Towards Understanding and Improving Security-Relevant Web Application Logging

Merve Sahin, Noemi Daniele
SAP Security Research
merve.sahin@sap.com,noemi.daniele@sap.com

ABSTRACT

Logging of security-relevant events is crucial in software development to gain visibility into the application's runtime, and to detect suspicious and malicious behavior. Various security guidelines (such as ISO 27002, CCM) mandate the software products to log certain security-relevant events for forensics purposes. In addition, security community (such as the OWASP Foundation) has come up with similar logging recommendations. On the other hand, the lack of sufficient and proper logging practices has been common in the software industry: In fact, "insufficient logging and monitoring" has been part of the OWASP Top 10 web application security risks for many years.

In this paper, we address the issue of insufficient security logging by identifying the security-relevant logging requirements from multiple security guidelines, and by looking at real-world logging practices in a large set of open source Java web applications. We analyze six logging guidelines and identify more than 33K security-relevant logging statements from 472 applications, with respect to different event categories. We present several observations on the log density, positioning, severity levels, use of logging utilities, and the common motivations for security-relevant logs. Our results show that the handling of security logs is not differentiated from the rest of the logging activity, and the current practices are not sufficient to facilitate the detection and investigation of security related issues. Finally, we draw attention to the need for more practical logging guidelines and automated tools to support developers in logging decisions.

ACM Reference Format:
Merve Sahin, Noemi Daniele. 2024. Towards Understanding and Improving Security-Relevant Web Application Logging. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3634737.3637647

1 INTRODUCTION

Airplanes are deployed with flight recorders that record the cockpit audio and flight data from various sensors. These devices provide invaluable information in case of a plane crash to understand what went wrong and how the mistakes can be avoided next time. Application-layer software logging is of similar importance: It provides valuable information about the activities and events that occur during an application's runtime [49]. One of the main motivations for application-layer logging is to collect the security-relevant events. This data enables the detection and investigation of security incidents (e.g., forensic analysis), and can be used to prove compliance with security standards or regulations (e.g., as an audit trail) [21].

In this paper we focus on the security-relevant logging in web applications, motivated by the fact that *Security Logging and Monitoring Failures* is one of the top ten most critical security risks for web applications, as reported by the OWASP Foundation [46]. There might be different reasons for the lack of proper security logging. For instance, previous studies identify the following issues. First, logging constitutes yet another concern for the developers in addition to various responsibilities such as functionality, performance, time-to-market; which often take a higher priority [12]. Second, developers might be responsible for a small part of the software and might lack a holistic view, which makes it difficult to know what is important to log [67]. Similarly, when the logging guidelines are not clear, developers might make personal decisions on how and what to log [27, 66].

In addition, web applications often have the access logs generated by the web server or proxy that record the HTTP requests and responses. Developers might think that such access logs would provide enough information for forensics and auditing purposes, ignoring the need for custom application event logging [49]. While some of the web attacks (such as injection attempts [50]) can be detected via access logs or Web Application Firewalls (WAF) that intercept and analyze the HTTP traffic, other types of attacks (such as the exploitation of business logic flaws [54] and access control vulnerabilities [53]) may not be immediately visible in the HTTP traffic. Moreover, web server logs may exclude the payload of HTTP POST requests due to the data volume and privacy concerns [22]. For such cases, application-layer logging and monitoring remain crucial to gain better visibility into the runtime. Indeed, the application possesses the most detailed information about the context of an event, such as the user details, permissions, actions taken by the user, outcomes, and the reasons for failures – that may not be available in other log sources [23, 49].

Considering all these challenges, our work aims to better understand the web application layer security logging. In particular, we aim to shed light on the following research questions.

RQ1. Which security events need to be logged in an application as a baseline? It is important for development teams to have a logging guideline of what and how to log [21]. While the development team can decide on the application-specific logging points, there are a number of security events common to most applications. Most of the existing security guidelines attempt to list such events. However, these lists of high level descriptions of logging requirements can become difficult to keep track of. It's important to understand the commonalities and the gaps between the existing guidelines, and to come up with a set of security events that can be used as a baseline. In this paper, we make a first analysis of some of the commonly used guidelines to extract their logging requirements.

Although there exists a study on the logging specifications for healthcare applications [37], to the best of our knowledge, this is the first study to analyze the generic security logging guidelines.

RQ2. What are the current security logging practices? It's important to understand the current security logging practices to see how they can be improved. While previous studies analyze logging practices (such as the use of logging utilities, log levels, log locations) in general [19, 25], there are no studies that focus on security-relevant logs.

RQ3. What is the added benefit of application-layer security logging, with respect to the HTTP access logs? As mentioned earlier, HTTP access logs might be considered sufficient for forensics or auditing purposes. However, application-layer logs can provide more detailed information about the context of a security event. We aim to understand which events developers often log at the application layer, and how the application-layer logs complement HTTP access logs in terms of the visibility of the events and application context.

We start our study by analyzing the security events that are required to be logged by six different security guidelines. By analyzing these six sources, we identify a set of events that cover all the listed requirements. We categorize these events under six main topic categories. This allows to see how the guidelines overlap, what they focus on, and what they miss.

We then use this list of events to extract a set of security-relevant keywords, mainly focusing on four of the six topics: authentication, authorization, abnormal behavior, and cryptography. Using these keywords, we look into 5,371 open source Java web applications to identify security-relevant logging statements in their source code. We identify 2,502 repositories with at least one security-relevant logging statement and select 472 repositories for further inspection. Overall, we analyze 33,934 logging statements in various ways.

Our main contributions and findings are as follows.

- **A preliminary analysis of security logging guidelines:** We provide a first analysis of logging requirements in security guidelines, in a comparative way. We find that, despite some overlaps, the guidelines focus on different set of events that are often vaguely described. This might make it very difficult for the developers to comply with multiple guidelines and produce high quality logs.
- **Identifying the common security logging practices:** We propose a method to identify open source web applications and security-relevant logs at a large scale. This allows us to make a first analysis of security-relevant logging in the wild. We observe that security-relevant logging is not differentiated from the rest of the logging activity, and identify various practices that might be problematic. For instance, we find that security-relevant events are often logged as debug/trace logs, which might be disabled in a production environment. Moreover, the use of security specific log levels or logging utilities are very rare, which might complicate the automated processing of security logs.
- **Understanding the need for application-layer security logging:** By making a manual analysis of 400 logging statements, we exemplify events and contextual information that cannot be easily observed in the HTTP traffic, thus requiring application-layer logging. We also draw attention to the lack of forensic-readiness in the current practices.

2 RELATED WORK

There is a large body of literature in the software engineering field that analyze where, what, and how to log, and propose methods to recommend log locations or improve the existing logs [27, 30].

In the security field, most studies focus on creating attack provenance graphs from different log sources, and address the related problems such as dependency explosion and the semantic gap between logs [29, 41, 43, 65]. These studies mainly make use of the system audit logs (i.e., OS level operations) that can be overwhelmingly high-volume and that lack the high-level semantics to understand what actually happened in the application layer [65]. Bates et al. [14] propose to extend the provenance idea for web services, by parsing and interpreting the communication protocols such as SOAP and SQL. Finally, Shen et al. [60] analyzes the access-deny logs in server software, and proposes a static analysis based method to help developers identify the missing logging locations, and the variables that need to be logged to understand the reason of the access deny.

Our study, on the other hand, takes a broader look at the security relevant events that should be logged by the application, e.g., for monitoring and compliance purposes. Application logs can provide further insights into what is happening internally in the application, and can help to obtain more precise and meaningful information for attack detection and analysis.

In this section we make a best effort to summarize the most relevant works to our study:

Security-relevant logging. A line of related work focuses on the event types that need to be logged for forensics purposes. King et al. [37] aim to improve the Electronic Health Record (EHR) systems, by collecting and comparing the existing logging specifications from 16 different sources. Authors find a discrepancy between the healthcare-related and non-healthcare-related specifications, and observe that 13 of the 16 specifications should be considered together, in order to achieve a 100% logging coverage of all security events. In this work we make a similar analysis, but focusing on web application logging. In a related study [38], authors analyze the logs generated by four open source EHR systems via a set of black-box test cases, and identify that certain user interactions are not logged.

Another study [36] proposes to use natural language software artifacts (such as 'software requirements specifications' or 'user guides') to extract verb-object pairs that describe an action that must be logged. Studying three different open source software systems, authors define 12 heuristics (such as CRUD actions, permission and session related actions). Similarly, [57] proposes a machine learning assisted tool to extract security requirements (such as confidentiality, integrity) from software artifacts. Finally, Ortiz et al. [58] aims to identify security-relevant logging locations, with the help of software misuse sequence diagrams created by a security engineer. The diagrams are then annotated with the information related to the software, and used in combination with the control flow graph of the application to identify the appropriate log locations. While this approach considers application specific misuse scenarios, it requires considerable amount of manual work. Moreover, it assumes that security incidents result in access or modification of sensitive data in the database. Such approaches can be helpful to identify application-specific security events, while our work aims to enumerate a set of generic security events common to all applications.

