Enabling Forensics by Proposing Heuristics to Identify Mandatory Log Events

Jason King, Rahul Pandita, and Laurie Williams
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
{jtking, rpandit, laurie_williams}@ncsu.edu

ABSTRACT

Software engineers often implement logging mechanisms to debug software and diagnose faults. As modern software manages increasingly sensitive data, logging mechanisms also need to capture detailed traces of user activity to enable forensics and hold users accountable. Existing techniques for identifying what events to log are often subjective and produce inconsistent results. *The objective of this study is to 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 systematically extract each verb and object being acted upon from natural language software artifacts for three open-source software systems. We extract 3,513 verb-object pairs from 2,128 total sentences studied. Two raters classify each verb-object pair as either a mandatory log event or not. Through grounded theory analysis of discussions to resolve disagreements between the two raters, we develop 12 heuristics to help determine whether a verb-object pair describes an action that must be logged. Our heuristics help resolve 882 (96%) of 919 disagreements between the two raters. In addition, our results demonstrate that the proposed heuristics facilitate classification of 3,372 (96%) of 3,513 extracted verb-object pairs as either mandatory log events or not.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification – *reliability, validation.*

General Terms

Measurement, Documentation, Design, Reliability, Security, Standardization, Verification.

Keywords

logging, accountability, security, nonrepudiation, forensics, natural language, software requirements.

1. INTRODUCTION

In the past, software engineers have implemented logging mechanisms to debug software, diagnose faults, and monitor network performance [1]. As modern software manages an increasing amount of sensitive data, software engineers also need to implement logging mechanisms that capture detailed traces of user activity to help provide a means of forensic analysis after a security or privacy 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 [2].

Yuan et al. [1] characterize logging practices for debugging and fault detection in open-source software systems. The researchers suggest that logging is often reactive and performed as an after-thought when an anomalous condition or breach has *already* occurred. The current state of logging mechanisms for nonrepudiation is similar. For example, a 2011 Veriphyr Survey of Patient Privacy Breaches [3] in the healthcare industry claims that 52% of survey participants indicated that their organization did not have adequate tools for monitoring inappropriate access to protected health information.

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 [4] 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 [5].

Although specifications exist for stating how to implement logging mechanisms for user accountability [6] [7] [8], 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?

The objective of this study is to 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. For this work, we define a *mandatory log event* as an action that must be logged to hold the software user accountable for performing the action.

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

- **iHRIS**: Open Source Human Resources Information Solutions[1] v4.2
- **iTrust**: Open Source Electronic Health Record System[2] v18
- **OCS**: Open Source Scholarly Conference Management System[3] v2.3.6

We then manually identify all pairs of verbs and objects acted upon from the software artifacts studied. Next, the first two authors 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 the authors' 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.

For this study, we define the following research questions:

- **RQ1**: How often do descriptions of mandatory log events appear in natural language software artifacts?
- **RQ2**: What similarities and differences exist in the grammar and vocabulary used in different software artifacts?
- **RQ3**: What factors help decide whether an action must be logged?

In addition, this research contributes the following:

- 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 set of considerations for requirements engineers to help clearly and unambiguously document mandatory log events in software artifacts.
- An oracle of mandatory log event classifications for three open-source software systems. The oracle is publically available on the project website[4].

The remainder of this paper is organized as follows: Section 2 discusses related research. Section 3 presents our methodology. Section 4 presents our results. Section 5 provides a discussion of our findings to answer our research questions. Section 6 presents our proposed heuristics. Section 7 discusses proposed considerations for authors of natural-language software artifacts. Section 8 discusses the threats to validity. Section 9 discusses limitations of our work and future work. Finally, Section 10 summarizes and concludes our work.

2. RELATED WORK

Software engineers have historically used software logging mechanisms for many different purposes. Yuan et al. [1] proposed an automated approach to improve fault diagnostic capabilities through logs. In particular, LogEnhancer[9] automatically suggests which variable values need to be recorded in each existing log message. Furthermore, Fu et al. [10] characterize industrial practices surrounding the use of logs and logging. However, all of these proposed approaches target the fault diagnostic capabilities of logging. In contrast, our work seeks to advance logging as a security mechanism to help mitigate repudiation threats and promote user accountability.

