Information and Software Technology 102 (2018) 146–159

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Mapping the field of software life cycle security metrics

Patrick Morrison*,[a], David Moye[a], Rahul Pandita[a,b], Laurie Williams[a]

[a] North Carolina State University, Raleigh, NC, USA
[b] Phase Change Software, LLC, Golden, CO, USA

ARTICLE INFO

Keywords:
Metrics
Measurement
Security

ABSTRACT

Context: Practitioners establish a piece of software's security objectives during the software development process. To support control and assessment, practitioners and researchers seek to measure security risks and mitigations during software development projects. Metrics provide one means for assessing whether software security objectives have been achieved. A catalog of security metrics for the software development life cycle could assist practitioners in choosing appropriate metrics, and researchers in identifying opportunities for refinement of security measurement.

Objective: The goal of this research is to support practitioner and researcher use of security measurement in the software life cycle by cataloging security metrics presented in the literature, their validation, and the subjects they measure.

Method: We conducted a systematic mapping study, beginning with 4818 papers and narrowing down to 71 papers reporting on 324 unique security metrics. For each metric, we identified the subject being measured, how the metric has been validated, and how the metric is used. We categorized the metrics, and give examples of metrics for each category.

Results: In our data, 85% of security metrics have been proposed and evaluated solely by their authors, leaving room for replication and confirmation through field studies. Approximately 60% of the metrics have been empirically evaluated, by their authors or by others. The available metrics are weighted heavily toward the implementation and operations phases, with relatively few metrics for requirements, design, and testing phases of software development. Some artifacts and processes remain unmeasured. Measured by phase, Testing received the least attention, with 1.5% of the metrics.

Conclusions: At present, the primary application of security metrics to the software development life cycle in the literature is to study the relationship between properties of source code and reported vulnerabilities. The most-cited and most used metric, vulnerability count, has multiple definitions and operationalizations. We suggest that researchers must check vulnerability count definitions when making comparisons between papers. In addition to refining vulnerability measurement, we see research opportunities for greater attention to metrics for the requirement, design, and testing phases of development. We conjecture from our data that the field of software life cycle security metrics has yet to converge on an accepted set of metrics.

1. Introduction

Software system builders, owners, operators, and users seek assurance that their interests, communications, and data are secure. McGraw [1] defines software security as "engineering software so that it continues to function correctly under malicious attack." Many aspects of the software development life cycle, including software requirements, design, implementation, and testing contribute to the security of the running software. Measuring whether security has been appropriately addressed at each stage of software development is likely to be a precondition to assuring the release of secure software. We seek to investigate whether some fundamental security questions that development teams might ask can be answered with security metrics such as:

- Have we considered security in each stage of the software life cycle?
- What are the security risks to which our software is exposed?
- What are the security assurances we can give to users of our software?

As a foundation for answering these questions, we examine the

* Corresponding author.
E-mail addresses: pmorrison@ncsu.edu (P. Morrison), cdmoye@ncsu.edu (D. Moye), rpandita@phasechange.ai (R. Pandita), lawilli3@ncsu.edu (L. Williams).
URLS: http://www.rahulpandita.me/ (R. Pandita), https://collaboration.csc.ncsu.edu/laurie/ (L. Williams).

https://doi.org/10.1016/j.infsof.2018.05.011
Received 1 July 2017; Received in revised form 29 May 2018; Accepted 29 May 2018
Available online 30 May 2018
0950-5849/

scholarly literature on the use of metrics for security during the software development life cycle. We adopt the term 'software life cycle security metrics' to describe metrics used to measure security-related aspects for software and the artifacts used during its development.

Providing useful metrics for the security of a software system is a difficult undertaking because: (1) we lack effective models and measurements of software security risk; (2) security must be considered from the earliest stages of software development, but may only be evident once the software is in use; (3) the people building and using the software must be considered in estimation of security risk and assurance [2]. Many elements of the software development life cycle contribute to software security. Pfleeger and Cunningham [3] consider dimensions ranging from the specification of systems to protocol verification to the psychology of software designers, users and attackers. The range of dimensions suggests that a wide range of metrics is needed to properly represent security for assessment and prediction.

We conjecture that comprehensive approaches to providing security in software are likely to rely on comprehensive measurement of security in software. As shown by industrial schemes such as Microsoft's Security Development Life cycle [4], the SafeCode initiative [5], and the Cigital "*Building Security In Maturity Model* [6]", security should be addressed at every phase of software development. Researchers seek theories to explain security properties, and empirical validation of measurements of those properties. Both industrial developers and researchers require an understanding of the available software security metrics

The goal of this research is to support practitioner and researcher use of security measurement in the software life cycle by cataloging security metrics presented in the literature, their validation, and the subjects they measure.

As a means of identifying the security properties measured, we conduct a systematic mapping study of the metrics that have been applied to measuring the security in the software development life cycle. According to Budgen et al. [7], systematic mapping studies are "intended to 'map out' the research that has been undertaken rather than to answer a detailed research question." We include metrics measuring the software, and the artifacts, processes and people involved in the software development life cycle, as well as metrics measuring aspects of security (e.g. measures of confidentiality) or its absence (e.g. counts of vulnerabilities).

To assess the extent of the field of software life cycle security metrics and their evaluation and use, we pose the following research questions to map out the state of the field:

RQ1: What software life cycle security metrics have been proposed in the literature?
RQ2: What are the roles, artifacts, and process elements being measured by software life cycle security metrics?
RQ3: What validation approaches are used for software life cycle security metrics?
RQ4: During what phases of the software development life cycle are security metrics measured?

Our initial search yielded a set of 4818 papers. We narrow the set to 71 papers that propose, evaluate or report on 324 security metrics for the software life cycle.

Our contributions include:

- A systematic map of the metrics used to evaluate the security properties of software and its development and use.
- An online catalog of software life cycle security metrics available on our project website [8].

The remainder of this paper is organized as follows: Section 2 provides a glossary and background information on metrics. Section 3 presents related work. Section 4 describes the methodology we follow in executing the mapping study. Section 5 provides our summarization of the data collected. Section 6 presents our discussion of the results. Section 7 reports on Limitations. Section 8 concludes.

2. Background

In this section, we present a glossary of metric-related terms, and literature with focus on software security metrics to provide grounding for: (1) the topic of security metrics in software development, (2) our mapping study, and (3) our classification scheme.

2.1. Definitions

Attribute: A property associated with a set of real or abstract things that is some characteristic of interest [9].

Attack: An intentional act by which an entity attempts to evade security services and violate the security policy of a system; A method or technique used in an assault [10].

Fault: An incorrect step, process, or data definition in a computer program [9].

Indicator: Any observable characteristic that correlates with a desired security property [11].

Measure: A way to ascertain or appraise value by comparing it to a norm; to apply a metric [12].

Measurement: The process by which numbers or symbols are assigned to attributes of subjects in the real world in such a way as to describe them according to clearly defined rules [13].

Metric: A quantitative measure of the degree to which a system, component, or process possesses a given attribute; the defined measurement method and the measurement scale [9]. *Note, some of the observed metrics are subjective or qualitative in nature and fail to meet this definition fully. Those metrics are included to better cover the state of the art, and to avoid dismissal of potentially useful research.*

Risk: The combination of the probability of an event and its consequence [14].

Security Metric: A metric which measures a security property [11].

Security Property: A property related to security concerns, for example confidentiality, integrity, availability, authentication, authorization, non-repudiation [1].

Software Security: We adopt McGraw's notion of "engineering software so that it continues to function correctly under malicious attack" [1]. For our notion of malicious attack, we also reference the IEEE definition of software security: "Protection of information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them." [14]

Vulnerability: A fault or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy [10].

2.2. Software life cycle security metrics

Software engineering researchers have used multiple definitions for the words "metric" and "measure". For the purposes of this review, we are liberal in the definitions we accept from the literature. A good metric should be conceptually specific, quantitatively measurable, practically attainable, consistently measured without subjective criteria, and time-dependent [15]. However, even when metrics appear to be useful, difficulties arise attempting to validate metrics and determine their overall usefulness and practicality [16]. In addition, metrics, including security metrics, are not valuable if the results of applying them cannot be understood effectively [17].

What is a *security metric*? Jansen [12] quotes and discusses three variant definitions from the literature, highlighting a diversity of usage and a reliance on human judgment rather than more repeatable measures. For our purposes, we define a security metric to be a metric whose attribute is a measure of a security property, or some indication of a violation of a security property. We further limit our study to the

domain of software development; software, the source code from which software is produced, the related test suites, documentation, developer tools, development team roles, software process phases, and the like.

3. Related work

Verendel [18] presented a survey focused on measuring operational security, addressing the ability to "function correctly under malicious attack." Our mapping study additionally considers the engineering of secure software, seeking measurements of the process, tools, people, and the software produced. Rudolph and Schwarz [11] surveyed scholarly articles on "security indicators", where an indicator is defined as "an observable characteristic that correlates with a desired security property." In addition to what Rudolph and Schwarz studied, we seek to characterize the subjects being measured. Meneely et al. [19] reviewed metric validation, and suggest a scheme for choosing validation criteria. We only consider validation in terms of the high-level approach chosen by the researchers, e.g., whether the metric(s) were evaluated through user study, theoretical analysis, or researcher opinion.

