

RESTGuardian: a system for controlling personal and sensitive data in REST API responses

Carlos Henrique Souza Silva

Federal University of Itajubá (UNIFEI)
Itajubá, Brazil
carloshsouza@unifei.edu.br

Bruno Tardiole Kuehne

Federal University of Itajubá (UNIFEI)
Itajubá, Brazil
brunokuehne@unifei.edu.br

Dionisio Machado Leite Filho

Federal University of Mato Grosso do Sul (UFMS)
Campo Grande, Brazil
dionisio.leite@ufms.br

Maycon Leone Maciel Peixoto

Federal University of Bahia (UFBA)
Salvador, Brazil
maycon.leone@ufba.br

Bruno Guazzelli Batista

Federal University of Itajubá (UNIFEI)
Itajubá, Brazil
brunoguazzelli@unifei.edu.br

ABSTRACT

Nowadays is essential to protect personal and sensitive data to ensure compliance with regulations such as the LGPD (Brazilian General Data Protection Law Number 13.709/2018) and the GDPR (General Data Protection Regulation Number 2016/679 EU). This paper presents the development of RESTGuardian, a controller for personal and sensitive data in responses from REST (Representational State Transfer) APIs (Application Programming Interfaces). RESTGuardian acts as a security middleware between the frontend and backend of a web or mobile application. It aims to alert developers so they do not create insecure APIs and can mitigate the risk of Excessive Data Exposure, as reported by the OWASP (Open Web Application Security Project). Additionally, it serves as a tool for Data Protection Officers (DPOs), offering a view of REST APIs in the development environment, focusing on LGPD compliance to prevent the deployment of insecure APIs in production.

KEYWORDS. Privacy. Data Protection Officers. Middleware.

Paper topics (D&SP - PO em Defesa e Segurança Pública, TEL&SI – PO em Telecomunicações e Sistemas de Informações)

1. Introduction

Over the years, the use of web and mobile applications has become common due to the convenience and accessibility they provide. With smartphones and other mobile devices, people can access information and services from anywhere with an internet connection, which enhances user experience [Adetunji et al., 2020]. As a result, individuals share personal and sensitive data in web and mobile applications due to a lack of awareness, seeking convenience, trusting the company to protect their privacy, and, in some cases, due to a lack of alternatives [Nguyen et al., 2021]. Typically, the servers of these systems consist of REST APIs, due to their ability to be flexible, scalable, and independent.

In 2019, the OWASP (Open Web Application Security Project) released a list of the top 10 security threats in APIs, highlighting Excessive Data Exposure as the third major threat. In the updated 2023 version, this position was replaced by the threat Broken Object Property Level Authorization, which is a combination of Excessive Data Exposure and Mass Assignment threats [OWASP, 2023]. This update addresses excessive data exposure, now in a new security context related to broken object property level authorization. Excessive data exposure occurs when an application while responding to an API, discloses more information than necessary for a user to perform a certain action [Genfer and Zdun, 2022]. API developers often make the mistake of exposing all object properties without considering their sensitivity, relying solely on client-side code to filter the data. From this issue, malicious actors can intercept and acquire users' personal and sensitive data. Therefore, there is a need to apply good information security practices and secure development to ensure that the source code of these APIs does not contain vulnerabilities.

In 2018, in Brazil, Federal Law No. 13,709 was approved, the General Data Protection Law (LGPD), which came into effect in August 2020. It was created to ensure the privacy, security, and control of individuals' data, both in physical and digital mediums, through important and mandatory guidelines for the collection, processing, and storage of personal data [Brasil, 2018]¹. The General Data Protection Law is inspired by the General Data Protection Regulation (GDPR)², which has been used in European Union countries since 2018. According to several security websites, Brazil remains at the top of the global data breach ranking, with an alarming number of sensitive data exposed in the last years³. Cybercriminals usually target email addresses, credentials, Individual Taxpayer Registration, National Registry of Legal Entities and passports.

In light of the above, it is understood that developers need to adapt to the new reality and seek knowledge about the LGPD and information security. This can ensure that personal and sensitive data are protected not only in managerial processes but also at the code level.

In the current context of data breaches, the LGPD plays a crucial role by establishing stringent requirements for the protection of user data by software development companies and service providers. Thus, this study aims to assist developers during the development process by using a control system for personal and sensitive data in REST API responses, to mitigate threats of excessive data exposure and serve as a supporting tool for Data Protection Officers (DPOs).

The hypothesis of this study is to analyze if it is possible to develop a software solution aligned with the LGPD that (i) assists developers during the development process, and (ii) serves as a tool for Data Protection Officers, to enhance the security level of data returned from REST APIs, preventing excessive data exposure.

