ABSTRACT

KING, JASON TYLER. Measuring the Forensic-ability of User Activity Logs. (Under the direction of Dr. Laurie Williams.)

According to a 2011 survey in healthcare, the most commonly reported breaches of protected health information involved employees snooping into medical records of others, including fellow employees, friends, relatives, celebrities, and other high-profile individuals. User accountability in software systems is important for mitigating repudiation threats and detecting security or privacy breaches. Logging mechanisms that capture detailed traces of user activity, including creating, reading, updating, and deleting (CRUD) data, facilitate meaningful forensic analysis following a security or privacy breach. However, software requirements often inadequately and inconsistently state “what” user actions should be logged, thus hindering forensic analysis.

Existing guidelines and specifications for logging mechanisms are too abstract (such as “log all data creations, views, updates, and deletions”) in outlining transactions that should trigger an event to be logged. Software engineers often incompletely implement logging mechanisms, causing gaps in traces of user activity. The objective of this research is to help software developers improve the forensic-ability of user activity logs by developing heuristics to support identifying mandatory log events; and measuring the degree to which user activity logs capture the mandatory log events. We develop a systematic heuristics-driven process for identifying log events that must be logged based on user actions described in natural language software artifacts. We then develop a systematic process for creating a black-box test suite for verifying the identified log events are logged. Using the results of executing the
black-box test suite, we propose and evaluate a security metric for measuring the forensic-ability of user activity logs.

To evaluate our process for identifying log events, we conducted a controlled experiment with 103 computer science students enrolled in a graduate-level software security course. All subjects were first asked to identify log events described in a set of requirements statements during the pre-period task. In the post-period task, subjects were randomly assigned statements from one type of software artifact (traditional requirements, use-case-based requirements, or user manual), one readability score (simple or complex statements), and one method for identifying log events (standards-, resource-, or heuristics-driven). We evaluated subject performance using three metrics: statement classification correctness (values from 0 to 1), log event identification correctness (values from 0 to 1), and response time (seconds).

Overall, log event identification correctness was low for all groups in our experiment. Given that subjects were required to read and understand descriptions of the methods for identifying log events on their own and in a relatively short period of time, subjects may have misunderstood or applied the methods incorrectly. According to our results, the heuristics-driven process did not significantly help subjects more correctly identify log events. Instead, subject performance at classifying statements that do not contain log events improved during the post-period task. The data suggest that the increase in correct identifications of statements that do not contain log events may be due to inflated guessing of many statements as not describing any log events.

To measure forensic-ability, we first systematically documented a black-box test case for each of the identified log events for a software system. We calculated forensic-ability as
the percentage of passing log event test cases. We performed case studies on two open-source software systems and one proprietary electronic health record system. For the open-source iTrust Electronic Health Record system, we calculated forensic-ability as 0.58 (58% of iTrust log event test cases pass). For the open-source Open Conference System (OCS), we calculated forensic-ability as 0.09 (9% of OCS log event test cases pass). For the ProprietaryEHR (PropEHR) software system, we calculate forensic-ability as 0.36 (36% of PropEHR log event test cases pass). To evaluate our metric, we demonstrated the actionability, appropriate continuity, non-uniformity, product/process relevance, and usability of our forensic-ability metric.
Measuring the Forensic-ability of User Activity Logs

by

Jason Tyler King

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2016

APPROVED BY:

Anne McLaughlin, Ph.D.  Emerson Murphy-Hill, Ph.D.

Munindar Singh, Ph.D.  Laurie Williams, Ph.D.
Committee Chair
BIOGRAPHY

Jason Tyler King was born in Rocky Mount, North Carolina, and raised on a family farm in rural Franklin County, North Carolina. He graduated from Louisburg High School in 2005. In both middle and high school, he served as school webmaster and designed the Franklin County school system website, logo, and motto. In 2009, Jason graduated summa cum laude from North Carolina State University as a Valedictorian with a Bachelor of Science degree in Computer Science. Enrolling in graduate school at North Carolina State University directly after earning his Bachelor of Science degree, Jason served as a teaching assistant for undergraduate software engineering courses, the undergraduate senior design capstone project course, and a graduate-level software security course. He received a Master of Science degree in Computer Science in 2011, and has served as an instructor for an undergraduate Java programming course while pursuing a Doctor of Philosophy degree in Computer Science.
ACKNOWLEDGEMENTS

A dissertation requires many years of hard work but would never be possible without unyielding support from many individuals.

First, I thank my advisor, Laurie Williams, for her support, patience, and guidance. I am the first generation in my family to attend college, so Laurie has always been very helpful in guiding not just me, but all her students. I also thank my committee members for their support and feedback: Anne McLaughlin (whose cognitive science course was one of the most engaging courses of my undergraduate career); Emerson Murphy-Hill; and Munindar Singh.

I also thank the entire Realsearch research group for support over the years, specifically: Patrick Francis, Eric Helms, Andy Meneely, Patrick Morrison, Rahul Pandita, Maria Riaz, John Slankas, Ben Smith, and Chris Theisen.

Finally, I think the funding agencies that have invested into not just my research, but the future of computer science: the National Collaborative for Bio-Preparedness project funded by the United States Department of Homeland Security; and the United States National Security Agency (NSA) Science of Security Lablet. Any opinions expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSA.
# TABLE OF CONTENTS

LIST OF TABLES .................................................................................................................. viii
LIST OF FIGURES ................................................................................................................ ix

1. Introduction ......................................................................................................................... 1
   1.1 Approach .................................................................................................................. 2
   1.2 Research Questions and Studies Performed .............................................................. 2
   1.3 Contributions ......................................................................................................... 4
   1.4 Outline .................................................................................................................... 5

2. Background ......................................................................................................................... 6

3. Related Work ...................................................................................................................... 8
   3.1 Nonrepudiation ....................................................................................................... 8
   3.2 Auditable Events .................................................................................................... 11
   3.3 Log Entry Content ................................................................................................. 13
   3.4 Immutability .......................................................................................................... 15
   3.5 Log Monitoring ..................................................................................................... 16

4. Systematic Mapping Study .............................................................................................. 19
   4.1 Introduction ......................................................................................................... 19
   4.2 Methodology ......................................................................................................... 20
      4.2.1 Identifying Venues and Databases ................................................................. 22
      4.2.2 Establishing a Quasi-gold Standard ............................................................... 23
      4.2.3 Eliciting Search Terms ................................................................................... 27
      4.2.4 Conducting Automated Searches ................................................................. 28
      4.2.5 Evaluate Search Performance ....................................................................... 29
      4.2.6 Including/Excluding Publications ................................................................. 30
      4.2.7 Data Extraction .............................................................................................. 34
   4.3 Results ...................................................................................................................... 35
   4.4 Discussion ................................................................................................................. 41
      4.4.1 Evaluating Logging Mechanisms ................................................................... 41
      4.4.2 Frameworks for Designing & Implementing Logging Mechanisms ............... 42
      4.4.3 Evaluation Protocols for Research on Logging Mechanisms ....................... 43
      4.4.4 Log Events Suggested by Existing Research ................................................. 44
      4.4.5 Open Research Areas on Logging Mechanisms ........................................... 45
   4.5 Threats to Validity ..................................................................................................... 46
      4.5.1 Selection of Search Terms and Databases ....................................................... 47
      4.5.2 Selection of Studies ......................................................................................... 47
      4.5.3 Reliability of Data Extraction ......................................................................... 47
   4.6 Conclusion .................................................................................................................. 48

5. Identifying MLEs in Natural Language Software Artifacts ............................................... 49
   5.1 Methodology ............................................................................................................ 51
      5.1.1 Step One: Selecting Software & Software Artifacts ....................................... 51
      5.1.2 Step Two: Preprocessing of Natural-Language Software Artifacts ............... 53
      5.1.3 Step Three: Extracting Verb-Object Pairs ....................................................... 53
7.7 Principles for Aiding Consistent Measurement of Forensic-ability .................. 151
7.8 Threats to Validity .................................................................................. 155
  7.8.1 Internal Validity .................................................................................. 155
  7.8.2 External Validity .................................................................................. 156
  7.8.3 Construct Validity ................................................................................. 157
7.9 Conclusion .............................................................................................. 157
8. Conclusions .............................................................................................. 159
  8.1 Contributions .......................................................................................... 160
  8.2 Limitations .............................................................................................. 161
  8.3 Future Work ............................................................................................ 161
9. REFERENCES ............................................................................................ 163
LIST OF TABLES

Table 1: Summary of Venues ........................................................................................................ 23
Table 2: Summary of Relevant Venues and Collected Citations .................................................... 24
Table 3: Publications in our Quasi-gold Standard ........................................................................... 27
Table 4: Summary of term and phrase frequencies in the quasi-gold standard publication titles, abstracts, and keywords ................................................................................................................... 28
Table 5: Final search query string .................................................................................................. 29
Table 6: Summary of search results ................................................................................................. 30
Table 7: Final set of publications included in our study ................................................................. 34
Table 8: Summary of data extracted for analysis ............................................................................ 34
Table 9: Summary of excluded publications from the full text reviews round .............................. 35
Table 10: Summary of data extracted from each publication .......................................................... 37
Table 11: Summary of top 5 verbs (all verbs vs. mandatory log event verbs) ................................. 58
Table 12: Confusion matrix for iTrust classifications ...................................................................... 60
Table 13: Confusion matrix for iHRIS classifications ..................................................................... 60
Table 14: Confusion matrix for OCS classifications ........................................................................ 60
Table 15: Summary of extracted verb-object pairs ........................................................................ 61
Table 16: Summary of Metrics for Evaluating Subject Responses ................................................ 83
Table 17: Summary of Experimental Factors and Levels ................................................................. 88
Table 18: Summary of Software Artifact Statements for each Software Artifact Type and Readability .................................................................................................................................................. 91
Table 19: Summary of Methods and Instructions provided for each ............................................... 93
Table 20: Summary of the Number of Subjects in each Treatment Group ..................................... 106
Table 21: Type III Tests of Fixed Effects, including effects involving method and other potentially significant effects for statement classification correctness ........................................... 108
Table 22: Classification correctness averages for Task*OracleHasMLEs ..................................... 109
Table 23: Classification correctness averages for Task*ArtifactType .......................................... 109
Table 24: Classification correctness averages for Task*Readability ............................................ 110
Table 25: Type III Tests of Fixed Effects, including effects involving method and other potentially significant effects for statement MLE identification correctness .................................. 111
Table 26: Type III Tests of Fixed Effects, including Effects involving Method and Other Potentially Significant Effects for Response Time ........................................................................ 114
Table 27: Response Time Averages (in seconds) for Task*ArtifactType*Readability .................. 114
Table 28: Mandatory Log Event Test Results .................................................................................. 141
Table 29: Malicious User Scenario Results ..................................................................................... 144
LIST OF FIGURES

Figure 1: Process for identifying relevant publications, adapted from (Zhang et al. 2011) ... 21
Figure 2. Coverage of verb-object pairs by heuristics, sorted in increasing order by largest number of verb-object pairs covered. ................................................................. 74
Figure 3: Overview of Experimental Tasks .................................................................. 85
Figure 4: Sample statement page, with a countdown timer, options for selecting whether the statement describes any MLEs, and (if so), text fields for identifying up to 10 MLEs.................................................................................................................. 97
Figure 5: Statement classification correctness for each method of identifying MLEs ........ 108
Figure 6. Statement MLE identification correctness for subjects using each method for identifying MLEs ......................................................................................... 111
Figure 7: Plot of response times for subjects in each method......................................... 113
Figure 8: Sample BDD user story template, as presented by Dan North (North) .......... 127
Figure 9: Example Cucumber feature file template. The “[user]”, “[verb]”, and “[object]” text are passed to the step definition as parameters, allowing for code reuse in the step definitions. The “Then” step definition also contains code to check for a recent timestamp (see Section 7.2.3.1) ........................................................................ 133
1. Introduction

Software systems that manage sensitive data are often required to maintain logging mechanisms that capture all user transactions with sensitive data (2010a; 2011a) to promote user accountability and rebuild traces of user activity should a security or privacy breach occur. According to a 2011 Veriphyr Survey of Patient Privacy Breaches (2011b) in healthcare, the top three most commonly reported breaches of protected health information (PHI) involved snooping into medical records of employees (35%), snooping into medical records of friends and relatives (27%), and loss or theft of physical records (25%). In addition, 52% of survey participants indicated that their organization did not have adequate tools for monitoring inappropriate access to PHI. Logging mechanisms should capture all transactions with PHI so that meaningful auditing of log entries can both proactively identify unauthorized PHI access on a continuous basis, as well asreactively provide a means for forensic analysis.

In software security, logging mechanisms provide a means of nonrepudiation to help answer questions regarding who, what, when, where, why, and how a security breach occurred. Logging mechanisms also help mitigate repudiation threats, threats associated with users who deny performing some action within the software system without other parties having any way to prove otherwise (Hernan et al. 2006; 2011a). Adequate logging mechanisms help strengthen nonrepudiation, the ability of the software system to mitigate repudiation threats, and strengthen user accountability.

However, software logging mechanisms remain vulnerable to repudiation threats by failing to record complete traces of user activity (King et al. 2012). Specific logging
requirements and mandatory log events for software systems may not be explicitly stated but are often implied by existing functional requirements specifications (Riaz et al. 2014). For our work, we define a mandatory log event (MLE) as a user interaction with a system resource that must be logged to hold the user accountable for performing the action.

The objective of this research is to help software developers improve the forensic-ability of user activity logs by developing heuristics to support identifying mandatory log events; and measuring the degree to which user activity logs capture the mandatory log events.

1.1 Approach

To address our research objective, we develop a systematic process for extracting a set of MLEs from functional requirements specifications. With a set of MLEs, we then systematically develop and execute a black-box test suite for evaluating whether each of the events is acceptably logged by the logging mechanism. We then use the results of the black-box test plan execution as the basis for a security metric for evaluating the forensic-ability of a given logging mechanism. We define forensic-ability as the ability to gather and examine evidence to determine who, what, when, where, why, and how a security or privacy breach occurs. As part of our work, we demonstrate software metrics validation criteria for the practicality and meaningfulness of our forensic-ability metric.

1.2 Research Questions and Studies Performed

We consider the following research questions to guide our work:

- RQ1: What features of functional requirements indicate a user activity that should be logged?
- RQ2: How well does a guided process for identifying verb-resource pairs help developers manually extract loggable user activity implied by functional requirements, compared to an unguided process for extracting loggable user activity?

- RQ3: How well does a forensic-ability metric demonstrate software metrics validation criteria for practicality and meaningfulness?

To address our research questions, we performed the following studies:

1. **Identifying MLEs Study** (RQ1) (King et al. 2015a): Two researchers manually identify MLEs in natural language software artifacts for three open-source software systems: iTrust\(^1\), an electronic health record system; IntraHealth International human resources management (iHRIS) Page Builder module\(^2\); and Open Conference System (OCS) conference management software\(^3\). Based on disagreements about which events should be classified as MLEs, we empirically-derive a set of heuristics to guide software engineers when identifying events that must be logged for holding users accountable.

2. **Using Heuristics to Identify MLEs -- A Controlled Experiment** (RQ2): We perform a controlled experiment with graduate students in a graduate-level software security course as our subjects. In the experiment, we study the performance of our heuristics-driven process for identifying MLEs (King et al. 2015a) against two alternative processes for identifying MLEs.

\[^1\] http://agile.csc.ncsu.edu/iTrust
\[^2\] http://www.ihris.org/
\[^3\] https://pkp.sfu.ca/ocs/
3. **Measuring forensic-ability study** (RQ3): We systematically black-box test user activity logging mechanisms for five software systems: iTrust; iHRIS Page Builder module; OCS; a commercial electronic health record system\(^4\); and OpenMRS\(^5\), an open-source electronic health record system.

### 1.3 Contributions

In this dissertation, we make the following contributions:

- A metric for repeatable calculation of the forensic-ability of user activity logs, along with first steps toward validation of the metric;
- A set of empirically-derived heuristics to assist software engineers in determining whether a given user action described in a software artifact must be logged;
- A systematic methodology for generating black-box test cases to verify MLEs are correctly logged;
- An oracle of MLE classifications for three open-source software systems (the oracle is publicly available on the project website\(^6\));
- Materials for further replication and analysis of our controlled experiment on MLE identification;
- Lessons learned on how to improve both identification of MLEs; and
- A reusable set of Cucumber BDD feature files and step definitions to facilitate black-box evaluation of two open-source software systems.

---

\(^4\) The commercial electronic health record system vendor requested to remain confidential  
\(^5\) [http://openmrs.org/](http://openmrs.org/)  
\(^6\) [http://go.ncsu.edu/NLPLogging](http://go.ncsu.edu/NLPLogging)
1.4 Outline

The remainder of this paper is organized as follows: Chapter 2 presents background on software logging mechanisms. Chapter 3 presents related work on software logging mechanisms. Chapter 4 presents our systematic mapping study for user activity logging mechanisms. Chapter 5 presents the development of our heuristics-driven process for identifying MLEs. Chapter 6 presents our controlled experiment for evaluating subject performance in identifying MLEs. Chapter 7 presents our methodology for black-box evaluation of software user activity logs and measuring forensic-ability of user activity logs. Chapter 8 summarizes the results of our work and discusses future research areas.
2. Background

Logging mechanisms that capture comprehensive traces of user activity may deter users from performing unauthorized behavior and also provide an accounting of data disclosures for assisting data privacy and forensics investigations. Software systems that manage sensitive data, such as healthcare software that manages protected health information (PHI) or commercial software that manages customer financial information, are typically required to maintain logging mechanisms that capture traces of user activity for holding users accountable (Scholl et al. 2008; 2010a; 2011a). Logging mechanisms must address multiple technical and organizational policies to be considered secure. For example, the Payment Card Industry Data Security Standards (PCI-DSS) address ten principle requirements for secure logging for user accountability (2010a; King and Williams 2012):

- **Nonrepudiation** – link access of system components to each individual user
- **Auditable events** – define the set of loggable user activity that should be recorded
- **Log entry content** – define the set of data that should be captured as part of each log entry
- **Timestamp reliability** – ensure all system clocks are synchronized
- **Immutability** – ensure log files cannot be altered or tampered with
- **Log backups** – ensure log files are backed-up to a secure location
- **Log monitoring** – ensure logs are audited on a regular basis
- **Log retention** – ensure log files are saved or archived for an appropriate length of time
- **Log disposal** – ensure log files are securely deleted or destroyed
- **Incident response** – ensure procedures are in place to respond to breaches or other security incidents discovered through auditing of log files

The Payment Card Industry (PCI) is a mature industry that recognizes the importance of security, since breaches of payment card information may result in significant financial losses to both individuals and organizations. Other industries, such as the electronic healthcare software industry, are recently maturing (2010b) and do not fully address the same security issues recognized in the PCI.
3. Related Work

In this section, we discuss the related research. We organize the related research according to five of the PCI-DSS principle requirements for secure logging mechanisms (see Chapter 2) that have associated research.

3.1 Nonrepudiation

Vance, Lowry, and Egget (Vance et al. 2013) discuss the importance of identifiability, the belief that one’s actions within a group cannot be associated with him or her individually as they become immersed in a collective group. When individuals sense that they are distinguishable within a group, certain behaviors tend to be curtailed. The researchers highlight the concept of evaluation, in which a person’s performance is assessed by another party according to a set of rules with implied consequences. The researchers perform a study to investigate whether the awareness of logging of user behavior influenced a person’s behavior. The researchers suggest awareness of logging increases perceived accountability and decreases a person’s intention to violate an access policy.

Cruz-Correia et al. (Cruz-Correia et al. 2013) evaluated the use of audit trails in Portuguese electronic health record (EHR) systems. Overall, the study concludes that existing audit trails are very poor and fail to provide traceability of user activity. In addition, existing audit trails were deemed inadequate for analyzing usage of the EHR system. The researchers also identified inconsistencies in log files, such as cases where log entries indicated a user performed an activity after another log entry indicated the user had already logged-out of the system. Oh et al. (Oh et al. 2014) also discuss auditing in healthcare exchanges. Oh et al. propose a design for auditing healthcare exchanges and evaluate their
design using the Audit Trail and Node Authentication (ATNA) protocol in healthcare. Garg, Jia, and Datta (Garg et al. 2011) investigate the viability of auditing to determine HIPAA policy violations in healthcare software by using incomplete logs of user activity.

Dekker and Etalle (Dekker and Etalle 2007) propose a framework for audit-based access control of EHR systems based on the presence of a logging mechanism that captures user actions and events. Similarly, Malin, Nyemba, and Paulett (Malin et al. 2011) discuss a process for generating access control policies for EHR systems by mining usage patterns from EHR log files. Malin, Nyemba, and Paulette transform a log of access to protected health information into probabilistic access control rules.

Jagadeesan et al. present an operational model of accountability (Jagadeesan et al. 2009). The model incorporates five distinct conditions to model accountability: upper bounds (every guilty agent is blamed by an auditor), lower bounds (everyone blamed by the auditor is guilty), overlaps (at least one blamed agent is guilty), liveness (auditors always blame at least one agent), and blamelessness (honest agents have a strategy to avoid being pronounced as guilty by an auditor). Using the model, the authors motivate the need for audit logs in accountability systems to prevent a user from achieving “absence of provable guilt” in the form of a repudiation threat.

Mazza, Potet, and Métayer (Mazza et al. 2011) present a framework for formally specifying liability in a contractual setting. The researchers also propose a log analyzer to establish the validity of liability claims. In a separate study, the authors propose criteria for architectures for logging mechanisms (Le Métayer et al. 2010).
Richard, Roussev, and Marziale (Richard Golden G. et al. 2007) propose a Forensic Discovery Auditing Module (FDAM). FDAM intends to verify the use of tools by examiners during the manipulation of digital evidence. FDAM logs accesses to digital evidence at the operating system level.

Sundareswaran, Squicciarini, and Lin (Sundareswaran et al. 2012) describe a framework for tracking usage of user data in the cloud. The logger captures access to each instance or copy of a user’s data. The logger can also ensure access and usage control based on the data owner’s preferences. Cederquist et al. (Cederquist et al. 2005) also discuss a framework for auditing compliance with usage policies. Zhu et al. (Zhu et al. 2011) propose an audit service in cloud systems for efficient anomaly detection of untrusted storage services. Shah et al. (Shah et al. 2007) discuss internal and external auditing of third-party online storage services so that customers can evaluate risks associated with using the service.

Kathiresshan, Xiao, and Xiao (Kathiresshan et al. 2011) propose a framework for capturing and analyzing network packets for logging and reproducing the entire history of networking events in the application layer. All incoming and outgoing traffic within the application layer is logged and processed into human-readable log files. Bo and Yang (Bo and Yang 2012) present a similar framework that records events through nodes in a network for the purpose of detecting attacks. Bo and Yang also identify a current lack of systematic research on quantified accountable logging. Xiao (Xiao 2009) presents a flow-net approach for recording network traffic data for intrusion detection and forensics, since logging mechanisms are often incomplete or disorganized. Huang, Bavier, and Peterson (Huang et al. 2006) discuss the use of a network auditing service called PlanetFlow in processing network
traffic to resolve complaints, limit liability, and mitigate problematic behavior in the network. For example, the utility can be used to identify which service served illegal or copyrighted content to hosts in the network.

Seneviratne (Seneviratne 2012) proposes a web protocol called Accountable HTTP (HTTPA) that would preserve provenance of data transmitted to and from web servers and allow the derivation of an audit trail of data use. A log would be created for each client-server transaction to record what data was accessed, the reason for the access, and agreed-upon usage restrictions. Robinson, Cook, and Shrivastava (Robinson et al. 2005) discuss a framework for an irrefutable web service audit trail to promote accountability of business-to-business interactions over the Internet.

Overall, research has identified the need for user accountability and nonrepudiation in software systems that manage sensitive information. Many researchers have proposed frameworks and architectures to promote the accountability of users, but we have identified no existing empirical work related to identifying the set of user activity that should be logged for promoting user accountability and nonrepudiation.

### 3.2 Auditable Events

Cruz-Correia et al. identified that not all important events are recorded in healthcare software audit trails (Cruz-Correia et al. 2013), such as session timeouts. Additionally, the researchers observed semantic issues in multiple systems related to a single action being recorded with different descriptions of the user activity that occurred.

Yskout, Win, and Joosen (Yskout et al. 2008) discuss an approach for transforming security requirements for logging mechanisms into an architectural model using Unified
Modeling Language (UML) to aid developers in designing and implementing logging. The approach focuses on modeling events that should be logged. The model allows for the notion of audit criticality for a developer to identify whether an event is “minimal”, “basic”, “detailed”, or “unspecified”. The model also allows the specification of actions that denote success or failures. The architecture can model explicit log calls by inserting calls to a logging interface at the start of an operation and/or at the end of the operation. However, this research seems to only handle the modeling of explicit logging requirements, such as “The starting and stopping of the ATM machine needs to be audited.” In our research, we are interested in loggable user activity implied by existing functional requirements. Hoisl and Strembeck (Hoisl and Strembeck 2012) also present a UML extension for modeling the specification of audit rules for software logging mechanisms using model-driven-development.

Etalle and Winsborough (Etalle and Winsborough 2007) discuss their approach for A-Posteriori PoLicy Enforcement (APPLE) by using logs to verify that actions taken by users in software systems were authorized. The researchers focus on four main types of log entries:

- modification entries, logging the creation and modification of documents
- input entries, logging the receipt of a document
- output entries, logging the transmission of documents to other users
- policy entries, logging the policy rules that show actions were authorized

Mazza, Potet, and Métayer (Mazza et al. 2011) propose a formal framework for the use of logging as electronic evidence. In the study, the researchers define a log event as a tuple: <type of event, source and destination agents, action executed>. 
Existing research on auditable events is limited to providing lists of suggested types of actions to log within a given framework without much, if any, justification. So far, we have identified no empirical research on identifying events that should be logged.

