special tables that can be accessed only through stored procedures, user rights are defined in encrypted format. Before execution of the procedure's business functionality, the database user's permission is checked against the defined business role. If user does not have permission of some business functionality, the error message will be generated (specification number 5). Procedures are designed to check (validate) values of parameters, and if an attacker tries to insert suspicious code in these parameters, the user, user ip address, computer name, procedure's name and parameters values are recorded in log table and alarm is raised (specifications 6 and 7). The design of stored procedure is presented at Figure 1.

```
Create procedure sp_UpdateEmpAge(par1 int, par2 int)
BEGIN
  if not fnCheckPermission(user, UpdateEmpAge)
  begin
    error_message('User is not allowed to perform this action')
    return
  end

  if not fnValidateParam(par1, int, UpdateEmpAge)
    return

  if not fnValidateParam(par2, int, UpdateEmpAge)
    return

  UPDATE employees
  SET age = par2
  WHERE employeeid = par1

END
```

Figure 1 – The design of Database stored procedure

Administrators have the right to change permissions of users on business logic – not to the database tables. It would be dangerous if an attacker could identify somehow as administrator. That is the reason why administrators can identify themselves only from LAN (specification 8).

When user connects to the database application through Secure Sockets Layer (SSL), the credentials are encrypted, so an attacker cannot see the clear text of username and password. If an attacker tries to save the data transferred through network from user to server in moment of user logging and retransmit it (identifying himself as user who has successfully logged), SSL protocol defeats this attack by using a nonce, a one-time unique number - connection id that cannot be repeated (server will recognize that it is repeated network block and will ignore it).

Testing of security vulnerabilities of such database can be performed in the following way:

- analyze stored procedure's source code and consider eventual Fault propagation analysis by using White Box technique,
- install tools for security testing, as mentioned in introduction, for automated vulnerability scanning following Black Box technique,
- in both mentioned techniques test majority patterns of SQL injection and XSS [16],
- test for vulnerabilities by identifying as regular database user and try to get administrative permissions,
- test if regular user can take control on OS by generating some exclusive SQL queries that can access (configuration) files in OS.

ISBN: 978-1-61804-019-0

Generally, it is a good practice to create test plans with precise description of each test scenario and to associate mitigation methods if some test scenario failed. In this case we cannot offer test templates but attack scenarios mentioned. Unfortunately, security testing is motivated by probing assumptions and areas of particular complexity to determine how a program can be broken

5 Conclusion

To protect software with database from an attacker, it is needed to continuously monitor and investigate all published cases of software vulnerabilities. One of the ways to improve security of database is to embed a shield on input data that might cause the potential security flaws, by using stored procedures with built-in validations on input data and to protect transfer of credentials for user authentication through standard web encryption techniques. As the full security cannot be guaranteed, the systematic approach to the testing of such database must be performed by using attack scenarios. We have prepared described database model in three architectures: Java-Oracle, VB .NET – MS SQL Server and PHP-MySQL in order to test proposed model.

Acknowledgement

This work has been done within the project 'Optimal Software Quality Management Framework', supported in part by the Ministry of Science and Technological Development of the Republic of Serbia under Project No.TR-35026.

References:

[1] S. Obradović, S.S.Ilić, V. Marković, *Responsibility of management related to data security*, Proceedings of international conference UNITECH, Gabrovo, Bulgaria 2009.

[2] C. Weissman, *Penetration Testing, Essay 11*, Department of Defense, "Trusted Computer System Evaluation Criteria," DoD 5200.28-STD, December 1985 (The Orange Book). www.acsac.org/secshelf/book001/11.pdf

[3] L. Khan, M. Awad, B. Thuraisingham, *A new intrusion detection system using support vector machines and hierarchical clustering*, The VLDB Journal, Volume 16, 2007. pp 507-521.

[4] Peng Liu, *DAIS*: *A Real-time Data Attack Isolation System for Commercial Database Applications*, Proceedings of 17th Annual Computer Security Applications Conference - ACSAC 2001. pp. 219-229

[5] Cenzic. Application security trends report Q1 2009. http://www.cenzic.com/downloads/Cenzic_AppSecTrends_Q1-Q2-2009.pdf

[6] G. T. Buehrer, B. W. Weide, P. A. G. Sivilotti, *Using Parse Tree Validation to Prevent SQL Injection Attacks*, Fifth International Workshop on Software Engineering and Middleware - SEM 2005 September 2005 Lisbon, Portugal

[7] K.V.N.Sunitha and M.Sridevi, *Automated Detection System for SQL Injection Attack*, International Journal of Computer Science and Security (IJCSS), Volume (4): Issue (4), pp. 426-435

[8] E. Adi, I. Salomo, *Detect and Sanitise Encoded Cross-Site Scripting and SQL Injection Attack Strings Using a Hash Map*, Australian Information Security Management Conference, 2010.

[9] Z. Yang, S. Sesay, J. Chen and Du Xu, *A Secure Database Encryption Scheme*, American Journal of Applied Sciences 1 (4): 327-331, 2004

[10] S. Burnett, S. Paine, RSA Security's *Official Guide to Cryptography*, Osborne/McGraw-Hill, 2001

[11] G. Jabbour, D. A. Menasce, Policy-*Based Enforcement of Database Security Configuration through Autonomic Capabilities*, Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems ICAS'08

[12] Min A Jeong, Jung-Ja Kim, Y. Won, *A Flexible Database Security System Using multiple Access Control Policies*, International Conference on Database and Expert Systems Applications - DEXA 2003, LNCS 2736, pp. 876–885, 2003.

[13] A. Hanna, H. Z. Ling, J. Furlong, M. Debbabi, *Towards Automation of Testing High-Level Security Properties*, The Eighth IAPR International Workshop on Document Analysis Systems DAS 2008, Nara, Japan

[14] A. Kieyzun, P. J. Guo, K. Jayaraman, M. D. Ernst, "Automatic Creation of SQL Injection and Cross-Site Scripting Attacks", Proceedings of the 31st International Conference on Software Engineering ICSE '09

[15] Justin Clarke, *SQL Injection attacks and defence*, Syngress Publishing, Inc. Elsevier, Inc., 2009

[16] P. Hope, B. Walther, Web Security Testing Cookbook, 1st Edition, O'Reilly Media, Inc., 2008.