The general objective was to develop open-source software capable of analyzing responses returned by REST APIs, seeking personal and sensitive data. Based on this, the system should

¹<https://lgpd-brazil.info/>

²<https://gdpr.eu/>

³<https://brazilian.report/tech/2023/03/04/andrade-gutierrez-data-breach/>
<https://hackenproof.com/blog/industry-news/brazilian-personal-data-exposure>

manage the APIs to ensure the non-exposure of unnecessary data, ensuring compliance with the LGPD from the development process. The system focuses on the development environment, where it plays the role of signaling possible data leaks, allowing corrections to be made before deploying the APIs in a production environment.

This paper is organized as follows: Section 2 addresses the theoretical background; the proposal for the research problem solution and system implementation are discussed in Section 3; finally, Section 4 contains the final considerations and premises for future work.

2. Theoretical Background

2.1. General Data Protection Law (LGPD)

LGPD came into effect on August 18, 2020, with the main objective of ensuring the protection of fundamental rights to freedom and privacy of Brazilians. LGPD also discusses data processing agents and the Data Protection Officer (DPO). The agents are the controller and the processor. The controller is the natural person or legal entity, whether public or private, who has the authority to make decisions related to the processing of personal data. They are responsible for defining the guidelines and purposes of this processing. The processor, who can also be a natural person or legal entity, whether public or private, processes personal data on behalf of the controller. The processor operates according to the instructions provided by the controller. In summary, the controller makes key decisions regarding the processing of personal data, while the processor carries out these activities following the controller's guidelines [Brasil, 2018].

The Data Protection Officer (DPO) is appointed by the controller and processor to act as an intermediary between them, the data subjects, and the ANPD (National Data Protection Authority) [Brasil, 2018]. Its main responsibilities include receiving complaints and communications from data subjects, providing explanations, and taking appropriate measures. The DPO also handles communications from the national authority and guides employees and contractors of the organization on the necessary practices to protect personal data. Additionally, the DPO carries out other tasks determined by the controller or regulated. These activities are essential for promoting a culture of personal data protection within the organization.

2.2. DPO as a Service

An important measure of compliance with the LGPD is the appointment of a data processing officer responsible for handling this information, with an efficient option to assist in this process being the hiring of a DPO as a Service. The DPO as a Service is an outsourced service that provides all the responsibilities of the DPO. It is an efficient solution for companies that wish to enjoy this important service without needing to invest in internal training or hire a new professional with the necessary requirements [DPO, 2018].

Opting for DPO as a Service facilitates compliance with regulations and streamlines LGPD management after implementation. The DPO professional or department is responsible for providing training, ensuring good data handling practices and security at all levels of the company, as well as handling requests from data subjects and the ANPD [DPO, 2018].

There are several tools to assist a DPO in its activities, including data management solutions, risk assessment, compliance and data analysis, privacy management, risk management, and compliance software. Additionally, communication and collaboration tools are essential for the DPO to interact with other stakeholders in the organization. These tools help automate tasks and make data protection more efficient [Pessoa et al., 2021]. At this point, the use of Middlewares becomes interesting.

Middleware simplifies development by avoiding repetitive custom integrations when connecting different data sources, application components, devices, or computing resources. It offers services and applications that communicate through messaging frameworks such as REST, JSON

(JavaScript Object Notation), XML (Extensible Markup Language), SOAP (Simple Object Access Protocol), or web services [Bakken, 2001].

2.3. OWASP - API Security Project

Considering the importance of personal data protection as established by LGPD, it is essential to adopt robust security measures, such as those proposed by the OWASP (Open Web Application Security Project) API Security Project.

API security encompasses strategies defined by OWASP to mitigate specific security vulnerabilities and risks aimed at protecting APIs against threats and ensuring the integrity and confidentiality of transmitted data. OWASP maintains an online community that produces articles, methodologies, documentation, tools, and technologies freely available in the field of web application security. The organization has also developed the Top 10 API Security Threats list, initially created in 2019, guiding risk mitigation. Furthermore, the 2023 version offers updated insights on the primary threats. OWASP's ongoing efforts strengthen application security and promote best practices in the API domain [OWASP, 2023]. For this research, two API vulnerabilities stand out:

1. **API3:2019 Excessive Data Exposure:** A security vulnerability in APIs where an excessive amount of confidential data is inadvertently exposed.
2. **API3:2023 Broken Object Property Level Authorization:** A security vulnerability in APIs where object property level authorization is improperly or inconsistently implemented. This flaw may allow an unauthorized user to access and modify specific properties of an object through the API, even if they do not have permission.

2.4. Information Security and Secure Development

The OWASP API Security project plays a crucial role in providing guidelines and effective solutions to mitigate vulnerabilities, ensuring secure and robust development, and promoting strong information protection. Information security encompasses best practices and strategies to preserve data integrity, protecting it against cyber-attacks and other threats that may affect both the company and its clients [Stallings, 2019].

