An Inferential Metamorphic Testing Approach to Reduce False Positives in SQLIV Penetration Test

Lei Liu1, Guoxin Su2, Jing Xu1, Biao Zhang1, Jiehui Kang1, Sihan Xu1, Peng Li1 and Guannan Si3
1College of Computer and Control Engineering, Nankai University, Tianjin, China
2School of Computing and Information Technology, University of Wollongong, Australia
3School of Information Science and Electrical Engineering, Shandong Jiaotong University, Jinan, China

Abstract—SQL Injection Vulnerability (SQLIV) has been the top-ranked threat to the Web security consistently for many years. Penetration tests, which are a most widely adopted technique to detect SQLIV, are usually affected by testing inaccuracy. This problem is even worse in inference-based, blind penetration tests for online Web sites, where Web page variations (such as those caused by built-in dynamic modules or user interactions) may lead to a large number of False Positives (FP). We present a novel approach called Inferential Metamorphic Testing (IMT) to reduce FP in SQLIV penetration tests. First, we define the notion of Inferential Metamorphic Relations (IMR), which is inherited from Mutational Metamorphic Testing (MMT). Second, we present a set of logic operators and mutation operators for generating IMR and deducing the background testing context. Finally, we present an iterative IMT process, which is based on the heuristic IMR generation and the background testing context deduction. Our empirical study demonstrates the effectiveness of our approach by a comparison to three famous SQLIV penetration test tools.

Keywords—web vulnerability; penetration test; metamorphic testing; SQL injection; mutation testing; inference-based testing

I. INTRODUCTION

Along with the development of Internet, the Web security issues have become increasingly severe. Among all the Web application vulnerabilities, SQL Injection Vulnerability (SQLIV) has been one of the top-most vulnerabilities for a long time [1], because SQLIV can be exploited by attackers through specially designed input on Web application to alter the logic of SQL queries into the background database [2]. Penetration tests, which are one of the most popular SQLIV testing technique, are a kind of dynamic tests that aim to expose SQLIV through the mock attacks of testers [3]. However, one highly disturbing problem of this technique is the testing inaccuracy, especially reflected in high False Positives (FP), which may reduce the credibility of testing tools and cost longer time for manual checking.

Blind SQLIV penetration tests are an inference-based technique that relies on observing the variation of a Web page content caused by the SQL queries with “True” or “False” logic from a tester [4]. This technique is proven to be effective in finding SQLIV even when database errors are hidden. However, blind SQLIV tests tend to result in a large number of FP, because of the frequent changes of Web page contents caused by user interactions and dynamic modules [5]. Furthermore, because of the increasing complexity of Web structures (such as a search engine), back-end programs of a Web page may exhibit behaviors that are similar to SQLIV. These dynamic variations of Web page contents actually disable the traditional test oracles to some extent, making the FP problem as an oracle problem essentially.

Many works have been done on improving the detecting capabilities of SQLIV penetration tests. Some focus on combining static and dynamic testing approaches that usually use static code analysis or server proxy to increase the testing capability of SQLIV penetration tests [6][7]. But, the need to access the source code or setting proxy by altering server programs, which are often infeasible for legacy systems or third-party outsource, limits the applicability of those approaches. Other works including test case mutation based methods [8][9] focus on increasing the testing coverage [10] and improving testing procedures [11]. However, few existing work deals with the problem of FP, especially for blind SQLIV penetration tests.

To address the FP problem in (especially blind) SQLIV penetration tests, in this paper we propose an Inferential Metamorphic Testing (IMT) approach. Our IMT approach is based on the technique of Mutational Metamorphic Testing (MMT) [12], which is a variant of the Metamorphic Testing (MT) [13] by employing test case mutations to solve the oracle problem. In our IMT approach, we first define a relation called the Inferential Metamorphic Relations (IMR). Then, we introduce a set of Logic Operators (LO) and Mutation Operators (MO), to generate IMR and deduce Inferential Testing Context (ITC). Finally, we present an iterative IMT process based on the heuristic IMR generation and ITC deduction. Our empirical study shows the effectiveness of our approach by a comparison to three famous SQLIV penetration test tools.

The remainder of this paper is organized as follows: Section II describes backgrounds and related works. Section III presents our IMT approach. Section IV presents the empirical study. Finally, Section V concludes the paper.

