A Survey of Metrics Employed to Assess Software Security

Hadeel Alabandi

Submitted to the graduate degree program in Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas School of Engineering in partial fulfilment of the requirements for the degree of Master of Science.

Thesis Committee:

Dr. Prasad Kulkarni: Chairperson

Dr. Andy Gill

Dr. Heechul Yun

Date Defended:

5/9/2016

The Thesis Committee for Hadeel Alabandi certifies
That this is the approved version of the following thesis:

A Survey of Metrics Employed to Assess Software Security

Committee:

Dr. Prasad Kulkarni: Chairperson

Dr. Andy Gill

Dr. Heechul Yun

Date Approved:

5/11/2016

Abstract

Measuring and assessing software security is a critical concern as it is undesirable to develop risky and insecure software. Various measurement approaches and metrics have been defined to assess software security. For researchers and software developers, it is significant to have different metrics and measurement models at one place either to evaluate the existing measurement approaches, to compare between two or more metrics or to be able to find the proper metric to measure the software security at a specific software development phase. There is no existing survey of software security metrics that covers metrics available at all the software development phases. In this paper, we present a survey of metrics used to assess and measure software security, and we categorized them based on software development phases. Our findings reveal a critical lack of automated tools, and the necessity to possess detailed knowledge or experience of the measured software as the major hindrances in the use of existing software security metrics.

Contents

Chapter 1

Introduction

Security of software is a difficult property to measure. However, it is important to know if a specific system is reliable and will work properly, even when it is attacked [23]. Researchers have suggested different metrics to evaluate if and how vulnerable a software is to external attacks. Such measurements give the level of inherent security in each software development phase. These metrics depend on specific software and developmental attributes and can be used to compute the software's security score. Since each kind of measurement evaluates security at different level or for specific attributes and characteristics, making a decision to choose the proper measurements depends on the type of system security evaluation is needed.

Metrics developed to measure a software's security and risk level have been designed to be quantitative [41] or objective [30], and help assess a system's *vulnerability risk score*. Security metrics are beneficial as they reveal the reliability and development quality for the software system. In addition, metrics can also help to minimize the impact of a software system attack, by creating awareness about the system weaknesses, and improving its security scale. Software metrics have been

designed to be employed during different stages of the software life-cycle – design, implementation and testing, deployment, and maintenance life-cycle stages. Metrics have been proposed to work at the abstract architectural level or the more concrete software coding level. Examples of some popular metrics used for security assessment include approaches like the Common Vulnerability Scoring System (CVSS) [26] and Common Vulnerabilities and Exposures (CVE) [12].

As software and technology pervades all aspects of our day-to-day life, security researchers are devising various metrics, approaches and techniques to measure software security [22]. At the same time, researchers, IT experts and software developers need to have the ability to assess various types of measurements techniques; to be able to compare between different security metrics in the same category or to find a suitable metric based on it's category or the software phase during which the measurement is to be taken. The main goal of this research is to present a survey for various security metrics and measurement approaches, along with a classification of the metrics based on the software development stages. This research will support security researchers by describing and comparing several mechanisms in a single document. Our work will also assist others by providing the knowledge about the impact and the benefits of each metric; and which security characteristics could be measured using certain techniques.

This thesis compiles a survey about software security and quality metrics, and mechanisms to measure a software's security level. The security metrics presented here have been collected from previous research papers in the same area. The past research works summarized in this work include security measurement approaches that present a specific value or score to indicate the software security on a defined scale. The metrics are categorized based on software development

phases – including design, implementation, testing, deployment or maintenance phase. There are also some metrics included in this survey that can be used in multiple software phases.

Chapter 2

Related Work

In this section we present related surveys on software security metrics. Mellado et al. present some metrics that measure software security level during the design software development phase [22]. This evaluation defines the software properties covered by each metric. This work also compares the various metrics based on their security properties for the design stage.

Morrison et al. also conduct a study of software security metrics [23]. Their classification is based on conventional metrics like dependency, effort and complexity. For each category, the authors describe the related papers, the software development phases covered along with their evaluation, where possible.

Chowdhury et al. present a survey of software security measurement approaches related to the source code defects and weakness [9].

Evesti et al. focus on "Self-Adaptive" systems in their work [14]. Self-adaptive systems can react against attacks and have the ability to detect vulnerabilities by observing the whole system's security activities. The authors discuss the advantages and the difficulties to measure such systems, and present the metric's requirements to measure them.

Verendel provides a survey for the quantitative approaches to measure security [38]. This work aims to validate if software security can be measured, and if the metric scores accurately represent the actual security risk.

Jonsson and Pirzadeh present a framework that defines metrics by dividing them based on how the system behaves and it's effect on the environment [16]. The authors section software characteristics into protective, behavioural and correctness features. They focus on the first and the second properties and describe how to define metrics based on them.

