International Journal of Advanced Technology in Engineering and Science
Vol. No.5, Issue No. 06, June 2017
www.ijates.com
ijates
ISSN 2348 - 7550

An Integrated Algorithm in Component Security Testing

Sakshi[1], Sukhdip Singh[2]

[1,2]*Computer Science and Engineering Deptt., DCRUST Murthal, Sonipat, Haryana (India)*

ABSTRACT

Component testing provides easy identification of bugs, threats, defects, vulnerabilities etc. and if component testing is performed beforehand it will be a very big help. But since the component based system are made up of third party components and these third party components are the collection of the specifications or the black box components, therefore the source code is not visible to the developer or the tester. Testing becomes difficult as a result of it. It is at this point that the mutation testing approach and various fault based injection approach comes into picture. When component testing is performed with the help of mutation testing and fault based injection techniques it generates a number of records, details called as monitor logs. These monitor logs are checked for the existence of any insecure or vulnerable component with the help of string searching algorithm. The existing string searching algorithm has results which can be improved hence a new integrated algorithm known as proposed algorithm is proposed.

Keywords- component testing; vulnerabilities; monitor logs; string searching.

I. INTRODUCTION

A component can be anything from an entity, module, interface etc. this term is used in various contexts for various objects. But in an understandable manner we will say that a component is a unit or a collection of various functions, objects etc which interact with each other to provide the functionality of the system. The component itself is a standalone entity capable of execution**.** Also it does not require any sort of compilation before its use with the other present components. The interface acts as the service provider for the components as with the help of these interfaces, the services of the components are made available.

- **Characteristics of Components**

a) Standardised: The component should be similar to the component model which may specify the interfaces, deployment etc.

b) Independent: Component should exists as a standalone entity, as a not dependent entity.

c) Composable: All the non internal interactions should occur with the help of well defined interfaces which are not private. The non internal access about attributes, methods should be possible.

d) Deployable: Components do not have to compiled before they can be deployed.

e) Documented: Components should be well documented by specifying their identities, objectives, interfaces etc.

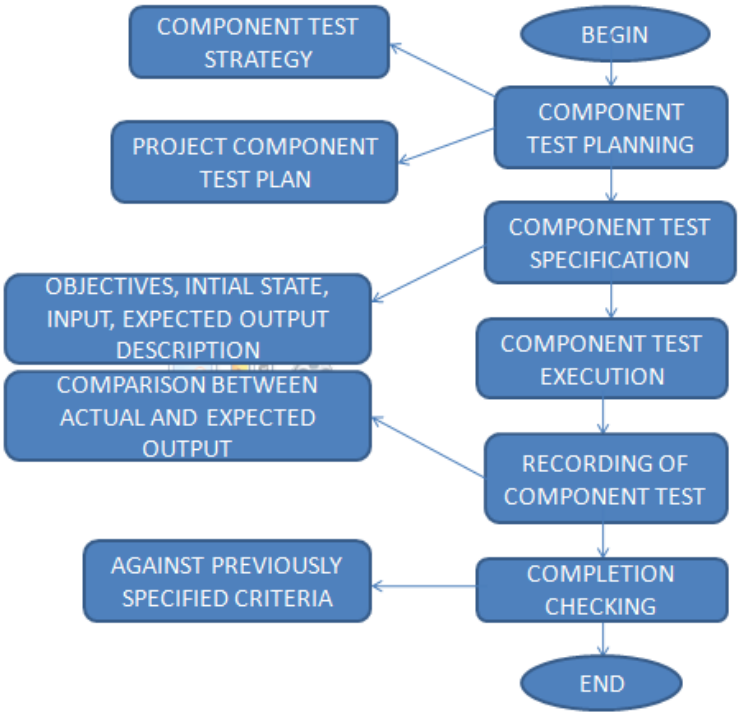


Fig1. Workflow of Component testing

In Component testing firstly planning is done which comprises of testing strategy for the entire process and test plan for the entire process. In specification phase the objectives, initial state of the components, expected output are being specified.

After execution of the cases, they compare the actual and expected output to mark the completion. The completion checking is done by matching against the previous criteria is satisfied, process is ended, repeated otherwise.

Component Security testing has become an interesting research topic and a growing field in the area of software testing.[1] Because of the reusability functionality, plug and play services the development and maintenance costs are reduced. The technology has improved efficiency of component but the security problem lies intact.[2,3]Component security vulnerability means that there are flaws, weaknesses in the software which pose to harm the software.[4] Some currently used component test methods still employ traditional methods. But for components fault injection based on environment and interface test method is also proposed.[5,6] Component exception mining monitor log includes looking for abnormal information. The monitor log and abnormal information are themselves strings so they could be mould to search and mine unsafe strings (pattern strings) from extra large string (main string).[7] The monitor log is the main string, and the keyword, which is also called the "insecure execution sequence", is the pattern string. A string matching technique is an operation of finding a pattern string in the main string. If matching succeeds, it will return a successful match location, which means the component has security vulnerability. If not, it means that the component has no security vulnerability[8].

Vulnerability testing checks for the reliability of the software and thus lower the chances of security risks in the software system.[9] Searching for the pattern occurrences in a database etc is a very fundamental solution seeking problem.[10] In network intrusion detection system, extraordinary high performance is demanded by

string matching to match a huge amount of network or traffic against a predefined database of malicious, harmful patterns.[11]The existing log matching methods, based on BF algorithm are straightforward. However, BF algorithm is not suitable for components. This is because the efficiency of BF algorithm is low, and the components will produce a large number of monitoring log sets during the running process. Despite the fact that Knuth- Morris-Pratt (KMP) algorithm, BM algorithm, and Sunday algorithm have improved BF algorithm, they still do not apply to a large set of monitoring logs. Hence, we propose a new string matching algorithm that is suitable for large main strings. The algorithm does not waste time in searching for the string at unnecessary and undesired places. Hence a lot of time is saved which is seen in terms of its execution efficiency when compared with the existing ml Sunday string searching algorithm.

