Framework for Inspection-Based: Checking the Effectiveness and Efficiency in PHP Source Code

Jamilah Din and Saipul Bahari Hasan
Department of Software Engineering and Information System, Faculty of Computer Science and Information System, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia.
jamilahd@upm.edu.my

***Abstract*—Code inspection process is one of the software inspection processes that is used to find faults, check, increase, and maintain the quality of the software. Typically, the source code inspection process will be conducted in order to find sources code-related issues such as Logical Errors, and Structured Query Language (SQL) Injections. Currently, source code inspection process is being done manually by the developer which leads to taking a long time to find faults as well as time-delay. Based on the literature reviews that had been done, many researchers have done a lot of work in this domain, but none of them have developed prototype containing Logical Errors and SQL Injections for Hypertext Preprocessor (PHP) structure source code in one prototype. Therefore, this research proposed a framework for identifying Logical Errors and SQL Injections. A prototype is developed to proof the concept of the framework. The proposed framework is evaluated using the prototype in terms of effectiveness and efficiency by comparing the manual code inspection and the prototype-based code inspection. The result shows the prototype-based is more effective and efficient compared to current practice (manual).**

***Index Terms*—Code Inspection; Logical Errors; PHP; SQL Injections.**

I. INTRODUCTION

Software inspection is one of the activities that should be emphasized to ensure the quality of software products based on reducing the number of source code [13] and in terms of controlling and increasing software quality during the development process [10]. Source code and design document can be inspected by system developer before the testing phase is conducted [13]. Fagan [7] expresses inspections as a "formal, efficient and cost-effective technique of discovery errors in design and code".

Code inspection process is used to find faults and to check, increase, and maintain the quality of the software. Typically, the source code will be inspected after the code is written, before testing is done. It is usually performed by different person [13]. The preceding statements show that code inspection is an extremely important process to companies in saving time and increasing productivity.

"Inspection and acceptance testing prior to delivery (verification and validation) should be completed before it is handed over to the next stage. All submissions shall be certified in accordance with any of the following methods of inspection, analysis, demonstration or testing. For the development of application systems, inspection and testing shall be made throughout the project." [15].

A survey done by Ganssle [5] presents some striking example of the value for source code inspections as follows:

i. International Business Machines Corporation (IBM) was able to remove 82% of all defects before testing even takes place.
ii. American Telephone & Telegraph Company (AT & T) found that inspections led to 14% increase in productivity and tenfold increase in quality.
iii. Hewlett Packard Enterprise Company (HP) found 80% of the errors detected during inspection were unlikely to be caught by testing.

Based on above surveys, it can be concluded that many benefits will be obtained from the inspection such as reducing the debugging times during the inspection process and spending less time in the mind-numbing weariness of maintenance.

The main objective of this paper is to evaluate the proposed framework using prototype in terms of effectiveness and efficiency through comparing the manual-based way of doing code inspection and the proposed prototype-based experiment that the proof of concept of the framework. The paper focuses on the related work in Section II. Section III explains about design and implementation of the prototype. Method used for both experiment (manual-based experiment and prototype-based experiment) is discussed in Section IV and Section V is about the discussion of the whole results, while Section VI states the conclusion and recommendations.

II. RELATED WORKS

This section discusses the various processes, techniques, methods, solutions, framework and models used by previous researches. It is suggested in the process of improving inspection of PHP structure source codes; this is done in order to gain sight and comprehension of the previous similar solutions in the current problem and solutions which are being investigated. The problems and the solutions that are being investigated in this research is how to inspect the Logical Error and SQL Injection for PHP source code.

A. Logical Errors

Deulkar et.al. [4] proposed a new model to detect logical and syntactical errors using machine learning and data mining for Java source code. This study was done because it is difficult to recognize the syntactical and logical error during generating a program by programmers. Many steps need to be done in this study as; 1) compiler construction, 2) programming construction, 3) comparing the programs, 4) deducing the errors, 5) classifying the errors, 6) recommending and giving the right solution and 7) embedding the correct solution in a program, to produce the new model. Kästner [9] produced a tool named as Varis for PHP source code. It was used for PHP-based web application.

It delivers editor services on the client-side code to support syntax error highlighting, code auto-completion and "jump to declaration". Three (3) approaches have to be performed to complete the whole process which consists of: 1) symbolic execution, 2) variability-aware parsing and 3) analysis.