Secure development is the process of creating software designed to be secure from the outset, rather than attempting to add security features after the fact. Several best practices can be followed to ensure that software is developed securely as presented in [Alenezi and Almuairfi, 2020], [Ramirez et al., 2020], [Trifonov et al., 2020], [Khan et al., 2022]. Furthermore, several technologies can be used to help ensure that software is developed securely. This includes static analysis, functional and unit testing, and testing the system software against formal models of its security functional requirements [DEVYANIN et al., 2021].

In light of the above, DevSecOps emerged as an integrated approach to information security and secure development, coupled with the practice of DevOps. DevSecOps enables the creation of software resilient to attacks and vulnerabilities throughout the entire development lifecycle [Rajapakse et al., 2022].

There are market-available solutions to enhance security and protect confidential information. Some of these solutions include the use of WAF (Web Application Firewall), which are security solutions designed to protect web applications against attacks and vulnerabilities [Amazon Web Services, 2023], data loss prevention (DLP) solutions [Tahboub and Saleh, 2014] [Faiz et al., 2020], and data monitoring and visualization systems [Bajer, 2017] [Shah et al., 2022].

RESTGuardian differs from these existing solutions by providing specific features for protecting personal and sensitive data in REST API responses, as well as emphasizing the unique benefits it brings to secure development and LGPD compliance. Therefore, it enables developers

to control and analyze sensitive data in REST API responses before making the developed system available to the production environment.

3. RESTGuardian

Based on studies conducted in the field, it was found that pressure and tight deadlines for code delivery can lead to failures caused by developers, which in turn can result in data leaks leading to non-compliance with LGPD. In this regard, an open-source solution was proposed to assist developers during the process of developing REST APIs and DPOs, called RESTGuardian. RESTGuardian allows control of personal and sensitive data in REST API responses, preventing unnecessary data exposure and ensuring compliance with LGPD.

It is made available through DockerHub⁴, enabling easy and quick installation, as well as deployment in various environments. The complete documentation of the system is also available in the Github repository⁵, providing detailed information on how to configure, use, and customize the system according to the specific needs of each user.

It is important to highlight that RESTGuardian is decoupled from the target application. This approach has the main advantage of ease of integration, as the only required modification is to direct the requests from the development application to RESTGuardian. In other words, it is sufficient to configure the application to send requests to RESTGuardian instead of directing them directly to the application's backend.

This separation between RESTGuardian and the target application provides greater flexibility and agility in the system's development and maintenance. If at any point it becomes necessary to disable RESTGuardian, this change can be made simply and quickly by altering only the request's destination URL so that they are directed straight to the application's backend.

This decoupling approach allows RESTGuardian to act as an intermediary between the frontend and backend of the target application without requiring changes to the code of the target system. This means that application development can occur independently, without directly depending on RESTGuardian. This flexibility provides greater freedom for adjustments and modifications, facilitating the evolution and maintenance of the system more efficiently.

3.1. System implementation

The RESTGuardian features a proxy server, and a web service comprising both a backend and a frontend. The developer is responsible for providing the external database for data storage, requiring it to be available and correctly configured for the system to establish connection and perform storage and retrieval operations accurately.

The proposed system was designed to be used in a development environment, acting as a security middleware between the backend and frontend of the application being developed by the developer/company.

3.1.1. Proxy server

The RESTGuardian's proxy server was implemented using the http-proxy library⁶ along with NodeJS⁷ and TypeScript (version 4.6.4)⁸. The choice of the TypeScript language offers significant benefits for the quality and consistency of the system. Being a strongly typed language, it helps to avoid type errors during development. This means that TypeScript verifies and enforces correct data type matching at compile time, thus reducing the likelihood of type-related errors during program execution.

⁴<https://hub.docker.com/repositories/carlos94souza>

⁵<https://github.com/carlos94souza/data-controller-system>

⁶<https://github.com/http-party/node-http-proxy>

⁷<https://nodejs.org/en>

⁸<https://www.typescript>

The use of the `http-proxy` library enables the proxy server to act as an intermediary between the frontend and backend of the target application. It receives client requests and forwards them to the appropriate destination, serving as a traffic redirector. The `http-proxy` library offers proxying features such as reverse proxying, allowing the proxy server to forward client requests to corresponding internal servers, ensuring greater control over request routing. Additionally, the library also provides request and response interception capabilities, offering additional flexibility in handling traffic between the frontend and backend of the application.

In this way, it is possible to intercept the response returned by the target application, capturing the JSON and using an algorithm that will traverse this JSON searching and comparing the keys with the list of restricted words saved in the database. Algorithm 1 presents the data identification code in TypeScript.