II. RESEARCH METHODOLOGY

The overall methodology followed in the research conducted can be broken down into following steps:

[1.] Monitor log generation: When component testing is performed with the help of mutation testing or fault based injection testing a huge amount of data records are created as a result of it. These data records are simply known by various names likes logs, records, monitor logs etc.

[2.] Data preprocessing: On the generated monitor logs various data preprocessing techniques are applied to obtain the processed data known as database.

[3.] Database creation: When the preprocessing techniques are applied on the monitor log, the database is obtained as result.

[4.] Pattern/ Pattern string matching: The database so generated comprises of a huge no strings and that collection of the entire strings is matched against a particular string known as pattern or pattern string or target string.

[5.] String searching and matching algorithm: The matching of the pattern against the database is done with the help of various string searching and matching algorithms.

[6.] Result: Here the resut is binary that is, either the pattern is matched against the database or not. Hence we will say matching is successful if the pattern is matched against the database, unsuccessful otherwise.

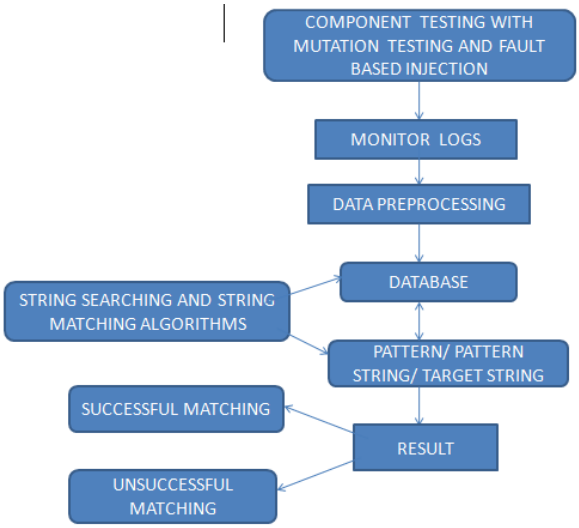


Fig2. Research Methodology

After performing fault based injection testing, mutation testing etc the logs known as monitor logs is generated which are preprocessed to form database. The database so generated or created is known by other names such as text or monitor log or dataset as will be used interchangeably throughout the research paper. The database so created is searched or looked out for the existence of a particular string represented by pattern or target string. The pattern or target string is searched in the database with the help of various string searching and string matching algorithms. If after applying the algorithm, the pattern string matches with the database it means the database contains vulnerable strings or patterns, else no vulnerable strings or patterns exists in the database and the database or monitor log is free from the vulnerable strings or vulnerabilities. If a match between the pattern and the database occurs we say that the result is successful matching, unsuccessful matching otherwise.

III. RESEARCH BACKGROUND

Component Based Software Engineering (CBSE) has emerged and become a major area and a renowned field of software engineering. This is due to the fact that it provides reuse facility along with the plug and play characteristic which not only aids in reducing the development and maintenance cost but also enhances the efficiency of the software. However there a numerous number of security problems that are encountered by developers as well the users of the software which demands a serious need to be resolved. The approach to test the component security is through the mutation test case generation and execution of the test sequences of components and the interface methods of components. Because of the above followed procedure the above used method produces a large no of records or logs also known as monitor logs. The monitor logs are then mined, classified and organized implicitly and explicitly for any security vulnerability or any security exception. At this particular stage the pattern and string matching algorithms came into picture. The monitor log also known as dataset or database acts as the main string and the pattern or the string which needs to be matched is the keyword. In this method or procedure we will conclude or we can say that there is component or string vulnerability if the matching of the keyword in the monitor log or the database is successful and a matching location is returned else the component is not vulnerable and no security exception exists in the monitor logs. In this manner the insecure exceptions or strings, pattern are recognized and captured. A number of pattern and string matching algorithms exists for this purpose eg Brute Force algorithm, KMP algorithm(Knuth Morris Pratt) etc. but they fails in some manner. In succession to all these existing algorithm and improved string searching algorithm is also proposed but which provides an efficiency of 11.5% and is advancement over ml Sunday string algorithm. But still there was a scope of improvement and to improve the efficiency there was a need of yet another algorithm and to bridge that gap an integrated algorithm is proposed.

IV. MOTIVATION

As we know that in black box testing the source code is not visible to us. Likewise in component testing we do not have access to the source code because it is also a black box testing. Even though component testing is effective in uncovering the flaws, defects, bugs , vulnerabilities in the software but to aid the testing we need some algorithms or testing techniques and we introduce the usage of string searching algorithm with component testing to identify whether there exists any bug or vulnerability in the source code or not. Thus

using string searching and matching algorithm for testing the component security is a very nice and effective approach.

V. VARIOUS STRING SEARCHING ALGORITHMS

- **Brute Force algorithm**

This is the early most algorithm to be proposed and hence the efficiency and the performance of this algorithm is the worst as compared to the other and existing algorithms. The procedure of this algorithm is described as the string to be found also called the pattern is aligned with the main string also called database and it is checked one by one position. If the math at the identical location is successful then the procedure is repeated for the entire pattern and main string. If the values of the string at identical locations are different then pattern string is moved a place back and then aligned with the main string and then matching is done. The entire process is repeated time and again until a math is found ie the matching is successful or the end of the string is reached.

The time complexity of brute force algorithm is worst among all the algorithms and is denoted as O(mn).

- **KMP algorithm**

The KMP algorithm was proposed by Knuth Morris Pratt. The working of this algorithm can be understood as follows. As in the case with BF algorithm where if at the identical positions the strings do not match the main string or the database is moved a step backwards, in kmp algorithm the main string is not moved backwards rather it is shifted to as right as possible and also the movement to right can be made to any number of positions.

The kmp algorithm is better and efficient as compared to bf algorithm but the main drawback of this algorithm is that it is hard to understand and grasp.

The time complexity of kmp algorithm is O(m+n).

- **BM algorithm**

Boyer and Moore algorithm is a very nice algorithm and a very efficient string searching algorithm. The searching capability of this algorithm is as high as nearly 5 times to that of kmp algorithm.

