UDC 004.9

Detection and correction of database schema integrity violation based on initialization scripts

O. Kondratiuk[1], M. Kolomitsev[1]

[1]*National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute»,*
Institute of Physics and Technology

Abstract

We present a new methodology for detecting and correcting database schema integrity violations. This technique uses initialization scripts, their pre-processing to compare with the current database schema. The result of the work is a prototype of the software product.

Keywords: database, database scheme, sql, initialization script

1. Introduction

One of the most important criteria for information system reliability is system database security. Attacks directed at it are in most cases critical, as they can partially or completely disrupt the system. One of the easiest and most common attacks is a code injection attack. SQL injection accounts for a significant portion (approximately 25 %) of all network attacks. Therefore, there is a need for modern methods of protection against this type of attacks, as well as, in the case of unauthorized changes - detection methods and methods for responding to changes in the database, so the work is devoted to the analysis of the effects of attacks and methods of detecting and responding to attack data.

2. Related Work

SQL injection is an attack that is a type of code injection attack. This exploit is performed by adding SQL code to the user input to gain access to unauthorized resources. SQLIAs can occur when a query is created by combining user-entered data, such as data entered into a web form, with unintentional data, including URL data (Uniform Resource Locator), data obtained from cookies, etc., without proper verification. The members of Rain Forest Puppy, the black-hat community, were the first to ever publish information about SQLIA in their article NT Web Technology Vulnerabilities [1]. SQL injection is one of the favorite attacks for many cybercriminals because it can be remotely executed. Commercially available vulnerability detection tools are also available to attackers, and using these resources, an attacker can find security and web vulnerabilities in a split second. SQL is a very flexible language and these attacks can be extremely secretive and can run through firewalls and intrusion prevention systems without much effort [2].

2.1. Types of SQL injection attacks

The types of SQL injection attacks are described in [3] and are listed below:

1) Classic SQL Injection
Classic SQL Injection is simple and easy to use. Allows the attacker to attack the database and immediately see the result of the attack. Lately, it is rare. The ability to perform classic SQL injection greatly simplifies the retrieval of useful information. Attacking using the classic SQL Injection technique takes place using a union statement or using a SQL query (semicolon). But not always SQL Injection vulnerabilities can be exploited in a similar way. In such cases they resort to the technique of exploiting the vulnerability by the "blind" method.
2) Blind SQL Injection
Blind SQL Injection - appears when a vulnerable request is some logic of the program, but does not allow any data to be returned to the returned Web page by the application. Blind SQL injection can be compared to the classic SQL statement implementation technique. Similar to the classic technique of exploiting such vulnerabilities, blind SQL Injection allows you to write and read files, retrieve data from a table, but only read in this case is character. The classic technique of exploiting such vulnerabilities is based on the use of true / false logical expressions. If the expression is true, then the Web application will return one content, and if the expression is false, the other. Relying on the differences of output for true and false constructions in the query, it becomes possible to search by character any data in the table or in the file.
3) Error-based SQL Injection
Error-based SQL Injection is a slightly more complex and time consuming type of attack that allows you to retrieve information about the entire database and the data stored in the database based on the DBMS errors. It is used if someone in

a hurry forgot to disable the error output. The essence of Error-based is that we can extract the information we need from the query by looking at the errors of the functions being called. One such feature in the MySQL database is extractvalue(). Error-based SQL Injection is the fastest operating technique for blind SQL injection. The essence of this technique is that different DBMS, with certain incorrect SQL statements, can put in the error message different requested data (for example, database version). This technique can be used when any SQL statement processing error committed in the DBMS is returned back by a vulnerable application.

