# ENHANCING SOFTWARE SECURITY MEASUREMENT WITH MUTATION TESTING

SREE RAM KUMAR T,

*Research Scholar, Madurai Kamaraj University, Madurai, India*

DR. ALAGARSAMY K,

*Associate Professor, Madurai Kamaraj University, Madurai, India.*

This paper presents a fault-injection based quantitative assessment of software security. There has been a great deal of interest in the recent times to quantitatively measure the security of software as software has permeated through a range of applications from entertainment to banking to e-governance. With the rapid increase in attempts to exploit the security weaknesses of software, this measurement of software security has assumed great importance. The method proposed in the paper is based on a technique called "Adaptive Vulnerability Analysis" which has been successfully applied to measure software security. We propose a mutation testing based enhancement to the method, which results in greater accuracy of the measured software security, and the claim is substantiated by empirical results.

*Keywords*: Security Metrics, Mutation Testing, Vulnerability Assessment

## 1. Introduction

Over the years, several researches have attempted to apply methodologies originally developed for software testing and software assessment to perform Software Security Assessment [1,2,3,4]. Especially, researchers in the domain of failure-tolerance and reliable software have found that the problem in Computer Security is a special case of failure tolerance, where software failure is the failure of a system to enforce the security policies defined for the system [5]. This lead Voas et. al. to adapt a technique called "Extended Propagation Analysis", originally used in assessing safety-critical software [6,7] and develop a dynamic software analysis algorithm called "Adaptive Vulnerability Analysis". Next, we present a brief overview of AVA.

## 2. Adaptive Vulnerability Analysis

Voas et. al. have worked on adapting a technique called "Extended Propagation Analysis", originally meant for assessment of safety-critical software [11,6,7] to perform Software Security Assessment [4]. They define a Securty Attack as a "dynamic event that occurs during the execution of a piece of software". According to them, a Vulnerability comprises 2 parts: a potential defect or weakness in an information system together with the knowledge required to exploit the defect. AVA uses what are called "perturbation functions" to inject infections into Program States. In particular, AVA uses a perturbation function called "flipBit" which allows a user to flip any bit from 0 to 1 or vice versa. For the test data set, they use both the normal operational profile Q and the inverse operational profile $\overline{Q}$. Intrusions are specified as predicates representing compromised or undesirable program states.

### *2.1 AVA Algorithm*

Let P denote the program, x denote an input, Q denote the normal usage probability distribution and $\overline{Q}$ the inverse usage probability distribution, l denote a program location in P.

1. For each location l in P that is appropriate, perform steps 2-7.
2. Set count to 0.
3. Randomly select an input x from Q or $\overline{Q}$, and if P halts on x in a fixed period of time, find the corresponding data state created by x immediately after the execution of l. Call this dada state Z.
4. Alter the sampled value of the variable a found in Z, creating $\overline{Z}$ and execute the succeeding code on $\overline{Z}$. The manner by which a is altered will be representative of the threat class that is desired.

5. Of the output from P satisfies PRED, increment count.
6. Repeat steps 3-5 n times, where n is the number of input test cases.
7. Divide count by n yielding $\psi_{alPQ}$, a vulnerability assessment for each line l. This means $1 - \psi_{alPQ}$ is the security assessment that was observed given P and Q.

### *2.2 Metrics from AVA*

MTTI (Mean Time To Intrude) is defined as the average time interval before sn intrusion will occur based on 3 things: input cases in Q and its inverse, the classes of failt injections that are used, and the classes of intrusions defined in PRED. MinTTI (Minimum Time To Intrude) is the shortest predicted period of time before any intrusion defined in PRED will occur. Let $\varepsilon_{lPQ}$ represent the probability that a randomly selected input x will execute location l.

$$MTTI = \left[ \frac{\sum_{i=1}^{M} \left[ \psi_{alPQ} . \varepsilon_{lPQ} \right] \left( \frac{programexecutions}{unitoftime} \right)}{M} \right]^{-1}$$

Where M is the number of locations where AVA was applied. The equation for MinTTI follows:

$$MinTTI = \left[ \max_{l} \left[ \psi_{alPQ} . \varepsilon_{lPQ} \right] \left( \frac{programexecutions}{unitoftime} \right) \right]^{-1}$$

## 3. Mutation Testing

Another area of research that has captured the attention of many researchers in the domain of software testing is Mutation Testing. In Mutation Testing, changes are made to the program to produce mutants, and test cases are generated to differentiate the original program from mutants [8]. Mutation operators are used for producing the mutants. These operators generally involve small syntactic changes such as replacing arithmetic + with arithmetic – or a > by <. The scientific principle behind this is the "Competent Programmer Hypothesis", which states that competent programmers make small mistakes [9].

