




Front End Application Security: Proposal for a New Approach

Renato Carauta Ribeiro^{1,2}, Edna Dias Canedo²^a, Bruno J. G. Praciano^{3,4}^b, Gabriel P. M. Pinheiro³,
Fábio Lúcio Lopes de Mendonça³ and Rafael T. de Sousa Jr.³^c

¹Computer Center, University of Brasília (UnB), Brasília - DF, Brazil

²Computer Science Department, University of Brasília (UnB), Brasília - DF, Brazil

³Cybersecurity INCT Unit 6, Decision Technologies Laboratory—LATITUDE, Electrical Engineering Department (ENE),
Technology College, University of Brasília (UnB), Brasília - DF, Brazil

⁴Department of Mechanical Engineering, University of Brasília (UnB), Brasília - DF, Brazil

Keywords: OAuth 2, Service Oriented Architecture (SOA), Angular 2, Security, TypeScript, Framework.


Abstract: The data processing center (CPD) of the University of Brasília (UnB) has the need of evolution of legacy systems and the communication between systems in an efficient and safe way. For this reason, it is needed to implement a centralized control system for authentication and authorization to access services, systems and information. The technologies used focus on what is most modern in the market. In this paper we will discuss the security of applications developed as part of the single page application (SPA) concept, focusing on security using the OAuth 2 framework, Angular front-end language and service-oriented architecture (SOA). It will show the development of a security module that turns security complexity into programming abstractions for the new client applications developed in the CPD. The security module developed by the UnB aims to centralize, modernize, and improve the security of University applications. The advantage of this module is its flexibility, abstraction concepts, centralization, and use of one of the standard security protocols used today, OAuth 2, which brings greater security to UnB applications.


1 INTRODUCTION


According to Sommerville (Sommerville, 2010), software development does not currently end where it is delivered, and continues to exist throughout its life cycle. Modifications in business, technological developments, or even errors encountered during the use of software can generate changes. An organization's systems must be easy to maintain and evolve quickly to keep up with new technologies and frequent changes in the business' rules of the organization. Software modernization is a topic under discussion at the UnB due to the need to modernize corporate systems developed by the CPD. Today, the scenario evidenced by the university is of diverse systems developed in VB, VB.Net, C#, PHP, ASP and Java that do not have proper communication, nor standardized security. Different architectures, models, and frameworks were used, which causes difficulty

in maintaining systems, especially in controlling improper access to information (Regateiro. et al., 2017; Óscar Mortágua Pereira. et al., 2017).

Systems rarely exist alone in a corporate environment; they coexist in a friendly manner, and exchange information securely. According to Cairns (Cairns Cade, 2017), security is often overlooked, and is considered a requirement of little importance and not given due value. For Cairns (Cairns Cade, 2017), security is an essential part of enterprise applications. According to Dolphine and Carnis (Dolphine Tiago; Carnis, 2017), over time, several proprietary solutions have been developed to solve the problem of security and information sharing between applications, such as Google AuthSub, Yahoo BB Auth, AOL Open Auth, FlickrAuth, among other solutions. Because of this scenario, the OAuth 2 framework was developed. This framework aims to standardize communication between applications, centralize server access control, and protect the organization's sensitive data and services. This framework emerges as a definitive solution to the security problem experienced at UnB.

^a  <https://orcid.org/0000-0002-2159-339X>

^b  <https://orcid.org/0000-0002-7423-6695>

^c  <https://orcid.org/0000-0003-1101-3029>

According to Hardt (Hardt, 2012), OAuth 2 is an authorization framework that enables applications to have restricted access to Hypertext Transfer Protocol services (HTTP) and Hypertext Transfer Protocol Secure (HTTPS) through identification of access tokens. It enables access control to services and resources safely and reliably. Several frameworks have been developed to provide communication between client and server quickly. One of the most famous is Angular, which is currently in version 4. The problem of a new technology is that it does not yet provide a standard solution for sharing information securely nor for access control.

