Protecting Customer Data and Ensuring GDPR Compliance with Privacy Static Analysis
Introduction

Protecting customers’ data has become vital for all organizations in order to safeguard their business and their reputation. The maturation of privacy regulations, in particular the recent enforcement of the European General Data Protection Regulation (GDPR), have led to stricter requirements and added substantial fines.

A data breach incident can take a heavy toll on a company. Ponemon Institute’s recent study “Cost of Data Breach Study” estimates the cost of an average data breach at $3.62 million. With peaks such as the Facebook incident (market cap $100 billion, or 20% of its total value), Equifax (costs $439 million) or TalkTalk (costs $60 million and loss of 100,000 customers), it is obvious that protecting sensitive information is a critical factor for the success of any company.

How does GDPR affect your company?

EU GDPR, enforced on May 25 2018, obliges all data processors to pay more attention to how they treat sensitive data of European citizens. In particular, all the organizations processing sensitive data are supposed to know what data they handle, why and how. It also imposes that the controller of sensible data adopts an approach based on data protection by design and by default.

The scope of GDPR is extremely broad, and ranges from very high level organizational to deep technical procedures. A large part of the compliance consists in verifying, designing and setting up policies and organizational processes to properly manage the minimal amount of sensitive data necessary for the services provided by the company. However, there is still a critical open question: how can organizations be sure that the software processing the sensitive data respects the constraints identified during the design of the information system?

Security-only solutions are not enough

Most technology solutions supporting GDPR are security-centered and provide means to protect the IT systems from cyber-attacks. While these efforts are an essential part of the equation, privacy challenges rarely have security-only solutions. An exhaustive knowledge of which data enters an application, how it is handled and where it can end up is fundamental to comply with GDPR requirements. Traditional vulnerability analysis, a key component in preventing data breach, should be integrated with privacy-oriented solutions, which take into account the nature of data (sensitive – not sensitive) depending on the context.

As most software was written before the GDPR, it was not designed taking into account Privacy. In this scenario, companies face a situation where they must conduct a thorough review of their applications to make sure that they do not constitute a risk in this context. Likewise, for all new developments, it is fundamental to ascertain that the privacy principles are successfully implemented from design to deployment.

Ensure the respect of your Privacy Policy on application level

Static analysis has been widely applied to automatically analyze software to identify bugs and defects. As such, it supports the GDPR process by efficiently identifying security vulnerabilities in applications.

Static analysis based on formal methods, such as Abstract Interpretation, permits the
modeling of a comprehensive data flow of a program. Building on this capability, the Julia suite has been extended with an innovative specific analysis to automatically verify whether a software satisfies a given privacy policy, thus permitting a significant reduction of data breach risk and adherence to GDPR compliance requirements.

In this White Paper we describe how static analysis for privacy works, which are its benefits and give a practical example of its application using the OWASP WebGoat 6.0.1 as a use case.

**How Privacy Static Analysis works**

Static analysis has been widely applied to prove various software properties, automatically. Its main idea is to create a mathematical model of the executions of a program and to statically prove (i.e., without executing the code) some properties on such model. A sound static analysis creates a model that covers all possible executions. Therefore, it can prove that all possible executions of the program under the analysis satisfy the given property.

Absence of runtime errors, correct synchronization between parallel threads, absence of security vulnerabilities such as SQL injection and cross-site scripting (XSS) are just some notable examples of properties that can be proven with static analysis. In particular, the latter analyses automatically track the flow of user inputs inside the program, and checks if these inputs might reach a critical API call (like the execution of an SQL query) without being sanitized.

The Julia GDPR checker applies such data flow analyses to detect leakages of sensitive data that are not allowed by a privacy policy defined in the GDPR compliance context. For a detailed description of the underlying technology please refer to (Ferrara & Spoto, Static Analysis for GDPR Compliance, 2018; Ferrara, Olivieri, & Spoto, Tailoring Taint Analysis to GDPR, 2018).

