Towards Automated Logging for Forensic-Ready Software Systems

Fanny Rivera-Ortiz
School of Computer Science
University College Dublin & Lero
Dublin, Ireland
Email: fanny.riveraortiz@ucdconnect.ie

Liliana Pasquale
School of Computer Science
University College Dublin & Lero
Dublin, Ireland
Email: liliana.pasquale@ucd.ie

***Abstract*—Security incidents can arise from the misuse of existing software systems. Thus, appropriate logging mechanisms should be implemented at the software level to support the detection and investigation of security incidents. However, due to insufficient logging, security incidents often go undetected for long periods. Moreover, even after a security incident is detected, there is not enough information to fully reconstruct how an incident occurred. Insufficient logging may be due to the limited security expertise of software developers, who may not know what are the most critical security incidents. Also, for large software systems and a multitude of potential misuse scenarios, it is cumbersome to identify when and what logging instructions should be implemented. In this paper, we propose a preliminary idea to automate the development of "*forensic-ready*" software systems. These systems can log a minimum amount of relevant data that can be used to detect and investigate potential security incidents. Our approach allows a security engineer to elicit a set of potential software misuse scenarios, expressed as annotated sequence diagrams. These diagrams are then used—together with a control flow graph of the software system— to identify the exact location where logging instructions should be placed and the information they should log. Finally, logging instructions can be injected into designated software system locations using Aspect-Oriented Programming. We illustrate our approach using an example of software misuse in a human resources management software system.**

***Index Terms*—Forensic Readiness; Forensic-Ready Software Systems; Logging; Logging in Software Systems; Digital Forensics**

I. Introduction

Security incidents are increasing and threats are becoming more diverse [1]. For example, in December 2018, Symantec was blocking daily more than 1.3 million unique web attacks on devices connected to the Internet [2]. For security incidents, we refer to malicious activities that can violate either an organization policy or a law [3]. Logs can store relevant information such as, username, date, timestamp, description of an event, which can shed light on how a security incident happened [4]. Although researchers have proposed the use of logs to detect and investigate security incidents [5], [6], [7], [8], [9], security breaches often go undetected for long periods, and/or it is not always possible to fully reconstruct how an incident occurred.

The Open Web Application Security Project (OWASP) has identified insufficient logging as one of the most critical vulnerabilities for web applications [10]. For example, the security breach targeting the Marriot Customer Reservation Software System in November 2018 [11], was undetected for four years [12] due to insufficient logging. Similarly, the digital investigators that were tasked to reconstruct the 2013 Yahoo data breach [13] —the largest in history— were not able to assess the real objective of the breach and identify the offenders. Insufficient logging may be due to the limited security expertise of software developers, who may not know what are the most relevant security incidents, which could occur. Also, for large software systems and a multitude of potential security incidents, it is cumbersome to identify when and what logging instructions should be implemented.

The notion of *forensic-readiness* has been proposed [14], [15] to highlight the importance for organizations to preserve digital data proactively (e.g., through logging), which may facilitate detection and investigation of potential security incidents. Forensic-readiness can maximise the potential to use digital evidence, when required, whilst minimizing the costs of an investigation. Although several approaches have been proposed to support forensic-readiness [14], [16], [17], [18], they mainly provide general guidelines at the organization level [19]. They do not indicate how logging should be implemented in existing software systems to collect information about the relevant, potential security incidents.

In this paper, we propose a preliminary idea to automate the development of "*forensic-ready*" software systems. These systems can log a minimum amount of relevant data that can be used to detect and investigate potential security incidents. Our underlying assumption is that security incidents may arise from the misuse of software (e.g., it may allow the illegitimate modification of sensitive records). Our approach requires a security engineer —who has expertise about potential security incidents— to elicit a set of potential software misuse scenarios, expressed as UML sequence diagrams. These are reverse engineered by replaying the incident on the software system. The UML sequence diagrams are then annotated with information about the software system state that is relevant to the misuse represented.

The annotated UML sequence diagrams are then used in combination with a control flow graph of the software system, to identify the exact location where logging instructions

should be placed and the information they should log. Finally, we can instrument the software system to inject appropriate logging instructions to designated software systems locations, using Aspect-Oriented Programming (AOP). Automated generation of logging instructions removes from the software developers the burden of making decisions about where to log and what to log, to detect a set of security incidents. Note that, while we aim to collect the minimal and relevant data, currently, our work does not consider privacy concerns arising from upcoming data protection regulations, such as the GDPR. We illustrate our approach using a running example that revolves around software for human resources management, which allows the illegitimate approval of travel requests.