Nguyen and Chua [12] focused on logical error detector for PHP source code. On their research, framework has been designed in order to assist the PHP developers to classify logic error in source code and to mechanize the steps of noticing errors of the new prototype application that are being developed. Three (3) types of logical error in PHP were detected, 1) equality condition formulation, 2) while-loop condition formulation and 3) for loop expression formulation. Stergiopoulos er. al [19] aimed to detect logical error of source code and explore vulnerabilities in a fuzzy logic using Java source code. In the fuzzy logic, researchers joint some information about flow analysis to generate new code profiling. While, symbolic execution is used to check crosschecking for dynamic invariant. This study was done because the authors believed to decrease faults in software inspection, it will be one of the most cost-effective methods that can be used. The method involves in this study list as, 1) for an Applications Under Test (AUT), dynamic variants' form is used to create a symbol of performance program, 2) to collect some data about a set of execution paths and program states along these paths and input data vectors a map of all program points can be executed in different paths, Java Pathfinder (JPF) tools from NASA Ames Research Center (NASA) was used in the analysis and 3) logical error identified by crosschecking data accumulated with the dynamic invariants gathered.

B. SQL Injections

Jingling and Rulin [8] suggests a new framework for PHP application which is detecting the security vulnerabilities. It is a combination of two (2) analyses which consists of static and dynamic analysis. It has been completed in order to ensure the detection is more efficient. It has been known as HHVM (HipHop Virtual Machine) Based Static Analysis. This study was produced because of difficulty to detect security vulnerabilities. The vulnerabilities are focused on SQL Injection, Cross-Site Scripting (XSS) and any file inclusion. Shahriar et. al. [16] presents how to identify SQL Injection (SQLI) vulnerabilities by client-side in three PHP applications. Typically, inputted value from user is one of the SQL queries accepting by client-side. SQLI occurs on vulnerabilities found in the source code during the process of data input is ended. We believed that a client-side (browser) is a first point of SQLI attack. This framework provides detection of malicious inputs causing SQLI at the client-side early, but also relieves the server-side for additional checking and acts as a complementary solution to other existing approaches but it uncovers for a complex form of SQL queries and a stored procedure after the attack takes place.

Garg and Singh [6] focused and studied the vulnerabilities of web applications. Five (5) vulnerabilities were explained in this study. It can give some information for a lot of new researchers to solve the associated problem. We believe that server-side mechanisms really important for common distributed system and web application to ensure a security at the higher level. Based on the problem of this research study, the vulnerabilities had been identified as remote code execution, SQL injection, format string vulnerabilities, cross site scripting (XSS) and username enumeration. Researcher said attackers give more attention in SQL injection. Attacker can retrieve some important information through database for the system. In this study, researcher just demonstrated the vulnerabilities, countermeasure and the critically without produced any model to solve the vulnerabilities. Das et. al. [3] proposes a solution on how to solve SQL Injection Attacks (SQLIA) according to weaknesses in web application. The solution was given based on current method to identify the SQLIA and produces a new effective method which is called as an effective detection method (DUD). This method can detect the same problem in line with dynamic query matching. The DUD has high detection rate, simple detection tool and also suitable for notice syntactical rules, valid trusted string database and static or pre-generated program code checking. This study identified the susceptibilities in web application associated with SQLIA like, 1) Bypassing Web Application Authentication, 2) Getting Knowledge of Database, 3) Injection with UNION query, 4) Damaging with additional injected query and 5) Remote execution of stored procedure.

Based on the above explanation, it can be concluded that some researchers conducted studies in the PHP structure source code in terms of Logical Errors and SQL Injections but the researcher could not find one of the researcher who combined both of the Logical Errors and SQL Injections in order to develop inspection prototype based on the combined approaches (Logical Errors and SQL Injections). It was mentioned previously, Logical Errors and SQL Injections are very important in problem that must be considered during the programming of software. Most of developers always use those techniques during system development in order to check Logical Errors and SQL Injections of source code. It is always used by novice programmer as well as expert programmer [12]. SQL Injections inspection is very useful for tracing SQL Injection attempts by hackers which can be prevented from getting access to any important records from unauthorized users [16]. On the other hand, it is also related to illegality-related issues [16]. For Logical Errors inspection, it is very useful to detect any bugs, errors, faults or defects which programmers are unaware during system development. For instance, Logical Errors can give the wrong value to user without conscious. It is reliability -related problem in terms of software quality. In this paper, we proposed inspection process that combine the Logical Errors and SQL Injections for PHP structure source code and develop prototype because both of them are important in assisting the programmer to detect any bugs, faults and defect in the early stages of software development.

III. Design and Implementation

A. Proposed Framework

The framework for Logical Errors and SQL Injection is the process to show how both of characteristics can run until the result will be showed. For Logical Errors, all lines of source code will be read line by line in order to ensure that source code will be detected while SQL Injections can only examine through input data by users. Many processes in the framework need to be executed to produce the result to user. Figure 1 shows the proposed framework.

