

After Brazil's General Data Protection Law: Authorization in Decentralized Web Applications

Jefferson O. Silva

silvajo@pucsp.br

Pontifical Catholic University of
São Paulo
São Paulo, Brazil

Newton Calegari

newton@nic.br

Brazilian Network Information
Center - NIC.br
São Paulo, Brazil

Eduardo S. Gomes

egomes@pucsp.br

Pontifical Catholic University of
São Paulo
São Paulo, Brazil

ABSTRACT

Decentralized web applications do not offer fine-grained access controls to users' data, which potentially creates openings for data breaches. For software companies that need to comply with Brazil's General Data Protection Law (LGPD), data breaches not only might harm application users but also could expose the companies to hefty fines. In this context, engineering fine-grained authorization controls (that comply with the LGPD) to decentralized web application requires creating audit trails, possibly in the source code. Although the literature offers some solutions, they are scattered. We present Esfinge Guardian, an authorization framework that completely separates authorization from other concerns, which increases compliance with the LGPD. We offer a case example and a brief discussion.

KEYWORDS

Access control, Decentralized Web Applications, Frameworks, Guardian, Solid

ACM Reference Format:

Jefferson O. Silva, Newton Calegari, and Eduardo S. Gomes. 2019. After Brazil's General Data Protection Law: Authorization in Decentralized Web Applications. In . ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

With the approval of the Brazilian General Data Protection Law (LGPD, Portuguese acronym) [7], several software companies may need to redesign the applications that handle the personal data of Brazilian citizens. The LGPD considers personal any data that directly or indirectly lead to the identification of a user [7]. Neglecting the LGPD requirements could mean incurring in fines up to 2% of companies' global revenue [7].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

LA WEB, May, 13-14, 2019, San Francisco, CA

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 6...\$15.00

The LGPD sets compliance requirements on the companies in charge of making decisions about the data processing (i.e., data controllers) and the companies that process personal data in the name of data controllers (i.e., data processors) [7]. Besides, the LGPD states that, in some cases, data controllers and data processors may be held liable, especially in cases where data breaches are harmful to users [7].

To avoid being classified as either data processors or data controllers (to avoid sanctions), some companies may redesign applications as decentralized web applications. In the context of this research, an application is considered decentralized when it does not hold users' data. Tim Berners-Lee and colleagues [12] propose a platform called Solid (derived from "Social linked data"), which can be described as a set of principles, conventions, and tools for building decentralized web applications. Solid is based on the principle that users should have full ownership of their data, which are stored in Web-accessible personal online datastores (pods) [12]. Pods are independent of web applications. For obtaining services, users need to authorize web applications to access their pods explicitly, by classifying web applications as trusted.

Using Solid alone leaves users solely responsible for controlling access to protected resources, which may not be enough to comply with the LGPD. The LGPD requirement of data governance (see Art. 50, Par. 2 in [7]) states that, among other things, companies should establish adequate policies to protect users' data. Nevertheless, in Solid web applications, a user would not have the means to prevent unauthorized access to their data, after classifying a web application as trusted. For example, a hospital web application may have a sensitive operation that reads personal data from the users' pods that should be accessible only by designated doctors. A violation of this access control policy would configure a data breach, in which case the hospital might still be held liable. Also, the liability risk might create the need for audits, in which case it would be necessary that the hospital demonstrated that it possesses appropriate controls, possibly directly in source code.

This context indicates that it is necessary to engineer fine-grained authorization controls, without losing the simplicity required in auditing the source code. Thus, we establish the following research question (RQ).

RQ: How to design fine-grained authorization controls to decentralized web applications that comply with the LGPD requirement of data governance?

The answer to our RQ may help companies to design fine-grained authorization policies in decentralized web applications, in compliance with the LGPD.

This work is organized as follows. In Chapter 2, we offer some background. In Chapter 3, we present Esfinge Guardian. In Chapter 4, we offer a case example. In Chapter 5, we present some related works. We conclude with a brief discussion in Chapter 6.

2 BACKGROUND

In this section, we offer some background for the understanding of the research problem domain.

2.1 Brazil's General Data Protection Law

The Brazil's General Data Protection Law (LGPD) is based on the General Data Protection Regulation (GDPR),¹ which aims at protecting the personal data of EU individuals. In total, around 120 countries adopt comprehensive privacy laws and regulations to protect personal data held by private and public bodies [2].

The LGPD applies to any individual or legal entity (public or private) with personal data processing activities that: are carried out in Brazil; offer or supply goods or services in Brazil or relate to individuals located in Brazil, and; involve personal data collected in Brazil.

The LGPD also requires companies to nominate a Data Protection Officer for not only monitoring the adoption of best practices but also reporting to the National Data Protection Authority.

