International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Strengthening Web Application Security: A Penetration Regression Test Selection Algorithm for Early Detection of Buffer Overflow Vulnerability

Shilpa R. G.*[1], Jhilmil Basu[2], T. P. Pushphavathi [3], P. V. R. Murthy [4]

Submitted: 04/02/2024 **Revised:** 11/03/2024 **Accepted:** 18/03/2024

Abstract*:* Web applications are prime targets for security breaches, making rigorous regression testing essential to prevent adverse impacts from modifications or enhancements. The aim of regression testing is to ensure that improvements or modifications to a program's functionality do not adversely affect its current operations. Regression testing is essential as it reduces the size of the test suite, thus reducing the time and effort for testing as a system or application is modified. Regression test selection methods are used widely in functional testing but not addressed in context with penetration or security testing. The traditional regression testing techniques and code coverage (branch coverage) based test adequacy measurements, are found to be inadequate. This paper proposes a novel algorithm for penetration regression test selection along with extended branch coverage criteria predominantly focusing on buffer overflow vulnerability. The algorithm is based on the control-flow structure of the program. Additionally this approach provides a systematic method to detect buffer overflow vulnerability in the unit testing phase of early software development life cycle for the practitioners.

Keywords: *Branch coverage, Buffer overflow, Code coverage, Penetration testing, Regression testing, Regression test selection, Security testing, Vulnerability*

1. Introduction

Web applications are susceptible to a surplus of security vulnerabilities, which malicious hackers can exploit to compromise their availability, integrity, or confidentiality. Insufficient input validation outlooks out as a significant security apprehension for the web applications [1]. Organizations are now concentrating more on strengthening the security posture of their web applications or software applications in view of these issues. This includes setting strong authentication procedures in practice, encrypting sensitive data, patching software components on a regular basis to eliminate known vulnerabilities, and performing comprehensive security assessments that encompass vulnerability scanning and penetration testing.

Penetration testing is employed to discover security weaknesses in web applications. Safeguarding sensitive data from hackers who might gain unauthorized access to the application is the primary objective of penetration testing. To identify flaws, penetration testers might simulate attacks on an intended web-based application through penetration testing [2].

Since decades, one of the most well-known safety risks is the buffer overflow vulnerability [3]. It is a kind of software vulnerability which occurs when a program attempts to write additional data than it was designed to hold to a buffer, which is a temporary storage section in computer memory. Data overflow into neighbouring memory locations might occur when the extent of data being written is higher than the buffer allocated for it. Buffer overflow vulnerabilities can be the basis for major glitches, such as corrupting data, crashing the impacted program, or even giving attackers access to run arbitrary code on the system with the privileges of the compromised program. Illegal access, privilege escalation, or the execution of malicious commands are likely consequences of buffer overflow [4].

Software developers frequently employ regression testing, which is one of the most common approaches for ensuring the efficacy of software products during development cycles [5] [6]. The implementation time of the testing process can be significantly reduced by employing regression testing techniques [7].

Regression testing ensures that recent changes to code haven't affected the application's already-existing functionalities. In software applications, there is always the possibility that adding, modifying, or removing code might result in the emergence of novel issues or unexpected behavior even in the parts of the program that were functioning properly before. Regression testing ensures that these modifications do not result in the failure of previously developed and evaluated software features.

Regression testing ensures that when new functionality is added to a program or a set of programs, the overall workability of the product remains unaffected. Regression testing has been applied largely in the context of functional testing but not much attention is paid in the context of security testing. Many classes of vulnerabilities may exist in a program or a set of programs. According to OWASP's most current report on top ten vulnerabilities, buffer overflow is listed as the third most exploited vulnerability. Regression testing is sensitive to buffer overflow vulnerability.

In the process of penetration testing of a web application, information gathered pertains to vulnerabilities, from all relevant sources, in the application that is being developed or released. When a software application is modified, identification of relevant

1,2,3,4 Faculty of Engineering and Technology, M.S. Ramaiah University of Applied Sciences, Bangalore, India
2ORCID ID: 0000-0001-9247-1843
3ORCID ID: 0000-0003-2322-1378
4ORCID ID: 0000-0001-6398-4762
** Corresponding Author Email: shilparg.ms.mc@msruas.ac.in*

old information pertaining to previous release and new information acts as an input to certain activities (For example to guide regression test selection activity).