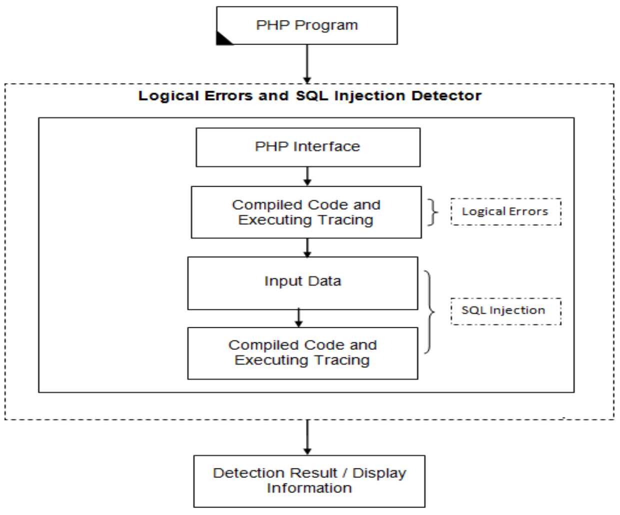

PHP Program
Logical Errors and SQL Injection Detector
PHP Interface
Compiled Code and Executing Tracing
Logical Errors
Input Data
SQL Injection
Compiled Code and Executing Tracing
Detection Result / Display Information

Figure 1: Proposed Framework

1) PHP Program Module

Any PHP structure program is used to inspect in this framework.

2) PHP Interface Module

Interface is used to accept the uploaded program file by programmer.

3) Input Data Modul

SQL Injections will use in this module to check the faults/vulnerability from the input data by users.

4) Compiled Code and Executing Tracing Module

A process in the program. The program can be inspected for two characteristics only which are Logical Errors and SQL Injections.

5) Detection Result / Display Information Module

Result from the program will be displayed on the screen to show whether it has an error or errors-free.

B. Prototype Design

Prototype design of the framework is defined in detail. This section covers information on the use case, activity diagram and screenshots of the framework interface.

1) Use Case

A prototype will be designed to prove the concept of the proposed framework. The prototype can only be used to inspect the Logical Errors and SQL Injections characteristics. At the same time, those characteristics can check the source code in one program. After the development is finished, the prototype will be measured based on effectiveness and efficiency for Logical Errors and SQL Injections as a quality of software.

Figure 2 shows the use case diagram of the prototype. This diagram shows the functionality of the prototype.

2) Activity Diagram

The workflow of the prototype process has been illustrated as Figure 3. The first process, prototype will trace any fault from the file uploaded by the user in terms of Logical Errors. After tracing Logical Errors has been done, the user will be asked through *pop up window* from framework whether to continue for SQL Injections or no. If the user clicks button 'Yes', interface for input data will be displayed and users can inspect SQL Injections through this screen using input data. Three (3) characteristics in Logical Error will be examined based on Equality Condition Formulation (ECF), While-loop Condition Formulation (WCF) and For Loop Expression Formulation (FEF) showed in Figure 4 and input data with ***' or 1=1--, 'or 1 =1 # '*** and ***' or '1' = '1'*** criteria for SQL Injections showed in Figure 5.

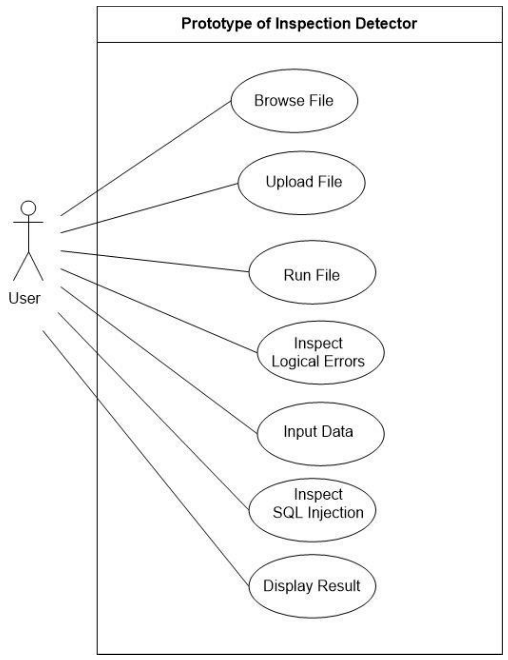


Figure 2: Use Case of Prototype

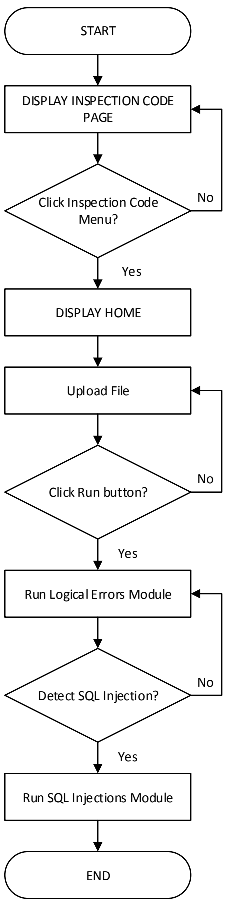

START
DISPLAY INSPECTION CODE PAGE
Click Inspection Code Menu?
No
Yes
DISPLAY HOME
Upload File
Click Run button?
No
Yes
Run Logical Errors Module
Detect SQL Injection?
No
Yes
Run SQL Injections Module
END

