Implementation of BroNIE with trade off time

Jyanab Khatun, Rajeshwar Gande

Department of Computer Science and Engineering

University of south Florida

Tampa, Florida 33621

jaynabk@mail.usf.edu rajeshwar@mail.usf.edu

Abstract: This paper defines and analyzes SQL injection prevention method called BroNIE. The definition of BroNIE's Property and analyze its implementation with trade off time is described in this paper. The definition is derived from NIE property which states that an application's taint inputs produce Nonecode insertion or expansion in output programs. The NIE property defined and described by Ligatti et al. open the space to analyze BroNIE. BroNIE is detection algorithm which check NIE property with CIAO. Malicious injection which do not occur due to involvement of injection of code, named such as nonecode injection attacks.

Keyword: BroNIE, algorithm, nonecode,SQL, Property.

I. INTRODUCTION

SQL injection attacks are most causes software error and vulnerability now a days. An SQL injection attack target the vulnerable web application to gain information or access about database service. When user input, its translate into SQL query and get the data from database. The attacker can inject nonecode or provide modified user's input. Thus attacker get the intendent information from database. Most of the SQL injection happen in user input. The malicious input produce a query and get the sensible information like user account credential or internal business data. The taint input may include nonecode. The nonecode concept was described by Ligatti et al[1] and used it to define NIE property. A typical example of nonecode injection in banking application, where output follows the program.

TRANSFER amount FROM user A TO user B WHERE password = 'qwerty123';

If the application does not have security to validate the input, the program can produce malevolent input as follows.

TRANSFER amount FROM user A TO user B WHERE password = 'qwerty123 OR 9=9--';

The output program dodge the password and produces the transfer the amount because 9=9 is always tautology. It makes WHERE clause to be true. The second reason is in SQL " - -" is comment line. After comment line everything is void which removes the final apostrophe here and transfer into valid program.

Many scholar researched on SQL code injection, but little on nonecode injection attack. Nonecode injection attacks do not employ malicious code rather they cause other parts of output program to become vulnerable.

II. Background: SQL Injection

Since SQL injection defined, the researcher defined many categories of SQL injection and developing detection and prevention technique. Here we try to discuss briefly.

Tautology:

Most known SQL injection technique is Tautology. To get access of unauthorized data is to insert tautology into query. The common object of this technique is to injected code in one or more conditional statements so that they always evaluate true. In SQL, if the WHERE clause of a SELECT or UPDATE statement is disunited with a tautology, then every row in the database table is included in the result set[2]. Generally this method result when the code either display all of the returned record or performs some action if at least one record is record. Below example can demonstrate the tautology.

SELECT accounts FROM TABLE tab WHERE user =’evil or 1=1 - -‘ AND password =’qwerty123’;

The code 1=1 injected in the condtional which turn WEHRE to be true always. The query evaluates to true for each row in database table and as result returns all of them. Reference [3].

Illegal/Logically Incorrect Queries

The main purpose of illegal queries attack is to identifying injectable parameter and performs database finger print so that attacker extracting data. The vulnerability leveraged by this attack is that the default error page returned by application servers is often overly descriptive. The error information help programmer to acquire vulnerable parameters and debug their application, which further helps to attackers gain information about the schema of the back-end database. This example attack’s goal is to cause a type conversion error that can reveal relevant data. To do this, the attacker injects the following text into input field pin: “convert(int,(select top 1 name from sysobjects where xtype=’u’))”. The resulting query is:

SELECT accounts FROM users WHERE login=’’ AND pass=’’ AND pin= convert (int,(select top 1 name from sysobjects where xtype=’u’))

In the attack string, the injected select query attempts to extract the first user table (xtype=’u’) from the database’s metadata table (assume the application is using Microsoft SQL Server, for which the metadata table is called sysobjects). The query then tries to convert this table name into an integer. Because this is not a legal type conversion, the database throws an error[3].