Our work is different than [22] and [16] since it categorizes metrics based on different software development phases. Morrison et al. used conventional metrics, and mentioned phase classification, but they do not present details about metrics and measurement approach for each phase as we do in this thesis [23]. While [14] focuses only on security measurements for "Self-Adaptive" systems, our survey covers many different kinds of software systems. The survey presented by [38] aims to validate quantitative measurements, while our work presents a detailed survey to help researchers and software developers find different mechanisms and approaches to evaluate the security risk of the software at different phases.

Chapter 3

Software Development Phases and Security metrics

In this section we describe metrics and measurement approaches used to assess the software security state and risk in each phase of software development process.

3.1 Design Phase Metrics

Design is a very important software phase as it is one of the earliest stages of the software development process. Failures and weaknesses discovered at this early stage significantly reduce the risk at the later development stages. In this section we present some important research works that have been suggested to measure and assess the security at the design phase. Some of these mechanisms are standards used to assist security design evaluation process, while others define design stage metrics that have been employed to evaluate the software security using a design file that contains the specification of the system.

3.1.1 Towards a Measuring Framework for Security Properties of Software [31]

This paper concentrates on determining the quantitative security properties that can be assessed, and considers the entire software (not just some parts of it). They use a *principle and practices* security list as a reference to select the properties to assess the security properties at the design phase, such as a line of defense, size of attack surface and complexity.

3.1.2 Security Metrics for Object-Oriented Designs [4]

The authors of this work define a set of design security metrics to measure the security level for the flow of confidential information in multi-class programs, especially object-oriented programs in the design phase. The metrics used in this work are based on security quality properties for the program design, and include the entire class design levels such as coupling, composition and design size. A note of SPARK's and UMLsec need be added to the design to be able to compute the metrics using their UML tool. The result of the computation is a number between 0 and 1, with 1 being the worst. This metric can be used to evaluate more than one design for the same program to find the most secure one.

3.1.3 Security Estimation Framework: Design Phase Perspective [29]

This work developed a framework that includes a procedure to assess security at the design phase. The procedure includes 5 different stages that aim to find the security factors and metrics, assess them, and finally rate the software security level.

3.1.4 NIST 800-55 Security Metrics Guide for Information Technology Systems [35]

This work produced a full-system measurement approach to assess the information system by checking if its security controls are powerful and beneficial. This work focused on the security control implementation of the system. The authors evolved three measurement approaches to evaluate the implementation of security policies, effectiveness of security services delivery, and impact of security failures. This approach supports the design development stage, and helps to take decisions regarding security controls needed in the system.

3.1.5 Common Criteria or ISO/IEC 15408 [15]

CC or common criteria defines the product's security functional requirements that will be assessed and verified in the evaluation process. The result of evaluating the product shows the security confidence level, which indicates how much it meets the requirement defined in the common criteria.

3.1.6 An Efficient Measurement of Object Oriented Design Vulnerability [1]

In this work the authors present a metric, called the vulnerability propagation factor (VPF), to measure the risk level of software. The assessment is based on how the classes communicate, and the possibility of spreading the vulnerability from one class to the other classes on the tree-like structure of the object-oriented design. The result of VPF is based on the number of affected classes for each vulnerable class; and lower VPF value, means lower risk level of the software design.

3.1.7 Design Phase Metrics Comparison

According to the research papers we have discuss above, we will show a comparison between their metrics, measurement approaches and standards. We found that The frameworks presented in [31] and [29], and the VPF metric in [1] can be used to assess the security of the overall software. All of the design metrics and standards show a result (i.e numrical, risk categories or levels) that can be used to compare the security of various designs. While the frameworks in [31] and [29], NIST 800-55, and CC (ISO/IEC 15408) can be used to assess any type of software designs, the metrics presented in [4] and VPF in [1] can only assess object-oriented designs.

3.2 Implementation and Testing Phase Metrics

There are several software defects that cannot be discovered early in the design phase. It is important and most cost efficient to detect such defects in the implementation software development phase to minimize their risk to the later phases. Researchers have developed several metrics to assess the security approaches employed to find vulnerabilities and assess the software risk during the implementation stage. When the software is implemented, it's significant to guarantee that it's reliable and secure before it is deployed. In this section we describe some of metrics to measure the level of security of the software source code during he implementation and testing phases.

3.2.1 Security Metrics for Source Code Structures [9]

This work defines three metrics to assess the soundness of a system's source code structures. The assessed security level is based on finding source code defects

and weaknesses. Coupling corruption propagation is one of the metrics presented to measure how defects and weakness in one part of the software affects other methods in the source code. The other two metrics presented are stall ratio and critical element ratio; the first one is about finding the statements that don't let the program make progress and they focus here on such statements in the loop. The last metric finds possible paths an unauthorized input could take that will lead to an attack on the code.

3.2.2 Prioritization of software security intangible attributes [11]

This approach proposes using code properties that can be measured to estimate those that are more difficult to quantify, like software security. It works using the Analytical Hierarchy Process (AHP) computation to assess the measurable attributes scores, and the assessment of system stakeholders using two-dimensional approach that uses an attributes' hierarchy. The main attributes are the intangible properties, then there are one or more levels of sub-attributes used to get the security priority score for the main one.