This paper is organized as follows. In Section II we provide some background about logging and review related research about logging in software engineering. In Section III we motivate our approach using a running example. In Section IV we explain our approach for supporting automated logging for forensic-ready software systems. Finally, in Section V we conclude the paper and discuss future research directions.

II. Background and Related Work

A log is a collection of events. An event indicates an occurrence of a state change in an environment [7]. For example, an event in a web server log can indicate whether or not a document was uploaded successfully. Logs can be generated by different sources such as operating systems, networks, digital devices, software systems [7]. Logs can have different purposes, such as anomaly detection, problem diagnosis, program verification, usage analysis, security monitoring [20]. For our approach, we are interested in using logs for security purposes to detect potential incidents determined by the misuse of software systems.

To implement logging statements in software systems developers can use off-the-shelf libraries, such as Log4j [21] for Java-based software systems. However, they still need to rely on their knowledge and expertise [8] to decide: a) where to log, i.e. the locations where logging statements should be placed, and b) what to log, i.e. the information to be recorded in the logging statements [20]. However, often developers do not have expertise on how to log for security purposes, because they may not be aware of how the software can be misused and what information should be collected to detect potential software misuses.

Researchers have proposed different approaches to give suggestions to developers about where to log and what to log, either for security and also other purposes. For example, Fu et al. [22] performed an empirical study using two large software systems at Microsoft to analyse the logging practices of developers in the industry, focusing on where developers log. In their study, they identify various categories of log statements, half of them are used to record unexpected situations (e.g., exceptions or function return errors), and the other half records normal execution information at critical execution points. Also, Fu et al. found that developers decide to add logging statements depending on exception types and context information. These findings were validated through a questionnaire survey performed with 54 experienced developers at Microsoft.

Zhu et al. [20] performed an empirical study with two large software systems at Microsoft and two open-source systems from GitHub. The purpose of the study is to develop a logging tool, LogAdvisor, which could learn automatically the common logging practice (e.g., where to log), from existing logging instances in the four studied software systems. These two approaches [22][20] guide developers about where logging should be performed, focusing on detection of runtime errors in software written in C#. However, the logging practices suggested in this work are not aimed to detect security incidents caused by the misuse of software systems.

Other researchers have proposed approaches to protect log files from tampering. Ma and Tsudik [23] propose a novel cryptographic technique to ensure the integrity of audit logs generated and stored on untrusted machines. Sinha et al. [24] propose an infrastructure for secure logging that is capable of detecting the tampering of logs by powerful adversaries residing on the device where logs are generated. However, this work focuses on protecting the integrity of logs but it does not tackle the creation of logs necessary to detect security incidents in software systems. Pinto Leite [5] assesses whether it is possible to detect operating system intrusions from the analysis of existing log files performed using search tools. However, this approach relies on existing log files in Unix Operating Systems and does not provide any suggestion to developers about how to generate new logs inside software systems for security purposes.

King et al. [6] assess whether general audit guidelines for electronic health record mechanisms adequately address non-repudiation. The authors found that only a small percentage of event types that relevant for non-repudiation are recorded. As a result, actions including the modification of patient demographics and assignment of user privileges can be executed without a trace of the user acting. Later, King and Williams [4] perform an exploratory study to identify the current state of logging practices in Health Care Systems. The authors base their study on an existing logging guidelines catalog [25] and existing 2014 Edition Approved Test Procedures for Health Information Technology in the United States. The authors define black-box test cases representing user actions that should be logged, and define specific expected log output based on the identified user actions. The authors observe that after running the test cases only a small percentage of expected log outputs are produced. King et al. [9] also performed an empirical study with graduate-level computer science students to evaluate whether their heuristics-driven method improves a software engineer's ability to identify mandatory log events in open-source systems as compared with using existing industry standards. Although this work [6], [4], [9] is relevant to assess the maturity of logging practices in existing health software systems, it does not provide explicit guidance about how logging statements should be implemented. Moreover, incidents and their criticality can vary depending on the

internal policies of the organization [7] and its assets. Also, a different software system can be misused differently by offenders. Thus, providing generic heuristics to guide logging practices may be ineffective. Instead, the location and the information to be recorded inside logging statements should depend on the internal policies of the organization and the software system itself.

