

Engineering Forensic-Ready Software Systems Using Automated Logging

Fanny Rivera-Ortiz^{1,2}

¹University College Dublin, Belfield, Dublin 4, Dublin Ireland

²Lero, The SFI Software Research Centre, Ireland

Abstract

A forensic-ready software system can produce logs that represent potential evidence in digital forensic investigations of potential security incidents. Logs can reveal the identity of offenders, what happened during an incident and when and how the incident was performed. However, it is hard for developers to implement security logging, especially in large software systems, because it requires identifying only the relevant and necessary information to be logged to support a digital forensics investigation. Moreover, developers might not have the security expertise to think in advance about potential security incidents that can occur. Existing approaches in the software engineering community have mainly supported logging to diagnose and fix either run-time or performance errors, without considering security logging. Other research approaches have focused on engineering forensic-ready software systems, without supporting the automated generation of security logs. The purpose of this PhD research is to assist software developers to build forensic-ready software systems, which can implement logging functionality to support digital investigations of potential security incidents. We propose a semi-automated approach called Forensic-Ready Logger (FRL). Our approach help developers to elicit a set of potential software misuse scenarios, expressed as annotated UML Sequence Diagrams. Such diagrams are subsequently used to identify the exact location where logging instructions should be placed and the information they should log. Finally, The Forensic-Ready Logger approach automatically injects logging instructions within the software system.

We want to evaluate first the conclusion validity and the performance of the Forensic-Ready Logger approach using a Computational Study where we will use a human resource management application built from scratch and a third-party open-source software system. Second, We will also assess the effectiveness of the Forensic-Ready Logger approach by using a Judgement study where we will seek software practitioners to provide feedback on its usefulness and usability.

Keywords

forensic readiness, logs, security incidents, software systems,

1. Introduction


A forensic-ready software system can produce logs that represent potential evidence in digital forensic investigations of potential security incidents. Logs can reveal the identity of offenders, what happened during an incident, and when and how the incident was performed [1, 2, 3].

In: J. Fischbach, N. Condori-Fernández, J. Doerr, M. Ruiz, J.-P. Steghöfer, L. Pasquale, A. Zisman, R. Guizzardi, J. Horkoff, A. Perini, A. Susi, M. Daneva, A. Herrmann, K. Schneider, P. Mennig, F. Dalpiaz, D. Dell'Anna, S. Kopczyńska, L. Montgomery, A. G. Darby, and P. Sawyer (eds.): Joint Proceedings of REFSQ-2022 Workshops, Doctoral Symposium, and Poster & Tools Track, Birmingham, UK, 21-03-2022, published at <http://ceur-ws.org>

✉ fanny.riveraortiz@ucdconnect.ie (F. Rivera-Ortiz)

ORCID [0000-0002-6853-6456](https://orcid.org/0000-0002-6853-6456) (F. Rivera-Ortiz)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

Therefore, it is crucial to add logging mechanisms in software systems to collect valuable information about security incidents that may occur. However, it is hard for developers to implement security logging, especially in large software systems, because it requires identifying only the relevant and necessary information to be logged to support digital investigations [1]. Additionally, developers might not have sufficient security expertise [1, 4] since they mainly focus on implementing functional and performance requirements [5] or fixing bugs [1] during their daily activities.

The purpose of this research is to help software developers to build forensic-ready software systems, which can implement logging functionality to support digital forensics investigations. Hence, developers need to decide where to log, which are the locations where logging statements should be placed, and what to log, which is the information to be recorded in the logs [1]. In this Ph.D. research, we propose a semi-automated approach called Forensic-Ready Logger (FRL)¹. The Forensic-Ready Logger Approach helps developers elicit a set of potential software misuse scenarios, expressed as annotated UML Sequence Diagrams. Such diagrams are subsequently used to identify the exact location where logging instructions should be placed and the information they should log. Finally, the Forensic-Ready Logger approach automatically injects logging instructions within the software system implementation.

This research abstract is organised as follows. Section 2 presents previous research related to Logging in the Software Engineering community. Section 3 addresses the research questions that our work considers and describes the Forensic-Ready Logger approach. Section 4 discusses the methodology and the evaluation we use, and Section 5 concludes.