### 3.3 Log Entry Content

Cruz-Correia et al. identify three categories of content that should be recorded for each log entry in healthcare software logging mechanisms: essential information, important information, and optional information (Cruz-Correia et al. 2013). Essential information includes the username, timestamp, patient identification, and identification of the data being accessed. Important information includes the start date, end date, computer IP address, patient location, reason for update, and a description of the event. Optional information includes the user role, chart access reason, document code, document type, orders entered, service/department, session, user’s place of work, source of access, outcome of event, participants ID, and event ID (Cruz-Correia et al. 2013). In addition to observing missing data in each log entry, the researchers indicated confusion of the format of existing audit trails, including events that span multiple lines and the omission of spacer characters to separate data fields.

Peyton, Doshi, and Seguin (Peyton et al. 2007) discuss an audit trail service for data-sharing applications. The researchers ensure privacy of users by implementing pseudonymous identities using an identity provider. For each log entry, the researchers define the following standard data fields that should be recorded: principal user ID, agent ID, client, provider, attribute name, usage (purpose of request), and timestamp.
Liu, Zhou, and Huang (Liu et al. 2005) claim that HIPAA-compliant audit trails should record the identity of the person who accessed the data, identification of the data being accessed, where the data was accessed, timestamp, type of access (create, read, update, delete), and status of the access (success/failure). The researchers focus on demonstrating a HIPAA-compliant architecture for logging and auditing access of clinical image data generated and stored through picture archiving and communication systems.

Ko, Jagadpramana, and Lee (Ko et al. 2011) discuss a file-centric logging mechanism for cloud environments that records accesses within the kernel spaces of virtual machines and physical machines. The researchers provide a sample list of data captured as part of each log entry in their framework: file access date/time, IP address, MAC address, machine type, user ID of the owner of the accessed file, group ID of the owner of the accessed file, user ID of the process owner who accessed the file, group ID of the process owner who accessed the file, and action performed on the file (create, read, write, socket send, socket receive, delete, etc.).

Huang, Bavier, and Peterson (Huang et al. 2006) discuss logging network traffic. The researchers capture network traffic flows by recording the following information: slice ID, IP protocol number, IP source address, TCP or UCP source port, IP destination address, and TCP or UDP destination port.

Similar to auditable events research, we have not identified any empirical research on determining what data should be captured as part of each log entry.
3.4 Immutability

In their evaluation of Portuguese EHR systems, Cruz-Correia et al. observed that none of the systems prevented or recorded accesses to the log files, themselves. In interviews, representatives also claimed they would need direct access to the EHR system database tables if they wanted to audit access of protected health information (Cruz-Correia et al. 2013). Sandler and Wallach (Sandler and Wallach 2007) highlight the importance of immutability in voting machines, in which the audit data and vote data of the machines studies could both be undetectably altered.

Ray et al. delegate the management of secure logging mechanisms to the cloud for cost efficiency (Ray et al. 2013). When presenting their architecture for the cloud-based log system, the researchers consider five properties: correctness of log data, tamper resistance of log entries (immutability), verifiability of log entries, confidentiality of log entries, and privacy of log entries. The researchers detail the use of encryption and cryptography in four protocols for the anonymous upload, retrieval, and deletion of log data in the cloud.

Accorsi (Accorsi 2011) also discusses a distributed architecture for logging mechanisms that uses public key cryptography and trusted computing modules. The architecture verifies the integrity and confidentiality of log entries to ensure that the origin of log data is reliable, storage of log data is tamper-evident, and log entries are encrypted.

Crosby and Wallach (Crosby and Wallach 2009) discuss a framework for tamper-evident logging that produces concise proofs of correct behavior. The framework allows queries of log entries in large-scale log servers, while also generating proofs that no other log entries were altered during the query.
Hasan and Winslett (Hasan and Winslett 2011) present an architecture for supporting immutability of transaction logs using the write-once-read-many (WORM) hardware. The audit helper component of the architecture implements real-time incremental auditing to help efficiently determine whether data has been tampered with. Bao et al. (Bao et al. 2010) propose a utility for the construction of tamper-evident logs. Like Crosby and Wallach’s approach (Crosby and Wallach 2009), the utility generates proofs and verifications of logging activity.

Accorsi and Hohl (Accorsi and Hohl 2006) focus on security of log data in pervasive computing systems when delegating logging tasks to marginally trusted modules. The researchers present a protocol built upon three steps: (1) mutual authentication of client and logging services, (2) initialization and construction of logfiles, and (3) acknowledgement of receipt from the log collector.

Overall, research on immutability of logging mechanisms seems widespread. Many researchers have presented frameworks and architectures to achieve immutability of log files using techniques such as encryption and proof-checking.

3.5 Log Monitoring

Cruz-Correia et al. found that representatives of Portuguese EHR systems very rarely or never accesses audit trails (Cruz-Correia et al. 2013). In addition, based on interviews conducted in the study, the researchers identified a lack of concern about the existence of audit trails, if they were maintained, or if they were properly secured.

Ramanathan et al. demonstrate how logging and audit services support compliance management in business applications (Ramanathan et al. 2007). In the paper, the authors
present an audit service architecture to facilitate recording auditable events, storing log entries securely, and archiving log files. However, the focus of the paper seems to involve the generation of reports from the stored log entries, and discussion of the many types of reports that may be generated (such as audit event history by user, user administration event history, and most-active-accessors reporting).

Eick, Nelson, and Schmidt (Eick et al. 1994) present a tool to graphically display lines of text of log files to facilitate auditing. The tool uses colors to distinguish different event types, and allows for direct manipulation of the visual to investigate interesting log entries or patterns.

The APPLE approach presented by Etalle and Winsborough (Etalle and Winsborough 2007) involves distributed auditing authorizes who regularly check whether users have obtained and used data documents in accordance with the governing policies. When a user is audited, the user must produce three pieces of information from the user’s log: (1) proof that the user possessed the appropriate policies to perform the activity, (2) proof that the user fulfilled the obligations required by the policies, and (3) proof that the user acquired the policies from trustworthy sources.

Myers, Grimaila, and Mills (Myers et al. 2010) present a methodology for gathering and correlating multiple web server log files for detecting malicious behavior. Nicholas et al. (Nicholas et al. 1999) also discuss analysis of web traffic log files to understand what users do while online. Heatley and Otto (Heatley and Otto 1998) also detect malicious behavior and fraud through data mining of log files. Heatley and Otto investigate whether user work
location, job code, file type, year of file access, month of access, hour of access, or workspace identification could be related to malicious user behavior.

In addition to tools to visualize log contents, researchers have also proposed frameworks and methodologies for auditing existing log entries. However, most, if not all, of the research on log monitoring assumes that the user activity and data transactions needed for auditing has already be accurately recorded in a log file.
4. Systematic Mapping Study

In this chapter, we present our methodology and results from performing a systematic mapping study to discover empirical research on user activity logging mechanisms. Section 4.1 introduces the systematic mapping study. Section 4.2 describes our methodology for including publications in our study. Section 4.3 presents results from our inclusion/exclusion process. Section 4.4 presents a discussion of our results.

4.1 Introduction

Systematic reviews, such as mapping studies, can help categorize existing empirical research on software user activity logs; identify typical research protocols used in logging research; and reveal context factors and measures used to evaluate software logging mechanism. Information gained through systematic mapping studies can help researchers formulate new research studies or replicate existing studies.

We conducted a systematic mapping study (Petersen et al. 2008) of empirical research on logging mechanisms for user accountability. The objective of the mapping study is to characterize existing empirical studies that explore software user activity logging mechanisms. We characterize the research on logging mechanisms in terms of the empirical protocol used, phase of the software lifecycle addressed by the research, and events that should be logged based on the research.

We define the following research questions to guide our mapping study:

- **RQ4.1:** What techniques exist for evaluating software logging mechanisms for user accountability?
• **RQ4.2:** What frameworks exist for designing or implementing logging mechanisms for user accountability?

• **RQ4.3:** What types of evaluation protocols are frequently used to research logging mechanisms for user accountability?

• **RQ4.4:** What events does research suggest should be captured by logging mechanisms for user accountability?

• **RQ4.5:** What open research areas exist in research for logging mechanisms for user accountability?

### 4.2 Methodology

To perform a systematic mapping study (Petersen et al. 2008), we follow the steps outlined by Zhang et al. (Zhang et al. 2011) for identifying relevant papers. Figure 1 visualizes our process for identifying relevant papers for our systematic mapping study. First, we identified venues (for example, conferences and journals) and databases from which to search for relevant studies. Next, we established a quasi-gold standard through a manual search. We then elicited search strings based on keyword analysis of our quasi-gold standard articles. Next, we conducted an automated search using the elicited search strings. After evaluating our search performance, we then moved forward by applying inclusion/exclusion criteria to our search results to identify relevant papers.
Figure 1: Process for identifying relevant publications, adapted from (Zhang et al. 2011)
4.2.1 Identifying Venues and Databases

To select possible venues (for example, conferences or journals) with relevant papers, we first collected a set of relevant venues from three primary computer science organizations: the Associated for Computing Machinery (ACM), the Institute for Electrical and Electronics Engineers (IEEE), and USENIX. We collected the full set of venues for each organization using the organization’s published index of venues and calendar of events. For each venue, we voted to include a venue as relevant for our study based on the following criteria:

- The venue name relates to user activity logs (or “audit trails”); or
- The venue name relates to security, privacy, and/or forensics; or
- The venue is a prominent software engineering venue.

To help consistency and reliability of selecting relevant studies, two researchers each voted on the full set of 484 potential venues. We documented the disagreements between the two researchers using the Cohen’s Kappa coefficient (κ). In statistics, κ is the measure of inter-rater agreement. A larger κ coefficient is considered an indicator of higher inter-rater agreement (Carletta 1996). For identifying relevant venues, κ = 0.81 for the two researchers. However, to be as inclusive as possible, we included any venue marked as relevant by at least one researcher. Table 1 summarizes the number of included venues collected from each organization.
Table 1: Summary of Venues

<table>
<thead>
<tr>
<th>Organization</th>
<th>Included Venues</th>
<th>Total Number of Collected Venues</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACM</td>
<td>12</td>
<td>99</td>
</tr>
<tr>
<td>IEEE</td>
<td>20</td>
<td>289</td>
</tr>
<tr>
<td>USENIX</td>
<td>6</td>
<td>96</td>
</tr>
<tr>
<td>TOTAL</td>
<td><strong>38</strong></td>
<td><strong>484</strong></td>
</tr>
</tbody>
</table>

For database search engines, we use the set of most common databases searched for systematic literature reviews according to the guidelines for identifying relevant studies (Zhang et al. 2011):

- IEEE Xplore\(^7\)
- ACM Digital Library\(^8\)
- ScienceDirect\(^9\)
- WebOfScience\(^10\)
- EngineeringVillage\(^11\) (Ei Compendex)
- SpringerLink\(^12\)

### 4.2.2 Establishing a Quasi-gold Standard

To establish a quasi-gold standard, we first collected citations for all available publications from each relevant venue for the past 5 years (2010-2015). Table 2 summarizes

\(^7\) [http://ieeexplore.ieee.org](http://ieeexplore.ieee.org)
\(^8\) [http://dl.acm.org/](http://dl.acm.org/)
\(^9\) [http://www.sciencedirect.com](http://www.sciencedirect.com)
\(^10\) [http://www.webofknowledge.com](http://www.webofknowledge.com)
\(^11\) [http://www.engineeringvillage.com](http://www.engineeringvillage.com)
\(^12\) [http://link.springer.com](http://link.springer.com)
the relevant venues and citations collected from each venue. In total, we collected 11,424 citations from all relevant venues from the past 5 years.

<table>
<thead>
<tr>
<th>Organization</th>
<th>Venue</th>
<th>Citations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACM</td>
<td>Annual Computer Security Applications Conference (ACSAC)</td>
<td>214</td>
</tr>
<tr>
<td></td>
<td>Automated Software Engineering (ASE)</td>
<td>473</td>
</tr>
<tr>
<td></td>
<td>ACM Symposium on Information, Computer and Communications Security (ASIACCS)</td>
<td>329</td>
</tr>
<tr>
<td></td>
<td>ACM Conference on Computer and Communications Security (CCS)</td>
<td>640</td>
</tr>
<tr>
<td></td>
<td>ACM Conference on Data and Applications Security and Privacy (CODASPY)</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>International Symposium on Empirical Software Engineering and Measurement (ESEM)</td>
<td>317</td>
</tr>
<tr>
<td></td>
<td>International Symposium on Engineering Secure Software and Systems (ESSoS)</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>Foundations of Software Engineering</td>
<td>392</td>
</tr>
<tr>
<td></td>
<td>International Conference on Security of Information and Networks (SIN)</td>
<td>287</td>
</tr>
<tr>
<td></td>
<td>International Conference on Software Engineering (ICSE)</td>
<td>957</td>
</tr>
<tr>
<td></td>
<td>ACM Transactions on Information and System Security (TISSEC)</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>ACM Transactions on Software Engineering and Methodology (TOSEM)</td>
<td>144</td>
</tr>
<tr>
<td>IEEE</td>
<td>Computer Security Foundations Symposium (CSF)</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>Cybersecurity</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Cyberspace Safety and Security</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE)</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>Symposium on Electronic Commerce and Security (ISECS)</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>Workshop on Evolving Security and Privacy Requirements Engineering (ESPRE)</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>European Intelligence and Security Informatics Conference (EISIC)</td>
<td>432</td>
</tr>
<tr>
<td></td>
<td>International Conference on IT Convergence and Security (ICITCS)</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>International Conference on IT Security Incident Management and IT Forensics (IMF)</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>International Symposium on Software Reliability Engineering (ISSRE)</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>International Conference on Privacy, Security, Risk and Trust (PASSAT)</td>
<td>757</td>
</tr>
<tr>
<td></td>
<td>International Conference on Privacy, Security and Trust (PST)</td>
<td>245</td>
</tr>
<tr>
<td></td>
<td>International Conference on Risks and Security of Internet and Systems (CRISIS)</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>Security and Privacy Workshops</td>
<td>184</td>
</tr>
<tr>
<td></td>
<td>Symposium on Security and Privacy</td>
<td>246</td>
</tr>
<tr>
<td></td>
<td>International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)</td>
<td>1068</td>
</tr>
<tr>
<td></td>
<td>Transactions on Dependable and Secure Computing</td>
<td>374</td>
</tr>
<tr>
<td></td>
<td>Transactions on Information Forensics and Security</td>
<td>1253</td>
</tr>
<tr>
<td></td>
<td>Security &amp; Privacy</td>
<td>805</td>
</tr>
<tr>
<td></td>
<td>Transactions on Software Engineering</td>
<td>486</td>
</tr>
<tr>
<td>USENIX</td>
<td>Workshop on Collaborative Methods for Security and Privacy (CollSec)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Workshop on Cyber Security Experimentation and Test (CSET)</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Workshop on Health Security and Privacy (HealthSec)</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Summit on Hot Topics in Security (HotSec)</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>Symposium on Usable Privacy and Security (SOUPS)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>USENIX Security Symposium</td>
<td>217</td>
</tr>
</tbody>
</table>
Next, we identified relevant publications to include in our quasi-gold standard. The author of this dissertation manually reviewed the title, abstract, and keywords for all 11,424 publication citations and voted to include publications based on the following inclusion criteria:

- The publication title, abstract, or keywords relate to software user activity logs;
- The publication title, abstract, or keywords relate to software user accountability;
- The publication title, abstract, or keywords relate to forensic analysis for security or privacy.

To help ensure consistency and repeatability, a second researcher voted on a subset of 1,148 (about 10% of the total publication citations). The subset of publication citations maintained the same ratio of included/excluded papers as voted by the author of this dissertation. For example, the author of this dissertation included 0.37% of the 11,424 total publication citations. Therefore, the subset of citations provided for the second researcher contained only 5 citations (about 0.44% of 1,148 total citations in the subset) that the author of this dissertation included. We documented the disagreements between the two researchers using the Cohen’s Kappa coefficient (κ). For identification of relevant publications, agreement between the two researchers was measured as κ=0.82. Overall, 42 total publications out of 11,424 were deemed relevant based on our inclusion/exclusion criteria.
Next, we read the full text of each of the 42 relevant publications to determine whether
the publication should belong to our quasi-gold standard. For including into our quasi-gold
standard, we use the following inclusion criteria:

- The publication is focused on user activity logs (or "audit trails"); and
- The publication is focused on using logs for forensics or security or privacy or
  accountability; and
- The publication is written in English.

Similarly, we excluded publications from our quasi-gold standard based on the following
criteria:

- The publication is focused on query logs; or
- The publication is focused on network logs; or
- The publication is focused on financial or process auditing; or
- The publication is not written in English.

After manually reading the full text of each publication, we identified 22 publications for our
quasi-gold standard. Table 3 presents the publications in our quasi-gold standard.
Table 3: Publications in our Quasi-gold Standard

<table>
<thead>
<tr>
<th>Publication Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecLaas: secure logging-as-a-service for cloud forensics (Zawoad et al. 2013)</td>
<td></td>
</tr>
<tr>
<td>Privacy-preserving audit for broker-based health information exchange (Oh et al. 2014)</td>
<td></td>
</tr>
<tr>
<td>Secure provenance: the essential of bread and butter of data forensics in cloud computing (Lu et al. 2010)</td>
<td></td>
</tr>
<tr>
<td>Efficient audit-based compliance for relational data retention (Hasan and Winslett 2011)</td>
<td></td>
</tr>
<tr>
<td>Policy auditing over incomplete logs: theory, implementation and applications (Garg et al. 2011)</td>
<td></td>
</tr>
<tr>
<td>Measuring the forensic-ability of audit logs for nonrepudiation (King 2013)</td>
<td></td>
</tr>
<tr>
<td>Characterizing logging practices in open-source software (Yuan et al. 2012b)</td>
<td></td>
</tr>
<tr>
<td>Log-based testing (Elyasov 2012)</td>
<td></td>
</tr>
<tr>
<td>Log Design for Accountability (Butin et al. 2013)</td>
<td></td>
</tr>
<tr>
<td>How to Do Application Logging Right (Chuvakin and Peterson 2010)</td>
<td></td>
</tr>
<tr>
<td>Implementation of Logging for Information Tracking on Network (Maeta et al. 2013)</td>
<td></td>
</tr>
<tr>
<td>Detection of Software Failures through Event Logs: An Experimental Study (Pecchia and Russo 2012)</td>
<td></td>
</tr>
<tr>
<td>S2Logger: End-to-End Data Tracking Mechanism for Cloud Data Provenance (Chun Hui et al. 2013)</td>
<td></td>
</tr>
<tr>
<td>A Group Based In-Block Logging for Flash Based Systems (Jin et al. 2013)</td>
<td></td>
</tr>
<tr>
<td>Towards Forensic Data Flow Analysis of Business Process Logs (Accorsi et al. 2011)</td>
<td></td>
</tr>
<tr>
<td>Explorative Visualization of Log Data to Support Forensic Analysis and Signature Development (Schmerl et al. 2010)</td>
<td></td>
</tr>
<tr>
<td>Flogger: A File-Centric Logger for Monitoring File Access and Transfers within Cloud Computing Environments (Ko et al. 2011)</td>
<td></td>
</tr>
<tr>
<td>Audit Mechanisms for Privacy Protection in Healthcare Environments (Blocki et al. 2011)</td>
<td></td>
</tr>
<tr>
<td>Secure Logging and Auditing in Electronic Health Records Systems: What Can We Learn from the Payment Card Industry (King and Williams 2012)</td>
<td></td>
</tr>
<tr>
<td>Cataloging and Comparing Logging Mechanism Specifications for Electronic Health Record Systems (King and Williams 2013)</td>
<td></td>
</tr>
<tr>
<td>Requirements and Design for an Extensible Toolkit for Analyzing EMR Audit Logs (Duffy et al. 2013)</td>
<td></td>
</tr>
</tbody>
</table>

4.2.3 Eliciting Search Terms

To elicit our search terms for querying the databases selected in Section 4.2.1, we perform an analysis of terms contained in the full set of our quasi-gold standard publication titles, abstracts, and keywords. Table 4 summarizes the frequencies of major terms and phrases related to our research topic contained in the quasi-gold standard publication titles, abstracts, and keywords.
Table 4: Summary of term and phrase frequencies in the quasi-gold standard publication titles, abstracts, and keywords

<table>
<thead>
<tr>
<th>Term/Phrase</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>143</td>
</tr>
<tr>
<td>security</td>
<td>83</td>
</tr>
<tr>
<td>log</td>
<td>81</td>
</tr>
<tr>
<td>logging</td>
<td>71</td>
</tr>
<tr>
<td>software</td>
<td>68</td>
</tr>
<tr>
<td>logs</td>
<td>64</td>
</tr>
<tr>
<td>forensics</td>
<td>63</td>
</tr>
<tr>
<td>forensic</td>
<td>62</td>
</tr>
<tr>
<td>audit</td>
<td>36</td>
</tr>
<tr>
<td>digital forensic</td>
<td>20</td>
</tr>
<tr>
<td>forensic analysis</td>
<td>15</td>
</tr>
<tr>
<td>secure logging</td>
<td>12</td>
</tr>
<tr>
<td>audit logs</td>
<td>9</td>
</tr>
<tr>
<td>logging and auditing</td>
<td>6</td>
</tr>
<tr>
<td>data provenance</td>
<td>5</td>
</tr>
<tr>
<td>secure logging and auditing</td>
<td>4</td>
</tr>
<tr>
<td>accountable log</td>
<td>4</td>
</tr>
</tbody>
</table>

4.2.4 Conducting Automated Searches

We combine the common terms/phrases from Table 4 into a search query string to perform automated searches of the databases selected in Section 4.2.1. For example:

“secure logging” AND “security” AND (“forensic” OR “forensics”)

Once we have a search query string, we open each database (IEEE Xplore, ACM Digital Library, ScienceDirect, Web of Science, Engineering Village, and SpringerLink) and enter the search query string to perform the automated search.

After executing the search, we download the citations for all papers in the search results. The citation includes the title, authors, keywords (if available), abstract (if available), year, and venue.
4.2.5 Evaluate Search Performance

To evaluate the performance of our search query string against the publications contained in our quasi-gold standard, we calculate two metrics (Zhang et al. 2011):

\[
\text{quasi - sensitivity} = \frac{\text{number of relevant studies retrieved}}{\text{total number of relevant studies}},
\]

\[
\text{precision} = \frac{\text{number of relevant studies retrieved}}{\text{number of studies retrieved}},
\]

where the \text{number of relevant studies retrieved} is the number of quasi-gold standard publications contained in the set of database search results; \text{total number of relevant studies} is the total number of publications in our quasi-gold standard; and \text{number of studies retrieved} is the total number of studies retrieved from conducting the automated search.

According to the methodology for identifying relevant software engineering papers (Zhang et al. 2011), the search query string must produce a quasi-sensitivity of at least 72% to be acceptable. After performing 14 rounds of automated searches using different configurations of search query strings, the search query string presented in Table 5 produced a quasi-sensitivity of 72.7%.

<table>
<thead>
<tr>
<th>Table 5: Final search query string</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&quot;forensic analysis&quot; OR &quot;log analysis&quot; OR accountability OR forensic) AND (&quot;secure log&quot; OR &quot;secure logging&quot; OR &quot;security log&quot; OR &quot;audit log&quot; OR &quot;audit logs&quot; OR &quot;logging mechanism&quot; OR &quot;software log&quot; OR &quot;log data&quot; OR &quot;access log&quot; OR &quot;access logs&quot; OR logger OR &quot;application logging&quot;) AND (&quot;security&quot;)</td>
</tr>
</tbody>
</table>
4.2.6 Including/Excluding Publications

After conducting the automated search using the search query string in Table 5, we collect a total of 1,355 citations. A breakdown of search results appears in Table 6.

<table>
<thead>
<tr>
<th>Database</th>
<th>Search Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE Xplore</td>
<td>39</td>
</tr>
<tr>
<td>ACM Digital Library</td>
<td>784</td>
</tr>
<tr>
<td>ScienceDirect</td>
<td>386</td>
</tr>
<tr>
<td>Web of Science</td>
<td>9</td>
</tr>
<tr>
<td>Engineering Village</td>
<td>49</td>
</tr>
<tr>
<td>SpringerLink</td>
<td>88</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>1355</strong></td>
</tr>
</tbody>
</table>

4.2.6.1 Round 1: Title & Abstract Review

In the first round of including/excluding publications for our study, the author of this dissertation reviewed the titles and abstracts of all 1,355 publications using the following criteria:

**Inclusion criteria:**

- The publication is focused on user activity logs (or "audit trails"); and
- The publication is focused on using logs for forensics or security or privacy or accountability; and
- The publication is written in English; and
- The publication is an empirical research paper (not a magazine or news article).

**Exclusion criteria:**

- The publication is focused on query logs; or
- The publication is focused on network logs; or
- The publication is focused on financial or process auditing; or
- The publication is not written in English; or
- The publication is not an empirical research paper.

After the author of this dissertation reviewed all 1,355 publications, to check consistency and repeatability, a second researcher reviewed a subset of the total publications. The second researcher voted on a subset of 271 publications (20% of the total publications). The subset contained the same ratio of included to excluded publications based on the votes of the author of this dissertation. For example, the author of this dissertation included 147 out of 1,355 publications (11%), so the subset for the second researcher contained only 30 publications included by the author of this dissertation (11% of the 271 total publications in the subset). We documented the disagreements between the two researchers using the Cohen’s Kappa coefficient (κ). For identification of relevant publications, agreement between the two researchers was measured as κ=0.75. Therefore, based on the value of the Cohen’s Kappa coefficient, we continued to the next round of full text reviews using the set of included publications from the author of this dissertation.