Tan [15] suggests the notion of forensic readiness as the capability of an organization to collect digital evidence from different logging sources, to be prepared to investigate a security incident. However, this approach suggests collecting digital evidence from existing logs that are found in operating systems, networks, and intrusion detection systems, without considering logs generated by software systems. Pasquale et al. [26] tailor the concept of forensic readiness to software engineering as a property that encapsulates the capabilities of the software to conduct digital forensic processes that maximize digital evidence [27] and produces evidence that satisfies the legal scrutiny in a court of law [28]. Alrajeh et al. [19] define a framework for evidence preservation requirements of forensic-ready systems. This framework ensures that only the minimum amount of data that is relevant to detect a security incident is collected. This work [19], [26] represents a first step towards implementing forensic-ready software systems. However, more effort is necessary to guide the implementation of logging statements inside software systems, to support the detection and investigation of software misuses.

III. Motivating Example

In this section, we present a running example that we use to motivate our approach.

Let us consider a Human Resources Management (HRM) software system which has the following modules: 1) Employees: it allows adding, modifying, deleting and viewing employees; 2) Users: it allows adding, modifying, deleting and viewing users of the HRM software system and their credentials; 3) Travel Requests: it allows creating, modifying, approving, deleting and viewing travel requests. Furthermore, the HRM software system has the following user levels or roles: a) Admin: can have access to all of the modules of the HRM software system; b) Manager: can only have access to modules Employees and Travel Requests; c) Employee: can only have access to module Travel Requests, and cannot approve travel requests.

Our incident example is caused by an insider, Henry Smith, who is the System Administrator of the HRM software system. Henry is going on a business trip and creates a travel request using the HRM software system, while his line manager, Arthur Jones, is away on holiday. Henry uses his privileges as a System Administrator to modify Arthur's credentials and impersonate Arthur, in order to approve his own travel request.

Fig. 1 describes the sequence of actions characterizing this security incident.

1) Henry connects to the HRM software system with the user level or role Admin.
2) Henry accesses the Travel Request module and creates a new Travel Request.
3) The Travel Request is created and sent to Arthur. However, Arthur is on holidays and cannot approve of it.
4) Henry needs approval for his travel request and connects as Admin to the HRM software system.
5) Henry goes to the Users module and modifies Arthur's password and exits the HRM software system.
6) Later, Henry misuses his privileges as a System Administrator because he impersonates Arthur, by connecting to the HRM software system. The HRM software system displays a message that: "A Manager has connected into the HRM software system."
7) Finally, Henry accesses the Travel Request Module, reviews the travel request and approves it. The HRM Software Systems displays a message: "The travel request has been approved by Arthur Jones."

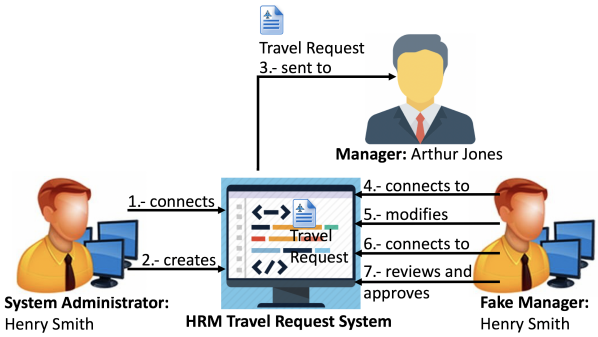

Fig. 1. Incident example representing a misuse of the HRM software system.

This scenario represents a security incident because it violates one of the organization's policies, i.e. "only a manager can authorize a travel request". If adequate logging is not performed, this security incident can go undetected or the investigation about this incident may be difficult because relevant evidence is missing to indicate the offenders' identity and the incident actions. Note that it is not possible to prevent Henry from modifying users' credentials, because he requires this access right to perform his duties as a System Administrator. Thus, it is necessary to augment the HRM software system with the functionality necessary to ensure accountability, which allows detecting software misuses and identifying victims and offenders [29]. In other words, it is necessary to implement logging functionalities that could record the user actions that lead to potential software misuses.