Savola's [20] security metrics taxonomy, characterized security metric properties and applications. At a high level, Savola provided three categories of metrics:

- Organizational metrics describe attributes of organizational programs and processes.
- Technical metrics describe software artifacts, e.g., requirements, specifications, designs, code.
- Operational metrics describe running systems and their environments.

We capture these categories through the use of a 'Subject Type' category for metrics on related subjects.

4. Methodology

We subdivided how we approached the mapping study into four components: our search strategy for identifying papers, our selection criteria for including papers, our classification scheme for collecting data on each metric, and our procedure for extracting metric information from each paper.

4.1. Search strategy

In this section, we describe the process used to conduct our systematic mapping study.

4.1.1. Databases

We based our selection of online databases on the common databases used in Software Engineering Systematic Literature Reviews (SLRs), and in Systematic Mapping Studies (SMSs), and on sources used in previous software security metric literature reviews [11,17]. The data sources in this study include online databases, conference proceedings, and academic journals. The list is as follows: ACM Digital Library, IEEE Xplore, and Elsevier.

4.1.2. Search terms and strategy

We developed a list of search terms with which to evaluate titles, shown in Table 1. For each term associated with a research question, we identified synonyms in the titles and abstracts of previous surveys [11,18] of security metrics. The synonyms are shown in Table 2. The first two authors developed terms and synonyms independently, and then combined lists, retaining terms and synonyms agreed on as most relevant.

After searching using various combinations of the proposed keywords and evaluating their performance using the set of papers in our Quasi-Gold-Standard (QGS, Section 4.2.1), we selected our basic search phrase:

(software OR computer OR information)
AND (security OR vulnerability OR assurance)
AND (metric OR indicator OR quant*)

We created database search phrases using the base search phrase and syntax appropriate to each database, searched the title, abstract, and full text fields of each database, and collected papers based on those search phrases.

Table 1
Research question keywords.

Research Question	Keyword
RQ1	"software", "security", "metric"
RQ2	RQ1 terms
RQ3	RQ1 terms + "validate", "evaluate"
RQ4	RQ1 terms + "phase", "life cycle", "process

Table 2
Keyword synonyms.

Keyword	Synonyms
software	"application", "computer", "information"
security	"vulnerability", "assurance"
metric	"measurement ", "measure ", "indicator ", "attribute", "property"
validate	"validation", "evaluate", "evaluating", "evaluation ", "quantify", "quantifying", "quantification", "quantified ", "quantitative", "assess", "assessment", "measure", "measurement", "appraisal", "analyze", "measuring", "analysis", "assurance", "scoring"

4.1.3. Search process evaluation

We followed Zhang et al. [21] in evaluating the quality of our search results. An ideal search process would return all relevant papers (*sensitivity* = 1.00), and only relevant papers (*precision* = 1.0). Such a set of papers would be a "*gold standard*." In practice, we do not know the set of relevant papers in advance, so we estimated, using what Zhang terms a "*quasi-gold standard*" (QGS) which is a set of relevant papers in the literature, chosen prior to the search process. The QGS is used as a measure of how well each search string locates relevant papers. Zhang et al. [21] defines sensitivity as the ratio of the number of retrieved papers to the total number of relevant studies. Quasi-sensitivity (QS) is the number of papers returned by a search to the number of returned papers that are present in the QGS. QS estimates how well the search string locates relevant papers within the searched corpus. By measuring the performance of the search in returning members of the QGS from a given search engine, compared with the number of QGS papers in the search engine, an estimate can be made of search performance. For example, if there are 10 of the QGS papers in the ACM library, and the search string returns 8 of them, QS would be 0.8.

4.2. Selection criteria

We developed a list of criteria to assess whether the papers found during the search process met our objectives.

1. Inclusion Criteria:
   - Related to measuring software security in software development artifacts and the software development process.
   - Measurements or metrics are main subject
   - Refereed publication
   - Published since 2000
   - Written in English
   - Available online

2. Exclusion Criteria:
   - Sources measuring security in non-software systems
   - Sources related to identity, anonymity, privacy
   - Sources related to forgery or biometrics
   - Sources related to network security
   - Sources related to encryption
   - Sources limited to database security
   - Books, dissertations
3. Study Quality Assessment: For papers that aligned with our inclusion criteria, and which were not excluded by our exclusion criteria, we developed a Quality Assessment Checklist for whether to include each paper, as follows:
   - Is a primary or secondary goal of the paper to describe, define, or evaluate a metric or measurement of software security?
   - Is the paper peer-reviewed?
4. Scoring: Each paper was independently rated by two raters, namely, the first two authors of this article. We established a scoring procedure for resolving differences between raters when disagreement was found.
   - Question Scoring Scale: No. 0, Partial: 0.5, Yes: 1.
   - Complete agreement
     (a) Yes from both raters: paper is selected
     (b) No from both raters: paper is rejected
   - Partial agreement combinations between 0 and 2. Raters discuss, find agreement, or agree to disagree.
     (a) Agreement processed according to the rules for complete agreement.

In the case of unresolved disagreement, papers are selected for evaluation in the subsequent step in the protocol.

4.2.1. Search results

The first and second authors identified a set of 39 software life cycle security metrics papers, developed by reviewing the papers collected in the previous security metrics literature reviews [11,18]. Each author read the titles and abstracts of the collected papers independently, applied the study quality assessment criteria, and made a list of candidates. The authors then discussed each list, applying the following selection procedure:

1. If both authors agreed the paper described a software life cycle security metric, it was included in the final QGS list.
2. If both authors agreed the paper did not describe a software life cycle security metric, it was excluded from the final QGS list.
3. Disagreement was discussed. If agreement could be reached, the appropriate action listed above was taken. If agreement could not be reached, the paper was included in the final list.

After our process, the QGS consisted of 17 papers. The results of each database search were compared with the QGS set of papers. The quasi-sensitivity for each database search is reported in Table 3.

We conducted three passes over the collected papers, applying the Study Quality Checklist and scoring procedure to each paper's title, abstract, and a 10-minute 'quick read' of each paper. After reviewing the titles and coming to agreement, the raters (first two authors) identified 4815 papers to continue investigating. After reviewing the abstracts, the raters identified 107 papers to continue investigating.

Table 3
Search results.

Database	Initial search	QS
ACM Digital Library	2302	0.77
IEEE Xplore	2000	0.80
Elsevier	1280	0.63

After giving each paper a 'quick read' and coming to agreement, the raters identified 71 papers for further investigation.

We assigned a reference code to each of the 71 papers, P1-P71. We used these codes throughout the remainder of the paper to refer to individual papers. See Table A1 for the reference codes and bibliographic references of the included papers. The QGS papers are marked with asterisks in Table A1.

4.3. Metric classification scheme

To support answering our research questions, we developed a set of data elements to be collected for each metric.

We begin with the IEEE definition of metric, "A quantitative **measure** of the degree to which a system, component, or process possesses a given **attribute**; the defined measurement **method** and the measurement **scale**" [9], defining the following data elements to describe each metric:

- Metric name: the "given name" of the metric defined or used in the paper (RQ1).
- Measure: A description of how the value of the metric is determined, in terms of the metric's Method and Scale (RQ2).
  - Method: Classification of how the metric value is measured, where "Quantitative" indicates objective, systematic, empirical measurements and "Qualitative" indicates subjective measurements based upon observation (RQ2).
  - Scale: denotes type of measurement scale (e.g., Nominal, Ordinal, Interval, Ratio), or specific instance of measurement scale (e.g., Count, Probability, Duration, Rate, Currency, Percentage) (RQ2).
- Subject Type: Classification of whether the metric measures a 'System', 'Component', or 'Process' (RQ2).
- Subject: Description of the entity, 'an object or event in the real world'[13], being measured. Extracted from papers, or synthesized and named based on assessment of extractors. For example, 'Source Code', 'Component', and 'Vulnerability'. The complete list is provided in Results, Section 5.2. (RQ1, RQ2)
- Attribute: Description of 'a feature or property' [13] of the subject being measured. Extracted from papers, or synthesized and named based on assessment of extractors. For example, 'Lines of Code' or 'Classified' (RQ2).
- Validation Approach: To characterize how metrics are validated, we recorded how the metrics are evaluated by the researchers in the paper. Values: Opinion, Theoretical, Academic user study, Industry user study, Reports from production (RQ3).
- Phase: We recorded the phase in the software development life cycle with which the metric is associated: Requirements, Design, Implementation, Testing, Operations (RQ4).

For demographic purposes, we assigned unique paper and metric numbers to identify each metric and its usage across papers. For audit purposes, we tracked the name of the extractor (first or second author who identified and extracted the metric from a particular paper) and auditor (first, second, or third author who cross-checked the metric extracted) for each row, as well as the extraction and audit dates.

4.4. Data extraction

The metric extractor (first or second author) read a paper, identified each metric defined or used in the paper, and collected the data for each element in the metric classification scheme. The first author applied the extraction procedure described in the classification guide to every paper in the final set of 71 papers, with spot checks by the second and third authors.

