SQL INJECTION TEST SYSTEM FOR STUDENTS

Dragi Zlatkovski
Faculty of Computer Science, UGD
Štip, Macedonia

Nataša Šuteva, Aleksandra Mileva
Faculty of Computer Science, UGD
Štip, Macedonia

ABSTRACT

We present a new web application as a vulnerable testing system for SQL injection attacks. Its purpose is to give opportunity to students of Computer security courses, to explore the nature of these modern attacks, to see how hackers exploit them and to learn how to protect their databases from them. Deployment of SQL injection attack on real web applications is illegal and it is subject to prosecution by law. With this software, we protect our students, and in the same time, we offer them real hacking for ethical goal.

I. INTRODUCTION

Web applications present to any user a system-independent interface for dynamically generated contents, usually through a web browser. They relieve the developer of the responsibility of building a client for a specific type of computer or a specific operating system. If you log on and check your bank account, or buy some items, or visit online auctions, you are using a web application. Many web applications interact with the back-end database, in a way, that they use user's input to construct SQL queries as a collection of statements, by which they can modify the structure of the database or/and manipulate its contents. It is usually a three-tiered architecture, consisting of a web-browser, an application server, and a back-end database server (Fig. 1).

Because of their nature, web applications are one of the most interesting targets of the contemporary hackers. According to the Open Web Application Security Project, in the OWASP Top Ten project [1], injection flaws, including SQL injection, are the most common and serious web application vulnerability. *SQL injection attack (SQLIA)* is a kind of injection attack that occurs when an attacker, through specially crafted input, causes given web application to generate malicious SQL query and send it to the database. Usually behind this attack, there is no or insufficient validation of user input. Additionally, there are many types of SQLIAs and their variations, and researches and practitioners are unaware of their diversity. Several comprehensive surveys of known SQLIAs and their countermeasures are given in [2, 3, 4, 5].

Any serious Computer security course must include web security as integral part. It is not enough to describe known web attacks to students, it is better for them to look and fill these attacks for real. Deployment of these attacks on real web applications is illegal and it is subject to prosecution by law. In the light of that, we offer to our students a real hacking on our test system, specially created for them. This test system is specially designed for several SQLIAs. For testing other web attacks (excluding SQLIA) legally, you can use Google Gruyere codelab [6]. Remember to read the fine print: "**WARNING:** Accessing or attacking a computer system without authorization is illegal in many jurisdictions. While doing this codelab, you are specifically granted authorization to attack the Gruyere application as directed. You may not attack Gruyere in ways other than described in this codelab, nor may you attack App Engine directly or any other Google service. You should use what you learn from the codelab to make your own applications more secure. You should not use it to attack any applications other than your own, and only do that with permission from the appropriate authorities (e.g., your company's security team)." Similar warning can be apply to our test system too. Another web page for legally trying SQLIA is [7].

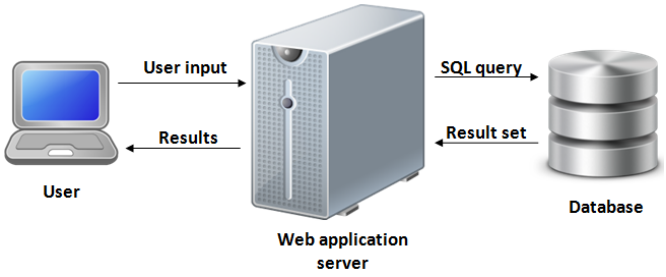

Figure 1: A three-tiered architecture for web application

The rest of the paper is organized as follows: Section 2 provides more information on SQL injection; Section 3 gives an explanation of the test system. Finally, we provide summary and conclusions in Section 4.

II. SQL INJECTION

SQL injection refers to a class of injection attacks in which the attacker manipulates the user input, generating malicious SQL query, which functions differently than the programmer intended. The basic concept behind this attack has been described for the first time by the hacker Jeff Forristal[1] [8]. Several papers describe SQL injection vulnerabilities, frequently exploited by attackers [9, 10, 11, 12].