IV. Logging to Develop Forensic-Ready Software Systems

Fig. 2 provides a general overview of our approach, which has three stages:

1) Incident Modelling: A *security engineer*, who is the one in the organization that has security expertise, runs a *software system* usage scenario that simulates the incident, to generate an *Incident Model*. This Incident Model is a UML sequence diagram that represents the steps involved

in the incident, without including the entire software behaviour. During this stage, our approach allows the *security engineer* to annotate the UML sequence diagram including relevant information for logging, such as where to log and what to log. The output of this stage is the enhanced *Incident Model* with the annotations provided by the security engineer.

2) Logging Instrumentation: The annotations produced by the *security engineer* are used together with a *control flow diagram* of the software system to determine the exact location of logging statements and specific data that these statements should record. This is important to ensure that logging statements provide relevant information to detect and investigate the *incident*. The output of this stage is a *file* that specifies where to log and what to log in the *software system* to detect the *incident*.
3) Logging Generation: Once the location and the information to be stored in the log statements is determined, our approach instruments the *software system* to inject the logging statements to the designated software system locations. This removes from the software developer the burden of deciding where and what to log to detect security incidents. When the software reaches the locations where logging statements are placed, a log will be generated. Thus, the software will produce a *log file* that can be analyzed to detect and investigate security incidents.

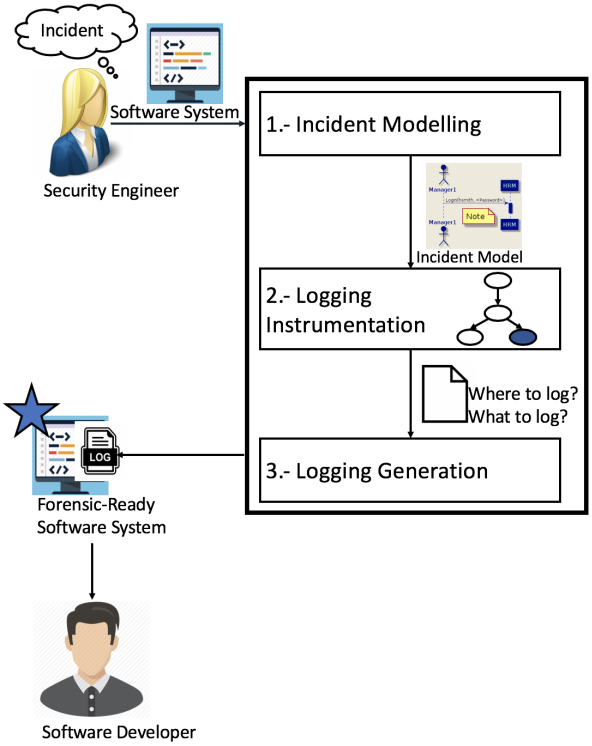

Fig. 2. General Overview of our Approach

Now, we explain in detail the three stages of our approach. The first stage, *Incident Modelling* (Fig. 2), allows a security engineer, to run several *software* usage scenarios that simulate the incidents of interest. For each incident, our approach generates automatically an *Incident Model*, which is a UML sequence diagram that only representing the sequence of method invocations associated with the steps of the incident. For our incident example described in (Fig. 1), a security engineer, has to replay the steps of the incident on the HRM software system.

To generate this initial UML sequence diagram, we took inspiration from two approaches. Briand et al. [30] suggest to reverse-engineer a Java Application. They create a UML sequence diagram by instrumenting the application using AspectJ, which is the implementation of Aspect-Oriented Programming for Java [31], [30]. Labiche et al. [32] improve the approach presented by Briand et al. [30] by dramatically reducing the information collected at runtime. The instrumented Java Application only collects the necessary information about Java methods such as caller and callee objects, method signature, class name and line number of the call. Similarly to Labiche et al. [32], we instrument a group of selected classes of the software system using AspectJ, to capture the execution traces associated with the steps of the incident. We collect the necessary runtime information about Java methods such as caller, callee objects, method signature, class name.