Analyzing logging practices. Another line of related work includes studies that analyze logging practices in open source and/or industry software. Yuan et al. [64] studies four open source projects (incl. source code, commit logs, revision history, bug reports) to see how developers modify the logging related code pieces in time. They find that 36% of the log messages have been modified at least once, and 98% of modifications are to change either the static content, the logged variables, or the log verbosity. The deletion or moving of the logging code is found to be very rare. Another study by Fu et al. [25] analyzes two large industrial software to identify the main logging locations (e.g., exceptions, return value check, assertion check, logic branch). Moreover, they conduct a survey on 54 developers to understand how they determine where to log. This study finds that logging decision is often related to the semantic functionality and context of the code snippet. For instance, certain keywords in source code are found to relate to higher logging ratios. Another interesting finding is that the developers often make logging decisions in the scope of function or block level, while the overall application logic and security related factors are considered less.

Pecchia et al. [52] analyze the source code and log entries in a large-scale critical industry software, to understand how and why developers log. They find a lack of standardized event logging and formatting. The three main purposes of logging are found to be execution tracing, event reporting, and dumping the execution state. Finally, Chen et al. [19] conduct a quantitative study on the degree of adoption of different Logging Utilities (LUs) among the 11K Java projects collected from GitHub. They compare the number of external and internal LUs used across different projects, with respect to the project size. They find that more than 95% of projects use ELUs (External Logging Utility), while ILUs (Internal Logging Utilities) are used in 12% of small projects, growing to 92% of use in very large projects. The study also gives insights about the different motivations behind the use of internal and external LUs.

Our study also makes various analyses of logging practices and related source code tokens. However, different from the previous work, we focus on web application-layer, and security-relevant logging.

3 PRELIMINARY ANALYSIS OF SECURITY-RELEVANT LOGGING REQUIREMENTS

Web applications deal with various types of security-relevant events depending on their functionality and context. While some of these events are application specific, there are certain components (such as user authentication) that are common to many applications. In this section, we aim identify a baseline for the security-relevant events that should be logged, in order to answer our first research question RQ1. For this, we make a qualitative analysis on the commonly accepted industry guidelines and security controls.

Indeed, software companies often need to comply with several standards, guidelines, and controls, to be able to operate in multiple countries and industries[1]. These guidelines recommend certain security-relevant events to be logged for forensics and incident response purposes. An example method that contains such a security-relevant event is given in Figure 1. This method checks the validity of a password reset token and generates a log if the user tries to reset the password with an expired token.

As mentioned earlier, while web applications may involve a diverse set of security-relevant events, the industry guidelines aim to provide a common basis for security logging best practices. This section analyzes five such industry recognized guidelines (ISO/IEC 27002, FedRAMP, NIST, ACSC, CCM) and also the set of security-relevant events listed by OWASP, to come up with a baseline of security events. We explain the nature of these sources below:

ISO/IEC 27002 is a series of information security controls from International Organization for Standardization (ISO), based on commonly accepted best practices [32]. It mainly provides implementation guidelines to be certified with the ISO 27001 information security management standard. Note that, ISO/IEC 27017 [31] guideline that focuses on cloud services also refers to ISO/IEC 27002 for logging requirements.

NIST Special Publication (SP) 800-series guidelines are a series of cybersecurity related guidelines, recommendations, and specifications from National Institute of Standards and Technology (NIST) in the US [44]. While the guidelines consist of a large variety of topics[2], we focus on SP 800-92 [34] that focuses on 'Computer Security Log Management', and SP 800-53 [45] that is about 'Security and Privacy Controls for Information Systems and Organizations'.

FedRAMP (Federal Risk and Authorization Management Program) is a government-wide program in the US that provides a security assessment and continuous monitoring framework for the cloud services and service providers. We refer to the FedRAMP Continuous Monitoring Strategy Guide [11] that lists a set of events. We also include FedRAMP's Q&A for cloud service providers [40] in this analysis.

ACSC (Australian Cyber Security Centre) is a government body that provides information security guidelines for organizations. We refer to the Information Security Manual (ISM) [9] that also outlines an event logging policy.

[1]See for instance: https://aws.amazon.com/compliance/programs/, https://www.atlassian.com/trust/compliance/resources, https://www.sap.com/about/trust-center/certification-compliance.html, https://www.oracle.com/corporate/cloud-compliance/
[2]https://pages.nist.gov/NIST-Tech-Pubs/SP800.html

```java
private boolean isValidResetToken(Sysprop s, String key, String token) {
  if (StringUtils.isBlank(token)) {
    return false;
  }
  if (s != null && s.hasProperty(key)) {
    String storedToken = (String) s.getProperty(key);
    // tokens expire afer a reasonably short period ~ 30 mins
    long timeout = (long) CONF.passwordResetTimeoutSec() * 1000L;
    if (StringUtils.equals(storedToken, token) && (s.getUpdated() + timeout) > Utils.timestamp()) {
      return true;
    } else {
      logger.info("User {} tried to reset password with an expired reset token.", s.getId());
    }
  }
  return false;
}
```

Figure 1: Example method containing a security-relevant event. Application: scoold, Java file: SigninController.java [5]

CLOUD CONTROLS MATRIX (CCM) is a set of security controls and implementation guidelines for cloud services, proposed by the Cloud Security Alliance (CSA) [10].

OWASP (Open Web Application Security Project) is a nonprofit organization that provides open source resources (guidelines, tools, documentations, trainings) to improve security. OWASP is supported by a large online community and several industry members. In this work we refer to the OWASP Logging Cheat Sheet [49].

Remarks:

We first want to emphasize that we do not aim to make an extensive systematization of all logging requirements from all available security guidelines. For instance, we exclude the logging requirements from industry-specific standards (such as PCI-DSS [28] for payment industry, HIPAA [63] for healthcare industry), and privacy related law and regulations (such as GDPR [6]). Our analysis is not exhaustive, however, it covers some of the most common global guidelines (such as ISO 27002, CCM, OWASP), as well as government-based sources from different countries (e.g., ACSC from Australia, NIST and FedRAMP from the US). Our purpose is to make a preliminary analysis that will be helpful in the second part of the study. At the same time, we identify various issues with the guidelines, and draw attention to the need for further studies in this domain.

Second, the guidelines and frameworks we analyze differ in nature: While OWASP focuses on web application security, FedRAMP and CCM focus on cloud service providers and applications. On the other hand, ISO 27002, NIST SP 800, and ACSC aim to provide generic information security guidelines. However, even these generic security guidelines mention events that apply in the context of web applications, such as 'failed authentication'.

Another important point is that, all the guidelines recommend organizations to do their own threat modeling exercise to identify and log the security events specific to their business (on top of the generic recommendations). However, as the ideal case of threat modeling may not always happen, the list of events mentioned in the guidelines forms a minimum baseline and a starting point for the developers. We analyze these events to understand their coverage and the main topics they address.

3.1 Method.

For this qualitative analysis, we first go through all the mentioned resources, to extract the text relevant to the security events that are recommended to be logged. We skim all the chapters of the guidelines, and identify the chapters related to logging and monitoring. We further search for keywords such as 'logging','auditing', 'logged', 'log', 'audit', 'record', 'monitor' to find any other relevant text[3]. We extract these pieces of raw text from the guidelines into a separate document.

The guidelines often mention the events dispersedly in different sections, which makes the identification of the events more difficult. Moreover, the events often do not follow a certain order or categorization, and they are described very broadly (e.g., "any authorized access", "input validation failures"). Although some sources provide examples for the events, most of the time there is no detailed explanation about the scope of the event.

In the next step, we extract the list of events that are required to be logged for each guideline. In this process we keep the event descriptions as is, however, we separate the descriptions into independent units when necessary. For instance, FedRAMP recommends to log 'addition or removal of users'. We divide this event description into two separate events that are 'addition of users' and 'removal of users'.

In the third step, we group together the event descriptions that have the same or similar meaning, from different guidelines. For instance: we could group the following five event descriptions:

- "attempted access that is denied" (ACSC),
- "records of [...] rejected data and other resource access attempts" (ISO 27002),
- "failed accesses related to systems" (NIST),
- "invalid access attempts" (CCM),
- "Authorization (access control) failures" (OWASP)
- " Failed attempts to access data and system resources" (ACSC - Operating system related section)

[3]Note that, we ignore the chapters related to privacy (e.g., handling of PII data), and related to the storage, processing, and security of generated logs, as they are out of scope for our study.

- "Unsuccessful attempted access" (ACSC - Database related section)

into one event, that is, "Failed authorization attempts".

We provide the output of this step in Appendix A: It includes all the event groupings, with reference to the related raw text from each guideline.

After the first grouping of the event descriptions, we further categorize the events according to their context. The categories are as follows:

- Authentication: Contains events related to users' account management, authentication and session management.
- Authorization: Contains events related to access control, management of privileges and permissions.
- Abnormal Behavior: Contains the unexpected events that the developers could anticipate, and embed monitoring points in the code (such as input validations, checks for fraud attempts).
- Cryptography: Relates to management and use of cryptographic keys and modules.
- Data & User Transactions: Relates to the logging of any interaction with the application/business data, such as user input and database transactions.
- Operational Events: Contains events related to the operation of the application, and its interaction with other layers such as the operating system and network connections.

