3. THE APPLICABILITY OF EXISTING METRICS FOR SOFTWARE SECURITY

3.1 Introduction

With the increasing inclination of people to use software systems for most of the purposes, comes a major challenge for software engineers – the engineering of secure software systems. The concept of “Computer Security” is being heavily researched and this perfectly makes sense in a world where e-commerce and e-governance are becoming the norms of the day. Along with their potential for making life easier and smarter for people, these systems also carry with them the danger of insecurity. Because any software system is an outcome of some software engineering process it makes sense to incorporate security considerations during the software engineering processes. This is easier said than done because traditional software engineering approaches are requirements driven and pay very little, if any, attention to security. Tom DeMarco stated, “You can’t control what you can't measure”. This clearly states the importance of metrics in software engineering. Traditional software metrics do not address the issue of security well and now with security becoming an imperative necessity of most software systems, these metrics have to be adapted to take into account the security aspect. This chapter discusses the applicability of some established metrics for the security aspect.

Since quantitative methods have proved so powerful in other sciences, computer science practitioners and theoreticians have worked hard to bring similar approaches to software development. Even though many software metrics are now available, most of the metrics have lacked a sound theoretical basis or a statistically significant experimental validation (Mills, 1988). Despite these problems, it appears that the judicious methodical application of software metrics can aid significantly in
improving software quality and productivity. Engineering of secure software systems seems to be one of the most important challenges confronted by software practitioners today and hence it is worth exploring the possibility of using metrics to aid the software engineers in this regard.

3.2 Existing Metrics And Their Suitability For Measuring Security

A brief overview of some of the available software metrics and their applicability for measuring security is presented below:

3.2.1 Size Metrics

A number of metrics attempt to quantify software “size”. The metric that is most widely used “Lines Of Code” or “LOC”, suffers from the obvious deficiency that its value cannot be measured until after the coding process has been completed.

3.2.1.1 LOC

“Lines of Code” is possibly the most widely used metric for program size. It would seem to be easily and precisely definable; however, there are a number of different definitions for the number of lines of code in a particular program. These differences involve treatment of blank lines and comment lines, non-executable statements, multiple statements per line, and multiples lines per statement, as well as question of how to count reused lines of code. The most common definition of LOC seems to count any line that is not a blank or comment line, regardless of the number of statements per line (Boehm, 1981; Jones, 1986). LOC has been theorized to be useful as a predictor of program complexity, total development effort, and programmer performance.

In the context of Security, the utility of the LOC metric is at best questionable because there seems to be no relationship between the LOC of a program and its security. Whether a program with more LOC is more or less secure than a program
with a fewer LOC is yet to be proved. LOC is also influenced by some other factors like the programming language used as some recent programming languages have the ability to deliver more functionality with fewer LOC.

### 3.2.1.2 Function Points

Albrecht has proposed a measure of software size that can be determined early in the development process. The approach is to compute the total function points (FP) value for the project, based upon the number of external user inputs, inquiries, outputs, and master files. The value of FP is the total of these individual values, with the following weights applied: inputs: 4, outputs: 5, inquiries: 4, and master files: 10. Function points are intended to be a measure of program size, and, thus, effort required for development.

### 3.2.1.3 Bang Metrics

DeMarco defines system Bang as a function metric, indicative of the size of the system. In effect, it measures the total functionality of the software system delivered to the user. Bang can be calculated from certain algorithm and data primitives available from a set of formal specifications for the software. The model provides different formulas and criteria for distinguishing between complex algorithmic versus heavily data oriented systems.

With regard to security, all the size metrics are of little, if any, utility, as the relationship between software size and software security is not yet established. Any attempt to reuse these metrics, for security, must first determine the relationship between size and security, if any exists. Probably, the size metrics can be helpful in predicting the effort to be expended for Security of the software.
3.2.2 Complexity Metrics

Numerous metrics have been proposed for measuring program complexity – probably more than for any other program characteristic (Mills, 1988). As is the case with size metrics, measures of complexity that can be computed early in the software development life cycle will be of greater value in managing the software process.