Existing and new vulnerability information has an impact on selecting which old penetration tests to be rerun and which new penetration tests to be designed and run. Only the relevant subset of old penetration tests and new penetration tests are to be run based on modification to application. Testing requires less effort and time as a result selecting regression tests is a significant challenge for security testing in addition. Hence time and effort for testing is also reduced. Regression test selection is a relevant problem for security testing too.

This paper proposes a systematic method regression test selection algorithm to detect buffer overflow vulnerability. The main objective is to develop relevant and effective algorithms for regression test selection. The proposed regression test selection algorithm is recommended to developers, especially when using unit testing. The algorithm aids the developer by mapping the security tests to paths in the program. Regression test selection methods are used widely in functional testing in practice, but not addressed in context with penetration testing. In the context of penetration testing regression test selection is not addressed adequately. The paper also provides a systematic method to detect buffer overflow vulnerabilities in the unit testing phase of early software development life cycle for the practitioners.

1.1. Abbreviations and Acronyms

SUT: System Under Test
OWASP: The Open Web Application Security Project
RTM: Regression test minimization
RTS: Regression test selection
TCS: Test case prioritization
CFG: Control Flow Graph
TU: Test Suite

1.2. Organization of Paper

The following sections contributes remaining parts of this paper: In Section 2, we investigate into the theoretical underpinnings of the regression testing process and explore existing methods in the field. Section 3 precisely outlines the systematic approach to penetration regression test selection and a regression test selection algorithm to detect Buffer Overflow Vulnerability. Following that, concept of code coverage criteria and algorithm for extended branch coverage algorithm is discussed. In Section 4, we analyze the outcome of a Buffer Overflow Vulnerability attack on code and also discuss the results. Finally, Section 5 offers concluding observations and insights drawn from the study's findings. Additional insights assembled from these sections provide an inclusive appreciative of the role of penetration regression test selection in augmenting the security posture of the software applications.

Novel Contributions addressed in the paper are as below:

- An algorithm for penetration regression test selection
- Extended branch coverage criteria predominantly focusing on buffer overflow vulnerability
- Systematic method to detect buffer overflow vulnerability for practitioners

2. Related Work

Regression test minimization (RTM), regression test selection (RTS), and test-case prioritization (TCP) are the three fundamental techniques for performing regression [8]. Test cases considered redundant or almost identical will be deleted or eliminated using the RTM approach. Test cases that achieve a set of criteria will be selected by RTS. Finally TCP will prioritize the test cases employing the previously established criteria [9].

Regression testing techniques primarily specific to evaluating modifications to existing software and its effected portions reduce effort by identifying changes and their effects. At the code or model level, the modification evaluation of impact can be carried out [10].

Regression testing techniques are divided into three groups by test case minimization, test case prioritization and test case selection [11]. Regression testing methods are primarily reported with an emphasis on test case selection. The challenge of selecting a subset of test cases to be employed in the software's change detection tests has been addressed by regression test selection (RTS) techniques. It requires selecting a portion of the tests from the prior iteration that are more probable to find faults using various approaches [12]. A regression test selection technique is proposed that that identifies changed components in programs belonging to a web application [13]. A technique introduced is a tool called SoRTEA that is designed and developed using the method call trace based approach. Regression tests are selected for web applications written in java using this approach. Static analysis is performed on the original and modified versions of the web application [14]. However, the technique omits regression test selection especially when it pertains to security.

The four categories of regression test selection techniques are as follows: ad hoc random, safe, coverage-based, and minimization [15]. Test cases are selected using techniques based on coverage, which identify the modified portions of the System Under Test (SUT) and the path and data dependence graph covered by tests [10].