4.2.6.2 Round 2: Full Text Review

After the initial round of including/excluding publications based on title and abstract, we must further narrow the set of relevant papers based on the actual content of the full text version of the papers. We followed the same format as the Round 1 review: the author of this dissertation voted on the full set of 147 publications output from Round 1, and a second researcher voted on a subset of publications to check consistency and repeatability. For full text reviews, we use the following expanded inclusion/exclusion criteria:
**Inclusion Criteria**

- The publication is focused on user activity logs (or "audit trails"); and
- The publication is focused on using logs for forensics or security or privacy or accountability; and
- The publication is written in English; and
- The publication is an empirical research paper (not a magazine or news article); and
- The publication is focused on software engineering for logs/logging mechanisms (for example, requirements, design, implementation, testing); and
- The full text of the publication is available.

**Exclusion Criteria:**

- The publication is focused on query logs; or
- The publication is focused on network logs; or
- The publication is focused on financial or process auditing; or
- The publication is not written in English; or
- The publication is not an empirical research paper; or
- The publication is focused on auditing or analyzing existing log files; or
- The publication is focused on preventing tampering of logs; or
- The full text of the publication is unavailable; or
- The publication is not focused on logs that record user activity.
After the author of this dissertation voted on all 147 publications based on the criteria above, the second reviewer voted on a subset of 30 publications (20% of the total set of 147 publications). The subset contained the same ratio of included to excluded papers as voted by the author of this dissertation. For example, the author of this dissertation included 10 out of 147 total publications (15%), so the subset contained only 4 publications included by the first reviewer (15% of the 30 total publications in the subset). We documented the disagreements between the two researchers using the Cohen’s Kappa coefficient (κ). For identification of relevant publications, agreement between the two researchers was measured as κ=0.63. Based on the value of the Cohen’s Kappa coefficient, the two researchers met to manually resolve disagreements to ensure consistent understanding and application of the inclusion/exclusion criteria. After resolving disagreements, we continued to the next round of the systematic process using 10 included publications.

4.2.6.3 Round 3: Snowball Reference Analysis

For each of the 10 publications included in the previous round of including/excluding, the author of this dissertation examined the set of references of the publication to identify any additional relevant publications that should be included in our study. However, no additional references were included based on the inclusion and exclusions criteria presented in Section 4.2.6.2. References examined in this round of inclusion/exclusion were primarily related to auditing or analysis of existing log files, or the referenced publications were not empirical research papers. For example, included publications cited “guidelines” for logging mechanisms or blog posts suggesting logging practices. Therefore, we only identified 10 relevant publications to include in our systematic mapping study.
### Table 7: Final set of publications included in our study

<table>
<thead>
<tr>
<th>ID</th>
<th>Publication Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>S2Logger: End-to-End Data Tracking Mechanism for Cloud Data Provenance (Chun Hui et al. 2013)</td>
</tr>
<tr>
<td>B</td>
<td>Flogger: A File-Centric Logger for Monitoring File Access and Transfers within Cloud Computing Environments (Ko et al. 2011)</td>
</tr>
<tr>
<td>C</td>
<td>Ideal log setting for database forensics reconstruction (Adedayo and Olivier 2015)</td>
</tr>
<tr>
<td>D</td>
<td>Secure log management for privacy assurance in electronic communications (Stathopoulos et al. 2008)</td>
</tr>
<tr>
<td>E</td>
<td>Characterizing logging practices in open-source software (Ding et al. 2012)</td>
</tr>
<tr>
<td>F</td>
<td>Two security patterns: least privilege and security logger and auditor (Fernandez et al. 2011)</td>
</tr>
<tr>
<td>G</td>
<td>Modifying without a trace: general audit guidelines are inadequate for open-source electronic health record audit mechanisms (King et al. 2012)</td>
</tr>
<tr>
<td>H</td>
<td>Where do developers log? an empirical study on logging practices in industry (Fu et al. 2014)</td>
</tr>
<tr>
<td>I</td>
<td>Log your CRUD: design principles for software logging mechanisms (King and Williams 2014)</td>
</tr>
<tr>
<td>J</td>
<td>Cataloging and comparing logging mechanism specifications for electronic health record systems (King and Williams 2013)</td>
</tr>
</tbody>
</table>

#### 4.2.7 Data Extraction

From each of the included papers in our study, we extract data from each publication to help answer our research questions. In addition, for systematic literature reviews, Kitchenham and Charters (Kitchenham and Charters 2007) recommend assessing the quality of included publications to evaluate the validity of the evidence presented in the publication.

Table 8 summarizes the data extracted for analysis as part of our systematic mapping study.

<table>
<thead>
<tr>
<th>Category</th>
<th>Data Extracted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Timeframe</td>
</tr>
<tr>
<td></td>
<td>Year published</td>
</tr>
<tr>
<td>Empirical Protocol</td>
<td>What is the explicitly-stated objective of the research?</td>
</tr>
<tr>
<td></td>
<td>What are the explicitly-stated research questions or hypotheses?</td>
</tr>
<tr>
<td></td>
<td>What are the explicitly-stated contributions?</td>
</tr>
<tr>
<td></td>
<td>What type of evaluation protocol do the researchers follow?</td>
</tr>
<tr>
<td></td>
<td>What are the explicitly-stated threats to validity?</td>
</tr>
<tr>
<td>Software Engineering</td>
<td>Phase</td>
</tr>
<tr>
<td></td>
<td>What phase of the software lifecycle does the research relate?</td>
</tr>
<tr>
<td>Log Events</td>
<td>What log events do the researchers require for logging mechanisms?</td>
</tr>
</tbody>
</table>
4.3 Results

Table 9 summarizes the reasons why publications were excluded in Round 2 with full text reviews. The most common reason for excluding a publication was publications focusing on auditing or analysis of existing log files. Twenty-seven publications were excluded for not focusing on logs that record user activity. Twenty-two publications did not have full text available for reviewing. In addition, 19 publications were not empirical research papers, but were instead magazine, editorial, or news articles. Thirteen publications were related to preventing tampering and ensuring immutability of logs.

<table>
<thead>
<tr>
<th>Exclusion Criteria</th>
<th>Publications Excluded</th>
</tr>
</thead>
<tbody>
<tr>
<td>The publication is focused on auditing or analyzing existing log files</td>
<td>53</td>
</tr>
<tr>
<td>The publication is not focused on logs that record user activity</td>
<td>27</td>
</tr>
<tr>
<td>The full text of the publication is unavailable</td>
<td>22</td>
</tr>
<tr>
<td>The publication is not an empirical research paper</td>
<td>19</td>
</tr>
<tr>
<td>The publication is focused on preventing tampering of logs</td>
<td>13</td>
</tr>
<tr>
<td>The publication is focused on network logs</td>
<td>2</td>
</tr>
<tr>
<td>The publication is not written in English</td>
<td>1</td>
</tr>
<tr>
<td>The publication is focused on query logs</td>
<td>0</td>
</tr>
<tr>
<td>The publication is focused on financial or process auditing</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>137</td>
</tr>
</tbody>
</table>

Table 10 summarizes the data extracted from each of the 10 publications included for our systematic mapping study.

In terms of empirical protocols, four of the 10 publications did not state any research objective. Five of the 10 publications did not state any research questions or hypotheses to guide the scientific inquiry. Seven of the 10 publications employed case studies as a form of
evaluating contributions. Two of the 10 publications provided only discussion as a form of demonstrating or evaluating the contributed approach. Similar, one publication provided a security analysis as an evaluation protocol.

For software lifecycle phase affected by the described approach, three publications relate to the requirements phase of developing logging mechanism. Three publications describe approaches related to the design of logging mechanisms. Six publications describe approaches related to implementation of logging mechanisms. Two publications describe approaches to testing logging mechanisms.

For log events described, two publications provided lists of specific events that should be logged (for example, “user login/logout” and “grant access right”). Six publications provided general descriptions of events that should be logged (for example, “security functions” and “data modification queries”). Two publications did not describe events that should be logged for the given approaches.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2013</td>
<td>enabling tracking of end-to-end data provenance within and across all guest and host physical machines in distributed virtualized environments</td>
<td>(none stated)</td>
<td>We introduce S2Logger, a data event logging mechanism which captures, analyses and visualizes data events in the cloud from the data point of view</td>
<td>Case study (simulation)</td>
<td>(none stated)</td>
<td>Design; Implementation</td>
<td>system call table events;</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2011</td>
<td>(none stated)</td>
<td>(none stated)</td>
<td>We present Flogger, a novel file-centric logger suitable for both private and public Cloud environments</td>
<td>Case study (simulation)</td>
<td>Does not consider network transport of logs; logs grow at a higher rate with the proposed approach</td>
<td>Requirements; Implementation</td>
<td>every file access in VMs and PMs</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>2015</td>
<td>(none stated)</td>
<td>(none stated)</td>
<td>This paper introduces the notion of the ideal log setting necessary for an effective reconstruction process in database forensics</td>
<td>Case study (real systems)</td>
<td>(none stated)</td>
<td>Implementation</td>
<td>data modification queries; data definition queries; metadata changes</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>2008</td>
<td>(none stated)</td>
<td>(none stated)</td>
<td>We propose an extended security model for logging in network providers, which also considers internal modification attacks</td>
<td>Security analysis discussion</td>
<td>(none stated)</td>
<td>Requirements; Design; Implementation</td>
<td>security functions; service management functions; network management functions</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2012</td>
<td>Our goal is to reveal issues with the current logging practices, therefore we only focus on the modifications (including deletions) to the previously existing logging code.</td>
<td>How pervasive is software logging? Is the current logging practice good enough? How are developers modifying logs?</td>
<td>This paper makes the first attempt (to the best of our knowledge) to provide a quantitative characteristic study of the current log messages within four pieces of large open-source software.</td>
<td>Case study (real systems)</td>
<td>Representativeness of the software and examination methodology</td>
<td>Implementation</td>
<td>(none stated)</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>2011</td>
<td>(none stated)</td>
<td>(none stated)</td>
<td>We present here two security patterns that describe fundamental aspects: Least Privilege and Security Logger/Auditor</td>
<td>Discussion</td>
<td>Can incur significant overhead since each object access has to be logged; A decision must be made by software designers as to the grain size at which objects are logged; There is a tradeoff between security and performance;</td>
<td>Design</td>
<td>(none stated)</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>2012</td>
<td>The objective of this paper is to assess electronic health record audit mechanisms to determine the current degree of auditing for non-repudiation and to assess whether general audit guidelines adequately address nonrepudiation.</td>
<td>RQ1: What events should be included in an EHR log file for non-repudiation? RQ2: What are the strengths and weaknesses of software auditing mechanisms in current open-source EHR systems?</td>
<td>We compare and contrast [results of evaluating audit mechanisms] and suggest techniques for healthcare software developers to strengthen EHR audit mechanisms</td>
<td>Case study (real systems)</td>
<td>Manual interpretation of audit log entries through each software’s user interface; May not be representative of the level of auditing that other software may provide; May have misjudged which requirements relate to protected health information for black-box assessment;</td>
<td>Testing</td>
<td>user login/logout; session timeout; account lockout; create data; update data; delete data; view data; query data; signature created/validated; export data; import data; system backup; system restore; grant access rights; modify access rights; revoke access rights</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>2014</td>
<td>We systematically study the logging practices of developers in industry, with focus on where developers log.</td>
<td>RQ1: What categories of code snippets are logged? RQ2: What factors are considered for logging? RQ3: Is it possible to automatically determine where to log?</td>
<td>We conduct an empirical study on logging practices in industry; We summarize five categories of logged snippets; We characterize both logged and unlogged code snippets of catch blocks and return-value-check snippets; We demonstrate the potential feasibility of predicting where to log</td>
<td>Case study (real systems)</td>
<td>Subjectiveness in the categorization of logged snippets; The degree to which the subject software systems are representative of true practice;</td>
<td>Implementation</td>
<td>unexpected situations; execution points;</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>2014</td>
<td>The objective of this research is to propose design principles for logging mechanisms based upon an exploratory case study in which we systematically evaluate logging mechanisms by supplementing the expected results of existing functional black-box test cases to include log output.</td>
<td>RQ1: What observations can we make to understand why the four studied EHR logging mechanisms do not capture some specific user actions? RQ2: What observations can we make about the general security of the four studied EHR logging mechanisms? RQ3: What principles of logging mechanism design, implementation, and testing may be proposed based on observations of the four studied EHR systems?</td>
<td>A systematic process for evaluating logging mechanisms based on existing black-box test suites; An evaluation of four popular open-source EHR system logging mechanisms; A set of principles to guide the design, implementation, and testing of logging mechanisms</td>
<td>Case study (real systems)</td>
<td>Principles are based solely on observations from four open-source EHR systems that may not accurately reflect the state of logging mechanism; Evaluation criteria are based on a subset of all NIST test procedures for health information technology</td>
<td>Design; Testing</td>
<td>actions performed by users</td>
<td></td>
</tr>
</tbody>
</table>
The objective of this research is to guide the design of electronic health record systems by cataloging suggested information that should be captured by logging mechanisms from both healthcare and non-healthcare documentation.

Table 10 Continued

<table>
<thead>
<tr>
<th>J</th>
<th>2013</th>
<th>The objective of this research is to guide the design of electronic health record systems by cataloging suggested information that should be captured by logging mechanisms from both healthcare and non-healthcare documentation.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RQ1: What data transactions should be logged in EHR systems?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RQ2: What security events should be logged in EHR systems?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RQ3: What log entry content should be captured for each log entry in EHR systems?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RQ4: What data transactions, security events, and log entry content are not included in healthcare documents, but are included in non-healthcare documents?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RQ5: Which document offers the most detailed specifications for data transactions, security events, and log entry content that should be captured by EHR logging mechanisms?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RQ6: What minimal set of documents covers 100% of the cataloged data transactions, security events, and log entry content?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A centralized catalog of data transactions, security events, and data elements for log entry content; Documented traceability between the source documentation and each collected data transaction, security event, and log entry content data element.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Discussion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Catalog may not fully represent all the necessary standards, requirements, regulations, and logging mechanism specifications available in the field; May have incorrectly interpreted certain descriptions of the data transactions, security events, and log entry content during categorization; Current methodology diminishes the benefit of having a specific list of PHI data objects and the CRUD actions that apply to each.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 data transactions; 77 security events</td>
</tr>
</tbody>
</table>
4.4 Discussion

In this section, we answer our research questions.

4.4.1 Evaluating Logging Mechanisms

**RQ4.1:** What techniques exist for evaluating software logging mechanisms for user accountability?

As shown in Table 10, only two of the 10 publications included in our study propose methods for evaluating software logging mechanisms. King et al. (King et al. 2012) perform a black-box evaluation of software logging mechanisms based on specific user actions extracted from a set of certification criteria for electronic health record (EHR) systems. The certification criteria describe functionality users of EHR systems should be able to perform; any user action that interacts with protected health information was deemed an event that should be logged. However, the certification criteria is not applicable in domains other than healthcare, so the approach is not generalizable.

Similarly, King et al. (King and Williams 2014) also propose a methodology for supplementing existing black-box test cases with expected log output. However, the proposed approach requires software systems to maintain existing black-box test suites for evaluating functionality in the system.

*Through our systematic mapping study, we were able to identify only two techniques for black-box evaluation of software logging mechanisms for ensuring necessary log entries are recorded following actions performed by users. Each of the identified techniques has limitations that prevent the technique from being generalizable.*
**4.4.2 Frameworks for Designing & Implementing Logging Mechanisms**

*RQ4.2: What frameworks exist for designing or implementing logging mechanisms for user accountability?*

Based on the publications included in our systematic mapping study, we identified two frameworks for designing and/or implementing logging mechanisms for user accountability. First, S2Logger (Chun Hui et al. 2013) provides tracking of data in the cloud, from creation to transmission across systems and network boundaries. The objective of the S2Logger framework is to facilitate real-time detection of security policy violations (such as loss of data or data leakage). The framework also supports effective forensic analysis following data security breaches. Overall, S2Logger provides enables tracking of data provenance in the cloud as data is transmitted from one location to another. The framework is implemented at the kernel layer of the Linux operating system and is designed to log all system call table events. Similarly, Flogger (Ko et al. 2011) is another framework designed for cloud-based systems. Flogger records file-centric accesses and transfer information from within the kernel spaces of both virtual machines and physical machines in the cloud. Flogger intercepts and logs every file access in virtual machines.

We also identified a security pattern for logging mechanisms (Fernandez et al. 2011). The proposed security logger/auditor pattern logs all security-sensitive actions performed by users and provides controlled access to the log entries for audit purposes. The pattern involves a LoggerAuditor component, which logs user actions as LogEntry objects; a
SecurityAdmin component, which can enable or disable logs in the system; and an Auditor, who can access and read logs to detect possible unauthorized actions.

_We found two frameworks, S2Logger and Flogger, to help log data access in cloud-based systems for facilitating data provenance. We found one framework, a security logger/auditor security pattern, to facilitate the design and implementation of logging mechanisms, regardless of domain._

### 4.4.3 Evaluation Protocols for Research on Logging Mechanisms

**RQ4.3:** What types of evaluation protocols are frequently used to research logging mechanisms for user accountability?

As shown in Table 10, cases studies appear in 70% of the publications and are most frequently used to evaluate research on logging mechanisms. Five of the seven case studies involve evaluation of real-world software systems. Two of the seven case studies involve evaluation using a simulated environment for measuring performance. Other forms of evaluation include discussion of examples, as well as discussions that include a security analysis of the proposed approach.

However, the included studies often omitted significant pieces of the empirical protocol, such as stating objectives, research questions, and threats to validity. Four studies did not describe a research objective. Five studies did not explicitly state research questions to guide scientific inquiry. Three studies did not state limitations or threats to validity of the proposed approach.
The most common evaluation protocol used in the identified studies involved case studies of real systems. Two studies contained case studies using simulated systems. Two studies contained discussions of examples of applying the proposed approaches. One study contained a discussion based on a security analysis of the proposed approach. Studies sometimes omit important aspects of the empirical protocol used for reporting, such as objectives, research questions, and threats to validity.

4.4.4 Log Events Suggested by Existing Research

*RQ4.4:* What events does research suggest should be captured by logging mechanisms for user accountability?

Only two studies describe specific events that should be logged (King et al. 2012; King and Williams 2013). In these two studies, combined, a total of 104 events are identified. However, the explicit purpose of one study (King and Williams 2013) was to identify log events specified throughout the literature. While the two studies identify events that should be logged, neither study provides a systematic way of determining how to decide which events to log. Instead, the two studies rely on existing standards and specifications that do not provide empirical justification.

Six studies describe general events that should be logged. For example, the S2Logger framework (Chun Hui et al. 2013) logs all system call table events. One study identifies general actions that should be logged in database systems (Adedayo and Olivier 2015), such as data modification queries, data definition queries, and metadata changes. A study on where developers log throughout source code (Fu et al. 2014) identified two broad
categories of log events: unexpected situations (such as exception conditions) and execution points (such as logging which branches are executed).

Two studies provide specific lists of events that should be logged, for a combined total of 104 different events (the researchers provide a complete catalog of events that should be logged (King and Williams 2013)). The set of specific events include atomic events (such as creating, reading, updating, or deleting data), along with non-atomic events (such as merging data), and security events (such as granting and revoking access privileges). Other studies provide lists of generic actions (such as “database modification actions”) that should be logged.

4.4.5 Open Research Areas on Logging Mechanisms

RQ4.5: What open research areas exist in research for logging mechanisms for user accountability?

Research on logging mechanisms seems to be a maturing field. The oldest study included in our mapping study was published in 2008. Two studies were published in 2011. Two studies were published in 2012. Two studies were published in 2013. Two studies were published in 2014. One study was published through August of 2015.

While the included studies each address different phases of the software lifecycle, from requirements, design, implementation, and testing, none of the studies describe systematic, generalizable approaches to identifying requirements for logging mechanisms; identifying events that should be logged; implementing logging mechanisms; or testing logging mechanisms. The two frameworks identified, S2Logger (Chun Hui et al. 2013) and Flogger
(Ko et al. 2011), were intended to solve specific problems in cloud-based systems. However, neither framework is generalizable beyond cloud-based systems. Similarly, both testing approaches identified (King et al. 2012; King and Williams 2014) rely on pre-determined sets of log events and existing black-box test suites, respectively.

In addition, as shown in Table 9, several publications already exist related to auditing or analysis of log files: 53 individual publications were excluded from this mapping study because they were focused on techniques for analyzing existing log files. In our mapping study, we were concerned with research on logging (the act of recording activity to log files), not auditing (the act of gathering information that exists within log files). While an abundance of research exists related to auditing or analysis of log files, we identified only 10 publications related to logging for user accountability for this mapping study. Most research seems to assume logs already contain data necessary to enable meaningful analysis. However, as shown in the studies included in our mapping study, logs are often disabled by default (Adedayo and Olivier 2015), incomplete (King et al. 2012; King and Williams 2014), or used for purposes other than tracking user activity (Ding et al. 2012; Fu et al. 2014).

Open research exists in the area of systematic identification of requirements for logging mechanisms and identification of events that should be logged; systematic testing of logging mechanisms; and design and implementation of logging mechanisms.

4.5 Threats to Validity

In this section, we discuss threats to validity of our systematic mapping study.
4.5.1 Selection of Search Terms and Databases

Our search terms and databases chosen for executing our search query string may prevent some relevant studies from being included in our mapping study. To mitigate this threat, we followed the methodology for selecting relevant studies in software engineering (Zhang et al. 2011). We searched six digital library databases and formed our search query string based on the repeatable process using a quasi-sensitivity metric for determining the acceptability of our search query string.

4.5.2 Selection of Studies

We were interested in empirical research on logging mechanisms for user accountability. Our inclusion and exclusion criteria may have excluded relevant studies focused on logging. The terms “logging” and “auditing” sometimes appear interchangeably in the field, so we may have excluded relevant studies that used the term “auditing” to describe the process of recording user activity as log entries into a log file. To mitigate this threat, we included publications that used the terms “logging” and “auditing” until the final round of reviews, in which we reviewed the full text of the publication to ensure the subject matter reflected the act of logging (recording events to log files).

4.5.3 Reliability of Data Extraction

We extracted data related to the empirical protocols (stated objectives, research questions, contributions, evaluation protocol, and threats to validity) from the included studies to answer our research questions. We also extracted data related to software engineering lifecycle involved in each study, as well as proposed log events. To mitigate this
threat, the extraction process was performed by a researcher with over five years of experience with logging mechanisms and empirical software engineering research protocols. Any ambiguities in data extraction were resolved by a second researcher.

4.6 Conclusion

In this study, we reported the results of our systematic mapping study by documenting and analyzing the current state of empirical research on logging mechanisms for user accountability. While we identified at least 53 studies related to auditing or analysis of log files, we identified only 10 primary studies related to the requirements, design, implementation, and testing of logging mechanisms for user accountability. Research on logging mechanisms for user accountability seems to be a young, but maturing field. Further empirical development and evaluation of logging mechanisms is needed to strengthen the use of software logs for holding users accountable and performing meaningful forensic analysis after a security or privacy breach.
5. Identifying MLEs in Natural Language Software Artifacts

A naïve reaction to address the problem of inadequate logging mechanisms involves logging “everything”. However, to comprehensively evaluate logging mechanisms, software engineers must first identify the set of “everything” to be logged. Logging “everything” often introduces resource and performance (Yuan et al. 2012a) issues. Furthermore, excessive logging also tends to clutter the audit trail for forensic analysis and hinder a system administrator's ability to detect anomalous conditions (2013a). In this chapter, we present our work on helping software developers identify what to log in their software system based on user actions described in natural language software artifacts.

Although specifications exist for stating how to implement logging mechanisms for user accountability (2013b; King and Williams 2013; 2014), no rigorous specification or systematic process exists to guide software engineers in determining what user activity must be logged. For example, consider the sentence from the Open Conference System user guide:

If you wish to begin creating new accounts immediately however (to begin assigning roles such as Track Directors), you can proceed by selecting the Create New User link.

Software engineers must first mentally process the natural-language structure of the sentence to identify each action described, and then determine which of the identified actions must be logged. In the example, “wish to begin”, “creating new accounts”, “assigning roles”, “proceed”, and “selecting the Create New User link” all describe actions that users may perform in the software, but how should a team of software engineers consistently and systematically determine whether each action must be logged or not?
Through this study, we help software engineers strengthen forensic-ability and user accountability by 1) systematically identifying mandatory log events through processing of unconstrained natural language software artifacts; and 2) proposing empirically-derived heuristics to help determine whether an event must be logged.

We study unconstrained natural language software artifacts for three open-source software systems:

- iHRIS\textsuperscript{13}: Open Source Human Resources Information Solutions v4.2
- iTrust\textsuperscript{14}: Open Source Electronic Health Record System v18
- OCS\textsuperscript{15}: Open Source Scholarly Conference Management System v2.3.6

We then manually identify all pairs of verbs and objects acted upon from the software artifacts studied. Next, two researchers individually classify each verb-object pair as being a mandatory log event or not (for the purpose of holding users accountable in the software system). Based on observations and discussions of disagreements of whether a verb-object pair must be logged, we develop a set of heuristics to help other software engineers identify mandatory log events in a given software system.

To guide our study on identifying MLEs, we defined the following research questions:

- **RQ5.1:** How often do descriptions of mandatory log events appear in natural language software artifacts?

---

\textsuperscript{13} http://www.ihris.org
\textsuperscript{14} http://agile.csc.ncsu.edu/iTrust
\textsuperscript{15} https://pkp.sfu.ca/ocs/
• **RQ5.2:** What similarities and differences exist in the grammar and vocabulary used in different software artifacts?

• **RQ5.3:** What factors help decide whether an action must be logged?

Section 5.1 presents our methodology. Section 5.2 presents our results. Section 5.3 provides discussion. Section 5.4 presents our empirically-derived heuristics for identifying MLEs. Section 5.5 discusses guidelines for authors of natural language software artifacts to facilitate identification of MLEs. Section 5.6 presents threats to validity. Section 5.7 summarizes our study on identifying MLEs.

### 5.1 Methodology

Our methodology for identifying MLEs consists of five key activities:

1) selecting software and associated software artifacts to use in the study;

2) preprocessing natural-language software artifacts;

3) extracting verb-object pairs from the software artifacts;

4) classifying the verb-object pairs as mandatory log events or not; and

5) comparing and reconciling differing annotations.