4.4.1. Metric categorization

To discover relationships between similar metrics with varying

names in different papers, we applied card sorting [22] to categorize the metrics. Card sorting is a technique for identifying commonalities between collections of data items by using multiple raters to compare and group related data items. Over the course of four sessions, the first, second, and third authors discussed each metric and its definition, and placed it in a proposed category and subcategory. When a metric did not fit into existing categories, we created a new category or subcategory. Over the course of categorizing the metrics, we moved metrics between categories and merged or removed categories to capture our shared understanding of the metric definitions. After we assigned each metric to a category, we took another pass, reviewed each metric and category definition and revised our choices.

5. Results

This section presents our results in the form of summaries of the data extracted from the selected papers. Based on the data extraction procedure, we tabulated the metrics, subjects, scales, evaluation means, and uses. The tabulated data provides an overview of 'evidence clusters' and 'evidence deserts' for software security in the software development life cycle. The full data set is available online on our project website [8].

5.1. RQ1: What software life cycle security metrics have been proposed in the literature?

In total, we identified 324 unique metrics in the 71 selected papers. The list of metrics is available online [8]. In this paper, we present each metric category and subcategory we identified (as described in Section 4.4.1), ordered by the number of metrics in each category/subcategory. We do not present each metric, but we give representative examples of metrics from each category. Table 4 shows our final list of eight categories with associated subcategories, with metric counts for each category and subcategory. The following sub-sections will discuss each of the eight categories.

Table 4
Security metric categories and dubcategories.

Metric category	Subcategory	Subcat Total	Cat Total
Security Properties			86
	Access Control	42	
	Confidentiality and Privacy	19	
	Integrity	10	
	Audit	9	
	Availability	6	
Implementation			84
	Source Code	44	
	Security Source-Code	30	
	Version Control	8	
	Data Flow	2	
Project Management			41
	Process	17	
	Design	17	
	Defect	7	
Incident		36	
	Vulnerability	20	
	CVSS	16	
Resource			34
	People	23	
	Cost	9	
	Time	2	
Defensibility			19
	Attackability	10	
	Attack Surface	9	
Aggregate			18
	Aggregate	18	
Security Requirement			16
	Usecase-Misusecase	12	
	Requirement	4	
Grand Total			324

5.1.1. Security Properties Metrics (86)

We identified metrics as part of the Security Properties Metrics category when the metric is primarily a measure of a security property (see Section 2.2). We identified five subcategories of Security Property metrics: Access Control Metrics, Audit Metrics, Availability Metrics, Confidentiality and Privacy Metrics, and Integrity Metrics.

Access Control Metrics (42): Papers P2 and P9 present lists of questions to verify the presence of desirable access control mechanisms in the system being measured, e.g. 'Authentication non-repudiation', 'Authorization', 'User authority', and 'Intrusion detection'. Paper P33 defines categorical metrics for detailing how authentication and authorization are implemented in the system being described, e.g. 'User authentication scheme' and 'User identification scheme' (e.g. token, password). Paper P38 expands on the idea of 'User identification scheme' to define 23 metrics for describing how passwords are implemented in a system, e.g. 'Alphabet Size', 'Maximum Life Time', and 'Users Training'.

At a more abstract level of design, P17 defines 'Compartmentalization' (M73), 'the number of independent components that do not trust each other (performs authentication and authorization for requests/calls coming from other system components) that the system is based on to deliver its function.'

Operational metrics for access control focus on tracking successful and unsuccessful access attempts on system resources. 'Monitoring system use' (M170, P4) is a count of unauthorized access attempts on system resources, and 'Login Attempt Count' (M162, P33) is a count of login attempts.

Confidentiality and Privacy Metrics (19): Papers P2, P23, and P33 present lists of questions to verify the presence and quality of mechanisms supporting confidentiality. 'Confidentiality security attribute' (M83, P23) includes a set of requirements for maintaining confidentiality 'in terms of required behavior during execution expressed in terms of data and transformation of data by programs.' 'Side-channel Vulnerability Factor' (M249, P59) 'measures information leakage through a side-channel by examining the correlation between a victimâs execution and an attackerâs observations.'

Integrity Metrics (10): Papers P2 and P33 present lists of questions to verify whether integrity concerns are addressed in the system being measured, covering, for example, 'Data integrity' (M148, P2), and 'Validation checks per input' (M143, P33). The Common Vulnerability Scoring System [23] (CVSS) 'Integrity Impact' metric (M146) is a categorical measure (None, Partial, Complete) of the impact of a vulnerability on the integrity of the affected software. 'Integrity' (M145, P30) is the ratio of 'risky' classes to total classes, where 'risky' is defined by a set of attributes of the measured class, e.g. 'references user data', 'references password'.

Audit Metric (9): Papers P2 and P33 present lists of questions to verify the presence and quality of audit mechanisms. 'Auditing' (M33,P2) is a question verifying the presence of techniques or technology for recording 'who did what, when and where?' 'Audit trail comprehensiveness' (M34, P33) is a question verifying that all failed authentication and authorization attempts are logged. 'Log File Protection Scheme' (M161, P33) is a check on the presence of a scheme for maintaining log file integrity. Operational Audit Metrics include 'Ratio of log interactions to system interactions' (M209,P33) and 'Audit logging' (M33, P4) the ratio of log files that are monitored to the total number of log files.

Availability Metrics (6): Papers P2 and P33 present lists of questions to verify whether availability concerns are addressed in the system being measured. 'Availability' (M100, P2) is a question verifying that a denial of service mitigation scheme is in place. 'Critical Service Interaction' (M99,P33) is a count of the services depended on by the system being measured.

5.1.2. Implementation Metrics (84)

We identified metrics as part of the Implementation Metrics category when the metric is primarily a measure of the source code, source code execution, or source code management. We identified four subcategories of Development/Implementation Metrics: Data Flow Metrics, Source Code Metrics, Security Source Code Metrics, and Version Control Metrics. We now discuss each of these subcategories.

Source Code Metrics (44): We found 84 references to traditional source code metrics, e.g. Source Lines of Code (SLOC), Churn, McCabe's Cyclomatic Complexity, and Coupling, in 21 papers: P1, P5, P9, P18, P19, P25, P28, P30, P33, P36, P44, P46, P47, P53, P55, P58, P60, P61, P68, P69, P71. We included object-oriented metrics in this category. While we focused on metrics of security properties in this paper, we make four observations on how source code metrics are applied in the context of security. First, traditional source code metrics have been used in varying combinations to predict vulnerabilities, so far with limited success. Second, traditional source code metrics are used to normalize security concerns to code size. For example, 'Relative Churn' (M53, P71) normalizes churn to source file size. Third, traditional source code metrics are often 'tweaked', or 'adapted', e.g. 30-Day Churn (M53, P71), Min Churn, Max Churn, Mean Churn, to broaden studies of the correlations between the base metrics and the dependent variables under study. Fourth, traditional source code metrics are sometimes augmented with security information to yield new metrics, for example, the 'Classified', and 'Critical' labels on attributes, methods, etc in P1.

Security Source Code Metric (30): Paper P1 defines the largest set of security source code metrics among the papers surveyed. In the framework designed by the researchers, attributes are marked as 'classified', as determined by the developer, and classes are marked as 'critical' if they contain classified attributes. The researchers build up a system of metrics for classes and methods that read and write the classified attributes and interact with critical classes, and then define a set of metrics representing various security properties in terms of the lower level definitions. For example, 'Classified Attributes Inheritance' (M55, P1) indicates that a class inherits classified attributes from another class, and 'Critical Class Extensibility' (M94, P1) indicates that a critical class is not final. Similarly, 'Variable Vulnerability' (M291, P5) is based on marking variables as security-relevant and evaluating their relationships.

Version Control Metric (8): The authors of Paper P68 use metrics such as 'Commit Count' (M71), 'Star Count' (M260), and 'Fork Count' (M130) for subject selection of repositories for vulnerability analysis.

Data Flow Metrics (2): *We collected measurements of data flow in the Data Flow metrics category*. Two papers, P45 and P46, define data flow metrics for identifying vulnerable code. An exemplary metric in this category is the Component Dependency Graph (CDG) discussed in P46.

5.1.3. Project Management Metrics (41)

We identified three subcategories of Project Management Metrics: Design Metrics, Process Metrics, and Defect Metrics. We now discuss each of these subcategories.

Design Metric (17): Paper P9 presents a series of checklist metrics for kernel management, component interfaces, and network planning. Paper P19 presents a series of metrics for measuring each stage of the software development life cycle, including measures of design effort, security design decisions, and the ratio of security design decisions to total design decisions.

Process Metric (17): Paper P15 presents four metrics describing the strength of verification applied to the security requirements and mechanisms of the system being measure: Coverage (M92), Depth (M101), Rigor (M234), and Independence of Verification (M137). Coverage is a categorical measure of the extent to which all security functionalities are examined during verification. Depth is a categorical measure of the portion of security functionalities that are examined during verification. Rigor is a categorical measure of the maturity of the security functionality verification process. Independence of Verification is a categorical measure of the extent to which security functionalities are evaluated by persons other than the implementers of the functionality.

Paper P52 includes a checklist of qualitative measures of process strength and risk, including CMMI Level (M66 ,P52), Development Risk (M112, P52), Environment Risk (M117, P52), and Project Management Risk (M202, P52).