Note that this categorization is not definitive, i.e., other ways of categorization would be possible. However, we think that it summarizes the main topics that the guidelines address.

In the final step, we present all the events we identified from the six sources in a comparative way, in Table 1. For each event, the table shows if a guideline explicitly includes the event (denoted with ●), or if it mentions the event in a limited context (denoted with ◐). An example of limited context is with ACSC, which mentions most of the events to be logged in either the operating system or database layers, rather than the application layer. Another example is CCM, which recommends to log the addition of new accounts only for the accounts with root or administrative privileges.

Caveats. Although we aim to separate the events into independent units, the event descriptions from the guidelines sometimes overlap. For instance, FedRAMP recommends to log "authorization checks", but does not specifically mention successful and failed authorization attempts. Similarly, NIST and CCM recommend to log "key management activities" but they do not specifically mention the "management of key changes". It would be possible to further group such overlapping events, however, that would blur the lines of whether an event is included in a guideline. Thus, for the sake of clarity and avoiding subjective decisions, we preferred to not group the overlapping events.

A note on Events vs. Alerts. The security-relevant events do not have to directly relate to an ongoing attack. Sometimes, a sequence of different events might indicate an attack that needs to be alerted. Such correlations and the writing of attack signatures are often handled by incident response teams, e.g., using Security information and Event Management (SIEM) tools that analyze aggregated log data. As an example, multiple *failed log-on* attempts from different users can be correlated, which might indicate a password spraying or credential stuffing attack. Or, the logging of a *file upload* event can help the incident response team to analyze the exploit of an XML External Entity (XXE) vulnerability [55], which leads to arbitrary file read on the system. Our work aims to identify such events that could be helpful in attack investigation. As mentioned earlier, application specific threat modeling and understanding of the attack landscape can help to better decide which events need to be logged.

A note on Success and Failure events. In terms of the log volume, one might think that the successful events do not need to be logged. However that is often not true: Recording the success events -such as the list of users successfully authenticated to a system- can be necessary for auditing and forensic analysis in the long term [21]. Moreover, success events may directly relate to attacks. For instance, if a regular user successfully executes a function that requires administrative rights, this might be result of a privilege escalation attack. Thus, a security-relevant event can be both a success, or a failure event. In relation to this, Chuvakin et al. [21] distinguishes between 'critical' and 'accounting' logs. They categorize the failures and high severity attack attempts as critical logs that require immediate action. The success events, status messages, and low impact attack probes are categorized as accounting logs that do not require immediate action.

3.2 Observations from the guidelines.

Table 1 presents our results containing 57 types of events in 6 categories. Our first observation from Table 1 is that, although there is some overlap, different guidelines focus on different set of events. Thus, compliance with one guideline may not mean that all the important events are logged. Moreover, it might be difficult for the developers to pinpoint the overlaps, as the guidelines provide descriptions in different formats and terminology, and in a coarse-grained and high-level manner (e.g., 'system events' or 'data changes'). In the future work, we aim to investigate developers' experiences in complying with such guidelines.

While all the guidelines cover authentication and authorization related events to a greater extent, most of them do not mention monitoring of abnormal behavior such as input validation failures, or suspicious activity. We observe that only one event is recommended by all the guidelines, which is, the logging of *successful authentications.* We find that OWASP provides the most comprehensive list of events, possibly because it specifically focuses on web applications, rather than providing generic recommendations.

We can see that even the baseline of requirements in Table 1 involve a large number of events to consider. On top of this, development teams need to think of the application-specific logging needs, and make sure that the security logs contain the right amount information for forensics and auditing purposes, and they use the right verbosity level. Indeed, complying with the guidelines and producing high quality, forensics-ready logs seem to be a very challenging task.

In the next section, we look at real-world source code to understand the current security logging practices in the wild. In particular, we use the list of events in Table 1 to come up with a set of keywords to identify the security-relevant logs.

		ISO 27002	NIST	ACSC	FEDRAMP	CCM	OWASP
	Successful authentications	●	●	●	●	●	●
	Failed authentication attempts		●	●	●	●	●
	Log-off	●		●			
	Assignment of users to tokens				●		●
	Addition of new tokens				●		●
	Removal of tokens				●		●
	Creation of new users / accounts	●	●	◐	●	◐	●
AUTHENTICATION	Deletion of users / accounts	●	●		●	◐	●
	Account modifications	●	●	◐		◐	
	Changes to authentication mechanisms					●	
	Account management events	●		◐	●	●	
	Authentication checks				●	●	
	Credential usage		●				
	Successful authorizations	●		◐			
	Failed authorization attempts	●	●	●		●	●
	Authorization checks				●	●	
	Privilege assignment	●	●				
	Use of privileges and privileged functions	●	●	◐	●	●	
AUTHORIZATION	Administrative activity		●	◐	●	●	●
	Changes to permissions			◐	●	●	
	Changes to privileges	●		◐	●	●	●
	Attempts to elevate privileges / permissions			◐		●	
	Access to (important) data and objects	●		◐	●	●	
	Input validation failures		◐				●
	Output validation failures						●
	Potential integrity violation		●				
ABNORMAL BEHAVIOR	Session management failures						●
	Sequencing failures						●
	Suspicious, unexpected behavior					●	●
	Fraud and other criminal activities						●
	Excessive usage		●				●
	Use of data encrypting keys				●	●	●
CRYPTOGRAPHY	Management of key changes				●		●
	Key management activities		●			●	
	User transactions, requests and responses	●	●	◐			
	Database queries		◐	●			
	Database failures			◐			
	Changes to database structure			◐			
DATA & USER TRANSACTIONS	Modifications to data			◐	●	●	●
	Data deletions				●	●	
	Data access				●	●	
	Import and export of data				●		●
	Submission of user generated content				●		●
	File accesses	●	●				
	Application / system startup		●	◐			●
	Application / system shutdown		●	◐			●
	Configuration changes	●	●	●	●		●
	Application failures and errors		●	●			●
	Application code file / memory changes						●
	File system errors						●
OPERATIONAL EVENTS	System events				●	●	●
	Process tracking				●	●	
	Use and management of network connections				●		●
	Creation and removal of system level objects				●	●	●
	Performance issues						●
	Connectivity problems						●
	Third party service errors						●

Table 1: List of events extracted from the guidelines. ● denotes that event is explicitly included, ◐ denotes that the event is mentioned in a limited context.

4 SECURITY-RELEVANT LOGGING IN THE WILD

In this section we aim to look into the real-world security-relevant logging practices in open source web applications. In particular, we focus on the Java programming language for several reasons: Java is one of the most popular programming languages [61, 62], commonly used for enterprise applications [16]. In addition, Java frameworks like Spring Security provide a mature enterprise application security layer [15] to handle most of the security relevant events. Thus, we believe that Java web applications can be a more reliable source of security-relevant logging compared to other languages. Moreover, previous work that study the logging practices in Java applications [19, 20] provide us a well-grounded baseline for our analyses.

4.1 Data collection

4.1.1 Application selection. We start with a list of public GitHub Java repositories collected from GHTorrent, made available by Chen et al. in [19]-[18]. This list contains 83,082 repositories with more than 5 stars, last updated on 2019-06-01. Among these, we select the repositories with at least 15 stars, corresponding to 43,986 repositories. We then download the source code of the most recent versions (as of 2022-11-24) of these repositories using GitHub APIs.

To be able to focus on the repositories with web components, we check if there is at least one Java file in the repository that imports one of the back-end Java web framework libraries. The libraries we look for are `org.springframework.web`, `javax.servlet`, `org.apache.struts`, `org.apache.wicket`, and `play.mvc`. (Note that we do not include libraries that only provide web clients such as `HttpCLient`, or `HttpUrlConnection`.) Some frameworks use other frameworks or libraries in the background. For instance, Sling [7] uses `javax.servlet` and Vaadin [8] uses Spring to deal with HTTP requests. Thus, our method covers these frameworks as well. This method of eliminating the repositories without web components yields 5,371 repositories that we further analyze for security-relevant logging statements.

4.1.2 Identifying the security-relevant logs. In this section we devise a heuristic-based technique to identify security-relevant logging statements. Note that, we do not claim to identify all of such statements, as this would require manual analysis of the projects. Our purpose is to automatically identify as many logging statements as possible, with high precision (i.e., minimizing the false positives).

We define a security relevant logging statement, as a (i) logging statement that includes a (ii) security-relevant keyword.

(i) Identification of logging statements.

We aim to identify the logging statements in two different formats. First, we look at the logging statements that use a logging utility (LU), which is a library or framework providing various functionality related to application logging (such as formatting, configuration, storage)[4]. The logging statements using such utilities often follow the pattern of:
<logger_object>.<log_level>(<log_description>)[5].