Listing 1: Data identification algorithm.

```

1  public async findPrivateData (responseObject: any, privateDataList:
    IPrivateDataList) {
2  const result = new Set()
3  if (Array.isArray(responseObject)) {
4      for (const item of responseObject) {
5          const children = this.findPrivateData(item, privateDataList)
6          for (const child of await children) {
7              result.add(JSON.stringify(child))
8          }
9      }
10 } else {
11     for (const key in responseObject) {
12         for (const type in privateDataList) {
13             if (privateDataList[type].includes(key)) {
14                 result.add(JSON.stringify({ name: key, type }))
15             }
16         }
17         if (typeof responseObject[key] === 'object') {
18             const children = this.findPrivateData(responseObject[key],
                privateDataList)
19             for (const child of await children) {
20                 result.add(JSON.stringify(child))
21             }
22         }
23     }
24     return Array.from(result).map(element => JSON.parse(element as string))
25 }

```

The function *findPrivateData* receives two parameters:

- **responseObject:** It is the object that will be analyzed for personal and sensitive data. It can be a simple object or an array containing objects.
- **privateDataList:** It is the list containing the words, being of the type declared by the interface called *IPrivateDataList*, which contains a list of words grouped by types (sensitive and personal).

The algorithm uses a recursive approach to traverse the object in search of personal and sensitive data. It starts by creating a new set called *result*, which will be used to store the found results. This set is used to ensure that there are no duplicates in the final result.

Next, the algorithm checks if the *responseObject* is an array. If it is, it iterates through each item of the array and recursively calls the *findPrivateData* function for each item. The results returned by these recursive calls are stored in the *result set*.

If the *responseObject* is not an array, the algorithm enters the *else* block. It iterates through each object property using a *for...in* loop. For each property, the algorithm checks if it matches any personal or sensitive data in the *privateDataList*. If a match occurs, the algorithm adds an object containing the property name and the corresponding type to the *result set*. After this, the algorithm checks if the property's value is an object. If it is, it recursively calls the *findPrivateData* function for the value of the object and stores the results in the *result set*.

After traversing all the object's properties, the function returns the *result set* converted into an array. Before making this conversion, the algorithm uses *JSON.stringify* to convert the objects in the *result set* into strings and then *JSON.parse* to convert these strings back into JavaScript objects.

In the end, the result of the *findPrivateData* function is an array of objects representing the personal and sensitive properties found in the original object, according to the provided list of restricted words.

Figure 1 presents the BPMN (Business Process Model and Notation) diagram illustrating the data identification process when intercepting the response of a REST API.