[8] states many advantages of mutation testing. It allows the user to target a particular class of faults. If a program passes a test suite that kills all mutants it is clear that the non-equivalemt mutants produced were not correct and this eliminates a set of faulty behaviors. It also gives us confidence in the test suite distinguishing between a correct program and a program with one of these types of faults.

The biggest disadvantage of mutation testing is that the number of mutants produced is often massive. [8] states a case where 22 mutation operators were applied resulting in 951 mutants for a program with only 28 executable statements.

### *3.1 Mutation Testing for Security*

There have been attempts to explore the potential of mutation testing in detecting vulnerabilities in a program and [10] is one of them, wherein mutation testing is apllied to reveal buffer overflow and SQL Injection vulnerabilities in software. [10] proposes and validates a number of operators that reveal the aforesaid vulnerabilities.

## 4. Enhancing AVA With Mutation Testing

We attempt to explore the possibility of enhancing the original AVA algorithm with mutation testing. For this purpose, we purpose the following change to step 3 of the AVA algorithm.

3. Select an input x from test cases generated using mutation testing and if P halts on x in a fixed period of time, find the corresponding data state created by x immediately after the execution of l. Call this dada state Z.

### *4.1 Case Study*

To study the improvement in performance obtained by the proposed modification to AVA, we investigated 4 open source programs and applied both the AVA and the enhanced AVA. For the generation of test cases required by step 3 of the modified algorithm, we follow the same procedure outlined in [10]. This test data set kills all the generated mutants obtained by applying the operators proposed in [10] for detection of buffer overflow vulnerabilities.

For each of the 4 programs we have 2 versions – the version with the vulnerability and the version with the vulnerability patched. The four open source programs we selected are essentially the same as selected by [10] to demonstrate the effectiveness of the proposed mutation operators. We tabulate the characteristics of the 4 programs.

**Table 1: Selected Open Source Applications**

| Application Name | Application Type | Source File, Function Name |
|---|---|---|
| Wu-ftpd-2.6.2 | Ftp server | ftpd.c, SockPrintf |
| Edbrowse-2.2.10 | Coomand line Editor Browser | http.c, ftpls |
| Rhapsody IRC-0.28b | Text based IRC Client Console for Unix | main.c, parse_input |
| Cmdftp-0.64 | Command line FTP Client | cmdftp.c, store_line |

### *4.2 Prototype Tool Implementation*

We implemented a tool that accepts a C p program unit. The location of the C program is specified in the appropriate Text box and a text file that contains all the test cases is created and its location specified in the appropriate text box. On Clicking the "Generate Metrics Using AVA" button the results of applying AVA to the program are displayed and on clicking the "Generate Metrics Using Enhanced AVA" button the results of applying our enhanced version of AVA to the program are displayed. The Tool is developed using VB.NET, .NET Framework 2.0 on Windows XP.

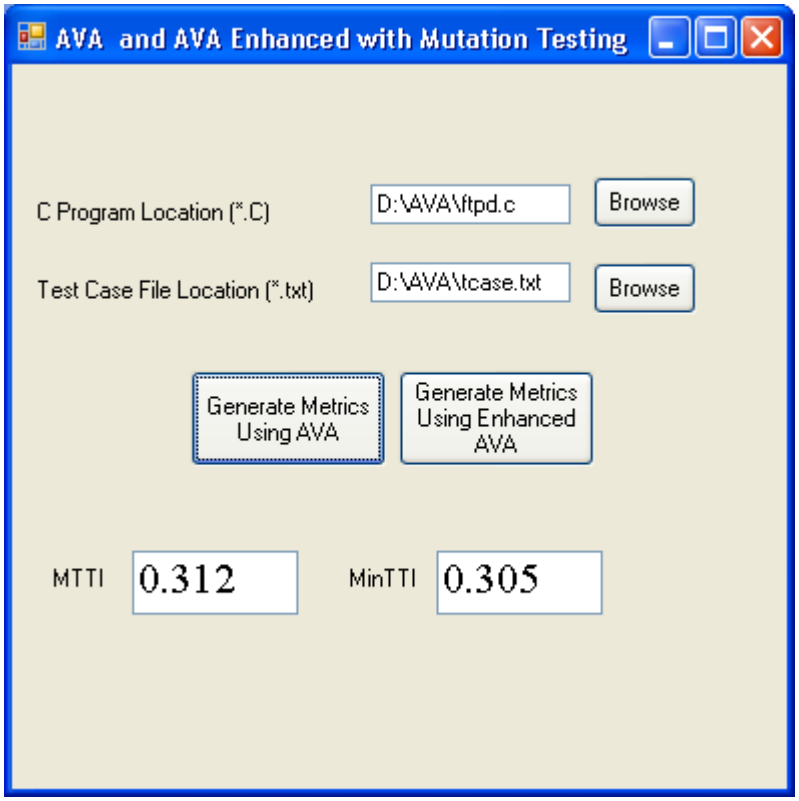


Figure 1: Screen Shot of the Developed Tool

### *4.3 Results and Analysis*

We tabulate the results of applying the AVA and AVA with the proposed enhancement to each of the 4 programs described above using the tool developed for the purpose.