#### 5.1.1 Step One: Selecting Software & Software Artifacts

We use the following inclusion criteria when selecting candidate software to use for this study:

1) The software developers must maintain a software requirements specification document;

OR
The software’s development community must maintain a user guide.

2) The software codebase must be readily available to install and deploy locally for planned follow-up studies.

Since we focus on logging for nonrepudiation and accountability, we need to identify the set of user activity possible in a software system. We used natural-language requirements specifications or user guides as the software artifacts for this study because: 1) they are readily available to software engineers; and 2) these documents typically are the primary resources that describe actions a user can perform in a software system.

For each candidate software application, we manually browsed the software’s website to locate applicable natural-language software artifacts. If no natural-language software artifacts were found, we contacted the software’s development community to find any natural-language software artifact that may be available. The inclusion criteria for selecting software artifacts to use in the study:

1) The artifact must be written in unconstrained natural language in English.

2) The artifact must describe a set of actions that users may perform in either:
   a) the entire software application; or
   b) at least one complete module of functionality within the software application.

Since we want to identify a complete (or near complete) set of mandatory log events for software in this study, we did not consider software artifacts that were incomplete or described only a subset of possible user activity for a given module of functionality in the system.
5.1.2 Step Two: Preprocessing of Natural-Language Software Artifacts

After selecting the natural-language software artifacts for our study, we processed the artifacts to make them amenable for use. We first converted the original natural-language software artifact documents into plaintext format to remove any non-natural language components like graphics, visuals, and embedded syntax. Converting to plaintext format also facilitated easier processing of the text that appeared in tables.

We next separated each sentence (typically delimited by a period followed by at least one whitespace or carriage-return ‘
’) in the document by manually opening the file in a text editor and splitting paragraphs so that individual sentences are contained on separate lines. After separating each sentence, we listed the extracted sentences on individual rows in a newly-created spreadsheet. We then proceeded with the next activity in our methodology: extracting verb-object pairs.

5.1.3 Step Three: Extracting Verb-Object Pairs

In this study, we considered each verb and the object being acted upon as a basic description of an action. In grammar, verbs are the fundamental constructs that express an action being executed against an entity (indicated by an object). We express a verb-object pair as a tuple of the form <verb, object>. For each verb identified, we lemmatize the term to obtain the base form, or lemma, of the verb. To extract verb-object pairs, we consider the following guidelines for each sentence:
• Explicitly stated verb-object pairs. Extract any verbs contained in the sentence, then identify any objects being acted upon by the verb
  o Example 1: “Doctors prescribe medications.”
    ▪ verb-object pair: <prescribe, medication>

• Implied verb-object pairs. Extract any words in the sentence whose lemma is a verbal (e.g., gerunds, participles, and infinitives are verbals that function as nouns in a sentence), then identify any objects being acted upon by the verbal
  o Example 2: “Creating a patient…”
    ▪ verb-object pair: <create, patient>
  o Example 3: “The submitted proposal…”
    ▪ verb-object pair: <submit, proposal>

• Compound verb-object pairs. For any sentence that contains compound verbs or more than one object for a single verb, we document multiple verb-object pairs to consider each individual combination of verb and object:
  o Example 4: “Doctors prescribe and update medications”
    ▪ verb-object pair: <prescribe, medication>
    ▪ verb-object pair: <update, medication>

Each software artifact sentence contained zero or more documented verb-object pairs. We documented each verb-object pair in the spreadsheet created in Section 5.1.2 on separate rows beneath the original, unchanged source sentence.
5.1.4 Step Four: Classifying Verb-Object Pairs

For each software artifact, two researchers individually classified each verb-object pair as being a mandatory log event or not based on their prior experience and knowledge of logging mechanisms for holding users accountable for their actions in software system. The first researcher (the author of this dissertation) had assisted with teaching of software engineering related courses to undergraduate computer science students from 2009-2015. The second researcher had over 2.5 years of industrial software development experience. To avoid introducing bias into our classifications, and to prevent over-restricting our classifications and potentially overlooking relevant verb-object pairs, we used only the following general guideline for our classifications:

A mandatory log event is a user interaction with a system resource that must be logged to hold the user accountable for performing the action.

For this study, we did not discriminate between actions performed upon general data, sensitive data, or protected data. The set of “sensitive” or “protected” data varies from one domain to another and between the opinions of different individuals. This study identified a full set of user activity performed upon any data in the system. User activities performed on sensitive or protected data would be a subset of the user activity identified using our current methodology and should be identified using expert knowledge within a given software system’s domain.

We created two copies of our spreadsheet containing the documented verb-object pairs from Section 5.1.33). Each of the two researchers received a copy of the spreadsheet and
classified each verb-object pair by annotating mandatory log event (Y) or not (N) beside each verb-object pair in the spreadsheet.

5.1.5 Step Five: Comparing and Reconciling Classifications

After performing individual classifications, we compiled each spreadsheet with individual classifications into a single spreadsheet for comparison. For each disagreement in our classifications, the two researchers met to discuss and justify their decisions. We documented key points discussed when resolving our discrepancies. When disagreements could not be resolved between the two researchers, a third researcher broke the tie and resolved the disagreement. We documented the final classification for each verb-object pair, as well as all disagreements and resolutions.

5.2 Results

We first discuss the software and related software artifacts selected for our study. Next, we discuss the verb-object pairs extracted from each source artifact. We then present the results of our classification of the verb-object pairs.

5.2.1 Software & Software Artifacts Studied

For our study, we selected three open-source software applications from different domains:

- iHRIS: Open Source Human Resources Information Solutions v4.2. According to the iHRIS community, 15 countries are using the software, with more than 675,000 health worker records currently supported in iHRIS.
- iTrust: Open Source Electronic Health Record System v18. An electronic health record (EHR) system developed and maintained by undergraduate software
engineering students at North Carolina State University and used by many researchers and educators as a test-bed (Meneely et al. 2012).

- OCS (Open Conference Systems): Open Source Scholarly Conference Management System v2.3.6. A conference management system developed and maintained by the Public Knowledge Project (PKP), a multi-university initiative developing free open-source software and conducting research to improve quality of scholarly publishing.

We collected the following three software artifacts, one for each of the selected software applications, which describe actions users may perform in the software:

- iHRIS: Content Management System traditional software requirements specification for the Page Builder module (the only module with a documented set of requirements).
- iTrust: Use-case based software requirements specification.
- OCS: “OCS in an Hour” booklet user guide (2008). We could not locate a requirements specification for this system, so we consider the user guide as a form of requirements specification (Berry et al. 2004).

We collected a total of 2,128 sentences from the three artifacts: 36 from iHRIS traditional requirements, 1,301 from iTrust use-case based requirements, and 791 from the OCS user guide.
5.2.2 Extracted Verb-Object Pairs

The iTrust use-case based requirements specification contained the most verb-object pairs (1,928), followed by the OCS user guide (1,479), then the iHRIS traditional requirements specification (106).

Table 11 summarizes the verbs that appeared the most in each software artifact. The most commonly occurring verb in the iHRIS traditional requirements specification is “allow”. For the iTrust use-case based requirements specification and the OCS user guide, the most commonly occurring verb is “is”, indicating that the artifacts frequently discuss system states (“A patient is a registered user”) or often use passive voice (“A prescription is created by a doctor”) when describing user activity.

<table>
<thead>
<tr>
<th>Software Artifact</th>
<th>All Verbs</th>
<th>Mandatory Log Event Verbs Only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Verb</td>
<td>Frequency</td>
</tr>
<tr>
<td>iHRIS traditional requirements</td>
<td>allow</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>edit</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>save</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>display</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>add</td>
<td>5</td>
</tr>
<tr>
<td>iTrust use-case based requirements</td>
<td>is</td>
<td>217</td>
</tr>
<tr>
<td></td>
<td>view</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>choose</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>select</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>enter</td>
<td>71</td>
</tr>
<tr>
<td>OCS user guide</td>
<td>is</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td>use</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>add</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>select</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>allow</td>
<td>52</td>
</tr>
</tbody>
</table>
In summary, the top five verbs appearing in the iHRIS traditional requirements were also the top five mandatory log event verbs. For the iTrust use-case based requirements specification, both mandatory log event verbs “view” and “enter” appeared in the set of most commonly used verbs in the entire document. For the OCS user guide, both mandatory log event verbs “add” and “allow” appeared in the set of most commonly used verbs in the entire document. The verb “select” also appeared commonly in both the iTrust use-case based requirements and the OCS user guide. Though many verbs frequently appeared throughout the natural language text, many of the most commonly used verbs in the use-case based requirements specification and user guide were not loggable and may clutter or hinder a software engineer’s ability to filter through the text to identify mandatory log events.

5.2.3 Classification Results

We documented the disagreements between the two researchers during the classification phase using the Cohen’s Kappa coefficient (κ). In statistics, κ is the measure of inter-rater agreement. A larger κ coefficient is considered an indicator of higher inter-rater agreement (Carletta 1996). Table 12, Table 13, and Table 14 present the confusion matrices of the initial classifications by the two researchers, before resolving disagreements and conferring with a third researcher. Section 5.3.3 provides insight into the differences in inter-rater agreement across the three systems.
Table 15 summarizes the final results of our classification, after resolving disagreements and conferring with a third rater. An average of 1.7 verb-object pairs were extracted per sentence. Of the verb-object pairs in each sentence, an average of 0.9 verb-object pairs were mandatory log events. Overall, 1,263 out of 2,128 (59%) sentences contained at least one verb-object pair that was a mandatory log event. Likewise, 2,060 out of 3,513 (59%) total verb-object pairs described mandatory log events.
Table 15: Summary of extracted verb-object pairs

<table>
<thead>
<tr>
<th>Software Artifact</th>
<th>Number of sentences</th>
<th>Number of verb-object pairs</th>
<th>Number of mandatory log event verb-object pairs</th>
<th>Average verb-object pairs per sentence</th>
<th>Average mandatory log events per sentence</th>
<th>Sentences that contain at least one mandatory log event</th>
</tr>
</thead>
<tbody>
<tr>
<td>iHRIS traditional requirements</td>
<td>36</td>
<td>106</td>
<td>96 (91%)</td>
<td>2.9</td>
<td>2.6</td>
<td>27 (75%)</td>
</tr>
<tr>
<td>iTTrust use-case based requirements</td>
<td>1301</td>
<td>1928</td>
<td>1217 (63%)</td>
<td>1.5</td>
<td>0.8</td>
<td>802 (62%)</td>
</tr>
<tr>
<td>OCS user guide</td>
<td>791</td>
<td>1479</td>
<td>747 (51%)</td>
<td>1.9</td>
<td>0.9</td>
<td>434 (55%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2128</strong></td>
<td><strong>3513</strong></td>
<td><strong>2060 (59%)</strong></td>
<td><strong>1.7</strong></td>
<td><strong>0.9</strong></td>
<td><strong>1263 (59%)</strong></td>
</tr>
</tbody>
</table>

5.3 Discussion

In this section, we discuss RQ5.1 and RQ5.2 and differences in the types of artifacts selected in the study. We also share observations about the top five most common verbs from Table 11, and about differences in inter-rater agreement between the two researchers.

5.3.1 Frequency of Mandatory Log Event Verb-Object Pairs

*RQ5.1: How often do descriptions of mandatory log events appear in natural language software artifacts?*

From Table 15, for iHRIS, each of the 36 total sentences contained on average 2.9 verb-object pairs. For the iHRIS traditional requirements specification, 75% of the total sentences in the requirements specification contained at least one mandatory log event.

From Table 15, for iTTrust, each of the 1,301 total sentences contained on average 1.5 verb-object pairs. For the iTTrust use-case based requirements specification, 62% of sentences contained at least one mandatory log event.
From Table 15, for OCS, each of the 791 total sentences contained on average 1.9 verb-object pairs. For the OCS user guide, 55% of sentences contained at least one mandatory log event.

Natural language requirements specifications and user guides that describe actions users can perform in the software can be valuable sources for inferring mandatory log events. In each of the three studied software systems, over half of all sentences contain at least one mandatory log event. On average, 59% of the full set of 2,128 sentences contains at least one mandatory log event verb-object pair, indicating that mandatory log events are frequently described in natural language software artifacts.

5.3.2 Comparing and Contrasting Software Artifacts

RQ5.2: What similarities and differences exist in the grammar and vocabulary used in different software artifacts?

We observed several differences among the three artifacts in this study. Even though both the iHRIS and iTrust artifacts are requirements specifications, the two artifacts employ two different styles of written requirements. iHRIS employs more traditional software requirements, very similar to those defined by IEEE-830 (1998). The traditional IEEE style suggests that requirements engineers write requirements from the perspective of the system and focus on what the software system should be able to accomplish. Traditional requirements appear in the format “The system shall…” The iTrust requirements specification, however, employs a use-case based style (Jacobson et al. 2011). With use
cases, requirements should be written from the perspective of the user (not the system) and focus on a user’s goals to perform specified actions within the software application.

Therefore, we expected the top five verbs in the entire iTrust artifact (see Table 11) to be similar to the top five mandatory log event verbs in the entire iTrust artifact, since the use-case based descriptions focus on actions the user should be able to perform in the software application. However, the iTrust use-case based requirements specification frequently uses non-loggable verbs when stating the software requirements, even though use-case based requirements typically focus on user actions. Instead, the top five mandatory log event verbs in the iHRIS traditional requirements are the exact same top five verbs that appear in the entire iHRIS artifact. One possible explanation of such similarity could involve the controlled nature of the traditional requirements style in the iHRIS artifact.

For instance, each of the 36 sentences in the iHRIS artifact follows the form “The system shall <action>…”, so the requirements are somewhat restricted to mainly verbs that relate to what the system should do. Natural language verbs such as “is”, “choose”, and “ignore” do not strictly relate to what the system can do, so they did not appear in the traditional-style iHRIS requirements. Instead, the traditional-style requirements in iHRIS consistently describe user actions in the form of “The system shall allow the user to <action>…”, which provides a consistent pattern for documenting actions that users can perform in the software application. In use-case based requirements, however, we did not observe any consistent patterns or constraints on how requirements are grammatically stated.
In use-case based requirements, sentences are allowed to freely follow any grammatical structure and pick from a larger variety of verbs, since use-cases are not constrained to describing only what the system shall do. The iTrust use-case based requirements often state preconditions or describe “states” of the system, in addition to specific user actions. For example, “A patient is a registered user of the iTrust Medical Records system.” In this sentence, the verb “is” describes a state of the system, but does not describe any action performed by a user.

The OCS artifact represents a user guide, not a software requirements specification. However, research suggested considering user guides (or user manuals) as requirements specifications (Berry et al. 2004) since the guides describe actions users should be able to perform in the software application, and, therefore, describe what the system should be able to do. In our study, the most commonly appearing verb in both the OCS user guide and the iTrust use-case based requirements specification was “is”, which suggests the OCS user guide also describes states or properties of the system (“OCS is designed to be a multilingual system”).

Grammar and vocabulary may affect the ability of software engineers to consistently identify mandatory log events. The somewhat constrained nature of traditional-style requirements specifications may make identifying mandatory log event verb-object pairs more straightforward since the requirements are limited to using verbs that describe what the system shall do or what the system shall allow users to do.
5.3.3 Differences in Inter-rater Reliability.

For this study, we compute the Cohen’s Kappa metric (Carletta 1996) for inter-rater reliability between the two researchers when classifying verb-object pairs as mandatory log events or not. For iTrust classifications, $\kappa=0.22$. For iHRIS classifications, $\kappa=0.62$. For OCS classifications, $\kappa=0.80$. The iTrust requirements specification was the first artifact examined and discussed, so inter-rater reliability was fairly low ($\kappa=0.22$), compared with inter-rater reliability in the iHRIS and OCS artifacts. Once we met to resolve disagreements in classification of mandatory log events for iTrust (see Section 5.1.5), the justifications discussed between the two researchers likely influenced classifications in the second artifact examined, the iHRIS traditional requirements specification.

As a result, inter-rater reliability for the iHRIS requirements specification increased to $\kappa=0.62$. In addition, the iHRIS requirements specification is a much shorter document (36 total sentences, compared to 1,301 sentences with the iTrust artifact) and uses consistent grammatical structure and terminology throughout, unlike the iTrust artifact. For example, the majority of sentences in the iHRIS requirements specification follow the form of “The system shall allow users to $<$action$>$…” or “The system shall $<$action$>$…”. Similarly, inter-rater reliability increased for the OCS user guide to $\kappa=0.80$. Discussions about disagreements in the iTrust and iHRIS classifications likely influenced classifications in the OCS artifact.

In summary, our results suggest that logging is very subjective, indicated by a low $\kappa$ for iTrust classifications ($\kappa=0.22$) where no previous discussions occurred between the two raters about what must be logged or why it must be logged. However, discussion of
disagreements in annotations helped develop mental guidelines for what is a mandatory log event, and agreement improved on subsequent classifications. We formulated a set of heuristics to help codify our informal mental guidelines to determine whether an action is a mandatory log event or not.

5.4 Heuristics for Determining Mandatory Log Events

*RQ5.3: What factors help decide whether an action is a mandatory log event?*

We use grounded theory analysis to empirically derive a set of twelve heuristics to help other software engineers determine whether a verb-object pair must be logged or not. Our analysis involved the documented discussions between the raters to resolve disagreements in classifications.

5.4.1 CRUD Actions

*Heuristic H1: If the verb involves creating, reading, updating, or deleting resource data in the software system, then the event must be logged.*

The most straightforward heuristic involves recording CRUD actions (create, read, update, delete), which are suggested in many academic, regulatory, and professional guidelines and specifications for implementing logging mechanisms (King and Williams 2013). The three software artifacts contain a total of 134 verb-object pairs that explicitly use the CRUD terminology.

*Heuristic H2: If the verb can be accurately rephrased in terms of creating, reading, updating, or deleting resource data in the software system, then the event must be logged.*
The unconstrained natural language used (specifically in use-case based requirements and user guides) may not easily map to the core CRUD actions. For example, “designate” appears in the iTrust use-case based requirements and OCS user guide. In these cases, we attempt to mentally rephrase the action using a core CRUD action. For example, “A patient designates a patient representative” can be mentally rephrased as “create a patient representative in the patient’s list of patient representatives”. Mentally rephrasing the action into a core CRUD action helps us determine that “designate” should be a mandatory log event. However, mental rephrasing must be carefully considered so that the meaning of the action does not change and that the rephrased action still falls within the scope of the software and intended functionality. In this study, the three software artifacts contain a total of 1,243 verb-object pairs that describe actions that can be mentally rephrased in terms of CRUD operations.

5.4.2 Read/View Actions

*Heuristic H3: If the verb implies the system displaying or printing resource data that is capable of being viewed in the user interface or on paper, then the event must be logged.*

In prior work (King et al. 2012; King and Williams 2013; King and Williams 2014), we discuss the importance of recording whenever a user views data, especially in a software system that manages sensitive data (2011b). The majority of classification disagreements between the two researchers involved actions that suggest reading or viewing of information. Specifically, the unconstrained natural language use-case based iTrust requirements and the OCS user guide use inconsistent terminology to describe “views” of data. For example, the
iTrust requirements specification often states “view”, “display”, “present”, “provide”, “read”, “see”, “show”, “list”, “analyze”, and “appear” interchangeably when describing the core action of a user accessing and viewing sensitive data in the system. The two researchers discussed differences between user-centric actions (“The user views immunizations for a patient”), system-centric actions (“The system lists immunizations for a patient”), and data-centric actions (“Immunizations for a patient appear”). After conferring with the third author, we determine that regardless of whether the action is user-centric, system-centric, or data-centric, if the data is displayed in the interface or printed and is therefore capable of being read, the action should be logged. In the three software artifacts in this study, a total of 397 verb-object pairs describe read-related actions.

5.4.3 Actions that Express Intent

Heuristic H4: If the verb expresses the intent to perform an action, such as “choose to”, “select to”, “plan to”, or “wish to”, then the intent event is not a mandatory log event.

Another primary source of disagreement between the two researchers involved actions such as “choose to create”, “select to delete”, “plan to remove”, and “wish to send”. The primary user action in “choose to create” involves creating data. Likewise, the primary user action in “select to delete” involves deleting data. The only mandatory log event verb-object pair for “choose to create an allergy” is <create, allergy>. The user cannot explicitly “choose” or “select” or “plan” or “wish” in the system, so these actions that express intent are not mandatory log events. In the three software artifacts in this study, a total of 351 verb-object pairs contain verbs that express intent and are not mandatory log events.
5.4.4 Actions that Express Permissions

*Heuristic H5:* If the verb expresses the granting or revocation of access privileges in the software system, then the event must be logged.

In “allow doctors to edit immunizations”, the edit action is classified as a mandatory log event. However, the term “allow” suggests the use of an access control security mechanism in the software system. In this example, and based on prior research on security events that should be logged (King and Williams 2013), we consider “allow” equivalent to “grant a user privilege” in an access control mechanism in the software. Granting or revoking a user privilege is a direct action the user may perform in the software. The two mandatory log event verb-object pairs for “allow doctors to edit immunizations” are: <allow to edit, immunizations> and <edit, immunizations>. In this study, a total of 126 verb-object pairs describe permissions that should be granted or prevented.

5.4.5 Context-critical Actions

*Heuristic H6:* If the verb is ambiguous, such as “provide” or “order”, context must be considered when determining if the event must be logged.

Some verbs may describe either mandatory log events or non-loggable events depending upon context. For example, “The conference organizer provides a schedule for a conference” suggests the act of creating a conference schedule in the software. However, the term “provide” can also describe a mandatory log event read/view action (such as “The system provides a list of medications”), as well as a non-loggable event (such as “The list of immunizations provides a means for doctors to view a patient’s vaccination history”).
Similarly, a doctor could “order lab procedures to be performed” (mandatory log event), or lab procedures could be “ordered alphabetically in a list” (not loggable). Context is critical in ambiguous cases where terms can potentially imply either a mandatory log event or a non-loggable event. In this study, we identify a total of 158 verb-object pairs that contain ambiguous verbs and require consideration of context to determine if the event must be logged.

**Heuristic H7:** If the verb describes an action that takes place outside the scope of the functionality of the software, then the event is not a mandatory log event.

Context is also important in cases where actions described are external or out of the scope of the software system. For example, the OCS user guide describes creating a PayPal business account. Since registering for a PayPal account happens outside the scope of the software system, the verb-object pair <create, PayPal account> is not loggable. We identify 314 verb-object pairs in this study that describe actions outside the scope of the software systems.

### 5.4.6 User Session Events

**Heuristic H8:** If the verb involves the creation or termination of a user session, then the event must be logged.

Throughout both the iTrust use-case based requirements specification and the OCS user guide, 94 total verb-object pairs described the need for users to authenticate into or log-out of the software system. An additional 6 verb-object pairs described the need for the user session
to timeout or terminate after a set amount of time for security reasons. Any action that involves the creation or termination of a user session must be logged.

5.4.7 Verbs that Describe States or Qualities, Not Events

Heuristic H9: If the verb describes a state or quality within the system, then the event is not a mandatory log event.

From Table 11, the most commonly occurring verb in both iTrust use-case based requirements specification and the OCS user guide is “is”. In the study, 253 total verb-object pairs describe states, not actions, in the software. For example, “A list of immunizations is available”, or “A patient is a registered user of iTrust”. A description of system states or qualities does not imply any user activity within the system and should not be logged.

5.4.8 Possession and Composition

Heuristic H10: If the verb describes possession or composition of a resource or quality, then the event is not a mandatory log event.

In the study, 57 total verb-object pairs describe possession of a resource or quality. For example, “The patient has a known interaction with a medication”, “The patients have dependents”, and “The row contains the doctor’s comments”. In these cases, neither <has, known interaction>, <has, dependent>, nor <contain, comments> is loggable.

5.4.9 Interface Actions

Heuristic H11: If the verb describes navigation or mechanical interaction with the software interface, then the event is not a mandatory log event.
The iTrust use-case based requirements specification and OCS user guide contain a total of 65 verb-object pairs that involve navigation. For example, “The doctor remains on the office visit page”, or “You can return to your account to see the progress of your submission”. Similarly, both software artifacts contain a total of 161 verb-object pairs that describe mechanical interaction with the software interface. For example, “The doctor types the patient’s name”, and “The author needs to click on Active Submissions”. In these cases, neither <type, patient name> nor <click, Active Submissions> is loggable since they do not describe what action the user is performing within the software. Instead, these verb-object pairs only describe how the user is interacting with the interface.

5.4.10 System Initialization and Configuration

**Heuristic H12:** If the verb describes initialization of the software or configuration of the software, then the event must be logged.

Only the OCS user guide described security events in which an administrative user initializes the software, upgrades the software, or installs new components. The OCS user guide contains a total of 8 verb-object pairs that describe system initialization and configuration. For example, “The Site Administrator can install additional locales as they become available”. In this sentence, the verb-object pair <install, locales> is a mandatory log event since it involves the administrative user configuring the system, which could potentially modify certain functionality or resources in the system.
5.4.11 Summary of Heuristics

Our heuristics facilitate classification of 3,372 (96%) out of 3,513 total verb-object pairs extracted from the three natural language software artifacts in this study as mandatory log events or not. Figure 2 presents a chart showing the increase in coverage of verb-object pairs as each heuristic is considered. H2 covers 1,243 (35%) of total verb-object pairs, which makes H2 the most applicable heuristic in our study. H3 covers an additional 397 (11%) of total verb-resource pairs. H4 covers an additional 351 (10%) of total verb-resource pairs. Overall, if a software engineer considered the set of \{H2, H3, H4, H7, H9, H11, and H6\}, roughly 84% of the total verb-object pairs would be covered. If a software engineer considers all 12 heuristics, 3,372 (96%) of the total verb-object pairs would be covered. As a result, 141 (4%) verb-object pairs do not fit under any of the proposed 12 heuristics. We do not observe any obvious patterns or consistencies among these 141 verb-object pairs to help justify additional reusable heuristics. A complete mapping of heuristics to verb-object pairs appears on the project website\(^{16}\).