In terms of user accountability, Vance et al. [11] discuss the importance of identifiability, the belief that one's actions within a group can 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 factorial survey of 96 information systems students to investigate whether the awareness of logging of user behavior influenced a person's behavior. The researchers suggest auditing increases a user's desire to engage in "approved" actions, thus decreasing intention to violate an access policy. In our work, we identify all mandatory log events described by a set of unconstrained natural-language software artifacts. By identifying the set of mandatory log events, software engineers may better design, implement, and test logging mechanisms to ensure adequate traces of user activity is captured to increase awareness of logging and to mitigate inappropriate user behavior in the software system.

Yskout et al. [12] 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 mechanisms. However, their approach is limited to the modeling of explicit logging requirements, such as "The starting and stopping of the ATM machine needs to be audited." In contrast, our work seeks to identify mandatory log events explicitly-stated or implied by existing functional requirements. For example, the sentence "Doctors may edit previously-created prescriptions" describes two mandatory log events: (1) <edit, prescription>, which involves the explicitly-stated verb "edit"; and (2) <create, prescription>, which involves the implied action "create" since "create" is expressed as an adjective instead of a verb.

In previous work [13], we performed: 1) a general auditable event evaluation; and 2) a specific auditable event evaluation of three open-source electronic health record (EHR) systems. For example, "view" is a general auditable event that does not specify which data resource is being viewed. Specific auditable events detail exactly which data resource is being acted upon, such as "view allergy data", "view medication data", and "view demographics data". The specific auditable event evaluation provided a much finer-grained, more meaningful evaluation of logging mechanisms since "view allergy data", "view medication

[1] http://www.ihris.org/

[2] http://agile.csc.ncsu.edu/iTrust

[3] https://pkp.sfu.ca/ocs/

[4] http://go.ncsu.edu/NLPLogging

data", and "view demographics data" were treated as three separate mandatory log events, compared to just "view data". The three open-source EHR systems studied logged an average of 12.6% of general auditable events in the study, compared to only 7.4% of specific auditable events. Overall, with such a lack of logged events, general auditable events provided by guidelines for logging mechanisms were deemed inadequate to ensure logging mechanisms capture useful traces of user activity. The current work incorporates the specific data resources being acted upon into the set of verb-object pairs.

In 2013, we compiled a catalog of suggested events that should be logged [6]. For the catalog, we collected three types of guidelines/specifications: 1) data transactions that should be logged, 2) security events that should be logged, and 3) data fields that should be captured for each log entry. From the 16 source documents in our catalog, we identified 11 data transactions, 77 security events, and 22 data fields for log entry content[5]. To *discover* 100% of the items in the catalog, a software developer would need to consider 13 out of the 16 source documents. No single source document of guidelines for mandatory log events exists to provide a comprehensive overview of what a software engineer should log. Therefore, for our current work, we present a methodology for identifying mandatory log events based on natural-language artifacts specific to the individual software system.

Another closely related area is the application of natural language processing techniques on software artifacts. Existing approaches [14] [15] [16] [17] [18] leverage natural language text in various software artifacts for software testing and quality assurance related tasks. The most closely related work [15] proposes an automated process to infer access control policies from unconstrained natural language requirements documents. Access control policies center around *who* is allowed or prevented access to perform an action in the system. For our study, we are only concerned with identifying *what* actions may be performed. As a result, we employ basic manual natural language processing and lemmatization when extracting verbs and objects, compared with more advanced natural language processing and machine learning approaches used in related work [15].

Riaz et al. [19] empirically derived templates for documenting explicit security requirements for software systems using existing natural-language software artifacts. For accountability-related security requirements, any natural language sentence that contained a subject acting upon a system resource implied the need for the software to log each time the subject performs the action on the resource. In this work, we further leverage natural language software artifacts, specifically requirements specifications and user guides, to identify user activity that can be performed in a software system.

3. METHODOLOGY

Our methodology 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. In this section, we describe the methodology, and in Section 4 we share the results of applying this methodology.

3.1 Step One: Selecting Software & Software Artifacts

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

a) The software developers must maintain a software requirements specification document.
OR
The software's development community must maintain a user guide.

b) 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. In the future, we plan to study additional types of natural-language software artifacts, as well.