We use regular expressions to match this pattern, and make sure that the <logger_object>.<log_level> part contains the 'log' or 'SLF4J' keywords[6]. After manual inspection, we update our regular expression to eliminate words that lead to false positives in this part, such as 'catalog, blog, logo, analog, dialog, logical'.

The second pattern we look for is the standard output stream functions of Java, such as System.out.println(<log_description>)[7].

Finally, we remove the logging statements from certain Java files (e.g., when the file name contains 'audit, logger, slf4j, log4j, test' words) that are likely to only include logging configuration, templates, or unit tests.

(ii) Identification of security-relevant keywords.

To identify the security-relevant logging statements, we check if the <log_description> part of the statement contains at least one security-relevant keyword. In order to identify the security-relevant keywords, we first look at the logging requirements collected in Section 3. We find that, for Authentication, Authorization, Abnormal Behavior, and Cryptography categories, it is more straightforward to come up with a specific set of keywords that could identify security-relevant logs with high precision. This is more difficult for the other two categories: For instance in the Data & User Transactions category, data fields, file names and parameter names can be arbitrary strings. Moreover, the Operational Events & Failures category contains very generic event descriptions such as 'system events' or 'application failures'. Thus, we focus on the first four categories in the rest of the analysis.

Combining the event descriptions from guidelines and our domain knowledge on the common terminology and attack types, we select the following keywords:

- General security context: security, audit.
- Authentication related keywords: authen, cookie, credential, password, accesstoken, access token, login, logout, logoff, log in, log out, log off.
- Authorization related keywords: authoriz, authoris, role, privilege, permission, accessright, access right, access granted, access denied.
- Abnormal Behavior related keywords: invalid, fraud, suspicio, tamper, excessive, violat, risk, threat, malicious, exploit, attack, vuln, insecur, unsafe, xxs, xxe, csrf, ssrf, denial, brute, abnormal, anomal, protected, expir, injection.
- Cryptography related keywords: crypt, certificate.

A note on the audit keyword. Auditing refers to the process of verifying whether a system follows a set of requirements (which are part of a regulation or a policy) that the system needs to comply with [21]. The logs that are generated for auditing purposes are often called the *audit logs.* For instance, an audit log can aim to hold a user accountable for their actions, by recording each event together with the user's identifier [13]. As the compliance and forensics related logs might be tagged with the *audit* keyword, we include it in our keyword list.

4.1.3 Selecting the final dataset. With the method described above, we find that 2,502 of the 5,371 Java repositories have at least one

[4]Examples of commonly used logging frameworks in Java are log4j, logback, SLF4J. However, the application developers can also create their custom logging utility.
[5]e.g., logger.debug("User registration failed. Password is not strong enough.");
[6]Note that, SLF4J serves as an abstraction layer to several logging frameworks in Java (e.g. java.util.logging, logback, log4j) [56]. As it does not contain the word 'log', we include it separately in the regular expression.
[7]e.g., System.out.println(userName + "'s password has been changed");

security-relevant logging statement. However, the total number of security logs remains low for most of the repositories: More than half of the repositories (1,253) contain at most 5 logging statements. To further increase the quality of the dataset, we eliminate some of the repositories according to three criteria: The total number of security-relevant logs, the security log density (number of security-relevant logs divided by total number of lines of code), and the security log file density (the number of files with a security log divided by the total number of files in the project). We plot the histograms and decide the following threshold values: number of logs > 5, log density > 0.0003, file density > 0.025. This leaves us with 33,934 security-relevant logging statements from 472 repositories.

To verify the accuracy of our log identification method, we randomly sample 500 logging statements and manually check if they indeed log security-relevant events. With 95% confidence, our method yields 98±1.2% precision. Examples of false positives include setter methods on logger objects[8], or methods to store the audit logs in a remote location [9].

As we do not manually analyze all applications to extract all security-relevant logs, we do not know the number of false negatives, and we cannot compute the recall.

In the rest of the study, we focus on these 472 repositories for all further analysis. On average, the selected repositories have 69 (SD=164) security-relevant logging statements from 25 (SD=43) Java files per repository. The maximum number of security logs found in a single repository is 1,511.

4.1.4 Limitations of the approach. Our approach achieves high precision, however it might miss to detect some of the security-relevant logs for the following reasons: First, it is possible that some logging statements are about security-relevant events, but they do not contain the keywords we searched for. Second, it is possible that some projects write logs to specific files, and our current method of matching keywords such as 'log' or 'System.out.print' might be insufficient to catch all the logging statements.

We again emphasize that we do not claim to identify and analyze all the Java web projects on Github, and all the security-relevant logging statements they contain. This would require manually analyzing all the projects. Our purpose is to gather a relatively large dataset (472 projects, 33K+ logging statements), with high precision.

4.2 Analysis of Security Logging Practices

This section aims to answer our second research question (RQ2), by looking at the different aspects of security-relevant logging practices used in the wild.

4.2.1 Logging Utilities. We analyze the use of Logging Utilities (LUs) in the 472 projects, with a focus on the security-relevant logs.

To identify the LUs, we extract the `import` statements in the source code that relate to logging related keywords such as `log, logging, logger, audit, log4j, slf4j`. In addition to collecting all the LUs, we separately extract the LUs involved in the security-relevant logging statements.

[8]e.g., (i) wikiLog.setWe_user_idx(loginUser.getWeUserIdx());
(ii) xTrxLog.setObjectClassType(AppConstants.CLASS_TYPE_PASSWORD_CHANGE);
[9]e.g., applicationAuditLogger.sendMessage(oAuth2AuditLog);

Similar to [19], we merge the logging utilities with the same package name in the top three levels: For instance: *de.metas.logging.LogManager* and *de.metas.logging.TableRecordMDC* are merged into *de.metas.logging* as a single logging utility.

Results. Among the 472 projects, we find that 95% (448) make use of LUs, and 88% (415) make use of LUs for security-relevant logs. In general, a project uses 2.5 (SD=1.3) different LUs on average. For security relevant logs, average number of LUs per project is 1.5 (SD=0.8). Thus, around half of the LUs in a project are used for security logging.

Looking at the number of distinct LUs across all projects, we find that security logs use 116 distinct LUs out of the total 230, again corresponding to a half.

External and internal LUs. Following the previous work by Chen et al. [19], we define a logging utility as internal if it is developed inside the project, and external if it is used as a third party library. In their study, Chen et al. analyzes the motivations behind the use of external and internal LUs in software [19]. They find that external LUs are often used for general purpose logging, and sometimes to interact with LUs from imported packages. They also find that the common reasons for developers to implement their own internal LUs are: defining a custom logging format, ease of configuration, and providing a common interface for different external LUs. Thus, we can say that, while external LUs are often used for general purpose logging, internal LUs are likely to be created for better customization of logging format and configuration.

In our context of security-relevant logs, we want to analyze if external LUs (general-purpose logging utilities) were sufficient for developers, or if they rather implemented internal LUs that could allow more customization.

For this, we look into the 415 projects that use LUs for both generic and security logging purposes: We find that 65% (272) of projects only use external LUs, 31% (128) use a mixture of internal and external LUs, and 4% (15) only use internal LUs. However, when it comes to security-relevant logs: 81% (335) of the projects use only external LUs, 14% (60) use a combination of external and internal, and only 5% (20) use internal LUs alone.

We further analyze all the internal LUs used in security-relevant logs, to see if any of them were specifically developed to handle security events. We find that only 4% (15) of the projects use a specific-purpose internal LU that include 'security' or 'audit' keywords somewhere in the imported package name. Moreover, all these projects also use other external LUs for security logging.

Takeaway: Our analysis shows that developers most often use general-purpose logging utilities for security-relevant logs. The development of security-specific LUs is very rare, and even in their existence, they are combined with other LUs in handling the security logs.

4.2.2 Analysis of Log Severity Levels. Log levels aim to categorize the logs according to the purpose of the log (e.g., debugging) and its severity (e.g., a failure or warning that needs to be addressed). Logging utilities often provide a number of log levels that are configured by default. Having carefully chosen log levels can help the incident respondents to better decide when to act, and to reduce the alert fatigue [39]. Previous work finds that developers often need to re-adjust log levels as an afterthought, to optimize the cost/benefit tradeoff of logging [64]. In practice, more verbose levels (info, debug, trace) are likely to be used during the software development phase, and to be disabled during deployment [27].

This section aims to analyze the use of log levels in security-relevant logging statements. We extract the log levels using regular expressions. As the different external and internal LUs in our dataset use slightly different naming for log levels, we combine the similar log level names to unify the naming: For instance, we combine "w, warn, warning, logwarn, logwarning, auditwarn" into a single category of "warn".

With this approach, 90% of logs are matched to eight main categories of log levels. 5% of logs use standard output without a log level. The remaining 5% use custom method names without a clear log severity level, such as: `.log()`, `.logevent()`, `.logall()`, `.recordevent()`, `.settext()`. However, some of these include the severity level within the log description. By searching for related keywords, we were able to extract the data for an additional 3% of logs.