\(^{16}\) http://go.ncsu.edu/nlplogging
5.5 Guidelines for Authors of Natural Language Software Artifacts

In this study, many classification disagreements between the researchers resulted from ambiguous and inconsistent use of terminology in the iTrust use-case based requirements specification and the OCS user guide. We propose a set of guidelines for requirements engineers to mitigate confusion and ambiguity to help software engineers who must perform natural language processing from software artifacts.

Use Consistent Terminology. In this study, we classified each of the following terms as a read/view action: “view”, “display”, “present”, “provide”, “read”, “see”, “show”, “list”, “analyze”, and “appear”. The two researchers frequently disagreed on whether verb-object pairs that contained an ambiguous verb like “provide” should be logged or not. Does the term “provide” describe the act of creating data, or does the term describe the act of displaying data so that the data can be viewed? However, if the author of the artifact consistently uses
the same term to describe a given action, many disagreements can be potentially avoided. Likewise, consistent terms would help reduce the number of verb-object pairs that are incorrectly classified as non-loggable because of ambiguity or confusion.

**Use Consistent Perspective.** In this study, we observed that core “read” actions were often described from three different perspectives within the same software artifact: (1) the user perspective (the user views | reads | sees | analyzes), (2) the system perspective (the system displays | presents | provides | shows | lists), and (3) the data perspective (the data appears). Using a consistent perspective when describing functionality of the software system helps constrain the terminology used, which helps reduce confusion and ambiguity when identifying mandatory log event verb-object pairs.

**Use CRUD Terminology.** In this study, we observed several terms that did not easily map to the basic create, read, update, delete actions identified in prior work [6]. For example, “manage” appeared in both the iTrust use-case based requirements specification and the OCS user guide. The term “manage” is ambiguous and could potentially mean either or all of create, read, update, or delete. Similarly, we observed the terms “make”, “indicate”, and “blog” in the OCS user guide. When describing actions that users may perform, authors should use CRUD terminology to mitigate ambiguity and explicitly describe the exact interactions with resource data. For example, use “create” instead of “make”, use “edit” or “add” (as appropriate) instead of “indicate”, and “create a blog entry” instead of “blog”. Otherwise, the reader may incorrectly infer the intended action and incorrectly classify the action as non-loggable.
5.6 Threats to Validity

Threats to external validity include the degree of representativeness of our studied software artifacts to real-world software artifacts. We address this threat by using real-world software artifacts for three open-source software systems. Another threat to external validity involves the possibility of over-fitting our heuristics to artifacts of a specific domain. To address this threat, we include natural-language software artifacts from three different domains: human resources management, healthcare, and scholarly conference management. Our methodology considers verbs and objects for identifying user activity, rather than relying on domain-specific terminology. Therefore, our methodology allows any natural language artifact that describes actions that users can perform in a software system to be considered, regardless of domain.

Threats to internal validity include the correctness of our extraction of verb-object pairs. We minimize this threat by having each rater validate and correct the list of verb-object pairs before annotating the pairs. An additional threat to internal validity includes the correctness of our annotations of mandatory log event verb-object pairs. We minimize the threat by designing the experiment such that two authors annotate each verb-object pair. In cases where two authors could not resolve disagreements, the third author broke the tie producing a majority vote. Furthermore, results of our annotations are publicly available on our project website.
5.7 Summary

Software logging has been a prevalent practice in production systems for decades. In addition to being valuable for software debugging and fault diagnosis, logging mechanisms can help mitigate repudiation threats and enable forensics after a security or privacy breach occurs. Research suggests logging is often subjective and arbitrary in practice (Yuan et al. 2012b). Although specifications exist to suggest how to implement logging mechanisms for user accountability (2013b; King and Williams 2013; 2014), no rigorous specification or systematic process exists to guide software engineers in determining what user activity should be logged for nonrepudiation. This work describes a systematic methodology to assist software engineers in identifying user activity that should be logged by: 1) extracting verb-object pairs from unconstrained natural-language software artifacts; and 2) proposing a set of 12 heuristics to identify verb-object pairs that describe mandatory log events. In addition, our heuristics facilitate classification of 3,372 (96%) of all verb-object pairs extracted from natural language software artifacts. Our results demonstrate that our 12 empirically-derived heuristics may help when identifying mandatory log events implied within unconstrained natural language software artifacts.
6. Using Heuristics to Identify MLEs – A Controlled Experiment

To help address the inadequacy and inconsistency of user activity logging mechanisms, in Chapter 5, we proposed a systematic heuristics-driven method for identifying mandatory log events (MLEs) from statements contained in natural language software artifacts. The method is driven by twelve empirically-derived heuristics. In this chapter, we design and execute a controlled experiment to evaluate whether our heuristics-driven method improves a software engineer’s ability to identify MLEs as compared with using existing industry standards.

*The objective of this study is to evaluate the use of our heuristics-driven method for identifying mandatory log events by software engineers in the context of software security.*

We conduct a controlled experiment (Shull et al. 2008) to compare the ability of software developers to identify MLEs using three different methods:

- **Standards-driven**: participants are guided by real-world logging specifications from existing healthcare and Payment Card Industry logging standards.
- **Resource-driven**: participants are guided by first identifying any sensitive resources, then identifying the actions users perform when interacting with the resources.
- **Heuristics-driven**: participants are guided by our method (see Chapter 5) that uses twelve empirically-derived heuristics to determine which user actions should be logged.

In addition, we consider whether the software artifact type (traditional requirements, use-case based requirements, or user manual) or readability score (lower or higher) of the natural
language software artifact may affect a software developer’s ability to identify MLEs. For this study, 103 graduate-level computer science students enrolled in an introductory software security course at North Carolina State University (NCSU) serve as our subjects. We conduct the study online using the Qualtrics17 online survey software.

Our research is guided by the following research questions:

- **RQ6.1:** How does the correctness of identifying whether a statement contains a mandatory log event differ among subjects using the standards-driven, resource-driven, and heuristics-driven methods?

- **RQ6.2:** How does the correctness of the identified mandatory log events differ among subjects using the standards-driven, resource-driven, and heuristics-driven methods?

- **RQ6.3:** How does the response time of subjects identifying mandatory log events differ among subjects using the standards-driven, resource-driven, and heuristics-driven methods?

Section 6.1 discusses our experiment planning. Section 6.2 describes the execution of our experiment. Section 6.3 describes how we synthesized the raw data collected from the experiment. Section 6.4 presents our analysis procedure. Section 6.5 presents our results. Section 6.6 presents discussion of our results. Section 6.7 describes lessons learned from our experiment. Section 6.8 presents threats to validity. Section 6.9 summarizes our controlled experiment on identifying MLEs.

17 http://www.qualtrics.com/
6.1 Experiment Planning

In this section, we outline the goal and hypotheses of our experiment, and then describe our experimental design, experimental units, and materials.

6.1.1 Goal, Metrics, and Hypotheses

Using the Goal-Question-Metric approach (Solingen et al. 2002), we define the following research objective:

*To evaluate the use of our heuristics-driven method for identifying mandatory log events by software engineers in the context of software security.*

Similarly, to answer our research questions, we state our null hypotheses as follows:

- **H₀¹**: Statement classification correctness is the same for all three methods used to identify MLEs.
- **H₀²**: The statement MLE identification correctness is the same for all three methods used to identify MLEs.
- **H₀³**: The response time of subjects on identifying MLEs for statements is the same for all three methods used to identify MLEs.

Table 16 summarizes the metrics used to address the research goals, questions, and hypotheses.

**Statement classification correctness** indicates whether the subject correctly classified a statement as containing MLEs or not. If the subject agreed with our oracle classifications, the value of the metric for the statement is 1. If the subject disagreed with our oracle, the value of the metric for the statement is 0.
Statement MLE identification correctness is, for a given statement, the proportion of unique subject responses that identified MLEs in our oracle relative to the total number of student responses, multiplied by the proportion of subject-identified MLEs in our oracle relative to the total number of MLEs in our oracle. This metric looks at both the overall identification rate, but also measures efficiency by penalizing students for giving redundant responses about the same MLE.

For example, suppose a subject provides 4 responses for a statement, but only 2 of the responses actually describe unique MLEs contained in our oracle. The proportion of subject responses that correctly identify MLEs would be 0.5 (2 subject responses each identify unique MLEs contained in the oracle ÷ 4 total subject responses). If our oracle contains 3 MLEs for the statement, then the proportion of MLEs identified by the subject would be 0.66 (2 MLEs identified by the student ÷ a total of 3 MLEs in the oracle for the statement). The statement MLE identification correctness for the statement would, therefore, be 0.33 (0.5 × 0.66). In contrast, if the 2 subject responses both identified the same MLE in our oracle, then the proportion of subject responses that correctly identify MLEs would be 0.25 (1 subject response identifies a unique MLE contained in our oracle ÷ 4 total subject responses). In this case, with duplicate subject responses for the same MLE, the statement MLE identification correctness for the statement would, therefore, be 0.165 (0.25 × 0.66).

The statement MLE identification correctness metric calculation penalizes subjects for over-specification of MLEs, or for repeating the same MLE multiple times. The value of the metric will equal 1 in only two cases: (1) if the subject responses for the statement exactly
identifies the set of MLEs in our oracle, with no omissions or extraneous responses; and (2) if both the subject and oracle agree that no MLEs are contained in the statement. We are primarily interested in statement MLE identification correctness for statements that do contain MLEs, based on our oracle.

**Statement response time** is the number of seconds a subject spent determining whether a statement contained MLEs, and if so, listing the MLEs for the statement. In Qualtrics, we use page submit time as the value of our response time metric. For each statement, we provide a limited amount of time (180 seconds) for subjects to identify whether the statement contains an MLE and, if so, to then identify each MLE.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Type</th>
<th>Metric Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statement Classification Correctness</strong></td>
<td>Indicates that the subject correctly (1) or incorrectly (0) identified whether the statement contained a MLE</td>
<td>Quantitative</td>
<td></td>
</tr>
<tr>
<td><strong>Statement MLE Identification Correctness</strong></td>
<td>Indicates the degree to which the subject responses exactly matched our oracle, with a higher value meaning the subject responses more correctly matched our oracle</td>
<td>Quantitative</td>
<td></td>
</tr>
<tr>
<td><strong>Statement Response Time</strong></td>
<td>Indicates the length of time (in seconds) the subject spent determining whether the statement contained MLEs, and if so, listing the MLEs for the statement</td>
<td>Quantitative</td>
<td>the number of seconds elapsed before the subject submits a response for a given statement</td>
</tr>
</tbody>
</table>
6.1.2 Experimental Design

For this experiment, we used a within-subject factorial design, taking baseline measurements (during the pre-period task) on each subject, and then taking the same measurements post-treatment (during the post-period task). A within-subject design allowed us to compare each subject’s measurements to himself or herself and account for significant variability between the subjects. We were primarily interested in finding significant differences in the responses between the pre- and post-period tasks, and determining whether those differences depended on the treatment.

After the pre-period task, subjects were randomly assigned to one of 18 treatment groups corresponding to the three factors: type of software artifact in which statements were contained (3 levels), readability of statements (2 levels), and method used to identify MLEs (3 levels). The factors are described in detail in Section 6.1.4.

The format of the experiment involved a pre-survey, orientation task, pre-period task, post-period (treatment) task, and post-survey. An overview of the experimental tasks appears in Figure 3. A complete copy of all experimental materials is available on the study website.
6.1.2.1 Pre-survey

In the pre-survey, subjects were first presented with sixteen questions asking how many years of academic and professional experience they have in computer science, software
engineering, software security, human resources software development, healthcare software development, conference management software development, natural language processing, and software logging. In addition, subjects are then presented with eight statements to rate (on a Likert-like scale of “Very Poor” to “Very Good”) their own ability to comprehend different aspects of English language and grammar, including the ability to understand spoken English, to identify parts-of-speech of words, and to use correct English grammar. Subjects are also asked how well they understood a 10-minute video lecture and 5-question multiple choice quiz on logging that was assigned as homework for the previous night, as well as how well they understood the role of logging mechanisms within the field of software security.

6.1.2.2 Tasks

Each subject completed an orientation task, pre-period task, and post-period task. Each task consisted of one instruction page at the start of the task, followed by one statement page for each statement assigned to the given task.

Orientation Task. Subjects first completed a orientation task. Since we used the Qualtrics survey system to administer the entire experiment, we included a brief orientation task to familiarize subjects with the layout of the Qualtrics interface. The orientation task presented the same set of instructions, time limitations, page layouts, and input fields as the pre- and post-period tasks.

Pre-period Task. After the orientation task, subjects completed a pre-period task, in which all subjects were provided the same set of statements from which to identify MLEs.
The pre-period task provided a baseline assessment of a subject’s ability to identify MLEs before the subject was randomly-assigned to a treatment group for the post-period task.

**Post-period Task.** During the post-period task, students were provided statements based upon their randomly-assigned treatment group.

### 6.1.2.3 Post-survey

After completing the treatment activity, subjects were presented with a post-survey. In the post-survey, we asked four retrospective questions related to the perceived difficulty of the pre-period task versus the post-period task. We also asked subjects to briefly describe the method they used when identifying MLEs. Finally, we asked for any additional comments or feedback about the study or study tasks. For subjects assigned to the heuristics-driven method treatment, we also requested feedback regarding the heuristics.

### 6.1.3 Experimental Units

Subjects in our experiment are computer science students enrolled in a 15-week graduate-level software security course at NCSU. We performed the experiment during the 12th week of the course. We required subjects to perform the tasks in the experiment during a seventy-five minute class period as course participation credit, but subjects could opt-out of allowing their responses to be included for the purpose of this research. Of the 110 students enrolled in the course, 103 students consented to participate in the study. Section 6.2.2 further discusses how students prepared for the experiment.

### 6.1.4 Experiment Materials

In our experiment, we control three factors, summarized in Table 17:
- Type of the natural language software artifact from which subjects are asked to identify MLEs. Since our heuristics-driven method involves identifying verbs and objects from natural language statements, we evaluate whether the type of software artifact affects MLE identification.

- Readability of the set of statements contained in the natural language software artifact from which subjects are asked to identify MLEs. Since our heuristics-driven method involves identifying verbs and objects from natural language statements, we evaluate whether the readability of software artifact affects MLE identification.

- Method used to identify MLEs. We evaluate whether a heuristics-driven method improves MLE identification compared to two alternative methods for identifying MLEs: standards-driven and resource-driven methods.

Table 17: Summary of Experimental Factors and Levels

<table>
<thead>
<tr>
<th>Factor</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of natural language software artifact</td>
<td>Traditional requirements specification</td>
</tr>
<tr>
<td></td>
<td>Use-case based requirements specification</td>
</tr>
<tr>
<td></td>
<td>User manual</td>
</tr>
<tr>
<td>Readability of statements</td>
<td>Simple</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
</tr>
<tr>
<td>Method used to identify MLEs</td>
<td>Standards-driven</td>
</tr>
<tr>
<td></td>
<td>Resource-driven</td>
</tr>
<tr>
<td></td>
<td>Heuristics-driven</td>
</tr>
</tbody>
</table>

6.1.4.1 Type of the Natural Language Software Artifact

To select the software artifact types to use in our experiment, we use the software artifacts contained in our previously-created oracle (King et al. 2015b) of MLEs in human resources, healthcare, and conference management software artifacts:
- **iHRIS**\(^{18}\) v4.2: Open Source Human Resources Information Solutions.
  - Software Artifact: traditional requirements specification.
  - According to the iHRIS community, 15 countries are using the software, with more than 675,000 health worker records currently supported in iHRIS.

- **iTrust**\(^{19}\) v18: Open Source Electronic Health Record System.
  - Software Artifact: use-case-base requirements specification
  - An electronic health record (EHR) system developed and maintained by undergraduate software engineering students at North Carolina State University and used by many researchers and educators as a test-bed (Meneely et al. 2012).

- **OCS**\(^{20}\) (Open Conference Systems) v2.3.6: Open Source Scholarly Conference Management System.
  - Software Artifact: user manual
  - A conference management system developed and maintained by the Public Knowledge Project (PKP), a multi-university initiative developing free open-source software and conducting research to improve quality of scholarly publishing.

---

\(^{18}\) [http://www.ihris.org](http://www.ihris.org)

\(^{19}\) [http://agile.csc.ncsu.edu/iTrust](http://agile.csc.ncsu.edu/iTrust)

\(^{20}\) [https://pkp.sfu.ca/ocs/](https://pkp.sfu.ca/ocs/)
6.1.4.2 Statement Selection based on Readability

To select statements to include in our experiment based on readability scores, we first split the natural language software artifacts into individual sections so that the statements in each section all describe related functionality. For the iHRIS traditional requirements specification, statements were hierarchically organized by topic. We sectioned the iHRIS statements based on the existing hierarchical organization topics. For the iTrust use-case based requirements specification, we sectioned the statements by individual use case. For the OCS user manual, we sectioned statements by first-order headings.

Next, we calculated\textsuperscript{21} readability metrics for each section of statements in each source software artifact. Since readability metric algorithms measure different features of written language (for example, syllables per word, words per sentence, ignoring compound words, etc.), instead of relying on a single readability metric, we used scores from three readability metrics: Fleish-Kincaid (Kincaid and Others 1975), Automated Readability Index (ARI) (Smith and Senter 1967), and SMOG grade(McLaughlin 1969). For our experiment, we calculated Fleish-Kincaid, ARI, and SMOG scores for each candidate section of statements from each source artifact. Next, we sorted the scores in ascending order from simplest to most complex. For our experiment, we selected a section (or set of sections) with overall simple readability, and a section (or set of sections) with overall complex readability. To ensure consistency in the number of statements selected for each software artifact type and readability, we select multiple sections for TraditionalRequirements-Simple, 

\textsuperscript{21} To calculate readability metric values, we use the calculators provided by https://readability-score.com/
Traditional Requirements-Complex, UseCase-Simple, UseCase-Complex, UserManual-Simple, and UserManual-Complex. For a baseline set of statements, we select a set of statements from the iTrust use-case-based requirements specification since the iTrust specification contained the most sections available from which to select.

Because of differences in the number and grammatical style of statements available from which to select, we were unable to select sections with identical readability scores across each of the three types of software artifacts. For example, readability scores of traditional requirements statements ranged from 76.8 to 89.1 for Fleish-Kinkaid; 2.5 to 4.7 for ARI; and 2.3 to 8.5 for SMOG. Table 18 summarizes the software artifact statements used in our experiment.

<table>
<thead>
<tr>
<th>Artifact Type</th>
<th>Readability</th>
<th>Number of Statements</th>
<th>Flesch-Kinkaid Reading Ease (0-100, where 100 is simple)</th>
<th>Automated Readability Index (Grade level, where 0 is simple)</th>
<th>SMOG Grade (Grade level, where 0 is simple)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Mid-range</td>
<td>17</td>
<td>73.1</td>
<td>12.1</td>
<td>11.6</td>
</tr>
<tr>
<td>(from a Use-case-based Requirements Specification)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traditional Requirements Specification</td>
<td>Simple</td>
<td>14</td>
<td>89.1</td>
<td>2.5</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>13</td>
<td>76.8</td>
<td>4.7</td>
<td>8.5</td>
</tr>
<tr>
<td>Use-case-based Requirements Specification</td>
<td>Simple</td>
<td>23</td>
<td>77.4</td>
<td>1.9</td>
<td>4.8</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>20</td>
<td>44.5</td>
<td>13.9</td>
<td>13.7</td>
</tr>
<tr>
<td>User Manual</td>
<td>Simple</td>
<td>13</td>
<td>64.5</td>
<td>7.7</td>
<td>8.3</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>16</td>
<td>33.7</td>
<td>13.4</td>
<td>13.5</td>
</tr>
</tbody>
</table>
6.1.4.3 Method Selection

To select the different methods subjects use to identify MLEs, we begin by including the heuristics-driven method at the center of our experiment. Next, we define a standards-driven method that reflects real-world software development by providing only actual industry standards from healthcare (2013c) and the Payment Card Industry (2010a) as guidance for subjects. Finally, since our heuristics-driven method focuses on identification of verbs, we define a resource-driven method that involves first identifying resources being acted upon, then identifying the actions performed against the resource. Neither the standards-driven or resource-driven methods provide heuristics for guidance. The full set of method descriptions can be found on the study website and in Table 19.
<table>
<thead>
<tr>
<th>Method</th>
<th>Instructions Provided</th>
</tr>
</thead>
</table>
| Standards-driven | **Industry Standard 1:** An audit log is a record of actions (queries, views, additions, deletions, changes) performed on data by users. Actions should be recorded at the time they occur. These actions include user authentication, user or system-directed signoff, data access to view, and receipt of data content from external provider/practitioner.  

**Industry Standard 2:** The software must create a secure audit record each time a user:  
- accesses, creates or updates sensitive data via the software;  
- overrides the consent directives of a user via the software;  
- accesses, via the software, data that is locked or masked by instruction of a user; or  
- accesses, creates or updates registration data on a software user.  

**Industry Standard 3:** Implement audit trails to link all access to system components to each individual user. Implement automated audit trails for all system components to reconstruct the following events:  
- all individual user accesses to data  
- all actions taken by individuals with root or administrative privileges  
- access to all audit trails  
- invalid logical access attempts  
- use of and changes to identification and authentication mechanisms -- including, but not limited to creation of new accounts and elevation of privileges -- and all changes, additions, or deletions to accounts with root or administrative privileges  
- initialization, stopping, or pausing of the audit logs  
- creation and deletion of system level objects |
| Resource-driven | **Technique:** In grammar, nouns are the fundamental construct that express objects, while verbs generally express actions being executed against an object. To identify user activity in natural language software artifacts:  
1. First identify all nouns in the given sentence. For example, "Doctors prescribe medications for patients." Three nouns appear in this sentence: doctors, medications, and patients.  
2. Then identify any verbs that act upon the nouns. In the sentence "Doctors prescribe medications for patients," no verbs act upon the nouns "doctors" or "patients". However, the verb "prescribe" describes an action performed on medication data.  
3. Finally, determine if the action taken upon the object is loggable. Since neither the nouns "doctors" or "patients" are acted upon by a verb, neither are loggable. However, "prescribe medications" describes the creation of data in the system and may be considered loggable. |


<table>
<thead>
<tr>
<th>Table 19 Continued</th>
</tr>
</thead>
</table>

**Heuristics-driven Technique:** In grammar, verbs are the fundamental construct that express an action being executed against an entity (indicated by an object). We express a verb-object pair as a tuple of the form \(<verb, object>\). To extract verb-object pairs, we consider the following guidelines for each statement:

- **Explicitly stated verb-object pairs.** Extract any verbs contained in the sentence, then identify any objects being acted upon by the verb.
  - Example 1: “Doctors prescribe medications.”
  - verb-object pair: \(<\text{prescribe}, \text{medication}>\)

- **Implied verb-object pairs.** Extract any words in the sentence whose lemma is a verbal (e.g., gerunds, participles, and infinitives are verbals that function as nouns in a sentence), then identify any objects being acted upon by the verbal.
  - Example 2: “Creating a patient…”
  - verb-object pair: \(<\text{create}, \text{patient}>\)
  - Example 3: “The submitted proposal…”
  - verb-object pair: \(<\text{submit}, \text{proposal}>\)

- **Compound verb-object pairs.** For any sentence that contains compound verbs or more than one object for a single verb, we document multiple verb-object pairs to consider each individual combination of verb and object.
  - Example 4: “Doctors prescribe and update medications”
  - verb-object pair: \(<\text{prescribe}, \text{medication}>\)
  - verb-object pair: \(<\text{update}, \text{medication}>\)

Once all verb-object pairs are identified for a given statement, consider the following set of heuristics to help determine if the verb-object pair should be logged or not:

\footnote{The set of heuristics appears in Section 5.4}
6.2 Experiment Execution

In this section, we discuss subject preparation and experiment execution procedure.

6.2.1 Preparation

Before participating in the experiment, subjects were required to watch a 10-minute video lecture on software logging in the context of software security. The video lecture did not reveal any techniques for identifying MLEs. Instead, the video provided overall descriptions of why logging is important for nonrepudiation, and outlined common weaknesses and vulnerabilities with logging mechanisms. Subjects also completed a 5-question multiple-choice quiz on logging that was assigned as homework for the previous night.

6.2.2 Procedure

At the beginning of the experiment, subjects were informed they were to complete a set of tasks for classroom participation credit. However, subjects were also informed of the option to consent to the inclusion of their task responses for the purpose of this study. Subjects were provided a URL to the experiment website, hosted by Qualtrics. Once the URL was provided to subjects, they were given until the end of the seventy-five minute class period to complete their responses to the study. Data collection and randomization of subjects into individual artifact type/readability/method groups were managed by the Qualtrics system. To ensure confidentiality, subjects were physically handed a random six-digit access code on a slip of paper at the beginning of the class period. During the study, subjects were
asked to enter the random six-digit access code, which ensured that personally identifiable information was never collected.

For each task, subjects were presented with two types of pages. First, each task began by presenting a page of instructions. After the instructions page, subjects were presented one page for each statement in the set of statements for the given task.

### 6.2.3 Instruction Page

For the orientation, pre-period, and post-period tasks, we first presented instructions for how to respond to the questions during the task. The instructions provided links to any additional reference materials. For example, during the post-period task, the instruction page described the method for identifying MLEs that was assigned to the subject. Subjects could spend as much time as necessary reading the instructions and any reference materials, with the understanding that they complete the entire experiment within the overall 75-minute class period. The instructions also informed subjects of a three minute time limit for providing a response on each statement page before being auto-advanced to the next statement. We imposed a three minute time limit for each statement to keep all subjects on pace to complete the task within the 75-minute time allotted for the experiment.