II. BACKGROUNDS AND RELATED WORKS

SQLIV occurs when specially crafted SQL clause inputs reach the background database of a Web application, and alter the logic of the original SQL queries with the attacker’s intentions [3][4], which may cause Serious security issues. Lots of online and legacy Web applications are exposed to the threat of SQLIV. Therefore, researches on discovering SQLIV on these kinds of online systems are very important.

Abstract—SQL Injection Vulnerability (SQLIV) has been the top-ranked threat to the Web security consistently for many years. Penetration tests, which are a most widely adopted technique to detect SQLIV, are usually affected by testing inaccuracy. This problem is even worse in inference-based, blind penetration tests for online Web sites, where Web page variations (such as those caused by built-in dynamic modules or user interactions) may lead to a large number of False Positives (FP). We present a novel approach called Inferential Metamorphic Testing (IMT) to reduce FP in SQLIV penetration tests. First, we define the notion of Inferential Metamorphic Relations (IMR), which is inherited from Mutational Metamorphic Testing (MMT). Second, we present a set of logic operators and mutation operators for generating IMR and deducing the background testing context. Finally, we present an iterative IMT process, which is based on the heuristic IMR generation and the background testing context deduction. Our empirical study demonstrates the effectiveness of our approach by a comparison to three famous SQLIV penetration test tools.

Keywords—web vulnerability; penetration test; metamorphic testing; SQL injection; mutation testing; inference-based testing

I. INTRODUCTION

Along with the development of Internet, the Web security issues have become increasingly severe. Among all the Web application vulnerabilities, SQL Injection Vulnerability (SQLIV) has been one of the top-most vulnerabilities for a long time [1], because SQLIV can be exploited by attackers through specially designed input on Web application to alter the logic of SQL queries into the background database [2]. Penetration tests, which are one of the most popular SQLIV testing technique, are a kind of dynamic tests that aim to expose SQLIV through the mock attacks of testers [3]. However, one highly disturbing problem of this technique is the testing inaccuracy, especially reflected in high False Positives (FP), which may reduce the credibility of testing tools and cost longer time for manual checking.

Blind SQLIV penetration tests are an inference-based technique that relies on observing the variation of a Web page content caused by the SQL queries with “True” or “False” logic from a tester [4]. This technique is proven to be effective in finding SQLIV even when database errors are hidden. However, blind SQLIV tests tend to result in a large number of FP, because of the frequent changes of Web page contents caused by user interactions and dynamic modules [5]. Furthermore, because of the increasing complexity of Web structures (such as a search engine), back-end programs of a Web page may exhibit behaviors that are similar to SQLIV. These dynamic variations of Web page contents actually disable the traditional test oracles to some extent, making the FP problem as an oracle problem essentially.

Many works have been done on improving the detecting capabilities of SQLIV penetration tests. Some focus on combining static and dynamic testing approaches that usually use static code analysis or server proxy to increase the testing capability of SQLIV penetration tests [6][7]. But, the need to access the source code or setting proxy by altering server programs, which are often infeasible for legacy systems or third-party outsource, limits the applicability of those approaches. Other works including test case mutation based methods [8][9] focus on increasing the testing coverage [10] and improving testing procedures [11]. However, few existing work deals with the problem of FP, especially for blind SQLIV penetration tests.

To address the FP problem in (especially blind) SQLIV penetration tests, in this paper we propose an Inferential Metamorphic Testing (IMT) approach. Our IMT approach is based on the technique of Mutational Metamorphic Testing (MMT) [12], which is a variant of the Metamorphic Testing (MT) [13] by employing test case mutations to solve the oracle problem. In our IMT approach, we first define a relation called the Inferential Metamorphic Relations (IMR). Then, we introduce a set of Logic Operators (LO) and Mutation Operators (MO), to generate IMR and deduce Inferential Testing Context (ITC). Finally, we present an iterative IMT process based on the heuristic IMR generation and ITC deduction. Our empirical study shows the effectiveness of our approach by a comparison to three famous SQLIV penetration test tools.

The remainder of this paper is organized as follows: Section II describes backgrounds and related works. Section III presents our IMT approach. Section IV presents the empirical study. Finally, Section V concludes the paper.