In accordance to related work, there are no security coverage requirements to help developers and testers check whether security tests have covered every application vulnerability location. Moreover, little attention is accorded to a more systematic investigation of regression test selection for security. (Specifically, native code (C) is also used by web applications built on Java technology. Such web applications' security may be compromised by vulnerabilities in C code. Various techniques for the regression test case selection applied for several programming paradigms are discussed [16]. For the objective of choosing test cases in regression testing, an approach integrating the class and state diagrams is put forth [17]. When any changes are made to the code, the class and the state diagram are modified. The components that were modified are indicated. Cases suitable for the change transitions are examined. Based on the code coverage of the test cases, a novel algorithm for test case prioritization is proposed [18]. Techniques for code-based regression testing are employed to identify vulnerabilities. In connection with regression testing techniques and security issues, it outlines how different methods are commonly used for security regression testing.

Despite significant improvements in vulnerability detection techniques that incorporate both static and dynamic analysis, buffer overflow attacks continue to be common [19]. Although buffer overflow vulnerabilities can be efficiently identified by static analysis techniques, there are additionally false negatives that may be reported [19]. Systematic testing of buffer overflows is attempted by the tool STOBO, however, the coverage metric used is termed interesting function coverage, subsumed by statement coverage. The STOBO tool intends to execute systematic testing of buffer overflows [20]. Coverage metric is employed as a subset of statement coverage however.

Due to limited budget and tester time constraints, the execution of all generated test cases remains unnoticed [21]. The issue of selecting which set of test cases to run becomes more and more crucial.

This procedure minimizes the test case set's size while ensuring that the test results' quality won't be affected. The test coverage on the intended components to be tested for the applications under test is significantly affected by the test case selection [21]. As an instance, the traditional approach of test case selection selects among the created test cases according to user preferences or pre-established methods and formulas [22].

The research on the topic of regression testing has been drawing a lot of attention recently. Regression testing in the context of functional testing is widely studied. However, not much attention is paid in the context of non-functional requirements like security. In regard to penetration security testing no systematic method of regression test selection is available as of our knowledge. One of the concern with buffer overflow vulnerability testing which is not consistently addressed is the technique for creating a sufficient number of tests for a program that meet the adequate coverage criteria for buffer overflow, taking into account how significant and crucial the vulnerability is. All these gaps are addressed in this work.

3. Proposed work

Regression testing assists to reduce the size of the test suite, which makes it essential when modifying a system or application. Testing takes less time and effort as a result. Choosing regression tests is a significant problem for security testing as well. Regression test selection mechanisms in the context of penetration testing are one particular activity. When designing regression tests for a penetration test, it is crucial to take into account the variations between previously known vulnerabilities in the program and recently discovered vulnerabilities, in addition to pertinent application pathways.

The aim of this research is to develop relevant and effective algorithms for such identified and chosen activities. It may be noted that the activity of penetration regression testing is carried out in test design as well as test automation/execution phases, however, the activity regression test selection is triggered by information gathered indicating the event that the application is modified.

An application may be modified because functionality (features) have changed or new ones introduced, or, only the code is changed for fixing bugs with the functionality remaining the same. Regression testing techniques can be employed to select a subset of an existing program's test suite, which decreases testing costs. Regression test selection methods are used widely in functional testing but not addressed adequately in context of penetration testing.

3.1 Penetration Regression Test Selection (When Functionality/Code Changes)

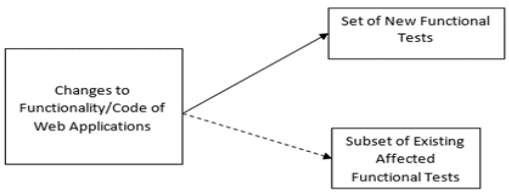


Fig.1. Penetration Regression Test Selection when functionality/code changes

Fig.1 illustrates the process of Penetration Regression Test Selection when Functionality/Code changes.

We address the regression test selection problem in the case wherein software is dynamically run during security testing to show the presence of vulnerabilities. In white-box or structural testing of a program P, with a set of test cases T, when the program is modified to P', with a corresponding set of test cases T', regression test selection is carried out essentially by considering the difference between P' and P in terms of new statements or branches [23].

In structural testing, a code coverage criterion such as statement or branch coverage is a major concern [23]. The idea is that the subset of the old test cases that traverse the modified statements or branches in the code (P') shall be considered by the regression test selection algorithms. Furthermore, new test cases may be required either because requirements changed or the structure of the program has undergone a change depending on whether test cases are designed based on the functionality or the structure of the program.