To improve the efficiency of searching the particular string it uses 3 given approaches.

1) Right to left scanning approach

2) Bad character approach

3) Good suffix approach

- **Sunday string searching algorithm**

A pattern string can be matched to the monitor log or the database or main string from left to right or right to left. Else the step taken in this case is similar to that of BM algorithm. But the distance of moving backwards is calculated.

- **ML Sunday string searching algorithm**

The Target string or pattern is matched with monitor log from back to start in proper order. If matching is successful, the matched location is returned, else if the target string is not equal to the monitor log or the database the matching fails or the matching is unsuccessful. The algorithm also looks for the strings or the characters that occur for the very first time and those characters are then matched with the monitor log. The time complexity of this algorithm is O(mn).

VI. AN INTEGRATED ALGORITHM

Name of candidate set	Character Candidate sets
Long_can	a to z, A to Z, 0 to 9 and special symbols !,@,#,%,^,& etc.
Medium_can	A to Z, 0 to 9
Short_can	0 to 9 and special symbols !,@,$,^ etc.

Table 1: Character candidate sets[12]

Proposed Algorithm:

1. n=1;
2. m=1;
3. ML=ds(n);
4. textLen=length(text);
5. pattern=TS(n,m);
6. psLen=length(pattern)
7. TS=pattern
8. n=length(ML);
9. m=length(TS);
10. pos=1;
11. S=0;
12. While(1)
13. Flag=1;
14. For i=1 to m
15. If(TS(i)=ML(pos+i-1))
16. Flag=0;
17. End
18. End
19. If(flag==1)
20. res=pos-1;
21. elseif(flag==0)
22. i=0;
23. for j= pos+1 to n-m+1
24. if(ML(j+m-1)==TS(m)))
25. i=j;
26. end

27. end
28. if(i==0)
29. else
30. pos=i;
31. end
32. end
33. end

Terminologies with their meanings:

n	Count of monitor log dataset
m	Count of unsafe component sequence or vulnerable strings or pattern
j	Subscript of character in ML(0<=j<n)
i	Subscript of character in TS(0<=i<m)
move	Places by which TS is moved back when next match is executed(0 is initial value)
pos	Subscript of first character in ML when current match is executed(pos=pos + move)
next	Subscript of $ML_{pos+m-1}$ next character when current match is executed(next = pos + m)

Table 2: Terminologies along with their meanings[13]

The proposed algorithm checks the vulnerability of the component by selecting a string of let's say 50 characters. Then the algorithm match the first character and the last character of the target string against the database created from the combinations of Long-can, Medium-can, Short-can as specified in the table. The matching process is as follows:

The algorithm matches the first and last character of the pattern with the database if it is matched against the database then execution time of algorithm along with the location at which the pattern is matched against the database is returned as an output.

If the first and last character of the pattern is not matched against the database the pattern is shifted to the distance equal to the length of pattern.

The main advantage of this algorithm is that it does not waste time in checking the unnecessary string. If the first and last character of the specific length pattern or the target string is not matched against the database then it is self understood that the characters present in between these locations will also not be matched. Hence instead of wasting time on the unnecessary and unwanted unmatched strings, this algorithm shifts the string to the distance equal to the length of the pattern.

VII. RESULTS AND ANALYSIS

To verify the effectiveness of proposed string- searching algorithm a number of tests were carried out on dataset that is monitor logs and the pattern string. All the tests can be performed on the machine having following specifications:

- **Hardware Requirements:**

f) Processor: Pentium IV or Higher
g) RAM:1GB or higher
h) Hard Disk: 10 GB or Higher.

- **Software Requirements:**

a) Platform: Windows 7 Professional

b) MATLAB 2012 and above

c) Microsoft office(Excel and Word)

The results of the research carried out are given and explained below.

Three candidate set or the database of various sizes are genearated. The monitor log contains the characters which are a part of the generated candidate set. The candidate sets are named long-can, med- can, short- can.

The names candidate sets, database, dataset, monitor log are used interchangeably.

The long-can database comprises of small case alphabets, capital case alphabets, digits and special symbols.

The med-can database comprises of capital case alphabets and digits.

The short-can comprises of digits and special symbols.

The character count of the dataset or database is denoted by n where value of n lies between 20, 30 and 50.

The results of the proposed algorithm are compared with the existing ML Sunday string searching algorithm and the comparison is done on the basis of execution time of different length character string and pattern.

For all the reasearch analysis being carried out on the strings and database the monitor log is the main string and method sequence set is the pattern string.[5]

- **Short-Can Analysis**

In the overall analysis 'n' represents the length of database strings and its value is 2000, 5000, 10000, and 20000.

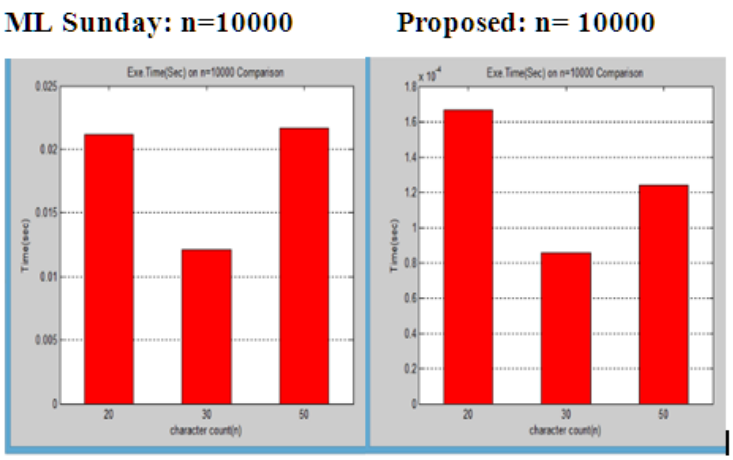


Fig3. n=10000 for short-can candidate set

In this case the monitor log datasets consists of the random characters from short candidate set called as short-can.

The character counts are 20, 30 and 50.