Figure 3: Activity Program of Prototype

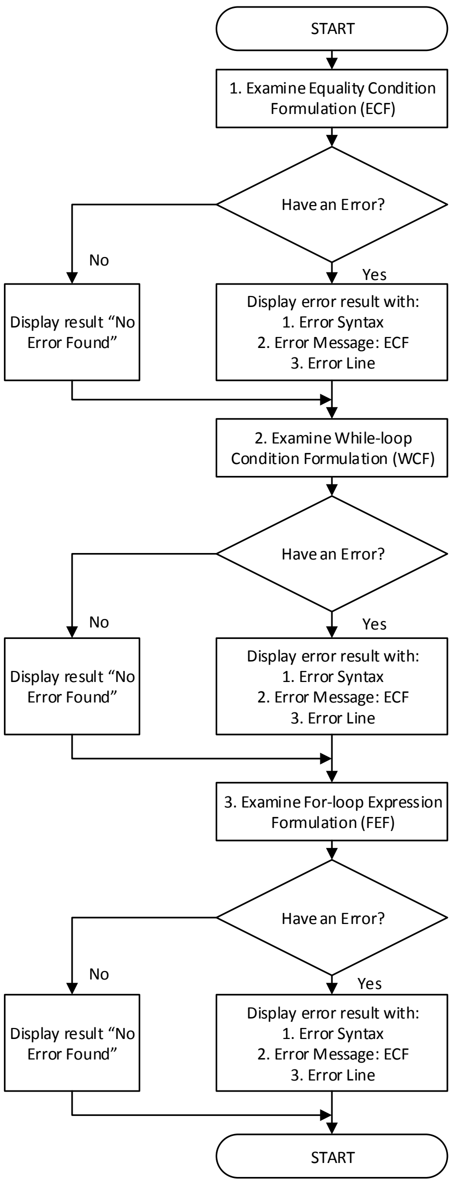

START
1. Examine Equality Condition Formulation (ECF)
Have an Error?
No
Yes
Display result "No Error Found"
Display error result with:
1. Error Syntax
2. Error Message: ECF
3. Error Line
2. Examine While-loop Condition Formulation (WCF)
Have an Error?
No
Yes
Display result "No Error Found"
Display error result with:
1. Error Syntax
2. Error Message: ECF
3. Error Line
3. Examine For-loop Expression Formulation (FEF)
Have an Error?
No
Yes
Display result "No Error Found"
Display error result with:
1. Error Syntax
2. Error Message: ECF
3. Error Line
START

Figure 4: Activity Program of Logical Error Modules

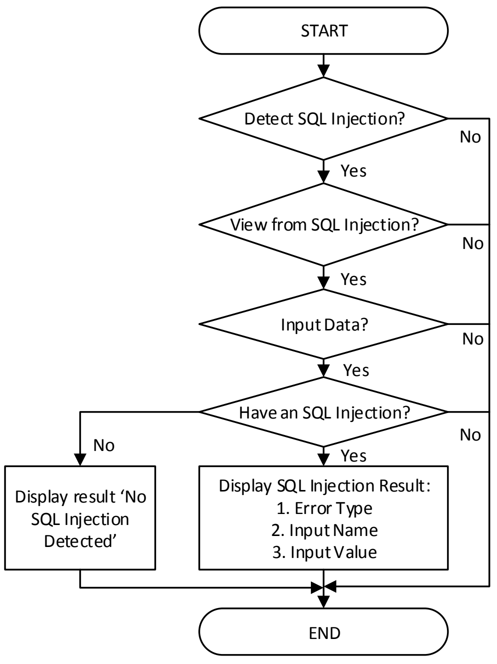
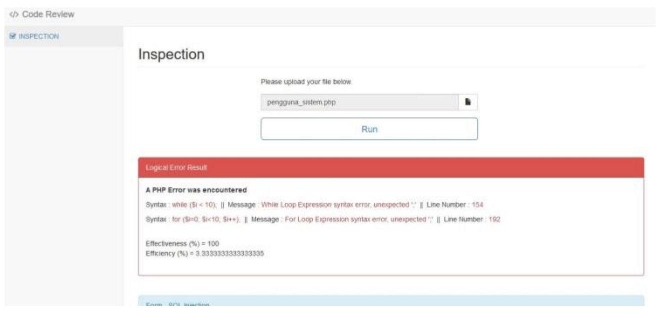


Figure 5: Activity Program of SQL Injections Modules

3) *Prototype Interface*

The screenshot of prototype is illustrated in Figure 6. It is used to prove the functionality of framework. At the end of the process, the result will be display whether 'No Error Found' or 'A PHP Error was encountered'.