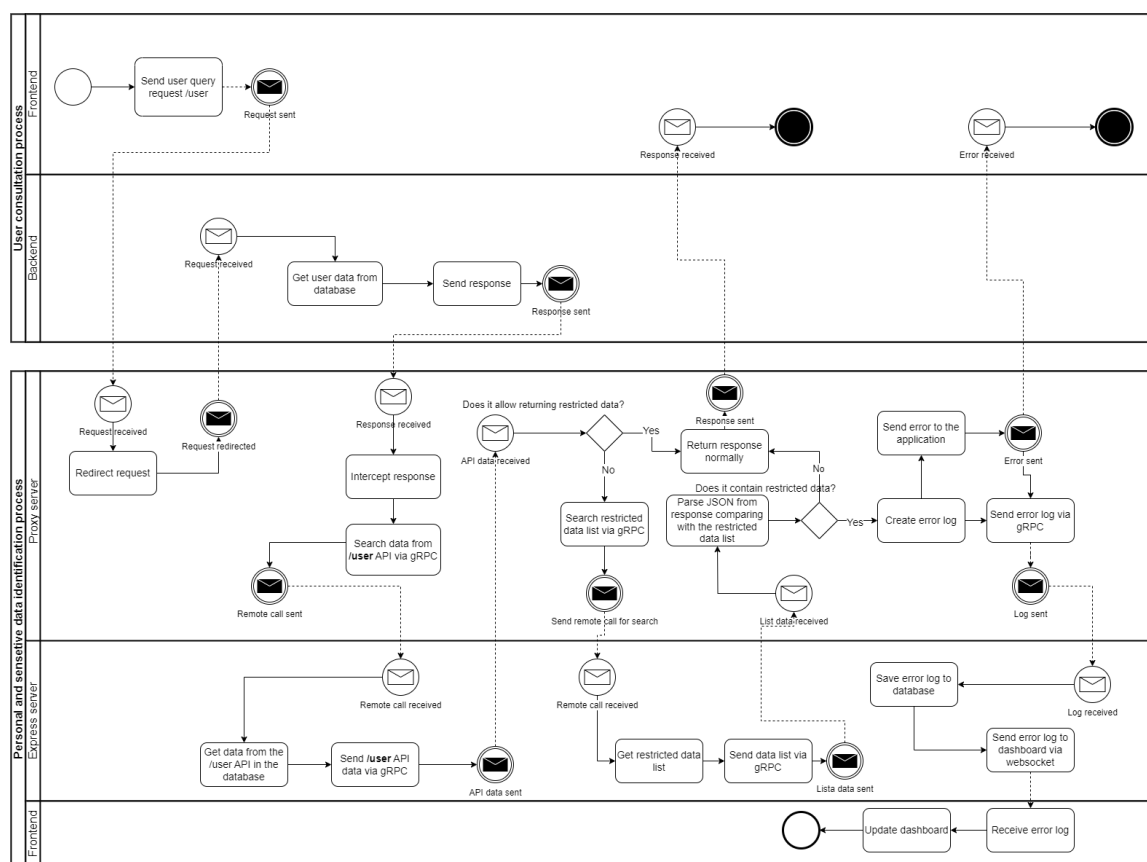


Figura 1: Response identification process from a user query API.

The BPMN diagram presents communications between the components in addition to the steps and interactions involved in analyzing and identifying sensitive data during the interception of the REST API response. It demonstrates how RESTGuardian operates in the process and presents the logic for identifying personal and sensitive data, following the restricted word list.

3.1.2. Web service

The RESTGuardian web service consists of two parts: the backend and the frontend. Both are implemented with TypeScript to provide an efficient and consistent development environment.

The backend was developed using the ExpressJS framework (version 4.18.1)⁹ due to its simplicity and ease in creating routes and handling HTTP (Hypertext Transfer Protocol) requests. Additionally, it includes a gRPC (Google Remote Procedure Call) server¹⁰ to handle requests made by the proxy server via Protobuf¹¹. For real-time communication with the frontend to display logs, websocket with the socket.io library¹² is used, allowing immediate detection of leaks in responses.

The frontend was built using the ReactJS library (version 18.2.0)¹³, combined with TypeScript. ReactJS is widely used for its ability to create interactive and reactive user interfaces. The use of TypeScript in the frontend also offers several benefits such as advanced autocomplete and type validation, which facilitates code development and maintenance.

The communication between servers in RESTGuardian is conducted using gRPC in conjunction with the Protobuf serialization format. This choice is made considering the performance and efficiency of communication between servers.

gRPC is an open-source framework developed by Google that facilitates communication between distributed services. It uses the HTTP/2 protocol as its underlying transport, providing fast, reliable, and low-latency communication. gRPC supports multiple programming languages, including TypeScript, which makes integrating the proxy server and the web service's backend efficient.

It is important to emphasize that gRPC and Protobuf provide an alternative to REST-based communication. While REST is widely used and suitable for many cases, gRPC excels in scenarios where server-to-server communication needs to be highly optimized and low latency.

3.2. Management of restricted APIs, users and data

The RESTGuardian provides comprehensive CRUD (Create, Read, Update, Delete) functionality for APIs, enabling the management and control of routes in the target application that will be read and analyzed by the proxy server. As new APIs are added to the target application, they need to be manually added to RESTGuardian, providing essential information such as the route name, the final route path (e.g., */user*), the request type (e.g., GET), and the definition of permission for returning confidential data. When adding a new route, the proxy server fulfills its role of intercepting the response. Additionally, RESTGuardian empowers administrators to adjust and update existing API configurations. This includes the ability to modify permissions for returning confidential data, edit route name, final path, and request type. This flexibility ensures precise control over API behavior and allows them to be tailored to the application's specific needs.

The query functionality allows visualization and analysis of existing API configurations and data. This is useful for verifying current route settings for proxy server operation. Finally, RESTGuardian also offers API removal functionality, allowing specific routes to be disabled or deleted if they are no longer needed or are obsolete.

Additionally, RESTGuardian offers user registration functionality. The system supports two types of users: administrator (*admin*) and regular user (*user*). When the application is executed, an administrator user is automatically created, who can manage other users and their permissions. The administrator user has additional privileges, allowing for adding, editing, and removing users from the system. This functionality enables precise control over who has access to error logs, API lists, and the list of restricted data. Thus, RESTGuardian provides an authentication and user management system, with a default administrator user to initiate the API management process, ensuring security and proper control over operations performed in the system.