The average matching time of the ML Sunday algorithm and the proposed algorithm are shown with the help of graph. From the analysis of graph we can say that the execution time of proposed algorithm is smaller than ml Sunday algorithm and in for some strings it is nearly equivalent to ml Sunday algorithm. But the overall efficiency for finding out the match of pattern string is higher than ml Sunday string searching algorithm.

ML Sunday: n= 20000 Proposed: n = 20000

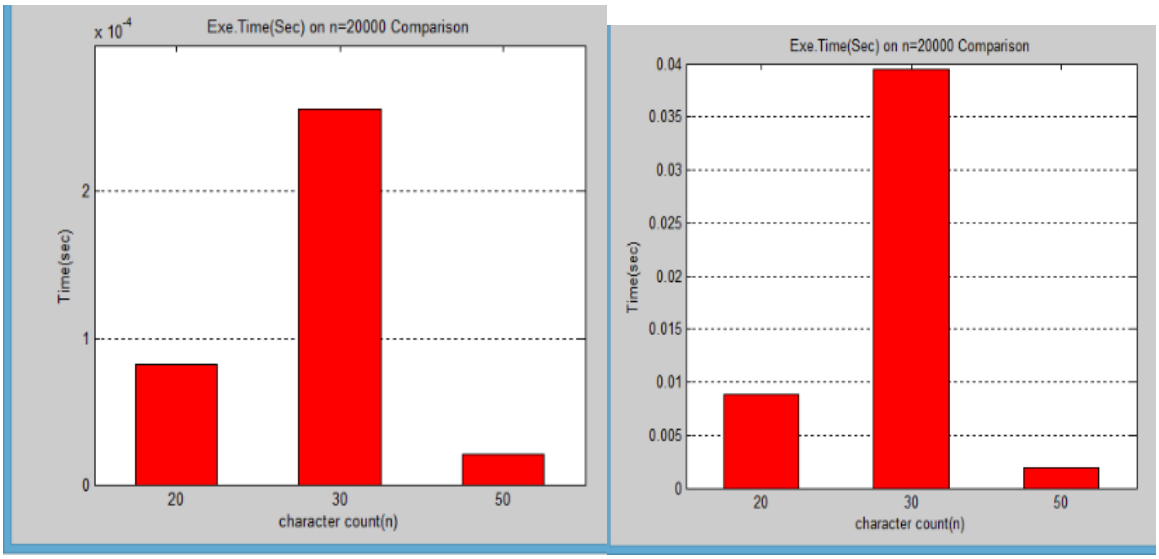


Fig4. n=20000 for short-can candidate set

ML Sunday: n= 5000 Proposed: n= 5000

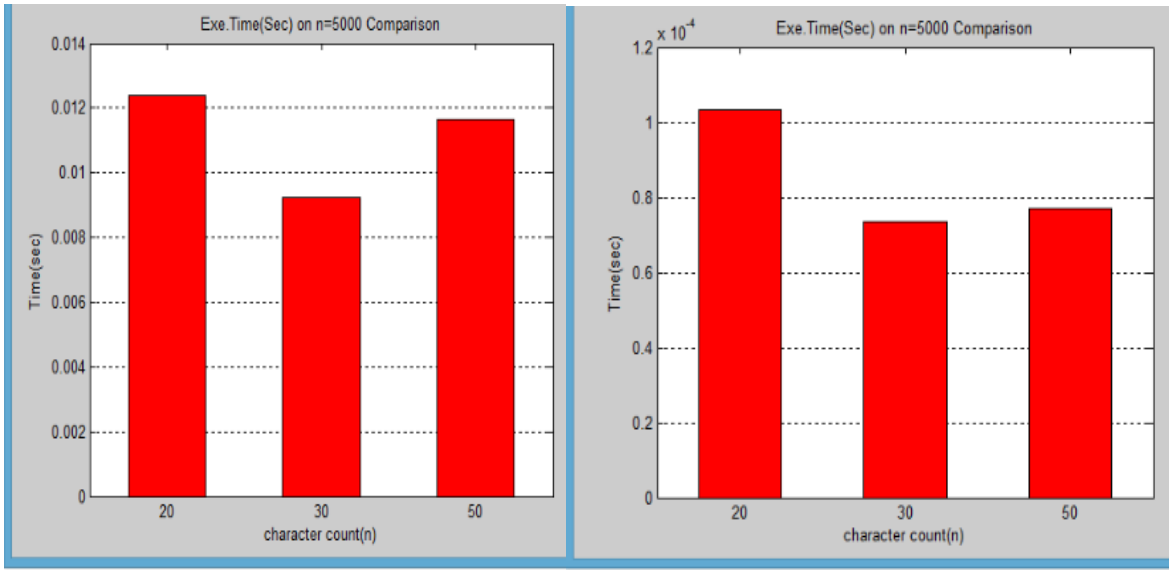


Fig5. n=5000 for short-can candidate set

ML Sunday: n=2000 Proposed: n=2000

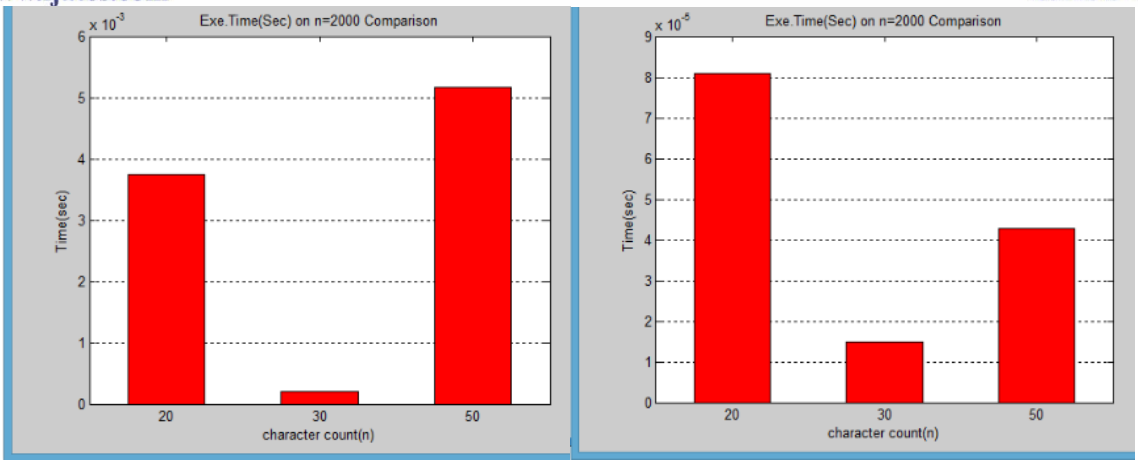


Fig6. n=2000 for short-can candidate set

```
exeTime =

    0.0110    0.0002    0.0057
    0.0142    0.0103    0.0130
    0.0220    0.0127    0.0228
    0.0103    0.0441    0.0022

foundAt =

        1298          70        1655
        4651        3373        3750
        7416        3910        6522
        3420       14099         635

fx >>
```

