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

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

Currently, decentralized Web Applications (Web Apps) 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), privacy violations could mean incurring in hefty fines. We propose the use of Esfinge Guardian framework for increasing compliance with the LGPD in the context of decentralized Web Apps.

KEYWORDS

Access Control, Decentralized Web Apps, Frameworks, Guardian, Solid

ACM Reference Format:
Jefferson O. Silva, Newton Calegari, and Eduardo S. Gomes. 2018. After Brazil's General Data Protection Law: Authorization in Decentralized Web Apps. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

With the approval of the Brazilian General Data Protection Law (LGPD, Portuguese acronym) [4], 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 [4]. Neglecting the LGPD requirements could mean incurring in fines up to 2% of companies' global revenue [4].

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) [4]. 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 [4].

To avoid being classified as either data processors or data controllers (as an attempt to avoid sanctions), some companies may redesign applications as decentralized Web Applications (Web Apps). In the context of this research, an application is considered decentralized when it does not hold users' data. Tim Berners-Lee and colleagues [7] propose a platform called Solid (derived from "**So**cial **li**nked **d**ata"), which can be described as a set of principles, conventions, and tools for building decentralized Web Apps. 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) [7]. Pods are independent of Web Apps. For obtaining services, users need to authorize Web Apps to access their pods explicitly, by classifying Web Apps 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 [4]) states that, among other things, companies should establish adequate policies to protect users' data. Nevertheless, in Solid Web Apps, a user would not have the means to prevent unauthorized access to their data, after classifying a Web App as trusted. For example, a bank Web App may have a sensitive operation that reads personal data from the users' pods that should be accessible only by account managers. A violation of this access control policy would configure a data breach, in which case the bank might still be held liable. Also, the liability risk might create the need for audits, in which case it would be necessary that the company demonstrated that it possesses appropriate controls, possibly directly in source code.

Thus, we establish the following research question (RQ).

RQ: How to design decentralized Web Apps to increase compliance with the LGPD requirement of data governance?

We claim that the answer to our RQ may help companies to design authorization solutions in decentralized Web Apps in increased 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.

LA WEB, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-9999-9/18/06...$15.00
https://doi.org/10.1145/1122445.1122456

2 BACKGROUND

In this section, we offer some background on the Brazil's General Data Protection Law, on decentralized Web Apps in Solid, and on access control in Solid.

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 [1].

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 who will be in charge of monitoring the adoption of best practices for personal data protection and for reporting to the National Data Protection Authority.

2.2 Decentralized Solid Web Apps

Traditional Web Apps (e.g., Facebook, Twitter, and Santander) implement private APIs, access control and data storage mechanisms. Because users cannot move their data to other platforms, these Web Apps become "data silos." We refer to these Web Apps as centralized Web Apps.

Solid is a platform that supports decentralized Web Apps, by relying on open standards and semantic web technologies. In the Solid platform, applications run in a browser or as mobile applications, while users data are stored in a Web-accessible personal online data store (pods). However, protocols for access controls, communications between servers and applications are specific to Solid [7].

Data used by Solid Web Apps 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 [9]. Solid servers are application-agnostic and can deal with both structured and unstructured data. Structured data is represented using RDF, a Semantic Web standard [2]. 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.

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 [7]. A WebID is an open and decentralized identification mechanism being developed by a W3C community group.[2]

[1] http://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf
[2] https://www.w3.org/community/webid/

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 [11].

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 Access Control 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).

The Solid project uses the Web Access Control (WAC) specification for controlling the access to protected resources. WAC specifies a decentralized cross-domain access control system, similar to existing access control models. 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 (specifically, by 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`.