The attack can be described in several steps. First, the attacker provides an abnormal input, which web application screens and accepts as legitimate. With this input, web application generates malicious SQL query and passes the query to the database. The database executes this query and produces some result as extraction of data, table deletion or addition, record deletion, modification or insertion, performing database finger-print, determining database schema, authentication bypassing, escalation of privileges, execution of a denial-of-service attack, evading detection or execution of remote commands (listed as attack intent in [2]). Sometimes, the database can return error messages, which can

[1] Jeff Forristal, also known as RFP and rain.forest.puppy, is a hacker currently employed at Zscaler Cloud Security.

also assist the attacker. In situation where the attacker has no knowledge of the query syntax or database's structure, forcing an exception may reveal more details about them, for example the table or its field names and types [9, 10]. Identifying the query syntax is mandatory for correctly exploitation of a given flaw. The back-end database can contain sensitive consumer or user information, so the resulting security violations can include identity theft, loss of confidential information, and fraud.

The main reason of SQL injection vulnerabilities is insufficient validation of user-supplied input.

As countermeasures, one can use escaping single quotes, limiting the input character length, filtering the exception messages, taking user input from predefined choices, using bind variables mechanism (or prepared statements, which implement bind variables in Java), using parameterized statements etc. Instead directly passing the user input in SQL queries, parameterized statements must be used, or else user input should be sanitized or filtered [3].

III. SQL INJECTION TEST SYSTEM

A. *Overview*

The SQL injection test system shows how SQL injection vulnerabilities can be exploited and how to defend against these attacks. The best way to learn things is by doing, so students will get a chance to do some real penetration testing, actually exploiting a real application.

The SQL injection test system by itself is a web application with back-end database in MySQL and PHP for server-side scripting. It consists of two parts, one is for students to perform the tests, and the other is for the teacher to see which students participated in which tests. The second part is still in development. Database schema is given on Fig. 2.

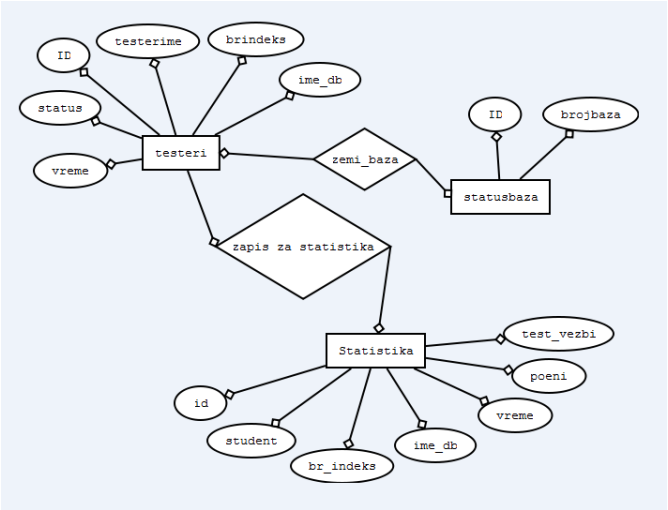

Figure 2: Database schema

The student side consists of three main parts. The first part is description of the different SQLIAs and the reasons why they happen, the second part is the test system itself and third part is the description of different countermeasures that students need to be familiar with.

B. *Usability*

As a first step, the student needs to register by the full name and index number. This information is needed to monitor the student's activity during interaction with the application, and to obtain statistics for the teacher. Next, the student performs logging, and after that, a two-hour session is created and a separate database is installed, by PHP script. The student's database carries the name "testdatabase_DBN" where DBN is a number that is auto incremental modulo 1801. The possibility to reach this number of different databases for two-hour period is very small in recent environment. If an error occurred while creating the database, the student gets a proper notification and is returned to the login page. Activity diagram is given on Fig. 3.

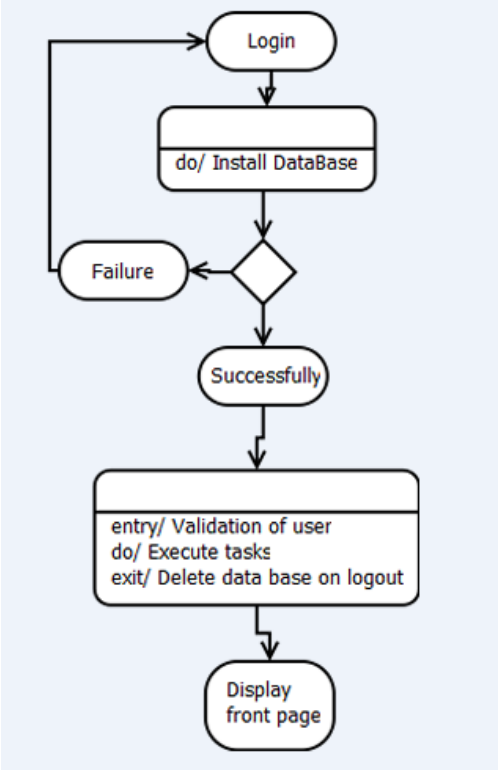

Figure 3: Activity diagram

If necessary tables in the database are successfully installed and filled with data, then the student is redirected in the area where he can start pentesting. The reason for existing of separate user copies of test database is twofold: the nature of some SQLIA, for example modification of database's schema and "sandboxing" from the other instances, so user instance won't be affected by anyone else using our test system.