3.2.2.1 Cyclomatic Complexity – \( v(G) \)

Given any Computer Program, its control flow graph \( G \) can be drawn, wherein each node corresponds to a block of sequential code and each arc corresponds to a branch or decision point in the program. The cyclomatic complexity of such a graph can be computed by a simple formula from graph theory, as \( v(G) = e - n + 2 \), where \( e \) is the number of edges, and \( n \) is the number of nodes in the graph.

3.2.2.2 Extensions to \( v(G) \)

Myers noted that McCabe’s cyclomatic complexity measure \( v(G) \), provides a measure of program complexity but fails to differentiate the complexity of some rather simple cases involving single conditions (as opposed to multiple conditions) in conditional statements. As an improvement to the original formula, Myers suggests extending \( v(G) \) to \( v'(G) = [l:u] \), where \( l \) and \( u \) are lower and upper bounds, respectively, for the complexity. This formula gives more satisfactory results for the cases noted by Myers (Myers, 1977). Stetter proposed that the program flow graph be expanded to include data declarations and data references, thus allowing the graph to depict the program complexity more completely. If \( H \) is the new program flow graph, it will generally contain multiple entry and exit nodes. A function \( f(H) \) can be computed as a measure of the flow complexity of program \( H \). The deficiencies noted by Myers are also eliminated by \( f(H) \) (Stetter, 1984).
3.2.2.3 Knots

The concept of program knots is related to drawing the program control flow graph with a node for every statement or block of sequential statements. A knot is then defined as a necessary crossing of directional lines in the graph. The number of knots in a program has been proposed as a measure of program complexity (Woodward, Hennel, & Hedley, 1979).

3.2.2.4 Information flow

The information flow within a program structure may also be used as a metric for program complexity. Henry and Kafura (Kafura & Henry, 1981) have proposed such a measure. Basically, their method counts the number of information flows entering (fan-in) and exiting (fan-out) each procedure. The procedure’s complexity is then defined as:

\[ C = \frac{\text{procedure} - \text{length}}{\text{fan-in} \times \text{fan-out}}^2 \]

All the complexity metrics have generally been related to programming effort, debugging performance, and maintenance effort. In the context of security, these metrics may also serve as indicators of the strength of security mechanisms needed by the program. But for this, evidence needs to be established that a relationship exists between the complexity of the program and the strength of the security mechanisms needed by it. For example, a more complex program may require more effort for security.

3.2.3 Halstead’s Product Metrics

Most of the product metrics proposed have applied to only one particular aspect of the software product. In contrast, Halstead’s software science proposed a unified set of metrics that apply to several aspects of programs, as well as to the
overall software production effort. Thus, it is the first set of software metrics unified by a common theoretical basis.

3.2.3.1 Program Vocabulary

Halstead theorized that computer programs can be visualized as a sequence of tokens, each token being classified as either an operator or operand. He then defined the vocabulary, N, of the program as:

\[ n = n_1 + n_2 \]

where \( n_1 \) = the number of unique operators in the program and
\( n_2 \) = the number of unique operands in the program.

Thus, \( n \) is the total number of unique tokens from which the program has been constructed (Halstead, 1977).

3.2.3.2 Program Length

Having identified the basic tokens used to construct the program, Halstead then defined the program length, \( N \), as the count of the total number of operators and operands in the program. Specifically,

\[ N = N_1 + N_2 \]

Where \( N_1 \) = the total number of operators in the program, and,
\( N_2 \) = the total number of operands in the program.

Thus, \( N \) is clearly a measure of the program size, and one that is directly derivable from the program itself. In practice, however, the distinction between operators and operands may be non-trivial, thus complicating the counting process (Halstead, 1977). Halstead theorized that an estimated value for \( N' \), designated, can be calculated from the values of \( n_1 \) and \( n_2 \) by using the following formula:

\[ N' = n_1 \log_2 n_1 + n_2 \log_2 n_2. \]
Thus, N is a primitive metric, directly observable from the finished program, while N’ is a computed metric, which can be calculated from the actual or estimated values of n1 and n2 before the final code is actually produced. Some studies have attempted to relate N and N’ to other software properties such as complexity and defect rates. Similar studies need to be done to explore any possibility of relationship between Halstead’s metrics and software security.