Similarly, it is possible to give access to a group of agents using the `acl:agentGroup` predicate. 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. It is

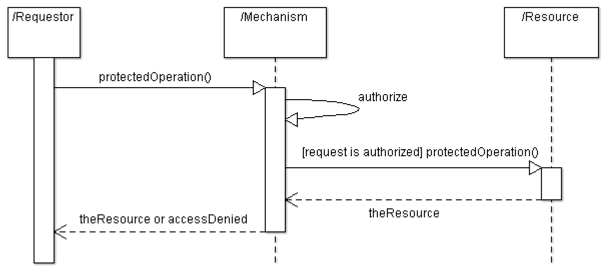

Figure 1: A conceptualization of the interception mechanism (Silva et al. [8])

also possible to classify Web Apps as trusted. Furthermore, not every document needs its own individual access control list file. Rather, it is possible to create an authorization to a container, which is a web location that contain multiple resources.

3 ESFINGE GUARDIAN

In this section, we present the Esfinge Guardian[3] 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. [5] and Silva et al.[8] 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.

[3]https://github.com/EsfingeFramework/guardian

We expand the example previously presented in the introduction. Consider the following access control policy of a hypothetical bank Web App. *Only managers can read sensitive data from the users' pods as long as they are inside the bank 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. `ManagerAuthorizer` is responsible for authorizing managers while `WithinFacilitiesAuthorizer` authorizes based on the users' location.

```
public class ManagerAuthorizer
               implements Authorizer<ManagersOnly> {
  public Boolean authorize( AuthorizationContext ctx,
                            ManagersOnly mo ) {
    Set<String> roles = ctx.subject("roles");
    //retrieve other relevant information from ctx
    return // managers 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: Manager and WithinFacilities Authorizers

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

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

Listing 3: Binding authorization annotations with respective implementations

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

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

Listing 4: A Web App 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 Apps 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

Three sets of works are relevantly related to this work the first consists of works that address privacy laws and regulations. We could not find relevant research on LGPD (possibly because it has been approved recently). Mantelero [6] offers a comprehensive view on GDPR.

The second set consists of works on frameworks that use metadata to adapt their behavior. The first author has relevant works on this topic [8]. Guerra et al. [5] also provide a comprehensive view on metadata-based frameworks.

The third set consists of works that focus on decentralized Web Apps. Kleek et al. [10] propose a decentralized architecture called WebBox to support easy maintenance and repurposing of one's data for private, social or public publishing, collaboration, and reuse. Sambra et al. [7] explain data ownership, their principles, pros, and possibilities. Dodson et al. [3] present Musubi a mobile social application platform that enables users to share any data type in real-time feeds created by any application on the phone. The authors' feed abstraction allows users to exercise access control easily. All data reside on the phone, granting users the freedom to apply applications of their choice.

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 Apps 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.

ACKNOWLEDGMENTS

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

REFERENCES

[1] David Banisar. 2011. Data Protection Laws Around the World Map. *SSRN Electronic Journal* (2011).

[2] 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 3textsuperscriptrd International Semantic Web User Interaction Workshop*.

[3] 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. https://doi.org/10.1145/2187836.2187866

[4] Brazilian Government. 2018. General Data Protection Law. http://www.planalto.gov.br/ccivil{_}03/{_}Ato2015-2018/2018/Lei/L13709.htm

[5] E M Guerra, J O Silva, and C T Fernandes. 2015. A Modularity and Extensibility Analysis on Authorization Frameworks. 2, 1 (2015).

[6] Alessandro Mantelero. 2013. The EU Proposal for a General Data Protection Regulation and the roots of the right to be forgotten'. *Computer Law & Security Review* 29, 3 (jun 2013), 229–235. https://doi.org/10.1016/j.clsr.2013.03.010

[7] Andrei Vlad Sambra, Essam Mansour, Sandro Hawke, Maged Zereba, Nicola Greco, Abdurrahman Ghanem, Dmitri Zagidulin, Ashraf Aboulnaga, and Tim Berners-Lee. [n. d.]. Solid: A Platform for Decentralized Social Applications Based on Linked Data. ([n. d.]).

[8] 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.

[9] Ashok Malhotra Steve Speicher, John Arwe. 2015. Linked Data Platform 1.0. W3C Recommendation. https://w3.org/TR/ldp/

[10] 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/

[11] 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.