3.2.3 Side-channel vulnerability factor: a metric for measuring information leakage [13]

The authors of this work define Side-channel vulnerability factor (SVF) as a security metric to measure the level of infiltration of the information which attacker can gain. The SVF metric is based on the patterns the attacker might use, and the execution patterns in the machine that could be attacked. This metric computes the interconnection between the two kinds of patterns to get the level of difficulty to attack the system and be able to exploit the information

leakage.

3.2.4 Deploying Suitable Countermeasures to Solve the Security Problems within an E-learning Environment [27]

This work explores the security issues and their exposures for E-learning systems, including techniques to fix some of these security problems. The authors present Mean Failure Cost (MFC) metric which depends on 4 elements: stakes, dependability, impact and threat vector. They explore some security metrics trying to minimize the effect of the stakes factor.

3.2.5 A Hierarchical Security Assessment Model for Object-Oriented Programs [5]

This work propose a multi-level tree-like model to measure the security of object oriented programs, especially the one written in Java. This security assessment is based on the properties of the code structure; it also depends on the static analysis where they evolve an automatic tool that works with Java bytecode after compiling. The security level final score is based on the classified data, and computed in different levels. The first level includes design properties obtained by the static analysis, then use them to get the metrics to assess upper-level of design properties in the next level. The third level uses the second level metrics to measure design concepts. The metrics are combined to get the final result of security evaluation of the system. This approach is for the developer, as it allows to compare the security level of two versions of a specific program.

3.2.6 A New Security Sensitivity Measurement for Software Variables [8]

The authors present a new metric to measure the full-system security level. It is based on the possibility of violating security properties when the system get attacked at the variable level. This metric uses model checking to verify for each security property if it will be violated when an attack occur. The measurement only uses the risky variables in the program code to assess the security level. This metric helps to minimize the security weaknesses and defects in the later stages; also it just focuses on the risky parts of the code, which is better than looking at the whole code as that might take a long time.

3.2.7 Evaluating Security Controls Based on Key Performance Indicators and Stakeholder Mission [32]

This work presents a Cyberspace Security Econometrics System (CSES) that helps to improve the security when the system operates, and to develop a secure system. CSES assesses the cost of possible violations of the system security for each stakeholder when it gets attacked using the mean failure cost. Determining the possible lose goes through computing stakeholder matrix, dependency matrix, impact matrix and threat matrix. Having the knowledge of the potential loss for each stakeholder in the system helps to manage and maintain the security risk.

3.2.8 Is Complexity Really the Enemy of Software Security? [33]

This paper investigates if the complexity of the source code impacts the vulnerability of software. The study in this paper is accomplished by assessing some of complexity metrics (i.e nesting complexity, paths and other 7 metrics) statically

at code-level to verify if the complexity is a valid indicator to high security risk of the system.

3.2.9 Implementation and Testing Phase Metrics Comparison

We discuss here a comparison between the security metrics that could be used to assess security during implementation and testing phase based on the research papers we have discussed above. We found that the metrics in [9], [11], [13], [5], [8] and [33] don't assess the security of the overall system. While measuring security in [9], [5] and [33] based on some of the code structure properties, It's depends on some security properties in [11], [13] and [8]. The final score of all implementation and testing phase metrics can be used to compare the security of different source codes (i.e numrical score); however, metrics in [27] and [32] used to maintain and monitor the systems. All measurement approaches can assess any type of system except those in [27] and [32] which assess large systems (i.e systems that has many users), and the measurement model in [5] which only evaluates object oriented source code.

3.3 Deployment Phase Metrics

Independent of the assessment performed (or not) by the software developers, it is also important for the software end-user to know if the software is secure, and to be able to compare the security risk level of different software products that provide similar levels of functionality so that they may choose the most secure one. There are some tools and metrics that have been proposed to enable measuring the level of software security by the end-user. In this section we present the most prominent software security metrics that can be employed by the end-users of

software, as they do not rely on any design or development-time knowledge or availability of source-code.

3.3.1 Analyses Of Two End-User Software Vulnerability Exposure Metrics [43]

This work defines two vulnerability exposure metrics that could be used by the end-user to assess the security exposure, and to be able to compare between two similar software products. These metrics are based on the number of vulnerabilities that have been detected and notified to the distributor, and the software life duration (how fast are the vulnerabilities fixed since they were discovered). One of the metrics is (AAV) average active vulnerabilities, which defines the average number of vulnerabilities for specific software that has been reported to and fixed by the software distributor; and the other one is (VFD) vulnerability free days metric that defines the likelihood of not having any live vulnerabilities in a certain day.

3.3.2 How dangerous is your Android app?: an evaluation methodology [6]

This paper presents a new security risk analyser that could be used by the end-user to assess software security for Android applications. This analyser uses both static and dynamic analysis. It works by finding the authorizations needed by the app, and then compares the authorizations with the tasks that the end-user has called in the run-time. Finer-grained models are used to profile the discovered risks based on it's risk classifications; and fuzzy logic approach is used to assess these risks. The final result of the analyser is a risk score, and the finer-grained

classification report.