Once the initial UML sequence diagram is generated, our approach allows the security engineer to annotate the UML sequence diagram, to include relevant conditions that determine when logging is necessary and/or additional information that should be logged. These annotations are related to entities that exist in our software systems, such as Java Classes or database tables. For our incident example, (Fig. 3) we will focus on step 3, which is highlighted in the rectangle. Here, we can observe that the security engineer adds the following notes. The first note (A) indicates that it is important to log who connects to the software system because this allows identifying the offender. In our incident example, we need to record that Henry is the employee who connects to the HRM software system using Arthur's credentials. The second note (B), mentions that is important to log whenever a user with the user level or role Manager connects to the HRM software system. This means that we are not interested in login operations to perform by employees with user levels of employee or admin. The third note (C) indicates that it is important to log whenever a travel request changes to status Approved because at this moment a security incident can occur. The output of this stage is a final UML sequence diagram that contains the relevant annotations from the security engineer about where to log and what to log to detect the security incident.

The second stage, *Logging Instrumentation* (Fig. 2), receives as an input the final UML sequence diagram. Then, our approach performs a control flow analysis of the incident, using Soot [33], which is a Java product created by the Sable research group from McGill University. Soot is a Java library that can be included in a Java Application to create a control flow diagram (CFG). In a CFD, Soot defines the entry and

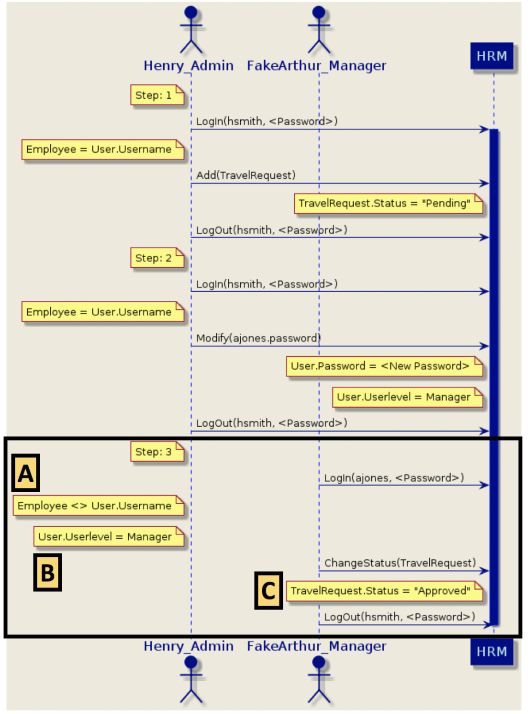

Fig. 3. Stage 1: Final UML Sequence Diagram.

exit methods at every point in the graph [33]. While we are creating the control flow diagram, we consider the annotations from the security engineer to determine the exact location and the specific information required in the software system to add logging mechanisms to detect the incident. Additionally, while determining where to log we consider the Java classes inside the software system that are relevant such as the ones that have access to the database and not the ones that implement the graphical interface in the software system.

For our incident example, we illustrate the generation of the control flow diagram for step 3 (Fig. 3), which is where the incident occurs. Fig. 4, shows the control flow diagram where the methods and the fields that are executed inside the HRM software system for step 3, which is where the incident happens. The LogIn, is the step indicated in the Final UML sequence diagram where the user connects to the HRM software system and it includes two methods inside the HRM software system: getUserName (A) and getUserLevel (B). The ChangeStatus is the other step described in the final UML sequence diagram which corresponds when the manager approves the travel request and it executes inside the HRM software system just one method: save (C). Additionally, to determine where to log and what to log, we consider the requirements to make forensic-ready systems defined by Pasquale et al. [26]: a) Relevance: Data preserved proactively should be relevant to potential incident cases and able to support or refute hypotheses explaining how incidents occurred; b) Minimality: Data preserved proactively should be minimal and should not include any information that is unnecessary for an investigation.

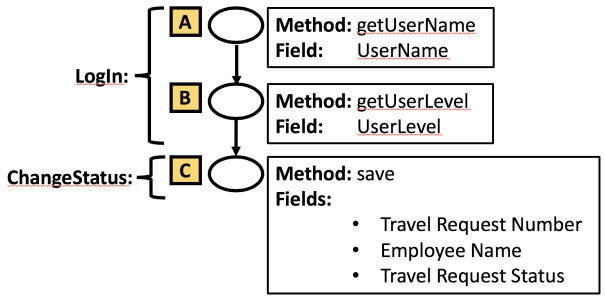

Fig. 4. Stage 2: Generating a Control Flow Diagram.