2.2 Decentralized Solid Web Applications

Traditional web applications (e.g., Facebook, CRMs, and hospital applications) rely on private APIs, exclusive access control mechanisms, and dedicated data storage systems. Because users cannot move personal data to other platforms, these web applications become "data silos." We refer to these web applications as centralized [16].

Solid is a platform that supports decentralized web applications, by relying on open standards and semantic web technologies [3]. In the Solid platform, applications run in a browser or as mobile applications, while users data are stored in pods [4].

Data used by Solid web applications are stored in users' pods. Although pods can be stored locally or remotely, they typically are stored in pod servers. Pod servers manage data according to the Linked Data Platform recommendation, enabling it to manipulate data items through HTTP requests [15]. Solid servers are application-agnostic and can deal with both structured and unstructured data. Structured data is represented using RDF, a Semantic Web standard [4, 10]. Application development based on Solid platform supports portability and interoperability, so applications can be seen as an interface that works with distributed data in multiple pod server implementations.

¹<http://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf>

Identity in the Solid context is based on WebID, which allows agents (a person, an organization, etc.) to create their identities using global unique identifiers - HTTP URIs [12]. A WebID is an open and decentralized identification mechanism being developed by a W3C community group.²

The main idea behind Solid is to put people back in control of their data, allowing them to choose where to store their data. Solid aims at aligning with the view of a decentralized web, where data is decentralized and stored in personal data storages, or distributed servers in which the user has full control over its data [17].

In a decentralized web model, resources created using one application and can be read and modified by a different one without prior agreement between the applications. It is possible because the applications are interfaces to access data stored in multiple locations, such as in a pod.

2.3 Authorization in Solid

Access control is typically split into two distinct procedures: authentication, and authorization. While authentication is concerned with determining whether an agent (e.g., user, group) is whom it claims to be, authorization is responsible for verifying if the agent is allowed to access a protected resource (e.g., document) or operation (e.g., read, write, append). In this research, we focus on authorization.

The Solid project uses the Web Access Control (WAC) specification for authorizing the access to protected resources. According to the specification, WAC has the following key features:

- (1) The resources are identified by URLs and can refer to any web documents or resources;
- (2) It is declarative – access control policies are written in regular web documents;
- (3) Users and groups are also identified by URLs (WebIDs);
- (4) It is cross-domain – all of its components, such as resources, agent WebIDs, and even the documents containing the access control policies, can potentially reside on separate domains;

```
# Contents of https://alice.databox.me/docs/file1.acl
@prefix acl: <http://www.w3.org/ns/auth/acl#> .

<#authorization1>
  a          acl:Authorization;
  acl:agent   <https://alice.databox.me/profile/card#me>; # Alice's WebID
  acl:accessTo <https://alice.databox.me/docs/file1>;
  acl:mode    acl:Read,
              acl:Write,
              acl:Control.
```

Listing 1: Example WAC Document

Listing 1 shows an example of a WAC document that specifies that Alice (as identified by her WebID <https://alice.databox.me/profile/card#me>) has full access (read, write, and control) to one of her web resources, located at <https://alice.databox.me/docs/file1>.

²<https://www.w3.org/community/webid/>

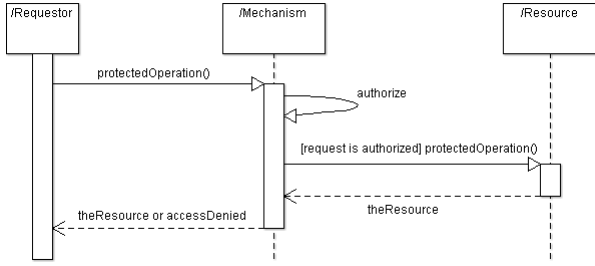


Figure 1: A conceptualization of the interception mechanism (Silva et al. [13])

Similarly, it is possible to give access to a group of agents using the `acl:agentGroup` predicate [1]. A group is a collection of members (or WebIDs) that needs to be specified in a different file. Moreover, it is possible to give access to all agents (public access) or yet to all authenticated agents. Besides, it is also possible to classify web applications as trusted. Furthermore, not every document needs its own individual access control list file. Rather, it is possible to authorize a container, which is a web location that contain multiple resources.

As mentioned, for controlling the access to the data in their pods, users need to specify WAC documents. The set of authorization modes that WAC access control systems offer does not allow the specification of fine-grained authorizations, required for decentralized web applications that needed to comply with the LGPD.

3 ESFINGE GUARDIAN

In this section, we present the Esfinge Guardian³ framework.

Essentially, the Esfinge Guardian's role is to intercept calls to protected operations. Figure 1 depicts the interception. As an example, consider a protected operation `debit()`, which should only be executed by the account owner. The call to `debit()` would be intercepted by Esfinge Guardian.