3.3.3 Measuring the attack surfaces of two FTP daemons [20]

The authors of this paper use the attack surface measurement approach to compare between different FTP server implementations. They choose ProfFTPD and Wu-FTPD, the most popular and open source FTP servers for this work. The comparison between the servers is based on the attack surface of the data, the channels and the number of methods. The measurement result is verified by looking at the co-relation between the number of vulnerabilities discovered in the past for each server implementation, and the attack surface metric result for each server.

3.3.4 Ontology-based Security Assessment for Software Products [40]

This work presents an evaluation of the trust of a software system by building an ontology for maintaining vulnerabilities. The ontology helps by providing information about the security-relevant functional and non-functional requirements of the software. The ontology also shows the proof of confidence regarding whether or not the software system is free of vulnerabilities. The approach depends on different standards like CVSS and CWE. To support the ontology approach, a self-controlled tool was also developed to assess the level of the software trustworthiness.

3.3.5 EVMAT: an OVAL and NVD based enterprise vulnerability modeling and assessment tool [44]

In this work, the authors implement a tool called EVMAT that automatically computes the score of enterprise vulnerability. They use a topology for modelling vulnerability by giving a weight to all of the company objectives, then use CVSS to find the severity score for each objective. EVMAT uses OVAL to collect the information about the software system features related to all resources, and NVD to get information about each vulnerability.

3.3.6 A model for quantitative security measurement and prioritization of vulnerability mitigation [37]

In this paper, the authors present a model for assessing the security risk quantitatively for each node in a network system. The measurement formula depends on gathering the vulnerabilities of a specific node, assessing their risk scores, and ordering them based on their roles and effects inside the network.

3.3.7 Security Metrics for Software Systems [41]

The authors in this work develop a tool to assess the trust level of software systems. This approach depends on the flaws that lead to higher number of system attacks, their risk scores and how frequent each vulnerability related to that flaw will occur. The authors get the list of vulnerabilities for the software system from the CVE database.

3.3.8 Temporal metrics for software vulnerabilities [42]

In this work, the authors try to adjust the metrics' equations provided by CVSS 2.0. They discuss that the CVSS 2.0 equations have an issue about the the impact score; if the impact is high or low, the score will be the same. They adjust f(impact) so it has multiple cases to represent integrity impact, confidentiality impact and availability impact accurately in the base score equation. In this paper, they focused more on adjusting temporal and environmental equations. A tool with an interface was also developed to help the end-user calculate the score automatically.

3.3.9 An Approach to Measuring a System's Attack Surface [17]

The paper presents an approach that uses the system attack surface resources as indicator to the security level for specific software. This technique provides the ability for end-users and developers to compare between two software's security levels. The metric computation is based on the system data, methods and channels.

3.3.10 An Approach to Analyzing the Windows and Linux Security Models [34]

This work presents different metrics to assess an operating system's risk level. This measurement is based on the possibility of privilege violations caused by one of the OS characteristics. Different formulae are presented for the various OS to measure risk. However, using these metrics needs an expert user. The risk level could be concluded from the final result of the specific formula for each specific OS.

3.3.11 Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security [24]

The paper presents a tools to help the administrator monitors and maintains the security level of the system. It shows the vulnerabilities in the system using a *privilege graph*, which is a tree-like structure. The security metrics employed here are computed using a Markovian approach, which is based on the potential paths the attacker could use to attack the system using the privilege graph. The result reflects the likelihood of attacking the system. The result is achieved using mean effort to failure (METF) metric; the lower the value of the metric, the higher the indicated risk level, and lower security level of the system.

3.3.12 SAVI: Static-Analysis Vulnerability Indicator [39]

The authors in this paper present a security measurement approach based on analyzing the source code statically for open-source web applications. The analysis inspects the potential vulnerabilities in the system, and then use them to compute the security measure. They propose a static analysis vulnerability indicator (SAVI) to compare between two or more applications based on their source code, to find the one that is most secure. This approach can only be used for open-source deployed software as it requires the source code.

3.3.13 Deployment Phase Metrics Comparison

We present a comparison between deployment phase security metrics and measurement approaches discussed above. The measurement approaches used in [6], [44], [37] [34], [24] and [39] can be used only to assess specific type of software. In [6], only andorid application can be evaluated; and in [44] and [24] secu-

rity assessments for enterprise systems presented. In addition, the network nodes risk level assessed (They didn't compute the risk level of the overall network) in [37]; and in [34] different metrics to assess the security of various operating systems described. SAVI metric presented in [39] can only used to evaluate the security of the deployed open source software as it requires the availability of the source code. All of the presented metrics show a quantitative result which help to either compare the security or risk level of different software that has the same level of functionality, or to maintain and monitor the system security. Vulnerability databases used to assist the software security evaluation in [40], [44], [37] and [41]. The metrics used in [17] and [20] are based on assessing the security of the attack surface; and the measurement model in [6] and the metric in [39] use static analysis to measure the software security.