ML Sunday

Fig7. Execution time with matched location for short-can candidate set

```
step =

     4

exeTime =

    0.0051    0.0000    0.0001
    0.0001    0.0001    0.0001
    0.0002    0.0001    0.0002
    0.0001    0.0004    0.0000

foundAt =

        1298          70        1655
        4651        3373        3750
        7416        3910        6522
        3420       14099         635

fx >>
```

Proposed Algo

Fig8. Execution time with matched location for short-can candidate set

The above presented two figures represents the execution time for both ml Sunday algorithm and proposed algorithm and the location of the matched string or the vulnerable string.

From the figure it can be noted that proposed algorithm provides smaller execution time than ml Sunday string searching algorithm for the same set of strings.

The figure describes the execution time for running the testing on short-can and if a matching occurs, then the matching location is also specified with the help of both the algorithms.

But the main differentiating factor is execution time for both the algorithms. When provided with same dataset from the short-can and checked for the existence of same string for both the algorithms, it is noted that the execution time for proposed algorithm is smaller than ml Sunday algorithm. Hence better efficiency is achieved.

Case: Med Can

The monitor log datasets strings are made up of characters from Medium-can. The character count of method sequence sets are 20, 30 and 50.

The average matching time of ml Sunday string searching algorithm and proposed algorithm are given and explained below. We can analyse with the help of figures that in some cases the execution of time of ml Sunday algorithm is less than proposed algorithm but the average execution time of proposed algorithm is lesser than the ml Sunday algorithm.