Paper P61 describes Oracle's philosophy and implementation of software life cycle security metrics. Process-specific metrics they mention include 'Number of security bulletins issued per year' (M252, P61), 'Volume of email correspondence with vulnerability handling team' (M292, P61), 'Use of (automated) tools' (M282, P61) by the development team.

Defect Metric (7): Paper P19 presents a set of count and ratio metrics for making project management decisions, based on the number of defects and vulnerabilities introduced during implementation. Defect Count (M95, P19, P44) is used for process feedback.

5.1.4. Incident Metrics (36)

We identified two subcategories of Incident Metrics: Vulnerability Metrics, and CVSS Metrics. We now discuss each of these subcategories.

Vulnerability Metric (20): Twenty papers defined or used a count of vulnerabilities, 'Vulnerability Count' (M293, P14, P18, P19, P25, P36, P44, P45, P46, P53, P54, P56, P58, P60, P61, P62, P64, P67, P68, P69, P71). We found multiple theoretical and practical definitions of Vulnerability Count. In terms of theory, authors provided five definitions of 'vulnerability' in the seven papers that defined the term:

- an instance of a fault in the specification, development, or configuration of software such that its execution can violate an implicit or explicit security policy (P18, P60, P62, P71, citing [24]);
- 'a defect, which enables an attacker to bypass security measures' (P36, citing [25]);
- 'a weakness in the security system that might be exploited to cause loss or harm' (P36, citing [26]);
- 'Vulnerability is a software defect that can be exploited to cause a security breach.' (P69)
- an 'unknown system characteristic' (P63)

In terms of practical definitions of vulnerability, we identified five schemes used to count vulnerabilities:

- CVE counts from public vulnerability databases (NVD, OSVDB,Bugtraq) (P14,P36,P53,P60,P64,P67,P68, P69, P71).
- CVE counts from public vendor vulnerability databases (MFSA, RHSR) (P18,P25,P46,P60,P68).
- CVE counts from privately-curated vulnerability databases based on public sources (P46).
- Vulnerability counts from development team vulnerability databases (Cisco) (P44,P62).
- Vulnerability counts from analysis tools (P39,P58).

Paper P7 presents three date attributes for vulnerabilities, Vulnerability Discovery Date (M296, P7), the date on which the vulnerability is discovered, Vulnerability Patch Date (M301, P7), the date on which the solution to the vulnerability is shipped, and Vulnerability Disclosure Date (M63, P7), the date on which the vulnerability is publicly announced.

While not made explicit, the status of a vulnerability as patched or un-patched can be inferred from the presence or absence of a Vulnerability Patch Date. Further the status of the vulnerability as being

open or resolved can be similarly inferred.

Vulnerability Density (M294, P28, P36, P58) is a count of vulnerabilities normalized to the number of (thousands of) SLOC in the vulnerable code unit.

'Structural Severity' (M263, P64) is a qualitative, categorical metric indicating the vulnerability's proximity to the attack surface.

'Window of exposure' (M307, P66) is the length of interval of time taken by the security operations team to patch a vulnerability once it has been disclosed.

'Vulnerability-Contributing-Commit' (M304, P71) indicates the presence of a commit in the version control repository that contributed to the introduction of a post-release vulnerability.

'Vulnerability Discovery Rate' (M297, P69) is a count of vulnerabilities per time unit (usually month).

'Vulnerability Free Days' (VFD) (M298, P14) is 'the percent of days in which the vendors queue of reported vulnerabilities for the product is empty.'

CVSS Metrics (16): Eight papers used CVSS [23] metrics (P8, P13, P21, P41, P42, P53, P64, P70). CVSS metrics describe characteristics and impact of vulnerabilities, and so can be seen as a more detailed notion of vulnerability than a simple count. The CVSS metrics are categorized as Base, Temporal, and Environmental, representing characteristics of the vulnerability that are constant, that change over time, and that are associated with the environment in which the vulnerability was found. Details of the metrics can be found in [23], and online [27].

5.1.5. Resource Metrics (34)

We identified metrics as part of the Resource Metrics category when the metric is primarily a measure of some cost or effort incurred in the course of security attacks or defense. We identified three subcategories of Resource Metrics: Cost Metrics, People Metrics, and Time Metrics. We now discuss each of these subcategories.

People Metrics (23): We identified five distinct classes of people metrics: Developer, Attacker, User, Project Manager, and Organization. 'Developer Count' (M107, P25, P53, P60) is a count of the number of developers who made changes to the software being measured. Authors defined Developer Count in the following ways:

- NumDevs - 'Files were changed by many developers' (P25).
- Number of Engineers - 'absolute number of unique engineers who have touched a binary and are still employed by the company' (P53).
- NumDevs - 'The number of distinct developers who changed the file' (P60).

P53 defines 'Number of Ex-Engineers' as the 'total number of unique engineers who have touched a binary and have left the company as of the release date of the software system.' Meneely (P71) defines variants of Developer count, including 'Number of distinct authors besides the commit author whose lines were affected by a given commit', and New Effective Author, indicating the first time a developer makes a commit.

Paper P25 contains a set of developer activity metrics (M185-M194) that describe developer relationships through graphs. 'Developer Risk' (M111, P52) measures concern over developer ability for a project. Similarly, 'Software Project Management Experience' (M251, P52) and 'User Risk' (M288, P52) measure expected strength and weakness of managers and users. 'Social Engineering Resistance' (M250, P29) measures how well users are able to resist attempts at social engineering. 'Information Leakage' (M139, P20) measures the degree of information transfer between developers on a software project.

Attackers are measured in terms of their capability (M28, P29), level of motivation (M29, P29, M30, P51), skill level (M31, P29, M31, P50), and speed (M32, P50). Two papers define attacker (threat agent) skill. Authors define Attacker Skill in the following ways:

- A qualitative assessment of attacker skill (Low, Medium, High) (P77).
- The expected value of a random variable in an item-response theory model (P92).

Organizations are measured in terms of the percentage of contribution to a piece of software's development (M192, P53) and the percentage of the organization that has received security training (M191, P61).

Cost Metrics (9): *We collected metrics related to financial concerns as well as attacker and defender effort, cost, and reward in the 'Cost Metric' category*.

We grouped measures of the value of an attack to an attacker and the cost of an attack to the defender as 'Attack Value' (M24), for example 'Damage Potential' (P34), 'Mean Failure Cost' (P24), and 'Annual Loss Expectancy' (P43).

We grouped ratios of Attack Value to Attack Effort as 'Attack Risk', for example 'Attackability' (M21, P22), and 'Security Risk Index' (M21, P29).

We grouped measures of the non-monetary costs of an attack to the attacker as 'Attack Effort' (M16, P22), for example the number of steps required to complete an attack (M216, P26), or the ease with which a vulnerability is exploited (M16, P29, M488, P64) Where 'Attack Effort' is measured in monetary terms (e.g. M14, P26, M292, P34), we refer to it as 'Attack Cost'.

Attack Cost: Two papers define Attack Cost (P26, P37). Authors defined Attack Cost in the following ways:

- 'a combination of everything the attacker must spend in order to successfully execute an attack (money, resources, time, effort, etc.)' (P26)
- 'Reflect the cost of T(Technique) in Attack Path. Let Cost(x) be the attack cost produced during the software Attack Path x. Attack Cost is quantifiable, and its unit is exp.' (P37)

By analogy with attacks, we defined 'Defense Effort', 'Defense Cost', 'Defense Value', and 'Defense Reward' (ratio of value to cost/effort). Two metrics focused on defense, 'Minimal cost for reduction of attacks' (M212, P26), and 'Return on Penetration Testing' (M337, P43).

Time Metric (2): Oracle Corporation measures 'Time to close bug/vulnerability' (M266, P61), a form of 'Mean time to repair'. 'Interval between violation and report' (M151, P33) is the length of the interval between the time the vulnerability is discovered and the time the vulnerability is reported to the development team. 'Attack Execution Time' (M17, P50) is the length of time required to execute the attack being measured.

5.1.6. Defensibility Metrics (19)

We identified metrics as part of the Defensibility Metrics category when the metric is primarily a measure of the ease or difficulty with which a system is attacked. We identified two subcategories of Defensibility Metrics: Attack Surface Metrics, and Attackability Metrics. We now discuss each of these subcategories.

Attackability (10): 'Attackability' (M27, P3) is the probability that an entity will be successfully attacked. 'Attack Count' (M15, P26) defines how many attacks on a system exist. The idea behind this metric is that the greater the number of attacks available for a system, the less secure the system is. 'Attack Prone' (M26, P47) is the likelihood that a component will be attacked (measured in Paper P47 by the presence of previous vulnerabilities). Paper P7 defines a set of metrics describing the probability that a system is secure, has been compromised, and has,

or has not, been repaired.

'Vulnerability Index' (VI) has three distinct definitions. In Papers P16 and P49, VI (M299) is the probability of a componentâs vulnerability being exposed in a single execution multiplied by number of expected executions. In Paper P56, VI is a qualitative (categorical) assessment of a state of a system (be it a router, a server or a client), which can be normal, uncertain and vulnerable. In Paper P49, VI indicates 'the percentage of vulnerability occurrences that have been removed from the corresponding artifact.'

Attack Surface Metrics (9): An 'attack surface' represents the paths in and out of a system (e.g. input and output parameters or fields), the data that travels those paths, and the code that protects the paths and the data [28].