**Configuring the Julia GDPR Checker**

The use of a checker specifically targeting privacy issues is significantly different from the use of the traditional checkers. The user, typically the DPO’s team, configures the checker with instructions on:

- The types of sensitive data and leakage points of the applications,
- How sensitive data might be accessed and leaked programmatically, and
- What flows are allowed by the privacy policy of the information system.

The GDPR checker receives this information through an Excel worksheet composed of 4 distinct sheets (Categories, SensitiveData, LeakagePoints, AllowedFlow). Such worksheet can be produced by running the GDPR checker’s Init phase.

**Categories of Sensitive Data and Leakage Points**

Not all types of sensitive data and leakage points are equal. For instance, name and surname of a person are definitely sensitive data, but social security and credit card numbers are more critical data from a privacy perspective. Therefore, leaking some particular types of sensitive data into a log could be problematic (e.g., if this concerns a credit card number) or not (e.g., if it concerns user’s name or identifier) Hence, the configuration of a GDPR analysis must include a categorization of sensitive data and leakage points.
As a running example, we applied the GDPR checker to the analysis OWASP WebGoat 6.0.1\(^2\). Since OWASP WebGoat "is a deliberately insecure web application maintained by OWASP designed to teach web application security lessons"\(^3\), it represents a good target for security and privacy analyses. In particular, we noticed that such application contains an object-oriented model of employees' data stored in the database, and it communicates over Internet much information. Therefore, we defined the categories of sensitive data and leakage points represented in Figure 1. These are aimed at representing the different kinds of data about employees (like their name and salary) and how such data might be transmitted to Internet or stored in the database.

**Specification of Sensitive Data and Leakage Points**

Once the interesting categories have been established, the GDPR checker needs to know how sensitive data is read and leaked by the statements of the program. If on the one hand such information needs to be manually specified, on the other hand the GDPR compliance process requires one to know how sensitive data could be accessed and leaked by the software, and thus it is already part of the GDPR compliance process.

**Sensitive data**

How software can read (potentially sensitive) data programmatically? This can happen through method calls returning a value, or by reading fields, both in the code of the application (for instance through method calls that access the data given by a user through a graphical interface) and in the code of the libraries. Therefore, the SensitiveData sheet contains the complete list of these components, and the User can annotate them with the category of sensitive data they retrieve. For instance, Figure 2 reports a part of the specification we applied to WebGoat: the field password of class WsSAXInjection (that represents a Web page) contains the user's password, while the various fields of class Employee retrieve different kinds of sensitive data.

**Leakage Points**

How software might pass (potentially sensitive) data...
to components outside the bounds of the main application? In this case, this might happen by writing a field, or passing a value to a parameter of a method call. However, this applies only to components in the libraries, since the application itself can leak data only by calling APIs of external libraries. Therefore, sheet LeakagePoints contains the complete lists of these components. For instance, we specified the analysis on WebGoat by annotating the constructor of a URL and of different HTML elements as Internet leakage points and method setString of PreparedStatement as leaking information to the database, as reported by Figure 3.

**Specification of the Privacy Policy**

The last part of the configuration of a GDPR analysis is the specification of a privacy policy. Hence, the sheet AllowedFlows specifies what categories of sensitive data are allowed to be disclosed to what categories of leakage points.

![Figure 4: Privacy policy specification](image)

Running the GDPR Analysis

With the information specified in the last Section, we are now in position to run the GDPR analysis. To do that, one needs to run the report phase of the GDPR checker passing the Excel worksheet. At the end of the analysis, Julia produces a GDPR report that contains all leakages of sensitive data that were not explicitly allowed by the given privacy policy. For each leakage, the user can inspect the complete data flow from the source of sensitive data to the leakage point, in order to understand how such data might be leaked. In this way, the user can check all the data flows that might potentially break the privacy policy identified during the design of the information system.

![Figure 5: WebGoat GDPR report](image)
For instance, Figure 5 reports a part of the GDPR report obtained on WebGoat. Such report highlights that there are several classes (all updating the profile of Employee) that leak the credit card number of employees to the database. Since the list of allowed flows presented in Figure 4 does not include the flow of CreditCard into DB, then such flow is reported as not allowed. Similarly, Julia discovers that there is a flow from a password to Internet, and this is reported as well since it is not one of the allowed flows.