II. BACKGROUNDS AND RELATED WORKS

SQLIV occurs when specially crafted SQL clause inputs reach the background database of a Web application, and alter the logic of the original SQL queries with the attacker’s intentions [3][4], which may cause Serious security issues. Lots of online and legacy Web applications are exposed to the threat of SQLIV. Therefore, researches on discovering SQLIV on these kinds of online systems are very important.
And penetration test [3][6] is one of the most important dynamic testing method for this situation.

Works on SQLIV penetration test include three main aspects: information gathering [11][14], test case generation [9][10] and response analysis [9][14], most of which are focused on increasing testing coverage [7][9][10] or improving testing procedures [8][10][11]. However, the insufficiency of testing accuracy (ie., high FP) in SQLIV penetration test has always been one of the most acute problems because of the lack of background testing context [4][7][12]. Some researching works have been conducted to reduce the impact of insufficient accuracy (ie., high FP), most of which are combined static and dynamic approaches [2][3][5]. However, the needs of accessing the source code or setting proxy by altering server programs is not always be feasible in situations of legacy systems or third-party outsource and thus limits the applicability of those methods. For black-box SQLIV penetration tests, the researches on solving the problem of high FP are still insufficient. Moreover, the FP issue has become increasingly severe after the appearance of inference-based SQLIV [2][5].

Fig. 1 shows an example of the classic decision process of blind SQLIV penetration tests, which can also explain the cause of FP in SQLIV penetration test. Traditional SQLIV can be exposed by database errors, but blind SQLIV can only be revealed by inferences, which database errors are likely hidden [2]. We assume the URL of the targeted HTTP request is “http://WebShop.Example.com/Customer?ID = 001&Session = 123”, and the original background SQL query of this request is like the query: “SELECT * FROM Customers WHERE ID = 001 AND Session = 123″. If the inputs of the above mentioned request are not properly filtered and “Session=123” is the injection point, a classic blind SQLIV penetration test pattern can be described as “(AND I=1 → Similar) & (AND I=2 → Different)”, which can also be described as the route “S0→S1→S2→S3”. Other decision paths show the situations of no vulnerabilities.

**Figure 1.** Blind SQL Injection Vulnerability (or FP) decision Process

This kind of SQLIV can be reduced by proper filtration and protection to some extent, but it still has the possibility of the occurrence of FP like the route “S0→S1→S2→S3”. The main reasons for this kind of FP can be concluded to three types: (1) random changes of HTTP responses caused by user interaction; (2) variation of Web page contents caused by dynamic modules, such as periodic advertisements; (3) the background functions have the similar behaviors to a SQLIV for some specific purposes, such as basic honey pot system. Such kind of FP are hardly avoided, and too much non-systematic verification may cause overmuch False Negatives (FN) and redundancy.

To address the above problem, this paper proposes a systematic approach to reducing the influence of FP during SQLIV penetration tests based on Mutational Metamorphic Testing (MMT). MMT is a mutation based test approach to tackle the oracle problem and generate test cases [12][13]. Its basic philosophy is to expose errors through the analyzing of related input-output pairs based on Metamorphic Relations (MR). The FP problem in SQLIV penetration tests is similar to the oracle problem, because those tests lack an absolutely accurate procedure to identify FP.

III. THE PROPOSED APPROACH

A. Inferential Metamorphic Testing

We propose an Inferential Metamorphic Testing (IMT) approach based on Mutational Metamorphic Testing (MMT) [12][13] to test FP in blind SQLIV penetration tests. Here, we define a test case that exposes a SQLIV alert as a seed. Our target is to verify whether a SQLIV alert is FP or not. We first define the Inferential Metamorphic Relation (IMR). We use P to denote the original program context under test, including the background SQL query and related program modules. Let I be the input test cases domain, and O be the corresponding output domain of I that includes the Response Logic of “True” and “False”. Let fi be a test case mutation function on I with applicability conditions C(x1, x2, ..., xn).