The use of the Angular language (Fain and Moiseev, 2016), (Uluca, 2018), (Agilar et al., 2016) is due to it being a well-accepted framework community, with growing investment from Google — the owner of this framework —, which has been building several tools to support this technology. It is a modern solution that is easy to develop and maintain. It includes well-defined standards, integrated automated test control and coding standards in its architecture, so that you can easily define coding rules.

This work focuses on solving the problem encountered for unauthorized access using the OAuth 2 (Hardt, 2012) framework along with the Angular language. The purpose of this paper is to provide a solution for securely sharing data between client and server, when sensitive information passed by the server to client applications should be stored. This work does not address the security of server applications, only the client side. This solution was applied at UnB as a case study to centralize authorization control of client applications. The new client applications, developed in Angular, have a security module, which makes authentication centrally. This module aims to centralize security, standardize and control all communication involving the client and server-side. This solution also has the provide a centralized login access to all UnB systems.

2 BACKGROUND

Angular is a framework for client-side application development. Developed by the company Google, it is an innovative solution that has recently wholly changed its architecture. In the first version of Angular, currently called Angularjs, the JavaScript programming language is used. This language is a native language that every browser supports. The problem with using this language, which was solved in the second version of Angular, is that none of the features present in object orientation can be used, the commu-

nication between controllers, services and server applications becomes more verbose and complex (Fain and Moiseev, 2016).

In the current version of Angular, Google has changed its use of JavaScript programming for the TypeScript language. TypeScript is a language that shares resources with JavaScript. Unlike JavaScript, *"TypeScript takes advantage of your understanding of other object models in a unique way"* without losing the performance of the JavaScript language (Bierman et al., 2014). The screen creation language used is Hypertext Markup Language (HTML). For page styling, the Cascading Style Sheets (CSS) language is used (Freeman, 2019). According to Murray et al. (Murray et al., 2016), the Model View Controller (MVC) architecture has been used for many years. This architecture separates the application into the following layers: The template is where it contains the application domain logic; the view is where data is available; the controller links the view to the model. The architectural model of Angular applications adapts this pattern and is called Model View Whatever (MV *). In this architecture, the model, vision classes have the same function as the MVC model, but the controller is replaced by what the Google team calls *"whatever works for you"* (Azañón Cáceres, 2015).

According to Angular.io (Angular.IO., 2010), the Angular framework is a modular development solution, that is, it is divided into modules. They serve to group the functionalities of a component or an application. An application developed with Angular must have at least one module, called the root module. Components are classes that control templates (application screens). Components communicate with templates through event binding and property binding. Services are any class with a narrow and well-defined purpose. Components are the largest consumers of services. The instantiation of services through components happens through dependency injection. This technique consists of providing a new instance of a class without instantiating this class directly by the class that uses it, which decreases coupling (Angular.IO., 2010).

The communication that a service makes between the client application and the server application to request data is made by the Hypertext Transfer Protocol (HTTP) (Belshe et al., 2015), using the basic communication verbs, which are: GET for data consultation; POST for data entry; PUT for data update; DELETE for data deletion. The information transmitted or requested by the Angular application comes via payload. The Angular framework changes the traditional form of application development, with almost com-

plete separation between client and server. Unlike its predecessor Angularjs (Ramos et al., 2018), (Chansuwath and Senivongse, 2016), this new framework facilitates the development of client applications and improves performance.

2.1 Framework Oauth 2

According to Bihis (Bihis, 2015), the Oauth 2 framework is currently one of the essential service authorization protocols and is being widely used as a solution by various companies such as Google, Facebook, Spotify, among others. This framework is a solution to the problem of unauthorized access to information. The Oauth 2 framework defines four basic entities (Bihis, 2015): 1. Resource Owner: is the user who authorizes an application to access its data; 2. Client: The application that requests access to the resource requested by the user to the server; 3. Resource Server: Server that hosts protected resources and responds to requests made by the client with the access token; 4. Authorization Server: Server that grants the access token after user identification and client request to the protected service (de Sousa Ribeiro et al., 2018).