⁹<https://expressjs.com/>

¹⁰<https://grpc.io/>

¹¹<https://protobuf.dev/>

¹²<https://socket.io/>

¹³<https://react.dev/>

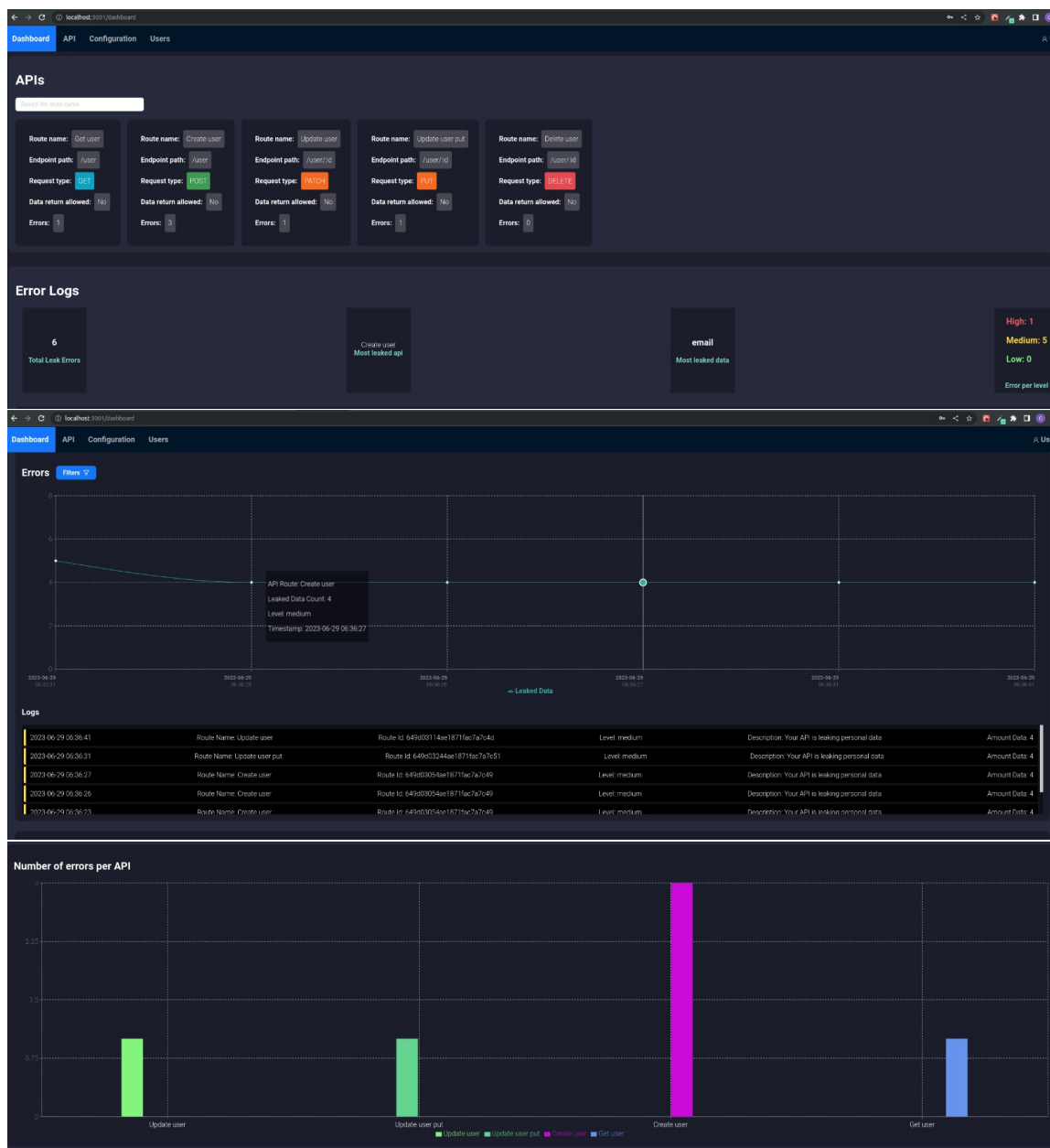


Figura 2: Dashboard after data leak in APIs.

With this, RESTGuardian allows the configuration of which data should be prohibited from being returned in API responses if the route has its permission to return personal and sensitive data disabled. By default, a list that includes personal and sensitive data is created, following the principles defined in LGPD.

Each item in the arrays will be compared with the keys in the JSON returned in an API response. However, the user has full flexibility to add, edit, and delete data, adapting the list according to their specific scenario. This functionality allows for granular control over which sensitive data should be blocked or filtered in API responses. By customizing the list of prohibited data, the user can ensure that confidential information, personal identification, addresses, phone numbers, or

any other data, are not exposed in API responses. By adapting the list of prohibited data, RESTGuardian offers an additional layer of security, ensuring that confidential data remains protected from accidental exposure.

RESTGuardian offers a comprehensive dashboard that displays a list of APIs, their relevant information, and the number of errors that occurred. The dashboard is a centralized tool that provides a comprehensive overview of the state of monitored APIs. A dashboard highlight is a real-time updated line chart, which displays the amount of leaked data in the APIs. This information is transmitted by the backend through websockets, allowing real-time leaks visualization.