**Definition 1: an Inferential Metamorphic Relation (IMR) is represented as follows:**

\[ R(P(x_1), P(x_2), ..., P(x_n)) \]

\[ P(f_1(x_1, x_2, ..., x_n)), P(f_2(x_1, x_2, ..., x_n)), ..., P(f_m(x_1, x_2, ..., x_n)) \]

In other words, an IMR is a relation R on \( O^{m+n} \) such that the Response Logic of P is consistent (shows the Expected Logic) on inputs \( x_1, x_2, ..., x_n \) ∈ I and \( f_1, f_2, ..., f_m \) are applicable on \( x_1, x_2, ..., x_n \) imply that \( R(P(x_1), P(x_2), ..., P(x_n)) \), \( P(f_1(x_1, x_2, ..., x_n)), P(f_2(x_1, x_2, ..., x_n)), ..., P(f_m(x_1, x_2, ..., x_n)) \), where \( x_1, x_2, ..., x_n \) are the seed test cases, and \( f_1(x_1, x_2, ..., x_n), f_2(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n) \) are the mutation functions on \( x_1, x_2, ..., x_n \).

The Response Logic here is the logic of the similarity of the injection response content (such as \( S_1 \) in Fig. 1) with the original response content (such \( S_0 \) in Fig. 1), including “True” or “False” corresponding to “Similar” or “Different”. The Expected Logic is the Response Logic of P injected by seeds \( x_1, x_2, ..., x_n \). The Response Logic of mutation test cases \( f_1(x_1, x_2, ..., x_n), f_2(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n) \) should be consistent with the Expected Logic if they are valid, where consistent represents the consistency of the background logical regularity, such as “True=’False’”. If the IMR \( R(P(x_1), P(x_2), ..., P(x_n)) \), \( P(f_1(x_1, x_2, ..., x_n)), P(f_2(x_1, x_2, ..., x_n)), ..., P(f_m(x_1, x_2, ..., x_n)) \) is verifiable to be consistent, then \( f_1(x_1, x_2, ..., x_n), f_2(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n) \) can be classified as New Seed Test Cases (NSTC). An IMR can be described as follows:

\[ C(x_1, x_2, ..., x_n) \Rightarrow R(P(x_1), P(x_2), ..., P(x_n)), P(f_1(x_1, x_2, ..., x_n)), P(f_2(x_1, x_2, ..., x_n)), ..., P(f_m(x_1, x_2, ..., x_n)) \)
Consider the logical structure of background SQL query in $P$, such as the one in the example in Fig. 1. A mutation function $f(y)$ is a combination of test case mutation operators, which can be defined on the input domain $I$ of blind SIVLIV penetration test cases $f(\text{AND } I=1) = \text{AND } I=1 --$, where $f(y)$ adds a comment symbol “--” at the end of the test case “AND $I=1$”. The Response Logic result of $P(\text{AND } I=1)$ is the Expected Logic of the injection logic of the mutation of the test case “AND $I=1$”. The above mutation function can derive an IMR such as:

$$C(\text{AND } I=1): P(\text{AND } I=1)=\text{True} \land P(\text{AND } I=1--) = \text{True} \Rightarrow \text{IMR: } P(f(\text{AND } I=1)) = P(\text{AND } I=1)$$

The applicability condition $C(\text{AND } I=1)$ here is $P(\text{AND } I=1)=\text{True}$, which is the Response Logic of the seed test case “AND $I=1$”. After we inject the mutation test case “AND $I=1--$”, the Response Logic of $P(\text{AND } I=1--)$ is still “True”, which can derive the above IMR. Making inference about testing context is a important feature in our approach because the testing context is unknown. The whole process of IMT for penetration tests can be described as follows:

1) For a target injection point, we choose a set of test cases (which involve exposing SIVLIV alerts) as seed Test Cases (STC), i.e., $\text{STC} = \{x_1, x_2, ..., x_d\}$, and make $\text{NSTC}=\text{STC}$, where $\text{NSTC}$ is the test case set of new seeds.

2) We infer the Inferential Testing Context (ITC) from seed test case set $\text{NSTC}$ and conditions $C(\text{NSTC})$, and deduce a new mutation function set $F=(f_1, f_2, ..., f_m)$ and a new IMR $= \text{MR}(P(x_1), P(x_2), ..., P(x_d), P(f_1(x_1, x_2, ..., x_d)), P(f_2(x_1, x_2, ..., x_d)), ..., P(f_m(x_1, x_2, ..., x_d)))$ from $C(\text{NSTC})$ and the Inferential Testing Context (ITC).

3) We apply every mutation function in set $F=(f_1, f_2, ..., f_m)$ on each seed test case $x (x \in \text{NSTC})$ and get a Mutational Test Case (MTC) set, i.e., $\text{MTC} = \{f_1(x_1, x_2, ..., x_d), f_2(x_1, x_2, ..., x_d), ..., f_m(x_1, x_2, ..., x_d)\}$.