Hardt (Hardt, 2012) says it is mandatory to store the token on the client-side, but does not say where the token should be stored. This framework requires that HTTPS be used in place of the HTTP protocol. Hardt (Hardt, 2012) conceptualizes four different streams to obtain user authorization: Authorization Code; Implicit; Resource Owner Password Credentials; Client Credentials.

In Authorization Code, there is an intermediate server between the client application and the protected resource, called an authorization server. Once the user requests the client application to access a protected resource, the client application redirects the user to the authorization server. The client receives the authorization code after the authorization server authenticates the user's credentials. By this mechanism of redirection to the authorization server, no user credential is trafficked across the network, which ensures additional security over other models (Denniss et al., 2019).

Implicit is a simplified authorization code solution. In this model, instead of using an authorization code for each client application, a token is issued to the user, without the need to authenticate the client. It is a very simplified flow, has the advantage of reducing the number of requests between the client application and the authorization server, but it is a less secure solution that relies on client-side token storage security (Li et al., 2019).

In Resource Owner Password Credentials, the user gets the access token from the credentials. This type of direct authorization can be done when the client application has high server reliability. Only one request is made with user credentials passed to the authorization server, which returns the access token to the protected resources. From there, only the token is exchanged between the client and the resource server (de Sousa Ribeiro et al., 2018).

In Client Credentials, the user gets the access token from the credentials. This type of direct authorization can be done when the client application has high server reliability. Only one request is made with user credentials passed to the authorization server, which returns the access token to the protected resources. From there, only the token is exchanged between the client and the resource server (de Sousa Ribeiro et al., 2018).

2.2 RSA Protocol Public Key Encryption Concept

According to Stallings (Stallings, 2017), public-key cryptography provides a radical paradigm shift. This type of encryption is asymmetric; that is, it uses two separate keys, one for information encryption and the other for information decryption. This type of encryption arose to solve two problems, one of the distribution of keys and another of the integrity of the information. One of the main features of this type of encryption is that it is impossible to discover one of the keys from the other. This type of encryption serves to add secrecy to the information that will travel on the network, or will be stored on the client. A key pair, one public and one private, are required so that one is used to encrypt text and the other to decrypt.

The security of this technique lies in the fact that if the information is encrypted with one of the keys — for example, the public key —, it can only be decrypted by the other key. One of the keys — usually the private one — must be kept confidential, hence it should never be shared. The public key can generally be shared over the network (Stallings, 2017).

In this type of solution, the public key must be shared on the network. The sender (who is sending the information) encrypts the clear text with the recipient's public key (which is the one who receives the information) producing an encrypted text. Upon receiving the message, the recipient decrypts the information and gets the clear text again. This type of solution ensures that the information is complete and has not been modified by a hacker.

According to Stallings (Stallings, 2017), RSA is a public key algorithm that uses the block technique

to encrypt and decrypt clear text. All the security involved in RSA lies in raising an integer to a high power. RSA is the most widely accepted and implemented general-purpose public-key encryption technique for the security of sensitive information. The reason that RSA is a secure and widely used protocol is due to its use to generate giant exponential keys, making it difficult to factor this number, taking years to discover and decrypt.

3 RELATED WORKS

Korva et al. (Korva et al., 2016) shows you step by step how to start and develop a simple Angular application, focusing on new developers in the area. It shows the basics about Angular, its architecture, the primary languages used: TypeScript, HTML, and CSS. It is mentioned that Angular supports authentication and security, but it does not address the problem of securely storing sensitive information on the client, nor the secure communication of information.

Kafle and Lindsey (Kafle and Lindsey,) show how important it is to be secure with information transfer using the REST API. The security framework used to restrict unauthorized access is Oauth 2. In this paper, the author focuses on security in two aspects: the first is the use of authentication and authorization; The second is the use of HTTPS / SSL protocol for information exchange. However, storing sensitive data on the client application side is not the focus of this work; only security provided for data communication using the REST protocol. Trnka and Cerny (Trnka and Cerny, 2018), in their work, talk about the growing use of the internet and the sharing of information over the network, the so-called Internet of Things. A solution for managing shared information securely is proposed using the Oauth 2.0 framework as an authorization center. This work is used to encrypt the information in the JSON Web Token (JWT) client, which is currently not a secure way to store sensitive information of the client.