Five of the selected papers used attack surface metrics (P11, P26, P34, P35, P64). We found two definitions of attack surface:

- A system attack surface is the set of ways in which an adversary can enter the system and potentially cause damage. (P11)
- A system attack surface is the subset of its resources that an attacker can use to attack the system. (P11, P26, P34, P35, P64)

'Sensitive Sink' (M248, P45) indicates that execution of the code labeled as a sensitive sink may lead to harmful operations, making it a form of exit point.

'Attack Graph Probability' (M18, P10) is defined as 'the intrinsic likelihood of an exploit *e* being executed, given that all the conditions required for executing *e* in the given attack graph are already satisfied.'.

Paper P67 defines 'Exploitability Risk' (M125), a qualitative measure of whether a vulnerability is accessible from a system's entry points.

Paper P11 developed a formalism for measuring attack surfaces and an 'Attack Surface Metric' (M23, P11) representing the attack surface size adjusted for the risk of sharing the data resources exposed through the attack surface. The authors of P11 conducted a validation study of the metric in Paper P34.

5.1.7. Aggregate Metrics (18)

We identified metrics as part of the Aggregate category when the metric is a combination of other metrics that aggregates across metrics and dimensions to present an overview of some aspect of the measured entity's security.

In the course of categorization, we identified a set of aggregates, combinations of other metrics designed to summarize the overall security state of the entity being measured. One of the simpler aggregates, 'CVSS Score' (M102, P41, P42, P54) is defined as a 'weighted average of CVSS impact (Confidentiality/Integrity/Access Impact) and exploitability (Access Vector, Access Complexity, Authentication) metrics.'. CVSS Score represents the severity of a single vulnerability. 'Security Resource Indicator' (M246, P28) combines numeric measures of the presence of four indicators: a security reporting email address, a security vulnerabilities list, a secure coding standard, and security configuration documentation. 'Security factor requirements' (M240, P2) combines metrics for Confidentiality, Privacy, Data Integrity, Authentication Non-repudiation, Auditing, and Intrusion detection to represent the overall security strength of a website. In contrast, 'Expected Vulnerability' (M124, P16) measures the overall security weakness of a system by aggregating measures of 'Vulnerability Index' (M299, P16) across the components of the vulnerable system. 'Security Metric' (M429a, P54) applies the same strategy as 'Expected Vulnerability', but aggregates CVSS Scores normalized to the number of vulnerability occurrences for the system being measured. Paper P1 represents the most substantial use of aggregates in the papers identified in our search. Beginning with the definition of classified attributes, the authors develop a system of metrics for the classes and methods that access the classified attributes, and then define a set of metrics representing various security properties in terms of the lower level definitions. For example, 'Readability of Classified Attributes' (RCA) (M219, P1) is composed of three lower level measures of attribute access via the system's classes and instances. RCA is, together with 'Readability of Classified Methods' (RCM) (M220, P1), a component of 'Fail-Safe Defaults' (PFSD) (M127, P1). PFSD is defined as a security principle metric, representing the degree to which the system's classified attributes are not accessible by default. A 'Total Security Index' (M275, P1) is computed using PFSD and six other aggregate security principle metrics representing the overall security of the software's object-oriented design.

5.1.8. Security Requirement Metrics (16)

We identified metrics as part of the Security Requirement Metrics category when the metric is primarily a measure of requirements and their specification, including, e.g., use cases. We identified two subcategories of Security Requirement Metrics: Usecase-Misusecase Metrics, and Requirement Metrics. We now discuss each of these subcategories.

Use-Misusecase Metrics (12): Paper P57 presented metrics for security requirements, security use cases, and security misuse cases, together with a checklist for measuring system security requirements. Paper P52 defined an overall measure of Requirements Risk (M230, P52). Two papers defined or used 'Security Test Case Count' (P19, P57), counts of security test cases and security misuse cases. Authors defined Security Test Cases in the following ways:

- 'represent security threats that the attacker might interact with to breach security and cause harm to the system' (P57);
- 'designed to detect security issues' (P19).

Given the basic definitions, the authors defined ratios of the security test case counts, relating them to, for example, total test case counts and failed/successful test cases.

Requirement Metrics (4) Paper P19 presented a series of metrics for measuring each stage of the software development life cycle, including counts of security requirements, missing security requirements, and the ratio of security requirements to total requirements.

Authors defined Security Requirement Count in the following ways:

- Total number of security requirements - number of security requirements identified during the analysis phase of the application (M245, P19).
- Ratio of security requirements - ratio of security requirements to the total number of requirements (M211, P19).
- Number of omitted security requirements - number of requirements that should have been considered when building the application (M244, P19).
- Ratio of the number of omitted security requirements - ratio of the number of security requirements not considered during analysis to the total number of security requirements identified during the analysis phase (M245, P19).
- Number of excluded security requirements that ensure input/output handling (M180, P57) - defined by counting 'No' answers of a set of questions, e.g. 'Is a specific encoding scheme defined for all inputs?', 'Are all the validations performed on the client and server side?'
- Number of excluded security requirements that ensure session handling - defined by counting 'No' answers for a set of questions, e.g. 'Is session identifier created on server side?', 'Is session identifier killed after a period of time without any actions?'

Table 5
Metric Ids by phase and subject.

Subject	Requirements	Design	Implementation	Testing	Operations	All
System	4	3	9	5	178	4
Source Code		5	98		10	
Component		3	65		6	8
Software version	3		12		17	
Design		9				
Requirements	8					
Repository			8			
Organization			1		5	2
Misuse case	8					
Commit			7			
Project						6
Security mechanism					4	
Service			1		2	
Model						2
User Account					1	
Total	23	20	201	5	223	22

5.2. RQ2: What are the roles, artifacts, and process elements being measured by software life cycle security metrics?

As described above in Section 5.1.5, we identified five distinct classes of people metrics: Developer, Attacker, User, Project Manager, and Organization. We identified 15 distinct subjects, sorted by metric count: System (147 metrics), Source Code (77), Component (71), Software version (29), Design (8), Misuse case (8), Requirements (8), Organization (7), Commit (7), Project (6), Security mechanism (4), Service (3), Model (2), and User Account (1). We identified five phases, sorted by metric count: Operations (151), Implementation (145), Requirements (22), Design (18), and Testing (5), with 22 metrics deemed applicable to all phases. We present the metrics organized by the software development phase in which they are measured, and the subject which they measure in Table 5. Please note that metrics may be repeated across categories (e.g. Vulnerability Count is used by various authors in the Design, Implementation, Testing, and Operations phases), inflating the metric counts in the table. Measures of running systems (System + Software Version) (151) comprise the largest collection of metrics, followed by measures of Source Code. Measures of the software life cycle and its non-source code artifacts are relatively few.

5.3. RQ3: What validation approaches are used for software life cycle security metrics?

We classified each metric in terms of six validation approaches. Table 6 lists each validation approach, the papers applying that validation approach, and the frequency of appearance of each technique by metric count and metric percentage. Papers and metrics may appear more than once, as some of the papers apply multiple validation approaches. Metrics counts and percentages reflect repeated use of metrics between papers.

5.4. During what phases of the software development life cycle are security metrics measured?

We classified each metric in terms of five development phases, and report on the phases measured by the metrics in the selected papers. Table 7 lists each development phase, the papers associated with that development phase, and the frequency of appearance of each phase by metric count and metric percentage. Papers and metrics may appear more than once, as some of the papers measure multiple development phases. Metrics counts and percentages reflect repeated use of metrics between papers.

The most common life cycle phase studied is Operations (50 studies), followed by Implementation (29 studies). The selected papers showed less emphasis on the Requirements (6 studies), Design (6 studies), and Testing (3 studies) phases of development. We identified four studies that defined metrics suitable for use across the development phases. For example 'Information Leakage' from P20 measures unwanted transfer of information between developers and project personnel during the development process.

6. Discussion

We report on the results associated with each research question in Section 5. In this section, we make observations on trends in the results, based on our analysis of the selected papers and identified metrics.

6.1. Themes in security metrics

In this section we present themes we identified in the collected set of metrics.

6.1.1. Vulnerability definitions and counts vary

Vulnerability Count was the most-referenced metric in our selected papers. However, vulnerabilities were counted in five different ways, none of which depended on the definitions given for vulnerability. Use of the CVE[1] records in the NVD database[2] was the most common vulnerability counting technique. Massacci and Nguyen (P46) observe that the NVD is a partial record of vulnerability data, and that curation of all available sources of data, for example other vulnerability databases, reports from software vendors, and bug tracking data internal to the development team is necessary to provide accurate vulnerability counts.

The NVD/CVE-based vulnerability counts are a record of public observations of security failures in software. Software development teams also track vulnerabilities internally and privately. Some projects (e.g. Google Chrome) handle vulnerabilities as a special case of defects, recorded and managed through the project's bug tracker. Some projects provide an email address for sending vulnerability reports, but do not publish their tracking mechanism or vulnerability data. Distinguishing whether a project maintains a list of vulnerabilities may come down to inquiries to the project team.

Because the definition of vulnerability varies, the implementation of

[1] Common Vulnerabilities and Exposures (CVE) is a list of identifiers for publicly known vulnerabilities.