A PHP Error was encountered

Syntax : while ($i < 10); || Message : While Loop Expression syntax error, unexpected ';' || Line Number : 154

Syntax : for ($i=0; $i<10; $i++); || Message : For Loop Expression syntax error, unexpected ';' || Line Number : 192

Effectiveness (%) = 100
Efficiency (%) = 3.3333333333333335

Figure 6: Prototype Interface

IV. EVALUATION

For the purpose to evaluate the prototype that has been developed to proof the framework, ten (10) programmers from Information Management Division, Ministry of Health Malaysia (IMD, MOH) who have 3-4 years programming experiences in PHP structure source codes have been chosen. All programmers were given five (5) samples programs which consist some errors for Logical Errors. The SQL Injections are tested through input data. Those programs were provided to measure the effectiveness and efficiency of the prototype. The samples were taken from a system (eTempahan Bilik Mesyuarat) using PHP structure source code in IMD, MOH. The programs have many lines to ensure the accuracy of the experiment. Explanation of the samples is shown in Table 1 below:

Table 1
Selection of Samples

ID	Files Name	Samples of Programs		
		LOC (Lines of Codes)	Total No. of Errors (Logical Errors)	Total No. of Vulnerabilities (SQL Injections)
1	pengguna_sistem.php	255	5	6
2	tambah_pengguna.php	346	7	18
3	borang_tempahan.php	609	9	24
4	tambah_bahagian.php	142	6	3
5	tukarkatalaluan.php	268	6	3

Logical Errors and SQL Injections of PHP structure source code can only be inspected in this paper. Logical Errors from Nguyen and Chua [12], have three (3) criteria were provided which are Equality Condition Formulation (ECF), While-loop Condition Formulation (WCF) and For Loop Expression Formulation (FEF) while SQL Injections from Sharma [18] were provided for attacking through input field with these conditions ***' or 1=1--, 'or 1 =1 # '*** and ***' or '1' = '1'***. Detail experimental procedures are explained below.

A. *Procedure of Manual-Based Experiment*

1) *Logical Errors*

All of programmers had been given five (5) samples which have some errors in those programs in softcopy form. After that, those programmers need to identify all errors in the source codes per hour. The result will be recorded for calculating the effectiveness and efficiency manually. These steps are used for manual-based experiment.

2) SQL Injections

The same samples with Logical Errors were given to programmers. Programmers need to identify which input field is vulnerable to hack by hackers per hour. Next, vulnerability identification by programmer will be inspected through real system to know that the vulnerability identification is correct or wrong. Based on the result, effectiveness and efficiency are calculated manually. Detail steps of Manual-Based Experiment are illustrated in Figure 7.

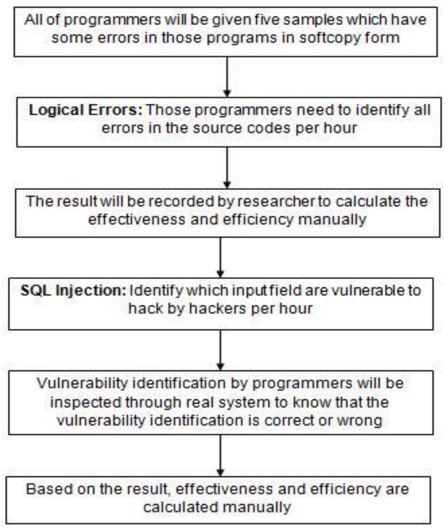

All of programmers will be given five samples which have some errors in those programs in softcopy form
Logical Errors: Those programmers need to identify all errors in the source codes per hour
The result will be recorded by researcher to calculate the effectiveness and efficiency manually
SQL Injection: Identify which input field are vulnerable to hack by hackers per hour
Vulnerability identification by programmers will be inspected through real system to know that the vulnerability identification is correct or wrong
Based on the result, effectiveness and efficiency are calculated manually

Figure 7: Steps of Manual-Based Experiment.

B. Procedure of Prototype-Based Experiment

1) Logical Errors

All programmers were given five (5) samples to be inspected. Those programmers will inspect the source code using the same laptop. Result based on effectiveness and efficiency automatically displayed by the prototype. A same laptop is used in order to get the consistent result.

2) SQL Injections

Through the previously listed vulnerability identification that was identified by programmers in manual-based experiment. It will also be inspected using this prototype to know the vulnerability of input field in those samples source code. Based on the result that was obtained by programmers, effectiveness and efficiency are calculated manually. Lastly comparison will be done for the result using real system and this prototype. Detail steps of Manual-Based Experiment are illustrated in Figure 8.

C. Evaluation on Effectiveness and Efficiency

Both of the results from manual-based experiment and prototype-based experiment will be compared to know which one is better. The comparison is evaluated and judged based on the effectiveness and efficiency of the Logical Errors and SQL Injections. Method for comparison of results followed the previous study by Oladele [14]. All results must be free false positive to obtain the accurate results. In the study Oladele [14], a false positive means the result should be correct even those programmers can find all the errors (Logical Errors and SQL Injections) as mentioned in specific time-frame. Method to calculate the effectiveness and efficiency for prototype-based experiment and manual-based experiment are shown as below.