For each candidate software application, we manually browse the software's website to locate applicable natural-language software artifacts. If no natural-language software artifacts are found, we contact 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:

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

b) The artifact must describe a set of actions that users may perform in either: 1) the entire software application; or 2) 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 do not consider software artifacts that are incomplete or describe only a subset of possible user activity for a given module of functionality in the system.

3.2 Step Two: Preprocessing of Natural-Language Software Artifacts

After selecting the natural-language software artifacts for our study, we process the artifacts to make them amenable for use. We first convert 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 facilitates easier processing of the text that appears in tables.

We next separate 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 list the extracted sentences on individual rows in a newly-created spreadsheet. We then proceed with the next activity in our methodology: extracting verb-object pairs.

[5] The full catalog can be found at http://go.ncsu.edu/loggingcatalog

3.3 Step Three: Extracting Verb-Object Pairs

In this study, we consider 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
  - 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
  - Example 2: "Creating a patient…"
    - verb-object pair: <create, patient>
  - 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:
  - Example 4: "Doctors prescribe and update medications"
    - verb-object pair: <prescribe, medication>
    - verb-object pair: <update, medication>

Each software artifact sentence contains zero or more documented verb-object pairs. We document each verb-object pair in the spreadsheet created in Section 3.2 on separate rows beneath the original, unchanged source sentence.

3.4 Step Four: Classifying Verb-Object Pairs

For each software artifact, the first two authors individually classify each verb-object pair as being a mandatory log event or not based on their personal experience and knowledge of logging mechanisms for holding users accountable for their actions in software system. The first author has assisted with teaching of software engineering related courses to undergraduate computer science students since 2009. The second author has over two-and-a-half 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 use only the following general guideline for our classifications:

A mandatory log event is an action that must be logged in order to hold the software user accountable for performing the action.

For this study, we do 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 identifies a full set of user activity performed upon any data in the system. User activities performed on sensitive or protected data (see Section 9 for more details on future work) 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 create two copies of our spreadsheet containing the documented verb-object pairs from Section 3.3. Each of the first two authors receives a copy of the spreadsheet and classifies each verb-object pair by annotating mandatory log event (Y) or not (N) beside each verb-object pair in the spreadsheet.

3.5 Step Five: Comparing and Reconciling Classifications

After performing individual classifications, we compile each spreadsheet with individual classifications into a single spreadsheet for comparison. For each disagreement in our classifications, the first two authors meet to discuss and justify their decisions. We document key points discussed when resolving our discrepancies. When disagreements cannot be resolved between the first two authors, the third author breaks the tie and resolves the disagreement. We document the final classification for each verb-object pair, as well as all disagreements and resolutions.

4. 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.

4.1 Software & Software Artifacts Studied

For our study, we select 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 [20].
- **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 collect 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 [21] for the Page Builder module.
- **iTrust**: Use-case based software requirements specification [22].
- **OCS**: "OCS in an Hour" booklet user guide [23] (We could not locate a requirements specification for this system, so we consider the user guide as a form of requirements specification [24]).

We collect 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.

Table 1: **Summary of top 5 verbs (all verbs vs. mandatory log event verbs)**

Software Artifact	All Verbs		Mandatory Log Event Verbs Only	
	Verb	**Frequency**	**Verb**	**Frequency**
iHRIS traditional requirements	allow	42	allow	42
	edit	16	edit	16
	save	13	save	13
	display	8	display	8
	add	5	add	5
iTrust use-case based requirements	is	217	view	120
	view	120	enter	71
	choose	89	display	52
	select	81	authenticate	48
	enter	71	edit	38
OCS user guide	is	137	add	51
	use	64	create	47
	add	53	allow	45
	select	54	submit	35
	allow	52	log in	28

4.2 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 5 includes a summary of the extracted verb-object pairs from our three software artifacts.

Table 1 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.

In summary, the top five verbs appearing in the iHRIS traditional requirements are also the top five mandatory log event verbs. For the iTrust use-case based requirements specification, both mandatory log event verbs "view" and "enter" appear 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" appear in the set of most commonly used verbs in the entire document. The verb "select" also appears commonly in both the iTrust use-case based requirements and the OCS user guide. Though many verbs frequently appear throughout the natural language text, many of the most commonly used verbs in the use-case based requirements specification and user guide are *not* loggable and may clutter or hinder a software engineer's ability to filter through the text to identify mandatory log events.