3.4 Maintenance Phase Metrics

Software developers also need to measure the software risk level in the maintenance stage. The evaluation during this stage is either to guarantee that the software is still secure, to check it's security level when it get attacked, or to check it's security after making some changes on the software. This section presents some of the security metrics that have been devised for *developers* (rather than end-users) to be used in the maintenance phase.

3.4.1 Using Software Structure to Predict Vulnerability Exploitation Potential [45]

To improve the measurement results of evaluating the vulnerabilities risk, and to minimize the need of user interference to supply the information needed to

measure the risk level, the authors presents a metric that depends on some of the software characteristics like reachability, existence of risky call, and entry point. The authors built a predictor that works using a machine learning approach and SVMS to determine the vulnerabilities exposure and effect.

3.4.2 Using Attack Surface Entry Points and Reachability Analysis to Assess the Risk of Software Vulnerability Exploitability [46]

Security metrics employing the CVSS metrics have a weakness since they don't examine the influence of the software characteristics during measuring the risk level. This work defines a metric that focuses on the software characteristics and design, while being based on vulnerability reachability. Whenever there is a vulnerability that could be exploited through an entry point using a system call, that entry point risk will be estimated by computing the ratio of the software's attribute privileges and types to the rights needed to attack it.

3.4.3 Enterprise Software Management Systems by Using Security Metrics [7]

This paper presents a scheme for ordering vulnerability priorities based on common vulnerability scoring system(CVSS). It describes base, temporal and environmental metrics used in CVSS. The authors also present number of operational metrics that could assess systems of large business, and can be used by managers, operational team, IT team and some other metrics.

3.4.4 Taxonomy of quality metrics for assessing assurance of security correctness [25]

This work proposes a classification that aims to show the quality level of the process of proving and validating the security accuracy. This process is a part of security assurance, and the quality properties needed are represented in the CC and ISO-IEC 15408, and the level accomplished is represented by SSE-CMM. This technique gives a quantified result which helps to simplify maintaining the operational phase's security procedure.

3.4.5 Comparing Vulnerability Severity and Exploits Using Case-Control Studies [2]

The authors propose to check the validation of vulnerabilities' severity scores as risk level indicators. A case-control approach is used to evaluate vulnerability and the data that could be used to attack the system. In addition, they check the validity of CVSS severity scores as risk level indicators, and they use a couple of elements that affect the risk level to improve the CVSS risk indicator accuracy.

3.4.6 Estimating risk levels for vulnerability categories using CVSS [36]

This work aims to improve the accuracy of measuring the security of a system. It puts vulnerability under specific classification, and tries to assess the risk level for each group of vulnerabilities. From the score of each group, it computes the final risk level for the whole system. This approach also uses vulnerability prioritization to determine what causes high risk in the system.

3.4.7 Maintenance Phase Metrics Comparison

We present a comparison of the metrics to assess security of the maintenance phase. All of them can evaluate the security for any type of software except the one in [7] which only measures the security of the enterprise system. While the metrics in [45] and [46] are based on software characteristics (i.e like reachability, existence of risky call, and entry point), those in [36], [2] and [7] are based on CVSS. The final result includes the scores of all the known vulnerability only in [25] and [36]. In addition, the final result of all metrics expect the one in [45] can be used to to monitor and maintain the software security, or to compare between two versions of the software; The metric in [45] don't show a security or risk score, it just shows if there is a vulnerabilities could lead to attack the system.

3.5 Multiple Phases Metrics

In this section, we discuss some of the metrics, approaches or tools that could be used during more than one phase of software development.

Design , Testing and Implementation Phase

3.5.1 Complexity, Coupling, and Cohesion Metrics be Used as Early Indicators of Vulnerabilities? [10]

As it is significant to increase the security level in the early stages of software development, the authors examine complexity, coupling and cohesion (CCC) metrics to explore if having less cohesive systems with greater coupling and complexity is a valid indicator to higher system risk. If the relation between CCC and the higher risk is determined, then they explore which one of the CCC metrics could

be used to assess the system security.

3.5.2 Automated software architecture security risk analysis using formalized signatures: Metric to measure the security at different architecture levels through the development cycle [3]

The authors present a scheme for detecting the system weakness during design and implementation phase, and a tool to analyse the system architecture of security defects. This scheme is based on security metrics and scenarios, and object constraint language (OCL). The formal OCL signature is used to find if there is a match for it in the system or to assess the security. Detecting a match indicates that system could be exposed to such an attack.

3.5.3 CWE (Common Weakness Enumeration) [21]

CWE helps to improve and manage the security level of software beside assisting security tools. It shows the weakness that can be evaluated for specific software in the implementation, operational and design phases.

Design and Deployment

3.5.4 A tool for security metrics modeling and visualization [19]

This work proposes MVS 2.0, which is a tool developed to assess and maintain the security in the system during the design and deployment phases. It includes various metrics ordered from the most comprehensive, which is located in the lower level, and less details can be given as we move up to the higher levels. This tool connects metrics with the system security details such as security requirements and controls to make it easy for the user to monitor the system, and this data is

always updated when there is an update or change in the system while it runs.