### 6.2.4 Statement Pages

After the instruction page, subjects were presented with one page for each statement in the set of statements assigned for the given task. The statement page presented the text of the current statement and a question asking if the subject thought any MLEs were described in the statement. If the subject indicated that the statement described a MLE, the subject was
then presented with a table of text fields for identifying up to ten MLEs and brief justifications for why the subject thought each MLE should be logged. In addition, the statement page displayed a countdown timer showing how much time was remaining before the page automatically submitted and advanced to the next statement. Figure 4 provides a sample statement page.

![Sample statement page](image)

Figure 4: Sample statement page, with a countdown timer, options for selecting whether the statement describes any MLEs, and (if so), text fields for identifying up to 10 MLEs
6.3 Synthesizing Raw Response Data

In this section, we describe our process for managing response data, classifying subject responses, resolving classification differences, and calculating our metric values. While synthesizing the raw response data, we removed subjects from the study if they copied-and-pasted the source statement as their response for MLE identification. For this experiment, 2 subjects were removed because they copy-and-pasted the text of the source statements, without actually identifying any MLEs, resulting in a total of 101 total subjects in our experiment.

6.3.1 Data Management

We created an individual Excel workbook to manage data for each individual subject response. Each workbook contained two sheets: a sheet for managing pre-period task response data, and a sheet for managing post-period task response data. On each data sheet, we listed each of the source software artifact statements assigned for the given task. For each source software artifact statement, we included two columns:

- Subject responses. In this column, we listed all of the responses provided by the subject for the given statement
- Oracle Verb-Object Pairs. In this column, we listed all of the verb-object pairs that appeared in our previously-created oracle (King, Pandita, and Williams 2015) for the given statement. Verb-object pairs that represent MLEs were explicitly highlighted to visually distinguish them from verb-object pairs that appeared as not loggable in the oracle.
6.3.2 Classifying Subject Responses

Two researchers individually classified each subject response as one of the following:

- Appears as loggable in the oracle: the response correctly identified a verb-object pair classified as an MLE in our oracle
- Appears as not loggable in the oracle: the response identified a verb-object pair classified as not an MLE in our oracle
- Not in the oracle, but loggable: The subject identified an MLE that is not contained in our oracle
- Not in the oracle, and not loggable: The subject identified an action that is not contained in our oracle, but the identified action was not a MLE.
- Not a description of an activity: The subject provided responses that did not describe actions. For example, “username” and “timestamp” are data elements that should be logged for each log entry, but they do not represent an action that must be logged.
- Activity is not described in the given statement: The subject provided a description of an activity that was neither explicitly nor implicitly described in the provided statement. For example, “login user” is not described in the statement “A user views data.”

For each subject response classified as Appearing as loggable in oracle or Not in the oracle but loggable, we further classified the completeness of the response as one of the following:
- Identified both the action and the resource being acted upon
- Identified only the action being performed
- Identified only the resource being acted upon
- Identified neither the action nor the resource

We also individually classified each of the MLE items in the oracle column as one of:

- Identified by the subject: at least one subject response described the given MLE
- Not identified by the subject: no subject response described the given MLE

### 6.3.3 Resolving Classification Disagreements

After individually classifying each subject response and each oracle MLE, we computed inter-rater agreement using the Cohen’s Kappa metric. A larger $\kappa$ coefficient is considered an indicator of higher inter-rater agreement (Carletta 1996). Our overall Cohen’s Kappa score for subject response classifications was $\kappa=0.70$, for completeness-of-response classifications was $\kappa=0.77$, and for oracle event classifications was $\kappa=0.78$. A breakdown of additional Cohen’s Kappa values for each pre-period and post-period task are available on our project website.

For disagreements in classifications, the two researchers met to justify their decisions. After discussing and resolving all disagreements, we produced a final set of classifications for each subject.

### 6.3.4 Metric Calculations

For each statement in the pre- and post-period tasks, we calculated the value of statement classification correctness, statement MLE identification correctness, and statement response
time based on Table 16: Summary of Metrics for Evaluating Subject Responses. Table 16. In a spreadsheet, we then recorded the metric values for each statement for each subject. Each row of the spreadsheet contained the following columns of data:

- **Subject ID**: the unique subject identifier.
- **Task**: the task in which the statement appeared: pre-period, or post-period.
- **Statement Type**: the type of artifact the statement was contained in: a traditional requirements specification; use-case based requirements specification; or user manual.
- **Readability**: the readability category of the set of statements the statement was contained: baseline for pre-period task statements; simple readability or complex readability for post-period task statements.
- **Method**: the method randomly-assigned for the subject to use when identifying MLEs: standards-driven; resource-driven; or heuristics-driven.
- **Statement ID**: the unique identifier for the given statement
- **Has MLEs in the Oracle**: 0, if the oracle does not contain any MLEs for the given statement; 1, if the oracle does contain MLES for the given statement.
- **Statement Classification Correctness**: the value of the statement classification correctness metric for the given statement.
- **Statement MLE Identification Correctness**: the value of the statement MLE identification correctness metric for the given statement.
6.4 Analysis Methodology

We first describe our procedure for analyzing statement classification correctness. Next, we describe our procedure for analyzing statement MLE identification correctness. Finally, we describe our procedure for analyzing statement response time. All analyses were performed using either SAS 9.4 or JMP\textsuperscript{22} Version 12.

6.4.1 Analyzing Statement Classification Correctness

To address \( H_{01} \), we determine whether any of the three methods for identifying MLEs significantly changed the subject’s rate of detection of loggable or non-loggable events between the pre- and post-period tasks. Therefore, for each subject, we averaged statement classification correctness across the four categories:

- Pre-period task statements that contain MLEs in the oracle
- Pre-period task statements that do not contain MLEs in the oracle
- Post-period task statements that contain MLEs in the oracle
- Post-period task statements that do not contain MLEs in the oracle

We fit a binomial generalized linear mixed model, which was preferred over a linear mixed model because of the large number of perfect classifications (where the average classification correctness equaled 1). This model was also able to incorporate different

\textsuperscript{22} http://www.jmp.com
variances for groups due to an unequal number of statements, which was especially true for the three groups UserManual-Simple, TraditionalRequirements-Simple, and TraditionalRequirements-Complex.

Since we are performing an exploratory study to determine which effects may affect a subject’s ability to correctly classify a statement as containing MLEs, we consider any effect with a p-value less than 0.1 to be a potentially significant effect for further exploration of the effect using pairwise contrasts.

The fixed effects of the model included all main effects and interactions for five factors: task period (pre- and post-), software artifact type, readability, method used to identify MLEs, and whether the statement contained MLEs in the oracle (yes or no). Random effects were included for each subject. We were uninterested in main effects for software artifact type, readability, and method, since these effects are averaged across the task periods and differences should only be significant during the post-period. Therefore, the primary focus of our analysis was to find significant interactions involving task period.

For the interaction effects found to be significant, we investigated the change in the statement classification correctness between the pre- to post-period tasks for each level of the other factors involved in the interaction. For example, if a significant interaction was found between task period and artifact type, we looked at this difference for all three levels of artifact type. T-tests with Tukey adjustments were used to test if these changes were significantly different from 0. Changes were deemed significant if the adjusted p-values were less than 0.01.
6.4.2 Analyzing Statement MLE Identification Correctness

To address $H_{02}$, we determine whether students can correctly identify the action and resource components of the MLE, itself. Therefore, we look at statement MLE identification correctness. For each subject in both the pre- and post-period tasks, we averaged statement MLE identification correctness for only the statements that contain MLEs in the oracle. We fit a linear mixed model to this data with the same fixed and random effects as we did in Section 6.4.1. While statement MLE identification correctness ranges from 0 to 1, 91% of subjects had an average MLE identification correctness greater than 0 and less than 1. Therefore, the linear mixed model is an appropriate approximation. A normal QQ-plot of the studentized residuals was used to verify the normality assumption.

The number of loggable statements was comparable across all the artifact-type/readability groups, so we did not need to account for unequal variances due to differing numbers of loggable statements. Since, at this stage of analysis, we are doing preforming an exploratory study to determine which effects may affect a subject’s ability to correctly identify MLEs for a statement, we consider any effect with a p-value less than 0.1 to be a potentially significant effect for further exploration of the effect using pairwise contrasts.

For the interaction effects found to be significant, we investigated the change in the proportion of MLEs identified between the pre- to post-period task for each setting of the other factors involved in the interaction. T-tests with Tukey adjustments were used to test if these changes were significantly different from 0. Changes were deemed significant if the adjusted p-values were less than 0.01.
6.4.3 Analyzing Statement Response Time

To address H03, we compare the mean page submit times for each subject. For each statement, students had up to 180 seconds (3 minutes) to determine whether a statement contained MLEs and, if so, to list the MLEs. For each subject, we averaged response time across the four categories:

- Pre-period task statements that contain MLEs in the oracle
- Pre-period task statements that do not contain MLEs in the oracle
- Post-period task statements that contain MLEs in the oracle
- Post-period task statements that do not contain MLEs in the oracle

We calculated the mean response time and standard deviation for each of the four groups of values above. Since the standard deviations were unequal across the four groups of values, we fit the data using a weighted linear mixed model with weights equal to the inverse standard deviation of response times for each participant in the pre- and post-period tasks.

The fixed effects of the model for response time included all main effects and interactions for five factors: task period (pre- and post-), software artifact type, readability, method used to identify MLEs, and whether the statement contained MLEs in the oracle (yes or no). Random effects were included for each subject. Just like in Section 6.4.1, the primary focus of our analysis was to find significant interactions involving task period.

For the interaction effects found to be significant, we investigated the change in the response time between the pre- to post-period tasks for each level of the other factors involved in the interaction. T-tests with Tukey adjustments were used to test if these changes
were significantly different from 0. Changes were deemed significant if the adjusted p-values were less than 0.01.

6.5 Results

In this section, we first summarize subject pre-survey responses. In addition, we answer our research questions regarding the statement classification correctness, statement MLE identification correctness, and response time of identifying MLEs. Table 20 summarizes the number of subjects in each treatment group during the post-period task.

<table>
<thead>
<tr>
<th>ArtifactType</th>
<th>Readability</th>
<th>Standards-driven</th>
<th>Resource-driven</th>
<th>Heuristics-driven</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional Requirements</td>
<td>Simple</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>UseCaseBased Requirements</td>
<td>Simple</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>UserManual</td>
<td>Simple</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>34</td>
<td>34</td>
<td>33</td>
<td>101*</td>
</tr>
</tbody>
</table>

*2 subjects were removed because they copy-and-pasted the source statement as their response, without actually identifying any MLEs.

6.5.1 Subject Experience

We computed the mean, median, and standard deviations of all metrics for academic and professional experience and English comprehension of subjects (see Section 6.1.3) for all combinations of the three factors (software artifact type, readability, and method used to identify MLEs), and found the values to be similar across all groups. The overall average for academic performance in computer science, software engineering, and software security was 5 years, 2.5 years, and 1 year, respectively. Subjects had minimal experience in human
resources, healthcare, and conference management software development, with averages of 0.2 years. Similar outcomes were observed with professional experience, but the averages tended to be lower. For example, the overall average for professional experience in computer science was 2 years. The consistency in scores across groups implies that initial ability of subjects is not a confounding factor. For example, no one group is significantly better at reading comprehension than another.

The pre-survey metrics were initially included in the task metrics analysis models, but none revealed any meaningful, significant relationships. Any significant relationship that was found with the procedure was due to outliers of the pre-survey metrics. For example, one subject claimed their academic experience in computer science was 15 years. Therefore, we did not consider the pre-survey metrics as potential covariates in the analysis of the task metrics.

6.5.2 Statement Classification Correctness

Figure 5 presents a plot of statement classification correctness for subjects using each method for identifying MLEs. When fitting a factorial model, we remove insignificant interactions that involve a large number of factors. Removing insignificant interactions led us to a model that included all main effects and two-factor interactions. Fixed effect tests of all effects that involve method, along with all potentially significant effects, are shown in Table 21. The full table of fixed effect tests can be found on our study website23.

23 http://go.ncsu.edu/loggingexperiment
Figure 5: Statement classification correctness for each method of identifying MLEs

Table 21: Type III Tests of Fixed Effects, including effects involving method and other potentially significant effects for statement classification correctness

<table>
<thead>
<tr>
<th>Effect</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>65.08</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>OracleHasMLEs</td>
<td>66.92</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>Task*OracleHasMLEs</td>
<td>131.55</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>Task*Readability</td>
<td>9.88</td>
<td>0.002</td>
</tr>
<tr>
<td>Task*ArtifactType</td>
<td>5.62</td>
<td>0.004</td>
</tr>
<tr>
<td>ArtifactType*OracleHasMLEs</td>
<td>3.43</td>
<td>0.034</td>
</tr>
<tr>
<td>Method*OracleHasMLEs</td>
<td>1.29</td>
<td>0.276</td>
</tr>
<tr>
<td>Task*Method</td>
<td>0.92</td>
<td>0.399</td>
</tr>
<tr>
<td>Readability*Method</td>
<td>0.70</td>
<td>0.500</td>
</tr>
<tr>
<td>ArtifactType*Method</td>
<td>0.70</td>
<td>0.594</td>
</tr>
<tr>
<td>Method</td>
<td>0.51</td>
<td>0.601</td>
</tr>
</tbody>
</table>
The most highly significant effects only involved task period and whether the statement contained an MLE in the oracle. The interaction between task period and whether the statement contained an MLE in the oracle was the strongest effect. Classification correctness averages, as well as their differences, for all four of the combinations appear in Table 22.

| Table 22: Classification correctness averages for Task*OracleHasMLEs |
|-----------------------------|------------------------|---------------------|
|                            | Task Period            |                     |
|                            | Pre | Post | Diff Means |
| OracleHasMLEs              |     |      |            |
| Yes                        | 0.62 | 0.55 | -0.07**    |
| No                         | 0.55 | 0.86 | 0.31**     |

** adjusted p-value < 0.01; * adjusted p-value < 0.05

Since we investigated any effect with a p-value less than 0.10, other possibly significant effects were the interactions Task*ArtifactType and Task*Readability, the latter of which was expected since readability only played a role in the post-period task. Classification correctness averages, as well as their differences, for Task*ArtifactType and Task*Readability are shown in Table 23 and Table 24, respectively.

<table>
<thead>
<tr>
<th>Table 23: Classification correctness averages for Task*ArtifactType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Period</td>
</tr>
<tr>
<td>Task Period</td>
</tr>
<tr>
<td>ArtifactType</td>
</tr>
<tr>
<td>TraditionalRequirements</td>
</tr>
<tr>
<td>UseCaseBasedRequirements</td>
</tr>
<tr>
<td>UserManual</td>
</tr>
</tbody>
</table>

** adjusted p-value < 0.01; * adjusted p-value < 0.05
Table 24: Classification correctness averages for Task*Readability

<table>
<thead>
<tr>
<th>Readability</th>
<th>Task Period</th>
<th>Pre</th>
<th>Post</th>
<th>Diff Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td></td>
<td>0.58</td>
<td>0.78</td>
<td>0.20**</td>
</tr>
<tr>
<td>Complex</td>
<td></td>
<td>0.59</td>
<td>0.69</td>
<td>0.10**</td>
</tr>
</tbody>
</table>

** adjusted p-value < 0.01; * adjusted p-value < 0.05

H₀₁: Statement classification correctness is the same for all methods used to identify MLEs.

*We fail to reject H₀₁. We observe no meaningful statistical difference in statement classification correctness for subjects using the standards-driven, resource-driven, and heuristics-driven methods.*

### 6.5.3 Statement MLE Identification Correctness

Figure 6 presents a plot of statement MLE identification correctness for subjects using each method for identifying MLEs. The full factorial model was appropriate in this scenario. Fixed effect tests of effects involving method, along with all potentially significant effects, are shown in Table 25.
Figure 6. Statement MLE identification correctness for subjects using each method for identifying MLEs

Table 25: Type III Tests of Fixed Effects, including effects involving method and other potentially significant effects for statement MLE identification correctness

<table>
<thead>
<tr>
<th>Effect</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task*Readability</td>
<td>9.37</td>
<td>0.003</td>
</tr>
<tr>
<td>Task<em>ArtifactType</em>Readability*Method</td>
<td>3.64</td>
<td>0.009</td>
</tr>
<tr>
<td>Task*Method</td>
<td>2.26</td>
<td>0.111</td>
</tr>
<tr>
<td>Task<em>ArtifactType</em>Method</td>
<td>1.47</td>
<td>0.220</td>
</tr>
<tr>
<td>Task<em>Readability</em>Method</td>
<td>1.44</td>
<td>0.242</td>
</tr>
<tr>
<td>Method</td>
<td>0.84</td>
<td>0.436</td>
</tr>
<tr>
<td>ArtifactType<em>Readability</em>Method</td>
<td>0.87</td>
<td>0.487</td>
</tr>
<tr>
<td>Readability*Method</td>
<td>0.11</td>
<td>0.900</td>
</tr>
<tr>
<td>ArtifactType*Method</td>
<td>0.11</td>
<td>0.977</td>
</tr>
</tbody>
</table>
For statement MLE identification correctness, we found a significant interaction with task*readability. However, the task*readability interaction does not involve the method used to identify MLEs, which is what we are interested in. We found a potentially significant four-way interaction between task period, software artifact type, readability, and method for identifying MLEs. The interaction suggests that the change from the pre- to post-period task might be affected by method, but the difference may depend on the artifactType/readability combination. Therefore, t-tests with Tukey-adjusted p-values were used to test the change in average MLE identification correctness from the pre- to post-period for all 18 treatment groups. The full table for the MLE identification correctness averages across all 18 artifactType*readability*method combinations can be found on our project website. We did not observe any significant differences in any of the artifactType*readability*method groups.

H₀₂: The statement MLE identification correctness is the same for all methods used to identify MLEs.

We fail to reject H₀₂. We do not observe any significant difference in statement MLE identification correctness between subjects assigned to standards-driven, resource-driven, and heuristics-driven methods for identifying MLEs.

6.5.4 Response time of Identifying MLEs

A plot of all response times for subjects in each method is provided in Figure 7. When fitting the factorial model, we remove insignificant interactions that involve a large

24 http://go.ncsu.edu/loggingexperiment
number of factors. Removing insignificant interactions led us to a model that included all main effects and three-factor interactions. Fixed effect tests of all potentially significant effects are shown in Table 26.

Figure 7: Plot of response times for subjects in each method
Table 26: Type III Tests of Fixed Effects, including Effects involving Method and Other Potentially Significant Effects for Response Time

<table>
<thead>
<tr>
<th>Effect</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>261.38</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>OracleHasMLEs</td>
<td>290.05</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>Task<em>ArtifactType</em>Readability</td>
<td>7.31</td>
<td>&lt;0.0001</td>
</tr>
<tr>
<td>Task <em>ArtifactType</em>OracleHasMLEs</td>
<td>2.84</td>
<td>0.062</td>
</tr>
<tr>
<td>Readability</td>
<td>3.48</td>
<td>0.064</td>
</tr>
<tr>
<td>Readability*Method</td>
<td>1.95</td>
<td>0.146</td>
</tr>
<tr>
<td>Task<em>Readability</em>Method</td>
<td>1.94</td>
<td>0.147</td>
</tr>
<tr>
<td>Task*Method</td>
<td>1.56</td>
<td>0.214</td>
</tr>
<tr>
<td>ArtifactType<em>Readability</em>Method</td>
<td>1.41</td>
<td>0.234</td>
</tr>
<tr>
<td>ArtifactType*Method</td>
<td>1.27</td>
<td>0.285</td>
</tr>
<tr>
<td>ArtifactType<em>Method</em>OracleHasMLEs</td>
<td>1.20</td>
<td>0.313</td>
</tr>
<tr>
<td>Method*OracleHasMLEs</td>
<td>0.57</td>
<td>0.568</td>
</tr>
<tr>
<td>Method</td>
<td>0.44</td>
<td>0.646</td>
</tr>
<tr>
<td>Readability<em>Method</em>OracleHasMLEs</td>
<td>0.34</td>
<td>0.712</td>
</tr>
<tr>
<td>Task<em>ArtifactType</em>Method</td>
<td>0.37</td>
<td>0.833</td>
</tr>
<tr>
<td>Task<em>Method</em>OracleHasMLEs</td>
<td>0.09</td>
<td>0.914</td>
</tr>
</tbody>
</table>

Both the main effect tests for task period and OracleHasMLEs were highly significant and explained most of the variation in the response times. The three factor interaction between task period, software artifact type, and readability was also significant. We present the average response times, as well as their differences, in Table 27. All differences between the pre- and post-period task for all artifactType*readability groups were significant at the 0.05 level.

Table 27: Response Time Averages (in seconds) for Task*ArtifactType*Readability

<table>
<thead>
<tr>
<th>ArtifactType</th>
<th>Readability</th>
<th>Task Period</th>
<th>Diff Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Pre</td>
<td>Post</td>
</tr>
<tr>
<td>TraditionalRequirements</td>
<td>Simple</td>
<td>59.61</td>
<td>29.50</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>84.69</td>
<td>27.70</td>
</tr>
<tr>
<td>UseCaseBasedRequirements</td>
<td>Simple</td>
<td>73.58</td>
<td>23.29</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>64.62</td>
<td>30.74</td>
</tr>
<tr>
<td>UserManual</td>
<td>Simple</td>
<td>73.96</td>
<td>29.24</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>73.02</td>
<td>38.84</td>
</tr>
</tbody>
</table>

** adjusted p-value < 0.01; * adjusted p-value < 0.05
**H₀₃**: The response time of subjects on identifying MLEs for statements is the same for all methods used to identify MLEs.

*We fail to reject* $H₀₃$. *We observe no meaningful significant difference in response time between subjects who used the standards-driven, resource-driven, and heuristics-driven methods for identifying MLEs.*

### 6.6 Discussion

In this section, we answer our research questions.

#### 6.6.1 Statement Classification Correctness

**RQ6.1**: *How does the correctness of identifying whether a statement contains a mandatory log event differ among subjects using the standards-driven, resource-driven, and heuristics-driven methods?*

Even though no statistically significant difference exists among the three different methods, statement classification correctness improved from the pre- to post-period tasks for statements that did not contain MLEs, for simple readability statements, and for the traditional requirements and use-case based requirements artifact types. The driving effect that makes the Task*OracleHasMLEs interaction so large is the increase in correct identification of statements that do not contain MLEs, with an increase of nearly 0.31, going from 0.55 in the pre-period to 0.86 in the post-period across every artifactType $\times$ readability $\times$ method combination. Proportions of correct identification of statements that contain MLEs dropped from 0.62 to 0.55, which was significant with an adjusted P-value of 0.0026. The
data suggest that the increase in correct identifications of statements that do not contain MLEs may be due to inflated guessing of many statements as not describing any MLEs.

In Table 23 and Table 24, we observe an increase in identification of whether statements do or do not contain MLEs, but the increase depends on both the artifact type and the readability of the statement. For example, the increase for simple statements was more dramatic than for complex statements (0.20 vs 0.10). The increases for TraditionalRequirements and UseCaseBasedRequirements and are similar (0.17 and 0.20, respectively) but the difference for UserManual is somewhat muted (0.07). However, we also observed that the UserManual group performed better in the pre-period than the TraditionalRequirements and UseCaseBasedRequirements groups, but the post-period task values for subjects assigned to UserManual statements were slightly worse.

No differences were found between the three methods for identifying MLEs. The correctness of identifying whether a statement contains a mandatory log event does not differ among subjects using the standards-driven, resource-driven, and heuristics-driven methods. Instead, subject performance as classifying statements that do not contain mandatory log events improved during the post-period task.

6.6.2 Statement MLE Identification Correctness

RQ6.2: How does the correctness of the identified mandatory log events differ among subjects using the standards-driven, resource-driven, and heuristics-driven methods?

Even though Type III Tests of MLE identification correctness indicated a possible significant interaction between task, artifactType, readability, and method, further analysis
did not reveal any significant differences between the three methods. The lack of significant differences suggests that the readability effect was strong enough to overcome any differences between the methods.

Lower values of statement MLE identification correctness could be due to a lack of identifying the action/resource of the MLE, over-specification of MLEs, or listing the same MLE multiple times for the same statement. Given that subjects were required to read and understand descriptions of the methods for identifying MLES on their own and in a relatively short period of time, subjects may have misunderstood or applied the methods incorrectly.

Statement MLE identification correctness did not significantly differ among subjects using the standards-, resource-, and heuristics-driven methods. Subjects may not have understood how to use the methods correctly within the time constraints.

6.6.3 Statement Response time

RQ6.3: How does the response time of subjects identifying mandatory log events differ among subjects using the standards-driven, resource-driven, and heuristics-driven methods?

From Table 26, the Task*ArtifactType*Readability interaction is significant (p=0.0009). However, the interaction may only be significant because the pre-period response times for the subjects in the TraditionalRequirements-Complex group were higher than both the UseCaseBasedRequirements- and UserManual-Complex response times, yet the post-period response times for subjects in the TraditionalRequirements-Complex group were less than the UseCaseBasedRequirements- and UserManual-Complex post-period response times. For example, we found the response times for subjects in the TraditionalRequirements-Complex
group dropped dramatically, from 84.69 seconds during the pre-period task, to 27.7 seconds during the post-period task.

We also observed that the statements that do not contain MLEs had an average response time of 36.1 seconds compared to 65.4 seconds for those that do contain MLEs. The lower response time for statements that do not contain MLEs makes sense, since a statement with no MLEs did not require the subject to spend time typing descriptions of the MLEs into the form. The overall decrease in response times in all treatment groups from the pre- to post-period task may be due to increased familiarity with the task of identifying MLEs or disinterest/pressure to finish the task before the end of the class period.

The three methods for identifying MLEs did not have significantly different response times. The lack of a significant difference in response times may be evidence that subjects assigned to the heuristics-driven method did not truly follow the instructions, since the heuristics-driven method included more reference materials for subjects to consider and should have taken longer than the standards- and resource-driven methods.

We did not observe any significant differences in response times between subjects who used the standards-driven, resource-driven, and heuristics-driven methods for identifying MLEs. This may, in part, be due to the time limit for responding to each statement.