2. Related Work

In the Literature review, we found that Logging has categories such as Information Systems (Workflows), Databases, and Software Engineering. Our research focused on Logging into Software Systems for Security purposes. Hence, our research does not consider the other two categories.

Previous software engineering research has focused on understanding how developers implement logging in software systems. For example, Chen and Ming [6] conducted an empirical study to understand which logging utilities are used by developers, how many logging utilities on average are used inside software projects, and the reasons associated with the use of multiple logging utilities in software systems (i.e., interaction between different logging utilities and formatting logging messages). Zhi et al. [7] conducted an empirical study with 10 open source and industrial projects written in Java to understand the current practice of logging configurations. Later, they presented 10 findings of logging configurations such as logging management, logging storage, logging formatting, and logging-configuration quality. Finally, they implemented a tool to detect invalid logs in logging configurations and reported these issues to developers. Other work, instead, proposed tools to automate the implementation of logging mechanisms to facilitate program comprehension, maintenance, and identify performance problems. For example, Li et al. [8] analysed manually seven open-source software systems to identify the placement of logging instructions. Based on this analysis, they propose a framework based on

¹A Java-based implementation of the system is available publicly at <https://github.com/friveraortiz/Projects.git>

deep learning to suggest logging locations at the block level. Then, Yao et al. [9] propose an automated approach, Log4Perf, which uses a statistical performance model of a software system to identify the locations that influence software performance. These represent the locations where logging should be performed to identify performance problems. However, all these approaches have not focused on analysing or facilitating security logging.

Researchers in the software engineering community have proposed approaches to guide security logging. For example, William and King [3] performed an empirical study in four open-source health care software systems to evaluate how logging practices can support the detection of security incidents. Also, they proposed seven principles indicating at a high level where and what information security logs should store (e.g., logs should indicate the action performed by the human user in the system). Ohmori [10] suggests storing network and server logs to detect a computer security incident. He defines the requirements for logging messages that should be recorded from computer networks and servers and proposes a new logging system architecture to support computer security incident response. However, these approaches provide rather high-level guidelines on how security logging should be performed and do not suggest exactly how logging statements should be implemented [11] by providing a tool to implement in a software system.

Finally, another line of work has tried to incorporate satisfaction of forensic readiness requirements [12] in the software development life-cycle. These requirements refer to the capability of software to collect and produce forensically sound evidence that is tamper-proof and admissible in court. Daubner et al. [13] propose high-level and unproved guidelines to verify whether a software system generates potential evidence and, thus, can be considered forensic-ready. Yu et al. [14] provides an approach to collect evidence related to drone incidents that is forensically sound and that could demonstrate whether the drone violated any regulations. Although this research supports evidence collection to facilitate drone incidents investigations, such evidence is only collected from physical sensors recording drone flight data. Alrajeh et al. [15] propose an approach to represent evidence preservation requirements considering the preservation of the minimal amount of data that would be relevant to a future digital investigation. This approach automatically generates a software specification indicating how to store minimal and relevant to investigate a security incident. However, it does not automate the implementation of logging statements within software systems.

Therefore, more effort is required to propose the usage of logs for security inside Software Systems[11] to provide information for forensic analysis to help answer who, what, when, where, and how a security incident happened[3][2][11][1].

3. Research Direction

3.1. Research Questions

The purpose of this research is to help software developers to build forensic-ready software systems, which can implement logging functionality to support digital forensics investigations. To achieve this aim, we answer the following research questions:

1. **RQ1:** How can we elicit security logging requirements that can be traced to software components and method invocations?

2. **RQ2:** How can we generate automatically logging instructions that comply with forensic-ready software systems requirements, such as relevance and minimality?

To answer RQ1 we provide an approach to generate automatically a model of a security incident by emulating the misuse of a software system. This model is represented as a sequence diagram, where message exchanges have a direct mapping to method invocations. We also allow annotating this model by specifying conditions that determine whether an incident is occurring. To answer RQ2, we use the information provided in the incident model to identify where logging should be implemented and which information should be logged. Finally, we propose an approach to instrument the software system to inject the logging statements at specific points of the software system implementation.