4) We inject all test cases in the mutation set $\text{MTC}$ to $P$, and record all the output Response Logic $P(f_1(x_1, x_2, ..., x_d)), P(f_2(x_1, x_2, ..., x_d)), ..., P(f_m(x_1, x_2, ..., x_d))$. If $P(f_m(x_1, x_2, ..., x_d))$ is consistent with the deduced IMR, then we consider this input-output pair as an evidence. If so, we add $f_m(x_1, x_2, ..., x_d)$ to set $\text{NSTC}$ and add the input-output pair $(f_m(x_1, x_2, ..., x_d), P(f_m(x_1, x_2, ..., x_d)))$ to the condition set $C(\text{NSTC})$.

5) We stop the IMT testing process whenever some prescribed Testing Threshold Expression (TTE) is satisfied. And another threshold Evidence Threshold Expression (ETE) is not satisfied. Then we consider this as an FP, and the testing procedure jumps to step 2) for next round of IMT.

From the above process of IMT, we can see that the standard of FP in this paper meets the Threshold Expression ($\text{ETE} \land \text{NTE}$). During an IMT, the Inferential Testing Context is based on inferences, so it is impossible to meet every IMRs. We use the threshold expression ETE to judge whether it is a FP. For instance, if we have “$\text{ETE}: (N_e < 4) \land (N_f < 2) \land (N_{Ne} < 2)$” ($N_e$, $N_f$, $N_{Ne}$ are variables defined by tester, which respectively represent the number of total evidences, “True” response logic evidences and “False” response logic evidences). And expression “$\text{TTE}: N_e > 100$” ($N_e$ represents the number of the generated test cases) means that, if the total number of evidences is less than four, or if the number of the evidences with the Response Logic “True” or “False” is less than two, then this alert is an FP.

### B. IMT Operators

#### 1) Mutation Operators

The proposed approach is based on test case mutation [8][12]. We introduce four kinds of Mutation Operators (MO) in this paper, including Keyword Mutation Operators (KMO), Condition Changing Operators (CCO), Syntax Changing Operators (SCO) and Mathematical Mutation Operators (MMO), as showed in Table I. Without further specification, the syntax of the operators is based on the Mysql database which is the most popular open source relational database.

<table>
<thead>
<tr>
<th>Cat.</th>
<th>Operators</th>
<th>Mutation Operators Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMO</td>
<td>$M_{KW}$ (input, $kw$)</td>
<td>Keyword mutation operation, $kw \in {\text{AND, OR, IF, SLEEP, HAVING, LIKE, IN, UNION, ORDER}}$</td>
</tr>
<tr>
<td>CCO</td>
<td>$M_{IT}$ (input)</td>
<td>Changing the logic of the input (test case) to “TRUE” logic predicate (Tautology)</td>
</tr>
<tr>
<td></td>
<td>$M_{CT}$ (input)</td>
<td>Changing the logic of the input (test case) to “FALSE” logic predicate (Contradiction)</td>
</tr>
<tr>
<td>SCO</td>
<td>$M_{MK}$ (input, $kw$, $L$)</td>
<td>Adding a clause with certain keyword to the input (test case), $kw \in {\text{AND, OR, LIKE, IN, ...}}$</td>
</tr>
<tr>
<td></td>
<td>$M_{MT}$ (input)</td>
<td>Removing a clause with certain keyword to the input (test case), $kw \in {\text{AND, OR, LIKE, IN, ...}}$</td>
</tr>
<tr>
<td>MMO</td>
<td>$M_{MR}$ (input)</td>
<td>Adding a comment operator to the end of the input (test case), such as “AND $I=2$” to “AND $I=2--$”</td>
</tr>
<tr>
<td></td>
<td>$M_{MC}$ (input)</td>
<td>Adding math calculating operations into the input (test case), such as “$I=2$” to “(11-10)=(1+1)”</td>
</tr>
</tbody>
</table>