To motivate our contribution, in regression test selection in security testing, we provide a simple example below. Consider program P for which the pseudo code is given below in Fig.2.

```
/* allocate a buffer buf of size MAX */
   buf = malloc(MAX*sizeof(char));
…
if (some condition)
… buf[i] … ------(1)
else
   … buf[j]…; ------(2)
```

Fig.2. Pseudo Code for Program P

The potential vulnerability sites in the above program P with respect to buffer overflow are (1) and (2). We need security tests that cover both the vulnerability sites mentioned above. Let us assume that there are two tests T1 and T2 to cover sites (1) and (2) respectively. Say the program is modified to P' as shown in Fig.3.

```
…
if (some condition)
… buf[i+1] … ------(1)
else
… buf[j]…; ------(2)
…
… buf[k]… ------(3)
```

Fig.3. Pseudo Code for Program P'

Now, the potential vulnerability sites in P' with respect to buffer overflow are (1), (2) and (3). New tests T3 (T4) may need to be designed to cover the potential vulnerability site (3) that is newly introduced. Of course, based on other factors such as the relative value of k to (i+1) and j and exploiting the structural properties of the program P', it may be inferred that a test traversing through vulnerability sites (1) or (2) in P' may also traverse through (3) but this may not be true in general, if an overflow occurred at (1) or (2) before reaching (3). Thus, in general, a new test may need to be designed to traverse through vulnerability site (3) in P' (when compared to the test suite for P). In addition, the old test T1 of P may need to undergo a change, or, at least executed on P' as there is a modification in site (1), with the subscript of buf changing from i to (i+1).

Thus, the algorithm for regression test selection in security testing can be based on the potential vulnerability sites that need to be re-

visited in a modified program P' and the newly introduced potential vulnerability sites in P. Buffer overflow errors can cause denial of service attacks. For each class of errors that lead to vulnerabilities in programs, the algorithm is supplied with the knowledge to infer corresponding potential vulnerability sites (for example, format string problem [24].

3.2 Systematic Procedure for Penetration Test Regression Test Selection (When Code Changes)

An algorithm that guides regression test selection in security testing is described below. The CFG (control flow graph) for each program unit in program P is constructed. CFG' for each program unit in P' is constructed. A CFG consists of basic blocks as nodes along with nodes for controlling flow. Potential sites are statements within a basic block. Relevant statements are annotated as potential vulnerability sites for different classes of errors such as buffer overflow and format string problem.

An operation called P' – P or CFG' – CFG is defined which is a set of sequences of edges or paths capturing, essentially, modified parts of the code (from P to P'). P'-P consists of newly introduced vulnerability sites, or, modified but existing vulnerability sites in P. The vulnerability sites may be modified, either directly or indirectly, through different values or expressions flowing into the existing site (data flow).

3.3 Algorithm for Regression Test Selection

The algorithm below is intra-procedural.

Step 1: Construct CFG for a program unit U; also construct the set of vulnerability sites in U.

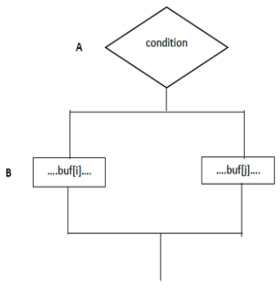
Step 2: Incrementally design test cases based on the specifications of U (also considering security requirements) and form a test suite TU. (Assume that TU is an adequate test suite with respect to the set of vulnerabilities in program unit U).

Step 3: Program unit is modified to form U' by developer (or CFG is modified and is now CFG').

Step4: Construct the modified set of vulnerability sites by performing CFG' – CFG for the program unit U.

Step 5: Design test cases to traverse the newly introduced vulnerability sites which are a subset of CFG'-CFG. For the other subset of CFG'-CFG which corresponds to existing vulnerability sites in PU or CFG but are modified , either, pick existing old test cases, if still relevant, or else design new tests.