Figure 2 summarizes the results from 93% of logs, with the 8 main categories of log levels. We see that around 38% of the logs use the `debug` or `trace` levels, which are aimed for fine-grained tracing and troubleshooting. These findings confirm the phenomenon described by Chuvakin et al. [21] as "the debugging logs masquerading as security audit logs": They suggest that debugging logs are more common in application logs compared to properly designed security audit logs, however, as they often miss the necessary details, they cannot be used in security investigations. Moreover, these logs might be disabled in production due to their high verbosity level that may lead to performance, memory or noise concerns.

Another 38% of logs correspond to log levels with low verbosity and high criticality (such as 'error, warn, severe, fatal'). However, we do not find any custom or specific log level related to security. Only 1.2% of logs include the 'audit' keyword in the log level.

Takeaway: We observe that security logging is not differentiated from the rest of the logging activity, and security logs often do not contain a specific tag that could make their processing more straightforward.

4.2.3 Logging Locations. In this section we analyze the types of code blocks in which security-relevant logs are located. We use the PROGEX [26] tool to identify the log location using the Abstract Syntax Tree (AST) representation of the Java source code. For 80% (26,975) of logging statements, we were able to successfully extract the type of code block.

Figure 3 depicts the results categorized under four types of code constructs: looping, selection, exception handling, and directly within the method or block. We find that majority of the security-relevant logs (55%) are located in a decision making block. This might be expected, as various security checks (e.g., authentication, authorization) are likely to be made via decision making constructs such as if-else or switch-case. Moreover, 22% of security logs are located in a catch block, and 16% are located directly inside a method.

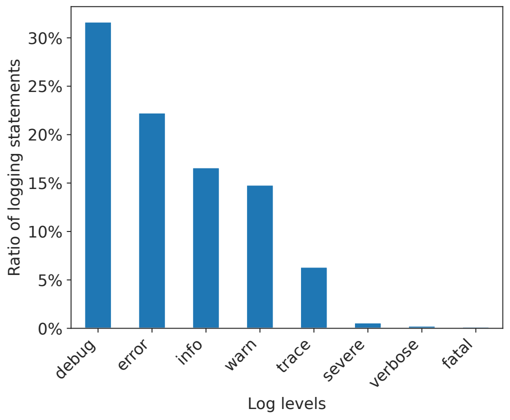

Figure 2: Log levels of security-relevant logs.

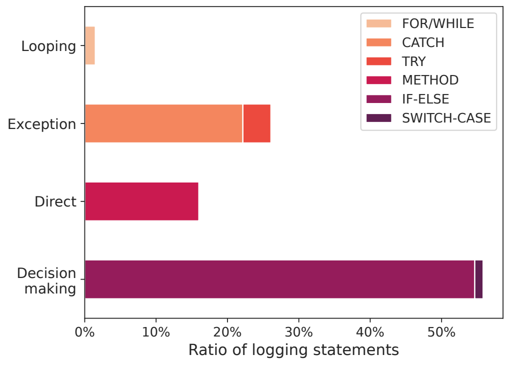

Figure 3: Logging locations.

We then take a deeper look into the types of exceptions caught in the catch blocks, where a security-relevant log is present. Our dataset contains 6,070 such catch blocks. We analyze the catch clauses, and whether the caught exception names include one of the security-relevant keywords we identified. Table 2 gives the top five most common exceptions and their percentages, for the two cases of exceptions (security-relevant and not). We can see that most of the logging statements (84%) reside in non-security-relevant, often generic exception catch blocks.

Takeaway: We find that most of the security-relevant logging statements are located in clauses that follow a decision making (branching) or exception try/catch statement. This shows that developers often log after checking whether a certain condition holds, or after the code blocks that might fail and raise an exception. However, our analysis on the exception catch statements again indicate a lack of specific consideration and handling of security-relevant events.

4.2.4 Event categories. The logging statements in our dataset relate to different types of security events. With a naive approach, we can correlate a logging statement to one of the event categories

	Security-relevant (16% overall)	Non-security-relevant (84% overall
1	PermissionException (2%)	Exception (35%)
2	AuthenticationException (1%)	IOException (5%)
3	CertificateException (1%)	NumberFormatException (4%)
4	AuthorizationDeniedException (1%)	IllegalArgumentException (3%)
5	SecurityException (0.8%)	Throwable (3%)

Table 2: Top 5 most common exceptions seen in the catch clauses where the security logs reside.

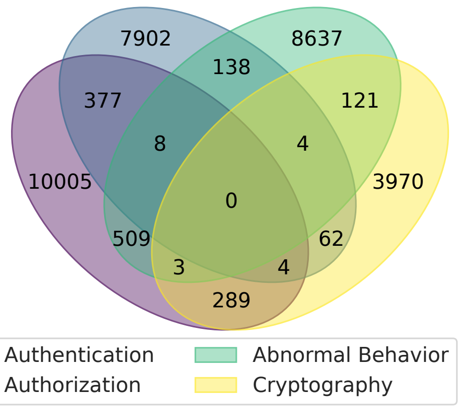

Figure 4: Venn diagram of the number of logs from the four categories.

(Authentication, Authorization, Abnormal Behavior, Cryptography), if the statement contains the respective keywords identified in Section 3. It is also possible that one logging statement relates to more than one event category. For instance, the following statement[10] logs an abnormal behavior related to authentication: *log.info("Replay attack detected - DENYING authentication request");*

Figure 4 shows a Venn diagram of the number of logs from the four event categories, and how they overlap. We observe that the largest portion of the logs (33%) relate to Authentication events, followed by Abnormal Behavior (28%) and Authorization (25%) categories. The largest overlaps are between Authentication & Authorization and Authentication & Abnormal Behavior.

Takeaway: While most of the logging guidelines we analyzed in Table 1 put less focus on the Abnormal Behavior category in comparison to Authentication and Authorization, we observe that the portions of logs in these categories are similar in real-world practice.

4.2.5 Methods containing the security logs. In this section we analyze the methods that include the security-relevant logging statements. In particular, we expect these methods to contain security-relevant code. Note that, as coding practices in the wild may not be perfect, a method might be dealing with more than one functionality, having low functional cohesion. Thus, a method with a logging statement of a certain event category (e.g. Authentication) might also include other types of events that need to be logged. In other words, there could be a one-to-many relationship between the methods and the event categories.

[10] Example taken from: https://github.com/vmware-archive/lightwave

For this analysis, we convert each method to a set of keyword tokens, *excluding the logging statements and Java-specific keywords*, and splitting the CamelCase strings. We then check if the method tokens include any of the security-relevant keywords we selected.

We find that 90% of the methods that have a logging statement in Authentication category also include authentication-relevant keywords in the method tokens. This ratio is 86% for Authorization and 93% for Cryptography categories. For these categories, Natural Language Processing (NLP) based techniques such as *topic analysis* [42], or classifiers that use term frequency analysis [25] can have a high success rate in suggesting logging locations.

On the other hand, for the Abnormal Behavior category, only 51% of the methods include tokens with the searched keywords. This is expected, as this category covers a large variety of events and the keywords related to abnormal behavior are less likely to be found in the source code. Thus, developing an automated log recommendation system might be more difficult for this category.

Takeaway: We find that, for certain event categories such as Authentication and Cryptography, the source code tokens can be useful to identify the code blocks that require logging.

4.3 The need for application-layer security logging

As briefly explained in Section I, web applications often have access logs generated by the web server, containing the HTTP request and responses. The information captured in HTTP access logs depends on the log configuration, however, theoretically it can include the full HTTP transaction such as the request and response body, and headers [33]. With this information, Web Application Firewalls or similar security analytics software can correlate the logs, infer various events (such as failed authentication), and look for attack patterns (such injection attempts) [47].

Developers may rely on HTTP access logs for part of the logging requirements. On the other hand, application-layer logs have the potential to provide more information about the internal state of the application and business logic related issues, which are not necessarily visible in the access logs.

In this section, we aim to answer the third research question (RQ3), that is to understand the added benefit of application-layer logging for security. We aim to find out the common reasons that developers log at the application layer, and how these logs can complement the HTTP access logs in terms of the contextual information they provide.

For this analysis, we randomly sample 100 logging statements from each of the four categories (400 in total) and manually review the logging statements (and their methods, when necessary). Our observations per category are as follows.