For our incident example, considering relevance and minimality to make our HRM software system forensic-ready, we determined that the convenient location in HRM software system (Fig. 5) to place logging statements to detect our incident is the method save (C), because in this method inside the software system, the following information can be available to help us to detect our incident: UserName (A), UserLevel (B), Travel Request Number, Employee Name and Travel Request Status (C).

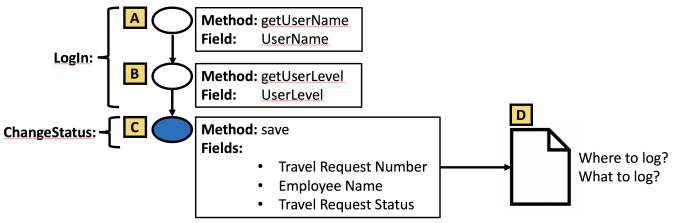

Fig. 5. Stage 2: Logging Instrumentation

While we are meeting the requirements of relevance and minimality, currently our approach does not consider privacy concerns such as GDPR.

The third stage, *Logging generation*, (Fig. 2) receives as an input the file which contains information on the software system related to where to log and what to log to detect the incident. Our approach instruments our software system just in the places where the incident is involved using Aspect J to generate logs that detect the incident. We also use Log4j, which is the Java logging framework to create logging statements that generate log files. Log4j is a highly configurable framework: It assigns levels of priorities to logs such as ALL, TRACE, DEBUG, INFO, WARN, ERROR and FATAL; it offers mechanisms to direct logging information to different destinations like a database, file, console, UNIX Syslog [21] and it allows that different types of output layouts in the logs such as XML, HTML, text [34].

For our incident example, we receive the file (Fig. 5) (D) that contains where to log and what to log. In this step, our approach reads this file and creates logging statements (Fig. 6) (A). Once, logging statements has been injected to our HRM software system, the security engineer must run again the HRM *software system* usage scenario that simulates

the incident and our approach will generate automatically a log file (B) that detects the incident and says: "The travel request has been approved." This log displays relevant fields that provide useful information to investigate this incident such as travel request number, the employee who requested the travel request *employee*; who approved the travel request *approved by*; from which computer the manager connected *IP Address*.

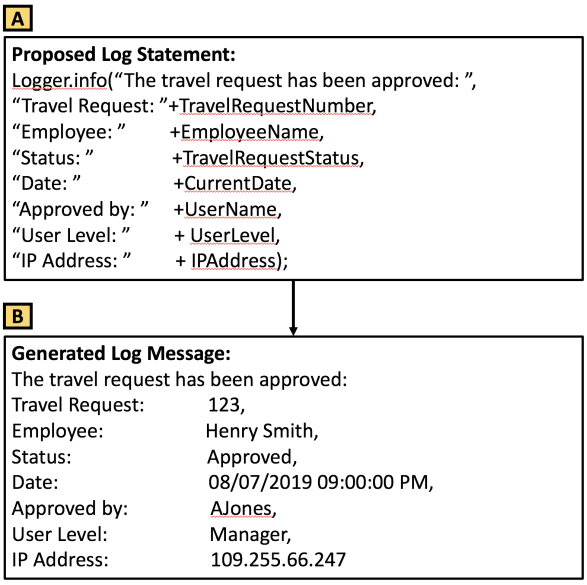

Fig. 6. Stage 3: Logging Generation.

During the logging instrumentation stage, our aim is to provide a practical way to make our software system to comply with one of the security principles called: Information Accountability defined as,*"the use of information should be transparent so it is possible to determine whether a particular use is appropriate under a given set of rules and that the system enables individuals and institutions to be held accountable for misuse."* [35] Our approach, provides transparency and accountability by generating a log file that detects a security incident. This log file makes bad actions visible and who is responsible to perform these actions.