The schema of student database is given on Fig. 4. The database is very elementary, intentionally. We need the table "Korisnici" (Users) for testing authentication SQLIA.

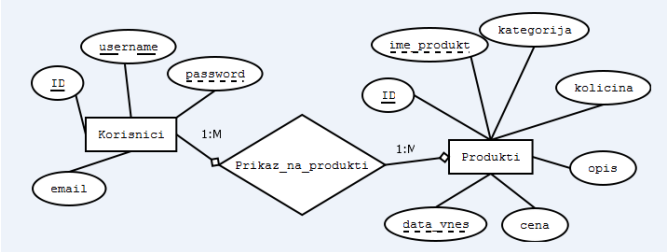

Figure 4: Student database schema

Logging out of the application, causes session termination and deletion of the user database.

C. Tests

The codelab is organized by types of SQLIA you can perform. There are 7 different exercises, with instructions that need to be followed (Fig. 5). They are:

1. Authentication bypassing – by gaining access as the first user in the table, without knowing any user names or passwords;
2. Finding a password - if you know the name of the password table and a user account;
3. Finding a user account;
4. Performing database fingerprinting;
5. Finding a database schema – for example, finding the names of tables;
6. Modification of the database schema – for example, table deletion;
7. Record manipulation – deletion, modification or insertion of a given record.

Figure 5: SQL Injection test system

Students need to follow the instructions patiently, because some answers are obtained as yes/no questions on log/not log base with many trials. The instructions are very clear, unambiguous and easy to understand, and the process of testing is not a problem to be performed. Figure 6 shows the screen shot of one exercise.

Figure 6: Authentication bypassing exercise

IV. Conclusions

This SQL injection test system can be useful tool as a codelab for students to learn different SQL injection attacks. For now, this system is still in development, and it exists only in the faculty's private VLAN, so only our students can use it. We plan to make this application more functional, and after we exceed and solve all the security problems, we will put this web application for public use.

References

[1] OWASP Foundation, Top Ten Most Critical Web Application Vulnerabilities, 2010. https://www.owasp.org/index.php/Top_10_2010.

[2] W. G. J. Halfond, J. Viegas, A. Orso, "A classification of SQL injection attacks and counter measures," Proceedings of the IEEE International Symposium on Secure Software Engineering (ISSSE 2006), Arlington, USA, March 2006.

[3] S-T Sun, T. H. Wei, S. Liu, S. Lau, "Classification of SQL injection attacks,", 2007.

[4] N. Patel, F. Mohammed, S. Sony, "SQL injection attacks: techniques and protection mechanisms", International Journal on Computer Science and Engineering, Vol. 3, No. 1, 2011, 199-203.

[5] D. A. Kindy, A. K. Pathan, "A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques," IEEE 15th International Symposium on Consumer Electronics (ISCE), 14-17 June 2011, 468-471.

[6] Google Gruyere CodeLab [Online]. Available: https://google-gruyere.appspot.com/

[7] http://sqlzoo.net/hack/

[8] rain.forest.puppy: NT Web Technology Vulnerabilities. Phrack Magazine Volume 8, Issue 54. December 25, 1998.

[9] C. Anley, "Advanced SQL injection in SQL server application," Technical report, NGSSoftware Insight Security Research (NISR), 2002. [Online]. Available: http://www.nextgenss.com/papers/advanced sql injection.pdf

[10] C. Anley, "(more)advanced SQL injection in SQL server application," Technical report, NGSSoftware Insight Security Research (NISR), 2002. [Online]. Available: http://www.nextgenss.com/papers/more advanced sql injection.pdf

[11] C. Cerrudo, "Manipulating microsoft SQL server using SQL injection," Technical report, Application Security, Inc., 2003. [Online]. Available: http://www.appsecinc.com/presentations/Manipulating SQL Server Using SQL Injection.pdf

[12] B. D. A. Guimarães, "SQL injection: Not only AND 1=1," 2nd Digital Security Forum in Lisbon, Portugal, June 2009.