Authentication category: 43% of the authentication related logs give information about the status of authentication attempts

```
@PostMapping(path = "/account/reset-password/init")
public void requestPasswordReset(@RequestBody String mail) {
    Optional<User> user = userService.requestPasswordReset(mail);
    if (user.isPresent()) {
        mailService.sendPasswordResetMail(user.get());
    } else {
        //Pretend the request has been successful to prevent checking which emails really exist
        //but log that an invalid attempt has been made
        log.warn("Password reset requested for non existing mail");
    }}
```

a) Application: jhipster-sample-app-hazelcast, Java file: AccountResource.java [4].

```
if ((Arrays.binarySearch(roles, realmConfig.getAdminRoleName()) > -1 || isRoleHasAdminPermission) &&
        !realmConfig.getAdminUserName().equals(loggedInUserName)) {
   log.warn("An attempt to assign user to Admin permission role by user : " +
           loggedInUserName);
   throw new UserStoreException("Can not assign user to Admin permission role");
}
```

b) Application: carbon-identity-framework, Java file: MultipleCredentialsUserProxy.java [2].

```
HttpResponse<InputStream> response = callGetPayment(paymentId, configuration, paymentContext.getConfigurationLevel());
if(HttpUtils.callSuccessful(response)) {
    return processRemotePayment(transaction, paymentId, purchaseContext, configuration, response);
} else {
    if(response.statusCode() == 404) {
        log.warn("Received suspicious call for non-existent payment id "+paymentId);
        return PaymentWebhookResult.notRelevant("");
    }
```

c) Application: alf.io, Java file: MollieWebhookPaymentManager.java [1].

```
@Override
public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException {
    try {
        defaultViPRTrustManager.checkClientTrusted(chain, authType);
    } catch (CertificateException e) {
        log.debug("Client certificate was not trusted by default trust manager, checking accept all certs config.
                Certificate: "+ chain[0]);
        // if setting for accepting all connections is set to true
        if (KeyStoreUtil.getAcceptAllCerts(coordConfigStoringHelper)) {
            log.warn("The following certificate is not trusted." + chain[0]);
        } else {
            log.debug("Accept all certs is set to false, the certificate will not be trusted", e);
            throw e;
        }}}
```

d) Application: coprhd-controller, Java file: ViPRX509TrustManager.java [3].

Figure 5: Example code pieces that motivate custom application logging.

such as login failures and successes. Although these events are often captured in the access logs, an important contextual information that application-layer logs can provide is the reason for failures. Our sample dataset includes various reasons such as invalid cookie, failure to connect to the authentication server, and failure to decrypt the password. However, only 26% of the authentication failure related logs provide the reason.

The second most common theme is the issues related to password reset or update requests (19%). An interesting event observed in two different applications is to log the password change/reset requests for non-existing users. Figure 5-(a) shows an example method, where the application pretends that the password reset email is sent to the non-existing user, instead of returning an error in the HTTP response. This way, the application aims to prevent user enumeration attacks (i.e., attacks trying to find out the registered users [24]). This invalid attempt will only be logged by the application, as it cannot be observed in the HTTP access logs.

Authorization category: Similar to the previous category, most of the authorization related logs are about the status of access requests, such as denied or granted access (44%). The second most common case is to log various information about a requested role (29%). An interesting example is given in Figure 5-(b), where the application logs a suspicious attempt to assign admin role to a user. This attempt may not be obvious in HTTP traffic, unless a specific response is returned.

Abnormal Behavior category: 56% of the statements in our sample log invalid values (such as parameters, configuration, database query, password). The second most common case is to log expired credentials and access/session tokens (11%). Again, the reasons for validation failures and expirations may not be visible in HTTP access logs. An interesting example is shown in Figure 5-(c): The application interacts with a third-party payment service, and generates a log in case the payment service returns an error due to a non-existing payment ID. This condition might indicate a parameter tampering attempt [48]. While the HTTP logs might show that the payment has failed, they may not capture the actual reason for the error (i.e., "invalid payment ID"), which means less information will be available for attack investigation.

Cryptography category: 55% of the statements in our sample relate to the creation, deletion, and use of certificates, and the errors happening during this process. Moreover, 24% of logs give information about the encryption or decryption processes and keys. Figure 5-(d) shows an example where the application logs whether it accepts an untrusted certificate, after checking the connection settings. Again, such information may not be visible to the web server, as it relates to the internal configuration of the application, and the client's certificate will be accepted.

These examples show the necessity of custom application-layer security logging in gaining visibility into the application runtime, and why HTTP logs alone may not be enough for attack detection and investigation.

On the other hand, among the 400 logging statements that we analyze manually, we find that 24% do not log any variables, and only 31% include an identity variable related to the user, session, or resource. Thus, from a forensics perspective, it's likely that these logs also miss certain data that could be useful in attack investigations.

5 DISCUSSION

Our study starts with the observation that insufficient logging and monitoring is one of the most common application security risks. Our work reveals three main challenges related to security-relevant logging.

The first challenge (revealed through RQ1 and RQ3) is **to make sure that all the necessary security events are logged.** Our analysis on guidelines (Section 3) shows that a large number of events needs to be considered, even before thinking of the application-specific issues. Moreover, the analysis in Section 4.3 shows that developers cannot just rely on HTTP access logs for all security-relevant events, and application-layer logging is a necessity.

The second challenge is **the ability to easily filter out the security-relevant logs**. While answering RQ2, we find that differentiating the security-relevant logs from the rest of the logging activity is very difficult. We find that the logs rarely have a security specific tag or log level, moreover, most of the events are logged at the debug level, using generic logging utilities.

The third challenge is **to make sure that important information is not missing in the logs**. This is a challenging problem, and our preliminary analysis in Section 4.3 shows that most of the logs fail to include important information such as user ids and reasons for failures.

We believe that these challenges can be addressed with better quality logging guidelines and automated methods to support developers in logging decisions.

The guidelines can be improved to explain the reasoning behind the events that need to be logged, providing examples of the different types security issues and attacks related to these events. For instance, the guidelines could explain that an invalid cookie should be logged, as it might indicate the tampering of the HTTP requests. A recent study [59] that interviews developers about their logging practices also notes that more than 60% of the developers would wish for pragmatic logging guidelines and automated logging tools.

Although many studies aim to automate or improve logging (e.g., by recommending log locations or adding the missing variables to be logged), these studies often do not consider security-relevant logging [17, 27], or focus on a specific issue (such as permission over-granting [60]). Our work shows that the use of NLP techniques based on security-relevant code tokens can be promising for this task. Finally, only few studies address the forensicability of logs (e.g., for non-repudiation [35]) and forensic-readiness of software systems [51, 58]. We believe that further efforts are needed to develop methods to define, measure, and ensure the forensicability of the application-layer logging statements.

6 CONCLUSION

In this paper we study the problem of insufficient security logging in web applications. We start by analyzing several security logging guidelines, extracting a set of events that need to be considered as a baseline. We draw attention to the difficulty of complying with the abstract requirements in the guidelines and making sure that the logs include the right amount of information for attack investigation. We then make a large scale analysis of security-relevant logging statements in 472 Java applications. Our analysis demonstrates the importance of application-layer security logs and suggests possible areas of improvement; such as the use of specific log levels or identifiers, to distinguish security logs from the rest of the logging activity and to make them easier to process by the incident response teams. Finally, our work shows the need for future research to better identify the forensics related requirements in security-relevant logs, and to develop new methods to automate or facilitate the logging of the necessary information.

ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation programme under project TESTABLE, grant agreement No 101019206.

REFERENCES

[1] alf.io. https://github.com/alfio-event/alf.io/.

[2] carbon-identity-framework. https://github.com/wso2/carbon-identity-framework.

[3] coprhd-controller. https://github.com/CoprHD/coprhd-controller.

[4] jhipsterHazelcastSampleApplication. https://github.com/jhipster/jhipster-sample-app-hazelcast.

[5] Scoold - Stack Overflow in a JAR. https://github.com/Erudika/scoold/.

[6] Complete guide to GDPR compliance. https://gdpr.eu/, 2022.

[7] Apache sling - bringing back the fun! https://sling.apache.org/, 2023.

[8] Build java web applications faster with vaadin. https://vaadin.com/, 2023.

[9] Australian Cyber Security Center. Information Security Manual. https://www.cyber.gov.au/resources-business-and-government/essential-cyber-security/ism, DECEMBER 2021.

[10] Cloud Controls Matrix Working Group, Cloud Security Alliance. Cloud Controls Matrix v4.0.5. https://cloudsecurityalliance.org/artifacts/cloud-controls-matrix-v4/, July 2021.

[11] FedRAMP. FedRAMP Continuous Monitoring Strategy Guide Version 3.2. https://www.fedramp.gov/assets/resources/documents/CSP_Continuous_Monitoring_Strategy_Guide.pdf, April 2018.

[12] Yasemin Acar, Sascha Fahl, and Michelle L Mazurek. You are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 3–8. IEEE, 2016.

[13] Sepehr Amir-Mohammadian, Stephen Chong, and Christian Skalka. Correct audit logging: Theory and practice. In Frank Piessens and Luca Viganò, editors, *Principles of Security and Trust*, pages 139–162, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[14] Adam Bates, Wajih Ul Hassan, Kevin Butler, Alin Dobra, Bradley Reaves, Patrick Cable, Thomas Moyer, and Nabil Schear. Transparent web service auditing via network provenance functions. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 887–895, 2017.

[15] Luke Taylor Ben Alex. Spring Security. https://docs.spring.io/spring-security/site/docs/3.1.x/reference/springsecurity-single.html, 2022.

[16] Ivan Blagojević. Most popular programming languages. https://99firms.com/blog/most-popular-programming-languages/#gref, 2023.