Union Query:

In Union query attacks, SQL allows two queries to be joined and returned as one result set. Attackers do this by injecting a statement of the form: UNION SELECT . Because the attackers completely control the second/injected query, they can use that query to retrieve information from a specified table. The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query. Example: Referring to the running example, an attacker could inject the text "' UNION SELECT cardNo from CreditCards where acctNo=10032 - -" into the login field, which produces the following query:

SELECT accounts FROM users WHERE login='' UNION SELECT cardNo from CreditCards where acctNo=10032 -- AND pass='' AND pin='

Assuming that there is no login equal to "", the original first query returns the null set, whereas the second query returns data from the "CreditCards" table. In this case, the database would return column "cardNo" for account "10032." The database takes the results of these two queries, unions them, and returns them to the application. In many applications, the effect of this operation is that the value for "cardNo" is displayed along with the account information[3].

Piggy-Backed Queries:

This attack perform denial of service attack and executing command remotely. In this technique, without modifying original data, the attacker tries to inject additional information which piggy back on the original query. The benign query execute normally where injected query executed as additional of first query. In the below example if the attacker inputs "'; drop table users - -" into the pass field, the application generates the query:

SELECT accounts FROM users WHERE login='doe' AND pass=''; drop table users -- ' AND pin=123

After execution of first query the database server recognize the query delimiter and execute the second query. The result of second query drop table users which destroy valuable information.

Stored Procedures:

This type of attack execute the stored procedure strored in database. The attacker determine the back end database and crafted the query to execute the stored procedure supported by specific database which interact with operating system. After determining the back end database the attacker may use piggyback technique. In the example attacker inject " ' ; SHUTDOWN;--" into the query, the result query is following

SELECT accounts FROM users WHERE login='doe' AND pass=' '; SHUTDOWN; -- AND pin=

The first query executed normally and second query executed as additional and result to shut down the system.

Inference:

In this attack, the query is modified to recast it in the form of an action that is executed based on the answer to a true/- false question about data values in the database. This type of attack happens with secure website. Timing attack is one kind of inference. By observing timing delay from database, user get information about database in Timing attack technique. To perform this attack the villain user structure their injected query in the form if/else format, whose branch directed to unknown about the content of database. Attacker user SQL statement using WAITFOR with one if the branch which takes a known amount time to execute. The response time of database can increase or decrease, the attacker can infer which branch has taken in his injection and therefore the result to the injection query.

III. Summary of contribution

So far we have discussed different technique of SQL injection attack. So many prevention techniques has been developed but those are not for nonecode. This paper define the Broken NIE and implement mechanism for practical application. Here we consider nonecode and our application prevent the SQL injection attack with code and nonecode also. To define BroNIE, first need to understand NIE property[1]. NIE is based on whether a untrusted input afftects the output program output program by inserting or expanding the nonecode token. On other side, untrusted input affects output program in any way without inserting of expanding the nonecode token, then we call it Borken NIE property. NIE property is similar with parameterized queries, but in later case, the hole for untrusted input tokens are manually specified in the application program. In our application instead manually define untrusted token we define automatically being a nonecode. This application run on runtime injection,hence take little longer time than parameterized query. Our implementation detect the injection attack with taint tracking mechanism. We use the taint mechanism to mark the injected token and then define them as nonecode token from untrusted source based on property.

Definition:

To define BroNIE in positive way we consider CIAO and NIE property[4]. CIAO property based on two subdefintion. First, which symnols in output application program constitute the code. Second, when the symbol injected into output program. The later developed using well known taint tracking mechanism. The first part can be code or nonecode. For general purpose, we take few assumption for language which is already define by Ligatti et al[1]. and Ray[4]. Here we assume taint tracking mechanism provide correct position of injection code and noencode. After we apply our implementation to detect injection attack. The definition of BroNIE can b as follows:

1) The output program affect in another way except injecting or expanding noncode or code.

Noneocde is closed value. Evey programming language contain closed value and open value. Closed values are valid in normal form and dynamically passive, which means by themselves cause no dynamic computation to occur. Typically noncode includes standalone string, integer literal, pointer, object, list etc. Where open values are free variable in any programming language and they are dynamically active.