3.2. The Forensic-Ready Logger Approach

We propose a semi-automated approach named Forensic-Ready Logger(FRL) [1]² to guide developers to implement logging statements inside software systems and to support organisations in the detection and investigation of software misuses [11].

The kind of software misuses that our FRL approach considers are based on the STRIDE[16] model. This is a model of threats that validates the application design. Some examples of threats are spoofing identity, tampering with data, repudiation, information disclosure, the elevation of privilege. Such software misuses are relevant to discover in a software system because they provide accountability of bad behaviour conducted by personnel in the organisation (e.g., Software Administrators, Managers, employees) or former employees that might have permissions to software systems and could seek revenge against the organisation[17].

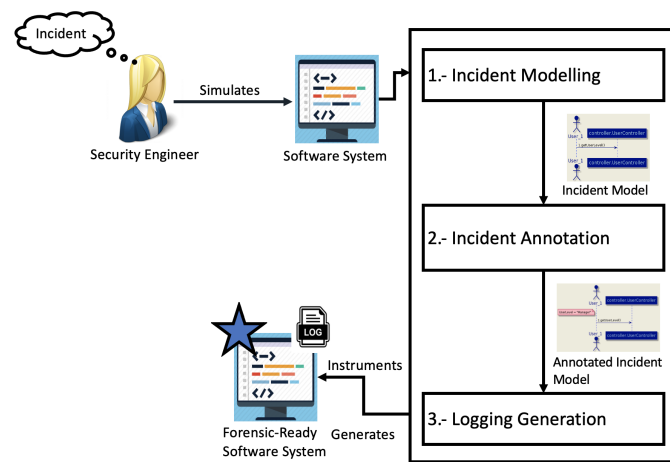


Figure 1: The Forensic-Ready Logger Approach

The Forensic-Ready Logger approach provides the following main functionalities:

²A Java-based implementation of the system is available publicly at <https://github.com/friveraortiz/Projects.git>

1.- Incident Modelling: We assume that a security engineer, who is the expert in security inside the organisation, runs a software system usage scenario that emulates an incident of interest.

The Security Engineer will decide which incidents are relevant to emulate based on previous work conducted by a security team in the organisation. First, the security team elicits a set of possible abuse cases[18] that describe the system's behaviour under attack. These abuse cases are related to possible software misuse scenarios. Second, the security team does a risk analysis[18], where they rank the abuse cases based on the priority of mitigating such risk. The Security Engineer will test the abuse cases using Penetration Testing[18, 19] to act as an adversary user[20].

An example of a misuse scenario is an insider, a system administrator of a human resource management system, exploits their privileges to impersonate a line manager and approve their own travel request. This scenario represents a security incident because it violates an organisation's policy claiming that only a manager can authorise a travel request [1].

From the execution of a software misuse, we provide a technique to generate an incident model, i.e. a UML sequence diagram that represents the steps of the security incident by referring specifically to software system components (e.g., packages, classes, and methods) [1] that relate to data changes without including the entire software behaviour[11]. The incident model is generated by instrumenting the software system using Aspect-Oriented Programming (AOP) [21]. This allow us to keep track of the sequence of method invocations that occur in a system when an incident happen. Note that, instead of recording all the method invocations, we consider only methods invocations leading to data access or modifications to the database. This assumption is based on the following observations. First, one of the OWASP Top 10 vulnerabilities [22] is accessing and/or modifying sensitive information. Furthermore, King and Williams [3, 23] and Chuvakin[2, 24] advice to insert logging statements related to data changes (e.g., creation, modification, consultation, and destruction).

2.- Incident Annotation: During this stage, the Security Engineer annotates the UML sequence diagram including relevant information for logging. Such annotations include conditions that should be satisfied when a specific method is executed to indicate that a security incident is taking place. In our previous example, a user login should be recorded only when the role of the user is a manager. This is important because it can indicate situations in which a malicious actor is impersonating a manager. We allow expressing annotations on the value of the input and output parameters of a specific method.