**Table 2: Results For the Unpatched (Vulnerable) Version**

| Application Name | Using AVA | | Using AVA With the Proposed Enhancement | |
|---|---|---|---|---|
| | MTTI | MinTTI | MTTI | MinTTI |
| Wu-ftpd-2.6.2 | 0.312 | 0.305 | 0.293 | 0.281 |
| Edbrowse-2.2.10 | 0.298 | 0.243 | 0.223 | 0.212 |
| Rhapsody IRC-0.28b | 0.372 | 0.363 | 0.321 | 0.313 |
| Cmdftp-0.64 | 0.341 | 0.332 | 0.297 | 0.263 |

**Table 3: Results For the Patched Version**

| Application Name | Using AVA | | Using AVA With the Proposed Enhancement | |
|---|---|---|---|---|
| | MTTI | MinTTI | MTTI | MinTTI |
| Wu-ftpd-2.6.2 | 0.778 | 0.732 | 0.792 | 0.753 |
| Edbrowse-2.2.10 | 0.812 | 0.792 | 0.877 | 0.818 |
| Rhapsody IRC-0.28b | 0.761 | 0.753 | 0.811 | 0.791 |
| Cmdftp-0.64 | 0.897 | 0.803 | 0.902 | 0.810 |

As can be observed from our results, the AVA with the proposed enhancement tends to give more accurate estimates of software security as is evinced by the low values it reports for the unpatched version compared to the standard AVA. A lower value for MTTI (and MinTTI) indicates a compromised security state. On the other hand, for the patched version the difference between the 2 tends to narrow down. Because the vulnerability under consideration has been patched, the attack surface is narrowed down and hence the higher values for MTTI and MinTTI.

## 5. Conclusion And Future Work

Assessment of Software Security has become pivotal in the currant era and attempts to provide quantitative measures of software security should prove very useful. Metrics can be very useful in assessing security provided they are used as relative measures and not absolute ones. AVA provides us with exactly such metrics. Mutation testing has the potential to uncover security vulnerabilities in software. Thus the combination of AVA with mutation testing yields better results as expected.

As part of future work, we plan to apply AVA and the enhanced AVA to programs with other vulnerabilities – known and unknown – and assess the improvement in performance brought about in these cases by the AVA with the proposed enhancement.

## 6. References

[1] T. Aslam, *A Taxonomy of Security Faults in the Unix Operating System*. Master's thesis, Purdue University, West Lafayette, Indiana., 1995.

[2] E. H. Spafford. Extending Mutation Testing to find environmental bugs. *Software Practice and Principle*, 20(2):181-189, February 1990.

[3] B. Beizer. *Software Testing Techniques*. Electrical Engineering/Computer Science and Engineering. Van Nostrand Reinhold, 1983.

[4] J. Voas, A. Ghosh, G. McGraw, F.Charron, K. Miller. *Defining an Adaptive Software Security Metric from a Dynamic Software Failure Tolerance Measure*. Proceedings of the eleventh annual conference on Systems Integrity, Software Safety, Process Security, 1996.

[5] J.Voas. *Testing Software for Characteristics Other than Correctness: Safety, Failure Tolerance, and Security*, Proceedings Of The 13th International Conference On Testing Computer Software, June1996
[6] J. Voas and K. Miller, *Predicting Software's minmum-time-to-hazard and mean-time-to-hazard for rare input events*. In Proc. Of the Int'l Symposium on Software Reliability Eng., pages 229-238, Toulouse, France, October 1995.
[7] J. Voas and K. Miller. *Dynamic testability analysis for assessing fault tolerance*, High Intergrity Systems Journal, 1(2):171-178,1994.
[8] John A. Clark, Haitao Dan, Robert M. Hierons, *Semantic Mutation Testing*, Proceedings of the Third International Conference on Software Testing, Verification and Validation Workshops, April 2010.
[9]R. A. DeMillo, R. J. Lipton, F.G. Sayward, *Hints on test data selection: Help for the practical programmer*, IEEE Computer 11 (4) (1978) 31-41.
[10] Hossain Shahriar, *Mutation Based Testing of Buffer Overflow Vulnerabilities*, Master's thesis, Queen's University, Canada, August 2008. p 34-53.