2) For all language L with alphabet € and program p= $\sigma_{1,}$- - -, σ_n where |p| =n and p_i= $\sigma_{i.}$. For all language $\underline{L}$ with alphabet $\underline{€}$ and template program $\underline{T}$= $\underline{\sigma_{1,}}$- - -, $\underline{\sigma_n}$ where |T| =n and T_i=$\underline{\sigma_{i.}}$. The

relation between program token and template token is $\underline{\sigma_i}' = \sigma_i$. Now remove all noncode token from program token and compare it with corresponding template token. If none of the following condition is satisfy, then we consider its exhibit BroNIE.

1) P[i]=T[i]
2) P[i] contains noncode and T[i]<=P[i]
3) P[i] is noncode.

Instead of BroNIE we can define this TcloNIE as noncode are Closed value and it depends on template program. The TcloNIE is decuce T is for template and co for Closed value.

Consider an example to understand the TcloNIE. The example is taken from reference [1].

S E L E C T b a l a n c e F R O M a c c t s W H E R E p w = ‘ ’ O R 1 = 1 - - ’

Draw the table

IV. Analysis of TcloNIE with Trada off Time

In this section we developed the application using Java. The snippet of code is provided to understand the mechanism. The algorithm runs on O(n). We have chosen Java because the type checking property which is secure than other language. Web Servers running Java frequently interface with SQL databases, making providing mechanisms designed to prevent SQL injection with accurate taint information a valuable research target[5]. The input of program token is injected code and is result of taint tracking application. Then we produce template token by removing all underline character and replace it by # instead of ϵ. The program take input from user and transform into it SQL query to execute. Then it goes to Taint tracking application and mark it where the noncode injected. The output pass through TcloNIE to examine it exhibit NIE property or not.

To analyse the time of TcloNIE we focused on while loop count. First we parse the program make into token. Then find each noncode token and replace it #. Then it goes to while loop to check three TCloNIE properties. The maximum iteration in while is length of character in query. So program run in O(n).

V. Discussion

The paper has presented TCloNIE, a general class of injection prevention technique in which output program needs to satisfy non-NIE property means injected symbols affect the output programs beyond inserting or expanding noncode tokens. TCloNIE check both code and noncode attacks on output program. TCloNIE works with taint tracking algorithm perform on output program of Taint Application.

Our purpose is to meet the requirement of SQL injection detection condition. First we discussed about different type of attack technique which has different prevention technique. But all previous technique work on code and its depend on type of attack. Then we try to discuss related work with BroNIE property and deduce TCloNIE definition. In section VI, example is given and discussed how TCloNIE implementation. The execution time is depend on length of output program and running time express in O(n). Our application work on all types of injection attack and perform better than parameterized query technique. In Future work, we try to execute TCloNIE without Taint tracking, for this we need to modify or do some prior work so that run time of TCloNIE will be same. The application remove disadvantage of parameterized query and automatically remove noncode token.The application would

need to focus on web application which use nonSQL query. Overall TCloNIE is lightweight SQL detecktion application which can use in mobile application.

Acknowledgement

This work is motivated by Jay Ligatti, the assistant professor of department of computer science and engineering at University of South Florida. Under his supervision we worked on this algorithm and prepare a lightweight application for general use.

Reference

1) Defining Injection Attacks
2) Using Parse Tree Validation to Prevent SQL Injection Attacks
3) A Classification of SQL Injection Attacks and Countermeasures
4) Defining and Preventing Code-injection Attacks
5) Analysis and Prevention of Code-Injection Attacks on Android OS
6) A Classification of SQL Injection Attacksand Countermeasures
7) Precise Detection of Injection Attacks on ConcreteSystems
8) Analysis and Enforcement of Web Application Security Policies
9) Efficient Character-level Taint Tracking for Java
10) TaintDroid: An Information-Flow Tracking System for Realtime PrivacyMonitoring on Smartphones
11) Research of SQL Injection Attack and PreventionTechnology