4.3 Classification Results

We documented the disagreements between the first two authors 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 [25]. Table 2, Table 3, and Table 4 present the confusion matrices of the initial classifications by the first two authors, before resolving disagreements and conferring with the third author. Section 5.3 provides insight into the differences in inter-rater agreement across the three systems.

Table 5 also summarizes the final results of our classification, after resolving disagreements and conferring with the third author. An average of 1.7 verb-object pairs are extracted per sentence. Of the verb-object pairs in each sentence, an average of 0.9 verb-object pairs are mandatory log events. Overall, 1,263 out of 2,128 (59%) sentences contain at least one verb-object pair that is a mandatory log event. Likewise, 2,060 out of 3,513 (59%) total verb-object pairs describe mandatory log events.

Table 2: Confusion matrix for iTrust classifications

		Author 1		
		Log	**Not Log**	**Total**
Author 2	**Log**	788	148	**936**
	Not Log	615	377	**992**
	Total	**1403**	**525**	**1928**
Cohen's Kappa κ=0.22				

Table 3: Confusion matrix for iHRIS classifications

		Author 1		
		Log	**Not Log**	**Total**
Author 2	**Log**	86	0	**86**
	Not Log	10	10	**20**
	Total	**96**	**10**	**106**
Cohen's Kappa κ=0.62				

Table 4: Confusion matrix for OCS classifications

		Author 1		
		Log	**Not Log**	**Total**
Author 2	**Log**	659	70	**729**
	Not Log	76	674	**750**
	Total	**735**	**744**	**1479**
Cohen's Kappa κ=0.80				

5. DISCUSSION

In this section, we discuss RQ1 and RQ2 and differences in the types of artifacts selected in the study. We also share observations about the top five most common verbs from Table 1, and about differences in inter-rater agreement between the first two authors.

5.1 Frequency of Mandatory Log Event Verb-Object Pairs

RQ1: *How often do descriptions of mandatory log events appear in natural language software artifacts?*

Table 5: Summary of extracted verb-object pairs

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

From Table 5, for iHRIS, each of the 36 total sentences contains on average 2.9 verb-object pairs. For the iHRIS traditional requirements specification, 75% of the total sentences in the requirements specification contain at least one mandatory log event.

From Table 5, for iTrust, each of the 1,301 total sentences contains on average 1.5 verb-object pairs. For the iTrust use-case based requirements specification, 62% of sentences contain at least one mandatory log event.

From Table 5, for OCS, each of the 791 total sentences contains on average 1.9 verb-object pairs. For the OCS user guide, 55% of sentences contain at least one mandatory log event.

Natural language requirements specifications and user guides 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.

5.2 Comparing and Contrasting Software Artifacts

RQ2: *What similarities and differences exist in the grammar and vocabulary used in different software artifacts?*

We observe 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 [26]. 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 [27]. 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 1) 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 suggesteds considering user guides (or user manuals) as requirements specifications [24] 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 Differences in Inter-rater Reliability.

For this study, we compute the Cohen's Kappa metric [25] for inter-rater reliability between the first two authors when classifying verb-object pairs as mandatory log events or not. For iTrust classifications, κ=0.22. For iHRIS classifications, κ=0.62. For OCS classifications, κ=0.80. The iTrust requirements specification was the first artifact examined and discussed, so inter-rater reliability was fairly low (κ=0.22), compared with inter-rater reliability in the iHRIS and OCS artifacts. Once we met to resolve disagreements for iTrust, the justifications discussed between the first two authors 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 κ=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 κ=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 κ for iTrust classifications (κ =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 (Section 6).

6. HEURISTICS FOR DETERMINING MANDATORY LOG EVENTS

RQ3: *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.

6.1 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 [6]. 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.

6.2 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 [6, 13, 28], we discuss the importance of recording whenever a user views data, especially in a software system that manages sensitive data [3]. The majority of classification disagreements between the first two authors involve 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 first two authors 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.

6.3 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 first two authors 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.

6.4 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 [6], 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.

6.5 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.

6.6 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.

6.7 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 I, 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.

6.8 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.

6.9 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.

6.10 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.