3.2.4 Quality Metrics

One can generate long lists of quality characteristics for software – correctness, efficiency, portability, maintainability, reliability and perhaps even security. Early examples of work on quality metrics are discussed by Boehm (Boehm, Brown, & Lipow, 1976; McCall, 1977). Unfortunately, the characteristics often overlap and conflict with one another; for example, increased portability may result in lowered efficiency. Although a good deal of work has been done in this area, it exhibits less commonality of direction or definition that other areas of metric research, such as software size or complexity.

3.2.4.1 Defect Metrics

The number of defects in the software product should be readily derivable from the product itself; thus, it qualifies as a product metric. However, since there is no effective procedure for counting the defects in the program, the following alternative measures have been proposed:

- Number of design changes
- Number of errors detected by code inspections
- Number of errors detected in program inspections
- Number of code changes required
The number of defects observed in a software product provides, in itself, a metric of software quality. These metrics can be extended to include “security defects” as well. For example the number of security design changes required, number of security related errors detected by code inspections and the number of code changes necessitated by security related aspects may be taken into account. These are likely to provide a good measure of the security mechanisms built into the program. But they have a major disadvantage: these metrics are available only lately in the software development life cycle.

3.2.4.2 Reliability Metrics

It would be useful to know the probability of software failure, or the rate at which software errors will occur. Again, although this information is inherent in the software product, it can only be estimated from the data collected on software defects as a function of time. If certain assumptions are made, these data can then be used to model and compute software reliability metrics. These metrics attempt to measure and predict the probability of failure during a particular time interval, or the mean time to failure (MTTF). The parallels between software reliability metrics and software security metrics have been discussed by Littlewood et al. (Littlewood et al., n.d.). The paper also highlights some challenges that have to be overcome before any meaningful attempt to devise metrics for security based on the reliability metrics. The paper also discusses the usage of probability-based framework to model security breaches analogous to the modeling of faults in the context of reliability. Time cannot be a good criterion when it comes to security and hence the paper suggests the usage of “effort” in the place of “time”. The paper also suggests having the Mean Time To Breach metric analogous to MTTF.
3.3 Discussion

Of all the metrics discussed above only the quality metrics seem to be the most likely candidates for consideration in the context of security metrics. But this too, is not without drawbacks. This indicates the need for the development of new metrics focused exclusively on security. Current measurements about security are highly subjective and need a high involvement of the human element. This needs to be reduced. Clearly, much more is to be done in the area of security metrics, as the currently available metrics do not address the security issue effectively. The work of Jansen can be a good starting point for the commencement of research in this area (Jansen, 2009). The paper suggests the usage of Historical data collection and data mining techniques and AI Assessment techniques in the development of security metrics. With the increasing demand for secure systems, the development of security metrics would be well worth the effort. Such metrics can be of immense help to the software engineers in the engineering of secure software systems. It needs to be pointed out here that, even in the context of security, different systems may require different levels of security. Another point worth mentioning here is that, currently security seems to be taken into account and measured only for systems where “very high” security is demanded such as in military systems. Problems faced in this context are similar to that faced in the development of “ultrahigh” reliable systems, the evaluation of which has been notably unsuccessful. This suggests that, in the development of security metrics, one should initially focus on systems where the security requirements are also modest. Once these attempts prove successful, work can be done for ultra-high secure systems as well.
3.4 Conclusion

This chapter attempted a survey of many existing metrics and explored their applicability for Software Security Measurement. The chapter finds that although some of the existing metrics can have some utility in Software Security Measurement, none of them provide a concise, correct and accurate estimate of Security of Software, and much effort is needed to justify the usage of these metrics for security. Metrics focused exclusively on security need to be developed and this requires a clear understanding of the applicability, utility and shortcomings of the existing metrics.