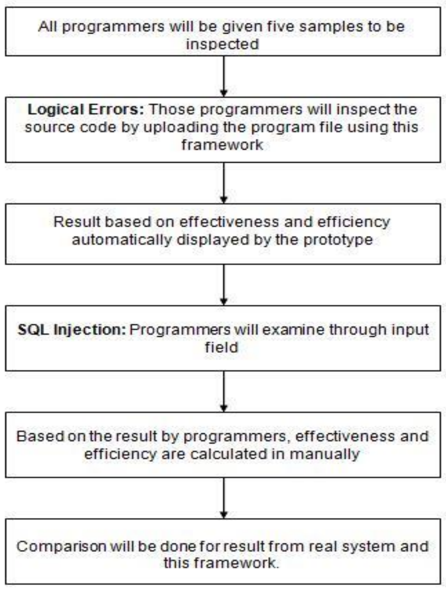

All programmers will be given five samples to be inspected
Logical Errors: Those programmers will inspect the source code by uploading the program file using this framework
Result based on effectiveness and efficiency automatically displayed by the prototype
SQL Injection: Programmers will examine through input field
Based on the result by programmers, effectiveness and efficiency are calculated in manually
Comparison will be done for result from real system and this framework.

Figure 8: Steps of Prototype-Based Experiment.

1) Logical Errors

a. Percentage of Effectiveness Calculation

Effectiveness refers to how many faults can be found by prototype dividing by total number of existing faults on the source code [14].

$$Effectiveness\,(\%) = \frac{a}{b}\,X\ 100 \qquad (1)$$

where: a = No of found fault in source code
b = Total number of existing fault

b. Percentage of Efficiency Calculation

Efficiency is referred as the number of found faults per hour [14].

$$Efficiency\ (\%) = \frac{e}{f}\,X\ 100 \qquad (2)$$

where: e = No of found fault in source code
f = 3600 seconds (per hour)

c. Average of Samples Calculation

Average will be calculated to all result samples.

$$Average\ Effectiven\,ess\,(\%) = \frac{All\ efectivene\,ss\ of\ result}{10} \qquad (3)$$

$$Average\ Efficiency\ (\%) = \frac{All\ efficiency\ of\ result}{10} \qquad (4)$$

2) For SQL Injections

a. Percentage of Effectiveness Calculation

Effectiveness refers to the number of real vulnerabilities detected dividing by total number of reported [11].

$$Effectiveness\ (\%) = \frac{g}{h}\ X\ 100 \quad (5)$$

where: g = No of real vulnerabilities detected
h = Total number of reported

b. Percentage of Efficiency Calculation

Efficiency is referred to the number of revealed vulnerabilities by programmers/hackers dividing by total number of revealed vulnerabilities by programmers/hackers reported [17].

$$Efficiency\ (\%) = \frac{i}{j}\ X\ 100 \quad (6)$$

where: i = No of revealed by hackers
j = Total number by hackers need to reveal

c. Average of Samples Calculation

Average will be calculated to all result samples.

V. Result and Discussion

For prototype-based experiment, effectiveness and efficiency results are generated by the prototype for Logical Errors. The formula is included in source code during process of system development. While, SQL Injections are counted manually because the method that was used unsuitable to count the result automatically.

Table 2 shows the comparison of the users for logical errors, and Table 3 shows the result for SQL injection.

Table 2
Comparison Result of Logical Errors

	Manual Experiment		Prototype Experiment	
ID Programs	Effectiveness (%)	Efficiency (%)	Effectiveness (%)	Efficiency (%)
1	90.000	0.125	100.000	0.139
2	97.142	0.185	100.000	0.194
3	90.000	0.225	100.000	0.250
4	94.500	0.159	100.000	0.167
5	96.667	0.161	100.000	0.167

Table 3
Comparison Result of SQL Injections

	Manual Experiment		Prototype Experiment	
ID Programs	Effectiveness (%)	Efficiency (%)	Effectiveness (%)	Efficiency (%)
1	94.500	1.103	100.000	11.025
2	95.556	3.327	100.000	33.269
3	95.833	4.448	100.000	44.874
4	90.000	0.522	100.000	5.222
5	93.333	0.541	100.000	5.609

Figure 9 shows the prototype-based experiment which present clear result compared to manual-based experiment. The result of prototype-based experiment shows 100% of all samples where it could identify all existing errors in program files compared to manual-based experiment. While, the result obtained from the manual-based experiment is 90% (sample 1), 97.142% (sample 2), 90% (sample 3), 94.5% (sample 4) and 96.667% (sample 5). For the Logical Errors, the prototype-based experiment that was proposed is more effective rather than manual-based experiment method. Similarly, Figure 10 presents that prototype-based experiment is really good in terms of efficiency compared to manual-based experiment. The difference result is insignificant between both experiments but prototype-based experiment result has higher result than manual-based experiment. All errors can be found in one hour by prototype-based experiment faster than manual-based method.