To extend the algorithm to an inter-procedural one, once a call node is encountered in CFG in step 1, the data flow information pertaining to the state of variables at the time of call is maintained and kept track of while computing the set of vulnerability sites within the called program unit. This is repeated until the leaves or program units that do not call others are encountered. Steps 4 and 5 also need to be modified accordingly.



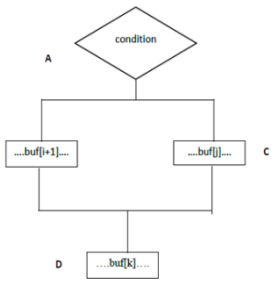


Fig.4. CFG for Program P **Fig.5.** CFG' for Program P'

Fig. 4. and Fig. 5 are the control-flow graphs (CFG and CFG') of the programs given in Fig. 2 and Fig. 3 respectively. Potential vulnerability sites in Fig. 4 are at nodes B and C with respect to possible buffer overflow error. Potential vulnerability sites in Fig. 5 are at nodes B, C and D, wherein C is unmodified from the previous version of the program. (CFG' – CFG) indicates changes in the sub-path AB (index i changed to (i+1)) at node B in CFG', in path ABD wherein D is a new node with buf[k] and in path ACD as well. Existing tests are used with or without modification and new tests designed based on the difference (CFG'-CFG) and data flow information at the modified vulnerability sites.

3.4 Code Coverage

In Traditional branch coverage, a set of tests may cover all the branches in a program, but it doesn't necessarily cover all the sites prone to buffer overflow vulnerability. A function processFunction() has been considered to illustrate traditional branch coverage.

```
void processFunction(int *buf1[10], int *buf2[5])
{
        int *i , *c1, *c2, *num1, *num2;
        int x1 = 9, x2 = 4, x3 = 5 , x4 = 0 ;
        i = (int *) malloc(sizeof(int));
           c1 = (int *)malloc(sizeof(int));
           c2 = (int *)malloc(sizeof(int));
           num1 = (int *)malloc(sizeof(int));
        num2 =  (int *)malloc(sizeof(int));
          *num1 = 20;
        *num2 = 100;
          if (* c1 > *num1)
        {
                *i = x1;
        }
        else
        {
                *i = x2;
        }
        if ( *c2  < *num2 )
        {
                *buf[i] = x3;
           }
        else
        {
                *buf2[i] = x4;
           }
}
```

Fig.6. Function processFunction()

The program unit processFunction() takes two pointer references of integer type. Throughout the program there are several pointer declarations and initializations. There are two if constructs in the program. A pointer variable i is considered that has multiple definitions across the decision constructs.

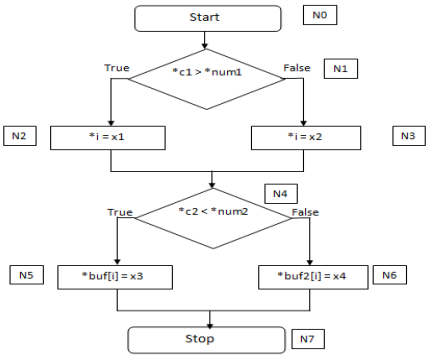


Fig.7. A Typical Control Flow Graph

Fig. 7 illustrates a control flow graph for a program unit. Two tests,

say T1 and T2, are chosen to achieve complete traditional branch coverage for the control flow graph. On applying traditional branch coverage, the following inferences can be derived.

- The test case T1 is chosen in such a way that it takes the path N0->N1->N2->N4->N5->N7. The path taken doesn't overflow the buffer at node N5.
- The test T2 is chosen in such a way that it takes the path N0->N1->N3->N4->N6->N7. The path taken doesn't overflow the buffer at node N5.

As evident from the figure, two test cases T1 and T2 were sufficient to achieve complete branch coverage. However, consider the path N0->N1->N2->N4->N6->N7. The buffer at node N6 can hold 5 elements. If this path is taken, a buffer overflow happens at node N6 since the buffer tries to reference the ninth element, which doesn't exist for the buffer at N6. Even though 100 percent branch coverage was achieved by traditional branch coverage, it didn't necessarily cover all the branches that may have led to buffer overflow. Therefore, extended branch coverage is proposed that ensures that all the buffer overflow sites are exercised with respect to their most recent buffer index definitions.