3.- Logging Generation: In this final stage, the Forensic-Ready Logger approach instruments the software system to inject the logging statements in the locations and fields specified in the previous stage. We use again AOP to inject such logging statements. This allows creating a log statement every time a method that is part of the incident model and satisfies the condition specified in the annotation is executed. The information that the log statement will contain to answer the who, what, where, how the security incident happened inside the software system is a) Information about the Operating System such as, name, current time, IP Address, Hostname, and UserName (i.e., Mac OS X, 02/02/2022 01:42:51 PM GMT, 192.168.1.150, Bob-MacBook-Pro.local, bobsmith); b) Information about the Method like, name, return type, return value (i.e., model.TravelRequestDatabase.save, boolean, true); c) Information about the attribute such as name, type (i.e., status, class model.Status, Approved). and d) Information about the

annotation type (i.e., Method, Parameter or Return Value).

Jørgen Bøegh[25] presents the ISO/IEC 9126 software quality model used by developers and researchers, which describes the quality requirements for software (e.g., Functionality, Reliability, Usability, Efficiency, Maintainability, Portability)[25]. Our research considers the Efficiency and Usability quality requirements.

Our FRL approach can have two usages scenarios that can show the trade-offs between enabling the capability of detecting security incidents in the software systems and these quality requirements. The first scenario is, the Security Engineer selects manually the relevant methods and necessary conditions to log to diagnose the security incident. Such a scenario impacts Usability because there is required manual input from the Security Engineer. It also affects efficiency in a positive way because the FRL approach will have a shortlist of methods and will perform in less time. The second scenario is, all the methods are logged without any human intervention. This scenario causes Usability that no manual input is required and it saves time for the Security Engineer. About efficiency, it causes more overhead because more methods will be included and it will require more time to perform our FRL approach.

Our Forensic-Ready Logger(FRL) approach is novel for two reasons. First, it generates a UML Sequence Diagram automatically that presents the Incident Model, which includes the methods invoked when the misuse scenario is replayed in the software system. We decided to use such a Sequence diagram because we considered the work of Briand et al.[26], who suggested reverse-engineering a Java Application by instrumenting the application using AspectJ[21]. However, such a diagram included the entire software system behavior and all components inside the system. Labiche et al.[27] improved this approach by representing in the UML Sequence Diagram the necessary components such as caller and callee objects, method signature, class name, and line number of the call. We considered the work from Labiche et al.[27] and we included in the UML Sequence Diagram the essential information about Java methods (e.g., caller, callee objects, method signature, class name)[11]. However, even with this consideration, a software system can have a multitude of methods. Generating a UML sequence diagram that includes all the method invocations that occurred during a security incident, might result in an extremely extensive UML sequence diagram which can be extremely difficult for a security engineer to understand the locations and the places to detect the security incident. To reduce dramatically the number of methods included in the UML sequence diagram, we include only the methods invoked when the misuse scenario is replayed in the software system. Furthermore, these methods perform data access or changes in the database[1]. Hence, our Incident Model represented as a UML Sequence Diagram provides an understandable and synthetic version of the misuse scenario in the software system.

Second, our FRL approach provides a practical way to transform a software system into a forensic-ready software system by generating and injecting logging instructions in the system implementation. The logs generated by the Forensic-Ready Logger approach provide potential evidence to demonstrate how a security incident related to software misuse took place [11].

4. Evaluation

To evaluate our Forensic-Ready Logger approach, we are using a mixed-method. First, we evaluate the technical part of the Forensic-Ready Logger (FRL) approach. We will assess the conclusion validity[28] of the FRL approach by verifying whether it can log data that is relevant[12] to an incident and avoid collecting unnecessary data (minimality[12]). To assess relevance will evaluate if the Forensic-Ready Logger (FRL) approach creates automatically a UML Sequence Diagram that contains the actors, packages, classes, and methods that are called when a Security Incident occurs in the software system. We will assess whether the Forensic-Ready Logger (FRL) approach includes logging instructions that generate a log that contains information to detect when and how a security incident of interest occurs. We will also evaluate the performance of the Forensic-Ready Logger (FRL) approach to generate an incident model and the logging instruction. To assess the external validity[28], we will conduct a computational study[29] using Java Systems, a Human Resource Management System implemented from scratch³ and a large scale open-source system available publicly, such as OpenHospital⁴. Once we have finished evaluating the technical part, we will seek the participation of Software Engineers to assess the effectiveness of the Forensic-Ready Logger (FRL) approach by performing a judgement study[29] where we will ask them to use the Forensic-Ready Logger approach to provide feedback on its usefulness and usability[25]. The elements that the software engineers will evaluate in our FRL approach are, First, the practicality of our approach to generating the misuse scenarios in the Incident Model, they will assess how useful it is such model to detect the security incident that could happen in the Software System. Second, is the practice of annotating the Incident Model, where they will assess the common mistakes that happen while annotating such Model. Third, the format of the generated logs that diagnose security incidents, where developers will evaluate if such format is understandable to detect security incidents inside the software system.

