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

Write the abstract here.

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, 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

With the approval of the Brazilian General Data Protection Law (LGPD),[1] several software companies may need to redesign the applications that handle the personal data of Brazilian citizens. LGPD considers personal any data that directly or indirectly lead to the identification of a user [REF]. Violations can lead to fines up to 2% of global revenue.

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). LGPD states that, in some cases, data controllers and/or data processors may be held liable for any cause of action that involves harm to data subjects [REF].

To avoid being classified as either data processors or data controllers, some companies may redesign applications as decentralized Web Applications (Web Apps). An application is considered decentralized when it does not hold users' data [REF]. Tim Berners-Lee and colleagues [REF] 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). 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.

Nevertheless, this approach leaves users solely responsible for controlling access to protected resources, which may not be enough to comply with the LGPD requirements of Privacy by Design. For example, a bank Web App may have an operation that reads personal data from the pods that should be accessible by account managers but not by analysts. A violation of this access control policy would configure a data breach, in which case the companies might be required to show that they applied appropriate controls, possibly from the software design.

The current body of literature does not ...

RQ1: How decentralized Web Apps could be designed for privacy to comply with LGPD in an auditable fashion?

The answer to our RQ may help companies...

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 Brazilian Data Protection Law

The LGPD is strong inspired by the European GDPR. The Brazilian Bill, as the European one, defines cross-border jurisdiction, thus the Bill is applicable to any organizations processing personal data of Brazilian residents, whether it is headquartered in Brazil or not.

LGPD has also included the right of data portability, the right of access to personal data by the owner, and the right of erasure. Differently of the GDPR, which imposes 30 days for the controllers to comply with these requests, the LGPD imposes 15 days.

The Brazilian law also requires companies to nominate a Data Protection Officer (DPO) 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 (ANPD).

[1] http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2018/Lei/L13709.htm

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, June 03–05, 2018, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

The regulation defines the concepts of personal data as "any data, isolated or aggregated to another, that may allow the identification of a nutral person or subject them to a certain behavior" [REF] (IAPP: https://iapp.org/news/a/the-new-brazilian-general-data-protection-law-a-detailed-analysis/); sensitive data refers to data that may be subject to discriminatory practices, such as political opinion, sexual life, religious belief, genetic and biometric data, and it should have additional security layers; Unless it is possible to reverse-engineering the anonimized data, the law does not apply to this kind of data.

In a technical perspective, the efforts related to the decentralization of the Web help to build systems that are privacy-friendly, respecting user's privacy and in compliance with the regulations. Ainda nao sei se esse paragrafo fica nessa section ou na proxima.

2.2 Solid and the Decentralized Web

Traditional social web applications, such as Facebook, Twitter, and others control its own data and have its own authenticatoin and access control mechanisms, transforming them into centralized applications. In the contrast of this approach, there are emergent solutions propposing a new perspective to enable decoupling application logic and user data, allowing it to create privacy-friendly services on the Web.

Solid is a platform that supports decentralized social web applications, relying on open standards and on semantic web technologies. In the Solid platform, applications run in a browser or as mobile applications, while users' data are stored in a Web-accesibile personal on-line datastore (pod). Protocols for authenticatoin, access control, communications among servers and applications are specified in the platform [3].

Data used by applications built on top of Solid is stored in users' pods, while pods can be stored on pod servers. Pod servers manage data according to the Linked Data Platform recommendation, enabling it to manipulate data items through HTTP requests [**?**]. Solid servers are, as defined by their authors [3], application-agnostic, and can deal with both structured and unstructured data. The first is represented using RDF, a Semantic Web standard, and the latter is any type of data, such as videos, images, HTML web pages.

Application development based on (in???) 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 own identities using global unique identifiers - HTTP URIs [4]. WebID is an open and decentralized identification mechanism being developed by a W3C community group[2].

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).

[2]https://www.w3.org/community/webid/

```
# 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.
```

Figure 1: Example WAC Document

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

# Group authorization, giving Read/Write access to a group, which is
# specified in the 'work-groups' document.
<#authorization2>
    a              acl:Authorization;
    acl:accessTo   <https://alice.example.com/docs/shared-file1>;
    acl:mode       acl:Read,
                   acl:Write;
    acl:agentGroup <https://alice.example.com/work-groups#Accounting>;
```

Figure 2: WAC document with group permission

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.

Figure 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`.

In Figure 2, we can see that 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. In this case, members of the group `Accounting` can read and write the Alice's resource located at `https://alice.example.com/docs/shared-file1`.

```
# Contents of https://alice.example.com/work-groups
@prefix  acl:   <http://www.w3.org/ns/auth/acl#>.
@prefix  vcard: <http://www.w3.org/2006/vcard/ns#>.

<> a acl:GroupListing.