3.5 Extended code coverage criteria measurement

Extended branch coverage algorithm has been proposed as traditional branch coverage doesn't necessarily cover all the branches having potential vulnerability sites (buffer overflow vulnerability). To define an extended branch coverage to address coverage of vulnerabilities, it is not merely (structural) branch coverage that is important but whether an edge containing a subscript operation is preceded by an edge defining or modifying the index in a test.

With the introduction of extended branch coverage, the regression test selection tool not only determines all the possible paths from decision points in a program, but also reports the branches that may be possible vulnerability sites that may be exploited. Traditional branch coverage cannot be exercised in security testing because vulnerability sites may go unseen. Hence, the extended branch coverage allows this additional improvement with confidence.

3.6 Extended Branch Coverage Algorithm

Algorithm 1: Algorithm ExtendedBranchCoverage
Begin
Step 1: Construct two control flow graphs CFG and CFG' for the original and modified input programs respectively.
Step 2: Traverse each and every edge in a control flow graph of a program
Step 3: For each control flow graph, collect all the subscript sites.
Step 4: For each collected subscript site, collect the most recent definitions of the subscript variable index along different paths that reach the subscript variable site as a use operation.
Step 5: Apply regression test selection algorithm only on paths where the subscript site is reachable from the most recent definition of its index variable.
End

The extended branch coverage algorithm begins by considering two graphs CFG and CFG'. CFG is the control flow graph of the original input program and CFG' is the control flow graph of the modified input program. All the possible paths in the control flow graph are recorded. For each control flow graph, all the subscript sites are collected. Once the subscript sites have been collected, the most recent definitions of the subscript index are mapped to the subscript site.

4. Results and discussion

The selection of the buffer overflow vulnerability stems from the predominant use of C code within web applications, frequently alongside languages like C++ or Java. Proposed algorithms for regression testing in security, precisely targeting buffer overflow vulnerabilities, go beyond mere consideration of the program's control-flow structure. They extend to encompass broader branch coverage criteria, taking into account not only the control-flow paths but also the definitions of index variables. This approach is critical for detecting potential buffer overflow sites, where index variables may accidentally trigger vulnerabilities. By incorporating these subtle considerations, such algorithms improve the robustness of security testing conventions, stimulating web applications against potential exploitation.

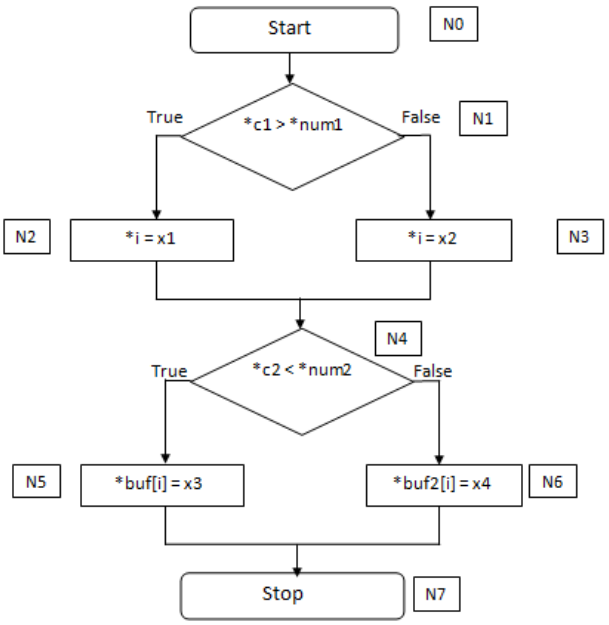


Fig.8. Illustration of Extended Code Coverage and Regression Testing

Traditional Branch Coverage: Two test cases are chosen that satisfy hundred percent branch coverage. The paths traversed by the two test cases are:-

- N0->N1->N2->N4->N5->N7 (1 test for *c1 > *num1 and *c2 < *num2)
- N0->N1->N3->N4->N6->N7 (1 test for *c1 < *num1 and *c2 > *num2)

However, buffer overflow occurs at *c1>*num1 and *c2 > *num2. According to the extended branch coverage and regression test selection algorithm the process is exercised.