Implementation and Maintenance

3.5.5 An Analyzer-based Software Security Measurement Model for Enhancing Software System Security [18]

The paper presents an approach that uses the analysis to gather security metrics over software. It helps to detect security weaknesses so it can be maintained to reduce software risk. Static analysis is used to collect the software security metrics during implementation phase (i.e code-level metrics). In other side, they use dynamic analysis to gather the maintenance phase metrics. In order to get the software security level, both metrics from static and dynamic analysis are integrated based on defined formulas.

Chapter 4

Discussion

In this thesis, we present a survey of security metrics and measurement approaches categorized by software development phases. In this section we describe some of the main findings of this work.

We found that researchers have developed many different metrics to measure the security level of software. While some metrics can only be used in one phase of software development, there are others that can be used over multiple phases. However, many current metrics don't show the security level for the overall software. Instead, several metrics represent a partial score that only covers some of the security properties. Likewise, some metrics just represent the score of one vulnerability in that software, which is less beneficial unless there is another way to utilize such partial scores to get an overall (security) value.

We also found that many metrics use the comprehensive CVE, NVD or other vulnerabilities databases to get a list of vulnerabilities and their details, which they need and use for their formulae and computations. Most popular vulnerability databases use a complex scoring system to compute the severity score of each discovered/reported vulnerability. CVSS [26] is the most popular standard open

scoring system (used by NVD) that helps an organization estimates its system vulnerability risks. The severity score in CVSS is based on base, temporal and environmental metrics. The computation formula for each of these main metrics is in turn based on a number of other sub-metrics that depend on the properties of the discovered vulnerability. Environmental score is based on the temporal score, which depends on the base score. The final CVSS score can be 10 or lower, with the higher number indicating a higher risk level. Another scoring system is CMSS [28] to assess the risk level of the misuse vulnerabilities that help in attacking the system. The CMSS score derived from the CVSS and its scoring formula based on the three metrics: base, environmental and temporal metrics.

We decided to focus our effort in this survey on *automated* tools that don't need human intervention. We discovered that there are only a few such tools. One such automated tool was presented in [6], which is designed to analyse and measure the risk level of Android applications. This tool can enable the end-user to make a security decision either by giving the final risk score as a numerical value and it's corresponding category (i.e low, average, high and not acceptable). It can also provide details about the operations that may cause risky behavior for the application, as well as the risk category for each one. Such analysis is helpful and significant to know, and can allow the user to make an installation decision based on the application's risk factor.

Another automated security measurement tool was proposed in [40]. This tool uses an ontology to maintain and monitor the vulnerabilities. The final result of the analysis shows the evidence of the trustworthiness (i.e the evidence that the software doesn't include any vulnerability) of the software. However, operating this tool needs some manual information, such as the temporal exploitability and

report confidence scores from the user to start the analysis. While this tool is compatible with all software binaries, the score given by this tool is based on the reported vulnerabilities retrieved by the ontology from the NVD database. As a result, its score may not be accurate as it doesn't cover any undetected defects and vulnerabilities.

EVMAT [44] is another automated tool developed to calculate the overall vulnerability score of an entire enterprise system. It can be used by system administrators as it requires some knowledge to specify accurate weights for the all of the system resources. The vulnerability details and resources features are automatically retrieved to be used in the final security score computation. This tool also uses the NVD database to get vulnerabilities informations. As the tool mentioned previously in [40], this tool also covers only the reported NVD vulnerabilities in it's computation.

Wang et al. also reveal a tool to calculate the adjusted CVSS metrics [42]. This tool is also not fully automated; however, the tool provides some information regarding the properties of attributes it needs to calculate its score.

A predictor was developed in [45] with the aim of minimizing the knowledge needed from the user. This approach uses machine learning and SVMS. It acquires some knowledge of the software structure properties, so it can also be used by developers. This tool does not measure the risk or security level of the system. Instead, it just tries to conclude if vulnerabilities could be used to attack the system or not.

Another tool, called MVS 2.0 [19], was developed to evaluate the system security during the design and deployment development stages. It needs some knowledge about MVS, how it works and it's output. This tool can be used by expert

users and developers who have the knowledge and the experience of how to use it.

Based on our exploration, we conclude that there is a general lack of automated tools for end-users to conveniently score the security of deployed software. This limitation is a serious issue only for measuring the software security in the deployment stage. However, automation for all phases will result in more accurate measurement than manual entry of the information, which might result in simple data or enormous database entries. Some of the tools also need knowledge or experience that the end-user at the deployment stage may not possess. In addition, tools that could be used by the end-user and don't require any knowledge are often restricted to specific kinds of applications or software, or it might be related to specific OS such as Android.

Chapter 5

Conclusion

This thesis conducted a survey of security metrics, and categorized them based on the software development stages. Our survey collects many important research works for security metrics and tools that measure the security or risk level of software. Our survey found a general lack of automated security measurement tools, especially for use by end-users in the deployment stage. Tools that need more user interference also necessitates more knowledge to use the tool. Some automated tools can be used only for specific types of software. We expect that this survey will help developers and researchers to understand the state of art in security metrics approaches and tools to measure security. This knowledge will help researchers develop better metrics and automated tools to advance the measurement of software security.