ML Sunday: n=20000 **Proposed: n=20000**

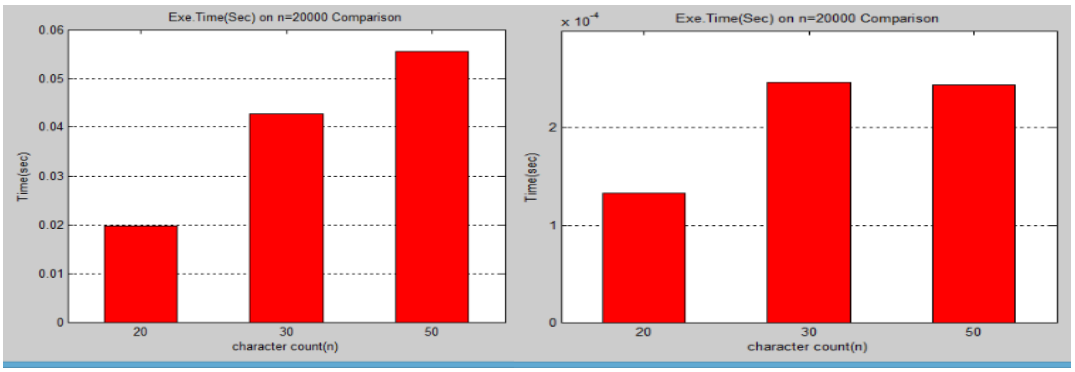


Fig9. n=20000 for med-can candidate set

ML Sunday: n=10000 **Proposed: n=10000**

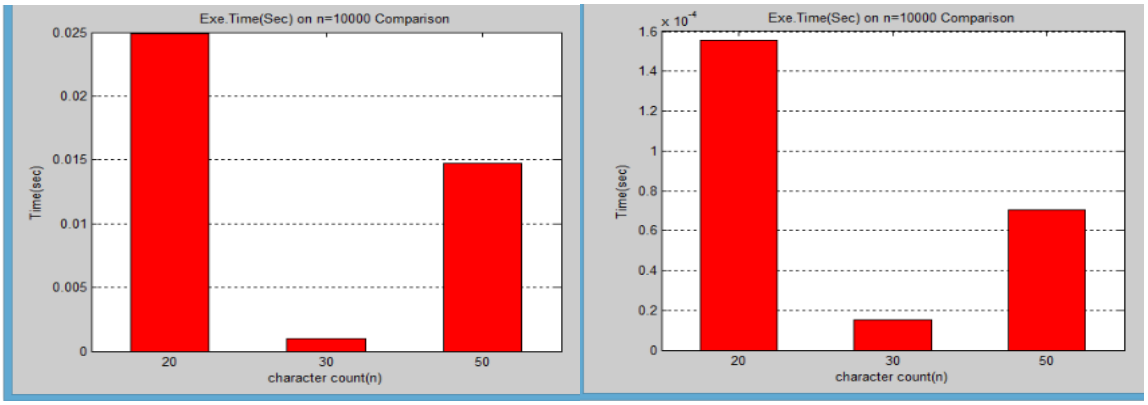


Fig10. n=10000 for med-can candidate set

ML Sunday: n=5000 **Proposed :n=5000**

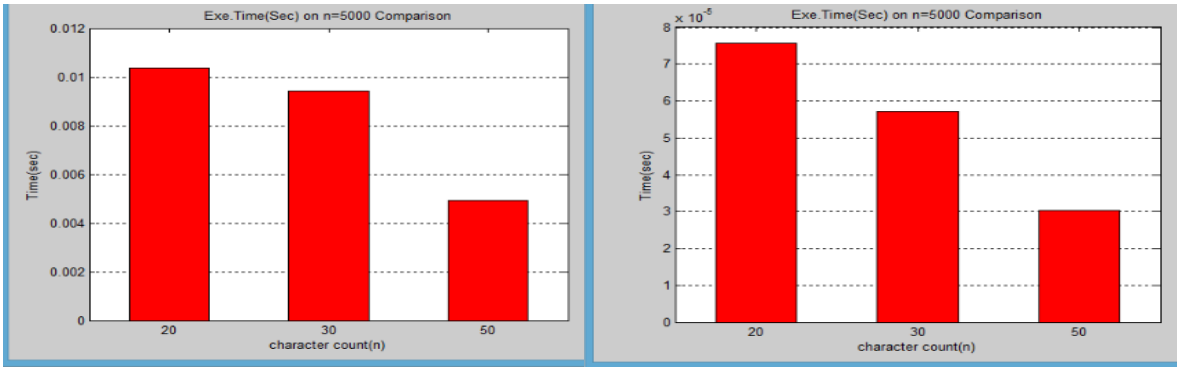


Fig11. n=5000 for med-can candidate set

ML Sunday: n=2000 **Proposed: n=2000**

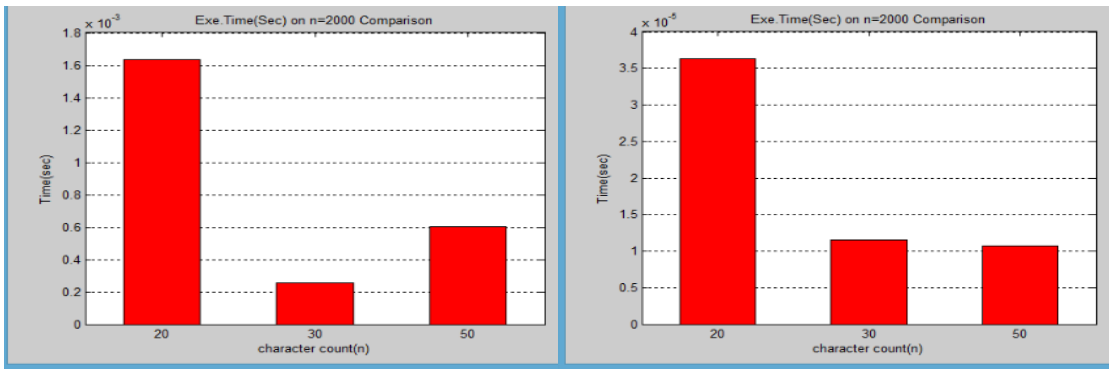


Fig12. n=2000 for med-can candidate set

Execution Time: Med Can

ML Sunday **Proposed Algo**

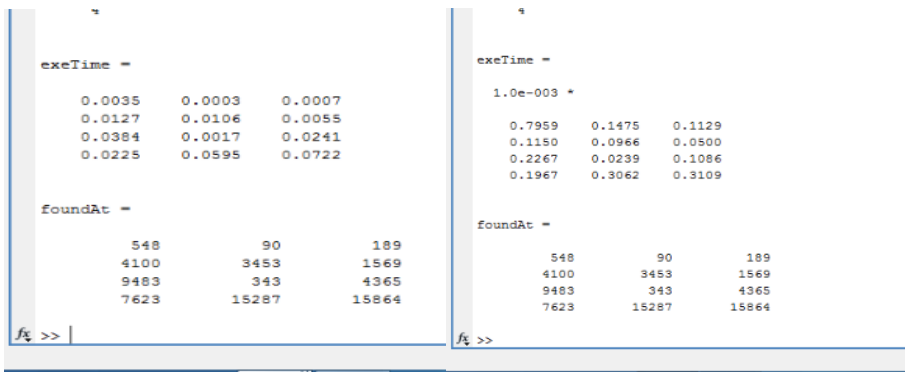
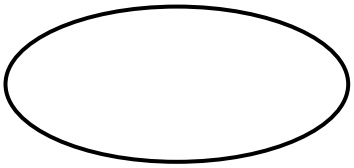


Fig13. Execution time with matched position for med-can candidate set

For the same set of string from the med-can and for the same pattern string is noted that the average execution time for matching is significantly less in case of proposed algorithm compared to that of ml Sunday algorithm.



Case: Long Can

The strings in the monitor log dataset consists of random characters from long-can. The character counts are 20, 30, 50. The monitor log is the main string to which the pattern string is matched. The average execution time of both the algorithms and also the matching location of vulnerable strings are given and shown with the help of graphs and figure. From the figures it is concluded that the efficiency of proposed algorithm is significantly higher than ml Sunday algorithm