4) Boolean-based SQL Injection
Boolean-based SQL Injection is one of the "blind" injections. The essence of the attack is to add a special subquery to the vulnerable parameter that the database will respond to either True or, unexpectedly, False. The attack does not allow the attacker to display all database data immediately to the attacker, but allows to get the contents of the database by flipping through the parameters one at a time, although this will require a time interval comparable to the contents of the database.
5) Time-based SQL Injection
Time-based SQL Injection is the next "blind" injection. In this case, the attacker adds a subquery, which causes the database to slow down or pause under some conditions. Thus, by comparing the response time to True and False queries, the attacker can retrieve the entire contents of the database character by character, but it will take longer than in the case of a Boolean-based attack.
6) Out-of-band SQL Injection
Out-of-band SQL Injection is a rare type. An attack can only be successful in certain circumstances, for example, if the database server can generate infrequent DNS or HTTP requests. Like Blind SQL, it allows you to collect character information about the data stored there.

2.2. SQL injection attack vectors

SQL injection attacks have the following vectors: [4]:

1) SQL manipulations
In this attack, the expression following the word "where" is manipulated to create behavior unexpected by the database programmer (for example, constructing an expression where the union statement can provide access to data that the user should not have access to.)
2) Code implementation
In this attack, the new SQL statement integrates with the previously presented SQL statement (for example, by adding an execute statement at the end of the general statement.) The limitation of this type of SQLIA is that the database must support multiple SQL statements on request.
3) Function call injection
It is a secondary injection of attack in which an attacker uses built-in database functions to invoke SQLIA, which manipulates data according to the needs of the attacker.
4) Buffer Overflow Attack
In this case, the data entered as input will greatly exceed the memory limits of the planned storage space. It will overwrite data pointers and can also be used to point to an executable file, forcing the system to execute any file that is intended by the attacker.

3. Existing methods of protection against database threats

There are many ways to defend against attacks, but not one of them provides a 100% security guarantee. After all, there is counteraction to every action, and a particularly interested seasoned hacker will probably find a way to access your database. However, let's look at the main ones: [5]

1) Creating a less privileged user
In most cases, visitors do not need to delete or update information. Let's imagine an online store. The user can request (SELECT) or leave an order (INSERT), unlike the administrator, who can take any action. So, it's better to create several different users. Grant all privileges to the administrator, and restricted to the ordinary user.
2) Disabling error messages
First of all, you must avoid the built-in MySQL function mysql_error (). A smart attacker can guess some of the database settings from the error message and sometimes see the connection settings. It is best to use mysql_error () only during development, but to remove it when you run the site on the server. As a result, the user does not recognize from the error message any important information such as database name, table name, user name and others. Thus, we make it difficult for a hacker to find out the structure of a SQL query using different injections.
3) Using stored procedures
Using stored procedures can also help reduce the risk of an attack.Using stored procedures can also help reduce the risk of an attack.
4) Using locking features
You can use the mysql_real_escape_string () function to process external data. This is a very powerful built-in PHP feature that can prevent SQL injection in most cases. You can try to implement SQL code after using mysql_real_escape_string () and test for vulnerability. This feature rejects many of the clever attack methods used by attackers.
5) Use regular expressions
Regular expressions are used to bring the input to one template. For example, here we check the client's email for validity and reject the SQL injection option. You can also use the built-in PHP functions is_array(), is_bool (), is_double (), is_float

(), is_int (), is_integer () and others to verify user data.

6) Input filtering
In some cases, the fields have a numeric type and are often not quoted. Therefore, "quotation" and the replacement of special characters in the escape sequence does not work. In this case, type checking helps; if the variable is not a number, the request should not be run at all.
7) Shielding special characters
Character Shielding - Replacing the control characters in the text with the appropriate text substitutions. To implement the code (closing the line beginning with the quotation mark, the other quotation marks before the end of the current closing quotation mark with quotation marks), it is impossible for some DBMS, including MySQL, to quote all the string parameters. In the parameter itself, the quotes are replaced with
", the apostrophe with
', the backslash with
(this is called" screen special characters ").