(i) Subscript sites are present at nodes N5 and N6.

(ii) For node N5, two possibilities exist for mapping the most recent definition of subscript index to the subscript site as shown in Table 1.

Table 1. Test cases for traditional branch coverage (a)

Condition	Subscript Site Node	Most Recent Subscript Definition
*c1 > *num1 and *c2 <*num2	N5	i = x1
*c1 < *num1 and *c2<*num2	N5	i =x2

(iii) For node N6, two possibilities exist for mapping the most recent definition of subscript index to the subscript site as shown in Table 2.

Table 2. Test cases for traditional branch coverage (b)

Condition	Subscript Site Node	Most Recent Subscript Definition
*c1 > *num1 and *c2 >*num2	N6	i = x1
*c1 < *num1 and *c2>*num2	N6	i =x2

(iv) A total of four test cases is required to map each most recent subscript index definition to its subscript site.
(v) The number of test cases required by extended branch coverage may exceed the number of test cases required for traditional branch coverage. The idea is to choose a minimal number of test cases required to map all recent subscript index definitions to their subscript sites.
(vi) Test cases need to be designed to make the most recent subscript index definition reach its corresponding subscript site, keeping in mind only feasible paths in the control flow graph.
(vii) In the modified version of the control flow graph CFG', existing test cases may be used from the original program's control flow graph CFG, existing test cases may be modified to cover paths with newly added paths having potential vulnerability sites and new test cases may have to be designed if existing tests no longer apply to newly added paths in CFG'.

5. Conclusion

Regression testing algorithms have primarily focused on functional testing, the advent of a tailored approach for addressing buffer overflow vulnerabilities results a significant improvement in security testing methodologies. This proposed work and algorithms not only fills a critical gap in security testing but also exhibits the adaptability of regression testing tools beyond traditional functional domains. Furthermore, this methodology offers practitioners a systematic way to identify buffer overflow vulnerabilities within the unit testing stage of the early software development life cycle. Furthermore, the algorithm's capability to assess existing test cases for reuse, modification requirements, and the need for new test cases represents a novel contribution to the field. Providing developers with targeted feedback on applicable test cases streamlines the testing process, easing concerns about test case significance and considerably reducing testing time and effort. By extending its applicability to areas such as concurrent programs and multithreading, as well as potential expansion to cover other vulnerabilities like cross-site scripting and SQL injection, the proposed regression testing tool showcases its versatility and robustness in enhancing software security.
Traditional branch coverage adequately addresses decision branches within a program, it often falls short in identifying vulnerability sites, particularly concerning buffer overflow vulnerabilities. This inadequacy highlights the essential for tailored methodologies in security testing. Extended branch coverage emerges as a novel solution, surpassing the limitations of traditional methods by not only encompassing all decision branches but also explicitly targeting potential vulnerability sites.
The proposed extended branch coverage as a tailored approach for security testing signifies a substantial step forward in mitigating buffer overflow vulnerabilities and strengthening penetration testing.

Author contributions

PVR Murthy, T P Pushphavathi: Conceptualization, Methodology, Software, **Jhilmil Basu, Shilpa R.G.:** Data curation, Writing-Original draft preparation, Software, Validation **Jhilmil Basu, Shilpa R.G, PVR Murthy, T P Pushphavathi:** Visualization, Investigation, Writing-Reviewing and Editing.

Conflicts of interest

The authors proclaim no conflicts of interest.