Proposed Evaluation. To evaluate our work we will follow an approach similar to the one proposed by King and Williams [4]. They tested four popular health care systems to assess whether sufficient logging was performed, to record information about some of the most relevant software misuses. Similarly, we aim to replay a set of relevant security incidents on the software system instrumented with logging instructions. In other words, we will replay the message sequences expressed in the sequence diagrams elicited during incident modelling. Our objective is to assess *relevance* and *minimality* of logs. To assess relevance we will verify that the logs generated indicate the occurrence of the message exchanges described in the sequence diagrams. We will also ensure that logging is performed only when the logical conditions indicated in the annotated sequence diagram are satisfied. To assess minimality we will verify that no logging is generated to indicate execution of operations that are not covered by the sequence diagrams. We will also ensure that logs do not record message exchanges that occur when the corresponding logical condition in the annotated sequence diagram is not satisfied. Finally, we will verify the possible downgrade in performance introduced by logging operations. In particular, we will compare the time to execute the security incident scenarios in the software system instrumented with logging with that necessary to replay the same scenario in the original software system.

V. Conclusion and Future Work

In this paper, we propose a novel approach to automate the development of forensic-ready 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.

For future work, we aim to develop a tool implementing the three stages that our approach (Fig. 2). We envision the first two stages of our approach (Incident Modelling and Logging Instrumentation) to be the most challenging. For Incident Modelling, we will instrument a designated set of relevant software system components with the capability to detect methods execution. When the security incident scenarios are simulated on the running software, those instructions will represent the method execution as a message exchange in the sequence diagram. To annotate the sequence diagram, it will be necessary to define a set of entities, which characterize the software system state, and on which the logical constraints can be expressed. To achieve this aim, we will take inspiration from existing work on feature identification and reverse engineering of statecharts. To support Logging Instrumentation, it will be necessary to map each message exchange in the sequence diagram to a subset of the control flow graph; this will allow locating the position where the logging instruction will be injected. In this stage, it is also necessary to consider privacy concerns such as GDPR while collecting the relevant and minimal data to detect a security incident. We will also consider strategies to reduce the amount of logging performed when security incidents have overlapping message exchanges in their scenarios.

Finally, for Logging Generation, we will add an engine that whenever log instructions have been injected and the software system is forensic-ready, an alert will be generated that says we should run again our software system and a log message detecting the incident will be displayed. Additionally, this engine will perform a deeper analysis of the log message generated and could also encrypt the log message.

Acknowledgment

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

References

[1] Symantec, "ISTR Internet Security Threat Report Volume 23. February 2018," p. 89, 2018. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf

[2] ——, "ISTR Internet Security Threat Report Volume 24. February 2019," p. 61, 2019. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf

[3] B. Carrier and E. Spafford, "An Event-Based Digital Forensic Investigation Framework," *Digit. Forensic Res. Work.*, p. 13, 2004.

[4] J. King and L. Williams, "Log Your CRUD: Design Principles for Software Logging Mechanisms," *Proc. 2014 Symp. Bootcamp Sci. Secur. (HotSoS'14).*, p. 10, 2014.

[5] J. Pinto-Leite, "Analysis of log files as a security aid," *Procedings 6th Iber. Conf. Inf. Syst. Technol. (CISTI'11).*, p. 6, 2011.

[6] J. King, B. Smith, and L. Williams, "Modifying without a trace: general audit guidelines are inadequate for open-source electronic health record audit mechanisms," *Proc. 2nd ACM SIGHIT Int. Heal. Informatics Symp.*, p. 9, 2012.

[7] A. A. Chuvakin, K. J. Schmidt, and C. Phillips, *Logging and Log Management. The Authorative Guide to Understanding the Concepts Surrounding Logging and Log Management.*, 2013th ed. Syngress, 2013.

[8] G. Rong, Q. Zhang, X. Liu, and S. Gu, "A Systematic Review of Logging Practice in Software Engineering," *Proc. Asia-Pacific Softw. Eng. Conf. (APSEC'17).*, p. 6, 2017.

[9] J. King, J. Stallings, M. Riaz, and L. Williams, "To Log, or Not to Log: Using Heuristics to Identify Mandatory Log Events–A Controlled Experiment," *Empirical Software Engineering*, vol. 22, no. 5, pp. 2684–2717, 2017.

[10] OWASP, "OWASP Top 10 - 2017," 2017. [Online]. Available: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

[11] J. Valinsky, "Marriott reveals data breach of 500 million Starwood guests," p. 2, 2018. [Online]. Available: https://www.cnn.com/2018/11/30/tech/marriott-hotels-hacked/index.html

[12] Reuters, "Over 500m accounts hit in Marriott data breach," p. 2, 2018. [Online]. Available: https://www.rte.ie/news/business/2018/1130/1014341-marriott-data-breach/

[13] A. Taylor, "The 16 biggest data breaches of the 21st century," p. 11, 2018. [Online]. Available: https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html

[14] R. Rowlingson, "A Ten Step Process for Forensic Readiness," *Int. J. Digit. Evid.*, vol. 2, no. 3, p. 28, 2004.