5. Conclusion

To detect security incidents, software systems need to be prepared in advance. Hence, this research presents how to transform software systems into forensic-ready software systems. Such systems can generate logs that detect and diagnose potential security incidents. Furthermore, this research contributes with a practical tool for software developers to implement security incidents in Software Systems. To the best of our knowledge, this is one of the first approaches to propose the automated generation of logging instructions that can cover relevant security incidents [11]. We have completed the development of the Forensic-Ready Logger(FRL) approach. Our next step is to create more attack scenarios that could happen in the Human Resource Management Software System (HRM) and test them using our FRL approach. After that, we will evaluate our Forensic-Ready Logger approach in a more complex Java Software System using several attack scenarios, and finally, we will conduct a Case Study where we will seek the participation of software developers to evaluate the practicality and usefulness of our

³A Java-based implementation of the system is available publicly at <https://github.com/friveraortiz/Projects.git>

⁴A Java-based open-source system <https://github.com/informatici/openhospital>

Forensic-Ready Logger approach.

Acknowledgments

This work was partially supported by Science Foundation Ireland grants 13/RC/2094_P2 and 15/SIRG/3501.

References

- [1] F. Rivera-Ortiz, L. Pasquale, Automated modelling of security incidents to represent logging requirements in software systems, in: Proceedings of the 15th International Conference on Availability, Reliability and Security (ARES 2020), ACM, Virtual Event, Ireland, 2019, pp. 1–8.
- [2] A. Chuvakin, K. Schmidt, C. Phillips, Logging and Log Management. The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management., Syngress, United States of America, 2013.
- [3] J. King, L. Williams, Log your crud: Design principles for software logging mechanisms, in: Proceedings of the 2014 Symposium and Bootcamp on the Science of Security (HotSOS'14), ACM, Raleigh, North Carolina, USA, 2014, pp. 1–10.
- [4] H. Assal, S. Chiasson, “think secure from the beginning”: A survey with software developers, in: Proceedings of the 2019 Conference on Human Factors in Computing Systems (CHI'19), ACM, Glasgow, Scotland, UK, 2019, pp. 1–13.
- [5] D. Oliveira, M. Rosenthal, N. Morin[†], K.-C. Yeh, J. Cappos, Y. Zhuang, It's the psychology stupid: How heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots, in: Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC'14), ACM, New Orleans, Louisiana, USA, 2014, pp. 296–305.
- [6] B. Chen, Z. M. J. Jiang, Studying the use of java logging utilities in the wild, in: Proceedings of the 42nd International Conference on Software Engineering (ICSE '20), ACM, Seoul, Republic of Korea, 2020, pp. 1–12.
- [7] C. Zhi, J. Yin, S. Deng, Y. Maoxin, F. Min, T. Xie, An exploratory study of logging configuration practice in java, in: Proceedings of the 35th IEEE International Conference on Software Maintenance and Evolution (ICSME'19), IEEE, Cleveland, Ohio, USA, 2019, pp. 1–11.
- [8] Z. Li, T.-H. P. Chen, W. Shang, Where shall we log? studying and suggesting logging locations in code blocks, in: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), ACM, Virtual Event, Australia, 2020, pp. 1–12.
- [9] K. Yao, G. B. de Pádua, W. Shang, S. Sporea, A. Toma, S. Sajedi, Log4perf: Suggesting logging locations for web-based systems' performance monitoring, in: Proceedings of the 9th ACM/SPEC International Conference on Performance Engineering (ICPE 2018), ACM, Berlin, Germany, 2018, pp. 127–138.
- [10] M. Ohmori, A design of a logging system for a computer security incident response, IPSJ SIG Technical Reports 46 (2019) 1–6.