[2] National Vulnerability Database (NVD) is United States governments repository of standards-based vulnerability management data.

Table 6
Studies by evaluation technique.

Technique	Papers	Metric count	Metric %
Industry Case Study	P11, P14, P16, P17, P25, P28, P32, P34, P35, P36, P39,P44, P45, P46, P47, P48, P49, P53, P58, P62, P67, P68, P69, P71	116	29%
Academic Case Study	P1, P2, P3, P8, P15, P18, P22, P31, P42, P52, P54, P55, P60, P63, P64, P65, P67	102	25%
Not Described	P13, P23, P33, P38, P4, P5, P51, P56, P6, P70	79	20%
Theoretical	P10, P12, P13, P19, P20, P26, P27, P30, P37, P40, P41, P50, P59, P6, P66, P7	64	16%
Opinion	P24, P29, P43, P57, P9	35	9%
Reports from production	P61	6	1%

Table 7
Studies by life cycle phase.

Phase	Papers	Metric Count	Metric %
Requirements	P19, P26, P27, P41, P57, P64	19	6%
Design	P17, P18, P19, P30, P31, P65	19	6%
Implementation	P1, P5, P9, P10, P12, P13, P18, P19, P22, P25, P28, P36, P44, P46, P47, P53, P55, P58, P60, P61, P62, P63, P64, P65, P67, P68, P69, P70, P71	142	46%
Testing	P19, P43, P48	4	1%
Operations	P2, P3, P4, P6, P7, P8, P11, P13, P14, P15, P16, P17, P18, P19, P22, P24, P25, P26, P27, P28, P29, P32, P33, P34, P35, P36, P37, P38, P39, P41, P42, P43, P44, P45, P49, P50, P51, P54, P56, P58, P59, P61, P62, P64, P65, P66, P67, P68, P69, P70	147	48%
All	P20, P23, P40, P52	20	7%

vulnerability count varies, and the availability of both varies, we recommend examining a given paper's definitions of vulnerability and vulnerability count, particularly when comparing between papers.

We observe that while vulnerability count is the most-used metric, there is no standard definition or data collection procedure. We advise caution in comparing results across papers.

6.1.2. Many metrics, few reused

Most (85%) security metrics have been proposed and evaluated solely by their authors. Less than 15% of metrics focus on requirements and design, and less than 2% focus on testing. In our data, 40% of the metrics are not empirically evaluated. There are artifacts and processes that remain unmeasured. We consider two potential reasons for why no baseline set of metrics is currently available:

- The trade-offs between confidentiality, integrity, and availability for, say, a social networking site versus a nuclear reactor control program, may lead teams to emphasis on different aspects of quality control during development, leading to different measures used between projects.
- Researchers focus on examining a subject in detail while development teams have to consider all development phases and artifacts. Identifying the gaps in researcher coverage of topics, as is addressed in this work, supports filling the gaps.

Potential research directions for security metrics would be for researchers to follow through on the evaluation and use of proposed metrics, to fill in the gaps where development artifacts and phases are not measured, and to characterize how software development projects choose security metrics.

As seen by the broad range of metrics and the narrowness of those reported in more than one paper/by one set of authors, the field of software life cycle security metrics has not settled on a baseline for measurement.

6.1.3. Most security metrics are subjective

The primary data sources for the metrics we survey are the subjective judgments of users, developers, managers, reviewers, and security researchers. In the metrics we survey, the two fundamental measurement operations are deciding on the assets to be protected, and deciding whether some behavior or outcome of the system is a violation of a security property, for example a vulnerability (see Section 6.1.1). Counting vulnerabilities depends on the counter's notion of some security property being violated. Even automated vulnerability counts (e.g., Paper P28) depend on the judgment of the implementers of the analysis tools used to raise warnings. Paper P1 developed a package of object-oriented security metrics founded on qualitative judgments about whether class and instance attributes are classified or critical. Paper P33 relies on qualitative assessments of security requirement and security mechanism criticality and strength. Attempts, such as those of paper P22, to define metrics measuring how well code or components supports a security principle, e.g., 'Least Privilege' are a valuable contribution because they define how they ground the measured security principle and enumerate the set of subjective decisions required to measure the principle.

We see a need for developing and evaluating standards for assessing security needs at every point in the development life cycle, analogous to Secure Coding Standards, e.g., Secure Requirements Standards, Secure Design Standards, and Secure Data Classification Standards. Where such standards exist, for example the Payment Card Industry Data Security Standards [29], we see opportunities for evaluation of the standard's effectiveness, development of test suites and fuzzers for thorough validation of software's compliance with the standards, and development of analogous standards for other types of data, e.g. social network graphs and password manager databases.

Judgments made by users, developers, managers, reviewers, and security researchers about what constitutes an asset, and what constitutes a vulnerability are currently the primary source of security-related measurements in software development.

Table 8
Metrics referenced in more than one paper and evaluated in an industrial context.

Metric Name	Metric Id	Paper Id	Metric category	Phase
Alert Count	M342	P44, P47	Implementation Metrics	Implementation
Commit Count	M522	P60, P68	Implementation Metrics	Implementation
CVSS Metrics	M518	P64, P67, P70, P13	Defensibility Metrics	Operations
Defect Count	M340	P19, P44	Project Management Metric	Implementation
Developer Count	M414	P25, P53, P60, P71	Resource Metrics	Implementation
Source Lines of Code (SLOC)	M295	P18, P28, P36, P44, P46, P47, P53, P58, P69	Implementation Metrics	Implementation
Vulnerability Density	M227	P28, P36, P58	Implementation Metrics	Implementation
Attack Effort	M292	P22, P29, P34, P37, P50	Resource Metrics	Operations
Attack Risk	M291	P22, P29, P34, P56	Resource Metrics	Operations
Attack Surface Metric	M289	P26, P34, P35	Defensibility Metrics	Operations
Attack Value	M290	P22, P34, P48, P43, P24	Resource Metrics	Operations
Security Resource Indicator	M231	P28, P58	Aggregate Metric	Operations
Vulnerability Count	M100	P14,P18, P25, P36, P44, P45, P46, P53, P58, P60, P61, P62, P68, P69, P71	Severity Metrics	Operations

6.1.4. Confidentiality and privacy are distinct, but overlap

Privacy is usually defined in terms of the rights of individuals, e.g. “Privacy requires that an individual has defined control over how his/her information will be disclosed.” [P23]. Confidentiality is usually defined in terms of access to data, e.g. “Confidential data access or confidential data transmission requires that unauthorized disclosure of one or more specific data items will not occur.”, [P23]. We excluded a set of papers based on their use of the term privacy. However, three selected papers, P2, P23, and P33, used the term in addition to confidentiality. Within the papers selected, the term ‘privacy’ was referenced. The paper author’s usage of the term ‘privacy’ overlapped with confidentiality in two of the three papers where it was used. P2 combined the terms into a Confidentiality-Privacy category. P23 defined an explicit and distinct set of privacy metrics in addition to defining a set of Confidentiality metrics. P33 named a set of privacy metrics, but the metrics measure whether encryption is used, the security of data transmission and storage, and login management, concerns that overlap with Confidentiality.

Confidentiality is typically defined in terms of software and operator control over access to data. Privacy introduces the notion of individual user control over data disclosure, which overlaps with confidentiality in two of the papers we evaluated.

6.2. What to measure?

Given the quantity and variety of metrics, one might ask where to begin when measuring security in software development. Lennon [30] suggests that security controls be implemented according to stakeholder priorities, with no more than 10–20 metrics in place to support manageability of the measurement program. We cannot speak directly to stakeholder priorities and goals, as they will vary, but we can illustrate an application of our collected metrics and their attributes. To illustrate the use of the data, and to set forth a baseline set of metrics, we select an example set of metrics drawn from our collected metrics, where the priorities are based on two criteria; the metric’s presence in two or more of our selected papers, and whether the metric’s evaluation was based on industry data. We exclude source code metrics, per our discussion in Section 5.1.2, with the exception of SLOC (M295), which is required to calculate vulnerability density (M227).

Table 8 presents the industry-evaluated metrics referenced in two or more papers from our results, organized by software development phase.

Our example set can be used as a baseline for discussion and adaptation to a given project, but the metrics are insufficient for measuring across the software development life cycle. Notably, the metrics in Table 8 apply only to the implementation and operations phases, as these phases have received the most attention by researchers in the literature we examine. To measure security concerns during the requirements phase, we recommend considering the metrics presented in papers P41 and P64. To measure security concerns during the Design phase, we recommend considering the metrics presented in paper P31. To measure security concerns during the testing phase, we recommend considering the metrics presented in papers P19, P43, and P48.

Selection of security metrics for a given project is currently a matter of stakeholder preference.

Given the size of the reduction produced by filtering out un-validated or single-author team metrics, we emphasize two observations:

- The set of validated and reused security metrics is much smaller than the complete set of metrics identified in our literature review. We caution that a given metric listed in the review be evaluated in the context of its intended use before being considered reliable, or, indeed, measurable.
- The set of validated and reused security metrics does not form a comprehensive set of measurements for the software development life cycle. Further researcher and practitioner effort will be required to develop a comprehensive set of SDLC measurements.