[15] J. Tan, "Forensic Readiness," p. 23, 2001.

[16] N. Beebe, "Digital Forensic Research: The Good, the Bad and the Unaddressed," p. 15, 2009. [Online]. Available: http://link.springer.com/10.1007/978-3-642-04155-6_2

[17] A. Pooe and L. Labuschagne, "A conceptual model for digital forensic readiness," *Procedings 11th Work. Inf. Secur. South Africa (ISSA'12).*, p. 8, 2012. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6320452

[18] M. Elyas, S. B. Maynard, A. Ahmad, and A. Lonie, "Towards A Systemic Framework for Digital Forensic Readiness," *J. Comput. Inf. Syst.*, vol. 54, no. 3, p. 21, 2014.

[19] D. Alrajeh, L. Pasquale, and B. Nuseibeh, "On Evidence Preservation Requirements for Forensic-Ready Systems," *Proc. ACM SIGSOFT Symp. Found. Softw. Eng.*, p. 10, 2017.

[20] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," *Proc. 37th IEEE/ACM Int. Conf. Softw. Eng. (ICSE'15).*, vol. 1, p. 10, 2015.

[21] Tutorial Points, "log4j Overview," 2019. [Online]. Available: https://www.tutorialspoint.com/log4j/log4j_overview.htm

[22] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," *Procedings 36th Int. Conf. Softw. Eng. (ICSE'14).*, p. 10, 2014.

[23] D. Ma and G. Tsudik, "A New Approach to Secure Logging," *ACM Transactions on Storage (TOS)*, vol. 5, no. 1, p. 2, 2009.

[24] A. Sinha, L. Jia, P. England, and J. R. Lorch, "Continuous tamper-proof logging using TPM 2.0," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8564 LNCS, p. 18, 2014.

[25] J. King and L. Williams, "Cataloging and Comparing Logging Mechanism Specifications for Electronic Health Record Systems," in *Presented as part of the 2013 {USENIX} Workshop on Health Information Technologies*, 2013.

[26] L. Pasquale, D. Alrajeh, C. Peersman, T. Tun, B. Nuseibeh, and A. Rashid, "Towards Forensic-Ready Software Systems," p. 4, 2018.

[27] R. Mckemmish, "When is digital evidence forensically sound?" *IFIP Int. Fed. Inf. Process.*, vol. 285, p. 13, 2008.

[28] B. Endicott-Popovsky, N. Kuntze, and C. Rudolph, "Forensic readiness: Emerging discipline for creating reliable and secure digital evidence," *J. Harbin Inst. Technol. (New Ser.*, vol. 22, no. 1, p. 9, 2015.

[29] W. Benghabrit, H. Grall, J. C. Royer, and M. Sellami, "Accountability for abstract component design," *Proc. to 40th Euromicro Conf. Ser. Softw. Eng. Adv. Appl. (SEAA'14).*, p. 9, 2014.

[30] L. C. Briand, Y. Labiche, and J. Leduc, "Toward the reverse engineering of UML sequence diagrams for distributed Java software," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 642–663, 2006.

[31] Eclipse, "Introduction to AspectJ," *Asp. Program. Guid.*, 2003. [Online]. Available: https://www.eclipse.org/aspectj/doc/next/progguide/starting.html

[32] Y. Labiche, B. Kolbah, and H. Mehrfard, "Combining Static and Dynamic Analyses to Reverse- Engineer Scenario Diagrams," *IEEE Int. Conf. Softw. Maintenance, ICSM*, p. 10, 2013.

[33] A. Einarsson and J. D. Nielsen, "A survivor's guide to Java program analysis with soot," *BRICS, Dep. Comput. Sci. Univ. Aarhus, Denmark*, p. 46, 2008.

[34] A. Tomar, "Logging With Log4j in Java," 2018. [Online]. Available: https://dzone.com/articles/logging-with-log4j-in-java

[35] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. Jay Sussman, "Information Accountability," *Commun. ACM*, vol. 51, no. 6, p. 6, 2008.