Esfinge Guardian is composed of eight main elements (see Guerra et al. [9] and Silva et al.[13] for in depth explanations). For brevity, we only present the three elements necessary to implement access control policies of our case example.

AuthorizationContext. It is the central entity that holds all the information required for an authorization, which includes the data for the subject, resource, and environment. That means all other entities should provide `AuthorizationContext` with enough information for authorization to occur.

Authorizer: Entity that implements the logic of the access control policy and may use information stored in `AuthorizationContext` if necessary. There must be at least one Authorizer. Every Authorizer must provide a response in the form of a "yes" or "no"; however, it must be possible to include other response types such as "Indeterminate."

AuthorizationMetada: Entity that indicates which resources or their operations must be intercepted by the authorization

mechanism. A requirement is that this element must be of metadata type so that it can be used declaratively. Esfinge Guardian uses Java annotations as the implementation of this element; however, it can be considered a general marking element that is independent of a specific technology.

4 CASE EXAMPLE

In this section, we present how Esfinge Guardian provides developers with appropriate tools for separating business concerns from the authorization logic without compromising simplicity.

We expand the example previously presented in the introduction. Consider the following access control policy of a hypothetical hospital web application. *Only designated doctors can read sensitive data from the patient's pods as long as they are inside the hospital facilities.*

Esfinge Guardian requires three steps for implementing this access control policy. The first step is to implement authorizers, which contain the authorization logic, as shown in Listing 2. `DesigDocAuthorizer` is responsible for authorizing the designated doctors while `WithinFacilitiesAuthorizer` authorizes based on the users' location.

```

public class DesigDocAuthorizer
    implements Authorizer<DesignatedDoctorsOnly> {
    public Boolean authorize( AuthorizationContext ctx,
        DesignatedDoctorsOnly ddo ) {
        Set<String> docs = ctx.subject("designated-doctors");
        //retrieve other relevant information from ctx
        return // authorization logic;
    }
}

public class WithinFacilitiesAuthorizer
    implements Authorizer<WithinHQ> {
    public Boolean authorize( AuthorizationContext ctx,
        WithinHQ w ) {
        Map<String> coordinates = ctx.subject("posit");
        //retrieve other relevant information from ctx
        return // authorize based on subject coordinates
    }
}
  
```

Listing 2: Authorizers' implementation

The second step requires the binding of authorizers to domain annotations, as shown in Listing 3.

```

// Retention and ElementType suppressed
@AuthorizerClass(DesigDocAuthorizer.class)
public @interface DesignatedDoctorsOnly {
}

@AuthorizerClass(WithinFacilitiesAuthorizer.class)
public @interface WithinHQ {
}
  
```

Listing 3: Binding authorization annotations with respective implementations

In the final step, developers should use the domain annotations (`@DesignatedDoctorsOnly` and `@WithinHQ`) to protect sensitive operations, as presented in Listing 4.

³<https://github.com/EsfingeFramework/guardian>

```
@WithinHQ
@DesignatedDoctorsOnly
public UserData readUserSensitiveData(
    User user, CallerLocation cl) {
    return // retrieve user data from pods;
}
```

Listing 4: A web application method that reads sensitive data from users with access control managed by Esfinge Guardian

We list two benefits of using Esfinge Guardian for managing access control in decentralized web applications compliant to LGPD. First, by providing a mechanism for separating authorization from business concerns, Esfinge Guardian eases that a specialized privacy team to work independently from other software engineers (the principle of least privilege). In this way, only the developers in the privacy team would become liable in case of data breaches. Second, the use of domain annotations makes the code more readable for audits.

5 RELATED WORKS

This research combines topics usually addressed separately. We are not aware of any research that addresses the LGPD from a software engineering perspective. Alternatively, some researchers studied approaches that map GDPR principles to software design. For example, Danezis et al. [5] provide an inventory of privacy design strategies and technical building blocks of various degrees of maturity from research and development. Koops and Leenes [11] discuss whether the GDPR Privacy-by-Design principle entails hard-coding privacy requirements into applications.

On decentralized web applications, Berners-Lee [3] introduces the concept of decentralized application architectures. Smith et al. [14] introduce a functional decentralized application called WebBox, while Dodson et al. [6] present Musubi, a disintermediated interactive social feeds application for mobile devices.

Concerning metadata-based frameworks such as Esfinge Guardian, Guerra et al. [8] investigate metadata usage in existing frameworks and document recurrent solutions as architectural patterns.

6 DISCUSSION AND CONCLUSION

The LGPD requires companies to adopt a comprehensive data governance approach, including data profiling, data lineage, data masking, test-data management, and data archives. Also, specialized professionals are required to design and handle personal data.