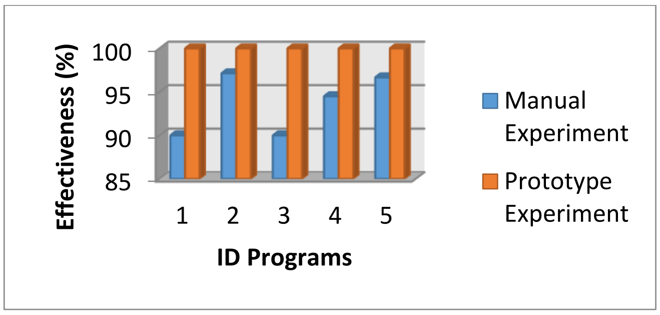


Figure 9: Comparison of Effectiveness for Logical Errors

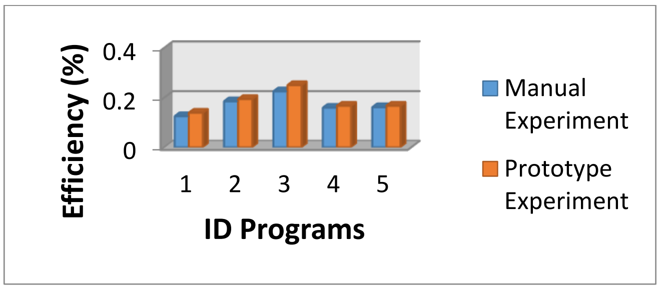


Figure 10: Comparison of Efficiency for Logical Errors

Figure 11 revealed that all samples to detect vulnerability of SQL Injection using prototype-based experiment were more effective than manual-based experiment. 100% was obtained by prototype-based experiment and 94.5% (sample 1), 95.556% (sample 2), 95.833% (sample 3), 90% (sample 4) and 93.333% (sample 5) were obtained from manual-based method. Prototype-based method can detect all total number of vulnerabilities reported. Figure 12 demonstrates slight difference between both experiments for SQL Injections but prototype-based method used by programmers is still more efficient than manual-based method. All results of samples are shown by prototype-based method is higher than manual-based method. Prototype-based experiment can inspect all previously identified vulnerability suggested by programmers. To sum up, the proposed and developed prototype-based experimented can address the problems of both characteristics (Logical Errors and SQL Injections) for PHP structure source code. On the other hand, it is more effective and efficient compared with manual-based method. It can be seen through analysis and result from Figure 9 to Figure 12.

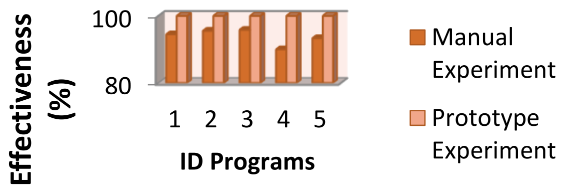


Figure 11: Comparison of Effectiveness for SQL Injections

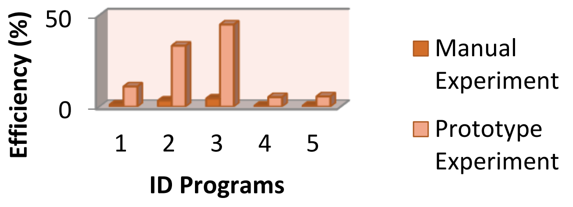


Figure 12: Comparison of Efficiency for SQL Injections

VI. Conclusion and Recommendations

The inspection process is divided into two (2) categories which are requirement inspection and source code inspection. This study focuses on source code inspection. Referring to the main objective, this research proposed a framework for identifying Logical Errors and SQL Injections for PHP structure source code. The PHP structure source code area had been chosen because there are only a few researchers involved in this area. Most of them focused on JAVA or C# source code. Some modules were produced to ensure the framework can inspect Logical Errors and SQL Injections accurately.

Five (5) modules have been identified consisting of PHP Program Module, PHP Interface Module, Input Data Module, Compiled Code and Execution Tracing Module and Detection Result/Display Information Module. Those modules play pivotal roles respectively to make sure the results are correct. To prove that the framework and the two (2) developed prototypes are efficient and effective for checking Logical Errors and SQL Injection, Equality Condition Formula (ECF), While Loop Condition Formulation (WCF) and For Loop Expression Formulation (FEF) were used as characteristic in Logical Errors to be tackled while SQL Injections focused to input data by users with ***or 1=1--, 'or 1 =1 # '*** and ***' or '1' = '1'*** criteria. Two (2) experiments were conducted to measure the effectiveness and efficiency of developed prototype. Manual-based experiment involves programmers in Information Management Division, Ministry of Health Malaysia which were needed to inspect the source code and input data manually. For prototype-based experiment, those programs were examined using this framework. Comparison for both results were performed to prove that the prototype is better compared to the manual based in terms of effectiveness and efficiency. There are several findings and recommendations to be highlighted in order to enhance this framework in further research which are add more criteria for SQL Injections to ensure that the framework is precise, conduct the experiment in large number of programmers and subject code and PHP Object Oriented Programming can be inspected using this framework.