Furthermore, the dashboard includes detailed logs that show the level of errors. A rating system is utilized, where the amount of leaked data is categorized as low (1 to 2 leaked data), medium (3 to 4 leaked data), and high (more than 5 leaked data). This classification helps quickly identify the severity of errors and take appropriate measures. To facilitate searching and analysis of logs, the dashboard features a filter that allows for customized searches. This enables finding specific APIs and facilitates the identification of data leaks based on the applied filter. Additionally, the dashboard presents a bar chart that compares APIs in terms of the number of returned errors. This visualization helps identify which APIs are experiencing a higher number of errors, allowing for effective prioritization in the troubleshooting process. Figure 2 presents the dashboard with data, logs, and a chart and Figure 3 shows the error returned for a request.

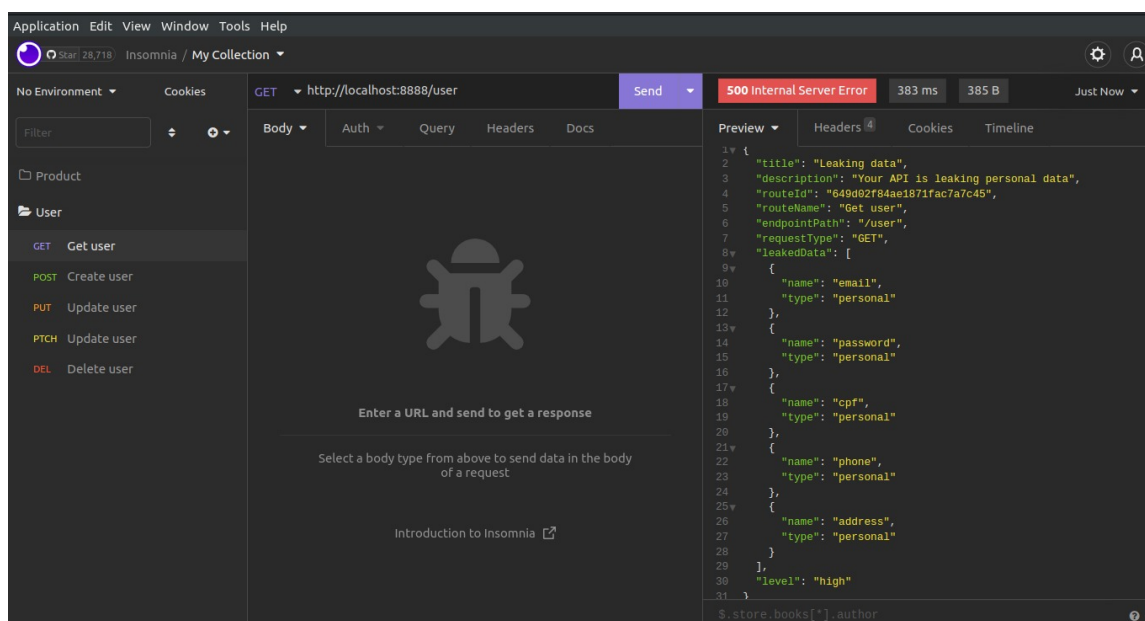


Figura 3: RESTGuardian configuration page.

Data Protection Officers can use RESTGuardian locally simply through Docker. In a scenario similar to the one presented earlier, it is possible to intercept responses from APIs hosted on cloud services such as AWS, Azure, Google Cloud, or others while RESTGuardian is running locally. To achieve this, it is necessary to provide the base URL of the APIs in the Docker compose. With this configuration, the DPO, along with developers, can configure RESTGuardian by registering the system's routes and evaluating which of them can or cannot return sensitive data.

As developers deliver the code, the DPO can check if any API is insecure in the deve-

lopment environment. By using the MongoDB¹⁴ cloud version, data is centralized in the cloud, ensuring that the dashboard is always up to date. With this setup, the DPO can extract insights and create reports from the data collected by RESTGuardian.

4. Conclusion

The pressure exerted by tight deadlines and the lack of security knowledge can lead to the creation of insecure code and excessive exposure of sensitive data. This study presented the proposal for a system called RESTGuardian. The system developed to control personal and sensitive data in REST API responses plays a crucial role in addressing the challenges faced in the context of secure development, data protection, and compliance with LGPD. It acts as security middleware, providing an additional layer of protection between the application's frontend and backend.

RESTGuardian provides the DPO with a comprehensive view of monitored APIs, allowing for the identification of potential data leaks and proactive corrective measures. This centralized, cloud-based solution enables effective analysis of API security and contributes to compliance with data protection regulations, ensuring the security and protection of personal and sensitive data in the development process.