- [11] F. Rivera-Ortiz, L. Pasquale, Towards automated logging for forensic-ready software systems, in: Proceedings of the 6th International Workshop on Evolving Security and Privacy Requirements Engineering (ESPRE'19), IEEE, Jeju Island, South Korea, 2019, pp. 1–7.
- [12] L. Pasquale, D. Alrajeh, C. Peersman, T. Tun, B. Nuseibeh, A. Rashid, Towards forensic-ready software systems, in: In Proceedings of the 40th International Conference on Software Engineering (ICSE'18), ACM, Gothenburg, Sweden, 2018, pp. 1–4.
- [13] L. Daubner, M. Macak, B. Buhnova, T. Pitner, Towards verifiable evidence generation in forensic-ready systems, in: Proceedings of the 2020 International Conference on Big Data (Big Data'20), IEEE, Virtual Event, USA, 2020, pp. 2264–2269.
- [14] Y. Yu, D. Barthaud, B. A. Price, A. K. Bandara, A. Zisman, B. Nuseibeh, Livebox: A self-adaptive forensic-ready service for drones, *IEEE Access* 7 (2019) 148401–148412.
- [15] D. Alrajeh, L. Pasquale, B. Nuseibeh, On evidence preservation requirements for forensic-ready systems, in: Proceeding of 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'17), ACM, Paderborn, Germany, 2017, pp. 559–569.
- [16] A. Hewko, Stride threat modelling: What you need to know, 2022. URL: <https://www.softwaresecured.com/stride-threat-modeling/>.
- [17] FBI, Medical equipment packaging company hacker sentenced, 2021. URL: <https://www.fbi.gov/news/stories/hacker-who-disrupted-ppe-shipments-sentenced-010621>.
- [18] G. McGraw, Software security, *IEEE Security & Privacy* 2 (2004) 80–83.
- [19] B. Arkin, S. Stender, G. McGraw, Software penetration testing, *IEEE Security & Privacy* 3 (2005) 84–87.
- [20] N. C. S. Centre, Penetration testing, 2017. URL: <https://www.ncsc.gov.uk/guidance/penetration-testing>.
- [21] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, Aspect-oriented programming, in: Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97), ACM, Jyväskylä, Finland, 1997, pp. 1–25.
- [22] OWASP, Owasp top 10-2021, 2021. URL: <https://owasp.org/Top10/#welcome-to-the-owasp-top-10-2021>.
- [23] J. King, R. Pandita, L. Williams, Enabling forensics by proposing heuristics to identify mandatory log events, in: Proceedings of the 2015 Symposium and Bootcamp on the Science of Security (HotSoS'15), ACM, Urbana, Illinois, USA, 2015, pp. 1–10.
- [24] A. Chuvakin, G. Petterson, How to do application logging right, *IEEE Security and Privacy* 8 (2010) 82–85.
- [25] J. Bøegh (????).
- [26] L. C. Briand, Y. Labiche, J. Leduc, Toward the reverse engineering of uml sequence diagrams for distributed java software, *IEEE Transactions on Software Engineering* 32 (2006) 642–663.
- [27] Y. Labiche, B. Kolbah, H. Mehrfard, Combining static and dynamic analyses to reverse-engineer scenario diagrams, in: 2013 IEEE International Conference on Software Maintenance, IEEE, 2013, pp. 130–139.
- [28] R. Feldt, A. Magazinius, Validity threats in empirical software engineering research - an initial survey, in: Proceedings of the 22nd International Conference on Software

Engineering and Knowledge Engineering (SEKE'10), ACM, Redwood City, San Francisco Bay, USA, 2010, pp. 1–6.

- [29] C. Williams, M.-A. Storey, N. A. Ernst, A. Zagalsky, E. Kalliamvakou, Methodology matters: How we study socio-technical aspects in software engineering, *ACM Transactions on Software Engineering and Methodology* 37 (2019) 1–22.