[17] Jeanderson Cândido, Jan Haesen, Maurício Aniche, and Arie van Deursen. An exploratory study of log placement recommendation in an enterprise system. Proceedings - 2021 IEEE/ACM 18th International Conference on Mining Software Repositories, MSR 2021, 2021.

[18] Boyuan Chen and Zhen Ming Jiang. Replication package. https://www.eecs.yorku.ca/~chenfsd/resources/icse2020_replication.zip, 2020. Accessed: 02/02/2023.

[19] Boyuan Chen and Zhen Ming Jiang. Studying the use of java logging utilities in the wild. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 397–408. IEEE, 2020.

[20] Boyuan Chen and Zhen Ming Jack Jiang. Characterizing logging practices in java-based open source software projects–a replication study in apache software foundation. *Empirical Software Engineering*, 22(1):330–374, 2017.

[21] A. Chuvakin, K. Schmidt, and C. Phillips. *Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management*. Elsevier Science, 2012.

[22] Anton Chuvakin and Gunnar Peterson. Logging in the age of web services. *IEEE Security Privacy*, 7(3):82–85, 2009.

[23] Anton Chuvakin and Gunnar Peterson. How to do application logging right. *IEEE Security Privacy*, 8(4):82–85, 2010.

[24] CodePath. Username Enumeration. https://guides.codepath.com/websecurity/Username-Enumeration, 2023. Accessed: 12/01/2023.

[25] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*, page 24–33, New York, NY, USA, 2014.

[26] Seyed Mohammad Ghaffarian. PROGEX (Program Graph Extractor). https://github.com/ghaffarian/progex, 2019.

[27] Sina Gholamian and Paul Ward. A comprehensive survey of logging in software: From logging statements automation to log mining and analysis. June 2022.

[28] Lindsay Goodspeed. PCI DSS v4.0 Resource Hub. https://blog.pcisecuritystandards.org/pci-dss-v4-0-resource-hub, MARCH 2022.

[29] Wajih Ul Hassan, Mohammad A. Noureddine, Pubali Datta, and Adam Bates. Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis. In *NDSS*, 2020.

[30] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. A survey on automated log analysis for reliability engineering. *ACM Comput. Surv.*, 54(6), July 2021.

[31] ISO/IEC. ISO/IEC 27017:2015 Information technology — Security techniques — Code of practice for information security controls based on ISO/IEC 27002 for cloud services. iso27017https://www.iso.org/standard/43757.html, 2015. Accessed: 08/03/2022.

[32] ISO/IEC. ISO/IEC 27002:2022 Information security, cybersecurity and privacy protection — Information security controls. https://www.iso.org/standard/75652.html, 2022. Accessed: 15/05/2022.

[33] Ivan Ristić. Mod Security Handbook: Getting Started - Audit Log. https://www.feistyduck.com/library/modsecurity-handbook-free/online/ch04-logging.html, 2017. Accessed: 05/01/2023.

[34] Murugiah Souppaya Karen Kent. NIST SP 800-92 guide to computer security log management. https://csrc.nist.gov/publications/detail/sp/800-92/final, 2006. Accessed: 15/05/2022.

[35] Jason King. Measuring the forensic-ability of audit logs for nonrepudiation. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1419–1422, 2013.

[36] Jason King, Rahul Pandita, and Laurie Williams. Enabling forensics by proposing heuristics to identify mandatory log events. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, pages 1–11, 2015.

[37] Jason King and Laurie Williams. Cataloging and comparing logging mechanism specifications for electronic health record systems. In *2013 USENIX Workshop on Health Information Technologies (HealthTech 13)*, Washington, D.C., August 2013. USENIX Association.

[38] Jason King and Laurie Williams. Log your crud: design principles for software logging mechanisms. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, pages 1–10, 2014.

[39] Rafal Kuć. Understanding logging levels: What they are & how to use them. https://sematext.com/blog/logging-levels/, 2022.

[40] LaTisha Raulston-Sloderbeck. FedRAMP Weekly Tips And Cues – November 14, 2018. https://cfocussoftware.com/office-365-government/fedramp-weekly-tips-cues-november-14-2018/, November 2018.

[41] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. High accuracy attack provenance via binary-based execution partition. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013.

[42] Heng Li, Tse-Hsun Peter Chen, Weiyi Shang, and Ahmed E Hassan. Studying software logging using topic models. *Empirical Software Engineering*, 23(5):2655–2694, 2018.

[43] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. Protracer: Towards practical provenance tracing by alternating between logging and tainting. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[44] NIST. NIST special publication 800-series general information. https://www.nist.gov/itl/publications-0/nist-special-publication-800-series-general-information, 2018. Accessed: 15/05/2022.

[45] NIST Joint Task Force. NIST SP 800-53 rev.5 security and privacy controls for information systems and organizations. https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final, 2020. Accessed: 15/05/2022.

[46] OWASP. OWASP Top 10:2021. https://owasp.org/Top10/, 2021. Accessed: 15/02/2022.

[47] OWASP. OWASP ModSecurity Core Rule Set. https://coreruleset.org/, 2023. Accessed: 05/01/2023.

[48] OWASP. Web Parameter Tampering. https://owasp.org/www-community/attacks/Web_Parameter_Tampering, 2023. Accessed: 12/01/2023.

[49] OWASP CheatSheets Series Team. Logging Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html, 2021. Accessed: 04/07/2022.

[50] OWASP Top 10 Team. A03:2021 – Injection. https://owasp.org/Top10/A03_2021-Injection/, 2021. Accessed: 04/07/2022.

[51] Liliana Pasquale, Dalal Alrajeh, Claudia Peersman, Thein Tun, Bashar Nuseibeh, and Awais Rashid. Towards forensic-ready software systems. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '18, page 9–12. Association for Computing Machinery, 2018.

[52] Antonio Pecchia, Marcello Cinque, Gabriella Carrozza, and Domenico Cotroneo. Industry practices and event logging: Assessment of a critical software development process. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 169–178, 2015.

[53] PortSwigger. Access control vulnerabilities and privilege escalation. https://portswigger.net/web-security/access-control, 2023. Accessed: 02/01/2023.

[54] PortSwigger. Business logic vulnerabilities. https://portswigger.net/web-security/logic-flaws, 2023. Accessed: 02/01/2023.

[55] PortSwigger. XML external entity (XXE) injection. https://portswigger.net/web-security/xxe, 2023. Accessed: 02/01/2023.

[56] QOS.CH. Simple Logging Facade for Java (SLF4J). https://www.slf4j.org/.

[57] Maria Riaz, Jason King, John Slankas, and Laurie Williams. Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 183–192, 2014.

[58] Fanny Rivera-Ortiz and Liliana Pasquale. Towards automated logging for forensic-ready software systems. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 157–163, 2019.
[59] Guoping Rong, Shenghui Gu, Haifeng Shen, He Zhang, and Hongyu Kuang. How do developers' profiles and experiences influence their logging practices? an empirical study of industrial practitioners. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 855–867, 2023.
[60] Bingyu Shen, Tianyi Shan, and Yuanyuan Zhou. Improving logging to reduce permission over-granting mistakes. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, August 2023.
[61] Erico Guizzo Stephen Cass, Preeti Kulkarni. Top Programming Languages 2022. https://spectrum.ieee.org/top-programming-languages-2022/, 2022.
[62] TIOBE. Tiobe index for march 2023. https://www.tiobe.com/tiobe-index/, 2023.
[63] U.S. Department of Health & Human Services. The HIPAA The Security Rule. https://www.hhs.gov/hipaa/for-professionals/security/index.html, Jan 2013.
[64] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. Characterizing logging practices in open-source software. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 102–112, 2012.
[65] Jun Zeng, Zheng Leong Chua, Yinfang Chen, Kaihang Ji, Zhenkai Liang, and Jian Mao. Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics. In *NDSS*, 2021.
[66] Cheng Zhang, Zhenyu Guo, Ming Wu, Longwen Lu, Yu Fan, Jianjun Zhao, and Zheng Zhang. Autolog: Facing log redundancy and insufficiency. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, APSys '11. Association for Computing Machinery, 2011.
[67] Jieming Zhu, Pinjia He, Qiang Fu, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. Learning to log: Helping developers make informed logging decisions. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 415–425, 2015.

A GROUPING OF THE EVENT DESCRIPTIONS FROM GUIDELINES

We provide the details of our analysis in Section 3.1, showing how the events listed in Table 1 corresponds to the raw text collected from the guidelines.