References

[1] A. Agrawal, S. Chandra, and R. A. Khan. An efficient measurement of object oriented design vulnerability. In *Availability, Reliability and Security, 2009. ARES '09. International Conference on*, pages 618–623, March 2009.

[2] L. Allodi and F. Massacci. Comparing vulnerability severity and exploits using case-control studies. *ACM Trans. Inf. Syst. Secur.*, 17(1):1:1–1:20, Aug. 2014.

[3] M. Almorsy, J. Grundy, and A. S. Ibrahim. Automated software architecture security risk analysis using formalized signatures. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 662–671, May 2013.

[4] B. Alshammari, C. Fidge, and D. Corney. Security metrics for object-oriented designs. In *Software Engineering Conference (ASWEC), 2010 21st Australian*, pages 55–64, April 2010.

[5] B. Alshammari, C. Fidge, and D. Corney. A hierarchical security assessment model for object-oriented programs. In *2011 11th International Conference on Quality Software*, pages 218–227, July 2011.

[6] A. Atzeni, T. Su, M. Baltatu, R. D'Alessandro, and G. Pessiva. How dangerous is your android app?: An evaluation methodology. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MOBIQUITOUS '14, pages 130–139, ICST, Brussels, Belgium, Belgium, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[7] A. N. P. Bhanudas S. Panchabhai. Enterprise software management systems by using security metrics. *International Journal of Science and Research*, 2012. (Accessed on 03/21/2016).

[8] X. Cheng, N. He, and M. S. Hsiao. A new security sensitivity measurement for software variables. In *Technologies for Homeland Security, 2008 IEEE Conference on*, pages 593–598, May 2008.

[9] I. Chowdhury, B. Chan, and M. Zulkernine. Security metrics for source code structures. In *Proceedings of the Fourth International Workshop on Software Engineering for Secure Systems*, SESS '08, pages 57–64, New York, NY, USA, 2008. ACM.

[10] I. Chowdhury and M. Zulkernine. Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities? In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1963–1969, New York, NY, USA, 2010. ACM.

[11] R. T. Colombo, M. S. Pessôa, A. C. Guerra, A. B. a. Filho, and C. C. Gomes. Prioritization of software security intangible attributes. *SIGSOFT Softw. Eng. Notes*, 37(6):1–7, Nov. 2012.

[12] CVE. Cve - common vulnerabilities and exposures. https://cve.mitre.org/about/, 2016.

[13] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan. Side-channel vulnerability factor: A metric for measuring information leakage. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 106–117, Washington, DC, USA, 2012. IEEE Computer Society.

[14] A. Evesti, H. Abie, and R. Savola. Security measuring for self-adaptive security. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, ECSAW '14, pages 5:1–5:7, New York, NY, USA, 2014. ACM.

[15] ISO/IEC. *ISO/IEC 15408:2005 Information technology- Security techniques - Evaluation criteria for IT security (Common Criteria v3.0)*, 2005.

[16] E. Jonsson and L. Pirzadeh. A framework for security metrics based on operational system attributes. In *Security Measurements and Metrics (Metrisec), 2011 Third International Workshop on*, pages 58–65, Sept 2011.

[17] P. K.Manadhata, K. M.C.Tan, R. A.Maxion, and J. M.Wing. An approach to measuring a system's attack surface. Technical Report ADA476805, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2007.

[18] S. T. Lai. An analyzer-based software security measurement model for enhancing software system security. In *Software Engineering (WCSE), 2010 Second World Congress on*, volume 2, pages 93–96, Dec 2010.

[19] O.-M. Latvala, J. Toivonen, J. Kuusijärvi, and A. Evesti. A tool for security metrics modeling and visualization. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, ECSAW '14, pages 3:1–3:7, New York, NY, USA, 2014. ACM.

[20] P. Manadhata, J. Wing, M. Flynn, and M. McQueen. Measuring the attack surfaces of two ftp daemons. In *Proceedings of the 2Nd ACM Workshop on Quality of Protection*, QoP '06, pages 3–10, New York, NY, USA, 2006. ACM.

[21] R. Martin. Common weakness enumeration (cwe v1.8). Technical report, National Cyber Security Division of the U.S. Department of Homeland Security, 2010.

[22] D. Mellado, E. Fernández-Medina, and M. Piattini. A comparison of software design security metrics. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, pages 236–242, New York, NY, USA, 2010. ACM.

[23] P. Morrison, D. Moye, and L. Williams. Mapping the field of software security metrics. Technical Report I-CA2301, Department of Computer Science, North Carolina State University, Raleigh, NC, 2014.

[24] R. Ortalo, Y. Deswarte, and M. Kaaniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Transactions on Software Engineering*, 25(5):633–650, Sep 1999.

[25] M. Ouedraogo, R. M. Savola, H. Mouratidis, D. Preston, D. Khadraoui, and E. Dubois. Taxonomy of quality metrics for assessing assurance of security correctness. *Software Quality Journal*, 21(1):67–97, 2011.