For each of these unallowed flows, the GDPR report represents the complete flow of the sensitive data to the leakage points.

For instance, Figure 6 shows one of the flows from the credit card to the database. In particular, the credit card number is retrieved by calling Employee.getCcn() (that returns the value of the tainted field Employee.ccn); it is then passed to method `setString` of a `java.sql.PreparedStatement` (method `setString` is tagged as a leakage point). In particular, line 207 of `CrossSiteScripting.UpdateProfile` contains the code `ps.setString(10, employee.getCcn())` that leaks the credit card number to the database.
The other flow is different and rather more complex. It involves the disclosure of a password to Internet, in particular, through an HTML component. Figure 7 reports this flow. The access of sensitive data and its leakage occur at line 166 and 165 of class \texttt{WsSAXInjection}, respectively:

\begin{verbatim}
165: return new B(HtmlEncoder.encode("You have changed the password for userid 
166: + changer.getId() + " to ' " + changer.getPassword() + " ' ");
\end{verbatim}

The sensitive data flows retrieved at line 166 is then passed to method \texttt{HtmlEncoder.encode}:

\begin{verbatim}
140: public static String encode(String s1)
141: {
142:   StringBuffer buf = new StringBuffer();
143:   for (i = 0; i < s1.length(); ++i) {
144:     char ch = s1.charAt(i);
145:     String entity = i2e.get(new Integer((int) ch));
146:     ... buf.append(ch);
147:     ...}
148:   return buf.toString();
149: }
\end{verbatim}

The flow graph explains that sensitive data is passed to the beginning of this method; it is then read at line 145, later read and assigned to local variable \texttt{ch} at line 147; it flows into variable \texttt{entity} at line 149; it is appended to \texttt{buf} at line 159; and it is finally returned to the callee at line 168. This example shows that the flow graph provides full detail about the propagation of sensitive data. This is invaluable in order to understand if and how the flow might represent a problematic security breach, violating the GDPR policy.

\textbf{Why Apply Privacy Static Analysis}

GDPR imposes significant sanctions, up to €20 million or 4\% of global turnover, in case of a data breach incident. Reducing data leakage risk and demonstrating the use of tailor-made solutions to support data protection is fundamental in \textit{lowering both the costs of a breach, and the related financial penalties}.

Adherence to GDPR requires several requirements to be respected. Julia for Privacy solution supports the DPO’s team in responding to these on different levels.

First, it permits to \textit{gain full awareness} of which data is handled by a software, and how. Applications are often designed for a variety of purposes and may store and handle data in ways that the organization is not aware of. GDPR requires that only the minimum necessary data is collected and handled.

A central benefit of the solution is the possibility to \textit{apply a specific Privacy Policy} to each type of data to define permitted and non-permitted operations. This is a key element in GDPR compliance to guarantee the correct handling of each type of data, and to reduce risk of sensitive data leakage.

As companies’ resources for GDPR compliance are limited, the solution has been designed to support a \textit{gradual approach} in defining and resolving privacy-related
issues. It is often not feasible to address all non-conformities at once, and knowing what the high-priority risks are makes resource allocation more efficient. Automatic reports of policy violations, ranked by severity, help to promptly address the riskiest situations. Pinpointing the cause of a data leakage or incorrect use of sensitive information is extremely complex inside a large software application. An intuitive, interactive UI makes it possible to “dive” inside the sensitive data flow streams to gain a total understanding of the context, and finally to eliminate the risk at its origin.

Conclusions

Traditional security solutions are not sufficient to ensure the protection of sensitive data and compliance with EU GDPR. Tailor-made solutions are needed, which have been designed specifically to respond to these requirements to ensure data protection by design and by default.

A privacy static analysis solution like Julia for Privacy can concretely help organizations in their efforts to achieve GDPR compliance on application level. It also significantly reduces the risk of data breach and the costs connected to such incidents.

Make sure software isn’t the weak link in your GDPR compliance. Contact us to learn more about Julia for Privacy and other JuliaSoft in-depth static analysis solutions.

Bibliography