References

[1] Xiaowei Li and Yuan Xue. A Survey on Server-Side Approaches to Securing Web Applications. *ACM Computing Surveys (CSUR)*, 46 (4), Article 54, 1-30, 2014
[2] Halfond, W., Choudhary, S. and Orso, A. Improving penetration testing through static and dynamic analysis-*Software Testing, Verification and Reliability*, 21(3), pp.195-214, 2011. http://doi.org/ 10.1002/stvr.450
[3] Benjamin A.K., Carla E.B., Hilmi O., Vijaykumar T.N., Detection and Prevention of Stack Buffer Overflow Attacks, Communications of the *Association of Computing Machinery, ACM*, 48 (11), 2005, pp.50-56.
[4] H. Do, Chapter Three - Recent Advances in Regression Testing Techniques, Editor(s): Atif M. Memon, Advances in Computers, Elsevier, Volume 103, 2016, Pages 53-77, ISSN 0065-2458, ISBN 9780128099414,https://doi.org/10.1016/bs.adcom.2016.04.004.
[5] Emelie Engström, Per Runeson, Mats Skoglund, A systematic review on regression test selection techniques
[6] Information and Software Technology, Volume 52, Issue 1, Pages 14-30, 2010, ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2009.07.001.
[7] Rahmani, Ani & Min, J & Maspupah, Asri. An empirical study of regression testing techniques. Journal of Physics: Conference Series. 2021. 1869. 012080. 10.1088/1742-6596/1869/1/012080.
[8] J.Bhandari P and Singh A 2017 Review of object-oriented coupling based test case selection in model based testing Proc. 2017Int. Conf. Intell. Comput. Control Syst. ICICCS 2017 2018- Janua 1161–5
[9] Banias O Test case selection-prioritization approach based on memoization dynamic programming algorithm Inf. Softw. Technol. 2019,115 119–30
[10] Felderer, Michael & Fourneret, Elizabeta. A systematic classification of security regression testing approaches.

International Journal on Software Tools for Technology Transfer. 2015.10.1007/s10009-015-0365-2.
[11] Yoo, S., Harman, M.: Regression testing minimisation, selection and prioritisation: a survey. Softw. Test. Verif. Reliab. 1(1), 121–141 2010
[12] Tarhini, Abbas, Zahi Ismail, and Nashat Mansour. "Regression Testing Web Applications". International Conference On Advanced Computer Theory And Engineering. New York: IEEE, 2008. 902-906. Print
[13] Allahbaksh Asadullah, Richa Mishra, M. Basavaraju, and Nikita Jain. "A call trace based technique for regression test selection of enterprise web applications (SoRTEA)". Proceedings of the 7th India Software Engineering Conference on ISEC '14.2014. DOI:http://dx.doi.org/10.1145/2590748.2590770
[14] Graves, T.L., Harrold, M.J., Kim, J.M., Porter, A., Rothermel, G.:An empirical study of regression test selection techniques. ACM Trans. Softw. Eng. Methodol. 10, 184–208 2001
[15] Sunidhi Puri, Abhishek Singhal, Abhay Bansal. "Study and Analysis of Regression Test Case Selection Techniques". International Journal of Computer Applications. 101, 3 September 2014, 45-50. DOI=10.5120/17671-8504
[16] Qurat Farooq. "Model-Based Regression Testing". Emerging Technologies for the Evolution and Maintenance of Software Models. 2012. 10.4018/978-1-61350-438-3
[17] Shahid, Dr Muhammad. "Code Coverage Information to Support Regression Testing". The International Conference on Informatics and Applications (ICIA.2012). 233-239
[18] Tao Y., Lingmin Z., Linzhang W., Xuandong L., An Empirical Study on Detecting and Fixing Buffer Overflow Bugs, IEEE International Conference on Software Testing, Verification and Validation (ICST), 2016. Chicago, IL, pp.91-101
[19] Paul E.B., Irena B. Defeating Buffer Overflow: A Trivial but Dangerous Bug, IEEE IT Professional, 2016.Vol.18, Issue 6, pp.58-61
[20] Khan, S.U.R., et al., A Systematic Review on Test Suite Reduction: Approaches, Experiment's Quality Evaluation, and Guidelines. IEEE Access, 2018. 10
[21] Bokil, P., P. Krishnan, and R. Venkatesh, Achieving Effective Test Suites for Reactive Systems using Specification Mining and Test Suite Reduction Techniques. ACM SIGSOFT Software Engineering Notes, 2015. 40(1): p.1-8
[22] Beizer, B. "Software Testing Techniques", Itp-Media, 2nd edition, 1990
[23] Shahriar, H. and Zulkernine, M. "Mitigating Program Security Vulnerabilities: Approaches and Challenges", Journal ACM Computing Surveys. 2012