JPrivacy: A Java Privacy Profiling Framework for Big Data Applications

Mohamed Abdellatif
Department of Computer Science
University of Miami
Coral Gables, USA
m.abdellatif@miami.edu

Iman Saleh
Department of Computer Science
University of Miami
Coral Gables, USA
iman@miami.edu

M. Brian Blake
Department of Computer Science
University of Miami
Coral Gables, USA
m.brian.blake@miami.edu

***Abstract*—Businesses and government agencies are continuously generating and collecting huge amounts of data and building related Big Data applications. Big Data applications involve the collaborative integration of APIs from different providers. A challenge in this domain is to guarantee the conformance of the integration to privacy terms and regulations. In this paper, we present JPrivacy, a privacy profiling framework for Big Data applications. JPrivacy proposes a model for privacy rules and provide the algorithms and related tools to check Java code against these rules. We show through experimentation that JPrivacy can effectively detect privacy violations by statically analyzing a piece of code.**

Keywords: Privacy, Big Data, Java, Static Analysis

I. INTRODUCTION

Given the inexpensive nature and availability of information storage media, individuals worldwide have exponentially increased their production and persistence of large amounts of data whether such data are captured as text, images, or sound. Analysis of these Big Data repositories introduces fascinating new opportunities for discovering new insights that contribute to different branches of science. The potential of Big Data comes however with a price; the users' privacy is often at risk. Guarantees of conformance to privacy terms and regulations are limited in current Big Data analytics and mining practices. Unlike relational databases that exhibit a clear structure, Big Data is characterized by its unstructured nature and the variety of data types including both textual and audio-visual material. Only the Big Data applications encapsulate the logic that makes sense of such unstructured repositories. Hence, our work comes to provide tools and frameworks to build trusted Big Data applications. Using our framework, Big Data developers are able to verify that their code complies with privacy agreements and that sensitive users' information is kept private regardless of changes in the applications and/or privacy regulations. Our work investigates the following research questions:

- RQ1: How to formally specify *privacy*? Can we devise machine-readable privacy rules?
- RQ2: How to extract privacy rules from natural language descriptions and formalize regulations such as the HIPAA?
- RQ3: How can we leverage the formal definition of privacy to reason about privacy conformance for a piece of code?
- RQ4: How to automatically generate tests from formal specification of privacy?

In this paper, we address RQ1 and RQ3. We present JPrivacy; a privacy profiling system for Java code. JPrivacy is based on a formal model for privacy rules and provide the algorithms and related tools to check Java code against these rules. Figure 1 shows the JPrivacy framework and its main components. JPrivacy takes as input a Java application and a natural-language description of privacy terms. It formalizes the privacy terms and checks the application's code for potential violations of these terms. JPrivacy can also leverage these terms in order to generate test cases. These test cases guarantee that an application continues to comply with the privacy regulations as the code and underlying Big Data repositories evolves.

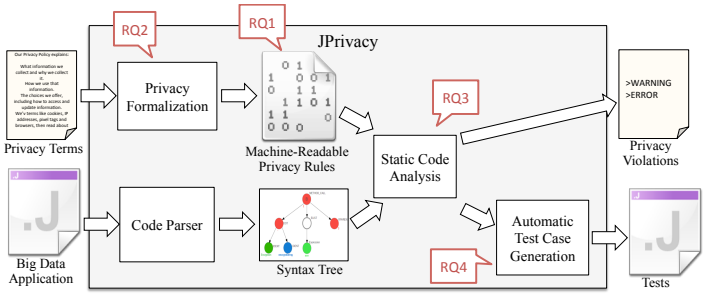

Figure 1. JPrivacy Framework

II. RELATED WORK

Research has recognized the importance of building security and privacy measures into software systems. Most work focuses however on the extraction of requirements from security-related policies and regulations [1][2][3]. Other work extract formal privacy requirements from legal regulations such as HIPPA [4][5]. The notion of privacy is defined in terms of access right to sensitive data. Once the requirements are defined, a traditional software engineering process can take place. As we consider complex collaboratively built Big Data applications, we are building privacy consideration into the software engineering process.

III. PRIVACY MODEL

Our proposed privacy model categorizes data from a privacy standpoint. We identify four categories of data; Critical Sensitivity (CS), High Sensitivity (HS), Moderate Sensitivity (MS), Low Sensitivity (LS) and Non-Sensitive (NS). As the names suggest, these categories associate a level of sensitivity to a piece of data. Next, we define the different operations on data that are relevant to a privacy checker. We identify three categories of operations that can be done on a data field: *reading, writing,* and *sharing*. By writing a data field, we mean having it persists in a file or a database. A data field can be written in encrypted or plaintext formats. On the other hand, sharing a data field is basically sending it as a parameter to third party software. This third party can be an API, a Web Service or simply a code module. The third party can be a trusted or untrusted.

COLLABORATECOM 2014, October 22-25, Miami, United States

DOI 10.4108/icst.collaboratecom.2014.257666

The definition of trust is domain-specific. These data operations are used to define restrictions on manipulating different categories of data, as will be detailed later. Figure 2 summarizes these data operations and Table 1 lists the different data categories and the corresponding operations. A '1' in the table denotes that the operation is allowed while a '0' denotes that the operation is not allowed and is considered a violation of the privacy regulations.