Based on an internet search, it has been proven to be insecure and easy to decrypt encryption. There are several sites¹ that can decrypt encrypted information with JWT-generated token. In his doctoral dissertation, Thomas (Fielding and Taylor, 2000), creator of the REST communication model, talks about how this new communication protocol works, which uses a lighter and more straightforward approach. It is emphasized in the thesis that we need to take care of security at all levels, from infrastructure to appli-

¹<http://calebb.net/>

cations. It revolutionizes by creating a protocol that is lighter than its previous one, the SOAP protocol (Eggen et al., 2004). But his work does not go deep into the security of an application through the REST protocol, using JSON.

Chen et al. (Chen et al., 2016) argued about the thousands of connected devices, which today are called the Internet of Things (IoT), and how Service Oriented Architecture (SOA) can provide interoperability between these devices. The article proposes a reliable, adaptable, and scalable solution to support SOA-based IoT system service composition applications. A technique based on distributed collaborative filtering has been developed for selected feedback using friendliness similarity rating, social contact, and filter-related communities of interest. Calculation is made to verify the best confidence to minimize convergence time and to select the best route by eliminating malicious nodes.

Krylovskiy et al. (Krylovskiy et al., 2015), reported the difficulty of integrating the various IoT platforms present in the world today is mentioned. Systems must be able to scale and evolve, embracing new technologies and changing business rules. This work uses the microservices technique to design an intelligent city IoT platform. The work demonstrated significant benefits provided by the microservice architectural style compared to the more traditional service-oriented architecture (SOA). Architectural standardization and application independence using microservices allowed the project components to be developed independently, even of the programming language, connected only through a standardized interface.

Chen et al. (Paillier, 1999) investigated a new computational problem, called the Composite Residuality Problem, and its applications for public-key cryptography. A new mechanism is derived from three encryption techniques: one permutation scheme and two computationally similar probabilistic encryption schemes to RSA. The proposed model to solve the researched problem was proven to be computationally safe. The article (Somani et al., 2010) addresses the increased utilization of the cloud computing solution and the problems and security and proper implementation of the cloud across the network. This research evaluates cloud storage methodology and cloud data security by implementing a digital signature with the RSA algorithm. The feasibility of cloud data storage using the RSA algorithm to ensure data security is proven.

Chen et al. (Chen et al., 2014) mentioned the industry's steadfast adherence to the OAuth standard, which has now been redesigned in version 2. The

study focuses on using the OAuth protocol for mobile application development. A study of the OAuth protocol is done, demonstrating its difficulties to be deployed in mobile applications, and a field study of over 600 popular mobile applications to verify how authentication and authorization is done in practice. The result of this study shows that of the 149 applications that use OAuth, 89 of them (59.7 %) were implemented incorrectly and therefore vulnerable. Soman et al. (Wang et al., 2009) article showed the need for an SOA system to integrate its data securely, reliably, and as it is rooted in the system. To solve this problem, a dynamic SOA-based data integration model is proposed. The model uses service-oriented architecture for data integration and sharing between heterogeneous systems, so integration between various systems can be easily facilitated.

These are some works that show how important it is to have light and secure communication for communicating information over the internet (Vijayakumar and Chokkalingam, 2019), (Huang et al., 2019). Application security is still one of the critical challenges facing us today. A secure system goes far beyond secure software or robust infrastructure. Unlike all the work presented above, this work focuses on the creation of a security module, developed in Angular language, to control access to restricted resources. It focuses on how to start the authentication and authorization flow, how to centralize request headers, and how to securely store the user's authorization token. This module uses abstractions from the front-end application for the developer's concern with security control in each request.