6.3. Fundamental security questions, revisited

We now return to the example ‘fundamental’ questions from the introduction, and discuss the application of the metrics we have collected to answering the questions.

6.3.1. Have we considered security in each stage of the software life cycle?

Metrics are available for each development phase we identified (Section 5.4 and Table 7). However, the requirements, design, and testing phases have relatively few metrics.

6.3.2. What are the security risks to which our software is exposed?

Resource Metrics (Section 5.1.5) can be used by the development team and other stakeholders to assign economic values to the risks involved in the use of the software. Defensibility Metrics (Section 5.1.6) can be used to assess the level of risk inherent to the design and implementation of the software. Incident metrics (Section 5.1.4) can be used as a direct measure of the risks that have occurred during the use of the software.

6.3.3. What are the security assurances we can give to users of our software?

The Security Properties Metrics (Section 5.1.1) can be used to characterize how security properties are implemented in software. The project management metrics (Section 5.1.3) can be used by the development team and other stakeholders to assess the development process for attention to quality. At present, there is no set of validated metrics that measures all aspects of the software development life cycle, so guarantees based on metrics cannot yet yield full assurance of security in the software produced by software development teams.

7. Limitations

If we have seeded our Quasi-Gold-Standard (QGS) with the wrong papers, we may have excluded relevant papers from our results. Likewise, size of QGS list may have excluded relevant papers from our results. We drew our results from three search engines, ACM, IEEE, and Elsevier, limiting our selection of papers to what is available in their indexes. Our QGS scores were low for Elsevier, suggesting that we may have missed relevant papers.

While we attempted to be comprehensive in our search strings and result parsing, our approach may have missed papers. Limiting our search to the scholarly literature excluded existing standards as well as industry experience reports disseminated by other means.

Software development organizations may choose not to report whether they are using metrics, limiting our observations to discussion of the scholarly literature.

Our metric classification scheme reflects our own biases in the data elements and values selected for each metric. We mitigated this bias by drawing on previous work where applicable (e.g. Savola [20], the IEEE glossary [9], and Fenton and Pfleeger [13]). Given the scheme, the choices made by individual raters are also subjective. We attempted to reduce bias by applying our two rater scheme.

Drawing inferences from the fields we classified depends on how accurately our choices match objective reality. Data elements we synthesized (Category, Measured Subject) are especially subject to this limitation, though we had two extractors check each metric-category assignment. We did not attempt a second approach, or a second set of extractors, to compare results, so our measures of validity need confirmation through replication.

8. Conclusion

Our systematic mapping study identified 324 unique metrics for measurement of security-related elements of and concerns in the software development life cycle. We developed a set of categories and subcategories for the metrics that may offer guidance for metric selection and a basis for metrics discussions. We observe that many metrics are focused on the characteristics of vulnerabilities in running software, and that relatively fewer metrics are available for measuring vulnerability prevention during development phases other than Implementation and Operations.

At present, security is measured primarily in its absence, as signified by the proportion of metrics dedicated to describing and counting vulnerabilities in software. As a consequence of the focus on vulnerabilities, most metrics are measured during the Implementation and Operations phases, once the faults causing the vulnerabilities are embedded in software. We see a need for further research into how to measure the presence of security, and how to identify fault introduction opportunities earlier in the development process, e.g. Requirements and Design, and in Testing, which can be conducted throughout the development process.

Even the most-used, most cited security metric, Vulnerability Count, must be carefully examined for the means used to measure it, in terms of what constitutes a vulnerability, and what the inclusion and exclusion rules are for including a vulnerability in the count. The CVSS standard is a step in the direction of standardization, but leaves a great deal of room for variation because rater judgment drives scoring. If vulnerability counts are shared between stakeholders, all participating stakeholders should be familiar with the definitions used.

Our database of collected metrics can be used as a guide to research needs in the field of software life cycle security metrics.

Acknowledgments

This material is based upon work supported with funding from the NSA's Science of Security Lablet at NCSU. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any entity of the United States Government. Thanks to the Realsearch research group for the helpful feedback.

Appendix

Table A1
Selected papers (* indicates QGS paper)

Paper Id	Paper
P1	Alshammari, Bandar; Fidge, Colin; Corney, Diane, A Hierarchical Security Assessment Model for Object-Oriented Programs, 2011
P2	Gonzalez, R.M.; Martin, M.V.; Munoz-Arteaga, J.; Alvarez-Rodriguez, F.; Garcia-Ruiz, M.A., A measurement model for secure and usable e-commerce websites, 2009
P3	Pham, N.; Baud, L.; Bellot, P.; Riguidel, M., A near real-time system for security assurance assessment, 2008
P4	Hajdarevic, K.; Allen, P., A new method for the identification of proactive information security management system metrics, 2013
P5	Xueqi Cheng; Nannan He; Hsiao, M.S., A New Security Sensitivity Measurement for Software Variables, 2008
P6	Yanguo Liu; Traore, I.; Hoole, A.M., A Service-Oriented Framework for Quantitative Security Analysis of Software Architectures, 2008
P7	Marconato, G.V.; KaÃhe, M.; Nicomette, V., A vulnerability life cycle-based security modeling and evaluation approach. 2012
P8	Scarfone, Karen; Mell, Peter, An analysis of CVSS version 2 vulnerability scoring, 2009
P9*	Sen-Tarng Lai, An Analyzer-Based Software Security Measurement Model for Enhancing Software System Security, 2010
P10	Wang, Lingyu; Islam, Tania; Long, Tao; Singhal, Anoop; Jajodia, Sushil, An Attack Graph-Based Probabilistic Security Metric, 2008
P11*	Manadhata, P.K.; Wing, J.M., An Attack Surface Metric, 2011
P12	Agrawal, Alka; Chandra, Shalini; Khan, Raees Ahmad, An efficient measurement of object oriented design vulnerability, 2009
P13	Wang, Ruyi; Gao, Ling; Sun, Qian; Sun, Deheng, An Improved CVSS-based Vulnerability Scoring Mechanism, 2011

(*continued on next page*)

Table A1 (*continued*)