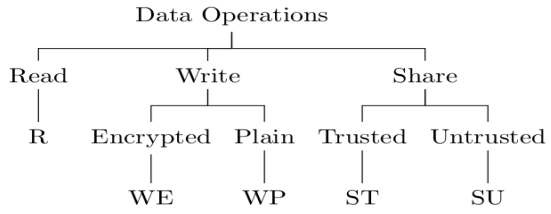

Figure 2. Data Operations

Table 1. Data Fields Categories and Operations

	R	WE	WP	ST	SU
CS	1	0	0	0	0
HS	1	1	0	0	0
MS	1	1	1	0	0
LS	1	1	1	1	0
NS	1	1	1	1	1

IV. PRIVACY STATIC CODE ANALYSIS

JPrivacy static code analysis extracts the Abstract Syntax Tree (AST) from the code. The proposed algorithm then traverses the AST in DFS pre-order traversal. At each node of the AST, the JPrivacy checker inspects the node's contents and checks for fields that are interesting from a privacy perspective. An AST node can denote the start of a *Class*, a *Method-Call*, and *Identifier* or a *Parenthesis*. Figure 3 shows the static analysis algorithm applied on the AST to detect violation. The algorithm is applied iteratively on the visited AST nodes.

Algorithm: Detect Violation

1: $n \leftarrow get\_next\_node()$
2: **if** $(n = import \ \ and \ \ get\_type(n) = untrusted)$ **then**
3: $\quad print\_violation(IMPORT, n)$
4: **else if** $(n = operation\_call \ \ and \ \ get\_type(n) \in interesting\_types)$ **then**
5: $\quad operation \leftarrow n$
6: **else if** $(operation \neq EMPTY$
7: $\quad\quad and \ \ n = data \ \ and \ \ is\_affected(n, operation))$ **then**
8: $\quad print\_violation(get\_type(operation), n)$
9: **else if** $(n = operation\_end)$ **then**
10: $\quad operation \leftarrow EMPTY$
11: **end if**

Figure 3. Privacy Static Code Analysis Algorithm

V. USE CASE

We used a sample Java code, shown in Figure 4, to evaluate the effectiveness of JPrivacy. The code sample writes, shares, and encrypts data items with various sensitivity levels. In this particular case, data items with names having "*ssn*" are considered to be CS, those with name having "*pwd*" as HS, "*lastName*" as MS, and finally "*firstName*" as LS. As shown in figure 5, JPrivacy was able to detect potential violations of privacy rules as defined in the previously illustrated privacy model. Identifying how programmers most commonly name data variables is an interesting complementary problem that we are currently investigating.

```
...
27. Network.sendToServer("ourserverurl.com", "SSN=" + ssn + "&pwd=" + howareyou_pwd);
28. Network.sendToServer("ourserverurl.com", "fName=" + firstName + "&lName=" + lastName);
29.
30. Network.querySocialMedia(Network.FB, "SSN=" + ssn + "&pwd=" + pwd);
31. Network.querySocialMedia(Network.FB, "fName=" + firstName + "&lName=" + lastName);
32.
33. FileUtils.writeStringToFile(new File("test.txt"), "SSN is " + ssn + " and password is " + pwd + "\n");
34. FileUtils.writeStringToFile(new File("test2.txt"), "First name is " + firstName + " and last name is " + lastName +
    "\n");
35.
36. FileUtils.writeStringToFile(new File("test.txt"), "SSN is " + Encryption.encrypt(ssn) + " and password is " +
    Encryption.encrypt(pwd) + "\n");
37. FileUtils.writeStringToFile(new File("test2.txt"), "First name is " + Encryption.encrypt(firstName) + " and last name
    is " + Encryption.encrypt(lastName) + "\n");
...
```

Figure 4. A Sample Java Code.

```
/sample_code/sample2/Example.java:1: Imported WRITE warning!
/sample_code/sample2/Example.java:2: Imported WRITE warning!
/sample_code/sample2/Example.java:27: pwd shared!
/sample_code/sample2/Example.java:27: ssn shared!
/sample_code/sample2/Example.java:28: lastName shared!
/sample_code/sample2/Example.java:30: pwd shared with untrusted API!
/sample_code/sample2/Example.java:30: ssn shared with untrusted API!
/sample_code/sample2/Example.java:31: firstName shared with untrusted API!
/sample_code/sample2/Example.java:31: lastName shared with untrusted API!
/sample_code/sample2/Example.java:33: pwd written in plain!
/sample_code/sample2/Example.java:33: ssn written!
```

Figure 5: A Sample Output Showing Privacy Violations.

VI. CONCLUSION

In this paper, we propose JPrivacy; a privacy static code analysis framework for Java code. JPrivacy is based on modeling privacy rules and statically checking code for compliance. Our experimentation shows that JPrivacy successfully detects and highlights potential privacy violations. We hence provide developers with a valuable framework to check their Big Data applications against privacy terms and regulations. In future work, we plan to automatically formulate privacy rules from natural language descriptions of privacy terms.

REFERENCES

[1] A. Siena, A. Perini, A. Susi, and J. Mylopoulos, "A meta-model for modelling law-compliant requirements," in *Requirements Engineering and Law (RELAW), 2009 Second International Workshop on*, 2009, pp. 45–51.

[2] T. D. Breaux and A. I. Antón, "Analyzing regulatory rules for privacy and security requirements," *Software Engineering, IEEE Transactions on*, vol. 34, no. 1, pp. 5–20, 2008.

[3] P. N. Otto and A. I. Antón, "Addressing legal requirements in requirements engineering," in *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, 2007, pp. 5–14.

[4] T. D. Breaux, M. W. Vail, and A. I. Anton, "Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations," in *Requirements Engineering, 14th IEEE International Conference*, 2006, pp. 49–58.

[5] M. J. May, C. A. Gunter, and I. Lee, "Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies," in *Proceedings of the 19th IEEE Workshop on Computer Security Foundations*, Washington, DC, USA, 2006, pp. 85–97.

[6] "Checkstyle." [Online]. Available: http://checkstyle.sourceforge.net/.