ML Sunday: n=20000 **Proposed: n=20000**

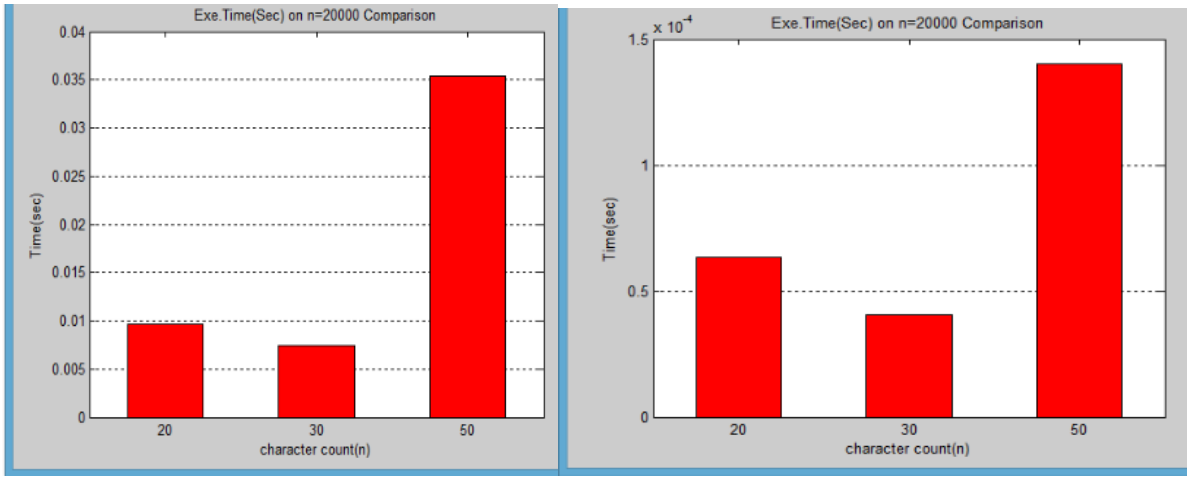


Fig14. n=20000 for long-can candidate set

ML Sunday: n=10000 **Proposed n=10000**

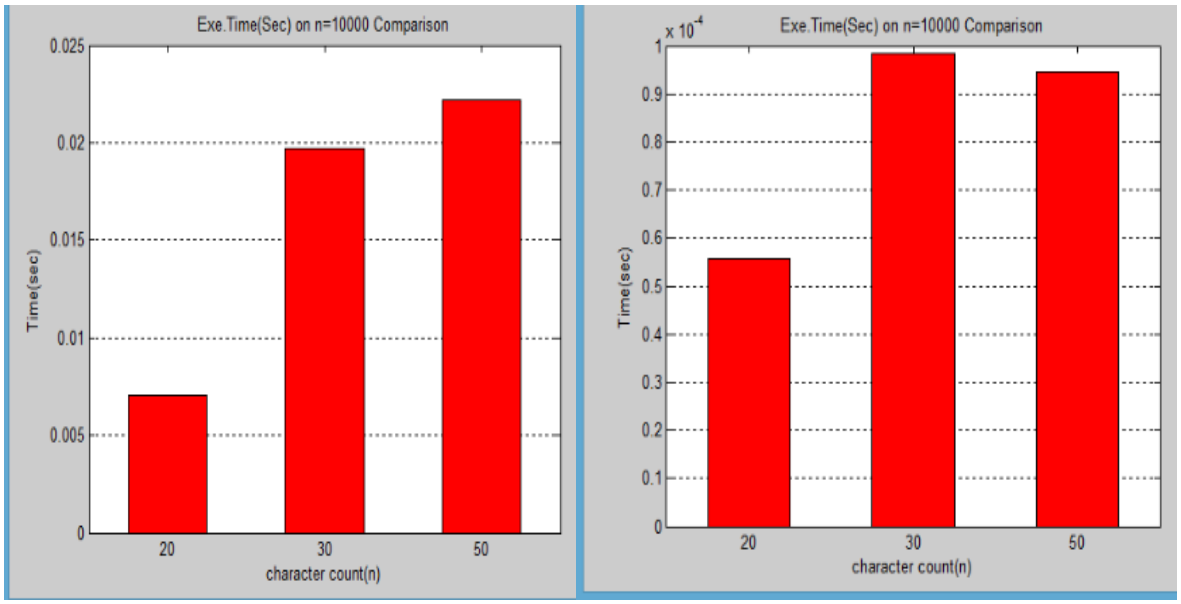


Fig15. n=10000 for long-can candidate set

ML Sunday: n=5000 **Proposed n=5000**

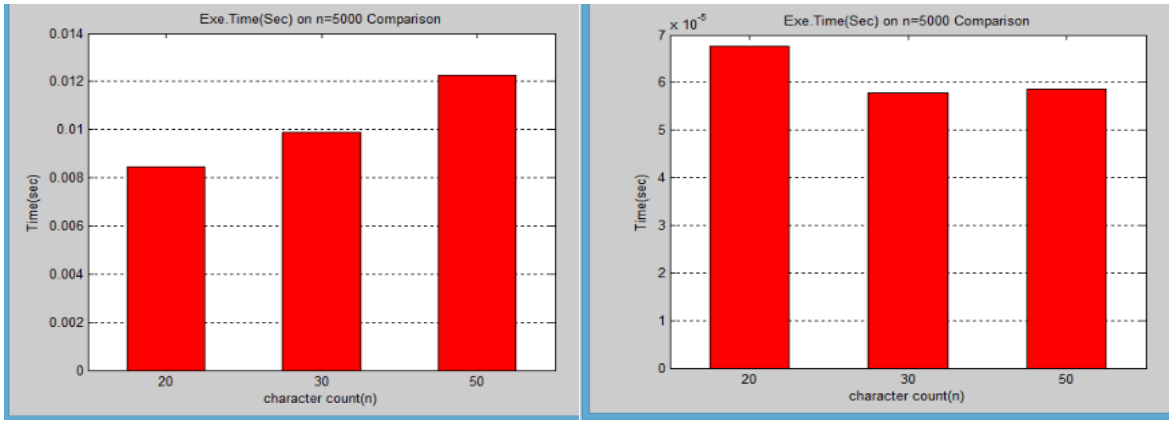


Fig16. n=5000 for long-can candidate set

ML Sunday: n=2000 **Proposed: n=2000**

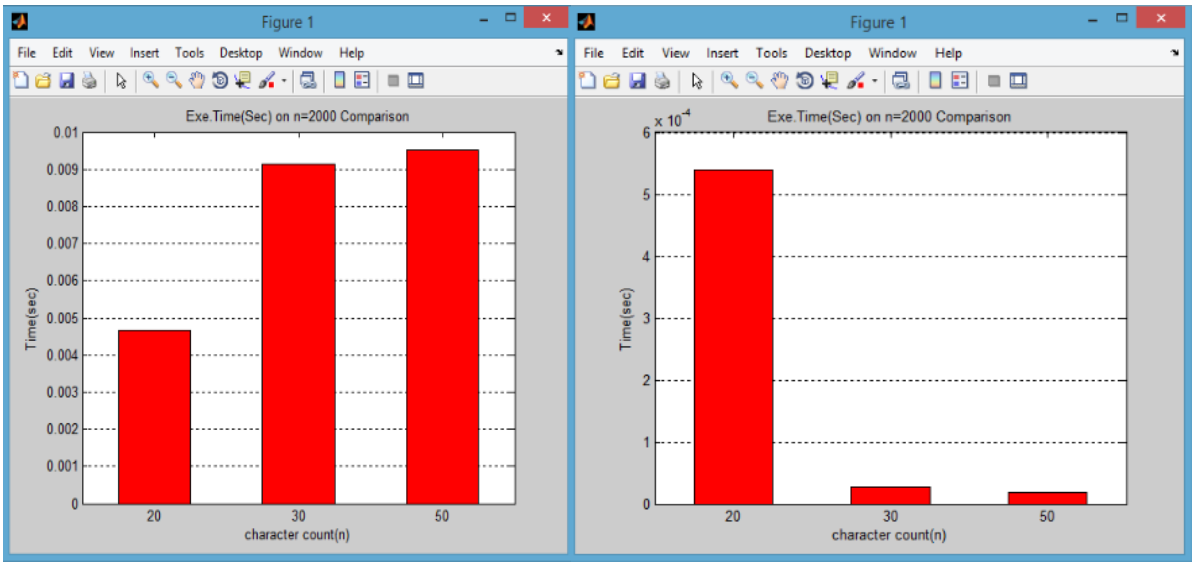


Fig17. n=2000 for long-can candidate set.

Execution Time: ML Sunday

```
exeTime =

    0.0024    0.0049    0.0050
    0.0101    0.0110    0.0139
    0.0087    0.0220    0.0257
    0.0117    0.0107    0.0422


foundAt =

         370         964         868
        3218        3525        3735
        2754        6776        6518
        3248        2376        9942

fx >>
```

Fig18. execution time and matched location for long-can candidate set

Execution Time: Proposed algo

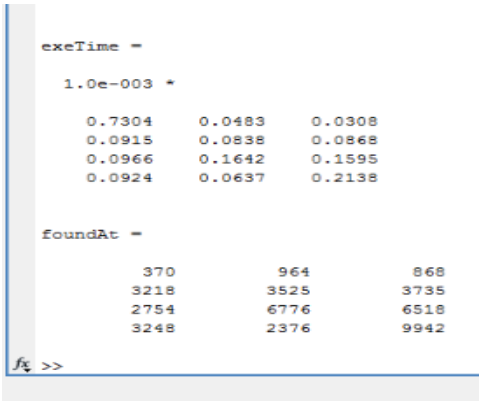


Fig19. execution time and matched location for long-can candidate set

VIII. CONCLUSION

Nearly hundreds of security and vulnerability flaws are unveiled by hackers, developers, users etc.[14] The focus is on the functionality of the component while performing component testing.[15] Component security testing through string searching algorithm has become nice and effective approach to provide sufficient degree of security. Hence many other string searching algorithms can be proposed in future.

REFERENCES

[1] Chen, J., Lu, Y., & Wang, H. (2012). Component security testing approach based on extended chemical abstract machine. *International Journal of Software Engineering and Knowledge Engineering*, *22*(01), 59-83.

[2] Briand, L. C., Labiche, Y., & Sówka, M. M. (2006, May). Automated, contract-based user testing of commercial-off-the-shelf components. In *Proceedings of the 28th international conference on Software engineering* (pp. 92-101). ACM.

[3] Chen, J., Wang, H., Towey, D., Mao, C., Huang, R., & Zhan, Y. (2014). Worst-input mutation approach to web services vulnerability testing based on SOAP messages. Tsinghua Science and Technology, 19(5), 429-441.

[4] Chengying, M., & Yansheng, L. (2006). Research Progress in Testing Techniques of Component-Based Software [J]. Journal of Computer Research and Development, 8, 011.