Paper Id	Paper
P14	Wright, Jason L.; McQueen, Miles; Wellman, Lawrence, Analyses of Two End-User Software Vulnerability Exposure Metrics, 2012
P15*	Ouedraogo, Moussa; Khadraoui, Djamel; Mouratidis, Haralambos; Dubois, Eric, Appraisal and reporting of security assurance at operational systems level, 2012
P16	Sharma, Vibhu Saujanya; Trivedi, Kishor S., Architecture Based Analysis of Performance, Reliability and Security of Software Systems, 2005
P17*	Almorsy, M.; Grundy, J.; Ibrahim, A.S., Automated software architecture security risk analysis using formalized signatures, 2013
P18	Chowdhury, I.; Zulkernine, M., March. Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?, 2010
P19	Sultan, K.; En-Nouaary, A; Hamou-Lhadj, A, Catalog of Metrics for Assessing Security Risks of Software throughout the Software Development Life Cycle, 2008
P20	Kanzaki, Y.; Igaki, H.; Nakamura, M.; Monden, A.; Matsumoto, K.I., January. Characterizing dynamics of information leakage in security-sensitive software process., 2005
P21	Mell, P.; Scarfone, K.; Romanosky, S., Common Vulnerability Scoring System, 2006
P22	Yanguo Liu; Traore, I., Complexity Measures for Secure Service-Oriented Software Architectures, 2007
P23	Walton, G.H.; Longstaff, T.A.; Linger, R.C., Computational Evaluation of Software Security Attributes, 2009
P24	Aissa, A.B.; Abercrombie, R.K.; Sheldon, F.T.; Mili, A., Defining and computing a value based cyber-security measure., 2012
P25*	Yonghee Shin; Meneely, A.; Williams, L.; Osborne, J.A., Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities, 2011
P26	Krautsevich L.; Martinelli F.; Yautsiukhin A., Formal approach to security metrics.: What does more secure mean for you?, 2010
P27	Michael, J.B.; Shing, Man-Tak; Cruickshank, K.J.; Redmond, P.J., Hazard Analysis and Validation Metrics Framework for System of Systems Software Safety, 2010
P28	Walden J.; Doyle M.; Lenhof R.; Murray J., Idea: java vs. PHP: security implications of language choice for web applications., 2010
P29	Vijayaraghavan, V.; Paul, S., iMeasure Security (iMS): A Novel Framework for Security Quantification, 2009
P30	Khan, S.A.; Khan, R.A., Integrity quantification model for object oriented design., 2012
P31	Liu, M.Y.; Traore, I.,Measurement Framework for Software Privilege Protection Based on User Interaction Analysis, 2005
P32	Hasle, H\a agen; Kristiansen, Yngve; Kintel, Ketil; Snekkenes, Einar, Measuring Resistance to Social Engineering, 2005
P33	Islam, S.; Falcarin, P., Measuring security requirements for software security, 2011
P34	Manadhata, Pratyusa; Wing, Jeannette; Flynn, Mark; McQueen, Miles, Measuring the Attack Surfaces of Two FTP Daemons, 2006
P35	Buyens, Koen; Scandariato, Riccardo; Joosen, Wouter, Measuring the interplay of security principles in software architectures, 2009
P36	Alhazmi, O. H.; Malaiya, Y. K.; Ray, I., Measuring, analyzing and predicting security vulnerabilities in software systems, 2007
P37	Huaijun Wang; Dingyi Fang; Ni Wang; Zhanyong Tang; Feng Chen; Yuanxiang Gu, Method to Evaluate Software Protection Based on Attack Modeling, 2013
P38	Villarrubia, C.; FernÃ!'ndez-Medina, E.; Piattini, M., Metrics of password management policy., 2006
P39	Shar, Lwin Khin; Hee Beng Kuan Tan, Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities, 2012
P40	LeMay, E.; Ford, M.D.; Keefe, K.; Sanders, W.H.; Muehrcke, C., Model-based Security Metrics Using ADversary VIew Security Evaluation (ADVISE), 2011
P41	Gallon, Laurent, On the Impact of Environmental Metrics on CVSS Scores, 2010
P42	Schryen, G.; Kadura, R., Open source vs. closed source software: towards measuring security., 2009
P43	BÃ¶hme, R.; FÃ©legyhÃ!'zi, M.,, Optimal information security investment with penetration testing., 2010
P44	Gegick, M.; Rotella, P.; Williams, L., Predicting Attack-prone Components, 2009
P45	Shar, Lwin Khin; Tan, Hee Beng Kuan, Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns, 2013
P46	Nguyen, Viet Hung; Tran, Le Minh Sang, Predicting Vulnerable Software Components with Dependency Graphs, 2010
P47	Gegick, Michael; Williams, Laurie; Osborne, Jason; Vouk, Mladen, Prioritizing Software Security Fortification Through code-level Metrics, 2008
P48	Khan, M.U.A.; Zulkernine, M., Quantifying Security in Secure Software Development Phases, 2008
P49	Sharma, Vibhu Saujanya; Trivedi, Kishor S., Quantifying software performance, reliability and security: An architecture-based approach, 2007
P50	Arnold, F.; Pieters, W.; Stoelinga, M., Quantitative penetration testing with item response theory, 2013
P51	Grunske, Lars; Joyce, David, Quantitative risk-based security prediction for component-based systems with explicitly modeled attack profiles, 2008
P52	Li, Minglu; Li, Jianping; Song, Hao; Wu, Dengsheng, Risk Management in the Trustworthy Software Process: A Novel Risk and Trustworthiness Measurement Model Framework, 2009
P53	Zimmermann, Thomas; Nagappan, Nachiappan; Williams, Laurie, Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista, 2010
P54*	Wang, Ju An; Wang, Hao; Guo, Minzhe; Xia, Min, Security Metrics for Software Systems, 2009
P55	Chowdhury, Istehad; Chan, Brian; Zulkernine, Mohammad, Security Metrics for Source Code Structures, 2008
P56	TrÄ, D., Security Metrics Foundations for Computer Security., 2009
P57	Abdulrazeg, A.A.; Norwawi, N.M.; Basir, N., Security metrics to improve misuse case model, 2012
P58	Walden, J.; Doyle, M.; Welch, G.A.; Whelan, M., Security of open source web applications, 2009
P59	Demme, J.; Martin, R.; Waksman, A.; Sethumadhavan, S., Side-channel vulnerability factor: A metric for measuring information leakage, 2012
P60	Meneely, A.; Williams, L., Strengthening the empirical analysis of the relationship between Linus' Law and software security., 2010
P61*	Davidson, M.A., The Good, the Bad, And the Ugly: Stepping on the Security Scale, 2009
P62	Gegick, Michael; Rotella, Pete; Williams, Laurie, Toward Non-security Failures As a Predictor of Security Faults and Failures, 2009
P63	Neto, A.A.; Vieira, M., Trustworthiness Benchmarking of Web Applications Using Static Code Analysis, 2011
P64	Younis, A.A.; Malaiya, Y.K.; Ray, I., Using Attack Surface Entry Points and Reachability Analysis to Assess the Risk of Software Vulnerability Exploitability, 2014
P66	Beres, Y.; Mont, Marco Casassa; Griffin, J.; Shiu, S., Using security metrics coupled with predictive modeling and simulation to assess security processes, 2009
P67	Younis, A.A.; Malaiya, Y.K., Using Software Structure to Predict Vulnerability Exploitation Potential, 2014
P68	Perl, H.; Dechand, S.; Smith, M.; Arp, D.; Yamaguchi, F.; Rieck, K.; Fahl, S.; Acar, Y.,VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits.,2015
P69*	Jinyoo Kim; Malaiya, Y.K.; Indrakshi Ray, Vulnerability Discovery in Multi-Version Software Systems, 2007
P70	Scarfone, K.; Mell, P., Vulnerability scoring for security configuration settings., 2008
P71	Meneely, A.; Srinivasan, H.; Musa, A.; Rodriguez Tejeda, A.; Mokary, M.; Spates, B., When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing Commits, 2013

References

[1] G. McGraw, Software Security: Building Security in, Addison-Wesley Professional, 2006.

[2] Four Grand Challenges in Trustworthy Computing, 2003. http://cra.org/uploads/documents/resources/rissues/trustworthy.computing_.pdf.

[3] S. Pfleeger, R. Cunningham, Why measuring security is hard, IEEE Secur. Priv. 8 (4) (2010) 46–54.

[4] M. Howard, S. Lipner, The Security Development Lifecycle, 8 Microsoft Press Redmond, 2006.

[5] SafeCode Initiative, Accessed: Jan 2018. http://www.safecode.org/.

[6] Cigital: Building Security In Maturity Model. Accessed: Jan 2018.

[7] D. Budgen, M. Turner, P. Brereton, B. Kitchenham, Using mapping studies in software engineering, Proceedings of Psychology of Programming Interest Group (PPIG), 8 Lancaster University, 2008, pp. 195–204.

[8] Project Website, Accessed: January 2018. https://sites.google.com/view/smssecmetrics/home.

[9] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990 (1990) 1–84, http://dx.doi.org/10.1109/IEEESTD.1990.101064.

[10] R. Shirey, Internet Security Glossary, Version 2, IETF, 2007. No. 4949 in Request for Comments. Published: RFC 4949 (Informational). http://www.ietf.org/rfc/rfc4949.txt .
[11] M. Rudolph, R. Schwarz, A critical survey of security indicator approaches, Proceedings of the 7th International Conference on Availability, Reliability and Security, (2012), pp. 291–300.
[12] W. Jansen, Directions in Security Metrics Research. http://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7564.pdf.
[13] N.E. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, 2nd ed., PWS Publishing Co., Boston, MA, USA, 1998.
[14] Std ISO IEC 16085 - 2006 - ISO/IEC 16085:2006, Standard for Software Engineering - Software Life Cycle Processes - Risk Management, Accessed: January 2018. https://standards.ieee.org/findstds/standard/16085-2006.html.
[15] N. Pham, L. Baud, P. Bellot, M. Riguidel, A near real-time system for security assurance assessment, Proceedings of the 3rd International Conference on Internet Monitoring and Protection, ICIMP '08, IEEE Computer Society, Washington, DC, USA, 2008, p. 152160.
[16] N.F. Schneidewind, Methodology for validating software metrics, IEEE Trans. Software Eng. 18 (5) (1992) 410–422.
[17] M. Ouedraogo, D. Khadraoui, H. Mouratidis, E. Dubois, Appraisal and reporting of security assurance at operational systems level, J. Syst. Softw. 85 (1) (2012) 193–208.
[18] V. Verendel, Quantified security is a weak hypothesis: a critical survey of results and assumptions, Proceedings of the 2009 Workshop on New Security Paradigms, ACM, 2009, pp. 37–50.
[19] A. Meneely, B. Smith, L. Williams, Validating software metrics: a spectrum of philosophies, ACM Trans. Softw. Eng.Methodol. 21 (4) (2012) 24.
[20] R.M. Savola, Quality of security metrics and measurements, Comput. Secur. 37 (2013) 78–90.
[21] H. Zhang, M.A. Babar, P. Tell, Identifying relevant studies in software engineering, Inf. Softw. Technol. 53 (6) (2011) 625–637.
[22] W. Hudson, Card sorting, in: M. Soegaard, R.F. Dam (Eds.), The Encyclopedia of Human-Computer Interaction, The Interaction Design Foundation, 2012.
[23] P. Mell, K. Scarfone, S. Romanosky, A Complete Guide to the Common Vulnerability Scoring System Version 2.0, FIRST-Forum of Incident Response and Security Teams, 2007, pp. 1–23. Published by
[24] I.V. Krsul, Software Vulnerability Analysis, Purdue University, 1998.
[25] C.P. Pfleeger, S.L. Pfleeger, Security in Computing, Prentice Hall Professional Technical Reference, 2002.
[26] Responding to Computer Security Incidents.(1990).
[27] Common Vulnerability Scoring System v3.0: Specification Document, Accessed: January 2018. https://www.first.org/cvss/specification-document.
[28] Attack Surface Analysis Cheat Sheet. Accessed: January 2018. https://www.owasp.org/index.php?title=Attack_Surface_Analysis_Cheat_Sheet&oldid=156006.
[29] PCI Security Standards Council. Accessed: January 2018. https://www.pcisecuritystandards.org/pcisecurity/.
[30] E.E. Lennon, It Security Metrics. ITL Bulletin, http://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7564.pdf.