4 CONTEXTUALIZATION

At the University of Brasília (UnB), there are several systems developed on numerous platforms and languages, which makes communication and maintenance of the systems difficult. All of these systems have a single access control system called an Access Control System (ACS). This system controls who has access to UnB applications. Access control is done through access profiles, where each user has several profiles. One of the major problems UnB faces is the difficulty experienced by servers, students, and teachers using systems that are obsolete or do not reflect UnB's real business rules. A serious recurring problem is system slowdowns because all application processing is centralized on servers. One solution under development is the modernization of systems for an easy-to-maintain platform, the creation of various services, and their consumption by client applications.

Client applications take much of the server processing and pass it on to the client.

A problem with this new architecture is the need for an integrated authentication and authorization control, which would facilitate user access and maintain information integrity. There is also a need to communicate with systems outside the university, which need automatic and secure access to UnB information. Information is routed through the network in the clear, which makes it easier for an attacker to misuse this information. The cookie which controls the user's session can easily be captured by an attacker and used on another machine.

Current systems do not have a session time control on the client, that is, the user logging into the system has a session time to do all their activities, automatically logs out afterwards, being necessary to log in again. There is no way for applications to restart the session after the time has elapsed without the user having to log in to the system again. The login base, systems and authentication databases are fragmented and not well integrated. In addition to systems that are built on outdated technology, other systems still maintained by UnB, were developed in the 1980s with VB and VB.Net technology. There is no separation of layers, namely, the rules and business are coupled, in most cases, in the vision of the applications.

Currently, a new application creation paradigm is being introduced in the University of Brasília's CPD, using a service creation paradigm rather than monolithic applications. Server-side REST services (Barbaglia et al., 2017), controlled by a service bus, and client applications that call these services, in a wholly decoupled manner, have begun to be introduced. Both old systems and new services developed suffer from the problem of not having a single and centralized authentication. For old services, an internal LDAP solution for database integration (Wegner et al., 2009) was developed. However, for new services and architecture being developed by CPD, it is necessary to develop new forms of service authentication and authorization in a secure and centralized way in a single authorization server, which in the case of UnB would be the service bus. Services as a means to authenticate and authorize the user.

5 PROPOSED MODEL

Several issues are mentioned in the current scenario at UnB. This work aims to address the security and information storage side of client applications by storing sensitive information securely by securely exchanging client-server information and sharing infor-

mation between client applications securely. New applications developed in Angular must have a module that abstracts and controls request header data, storage, and access token sharing.

A security module for Angular applications is under development. This module aims to solve most of the problems of authentication, storing sensitive information on the client application side, and sharing this information securely. Using a separate module to take care of client application security facilitates development and removes from the developer the need to worry about storing and sharing sensitive data between client and server. When accessing an application, the security module redirects the user to an authentication screen with the client id. If the client id has already been authenticated over the same IP connection, the server checks and returns the user's access token. If the client has not been authenticated, the system redirects to a central authentication screen, where the user enters his login and password, and if valid, the authorization server authenticates the client and returns the client access token. The client sends the client access token and retrieves the user access token, which will be used to access the restricted contents. Figure 1 shows how this flow is made.

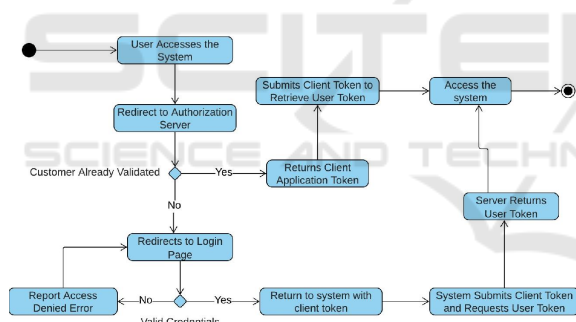


Figure 1: Basic Authentication Flow.

This flow shows a user's sequence of steps to gain access to systems. The strong advantage of this approach is that user credentials do not travel across the network. Since the authentication screen is on the server and not on the client, namely, it is a direct call to the server, it ensures security for user authentication.