[5] Chen, J., Lu, Y., & Xie, X. (2007, December). Testing approach of component security based on fault injection. In Computational Intelligence and Security, 2007 International Conference on (pp. 763-767). IEEE.

[6] Jinfu, C., Yansheng, L., & Xiaodong, X. (2009). A fault injection model of component security testing. Journal of Computer Research and Development, 7, 012.

[7] Chen, J., Zhu, L., Xie, Z., Omari, M., Ackah-Arthur, H., Cai, S., & Huang, R. (2016). An effective long string searching algorithm towards component security testing. China Communications, 13(11), 153-169.

[8] Chen, J., Cai, S., Zhu, L., Guo, Y., Huang, R., Zhao, X., & Sheng, Y. (2016). An improved string-searching algorithm and its application in component security testing. Tsinghua Science and Technology, 21(3), 281-294.

[9] Chen, J., Wang, H., Towey, D., Mao, C., Huang, R., & Zhan, Y. (2014). Worst-input mutation approach to web services vulnerability testing based on SOAP messages. Tsinghua Science and Technology, 19(5), 429-441.

[10] Faro, S., & Külekci, M. O. (2013, January). Fast packed string matching for short patterns. In 2013 Proceedings of the Fifteenth Workshop on Algorithm Engineering and Experiments (ALENEX) (pp. 113-121). Society for Industrial and Applied Mathematics

[11] Le, H., & Prasanna, V. K. (2013). A memory-efficient and modular approach for large-scale string pattern matching. IEEE Transactions on Computers, 62(5), 844-857.

[12] Chen, J., Cai, S., Zhu, L., Guo, Y., Huang, R., Zhao, X., & Sheng, Y. (2016). An improved string-searching algorithm and its application in component security testing. Tsinghua Science and Technology, 21(3), 281-294

[13] Chen, J., Cai, S., Zhu, L., Guo, Y., Huang, R., Zhao, X., & Sheng, Y. (2016). An improved string-searching algorithm and its application in component security testing. Tsinghua Science and Technology, 21(3), 281-294

[14] Tung, Y. H., Lo, S. C., Shih, J. F., & Lin, H. F. (2016, October). An integrated security testing framework for Secure Software Development Life Cycle. In Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific (pp. 1-4). IEEE.

[15] Chen, J., Lu, Y., & Xie, X. (2009, August). CSTS: A Prototype Tool for Testing COM Component Security. In Hybrid Intelligent Systems, 2009. HIS'09. Ninth International Conference on (Vol. 3, pp. 83-88). IEEE.