[26] K. S. P. Mell and S. Romanosky. A complete guide to the common vulnerability scoring system (cvss 2.0). Technical report, NIST and Carnegie Mellon University, 2007.

[27] N. Rjaibi and L. B. A. Rabai. Deploying suitable countermeasures to solve the security problems within an e-learning environment. In *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN '14, pages 33:33–33:38, New York, NY, USA, 2014. ACM.

[28] E. Ruitnbeek and K. Scarfone. The common misuse scoring system (cmss): Metrics for software feature misuse vulnerabilities. Technical Report 7517, National Institute of Standards and Technology, NIST Interagency Report, Jul 2009.

[29] R. A. K. S. Chandra and A. Agrawal. Security estimation framework: Design phase perspective. In *2009 Sixth International Conference on Information Technology: New Generations*, pages 254–259, April 2009.

[30] R. Savola. Towards a security metrics taxonomy for the information and communication technology industry. In *Proceedings of the International Conference on Software Engineering Advances*, ICSEA '07, pages 60–, Washington, DC, USA, 2007. IEEE Computer Society.

[31] R. Scandariato, B. De Win, and W. Joosen. Towards a measuring framework for security properties of software. In *Proceedings of the 2Nd ACM Workshop on Quality of Protection*, QoP '06, pages 27–30, New York, NY, USA, 2006. ACM.

[32] F. T. Sheldon, R. K. Abercrombie, and A. Mili. Evaluating security controls based on key performance indicators and stakeholder mission. In *Proceedings of the 4th Annual Workshop on Cyber Security and Information Intelligence Research: Developing Strategies to Meet the Cyber Security and Information Intelligence Challenges Ahead*, CSIIRW '08, pages 41:1–41:11, New York, NY, USA, 2008. ACM.

[33] Y. Shin and L. Williams. Is complexity really the enemy of software security? In *Proceedings of the 4th ACM Workshop on Quality of Protection*, QoP '08, pages 47–50, New York, NY, USA, 2008. ACM.

[34] X. Song, M. Stinson, R. Lee, and P. Albee. An approach to analyzing the windows and linux security models. In *5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering,Software Architecture and Reuse (ICIS-COMSAR'06)*, pages 56–62, July 2006.

[35] M. Swanson, N.Bartol, J. Sabato, J. Hash, and L. Graffo. Security metric guide for information technology systems. In *NIST Sepical Publication 800-55 Revision 1*. National Institute of Standards and Technologies, 2008.

[36] A. Tripathi and U. K. Singh. Estimating risk levels for vulnerability categories using cvss. *Int. J. Internet Technol. Secur. Syst.*, 4(4):272–289, May 2012.

[37] A. Tripathi and U. K. Singh. A model for quantitative security measurement and prioritisation of vulnerability mitigation. *Int. J. Secur. Netw.*, 8(3):139–153, Nov. 2013.

[38] V. Verendel. Quantified security is a weak hypothesis: A critical survey of results and assumptions. In *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*, NSPW '09, pages 37–50, New York, NY, USA, 2009. ACM.

[39] J. Walden and M. Doyle. Savi: Static-analysis vulnerability indicator. *IEEE Security Privacy*, 10(3):32–39, May 2012.

[40] J. A. Wang, M. Guo, H. Wang, M. Xia, and L. Zhou. Ontology-based security assessment for software products," presented at the. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies, Oak*, 2009.

[41] J. A. Wang, H. Wang, M. Guo, and M. Xia. Security metrics for software systems. In *Proceedings of the 47th Annual Southeast Regional Conference*, ACM-SE 47, pages 47:1–47:6, New York, NY, USA, 2009. ACM.

[42] J. A. Wang, F. Zhang, and M. Xia. Temporal metrics for software vulnerabilities. In *Proceedings of the 4th Annual Workshop on Cyber Security and Information Intelligence Research: Developing Strategies to Meet the Cyber Security and Information Intelligence Challenges Ahead*, CSIIRW '08, pages 44:1–44:3, New York, NY, USA, 2008. ACM.

[43] J. L. Wright, M. McQueen, and L. Wellman. Analyses of two end-user software vulnerability exposure metrics. In *Proceedings of the 2012 Seventh International Conference on Availability, Reliability and Security*, ARES '12, pages 1–10, Washington, DC, USA, 2012. IEEE Computer Society.

[44] B. Wu and A. J. A. Wang. Evmat: An oval and nvd based enterprise vulnerability modeling and assessment tool. In *Proceedings of the 49th Annual Southeast Regional Conference*, ACM-SE '11, pages 115–120, New York, NY, USA, 2011. ACM.

[45] A. A. Younis and Y. K. Malaiya. Using software structure to predict vulnerability exploitation potential. In *Software Security and Reliability-Companion (SERE-C), 2014 IEEE Eighth International Conference on*, pages 13–18, June 2014.

[46] A. A. Younis, Y. K. Malaiya, and I. Ray. Using attack surface entry points and reachability analysis to assess the risk of software vulnerability exploitability. In *High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on*, pages 1–8, Jan 2014.