In this work, we show how Esfinge Guardian can be used to manage authorizations in decentralized web applications to increase compliance with the LGPD's data governance requirements. Besides the offered examples, Esfinge Guardian could be used to anonymize personal data, with the Authorizers filtering information that could lead to users identification. Our next works include applying Esfinge Guardian to a real web application and investigating the consequences.

ACKNOWLEDGMENTS

To the Brazilian Network Information Center (NIC.br) and the Web Technologies Study Center (Ceweb.br) for supporting this research.

REFERENCES

- [1] [n. d.]. Basic Access Control ontology. <https://www.w3.org/ns/auth/acl/#>
- [2] David Banisar. 2011. Data Protection Laws Around the World Map. *SSRN Electronic Journal* (2011).
- [3] Tim Berners-Lee. 2009. Socially Aware Cloud Storage. <https://www.w3.org/DesignIssues/CloudStorage.html>
- [4] David Berners-Lee, Tim and Chen, Yuhsin and Chilton, Lydia and Connolly, Daniel and Dhanaraj, Ruth and Hollenbach, James and Lerer, Adam and Sheets. 2006. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop*.
- [5] George Danezis, Josep Domingo-Ferrer, Marit Hansen, Jaap-Henk Hoepman, Daniel Le Metayer, Rodica Tirtza, and Stefan Schiffner. 2015. Privacy and Data Protection by Design - from policy to engineering. December (jan 2015). <https://doi.org/10.2824/38623arXiv:1501.03726>
- [6] Ben Dodson, Ian Vo, T.J. Purtell, Aemon Cannon, and Monica Lam. 2012. Musubi: Disintermediated Interactive Social Feeds for Mobile Devices. *Proceedings of the 21st international conference on World Wide Web - WWW '12* (2012), 211.
- [7] Brazilian Government. 2018. General Data Protection Law. http://www.planalto.gov.br/ccivil/_03/_Ato2015-2018/2018/Lei/L13709.htm
- [8] Eduardo Guerra, Clovis Fernandes, and Fábio Fagundes Silveira. 2010. Architectural patterns for metadata-based frameworks usage. *Proceedings of the 17th Conference on Pattern Languages of Programs - PLOP '10* October 2010 (2010), 1–25. <https://doi.org/10.1145/2493288.2493292>
- [9] E M Guerra, J O Silva, and C T Fernandes. 2015. A Modularity and Extensibility Analysis on Authorization Frameworks. 2, 1 (2015).
- [10] James Hollenbach, Joe Presbrey, and Tim Berners-Lee. 2009. Using RDF metadata to enable access control on the social semantic web. *CEUR Workshop Proceedings* 514 (2009).
- [11] Bert-Jaap Koops and Ronald Leenes. 2014. Privacy regulation cannot be hardcoded. A critical comment on the privacy by design' provision in data-protection law. *International Review of Law, Computers & Technology* 28, 2 (may 2014), 159–171. <https://doi.org/10.1080/13600869.2013.801589>
- [12] Andrei Vlad Sambra, Essam Mansour, Sandro Hawke, Maged Zereba, Nicola Greco, Abdurrahman Ghanem, Dmitri Zagidulin, Ashraf Abounaga, and Tim Berners-Lee. [n. d.]. Solid: A Platform for Decentralized Social Applications Based on Linked Data. ([n. d.]).
- [13] J.O. Silva, E.M. Guerra, and C.T. Fernandes. 2013. An extensible and decoupled architectural model for authorization frameworks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 7974 LNCS.
- [14] Daniel Alexander Smith, Max Van Kleek, Oshani Seneviratne, Monica schraefel, Alexandre Bertails, Tim Berners-Lee, Wendy Hall, and Nigel Shadbolt. 2012. WebBox: Supporting Decentralised and Privacy-respecting Micro-sharing with Existing Web Standards. In *21st International World Wide Web Conference*.
- [15] Ashok Malhotra Steve Speicher, John Arwe. 2015. Linked Data Platform 1.0. W3C Recommendation. <https://w3.org/TR/ldp/>
- [16] Max Van Kleek, Daniel Smith, Nigel Shadbolt, and M.c. Schraefel. 2012. A decentralized architecture for consolidating personal information ecosystems: The WebBox. *Pim 2012* (2012). <http://eprints.ecs.soton.ac.uk/23200/>
- [17] Ruben Verborgh. 2018. Decentralizing the Semantic Web through incentivized collaboration. In *Proceedings of the 17th International Semantic Web Conference: Blue Sky Track (CEUR Workshop Proceedings)*, Marieke van Erp, Medha Atre, Vanessa Lopez, Kavitha Srinivas, and Carolina Fortuna (Eds.), Vol. 2189.