- **Authentication Category**
  - Account management events
    - Use and management of user IDs and authentication information (ISO 27002)
    - Account management (FEDRAMP)
    - User or group management (ACSC-OS section)
    - Account management events (CCM)
  - Changes to authentication mechanisms
    - Use of and changes to identification and authentication mechanisms (CCM)
  - Authentication checks
    - Authentication checks (FEDRAMP)
    - Authentication checks (CCM)
  - Successful authentications
    - Successful account logon (FEDRAMP)
    - Successful account login (CCM)
    - Log-on (ISO 27002)
    - Logon (ACSC)
    - Successful authentication attempts (NIST)
    - Authentication success (OWASP)
    - Database log-ons (ACSC-DB section)
  - Failed authentication attempts
    - Unsuccessful account logon (FEDRAMP)
    - Unsuccessful account login (CCM)
    - Failed authentication attempts / logons (NIST)
    - Failed logon (ACSC)
    - Authentication failures (OWASP)
    - Login failures (CCM)
  - Log-off
    - Log-off (ISO)
    - Logoff (ACSC)
    - Database log-offs (ACSC-DB section)
  - Assignment of users to tokens
    - Assignment of users to tokens (FEDRAMP)
    - Assigning users to tokens (OWASP)
  - Addition of new tokens
    - Addition of tokens (FEDRAMP)
    - Adding tokens (OWASP)
  - Removal of tokens
    - Removal of tokens (FEDRAMP)
    - Deleting tokens (OWASP)
  - Creation of new users / accounts
    - Addition of new users (ACSC-DB section)
    - Account creation (NIST)
    - Creation of identites (ISO 27002)
    - Addition of users (OWASP)
    - Addition of users (FEDRAMP)
    - Additions to accounts with root or administrative privileges (CCM)
  - Deletion of users / accounts
    - Account deletion / removal (NIST)
    - Deletion of identities (ISO 27002)
    - Removal of users (FEDRAMP)
    - Deletion of users (OWASP)
    - Deletions to accounts with root or administrative privileges (CCM)
  - Account modifications
    - Account changes / modification (NIST)
    - Changes to accounts (ACSC-OS section)
    - Password changes (NIST)
    - Modification of identities (ISO 27002)
    - Changes to accounts with root or administrative privileges (CCM)
    - Account enabling (NIST)
    - Account disabling (NIST)
  - Credential usage
    - PIV credential usage (NIST)
    - External credential usage (NIST)
- **Authorization**
  - Successful authorizations
    - Successful system/data/resource access attempts (ISO)
    - Successful attempted access (ACSC-DB section)
    - When access was granted (ACSC-Personnel Security section)
  - Failed authorization attempts
    - Failed accesses (NIST)
    - Attempted access that is denied (ACSC)
    - Invalid access attempts (CCM)
    - Authorization (access control)a failures (OWASP)
    - Rejected system/data/resource access attempts (ISO)
    - Failed attempts to access data and system resources (ACSC-OS section)
    - Unsuccessful attempted access (ACSC-DB)
  - Authorization checks
    - Authorization checks (CCM)
    - Authorization checks (FEDRAMP)

- Privilege assignment
  - Account privilege assignment (NIST)
  - Access rights granted to a user ID (ISO 27002)
  - All privileges allocated (ISO 27002)
- Use of privileges and privileged functions
  - Use of privileges (NIST)
  - Use of privileges (ISO 27002)
  - Use of special privileges (ACSC-OS section)
  - Execution of privileged functions (NIST)
  - Privilege functions (CCM)
  - Privilege functions (FEDRAMP)
  - All privileged access to systems (ISO)
  - Attempts to use special privileges (ACSC-OS section)
  - Use of privileged access (ACSC-Personnel Security section)
  - Use of privileged accounts (CCM)
- Administrative activity
  - Use of systems administrative privileges (OWASP)
  - Access by application administrators (OWASP)
  - All actions by users with administrative privileges (OWASP)
  - All administrator activity (FEDRAMP)
  - Actions by users with administrative privileges (FEDRAMP)
  - Actions taken by any individual with root or administrative privileges (CCM)
  - Administrative privilege usage (NIST)
  - All administrator activity (CCM)
  - Management of system administrative privileges access (FEDRAMP)
  - Database administrator actions (ACSC-DB section)
- Changes to permissions
  - Changes to user roles, database permissions (ACSC-DB section)
  - Permission/policy changes (FEDRAMP)
  - Permission/policy changes (CCM)
- Changes to privileges
  - Changes to privileged accounts and groups (ACSC-Personnel Security section)
  - Changes to privileges (OWASP)
  - When the level of access was changed (ACSC-Personnel Security section)
  - Management of changes to privileges (FEDRAMP)
  - Changes to users' logical and physical access rights (ISO 27002)
  - Elevation of privileges (CCM)
  - When access was withdrawn (ACSC-Personnel Security section)
  - Changes to privileged accounts (CCM)
- Attempts to elevate privileges/permissions
  - Attempts to elevate privileges (ACSC-DB section)
  - Attempts to escalate permissions (CCM)
- Access to (important) data and objects
  - Access to important data and processes (ACSC-OS section)
  - Access to particularly important data (ACSC-DB section)
  - Object access (FEDRAMP)
  - Object access (CCM)
  - Individual user accesses to systems (CCM)
  - Recording who accesses information (ISO 27002)

- **Data and user transactions**
  - User transactions, request and responses
    - Client requests and responses (NIST)
    - DNS and HTTP requests (ACSC-OS section)
    - transactions executed by users in applications (ISO 27002)
  - Database queries
    - Any query containing comments, multiple embedded queries (ACSC-DB section)
    - Query parameters (NIST)
    - Search queries initiated by users (ACSC)
  - Database failures
    - Any query or database alerts or failures (ACSC-DB section)
  - Changes to database structure
    - Changes to database structure, (ACSC-DB section)
  - Modifications to data
    - Modifications to data (ACSC-DB section)
    - Data changes (FEDRAMP)
    - Data changes (CCM)
    - Data changes (OWASP)
  - Data deletions
    - Data deletions (CCM)
    - Data deletions (FEDRAMP)
  - Data access
    - Data access (CCM)
    - Data access (FEDRAMP)
  - Import and export of data
    - Import and export of data, including screen-based reports (FEDRAMP)
    - Data import (OWASP)
    - Data export (OWASP)
  - Submission of user generated content
    - Submission of user-generated content, especially file uploads (FEDRAMP)
    - Submission of user generated content – especially file uploads (OWASP)
  - File accesses
    - File access successful/unsuccessful (optional) (NIST)
    - Files accessed and type of access (ISO 27002)

Abnormal behavior

- Input validation failures
  - Input validation failures e.g., invalid parameter names and values (OWASP)
  - Use of manual override capability of input validation (NIST)
- Output validation failures
  - Output validation failures e.g., database record set mismatch (OWASP)
- Potential integrity violation
  - Potential integrity violation (NIST)
- Session management failures
  - Session management failures e.g., session cookie modification (OWASP)
- Sequencing failures
  - Sequencing failure (OWASP)

  - Suspicious, unexpected behavior
    - Detection of suspicious activity (CCM)
    - Suspicious, unacceptable or unexpected behavior (OWASP)
  - Fraud and other criminal activities
    - Fraud and other criminal activities (OWASP)
  - Excessive usage
    - Usage information for security monitoring (number of transactions in a certain period) (NIST)
    - Excessive use (OWASP)
- **Cryptography**
  - Use of data encrypting keys
    - Use of data encrypting keys (FEDRAMP)
    - Use of data encrypting keys (OWASP)
    - Key usage (CCM)
  - Key management activities
    - Key management related activities (ISO 27002)
    - Key generation, usage, purpose, rotation, revocation, destruction, activation, suspension etc. (CCM)
  - Management of key changes
    - Management of key changes (FEDRAMP)
    - Key changes (OWASP)
- **Operational events**
  - Application/system startup
    - Application startup (NIST)
    - Application startup (OWASP)
    - System startup (ASCS-OS section)
  - Application/system shutdown
    - Application shutdown (OWASP)
    - Application shutdown (NIST)
    - System shutdown (ACSC-OS section)
  - Configuration changes
    - Application config changes (NIST)
    - Security/privacy attribute changes (NIST)
    - System accesses associated with configuration changes (NIST)
    - Configuration changes (OWASP)
    - Modifications to configuration (OWASP)
    - Established configurations and changes to software, services, networks (ISO 27002)
    - Changes to system configuration (ISO)
    - Changes to system configurations (ACSC-OS section)
    - Policy change (FEDRAMP)
  - Application failures and errors
    - Application crashes and error messages (ACSC-OS section)
    - Application failures (NIST)
    - Application errors (OWASP)
    - Crashes and any error messages (ACSC)
    - Service failures and restarts (ACSC-OS section)
    - Syntax and runtime errors (OWASP)
  - Application code file / memory changes
    - Application code file / memory changes (OWASP)
  - File system errors
    - File system errors (OWASP)
  - System events
    - System events (FEDRAMP)
    - System events (CCM)
    - System events (OWASP)
  - Process tracking
    - Process tracking (CCM)
    - Process tracking (FEDRAMP)
  - Use and management of network connections
    - Management of network connections (FEDRAMP)
    - Use of network connections (OWASP)
  - Creation and removal of system level objects
    - Creation and removal of system level objects (FEDRAMP)
    - Creation and deletion of system-level objects (CCM)
    - Creation and deletion of system level objects (OWASP)
  - Performance issues
    - Performance issues (OWASP)
  - Connectivity problems
    - Connectivity problems (OWASP)
  - Third party service errors (OWASP)