Another important mechanism for authentication is the use of CAPTCHA, which prevent automated brute force attacks on the server. Another essential feature used is the control of communication between client and server through the user's IP, which is passed in the request header when the user authenticates. After some unsuccessful user access attempts, the authentication system blocks further access attempts by this IP for a specified time. After user authentication,

the security module stores the access token sent by the authentication server.

The access token is sent encrypted by the RSA public-key encryption algorithm and transformed into base64. The security module retrieves the token that came in the authentication server response header and stores this encrypted token in an instance variable, which is stored in the Authentication Service class. This ensures complete token storage security on the client application side. Figure 2 shows how the restricted services access token storage stream works.

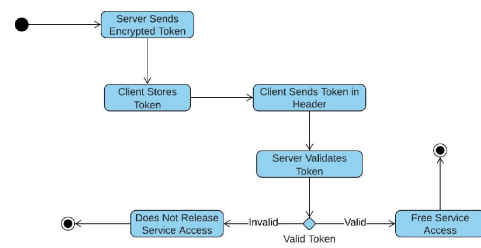


Figure 2: Restricted Services Access Token Storage Flow.

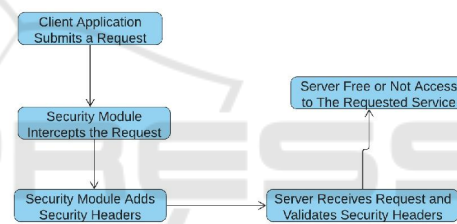


Figure 3: Headers Addition Flow in a Request.

Another essential feature that should be used to strengthen security between the client application and the server application is the use of Hypertext Transfer Protocol Secure (HTTPS). Hardt (Hardt, 2012) says that using HTTPS is required to implement the OAuth 2 framework. The goal of using HTTPS is to provide extra security as it encrypts all information with the public key signing method. This encryption is done through digital certificate, which ensures that the message sent by the client is authentic.

One feature of the security module, which was mentioned above, is the automatic implementation of the HTTPS request headers. The importance of this is beyond the abstraction for the developer, who does not need to implement the headers required for services authorization in each request. It also centralizes the implementation of the headers of all requests between client and server, in a single module, making it easy to include or exclude any header without having to modify the code of the application that uses this module.

Figure 3 shows how the flow from a client application using the module and security works, sending

a request to the server to validate the access control headers added by the security module. It is essential that the server can validate access to services requested by the client application and enable access or return an access denied message. This section introduces the importance of using the security module in front-end applications to authenticate the user along with the server, store information securely and encrypted, provide secure communication with HTTPS, abstract and centralize authentication and authorization functionality.

6 THREATS AND VALIDITY

During this work, we sought to select the primary references related to each of the topics covered in this research. However, there is no guarantee that all relevant works were selected during the literature review. Some relevant articles may not have been selected during the process. To try to minimize this threat, we researched the RFC on OAuth 2 and the key references on Angular language. Another impediment to the implementation of the new security solution using the OAuth 2 protocol is the absence of this type of solution for Angular technology. What has been done to minimize the potential impediment is to test and implement the solution in parts and verify its effectiveness.

Another problem encountered is that there are several authentication databases, causing user authentication to take a long time. An integrated memory base was developed for direct consultation of the security system created in this article. Another threat was found in the sponsorship of this new solution by UnB managers. To minimize this problem, a pilot system was developed for managers to validate the development, demonstrating the effectiveness of the security system deployed in the new architecture.

There are other systems that UnB wants to deploy to replace legacy systems. As a result, the need to develop new systems would be reduced, along with the importance of the security system specified in this article. To mitigate this threat, a platform has been developed to adapt to any technology that uses the OAuth 2 protocol and is easily integrated with other external systems. With future evolution and new versions of Angular technology, and with the advent of new languages and technologies, the developed system may become obsolete. One solution to this threat is to specify a well-developed architecture for the security system. It should be focused on implementing the OAuth 2 protocol so that adaptations, evolution, and possible technology changes can be natural. The

security system must always be on the latest version of Angular technology.