6.11 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 1 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

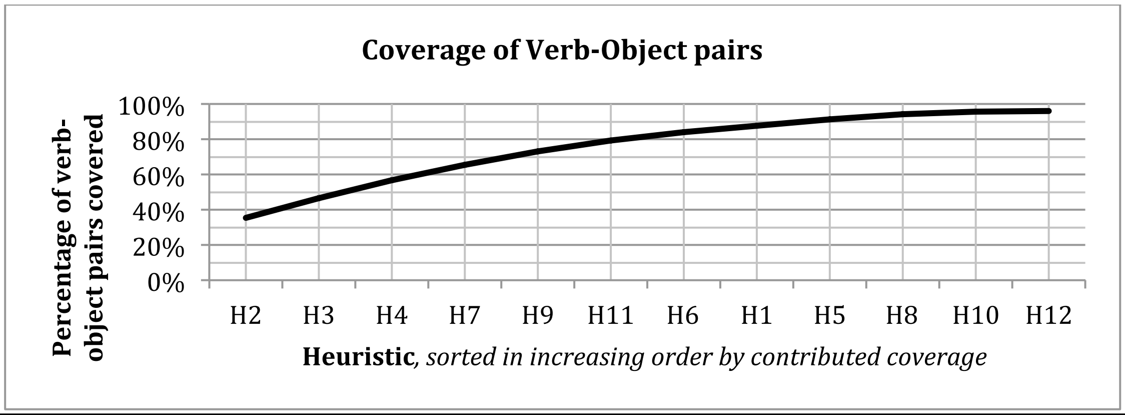

Figure 1. Coverage of verb-object pairs by heuristics, sorted in increasing order by largest number of verb-object pairs covered. Heuristic 2 covers the most verb-object pairs (35%); Heuristic 2 and Heuristic 3, together, cover approximately 47% of verb-object pairs; Heuristics 2-4, together, cover approximately 57% of verb-object pairs, etc. Overall, our 12 heuristics cover about 96% of the verb-object pairs in this study.

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.

7. CONSIDERATIONS FOR AUTHORS OF NATURAL-LANGUAGE SOFTWARE ARTIFACTS

In this study, many classification disagreements between the first two authors 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 considerations 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 first two authors 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.

8. 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.

9. FUTURE WORK

In the future, we plan to conduct additional studies on natural language artifacts from other domains. We also plan to incorporate additional types of natural-language software artifacts that describe user activity, such as feature requests, to further minimize threats to validity. Two additional avenues for future work include prioritization of mandatory log events and automation of our heuristics.

Some software engineers propose that logging mechanisms log "everything" as a naïve solution. However, to comprehensively evaluate a logging mechanism, software engineers must first identify the set of "everything" that should be logged. Based on our methodology, our set of mandatory log events is limited to verb-object pairs explicitly-stated or implied within natural language software artifacts. However, software engineers may wish to further limit or prioritize the set of mandatory log events to only actions that involve user interactions with a predefined set of sensitive or protected data. Since the definition and set of "sensitive" and "protected" data varies among domains, expert knowledge must be incorporated into the process for identifying mandatory log events. Future work on identifying domain-specific sets of sensitive data could help further refine our current methodology to produce a more prioritized and concise set of mandatory log events for which users should be held accountable.

The results of the current study also demonstrate the effectiveness of simple heuristics to determine what user activity should be logged. In the future, we plan to conduct user studies to further evaluate the effectiveness of the proposed heuristics in assisting developers with logging decisions. We also plan to automate the task of inferring mandatory log events from natural language software artifacts and conduct experiments and user studies to evaluate the effectiveness of automating the heuristics to identify mandatory log events.

10. CONCLUSION

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 [1]. Although specifications exist to suggest how to implement logging mechanisms for user accountability [6] [7] [8], 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.

11. ACKNOWLEDGMENT

This work is supported by the USA National Security Agency (NSA) Science of Security Lablet. Any opinions expressed in this report are those of the author(s) and do not necessarily reflect the views of the NSA. We thank Patrick Francis for assisting with identification of verb-object pairs for iTrust. We also thank the Realsearch research group for providing helpful feedback on this work.

12. REFERENCES

[1] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in Proceedings of the 34th International Conference on Software Engineering, Zurich, Switzerland, 2012.

[2] H. Shawn, L. Scott, O. Tomasz, and S. Adam. (2006). Uncover Security Design Flaws Using the STRIDE Approach. Available: http://msdn.microsoft.com/en-us/magazine/cc163519.aspx

[3] (August 2011). 2011 Survey of Patient Privacy Breaches. Available: http://www.veriphyr.com/landing/HIPAA_violation_survey/

[4] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage, "Be conservative: enhancing failure diagnosis with proactive logging," in Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation, Hollywood, CA, USA, 2012.