Five important aspects can be considered for future work: i) data leakage detection in JWTs (JSON Web Tokens) returned in REST APIs; ii) addition of compatibility with other NoSQL databases; iii) deployment of RESTGuardian on cloud platforms; iv) automation of the inclusion of new APIs from a system into RESTGuardian; and v) creation of a use case demonstrating the use of RESTGuardian in a real-world environment of a startup.

Acknowledgment

The authors would like to acknowledge the infrastructure provided by the Federal University of Itajubá, the Federal University of Mato Grosso do Sul, and the Federal University of Bahia. The codes are available at <https://github.com/carloshssouza/data-controller-system>. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil, grant #403231/2023-0.

Referências

- Adetunji, O., Ajaegbu, C., Otuneme, N., and Omotosho, O. J. (2020). Dawning of progressive web applications (pwa): Edging out the pitfalls of traditional mobile development. *Am. Sci. Res. J. Eng. Technol. Sci.(ASRJETS)*, 68(1):85–99.
- Alenezi, M. and Almuairfi, S. (2020). Essential activities for secure software development. *Int. J. Softw. Eng. Appl*, 11(2):1–14.
- Amazon Web Services (2023). What are AWS WAF, AWS Shield, and AWS Firewall Manager? Available at: <https://docs.aws.amazon.com/waf/latest/developerguide/what-is-aws-waf.html#waf-intro>; Accessed in 26/02/2024.
- Bajer, M. (2017). Building an iot data hub with elasticsearch, logstash and kibana. In *2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, p. 63–68. IEEE.
- Bakken, D. (2001). Middleware. *Encyclopedia of Distributed Computing*, 11.

¹⁴<https://www.mongodb.com/>

- Brasil (2018). Law number 13.709, august 14th, 2018. *Diário Oficial da República Federativa do Brasil*. URL https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709compilado.htm.
- DEVYANIN, P. N., TELEZHIKOV, V. I., and KHOROSHILOV, A. V. (2021). Building a methodology for secure system software development on the example of operating systems. *Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS)*, 33(5): 25–40.
- DPO, K. (2018). General data protection regulation: The role of the future data protection officer (dpo). CNPQ - National Council for Scientific and Technological Development.
- Faiz, M. F., Arshad, J., Alazab, M., and Shalaginov, A. (2020). Predicting likelihood of legitimate data loss in email dlp. *Future Generation Computer Systems*, 110:744–757.
- Genfer, P. and Zdun, U. (2022). Avoiding excessive data exposure through microservice apis. In *European Conference on Software Architecture*, p. 3–18. Springer.
- Khan, R. A., Khan, S. U., Khan, H. U., and Ilyas, M. (2022). Systematic literature review on security risks and its practices in secure software development. *ieee Access*, 10:5456–5481.
- Nguyen, T. T., Backes, M., Marnau, N., and Stock, B. (2021). Share first, ask later (or never?) studying violations of {GDPR’s} explicit consent in android apps. In *30th USENIX Security Symposium (USENIX Security 21)*, p. 3667–3684.
- OWASP (2023). Owasp api security - top 10 — owasp. Available at: <https://owasp.org/www-project-api-security/>. Accessed in 26/02/2024.
- Pessoa, C. R., Nunes, B. C., de Oliveira, C., and Marques, M. E. (2021). Effects and projections of the brazilian general data protection law (lgpd) application and the role of the dpo. In *Handbook of Research on Digital Transformation and Challenges to Data Security and Privacy*, p. 195–208. IGI Global.
- Rajapakse, R. N., Zahedi, M., Babar, M. A., and Shen, H. (2022). Challenges and solutions when adopting devsecops: A systematic review. *Information and software technology*, 141:106700.
- Ramirez, A., Aiello, A., and Lincke, S. J. (2020). A survey and comparison of secure software development standards. In *2020 13th CMI Conference on Cybersecurity and Privacy (CMI)-Digital Transformation-Potentials and Challenges (51275)*, p. 1–6. IEEE.
- Shah, N., Willick, D., and Mago, V. (2022). A framework for social media data analytics using elasticsearch and kibana. *Wireless networks*, p. 1–9.
- Stallings, W. (2019). *Information privacy engineering and privacy by design: Understanding privacy threats, technology, and regulations based on standards and best practices*. Addison-Wesley Professional.
- Tahboub, R. and Saleh, Y. (2014). Data leakage/loss prevention systems (dlp). In *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, p. 1–6. IEEE.
- Trifonov, R., Nakov, O., Pavlova, G., Manolov, S., Tsochev, G., and Nakov, P. (2020). Analysis of the principles and criteria for secure software development. In *2020 28th National Conference with International Participation (TELECOM)*, p. 125–128. IEEE.