7 CONCLUSIONS

This work presents a security model for front end applications developed in Angular. It was presented the importance of using security not only on the server, but also on client applications, which currently meet most of the rules and business. The proposed model is being developed by the University of Brasília (UnB) to improve the security of new applications. The advantage of this model is the flexibility, security, abstraction, and centralization of responsibilities that such a model provides. The proposal presented does not focus on the security of client applications, but only mentions some essential steps that a server application must take to integrate with the client securely. For future work, it is suggested that a server-side framework is implemented to integrate with the proposed model. It would be interesting to implement other security techniques besides using OAuth 2.

Another factor that was not addressed in this article was the performance of an Angular application compared to other frameworks in the market. A comparative approach to whether Angular is the fastest, most flexible, or best performing, maintainable, among other essential features, would be interesting. With the ever faster advancement of technology and the extreme need to modernize UnB systems, this solution is also a viable proposition for several other agencies or companies facing security issues, which is a crucial part of any application.

ACKNOWLEDGMENTS

This work was supported by the Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES), Grants 23038.007604/2014-69 FORTE and 88887.144009/2017-00 PROBRAL; the Brazilian National Council for Scientific and Technological Development (CNPq), Grants 303343/2017-6, 312180/2019-5 PQ-2, BRICS2017-591 LargeWiN, and 465741/2014-2 INCT on Cybersecurity; The Brazilian Federal District Research Support Foundation (FAP-DF), Grants 0193.001366/2016 UIoT and 0193.001365/2016 SSDDC; the LATITUDE/UnB Laboratory (Grant 23106.099441/2016-43 SDN); the Ministry of Economy (TEDs DIPLA 005/2016 and ENAP 083/2016); the Institutional Security Office of the Presidency of the Republic (TED ABIN

002/2017); the Administrative Council for Economic Defense (TED CADE 08700.000047/2019-14); and the Federal Attorney General (TED AGU 697.935/2019).