4. Statement of the problem

This paper deals with the problem of the integrity of the database structure, that is, altering the database schema due to unauthorized changes, one example being SQL injection attacks. The prevalence of SQLIA and the potential damage they can cause, including theft of personal data and denial of service to a web server, require productive solutions and quick response to the presence of these attacks on the system. Therefore, the main task is to analyze the existing methods for detecting changes in the database schema and to analyze methods for responding to the original database schema. And actually, on the basis of this analysis, the proposal of a new methodology that will both detect database schema violations and offer options for their solution. The object of the study is a database whose structure changes unauthorized. The subject of the study is to identify and correct the integrity of the database schema. The purpose of the work is to develop a methodology for detecting and correcting database schema integrity violations based on initialization scripts. The scientific novelty is that the developed methodology is a complete solution, which allows not only to detect violations of the database schema, but also to offer solutions that do not exist in the scientific community. This technique uses database schema initialization scripts to compare with the current schema, which is also not described or implemented. The practical value of the results of the work can be used to implement a corporate or open source product, which can be useful for both leading IT companies and small businesses using databases in business. Our goal is a technique that will quickly detect that the database schema has been modified in accordance with the initial initialization and will offer a possible solution. To do this, we will use database initialization scripts to represent the original DB schema and compare it with the current DB schema. In fact, a mismatch will indicate changes, most often unauthorized ones, and a method that will, based on the rules, prompt the database administrator to enter commands to bring the system back to the original scheme. In this section we will describe the general scheme of the methodology, which will consist of the following steps: the stage of presentation and analysis of the initial scripts, the stage of analyzing the current state of the schema database, the step of comparing the schemas, the response and correction phase, which will result in bringing the scheme to the initial scheme.

5. General scheme of the technique

This section will show a general outline of a new methodology that aims at this work, a methodology for identifying and correcting integrity violations of a database schema, which consists of the following steps:

1) Stage of presentation and analysis of initial scripts.
This step is to analyze what the schema was embedded in the initiation script of the database, to bring the initial script of the database schema to the general view, which will present the current schema of the database, and to present its structure for later use.
2) The step of analyzing and representing the current state of the database schema.
The purpose at this stage is to analyze the current state of the database schema and present the current database schema in the same form as the original database script, for later comparison.
3) Phase comparison scheme.
The comparison step consists in the fact that one of the variants of the comparison algorithm of the initial and current schema of the database for compliance with the specified criteria should be offered. This algorithm outputs matching results. If the schemes do not match, which in most cases indicates unauthorized changes, we proceed to the next stage, namely the response and correction phase.
4) Stage of response and correction.
The last is a step which, if compared, shows that the schematics did not match, will suggest a remedy. If, however, the schemes match, then this stage will not be used. This stage is complementary to all work and final throughout the scheme.

5.1. Stage of presentation and analysis of initial scripts

First, an integral part is the stage, which consists of analyzing the original scripts and presenting them to a specific kind that can be easily compared. The initial DB script is a set of SQL commands at which to run a relational database schema, so there are several options for presenting the original script as an initial schema:

1) Execution of scripts
2) Analysis of text commands

The disadvantage of the first option is that the script requires some custom environment, such as a test database. In the case of Option 2, no additional