KMO includes $M_{KW}$, and $M_{KW}$(input, $kw$) represents the operation that changes first keyword to “kw”. For example, $M_{KW} (“\text{AND } I=1”, “\text{OR})$ changes the first keyword “AND” of the test case “AND $I=1$” to “OR”. $M_{IT}$ or $M_{CT}$ in CCO can change the logic condition to a tautology or a contradiction. SCO can change the syntax structure of the original SQL query, in which $M_{MK}$ or $M_{MT}$ represents the operation of adding or removing a sub-query of specific Keyword, respectively. $M_{MK} (“\text{AND } I=1”, “\text{AND})$ and $M_{MT}$ in SCO can change the logic condition to a tautology or a contradiction. MMO includes two mathematical mutation operators, namely $M_{MR}$ and $M_{MC}$. The operator $M_{MR}$ conducts an operation of randomization to number or character values in a test case, such as “OR 867=867”. Operator $M_{MC}$ will add one comment symbol “--” at the end of a test case, such as “AND $I=1--$”. MMO includes two mathematical mutation operators, namely $M_{MR}$ and $M_{MC}$. The operator $M_{MR}$ will add one comment symbol “--” at the end of a test case, such as “AND $I=1--$”. MMO includes two mathematical mutation operators, namely $M_{MR}$ and $M_{MC}$. The operator $M_{MR}$ will add one comment symbol “--” at the end of a test case, such as “AND $I=1--$”.
original SQL query into two parts. We define them as Where (Contradiction) and Right Condition (Injection Logic) at the injection point (located in the intersection or union operation on the original SQL query, which is related to the keywords "AND", "UNION", etc.). Such as, the logic formula of test case ", which logic expression is "", which logic formula is ".

Operator \( \cup \) conducts union operation, which is related to the keywords "OR", "UNION", etc. Such as, keyword "OR" conducts a union calculation in an injection clause "\( \text{OR } 1=2 \)" which logic formula is "\( \cup \phi \)". Operator \( \text{Cmt} \) is a comment operator, which is corresponding to a comment symbol "--". Operator \( \vartriangleright \) shows the position of the injection point in a logical calculation formula, which will be replaced by an injection logic formula.

2) Logic Operators

To deduce the Inferential Testing Context (ITC) and get the Inferential Metamorphic Relation (IMR), we introduce a series of Logic Operators (LO) presented in Table II. With them, we can infer IMR or ITC with logic calculations. Operator \( U \) represents the query logic that can have a full set of data from a table in the background database, such as tautology "\( 1=1 \)". \( \phi \) is corresponding to the query logic of getting an empty data set from a table, such as contradiction "\( 1=2 \)". \( S_L \) and \( S_R \) respectively represents the left part and the right part of the original SQL query that divided by the injection point, which are subsets of a full set \( U \), and not empty, as showed in Fig. 2. \( \cap \) represents intersection logic operator, which conducts the intersection operation in a logic calculation. For instance, the logic formula of test case "\( \text{AND } 1=1 \)" is "\( \cap U \)". Operator \( \cup \) conducts union operation, which is related to the keywords "OR", "UNION", etc. Such as, keyword "OR" conducts a union calculation in an injection clause "\( \text{OR } 1=2 \)" which logic formula is "\( \cup \phi \)". Operator \( \text{Cmt} \) is a comment operator, which is corresponding to a comment symbol that can remove the logic of the right part after the injection point in the original SQL query, such as comment symbol "--". Operator \( \vartriangleright \) shows the position of the injection point in a logical calculation formula, which will be replaced by an injection logic formula.

C. IMR Generation and ITC Deduction

The generation of Inferential Metamorphic Relation (IMR) for SQLIV penetration test (especially blind SBLIV) is based on the testing context assumption and deduction, called Inferential Testing Context (ITC) Inferring. And the deduced IMR can be utilized to conduct IMT and deduce more detailed information of ITC.

Fig. 2 displays the logical structure of a typical SQLIV test case. A common SQLIV payload (test case) is usually consisting of four parts: Prefix, Keyword, Condition and Suffix. Where, Prefix and Suffix are used for syntax fixing. Keyword is the injection keyword to conduct additional intersection or union operation on the original SQL query. Condition is normally logical condition of tautology or contradiction formula. At the injection point (located in the Where clause in Fig. 2), a SQL injection payload divides the original SQL query into two parts. We define them as Left Condition (LC) and Right Condition (RC). Usually, because of the effect of Prefix and Suffix, the influence of a common SBLIV test case can be limited to the condition expression in the targeted sub clause, such as Where clause here. And the targeted sub clause is also separated into Left Condition (LC) and Right Condition (RC) by the injection test case.