<#Accounting>
    a                 vcard:Group;
    vcard:hasUID      <urn:uuid:8831CBAD-1111-2222-8563-F0F4787E5398:ABGro
    dc:created        "2013-09-11T07:18:19+0000"^^xsd:dateTime;
    dc:modified       "2015-08-08T14:45:15+0000"^^xsd:dateTime;

    # Accounting group members:
    vcard:hasMember <https://bob.example.com/profile/card#me>;
    vcard:hasMember <https://candice.example.com/profile/card#me>.
```

Figure 3: WAC document listing group members

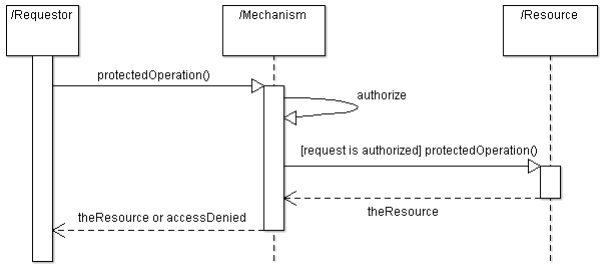

Figure 4: A conceptualization of the interception mechanism. Silva et al. [5]

Figure 3 depicts the listing of a group. In this case, `Bob` and `Candice` belong to the `Accounting` group. Additionally, it is possible to give access to all agents (public access) or yet to all authenticated agents. It is 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.

2.4 Frameworks (temp)

Guerra et al. [2]

3 ESFINGE GUARDIAN

In this section, we present the Esfinge Guardian[3] framework. Essentially, the Esfinge Guardian framework is an intercepting mechanism between a requester and a protected operation. Figure 4 illustrates Guardian intercepting a requester's access to a protected operation on a resource.

As depicted in Figure 5, Guardian is composed of eight main elements.

`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. It is also an interface with the user, meaning that all other points must be hidden from the user.

`GuardianInterceptor`: is the entity responsible for abstracting the different existing interception technologies such as aspect-orientation, CGLib, dynamic proxy etc.

`Invoker`: is an entity with the ability to mimic the operation performed by the subject on a protected resource. In the Esfinge Guardian framework, this element can execute methods; however, it is important to note that is just one of the possibilities since the architectural model is general.

`Populator`: It is the entity that contains the data extraction logic for authorization. Information for authorization can be anywhere such as databases, files, shared variables, user session, arguments, Internet, etc. For this reason, Populator is an entity that knows how to obtain information from all these places.

`PopulatorProcessor`: Entity that gathers and executes all defined Populators in the application.

`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 its response to the AuthorizationProcessor, usually a "yes" or "no"; however, it must be possible to include other response types such as "Indeterminate".

`AuthorizerProcessor`: Entity that contains the combining algorithm for all the Authorizers defined in the application.

`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 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 1. `ManagerAuthorizer` is responsible for authorizing managers while `WithinFacilitiesAuthorizer` authorizes based on the users' location.

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

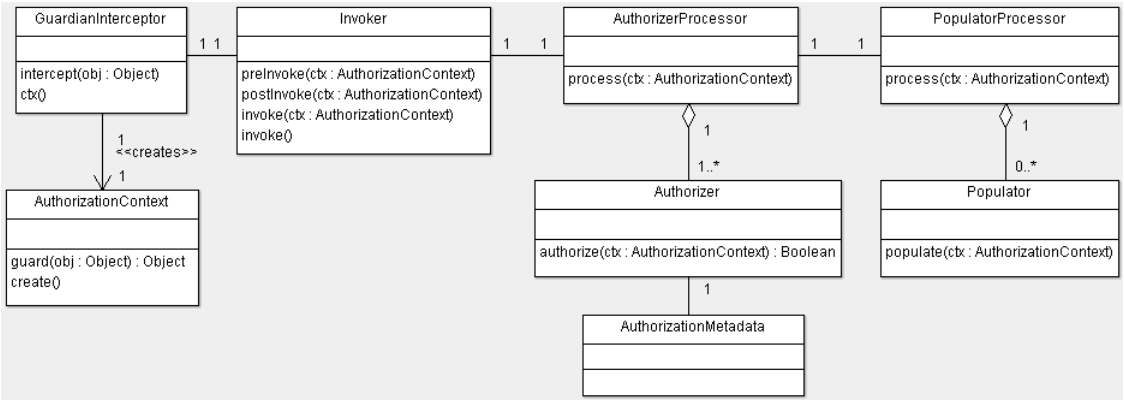

Figure 5: Esfinge Guardian class diagram (Silva et al. [5])

```
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 1: Manager and WithinFacilities Authorizers

The second step requires the binding of authorizers to domain annotations, as shown in Listing 2.

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

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

Listing 2: Binding authorization annotations with respective implementations

Finally, developers should use the domain annotations (`@ManagersOnly` and `@WithinHQ`) to protect sensitive operations, as presented in Listing 3.

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

Listing 3: 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 DISCUSSION

Here goes some discussion.

6 RELATED WORKS

A bit about others work.

7 CONCLUSION

Here goes a conclusion.

8 BACKUP

1.LGPD is based on the General Data Protection Regulation (GDPR),[4] 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].

[4]http://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

REFERENCES

[1] David Banisar. 2011. Data Protection Laws Around the World Map. *SSRN Electronic Journal* (2011). https://doi.org/10.2139/ssrn.1951416

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

[3] Essam Mansour, Andrei Vlad Sambra, Sandro Hawke, Maged Zereba, Sarven Capadisli, Abdurrahman Ghanem, Ashraf Aboulnaga, and Tim Berners-Lee. 2016. A Demonstration of the Solid Platform for Social Web Applications. In *Proceedings of the 25th International Conference Companion on World Wide Web (WWW '16 Companion)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 223–226. https://doi.org/10.1145/2872518.2890529

[4] Andrei Vlad Sambra, Essam Mansour, Sandro Hawke, Maged Zereba, Nicola Greco, Abdurrahman Ghanem, Dmitri Zagidulin, Ashraf Aboulnaga, and Tim Berners-Lee. 2016. Solid: A Platform for Decentralized Social Applications Based on Linked Data.

[5] 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. https://doi.org/10.1007/978-3-642-39649-6-44