REFERENCES

- Agilar, E., Almeida, R., and Canedo, E. (2016). A systematic mapping study on legacy system modernization. In *SEKE*, pages 345–350. KSI Research Inc. and Knowledge Systems Institute Graduate School.
- Angular.IO. (2010). Architecture overview. template syntax.
- Azañón Cáceres, C. A. (2015). *Desarrollo de la aplicación móvil multiplataforma ToCook*. PhD thesis, ETSI.Sistemas.Infor.
- Barbaglia, G., Murzilli, S., and Cudini, S. (2017). Definition of REST web services with JSON schema. *Softw., Pract. Exper.*, 47(6):907–920.
- Belshe, M., Peon, R., and Thomson, M. (2015). Hypertext transfer protocol version 2 (HTTP/2). *RFC*, 7540:1–96.
- Bierman, G. M., Abadi, M., and Torgersen, M. (2014). Understanding typescript. In *ECOOP*, volume 8586 of *Lecture Notes in Computer Science*, pages 257–281. Springer.
- Bihis, C. (2015). *Mastering OAuth 2.0*. Packt Publishing Ltd.
- Cairns Cade, S. D. (2017). The basics of web application security.
- Chansuwath, W. and Senivongse, T. (2016). A model-driven development of web applications using angularjs framework. In *ICIS*, pages 1–6. IEEE Computer Society.
- Chen, E. Y., Pei, Y., Chen, S., Tian, Y., Kotcher, R., and Tague, P. (2014). Oauth demystified for mobile application developers. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14*, page 892–903, New York, NY, USA. Association for Computing Machinery.
- Chen, I., Guo, J., and Bao, F. (2016). Trust management for soa-based iot and its application to service composition. *IEEE Transactions on Services Computing*, 9(3):482–495.
- de Sousa Ribeiro, A., Canedo, E. D., and de Freitas, S. A. A. (2018). An implementation of the oauth 2.0 for an enterprise service bus. In *ICCSA (1)*, volume 10960 of *Lecture Notes in Computer Science*, pages 469–484. Springer.
- Denniss, W., Bradley, J., Jones, M. B., and Tschofenig, H. (2019). Oauth 2.0 device authorization grant. *RFC*, 8628:1–21.
- Dolphine Tiago; Carnis, C. (2017). The basics of web application security.
- Eggen, R., Ahuja, S. P., Elliott, P., and Eggen, M. (2004). Efficiency considerations between common web applications using the soap protocol. In *Communications, Internet, and Information Technology*, pages 461–465. IASTED/ACTA Press.
- Fain, Y. and Moiseev, A. (2016). *Angular 2 Development with TypeScript*. Manning Publications Co.
- Fielding, R. T. and Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Doctoral dissertation.
- Freeman, A. (2019). Understanding typescript. In *Essential TypeScript*, pages 35–40. Springer.
- Hardt, D. (2012). The oauth 2.0 authorization framework.
- Huang, C., Chen, H., Tzeng, Y., and Li, P. (2019). Adaptive and service-oriented embedded system for information security applications. *Computers & Electrical Engineering*, 73:145–154.
- Kafle, A. and Lindsey, A. Security in the representational state transfer api.
- Korva, J. et al. (2016). Developing a web application with angular 2: Graphical editor for happywise’s cove trainer.
- Krylovskiy, A., Jahn, M., and Patti, E. (2015). Designing a smart city internet of things platform with microservice architecture. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 25–30.
- Li, W., Mitchell, C. J., and Chen, T. (2019). Oauthguard: Protecting user security and privacy with oauth 2.0 and openid connect. In *SSR*, pages 35–44. ACM.
- Murray, N., Lerner, A., Coury, F., and Taborda, C. (2016). *ng-book 2: The complete book on angular 2 (volume 2)*.
- Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In Stern, J., editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Ramos, M., Valente, M. T., and Terra, R. (2018). Angularjs performance: A survey study. *IEEE Software*, 35(2):72–79.
- Regateiro., D. D., Óscar Mortágua Pereira., and Aguiar., R. L. (2017). Spdc: Secure proxied database connectivity. In *Proceedings of the 6th International Conference on Data Science, Technology and Applications - Volume 1: DATA*, pages 56–66. INSTICC, SciTePress.
- Óscar Mortágua Pereira., Semenski., V., Regateiro., D. D., and Aguiar., R. L. (2017). The xacml standard - addressing architectural and security aspects. In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS*, pages 189–197. INSTICC, SciTePress.
- Somani, U., Lakhani, K., and Mundra, M. (2010). Implementing digital signature with rsa encryption algorithm to enhance the data security of cloud in cloud computing. In *2010 First International Conference On Parallel, Distributed and Grid Computing (PDGC 2010)*, pages 211–216.
- Sommerville, I. (2010). *Software Engineering*. Addison-Wesley, Harlow, England, 9 edition.
- Stallings, W. (2017). *Cryptography and network security: principles and practice*. Pearson Upper Saddle River.

- Trnka, M. and Cerny, T. (2018). Authentication and authorization rules sharing for internet of things. *Software Networking*, 2018(1):35–52.
- Uluca, D. (2018). *Angular 6 for Enterprise-Ready Web Applications: Deliver production-ready and cloud-scale Angular web apps*. Packt Publishing Ltd.
- Vijayakumar, K. and Chokkalingam, A. (2019). Continuous security assessment of cloud based applications using distributed hashing algorithm in SDLC. *Cluster Computing*, 22(5):10789–10800.
- Wang, J., Yu, A., Zhang, X., and Qu, L. (2009). A dynamic data integration model based on soa. In *2009 ISECS International Colloquium on Computing, Communication, Control, and Management*, volume 2, pages 196–199.
- Wegner, T., Uptmoor, F., and Kemper, J. (2009). Using LDAP as a management solution for distributed osgi. In *ICDS*, pages 143–148. IEEE Computer Society.