Essentially, SQL injection is to change the original query logic with the payload's logic, and conducts the altered query logic calculation. The original SQL query logics can be described by Inferential Testing Context (ITC) 

\[
\text{LC} \cap \text{RC} (\text{IL} \in \{U, \phi, S_L\}, \text{RC} \in \{U, \phi, S_R\}, \phi \in \{\cap, \cup\}),
\]

where \( \text{LC} \) and \( \text{RC} \) are the Left and the Right part of a query logic Condition, which is divided by a SQL injection payload (test case). ITC is actually the formula abstraction form of background test context. The SQLIV test cases can also be described by Injection Payload Expression (IPE) 

\[
\phi \in \{\cap, \cup\}.
\]

If there exists a SQLIV, we can deduce its Response Logic from the logic calculation of the Inferential Test Context (LC \( \cup \) RC) and the Injection Payload Expression (\( \phi \)).

Table III shows the examples of basic IMR and ITC deduction criteria, in which we take a pair of test cases "\( \text{AND } 1=1 \)" and "\( \text{AND } 1=2 \)" as seeds, and the related ITC is "\( \text{LC} \cap \text{RC} \)". From the previous test, the conditions we obtained are \( C("\text{AND } 1=1", \text{AND } 1=2") = \{("\text{AND } 1=1", \text{True}), ("\text{AND } 1=2", \text{False})\} \). The Injection Payload Expression (IPE) of these test cases are "\( \cup U \)" and "\( \cap \phi \)". To infer the possible logic of ITC, we make an assumption that all the combinations of \( \text{LC}, \vartriangleright \) and \( \cap \) are possible for the ITC "\( \text{LC} \cap \text{RC} \)". We get \( |\text{IL}| \times |\vartriangleright| \times |\text{RC}| \) kinds of testing context scenarios in our approach. When situations that involving both \( S_L \) and \( S_R \), we still need to consider four kinds of set relationships between \( S_L \) and \( S_R \), including "\( S_L \subset S_R \)"; "\( S_L \supset S_R \)"; "\( S_L \cap S_R = \phi \)"; and "\( S_L \cap S_R \neq \phi, \phi \neq S_L \cup S_R \neq S_L \). And the number of basic IPEs we use in this paper is \( |\cap| \times |\text{IL}| \times |\vartriangleright| \). With the proposed IPE and IPE, we can obtain the list of Inferential Metamorphic Relations (IMR). Table III shows part of basic testing context scenarios we used, which logic expressions are "\( U \cup \phi \)" and "\( S_L \cup S_R (S_L \subset S_R) \). The columns of Test Case and Response Logic (\( \text{RP} = P(\text{TC}) \)) are actually the forms of IMR for IMT, such as \( P("\text{AND } 1=1") = P("\text{AND } 1=2") = \text{True} \).
To generate basic IMRs, we choose four mutation operators $M_{op}$, $M_{2}$, $M_{f}$ and $M_{m}$, which has little affection on the grammar structure of the original query. The mutation criteria are based on test case expression IPE: “$\diamond\exists \# ((L \in \{U, \emptyset\}, \diamond \in \{\land, \lor\}, \# \in \{\text{Cmt}, \text{null}\})$”. The condition logic should mutate between $\land$ and $\lor$. After a round of IMT iteration, the IMRs that trigger $\text{evidences}$ will be added to the conditions $C$, and related test cases will be marked as seeds. Meanwhile, more information about ITC can be derived from new IMRs. For instance, if $P(\text{OR } 1=1)=!P(\text{OR } 1=2)=\text{False}$ is valid, then “$S_l \land \neg \land S_R$ ($S_l \subset S_R$)” can be inferred as the most possible ITC, because other logical relation scenarios are not consistent with this IMR.

With new ITC and the extended seeds, we can generate more proper IMR and conduct systematic tests until the predefined Testing Threshold Expression (TTE) is reached.

### IV. EMPIRICAL STUDY

#### A. Experiment Implementation

To demonstrate the effectiveness of our approach, a prototype tool is developed to implement the proposed IMT approach in the environment of “Visual Studio 2010 + .Net 3.5 + C#”, which is utilized to test and discover FP produced by other SQLIV penetration test tools. The input of the tool is the basic URL and test information of discovered alerts of a SQLIV penetration test tool, and the output is whether these alerts are TP or FP.