Acknowledgment

Thank you to the Ministry of Education (MOE) and Research Management Center, Universiti Putra Malaysia (UPM) for the financial supports through FRGS Vote No: 08-01-15-1726FR.

References

[1] O. S. Akinola, and A. O. Osofisan, "An empirical comparative study of checklist- based and ad hoc code reading techniques in a distributed groupware environment," *International Journal of Computer Science and Information Security*, vol. 5, no. 1, pp. 25–35, 2009.

[2] A. Bacchelli, and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proc. International Conference on Software Engineering*, 2013, pp. 712–721.

[3] D. Das, U. Sharma, and D. Bhattacharyya, "An approach to detection of SQL injection attack based on dynamic query matching," *International Journal of Computer*, vol. 1, no. 25, pp. 28–34, 2010.

[4] K. Deulkar, J. Kapoor, P. Gaud, and H. Gala, "A novel approach to error detection and correction of c programs using machine learning and data mining," *International Journal on Cybernetics & Informatics*, vol. 5, no. 2, pp. 31–39, 2016.

[5] J. G. Ganssle, "A guide to code inspection," Available at http://www.ganssle.com/Inspections.pdf, Retrieved on 02 June 2017, 2001.

[6] A. Garg, and S. Singh, "A review on web application security vulnerabilities," *International Journal*, vol. 3, no. 1, pp. 226, 2013.

[7] M.E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no.3, pp. 182–211, 1976.

[8] Z. Jingling, and G. Rulin, "A new framework of security vulnerabilities detection in PHP web application," in *Proc. 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2015, pp. 271–276.

[9] H.V. Nguyen, C. Kästner, and T.N. Nguyen, "Varis: IDE support for embedded client code in PHP web applications," in P*roc. International Conference on Software Engineering*, 2015, pp. 693-696.

[10] P. G. Koneri, G. De. Vreede, D. L. Dean, A. L. Fuhling and P. Wolcott, "The design and field evaluation of a repeatable collaborative software code inspection process," in *International Conference on Collaboration and Technology*, 2005, pp. 325–340.

[11] A.P.S. Matsunaga, N. Antunes, and R. Moraes, "Coverage metrics and detection of injection vulnerabilities: an experimental study," in *Proc. 12th European Dependable Computing Conference (EDCC)*, 2016, pp. 45–52.

[12] T. Nguyen and C. Chua, "A logical error detector for novice PHP programmers," in *Proc of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2014, pp. 215–216.

[13] **A.** Nuc, and C. Ivan, "REVEDERE – Distributed support system for code review process," *International Journal of Computer Application*, vol. 115, no.14, pp. 1–6, 2015.

[14] R. O. Oladele, and H. D. Adedayo, "On empirical comparison of checklist-based reading and adhoc reading for code inspection," *International Journal of Computer Application*, vol. 87, no. 1, pp. 35–39, 2014.

[15] Pekeliling, S., and Bil, P. (2013). Pemilikan Kod Sumber (Source Code) dan/atau Intellectual Pemindahan Teknologi (Transfer of Technology) - Pemindahan, 1(iii).

[16] H. Shahriar, S. North, and W. Chen, "Early Detection of Sql Injection Attacks," *International Journal of Network Security and its Application*, vol. 5, no. 4, pp. 53–65, 2013.

[17] A. A. M. Sharadqeh, M. Alnaser, O. Al. Heyasat, A. A. Abu-ein, and H. Moh, "Review and measuring the efficiency of SQL injection method in preventing e-mail hacking," *International Journal of Network Security and its Application*, vol. 5, no. 6, pp. 337–342, 2012.

[18] C. Sharma and C.S. Jain, "SQL injection attacks on web applications," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 3, 1268–1272, 2014.

[19] G. Stergiopoulos, P. Katsaros, and D. Gritzalis, "Automated detection of logical errors in programs," in *Lecture Notes in Computer Science (LNCS)*, Springer-Verlag Berlin Heidelberg, vol. 8924, 2014, pp. 35-51.

[20] H. Uwano, M. Nakamura, A. Monden, and K. Matsumoto, "Analyzing individual performance of source code review using reviewers' eye movement," in *Proc. Eye Tracking Research & Applications (ETRA)*, 2006, pp. 133–140.