[5] (2013). CWE-779: Logging of Excessive Data. Available: http://cwe.mitre.org/data/definitions/779.html

[6] J. King and L. Williams, "Cataloging and Comparing Logging Mechanism Specifications for Electronic Health Record Systems," presented at the 2013 USENIX Workshop on Health Information Technologies, Washington, DC, USA, 2013.

[7] (2014). Apache Commons Logging. Available: http://commons.apache.org/proper/commons-logging/

[8] (2013). Common Event Expression: A Unified Event Language for Interoperability. Available: http://cee.mitre.org

[9] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving Software Diagnosability via Log Enhancement," presented at the 16th International Conference on Architecture Support for Programming Language and Operating Systems (ASPLOS'11), Newport Beach, CA, USA, 2011.

[10] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," in Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 2014.

[11] A. Vance, P. B. Lowry, and D. Eggett, "Using accountability to reduce access policy violations in information systems," Journal of Management Information Systems, vol. 29, pp. 263-289, 2013 2013.

[12] Y. Koen, "Transforming Security Requirements into Architecture," in International Conference on Availability, Reliability and Security, 2008, pp. 1421-1428.

[13] J. King, B. Smith, and L. Williams, "Modifying without a trace: General audit guidelines are inadequate for open-source electronic health record audit mechanisms," in Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium, Miami, Florida, USA, 2012, pp. 305-314.

[14] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar, "Inferring method specifications from natural language API descriptions," in Proceedings of the 34th International Conference on Software Engineering, Zurich, Switzerland, 2012.

[15] J. Slankas and L. Williams, "Access Control Policy Extraction from Unconstrained Natural Language Text," in Social Computing (SocialCom), 2013 International Conference on, 2013, pp. 435-440.

[16] L. Tan, D. Yuan, G. Krishna, and Y. Zhou, "/*icomment: bugs or bad comments?*," SIGOPS Oper. Syst. Rev., vol. 41, pp. 145-158, 2007.

[17] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: towards automating risk assessment of mobile applications," in Proceedings of the 22nd USENIX conference on Security, Washington, D.C., 2013.

[18] S. Thummalapenta, S. Sinha, N. Singhania, and S. Chandra, "Automating test automation," in Proceedings of the 34th International Conference on Software Engineering, Zurich, Switzerland, 2012.

[19] M. Riaz, J. King, J. Slankas, and L. Williams, "Hidden in Plain Sight: Automatically Identifying Security Requirements from Natural Language Artifacts," in 22nd IEEE International Requirements Engineering Conference, 2014.

[20] A. Meneely, B. Smith, and L. Williams, "iTrust Electronic Health Care System: A Case Study," in Software and Systems Traceability, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds., ed: Springer, 2012.

[21] (2014). iHRIS: Open Source Human Resources Information Solutions. Available: http://www.ihris.org/

[22] (2014). iTrust: Role-Based Healthcare. Available: http://agile.csc.ncsu.edu/iTrust

[23] (2008). OCS in an Hour: An Introduction to Open Conference Systems Version 2.1. Available: http://pkp.sfu.ca/files/OCSinanHour.pdf

[24] D. Berry, K. Daudjee, J. Dong, I. Fainchtein, M. A. Nelson, T. Nelson, and L. Ou, "User's Manual as a Requirements Specification: Case Studies," Requirements Engineering, vol. 9, pp. 67-82, 2004.

[25] J. Carletta, "Assessing agreement on classification tasks: the kappa statistic," Comput. Linguist., vol. 22, pp. 249-254, 1996.

[26] "IEEE Recommended Practice for Software Requirements Specifications," vol. 830, ed: Institute of Electrical and Electronics Engineers, 1998.

[27] I. Jacobson, I. Spence, and K. Bittner, "Use-Case 2.0: The Guide to Succeeding with Use Cases," ed: Ivar Jacobson International, 2011, pp. 1-55.

[28] J. King and L. Williams, "Log Your CRUD: Design Principles for Software Logging Mechanisms," in Proceedings of the Symposium and Bootcamp on the Science of Security (HotSoS), Raleigh, NC, USA, 2014.