Three state-of-the-practice benchmarking tools are chosen to assess our approach, including $WVS$ $Acunetix^1$, $IBM$ $Appscan^2$ and $Sqlmap^3$, which are three of the most used and famous penetration test tools. To protect their confidentiality and avoid brand comparison, we refer them as Tool A, Tool B and Tool C (without a particular order).

For target dataset, we choose 3567 URLs, including 1067 URLs obtained from a web application vulnerability evaluation project Wavsep $^4$ (The Web Application Vulnerability Scanner Evaluation Project), which is an open source project of OWASP [1] and contains 133 SQLIVs, and 2500 real Internet URLs with highly dynamic Web pages that may cause FP in high possibility.

#### B. Preparation

To prepare our experiment, benchmarking Tools A, B and C are used to testing on 3567 Web pages of the target dataset to collect the data of their alerts, TP and TF, as showed in Table IV. Here, we manually check to identify FP and TP from the alerts reported by the three test tools. In the table, URL$\#$, TP$\#$ and FP$\#$ represent the number of URLs, TP and FP respectively. FP$\_R$ is the result of FP Rate, and we have “FP$\_R = FP\#/TF\#$”. FP$\_F$ represents the FP occurring Frequency on each Web page, and we have “FP$\_F = FP\#/URL\#$. Rpsts represents the total HTTP requests sent during testing, which can be used to evaluate testing efficiency of assessed approaches. The total numbers of URL$\#$, TP$\#$ and FP$\#$ are the sum of three tools, and an alert includes the information of the corresponding URL, injection point and test cases.

#### C. Experiment and Results

In the experimental phase, we conduct our prototype tool on all alerts that three benchmarking tools generate, and we

---

have $T.Alerts# = T.TP# + T.FP# = 421$, in which $T.$ represents the “Total Number”. Table V displays the detailed FP testing and reducing results, in which $T.TP#$ and $T.FP#$ means the total number of TP and FP caused by three benchmarking tools. $TPV$ represents the TP that verified by our approach, which can evaluate the test coverage. $FPVm$ represents the number of FP that discovered and Removed by our approach. $FPVm_R$ represents the Rate of identified and Removed FP in all the FP previously reported, and $“FPVm_R = FPVm/T.FP#”$. Parameter $Rqsts$ here is the total number of HTTP requests sent by our approach during test, which can assess the efficiency of our approach.

We use three different Threshold Expressions (TE) as the judges of FP respectively, which including Evidence Threshold Expression (ETE) and Testing Threshold Expression (TTE), and the strict degree increase from TE1 to TE3 gradually. The TE we use is “TE n: $!ETE \land TTE$”, in which “$ETE = (N_T \geq 2n) \land (N_E \geq n) \land (N_I \geq n)$”, and “$TTE = (N_T \geq 100)$”. Here, $N_E, N_T, N_I$ and $N_I$ represents the number of the discovered evidences, “True”, “False” evidences and used test cases respectively. Here, the standard of FP is that, TTE is satisfied and ETE is not satisfied.

The results in Fig. 3 shows that the more strict the TE degree applied the more FP discovered. When “$n=3$”, the FP removing rate reaches to 98.39%. And there is no missing TP in our experiment, which means that our approach can maintain good coverage. The HTTP requests sent by our approach are much less than those in an entire testing, which means that our approach will not bring about too much additional resource consumption. From the current experiment, the results we obtained so far are promising.

**V. CONCLUSIONS**

We have presented a mutation-based Inferential Metamorphic Relation (IMR) and its mutation operators. Second, we presented logic expressions calculation based on a set of Logic Operators (LO). These logic expressions and logic operators can be used to generate IMR and deduce Inferential Testing Context (ITC). Finally, we presented an iterative IMT process based on IMR and the inferred ITC heuristically. Our empirical study on three benchmarking tools demonstrates the effectiveness of IMT on reducing FP in SQLIV penetration tests. For future work, we are studying on the application of the proposed approach to other vulnerabilities testing scenarios.

**ACKNOWLEDGMENT**

This research work was supported by the National Natural Science Foundation of China (Grant No. 61402264 and 61502260) and the Science and Technology Planning Project of Tianjin (Grant No. 17JCGCTJ30700).

**REFERENCES**