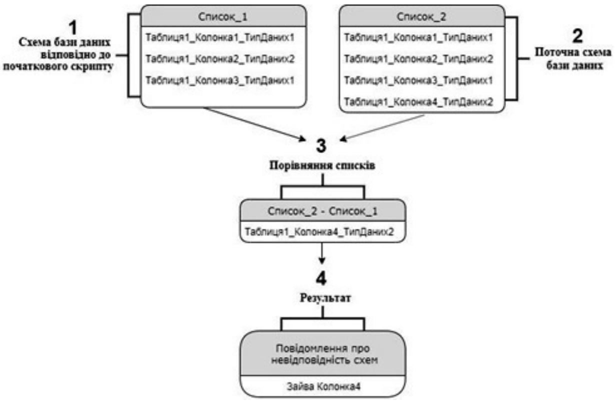

1
Схема бази даних відповідно до початкового скрипту
Список_1
Таблиця1_Колонка1_ТипДаних1
Таблиця1_Колонка2_ТипДаних2
Таблиця1_Колонка3_ТипДаних1
Список_2
Таблиця1_Колонка1_ТипДаних1
Таблиця1_Колонка2_ТипДаних2
Таблиця1_Колонка3_ТипДаних1
Таблиця1_Колонка4_ТипДаних2
2
Поточна схема бази даних
3
Порівняння списків
Список_2 - Список_1
Таблиця1_Колонка4_ТипДаних2
4
Результат
Повідомлення про невідповідність схем
Зайва Колонка4

Fig. 1. Scheme of the algorithm of comparison of schemes

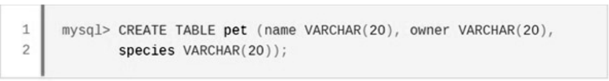
environments are required, and this is an advantage. SQL (structured query language) - a declarative programming language for user interaction with databases, used for querying, updating and managing relational databases, creating a schema database and its modification, a database access control system consisting of [6]:

1) DDL (Data Definition Language) — working with the base structure,
2) DML (Data Manipulation Language) — working with strings,
3) DCL (Data Control Language) — work with rights,
4) TCL (Transaction Control Language) — working with transactions.

Since we are interested in creating a base structure, we will consider DDL. DDL is a family of computer languages used in computer programs or database users to describe data structures. DDLs have their own functionality, organized by the initial word in the statement (query), which is almost always a verb. In the case of SQL:

1) Create
2) Alter
3) Drop

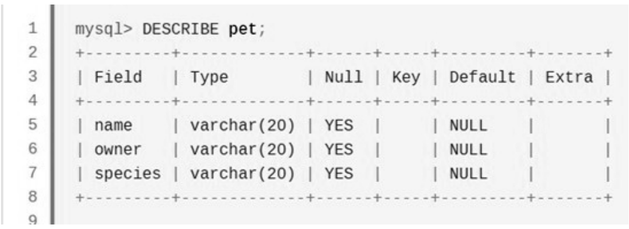
DDL as well as SQL are languages with strict static typing and which are described in the following dialects: SQL-86, SQL-89, SQL-92, SQL: 1999, SQL: 2003, SQL: 2006, SQL: 2008, SQL: 2011, SQL: 2016, so they are often independent of the specific DBMS, which is another benefit. Let's consider the following example of a SQL script [7]:



```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
       species VARCHAR(20));
```

Fig. 2. An example of SQL command

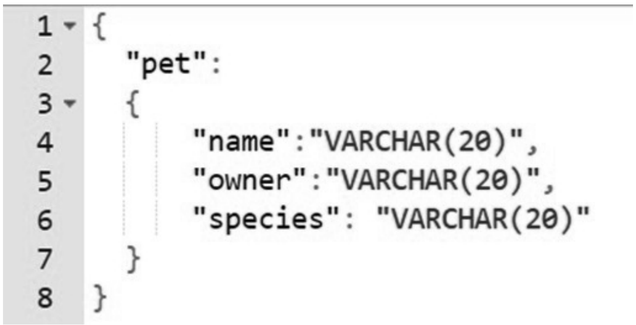
This script creates a table named "pet" that will contain the following columns: name that corresponds to the name of the animal that has a VARCHAR data type (20), that is, a text type, with a maximum length of 20 owner that corresponds to the name of the host whose data type is VARCHAR (20), that is, a text type, with a maximum length of 20 species corresponding to the host name in which the data type VARCHAR (20), that is, the text type, with a maximum length of 20 After executing this command, you can check the following command: DESCRIBE <table name>, where <table name> is the name of the table, in our case "pet".



```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

Fig. 3. An example of program execution

As we can see in the figure, a table with three columns and the following data types was created. Therefore, it was decided to create a JSON file that would fit the database schema according to the original script and the current database. The following JSON file structure is proposed:



```
{
  "pet":
  {
      "name":"VARCHAR(20)",
      "owner":"VARCHAR(20)",
      "species": "VARCHAR(20)"
  }
}
```

Fig. 4. An example of a JSON file

This is possible due to strict static typing. For example, the words CREATE TABLE are always followed by the name of the table being created. Following the name of the table, the column name and its type of comma-separated data are given in parentheses, so it is possible to create a software implementation that will convert an SQL script to a JSON file type that will be implemented in PYTHON3 using plain text, and embedded the JSON library.

5.2. Description of the step of analyzing and presenting the current state of the database schema

This step is the next step after submitting and analyzing the initial scripts. At this point, it is necessary to analyze the current state of the database schema and present the current database schema in the same form as the original script will be presented for further comparison. Of all the existing and analyzed methods for obtaining the current schema, it can be concluded that there are many ways to obtain the current state of the database schema. Some options have a graphical interface, some have console applications. Based on our needs, it was decided that we would be comfortable with getting the current database schema either from the command line or through the SQL command DESCRIBE, as it would be easy to do through the Python3 programming language and easy to connect with other modules of our methodology. Of all the options listed, the following remain:

1) Mysqldump;
2) pg_dump;
3) SQL command describe;
4) sp_columns

Analyzing the properties of each, it was highlighted that the SQL commands DESCRIBE and sp_columns require additional actions to analyze the current state from the output field, and the Mysqldump and pg_dump commands formulate an SQL script similar to the initialization script, which leads to the script the initialization and the current SQL script, the same steps are required, namely the actions suggested in section 5.1. Because Mysqldump works with MySQL, and pg_dump with PostgreSQL, so you need to keep track of which database you want to work with and invoke one application or another, but work is scientific and its purpose is to come up with an idea so let's just focus on using pg_dump and the PostgreSQL database. The result of all actions will be: getting a script of the current state -> using the method of analyzing and presenting scripts (p.5.1) -> we get JSON with the current structure, which can be easily compared and conclude about the integrity of the database structure, so you can proceed to the next stage , namely: comparing schemas.

5.3. Phase comparison scheme

The stage of comparison of schemes is important, because it can be used to draw conclusions about the existence of unauthorized changes and further actions in the methodology. The essence of this step is that it is necessary to offer one of the variants of the algorithm of comparison of the initial and the current scheme of the database for compliance with the specified criteria. After the comparison, the algorithm outputs the results about the matches. If the schemes do not match, which in most cases indicates unauthorized changes, we move on to the next step, namely the response and correction method. The comparison algorithm in our case will be as follows:

1) Read a JSON file that matches the database schema according to the original script in the list.
2) Read a JSON file that matches the current database schema to the list.
3) Compare lists by subtracting the current schema list from the original schema list.
4) Receive the result and output it to the console as a message about the compliance or inconsistency of the schema data:
   a) If the schematics are the same, a message is displayed in the console that no discrepancies were detected.
   b) If the schematics do not match, the console displays messages that found inconsistencies in the form:
      – Extra column <column name>;
      – Extra table <table name>;
      – Column types <column name> does not match.

Let's present this algorithm in the form of the scheme shown below:

Therefore, if no schema changes are detected, the database is in the same state as the initialization script. It follows that no action is needed to correct the DB schema. If the schematics do not match and the algorithm produces discrepancies in them, then we go to the response and correction phase.

5.4. Stage of response and correction

This stage is the last step in the development of the methodology and is based on the results of the previous stage. It only applies if, when comparing the circuits, it turns out that they did not match. Its main idea is to respond to the inconsistency of database schemas and to offer a correction for this violation, which in most cases indicates unauthorized changes. Section 1 reviews and analyzes existing approaches to database recovery, and since each contains some drawbacks, this paper will propose a new method that will offer commands to the database administrator to restore the database schema to its original state. The principle of operation is as follows: after comparing the schemas, we get which tables / columns / data types do not match the original schema, and we propose commands based on a set of rules. An example is the following discrepancy:

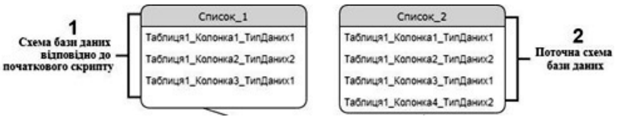

1
Схема бази даних відповідно до початкового скрипту
Список_1
Таблиця1_Колонка1_ТипДаних1
Таблиця1_Колонка2_ТипДаних2
Таблиця1_Колонка3_ТипДаних1
Список_2
Таблиця1_Колонка1_ТипДаних1
Таблиця1_Колонка2_ТипДаних2
Таблиця1_Колонка3_ТипДаних1
Таблиця1_Колонка4_ТипДаних2
2
Поточна схема бази даних

Fig. 5. Example of database schema mismatch

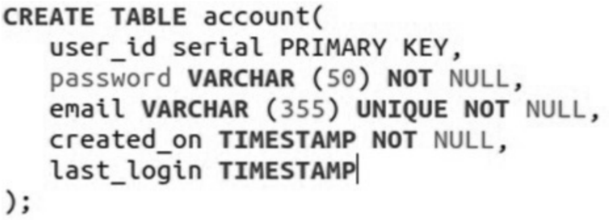
So, in the difference obtained, you can see that Table1 —- Column4 —- Data type2 exists, which does not exist in the original schema, so the method will offer the following command: ALTER TABLE "Table1" DROP "Column4". Therefore, after executing this command, the database schema will change to the original schema according to the initialization script.

6. Results of work

The result of the work is a prototype that will be able to correct some changes to the database schema. The following correction vectors are considered at this stage:

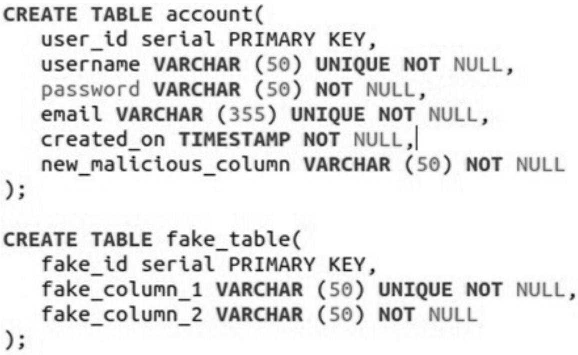
1) Find extra tables
2) Find the columns in which the data type was changed
3) Find deleted columns
4) Finding extra columns

An example would be the following situation:



```
CREATE TABLE account(
   user_id serial PRIMARY KEY,
   password VARCHAR (50) NOT NULL,
   email VARCHAR (355) UNIQUE NOT NULL,
   created_on TIMESTAMP NOT NULL,
   last_login TIMESTAMP
);
```

Fig. 6. An example of an initiation script for a combination of cases

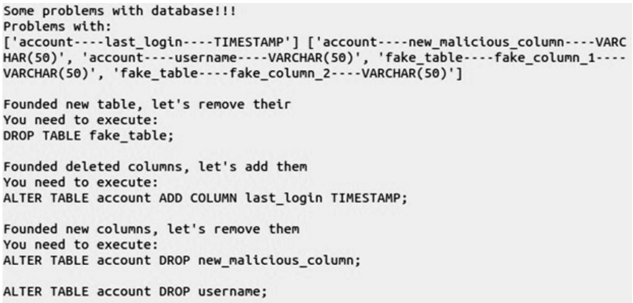


```
CREATE TABLE account(
   user_id serial PRIMARY KEY,
   username VARCHAR (50) UNIQUE NOT NULL,
   password VARCHAR (50) NOT NULL,
   email VARCHAR (355) UNIQUE NOT NULL,
   created_on TIMESTAMP NOT NULL,
   new_malicious_column VARCHAR (50) NOT NULL
);

CREATE TABLE fake_table(
   fake_id serial PRIMARY KEY,
   fake_column_1 VARCHAR (50) UNIQUE NOT NULL,
   fake_column_2 VARCHAR (50) NOT NULL
);
```

Fig. 7. An example of a current DB scheme for a combination of cases

As we can see from 6 and 7, in the current schema there is a new fake table_table and there are also two new columns, namely username VARCHAR(50) and new_malicious_column VARCHAR (50) and deleted column last_login TIMESTAMP. The launch of our program produced the following results:



```
Some problems with database!!!
Problems with:
['account----last_login----TIMESTAMP'] ['account----new_malicious_column----VARC
HAR(50)', 'account----username----VARCHAR(50)', 'fake_table----fake_column_1----
VARCHAR(50)', 'fake_table----fake_column_2----VARCHAR(50)']

Founded new table, let's remove their
You need to execute:
DROP TABLE fake_table;

Founded deleted columns, let's add them
You need to execute:
ALTER TABLE account ADD COLUMN last_login TIMESTAMP;

Founded new columns, let's remove them
You need to execute:
ALTER TABLE account DROP new_malicious_column;

ALTER TABLE account DROP username;
```

Fig. 8. The result of the program is a combination of cases

As we can see from 8, he detected all the database schema changes and suggested the following SQL commands: DROP TABLE fake_table; ALTER TABLE account ADD COLUMN last_login TIMESTAMP; ALTER TABLE account DROP new_malicious_column; ALTER TABLE account DROP username;

All of these commands will bring the current database schema to the original one.

7. Conclusions

This paper presents a technique for detecting and correcting database schema integrity violations based on initialization scripts. The problem of the integrity of the database structure, that is, changing the database schema is due to unauthorized changes, one example being SQL injection attacks. During the work, database types, existing threats to OWASP databases, code injection attacks were reviewed and analyzed, and SQL injection attacks, their implications for relational databases, and existing methods of database security threats were addressed. The study also analyzed existing methods for detecting and correcting database breaches and, based on their shortcomings, proposed a new methodology. This technique consists of the following steps: the step of presenting and analyzing the initial scripts, the step of analyzing the current state of the schema database, the step of comparing the schemas, the response and correction phase, which will result in bringing the schema to the initial schema. The software implementation of the methodology was implemented in Python3 programming language. The developed methodology is a complete solution that allows not only to detect database schema violations, but also to offer solutions that do not exist in the scientific community. This technique uses database schema initialization scripts to compare with the current schema, which is also not described or implemented. The proposed methodology can be used to implement a corporate or open source product, which can be useful to both leading IT companies and small businesses using databases in business.

References

[1] “Rain forest puppy. nt web technology vulnerabilities,” *Phrack Magazine*, vol. 8, no. 54, 1998.

[2] “Naive algorithm for pattern searching.” http://www.geeksforgeeks.org/searching-for-patterns-set-1-naive- pattern-searching/. Accessed: 2019-10-02.

[3] “Why is sql injection the most dangerous kind of vulnerability?.” https://acribia.ru/articles/why_sql_injection_is_the_most_dangerous_type_of_vulnerability. Accessed: 2019-10-02.

[4] S. Kost, “An introduction to sql injection attacks for oracle developers,” *Integrity Corporation*, 2007.

[5] “Sql injection protection.” http://www.securityscripts.ru/articles/sql-injection.html. Accessed: 2019-10-02.

[6] “Sql.” https://uk.wikipedia.org/wiki/SQL. Accessed: 2019-10-02.

[7] “Creating a table.” https://dev.mysql.com/doc/refman/8.0/en/creating-tables.html. Accessed: 2019-10-02.