6.7 Lessons Learned

When designing the experiment, we considered a number of practices to minimize biases and subjectivity. For example, we accounted for each subject’s prior experience so that we can reliably compare the effects of the factors in the experiment. Providing a pre-period task
helped in establishing a baseline of subject performance, while the pre-survey gathered self-reported data on subject experience. The post-survey solicited feedback from subjects related to the method for identifying MLEs, as well as any observations related to the overall experiment design and execution. In this section, we provide an overview of the lessons learned as part of the current experiment.

6.7.1 Provide Both Verbal & Written Instructions

We noticed that a number of subjects seemingly misunderstood the instructions. Instead of identifying the MLEs in terms of actions performed by users (such as “view prescription”), some subjects focused on identifying the contents of the log entry (such as “timestamp” and “username”). While identifying log entry contents is an important aspect of accountability, identifying the actions that should trigger a log entry in the first place is necessary for adequately capturing user activity to support meaningful forensics. To improve the subject’s understanding of the task, explicit verbal instructions may be required to mitigate the risk of subjects ignoring or quickly scanning written instructions.

6.7.2 Remove Time Constraints

According to post-survey responses, several subjects indicated that, in some cases, they required more than the maximum of 3 minutes allocated per statement to identify the MLEs as part of the pre- and post-period tasks. We observed that, for all statements in the study, subjects spent the full 180 seconds on 45% of the statements presented. However, other subjects noted that working on one statement after the other can be tiring, and the ability to pause the task or to revisit responses to previous statements might be helpful. In future
experiments, one should consider allowing subjects to control the pace of the task while assigning an overall time limit to encourage task completeness.

### 6.7.3 Provide More Training on Methods for Identifying MLEs

For the post-period task, after subjects were randomly assigned a treatment group, subjects were presented with an instruction page describing the method they were assigned for identifying MLEs. Even though we allowed subjects to remain on the instruction page as long as necessary to read and understand how to use the method, the value of our response time metric did not significantly differ for subjects using different methods for identifying MLEs, indicating that subjects may not have actually used the methods. For example, the heuristics-driven method provided a description of how to identify all verb-object pairs (see Chapter 5) contained in a statement, as well as all 12 heuristics. Neither the standards-driven nor resource-driven methods included instructions or reference material as lengthy as the heuristics-driven method. Logically, subjects using the heuristics-driven method should have spent more time than subjects in the standards- and resource-driven methods when identifying MLEs for a given statement. The subjects using the heuristics-driven should have referred to the 12 heuristics for each action they identified, leading to higher response times.

### 6.7.4 Block by Subject Pre-period Performance

We underestimated the variability of subject performance during the pre-period task. By random chance, we observed that subjects had high baseline performance in certain groups. For example, response times for TraditionalRequirements-Complex subjects were higher than for other artifactType*readability groups. In addition, UserManual-Simple subjects had
a lower MLE identification correctness during the pre-period than other groups, so the subject performance had more room for improvement during the post-period task than for other treatment groups. By blocking by subject baseline performance, we could have mitigated the random chance of observing significant interactions due to unusually high or low pre-period task performance.

6.8 Threats to Validity

We have considered the following threats to validity.

6.8.1 Internal Validity

Selection: Subjects were randomly assigned to the 18 different artifact type × readability × method groups. Unbalanced groups in terms of subject expertise could result. However, based on the mean, median, and standard deviations of all metrics for academic and professional experience, and English comprehension of subjects for all combinations of the three factors, groups were evenly balanced in terms of expertise.

Instrumentation: When measuring time spent on the task, we automatically recorded time spent in identifying MLEs for each input statement to have a more accurate measure, compared to self-reporting by participants.

6.8.2 External Validity

Representativeness of sample population: Subjects in the study were enrolled in a graduate course on software security, and the study is conducted at the end of the course. In this sense, the subjects can be considered representative of entry-level, non-expert security practitioners.
Experimental constraints that limit realism: Subjects could use a maximum of 3 minutes per input statement to identify MLEs, which may affect the quantity and completeness of the identified MLEs. Similarly, subjects had to first read and understand how to use a given method for identifying MLEs before they could begin the post-period task. To help mitigate this threat, we did not impose a time limit for reading and understanding how to use a given method. Subjects only began the post-period task when they felt that they had learned the method and chose to continue to the post-period task.

6.8.3 Construct Validity

Mono-operation bias: We selected the tasks from three different software artifact types and multiple levels of readability within each artifact type. Thus, we mitigate the threat of introducing biases due to familiarity of subjects with a particular type of artifact. However, even though most subjects were unfamiliar with the iTrust use-case based requirements specification, all subjects were familiar with the actual iTrust software system web-based interface. The iTrust web-based interface was used throughout several exercises and activities involved in the software security course in which the subjects were enrolled.

6.8.4 Conclusion Validity

Reliability of measures: We minimized biases during the evaluation of the responses by employing quantitative measures, having multiple independent evaluators, and using an oracle of MLEs created beforehand.
6.9 Summary

In this study, we conducted a controlled experiment to evaluate the effectiveness of three different methods for identifying MLEs, including a standards-driven, resource-driven, and our previously-proposed heuristics-driven method based on empirically-derived heuristics (King et al. 2015a). We evaluate the three methods using statements from three different types of software artifacts and two levels of readability scores. We compare the methods in terms of statement classification correctness, statement MLE identification correctness, and statement response time in identifying MLEs for a given statement. Based on the results of our statistical analysis, we found that no significant difference exists among the three methods in terms of statement classification correctness, statement MLE identification correctness, or statement response time. For future replications of the experiment, we suggest providing both verbal and written instructions, removing all time constraints, and providing more training on how to use each of the methods for identifying MLEs.
7. Measuring Forensic-ability

We define *forensic-ability* as the ability to gather and examine evidence to determine who, what, when, where, why, and how a security or privacy breach occurs. To understand the degree to which a user activity log enables forensic analysis, we propose that software developers should measure the percentage of mandatory log events (MLEs) (see Chapter 5) that they have logged. Many software development teams already use automated testing tools, such as xUnit or Behavior-Driven Development (BDD) frameworks for black-box verification of software requirements. We utilize a means to leverage existing black-box test suites to compute the coverage of MLEs to determine the forensic-ability of log files. In this chapter, we describe our methodology for measuring forensic-ability through black-box verification of mandatory log events identified for a given software system (see Chapter 5).

*The objective of this research is to help software developers improve user activity logs by proposing and validating a forensic-ability metric that uses automated black-box testing to systematically measure the coverage of mandatory log events captured in user activity logs.*

In this chapter, we present a systematic process for developing black-box test cases to verify that each of the identified MLEs for a given software system is correctly captured in the user activity log. For our process, we leverage existing black-box test specifications and add expected log output to each test case’s expected results. If no black-box test exists for an MLE, we create a new test case to ensure each of the MLEs has an associated test case for verification. We automate the test suite to calculate the forensic-ability of the software system in terms of the percentage of MLEs correctly logged in the system. To further
validate the meaningfulness of our work, an external panel of experts devises sets of malicious user stories for each of the studied software systems. We then execute the malicious user stories to compare how the amount of logged malicious activities compares to the calculated forensic-ability of each software system.

This study is guided by the following research questions:

**RQ7.1**: How valid is our forensic-ability metric?

**RQ7.2**: How does the calculated forensic-ability metric value compare to the amount of malicious activities logged by the software system?

To evaluate our process for measuring forensic-ability, we perform case studies using the Cucumber BDD framework on three software systems:

- **iTrust**\(^{25}\): Open Source Electronic Health Record System v18
- **OCS**\(^{26}\): Open Source Scholarly Conference Management System v2.3.6
- **ProprietaryEHR**\(^{27}\): a commercial Electronic Health Record and Healthcare Practice Management software system

Section 7.1 provides background on behavior-driven development practices for verifying system functionality. Section 7.2 presents our methodology for measuring forensic-ability. Section 7.3 describes our methodology for generating malicious user scenarios as a means of helping validate our forensic-ability metric. Section 7.4 describes subjects used for our case study on three software systems. Section 7.5 presents the results of our case study. Section

\(^{25}\) http://agile.csc.ncsu.edu/iTrust
\(^{26}\) https://pkp.sfu.ca/ocs/
\(^{27}\) The software vendor requested to remain anonymous
7.6 discusses our results and answers our research questions. In Section 7.7, we describe principles for helping software developers measure forensic-ability. Section 7.8 describes threats to validity of our process for measuring forensic-ability. Finally, Section 7.9 summarizes our work.

7.1 Background

In this section, we provide back-ground information on Behavior-Driven Development, which we used when verifying MLEs are actually logged. In addition, we provide background information about the Cucumber BDD framework.

7.1.1 Behavior-Driven Development

Since MLEs describe user actions that must be logged, and since user actions depend on features of a software system, we leverage Behavior-Driven Development (BDD) techniques (North) for black-box verification of logging mechanisms in this study. Introduced by Dan North in 2004, BDD practice requires that user stories with specific user scenarios be explicitly documented as an alternative to test-driven development (Beck 2002).

A user story describes the scope of a software feature along with acceptance criteria for the feature (North). BDD user stories are intended to be documented in a way that the scope of the software feature is easily understandable by business professionals, analysts, software developers, and software testers. A sample template for a BDD user story as described by Dan North appears in Figure 8. BDD user scenarios help serve as acceptance criteria by providing a specific description of how a user interacts with the software system to accomplish the given user story. BDD user scenarios are written in structured natural
language following the Given-When-Then vocabulary. “Given” statements describe preconditions or the state of the system before a user performs an action. “When” statements describe the action performed by the user. “Then” statements describe acceptance criteria for determining if the feature has been implemented correctly or not.

Title (one line describing the user story)
Narrative:
As a [role]
I want [feature]
So that [benefit]

Acceptance Criteria: (presented as scenarios)

Scenario 1: Title
Given [context]
  And [some more context]...
When [event]
Then [outcome]
  And [another outcome]...

Figure 8: Sample BDD user story template, as presented by Dan North (North)

7.1.2 Cucumber BDD Framework

BDD is often facilitated through the use of a tool such as JBehave, Concordian, or Easyb. Our methodology should be compatible with most major BDD tools. However, for this study, we select the Cucumber BDD framework to automate our black-box testing effort because we have previous experience using the framework. The Cucumber Framework involves two primary concepts: feature files and step definitions.
7.1.2.1 Feature Files

In Cucumber, a feature file contains a structured natural language description of a single BDD user story. Feature files typically contain a title for the feature, narrative to help understand the purpose of a feature, acceptance criteria for verification, and descriptions of one or more user scenarios for the user story. Only minor differences exist between syntax for a Cucumber feature file and the syntax of BDD user stories as presented in Figure 8. For example, feature files require a user story title to be pre-ceded by the keyword “Feature”.

7.1.2.2 Step Definitions

In Cucumber, individual steps of a scenario are implemented in step definitions. Step definitions are source code that maps the natural language Given-When-Then steps of a user scenario in a feature file into concrete method or function calls in the software system. Each step definition is annotated with a regular expression that matches with one or more Given-When-Then steps in the set of feature files. “Then” step definitions typically include verification checks using assert statements provided by testing frameworks.

7.2 Methodology: Measuring Forensic-ability

In this section, we describe our methodology for repeatedly and consistently measuring the coverage of MLEs captured in a user activity log. We first require a set of MLEs to be identified for a software system before proceeding with the methodology presented in this section. We propose using our heuristics-driven process to determine the set of MLEs for a given software system (described in Chapter 5).
In this section, we first describe our methodology for augmenting existing black-box test specifications with expected log output. As an alternative, we then present our methodology for generating a new automated black-box test suite by reverse-engineering BDD feature files from our set of MLEs. Next, we describe how we run test cases and record the results. Finally, we present the formula for calculating our forensic-ability metric.

7.2.1 Method A: Software with Existing Automated Black-box Test Cases

For software systems that contain existing and up-to-date black-box test plans that are automated, we leverage the existing test cases to facilitate the calculation of our forensic-ability metric. We ensure that one test case exists for each of the MLEs for the software system. For each MLE, we either:

- find an existing test case that involves performing the described action, then add an expected log outcome for verifying that the MLE is logged; or
- if no test case already exists that performs the described action, we create a new test case in the test suite for verifying that the MLE is logged.

We continue until all MLEs have been tested in either an existing test case or a new test case. In a spreadsheet, we document four pieces of information for each MLE:

- **MLE**: the textual description of the MLE
- **Source**: the source statement from the natural language software artifact that describes the MLE (traceability)
- **Test Type**: “E” if an existing test case was amended to verify the MLE; or “N” if a new test case was created to verify the MLE (traceability)
- **Test ID:** the test case ID of the existing or new test case where the MLE is verified

  In some cases, multiple MLEs may describe the same testable event (for example, “authenticate user” versus “login user” versus “sign in user”). If so, we amend or create a test case for only one of the MLEs and mark the duplicate or redundant MLEs as “RED with X”, where X is the ID of the MLE for which we amended or created a test case.

  We are now ready to run the automated test suite.

### 7.2.2 Method B: Software without Existing Black-box Test Cases

If a given software system does not maintain an existing or up-to-date automated black-box test suite, we reverse-engineer a test suite for calculation of our forensic-ability metric. Since MLEs describe user activities that must be logged, and since user activities depend on features of a software system, we choose to automate our black-box verification of MLEs using BDD practices. In this section, we discuss generation of user stories and features files for the Cucumber BDD framework.

In previous work (King et al. 2015b), we obtain a set of MLEs described as verb-object pairs. For example, <view, prescription report>, <authenticate, administrator>, and <update, contact information> are three verb-object pairs representing events that must be logged.

We begin generating user stories by using the template BDD user story in Figure 8. We discuss each step of filling-in the template in the following sections.

#### 7.2.2.1 Step 1: Specifying a User Story Title

Since BDD user stories must describe an activity to be performed in the software, we consider each of our MLEs as a separate BDD user story. We set the title of each user story
to be the combined verb-object text. For example, BDD user story titles for the previous examples would be “view prescription report”, “authenticate administrator”, and “update contact information”.

7.2.2.2 Step 2: Filling-out Narrative Details

Since we are reverse-engineering BDD user stories for software that has already been implemented, we do not know the intended role who desired the feature nor the intended benefit of the feature. Therefore, we do not fill-out any of the narrative information regarding [role] or [benefit]. We do, however, know the description of the intended feature. We replace the [feature] placeholder with the combined verb-object text, such as “I want to view (a) prescription report”, “I want to authenticate (as) administrator”, and “I want to update contact information”. We manually edit the text as needed to maintain correct grammar and semantics.

7.2.2.3 Step 3: Creating a Scenario

Each user story must contain scenarios as part of the acceptance criteria to verify the feature described. Since our goal involves determining if an MLE is logged, we only create a single scenario for each user story. We first read the activity described in the user story title, then manually document a specific scenario for the activity using the standard Given-When-Then vocabulary. Writing a scenario may require certain domain knowledge or knowledge of the software system, itself, so we reference documentation for the given software as needed. For example, for a health record system, we may have to consult documentation to determine
which type of user (a doctor versus a nurse) has the ability to create a new allergy indicator for a patient.

Finally, we define an explicit “Then” outcome to verify the activity was logged. We provide the following “Then” outcome to each scenario:

*Then the log should contain the following: [user] [verb] [object]*

where [user] is the user who performed the action, [verb] is the action described in the user story title, and [object] is the resource being acted upon. We do not include a field for specifying a timestamp (see section 7.2.3.1) in the feature file because doing so would make each verification step time-dependent and unrepeatable. Instead, we implement a timestamp check in our step definition to ensure the timestamp is recent.

### 7.2.2.4 Step 4: Implementing User Stories using Cucumber

In this section, we describe how we generate Cucumber feature files and Cucumber step definitions in Java.

**Cucumber Feature Files**

In Cucumber, BDD user stories are documented in feature files. The Cucumber feature files follow similar syntax as the BDD user story template in Figure 8. Therefore, we are able to take our generated user story from Section 7.2.2 and paste it into a feature file with only three minor edits:

1. the text of the source statement from which the MLE was described (see Chapter 5) must be preceded by a “#” to indicate the text is a comment,
2. the title of the user story (see section 7.2.2.1) must be preceded by the keyword “Feature:”, and

3. the [user], [verb], and [object] are each surrounded by quotation marks.

In Cucumber, quotation marks allow us to reuse code by passing the text as a parameter to the step definition. We can then use the same step definition to check log contents for all of our scenarios. Since logging mechanisms are implemented differently, some logging mechanisms may have a dedicated field for a description of the action (create, read, update, delete), and a separate field for describing what data was accessed. If we searched the log file for the combined verb-object pair, we would never find the exact text since the phrases appear in separate fields in the log file.

An example feature file appears in Figure 9.

```
# [text of source statement]
Feature: [title]

Narrative:
   As a [role]
   I want to [feature]
   So that [benefit]

Acceptance Criteria:
* [event] must generate a log entry

Scenario: [scenario title]
   Given [context]
   When [event]
   Then the log should contain the following: “[user]” “[verb]” “[object]”
```

Figure 9: Example Cucumber feature file template. The “[user]”, “[verb]”, and “[object]” text are passed to the step definition as parameters, allowing for code reuse in the step definitions. The “Then” step definition also contains code to check for a recent timestamp (see Section 7.2.3.1)
Cucumber Step Definitions

We implement our Cucumber step definitions using the Java\textsuperscript{28} language and Selenium\textsuperscript{29} browser automation. Since the three software systems in our case study are web applications, Selenium allows us to control the web browser from the black-box perspective to perform the activities contained in the BDD user stories. We open the software application in our Firefox web browser and use Selenium to record the steps we take when performing the action described in the given step. We export the recorded Selenium steps as Java code and paste the code into the appropriate step definition.

We continue until all MLEs have been verified in a user scenario in a feature file. In a spreadsheet, we document four pieces of information for each MLE:

- **MLE**: the textual description of the MLE
- **Source**: the source statement from the natural language software artifact that describes the MLE (traceability)
- **Feature File**: the name of the feature file where the MLE is verified (traceability)
- **Scenario**: the title of the scenario in which the MLE is verified (traceability)

In some cases, multiple MLEs may describe the same testable event (for example, “authenticate user” versus “login user” versus “sign in user”). If so, we generate a Cucumber feature file for only one of the MLEs and mark the duplicate or redundant MLEs as “RED with X”, where X is the ID of the MLE for which we generated a Cucumber feature file.

\begin{footnotesize}
\begin{itemize}
\item \textsuperscript{28} http://www.java.com
\item \textsuperscript{29} http://www.seleniumhq.org
\end{itemize}
\end{footnotesize}
We are now ready to run the Cucumber feature files.

7.2.3 Executing the Automated Test Suite

We run the automated test suite so that all of the MLE test cases are executed. For completeness, if any test cases cannot be automated, we manually execute the black-box test case using the software user interface.

7.2.3.1 Mandatory Log Content

In previous work (King and Williams 2013), we identify 22 unique fields that developers could potentially capture for each log entry to help answer who, what, when, where, why, and how each event occurred. However, some of the 22 fields may not be appropriate for all software systems or in certain domains. For the purpose of our current study, and to be consistent with the minimum fields mandated in the healthcare and Payment Card industries (2010c; 2013c), we only require the four most common fields be captured:

- **User identification:** who performed the action
- **Timestamp:** when was the action performed
- **Event Description:** what action was performed
- **Objects Affected:** what data was accessed

If any of the above fields are missing or incorrect, the test case shall fail.

7.2.3.2 Documenting Test Results

In our spreadsheet, we add an additional field to record the results of each test case for each MLE as follows:
• **PASS:** the MLE is correctly logged with an accurate description of the user who performed the action, timestamp the action was performed, description of the action performed, and description of the objects/data affected (see section 7.2.3.1)

• **FAIL:** the MLE is not correctly logged with a description of the user, timestamp, event description, or objects/data affected

• **BROKEN:** the feature in which the user performs the action is broken, so the MLE cannot be verified

• **MISSING:** the feature in which the user performs the action cannot be found in the software, so the MLE cannot be verified

• **UNVERIFIED:** the researchers were not able to verify whether a log entry was recorded. For the ProprietaryEHR system, we did not have access to all log tables. The vendor claims these events are logged, but the researchers were not able to verify the claims.

• **REDUNDANT:** the researchers marked this MLE as a duplicate or redundant testable event with another MLE

• **NA:** when automating the test case for an MLE, the researchers discover the action described in the MLE is not applicable for the given system (for example, the action may be out-of-scope of the software, or the action requires interaction with a third-party software system)

Once we have recorded test results for each of the MLEs for a software system, we can now calculate the forensic-ability metric.
7.2.4 Calculating the Forensic-ability Metric

Once we obtain the test results for each of the MLEs for a software system, we would ideally calculate forensic-ability as follows:

\[
\text{forensic-ability} = \begin{cases} 
\frac{\text{# MLEs correctly logged}}{\text{# total MLEs}}, & \text{# total MLEs} > 0 \\
0, & \text{# total MLEs} = 0
\end{cases}
\]

However, in reality, we must account for broken functionality, missing functionality, and functionality that may be untestable or unverifiable:

\[
\text{forensic-ability'} = \begin{cases} 
\frac{\text{# MLEs correctly logged}}{\text{# total testable MLEs}}, & \text{# total testable MLEs} > 0 \\
0, & \text{# total testable MLEs} = 0
\end{cases}
\]

where

\[
\text{# total testable MLEs} = \\
\text{(# of TOTAL MLEs)} - \text{(# of REDUNDANT MLEs)} - \text{(# of NA MLEs)} - \text{(# of UNVERIFIED MLEs)}
\]
We exclude calculation of MLEs marked as duplicates or redundancies from the calculation of our metric since we did not test the redundant MLEs. We also exclude NA test results in our metric calculation since an “NA” result indicates the action described in the MLE is not applicable for the given system and is untestable. Finally, we exclude the unverified MLEs so that the forensic-ability metric is not negatively skewed. Overall, the forensic-ability metric represents the ratio of MLEs correctly logged out of the total applicable MLEs for the software.

7.3 Methodology: Malicious User Scenarios

To further validate meaningfulness of our proposed forensic-ability metric, we need to demonstrate that a higher calculated forensic-ability metric value suggests a greater amount of malicious user activities being recorded in a given user activity log. To compile a sample set of malicious activity for our case studies, we recruit two external experts. The experts must satisfy the following criteria:

- Each expert must have knowledge and experience with software security and privacy concepts
- Each expert must have no prior knowledge or experience of the user activity logging mechanisms in each of the studied software systems.

The experts first explore the software user interface to identify possible actions that a malicious user could perform, and then they collaboratively document malicious user stories for each of the proposed malicious activities.
7.3.1 Software User Interface Exploration

We first provide the two experts with access to the open-source software so they can freely explore available functionality in the software user interface. For proprietary software, we could not provide access to the software because the software vendor requested that the software remain confidential. Instead, we provide the experts with a written list of available functionality on each page of the interface for proprietary software.

7.3.2 Documenting Malicious User Stories

Next, we ask the experts to formally document user stories with scenarios for each of the malicious activities they devised through exploration of the software user interface. Following the user story template provided in Figure 8, we ask the experts to describe the malicious intent of each activity as part of the user story narrative. The experts document the malicious user stories in a text document.

7.3.3 Executing Malicious User Scenarios

To determine how many of the actions contained in the user scenarios are logged correctly, we manually execute the steps outlined in each user story documented by the experts. We then check the user activity logs to determine if the action was recorded with the following information:

- User identification: who performed the action
- Timestamp: when was the action performed
- Event Description: what action was performed
Objects Affected: what data was accessed

If any of the above fields are missing or incorrect, we mark the user scenario as not logged.

7.4 Case Study Subjects

For our case studies, we select software that meets the following criteria:

- The software must maintain a dedicated log for recording user activities
- The software must be available to install for black-box testing
- The software must already have a set of identified MLEs

For the purpose of this study, we restrict ourselves to only software that maintains dedicated logs for user activities because a software system with no activity logs will likely have a forensic-ability value of 0. We do not restrict ourselves to using only software with existing black-box test specifications. Instead, we want our forensic-ability metric to be applicable for all software development teams who may or may not have existing or up-to-date black-box test specifications.

Since we derived MLEs for three open-source software systems in a previous study (King et al. 2015b), and since the three systems are all available for black-box testing, we select the following three software systems:

- iTrust: Open Source Electronic Health Record System v18. An electronic health record (EHR) system developed and maintained by undergraduate software engineering students at North Carolina State University and used by many researchers and educators as a test-bed (Meneely et al. 2012).
• OCS (Open Conference Systems): Open Source Scholarly Conference Management System v2.3.6. A conference management system developed and maintained by the Public Knowledge Project (PKP), a multi-university initiative developing free open-source software and conducting research to improve quality of scholarly publishing.

• PropEHR (Proprietary Electronic Health Record System). The vendor of this software wishes to remain confidential.

7.5 Results

In this section, we present the results from our three case studies.

7.5.1 Results from Measuring Forensic-ability

Table 28 summarizes the results from executing our Cucumber feature files. We were unable to test 6 MLEs in iTrust and 16 MLEs in PropEHR because of broken functionality. We were unable to locate functionality needed to test 27 MLEs in iTrust, 24 MLEs in OCS, and 8 MLEs in PropEHR.

<table>
<thead>
<tr>
<th></th>
<th>TESTED MLEs</th>
<th>FAIL</th>
<th>BROKEN</th>
<th>MISSING</th>
<th>UNVERIFIED</th>
<th>REDUNDANT</th>
<th>NA</th>
<th>TOTAL MLEs</th>
<th>FORENSIC-ABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>iTrust</td>
<td>138</td>
<td>66</td>
<td>6</td>
<td>27</td>
<td>0</td>
<td>768</td>
<td>212</td>
<td>237</td>
<td>1217</td>
</tr>
<tr>
<td>OCS</td>
<td>18</td>
<td>164</td>
<td>0</td>
<td>24</td>
<td>0</td>
<td>405</td>
<td>135</td>
<td>207</td>
<td>747</td>
</tr>
<tr>
<td>PropEHR</td>
<td>39</td>
<td>45</td>
<td>16</td>
<td>8</td>
<td>74</td>
<td>43</td>
<td>4</td>
<td>108</td>
<td>229</td>
</tr>
<tr>
<td>TOTAL</td>
<td>195</td>
<td>272</td>
<td>22</td>
<td>59</td>
<td>74</td>
<td>1216</td>
<td>351</td>
<td>552</td>
<td>2193</td>
</tr>
</tbody>
</table>
In terms of automating test case code, we had difficulty automating the test cases for 2 MLEs in iTrust and 2 MLEs in OCS. For iTrust, one MLE requires a user session to be terminated after 10 minutes of inactivity, but we had trouble getting Selenium and Cucumber to successfully wait for 10 minutes. Also in iTrust, one MLE requires a user to close the iTrust application, but we had trouble getting Selenium to close the iTrust window without relinquishing the WebDriver used for the next set of test cases. In OCS, an MLE for closing a conference registration type involves setting a date at which registration closes; once the date passes, registration is automatically closed. We had trouble with mocking the system time to verify that registration was successfully closed.

We were unable to verify log entries for a total of 74 MLEs in PropEHR. For 65 of the MLEs, no log entries appeared in the designated logs. For example, after editing family health history for a patient, the “Family Health History” log was blank, containing no entries. For the remaining 8 unverified MLEs, no related logs could be located. For example, after creating a new immunization for a patient, we were unable to locate an “Immunizations” log. The PropEHR vendor indicated log entries do exist in the database-level log tables, but we did not have access to the database log tables to verify. We exclude the unverified MLEs from the metric calculation so that we do not unfairly skew the metric negatively for PropEHR.

We identified 768 redundant MLEs for iTrust. We identified 405 redundant MLEs for OCS. Section 7.6 provides more detail about redundancies in the three software systems.
In iTrust, 212 MLEs were beyond the scope of capabilities of the software interface and
deemed NA, along with 135 MLEs in OCS. For example, iTrust MLEs include “confirm
changes”. In the actual iTrust interface, no inputs or buttons allow a user to confirm changes.
Instead, the act of “confirm changes” is implicit and cannot be explicitly executed in the
interface for testing. Similarly, for OCS, MLEs include “run OCS”, “edit the OCS
configuration file”, and “register for a PayPal account”, each of which are performed outside
of the application interface and cannot be automated with Selenium. For PropEHR, 3 MLEs
were untestable because the activity required connectivity with an e-prescription service
provider. In addition, we could not test 1 MLE for creating users because we did not have the
appropriate permissions in the software provided by the vendor.

In summary, for iTrust, 138 MLE test cases pass out of the 237 unique MLEs tested. For
OC, 18 MLE test cases pass out of the 207 unique MLEs tested. For PropEHR, 39 MLE test
cases pass out of the 108 unique MLEs tested.

Our test results indicate the forensic-ability of iTrust is 0.58, the forensic-ability of OCS
is 0.09, and the forensic-ability of PropEHR is 0.36. To ensure accuracy of our test results,
we manually checked log files to verify that log event descriptions were consistent with our
MLE descriptions or whether log event descriptions used different vocabulary or phrasing
(see section 7.7). For iTrust, we tested against the “Transaction Log” feature provided in the
iTrust database. For OCS, we tested against the conference “Event Log” feature, which
involved a separate log file for each conference. We did not locate any OCS log interface that
captured system-level events such as authentication or creation of new users. For PropEHR,
we tested against audit reports functionality. Finally, we verified our results with development groups for each software system.

### 7.5.2 Results from Malicious User Scenarios

Table 29 summarizes the number of malicious user scenario actions correctly logged in the three software systems. In total, the experts collaborated to develop 41 malicious user scenarios for iTrust, 42 malicious user scenarios for OCS, and 44 malicious user scenarios for PropEHR.

<table>
<thead>
<tr>
<th></th>
<th>Logged</th>
<th>Not Logged</th>
<th>NA</th>
<th>Total Applicable</th>
<th>Percent Logged</th>
<th>Forensic-ability (from Table 28)</th>
</tr>
</thead>
<tbody>
<tr>
<td>iTrust</td>
<td>29</td>
<td>7</td>
<td>5</td>
<td>36</td>
<td>81%</td>
<td>0.58</td>
</tr>
<tr>
<td>OCS</td>
<td>15</td>
<td>15</td>
<td>12</td>
<td>30</td>
<td>50%</td>
<td>0.09</td>
</tr>
<tr>
<td>PropEHR</td>
<td>17</td>
<td>18</td>
<td>9</td>
<td>35</td>
<td>57%</td>
<td>0.36</td>
</tr>
</tbody>
</table>

We were unable to perform some of the malicious user scenarios in the software. For iTrust, 7 of the malicious user scenarios describe functionality that does not exist in the system. For example, iTrust allows users to create and update appointment types, but does not allow a user to delete an appointment type. For OCS, 12 of the malicious user scenarios describe functionality that cannot be performed in the user interface. For example, a conference submission review cannot edit their scores for a paper submission after the reviewer has already submitted his or her scores. Similarly, in PropEHR, the software prevented us from performing 9 user scenarios because we did not have a configured connection to an e-prescription provider.
7.6 Discussion

In this section, we discuss our results and RQ7.1 and RQ7.2.

7.6.1 Metric Validation

*RQ7.1: How valid is our forensic-ability metric?*

We propose that forensic-ability can be repeatedly and consistently calculated by following a systematic process for determining the ratio of correctly logged MLEs to the total number of unique testable MLEs for a software system. However, any software metric must be carefully validated.

When proposing the use of new software metrics, re-searchers must demonstrate that the software metrics are acceptable for the metric’s intended use. The process of demonstrating acceptability is software metrics validation. Meneely et al. (Meneely et al. 2011) catalog a set of 47 metrics validation criteria described in software engineering literature. Meneely et al. suggest researchers should follow a step-by-step process for selecting appropriate validation criteria based on the intended use of the metric, instead of arbitrarily selecting from the 47 validation criteria at random. For this paper, we begin by identifying the intended use of our forensic-ability metric and the advantages we intend to demonstrate: practicality and meaningfulness. Using the step-by-step process provided by Meneely et al. we demonstrate the following metrics validation criteria (Meneely et al. 2011):

7.6.1.1 Actionability

A metric has actionability if it allows a software manager to make an empirically informed decision based on the software product’s status (Meneely et al. 2011). A software
manager could decide to refocus development efforts for security and privacy based on the forensic-ability metric value. With 0.58, 0.09, and 0.36 forensic-ability, respectively, software managers for iTrust, OCS, and PropEHR may realize that their user activity logs are inadequate and could result in malicious user activity being undetectable. A software manager could make an empirically informed decision to focus more development effort on logging additional MLEs, demonstrating the actionability of our metric.

7.6.1.2 Appropriate Continuity

A metric has appropriate continuity if the metric is defined (or undefined) for all values according to the attribute being measured (Meneely et al. 2011). Since our forensic-ability metric is a ratio of correctly logged MLEs to the total number of unique tested MLEs, our metric value will range from [0,1], inclusive. However we have defined the metric value to be 0 in the event the total number of unique tested MLEs is less than or equal to zero. Having 0 total unique tested MLEs would cause the denominator to be 0 and lead to division by zero. Our metric is, therefore, defined for all values between [0,1], demonstrating appropriate continuity.

7.6.1.3 Non-uniformity

A metric has non-uniformity if it can produce different values for at least two different entities (Meneely et al. 2011). We performed three case studies to evaluate our methodology for calculating forensic-ability. For iTrust, OCS, and PropEHR forensic-ability values were calculated as 0.58, 0.09, and 0.36 respectively. Our methodology for calculating
forensic-ability produces distinct metric values for three different entities, demonstrating non-uniformity.

7.6.1.4 Product or Process Relevance

A metric has product or process relevance if it can be “tailored to specific products or processes” (Meneely et al. 2011). To calculate the forensic-ability of a user activity log, one must first identify the MLEs for the given software product. To identify MLEs for the given software product, we extract verb-object pairs from natural language software artifacts for the specific software system. Therefore, our metric is tailored specifically to each software product, as the set of MLEs for two software applications are different and depend on the set of features available to users. Our three case studies in human resources management, healthcare, and conference manage also indicate that the forensic-ability metric is transferrable across domains, demonstrating product or process relevance.

7.6.1.5 Usability

A metric has usability if it can be cost-effectively implemented in a quality assurance program (Meneely et al. 2011). User activity logging must be performed to ensure user accountability, mitigate repudiation threats, and enable meaningful forensic analysis after a security or privacy breach. With automated black-box tests, calculation of the forensic-ability metric is continuous. Software developers should strive to achieve maximum forensic-ability with the least amount of effort, which is why our methodology aims to incorporate measurement of forensic-ability into a development team’s existing black-box testing efforts. Additional effort is required to first identify MLEs from natural language requirements.
specifications for a software system. However, the task of identifying MLEs needs to be performed only once for each set of requirements. In the event requirements are added or changed, only the new or updated requirements would need to be re-processed to identify new or removed MLEs.

For software development teams who do not maintain existing automated black-box test suites, effort is required to generate a black-box test suite from scratch. However, software development teams who incorporate testing of MLEs from the beginning of the software lifecycle distribute the testing effort as the software is developed instead of undertaking MLE testing in one chunk later.

We demonstrate our forensic-ability metric to be valid with respect to actionability, appropriate continuity, non-uniformity, product or process relevance, and usability.

7.6.2 Forensic-ability vs Malicious User Scenarios

**RQ7.2:** How does the calculated forensic-ability metric value compare to the amount of malicious activities logged by the software system?

For this study, two external experts in software security and privacy collaborated to create a set of malicious user scenarios representative of actions a real-world malicious user may try to perform. For example, in iTrust, malicious scenarios included printing lab results to obtain hardcopies of protected health information, increasing dosages for existing prescriptions, and creating fake users in the software. In OCS, malicious scenarios included adding fake reviewers to a conference paper submission, adding fake email templates, and adding authors to previously-submitted conference papers. In PropEHR, malicious scenarios
included adding fake health insurance company information, creating phony mental health evaluations, and changing a user password.

Table 29 summarizes the number of malicious user scenarios that are logged for each software system. For iTrust, 81% of the expert-generated malicious user scenarios involved activities that were correctly logged in the iTrust user activity log. For OCS, only 50% of the expert-generated malicious user scenarios involve actions that are correctly logged. Comparing the percentage of logged malicious user scenarios to the calculated forensic-ability metric values for each software system, we observe that iTrust has both the highest percentage of malicious user activities logged, as well as the highest forensic-ability. Likewise, OCS has the lowest percentage of malicious user activities logged, as well as the lowest forensic-ability. Of the three software systems, PropEHR has the middle value for both percent of malicious user scenarios logged and forensic-ability.

However, the difference between malicious user scenarios logged and the calculated forensic-ability for each system is not equal for each of the three software systems. For example, while iTrust has a forensic-ability of 58% and 81% of malicious user scenarios were logged, OCS has a forensic-ability of 9% with 50% of malicious user scenarios logged. After investigating the disproportionate amount of malicious logged activity in OCS, we observed that the two researchers who generated the malicious user scenarios focused on the core set of user activity actually captured by the OCS user activity log. OCS user activity logs focus on capturing user interactions with submitted conference papers, which are the primary sensitive resources managed by the system. However, we calculated forensic-ability
based on MLEs identified by our heuristics-driven method, which does not consider the level of sensitivity or importance of resource data.

For the three systems studied, we demonstrate that higher forensic-ability indicates a higher number of malicious user scenario activities logged. However, the degree of sensitivity or level of importance of resource data may influence what software developers currently consider loggable, as well as what resources malicious users consider more worthwhile to access.

### 7.6.3 Differences in Forensic-ability

With the highest forensic-ability, iTrust user logs capture many (but not all) activities performed by users. Since both iTrust is an electronic health record system and stores patient health information protected by government regulations (2011c), software developers seem to have motivation for adequately logging user activities. However, the OCS log records only changes to conference timelines (such as the start date, end date, and dates submissions are accepted) and submission-related events (such as assigning reviewers and recommending for/against acceptance). Neither viewing a submission nor creating new users in the system is logged. Conference paper submissions typically contain unpublished intellectual content, and snooping into submissions of colleagues or competitors could go undetected in the current OCS activity logs since viewing of submissions is not logged. Similarly, OCS logs do not seem to address security event logging for capturing user authentication events or the granting /revoking of user privileges.
Overall, while a given software system may not always manage governmentally protected data, logging user activity is still important for software that manages any data that could be considered critical or sensitive. In addition, software developers who consider misuse cases (Alexander 2003) (such as rogue administrators creating fake accounts or manipulating user privileges) may better understand the need for user activity logs to enable user accountability and forensic analysis.

7.7 Principles for Aiding Consistent Measurement of Forensic-ability

While generating black-box test cases for MLEs, we encountered several challenges with identifying redundant MLEs, thus slowing our progress. Table 29 summarizes the most commonly appearing redundant MLEs. While 42 iTrust MLEs and 13 OCS MLEs are redundant descriptions of authentication events, not all redundancies were simple to identify. In this section, we describe the most common challenges faced and provide a set of empirically-derived principles for software developers to consider to facilitate efficient calculation of forensic-ability.

Mandatory log events should be described from the perspective of the user. We identified many redundant user actions in iTrust and OCS. However, identifying redundant actions was challenging. User stories and MLEs both describe activities performed from the user perspective. Based on our methodology, MLE descriptions depend on the perspective of the source statement in the natural language requirement artifact in which the MLE is described. For example, the following statements describe the same core MLE from different perspectives:
System perspective: The system displays a list of prescriptions.

Data perspective: The list of prescriptions is provided for the user.

User perspective: The user views a list of prescriptions.

In our case studies, we observed that some MLEs for the software systems describe the system displays data for users. However, for user stories, these MLEs must be written from the user perspective: the user views data. For iTrust, 9 MLEs described system “display” actions that had to be rewritten as user “view” actions. In addition, 8 iTrust MLEs required rewriting system “present” actions as user “view” actions, 1 MLE required rewriting system “provide” as a user “view” action, and 1 MLE required rewriting an system “obtain” action as an user “enter” action. For OCS, 5 MLEs described “appear” actions that had to be rewritten as user “view” actions, along with 1 MLE describing a “show” action, 1 MLE describing a “present” action, and 1 MLE describing a “provide” action.

Inconsistent perspectives of MLE descriptions slowed our black-box testing effort. By consistently describing MLEs from the perspective of the user, software developers can quickly translate an MLE into a user story. Software developers may also encounter less confusion about whether an MLE applies to a given system or whether the MLE is redundant with another MLE. Less time is spent rewriting MLEs before testing, which improves efficiency when measuring forensic-ability.

Use consistent vocabulary when referring to the same core user activity. The vocabulary used to describe MLEs also made identifying redundant MLEs challenging. When generating test cases for the three systems, we observed that many MLEs described the
same core user activity using different vocabulary. For example, in iTrust, “cancel referral” and “delete referral” both describe the same action. Similarly, “view remote monitoring levels” and “view telemedicine reports” both describe the same user action, since “remote monitoring levels” and “telemedicine data” are synonymous in iTrust. In addition, log entries in iTrust often used different vocabulary to describe the MLE action that was performed. Out of the 138 correctly logged MLEs for iTrust, 90 (65%) of the event descriptions in the log used different vocabulary or phrasing than the identified MLE. For OCS, out of the 18 correctly logged MLEs, 12 (67%) of the event descriptions in the log used different vocabulary or phrasing than the identified MLE. For example, in iTrust, “designate health care provider” is actually logged as “declare health care provider”, “authenticate user” is actually logged as “login succeeded” (with the misspelling of “succeeded”), and “read appointment details” is actually logged as “view scheduled appointment”.

Inconsistent descriptions of the same core user activities slowed our black-box testing effort. By using consistent vocabulary when describing the same core user activity, measuring forensic-ability becomes more efficient: less time is spent investigating whether two different MLEs are actually redundant. Similarly, less time is spent investigating whether an event description in a log entry accurately represents the MLE tested.

Log only the overall activity that was performed, but also indicate which individual data objects were affected. In all three systems, we observed hierarchical relationships between some MLEs. For example, some iTrust MLEs describe “enter demographics”, “enter a first name”, “enter a last name”, “specify a specialty”. However, entering a first name, last
name, and specialty are all subcomponents of entering demographics for a user. Because entering demographics implies that a user is entering a user’s first name, last name, and specialty, recording each of these MLEs are separate events would congest a log file with excessive information (2013a). To simplify logging, we propose that software developers log only the overall activity (such as “enter demographics” or “update demographics”), but record an additional field for the log entry that indicates what data objects were affected (first name, last name, specialty). In our logging catalog (King and Williams 2013), we identified 22 pieces of data fields that developers should consider recording as part of each log entry.

Individual data objects that are subcomponents of an overall activity may create excessive log entries. By logging only the overall activity, we improve efficiency when measuring forensic-ability by reducing the number of overall MLEs that require separate, unique test cases.

Do not implement features for which mandatory log events have not been identified. By implementing “extra” features not described in the software requirements or for which MLEs have not been systematically identified, software developers may introduce user activities that are not traceable for forensic analysis. For example, “extra” functionality that allows a user to modify prescription data may allow hospital employees to change critical medication dosage values without being traceable for user accountability should a medication overdose occur. Implementing “extra” functionality dilutes the meaning of the forensic-ability metric. If a software application has additional functionality for which MLEs have not been identified, a calculated forensic-ability of 0.9 falsely implies 90% of the
activities users may perform are captured in the log. However, in reality, the extra functionality may not log any user activities, and the actual forensic-ability may be much less than 0.9. When automating test cases for OCS, we observed features in the user interface that were not described in the software artifact from which we identified MLEs for OCS. For example, we identified an MLE for “activate event logging”, but we also noticed an option for disabling event logging in the OCS interface. However, the OCS software artifact did not mention the disable event log feature. While this is a limitation of the software artifact studied (an OCS user manual), the mismatch still highlights the need for consistency between the features for which MLEs have been identified and the features that are actually implemented in the software.

Implementing extraneous features without first identifying MLEs may dilute the meaningfulness of our forensic-ability metric. To preserve the meaning of forensic-ability for a given system, software developers should refrain from implementing “extra” functionality without first documenting the features and identifying MLE for the features.

7.8 Threats to Validity

In this section, we discuss threats to internal validity, external validity, and construct validity. We also describe how we address the threats to validity in our current study.

7.8.1 Internal Validity

Threats to internal validity include the correctness of our identification of redundant MLEs and MLEs that were out-of-scope of the software interface. In our case studies, we are outsiders to the development of the three software systems: we were not involved with the
requirements, analysis, or implementation of any of the three systems. Therefore, with our limited knowledge of the features of each system, we may have incorrectly indicated redundant or out-of-scope MLEs. To address this threat, we intend our methodology for measuring forensic-ability to be used by a software system’s developers throughout the software development lifecycle. As a result, the software developers will have more extensive knowledge of the features they implement to more accurately classify MLEs as redundant or out-of-scope. An additional threat to internal validity involves the correctness of our test cases in verifying a given MLE. Because of our lack of involvement with the development of the software systems, we may have tested the incorrect feature when verifying an MLE. To minimize this issue, we intend the primary users of our methodology to be software developers who have extensive internal knowledge of the systems they implement.

7.8.2 External Validity

Threats to external validity include the degree of representativeness of our forensic-ability metric to real-world software systems. To minimize this threat, we study OCS and PropEHR, which are both real-world software systems actually deployed by individuals and organizations internationally. Another threat to external validity includes the possibility of limiting our forensic-ability metric to a specific domain. To address this threat, we consider software from two different domains: healthcare (iTrust and PropEHR), and academic conference management (OCS). Our methodology involves black-box verification of MLEs and does not rely on specific domain knowledge or experience.
Similarly, the set of malicious user scenarios may not be indicative of real-world malicious activities. To minimize this threat, we ask two security and privacy experts to collaborate on a set of malicious user scenarios for each of the software systems.

### 7.8.3 Construct Validity

Threats to construct validity include the possibility that our metric for forensic-ability does not accurately measure the ability to perform forensic analysis. Since an auditor must be able to infer exactly what actions a user performed based on entries in a log file, our metric involves verifying user actions (described in MLEs) are correctly logged. We base the calculation of our metric on the ratio of correctly logged MLEs to the number of applicable unique MLEs for a given software system.

Similarly, we take a first step toward validating our forensic-ability metric by having a pair of security and privacy experts generate sets of malicious user scenarios for each software system studied. By comparing our forensic-ability metric values to the number of expert-generated malicious user scenarios logged, we demonstrate that (for our case studies), higher forensic-ability indicates a larger number of malicious user scenarios that are correctly logged.

### 7.9 Conclusion

While confidentiality, integrity, and availability have traditionally been the most common security objectives in software engineering, user accountability has emerged as an additional important security objective as more software manages sensitive data. User activity logs are an electronic “security camera” to enable auditors to perform forensic
analysis by providing concrete evidence that a user has performed certain actions in a software system. A user activity log should provide evidence about who, what, when, where, how, and why a security or privacy breach occurred. However, software developers often inadequately and inconsistently capture “what” user actions must be logged, and no systematic process exists for black-box verification of a log file’s ability to enable forensic analysis.

We present a repeatable methodology for measuring forensic-ability by performing black-box testing for verification of mandatory log events (MLEs). We describe case studies of our methodology for measuring forensic-ability in two domains: healthcare (iTrust and PropEHR), and academic conference management (OCS). After systematically creating and executing 630 total black-box test cases for MLEs across the three systems, we determine the forensic-ability of iTrust is 0.58, and the forensic-ability of OCS is 0.09, and the forensic-ability of PropEHR is 0.36. The forensic-ability values suggest many malicious user activities may be untraceable in iTrust and PropEHR, while a potentially larger amount of malicious user activities may be untraceable in OCS. When comparing forensic-ability to the number expert-generated of malicious user scenarios logged, we observe that higher forensic-ability tends to indicate a higher number of malicious user activities logged.

By measuring the forensic-ability of user activity logs throughout the software development lifecycle, software engineers can make empirically informed decisions about the degree to which a user activity log enables forensic analysis.
8. Conclusions

To facilitate forensic analysis after a security or privacy breach, we have proposed and evaluated a systematic approach for: (1) identifying mandatory log events (MLEs) described in natural language software artifacts; (2) generating black-box test cases for verifying each MLE is accurately logged; and (3) measuring the forensic-ability of user activity logs. By demonstrating software metrics validation criteria (see Section 7.6.1), we describe the practicality and meaningfulness of calculating forensic-ability throughout the software lifecycle while designing, implementing, and testing user activity logging mechanisms.

In our systematic mapping study, we identified an overall lack of empirical research on logging mechanisms for user accountability. Frameworks exist for cloud-based systems and facilitating data-provenance, but no generalizable or systematic approaches exist for identifying events that should be logged or evaluating logging mechanisms. While the terms “logging” and “auditing” are sometimes used interchangeably, we found at least 53 publications related to the process of analyzing (auditing) log contents, but only 10 empirical research studies related to the act of logging (recording events into log files).

In our second study, we proposed a systematic process for identifying mandatory log events described in natural-language software artifacts. We proposed a set of empirically-derived heuristics to help guide software engineers when deciding if an action should be logged or not. Next, we performed a controlled experiment to compare how well subjects used our heuristics-driven process compared to two alternative processes. While the heuristics-driven process did not reveal any statistically significant performance
improvement, additional replications of the study are needed to mitigate confounding factors that may have influenced the results in our initial experiment execution.

Once we identified a set of MLEs using our heuristics-driven process, our fourth study proposes and evaluates a systematic process for evaluating software logging mechanisms from a black-box perspective. In addition, the fourth study outlines our process for measuring forensic-ability of user activity logs. We also take initial steps to validating the forensic-ability metric by performing case studies on three software systems.

8.1 Contributions

We have contributed the following to the body of knowledge for user activity logging and enabling forensic analysis:

- Empirical confirmation that user activity logs are inadequate and logging user activity continues to be a problem;
- A metric for consistent calculation of the forensic-ability of user activity logs, along with first steps toward validation of the metric;
- A set of empirically-derived heuristics to assist software engineers in determining whether a given user action described in a software artifact must be logged; and
- A systematic methodology for generating black-box test cases to verify mandatory log events are correctly logged.
8.2 Limitations

Based on our current work, we calculate forensic-ability using MLEs identified by our heuristics-driven method. However, our heuristics-driven method does not consider the level of sensitivity or importance of resource data with which users interact. Instead, we currently consider each MLE to have equal importance for the purpose of logging to rebuild traces of user activity. In some domains, such as the healthcare and finance industries, interactions with certain pieces of resource data may be deemed more important to log. For example, logs of user interactions with social security numbers, credit card data, or other personally-identifiable information may be more essential when responding to security or privacy breaches of sensitive data. Since the degree of sensitivity or level of importance of resource data varies by domain and by software system, our approach for measuring forensic-ability currently treats all MLEs as equal.

8.3 Future Work

In Section 7.6.2, we observed differences in the relationship between the amount of malicious user activities logged versus the calculated forensic-ability of software systems. Future work should further investigate the disproportionate difference between malicious activities logged versus forensic-ability to help understand why malicious users choose to interact with certain types of data, as well as why and how software developers chooses to focus on logging interactions with certain types of data.

This dissertation work proposes a process for considering forensic-ability from the beginning of the software development lifecycle by identifying MLEs during the
requirements engineering effort. However, additional research is also needed to help software developers proactively calculate forensic-ability and implement user activity logging mechanisms with high forensic-ability from the beginning, instead of waiting until after a security or privacy breach to reactively patch-together logging mechanisms as an afterthought. Additionally, future work should revisit existing auditing and log analysis research to empirically observe differences in auditing and analysis performance on logs that contain more complete traces of user activity based on our method for identifying MLEs and implementing logs with high forensic-ability.
9. REFERENCES


(1998) IEEE Recommended Practice for Software Requirements Specifications. 830:

