ICALIA LABS

Security Policies

Abstract

Security has become one of the most valuable application concerns for online access. From data privacy, secured communication channels, continuous integration, and more, to customer-trust and developer's confidence, all are dependencies on how secure an online application is. Access from an uncountable number of devices is a reality, and we strive to provide a foolproof security system.

This document outlines some of the security measures we take in consideration while building an online application, starting from basic core design principles, so we can move into different techniques and more advanced practices, that will ensure a high armored security system. It is important to say that most of these techniques are based on documentation and best practices out there, and depending on the application to build, we may adapt to provide the best experience regarding security.

Index

Online applications

Nowadays most of the applications we have to work on are performing financial transactions, storing sensitive data on the databases, integrating multiple providers, and even connecting to legacy applications through a service layer. Hence security is a main software attribute for the whole organization, as being online opens a number of threats we need to be prepared for, such as session hijacking, password cracking, man-in-the-middle which affects directly into user trust, and therefore revenue.

“We are always improving our practices, to continuously review our software code for security holes, through our streamlined automated process.”

As the applications grow, they require more sensitive information from the users, security numbers, financial data, government-related information and so much more in order to provide a better and integrated experience, and even user-specific features, such as customized dashboards, reports and in more sophisticated applications the growing trend of Artificial Intelligence (AI). We are committed to this by always improving our practices, to continuously review our software code for security holes, through our streamlined automated process.

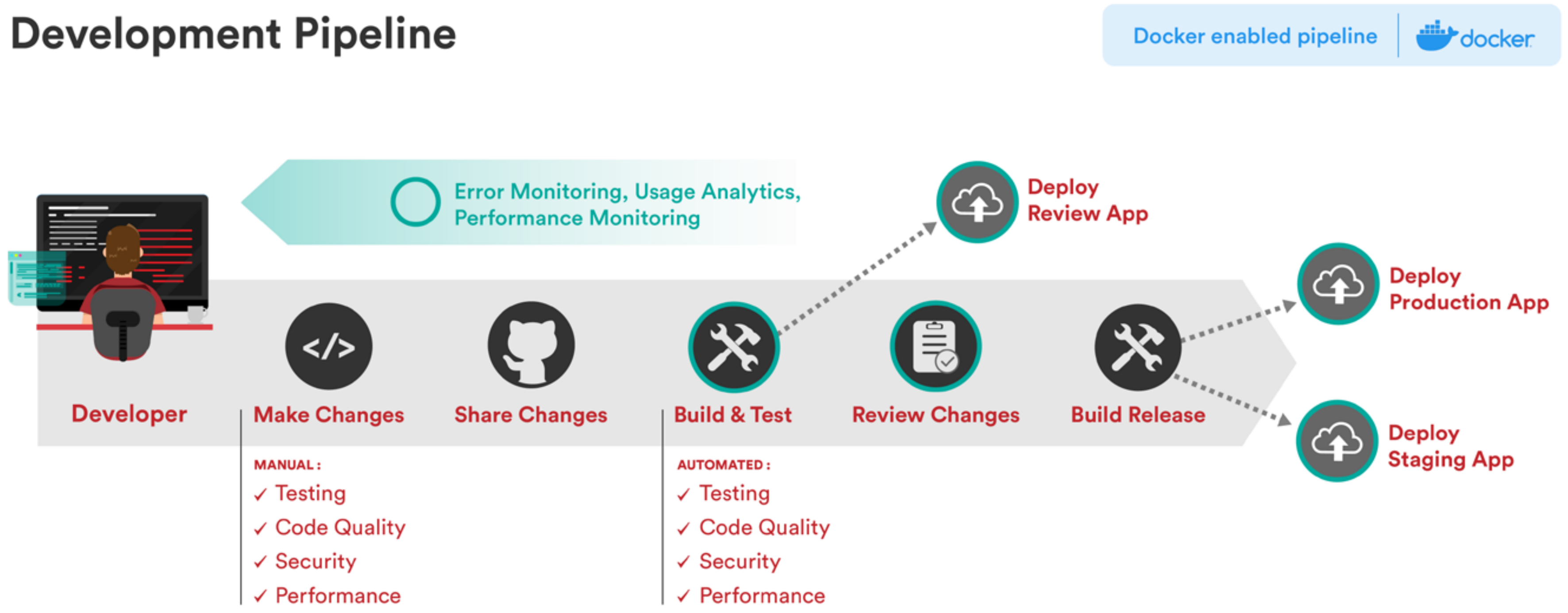
Core security concerns

All of the applications developed at Icalia Labs have a set of key security concerns ensuring a basic but strong secure layer for the online applications whether it is a startup or an enterprise.

We also consider that alongside *The Twelve-Factor App* practices, Test-Driven Development and the usage of Docker through our whole pipeline enables the teams to deliver features, hot-fixes, and bugs on-demand and with confi-dence, which impacts directly on the end-user experience and therefore the business outcomes.
(See Development Pipeline below)

Key security included:

- ✓ Password encryption
- ✓ Secure connections with Secure Socket Layers (SSL)
- ✓ Role-based access
- ✓ Automated security checks and patches on dependencies
- ✓ SQL injection prevention
- ✓ Cross-site scripting (XSS) protection
- ✓ Logs data obfuscation for sensitive data
- ✓ Usage of environment variables for access.



Key security concerns and how we tackle them.

Based on our experience, we have crafted a list of the most common and core security vulnerabilities every online application should address, and how we are tackling them.

Man-in-the-middle-attacks

Under this type of attack, the hacker intercepts the traffic between the browser and the application, in other words retrieving information from the user. To prevent this it is necessary to secure the communication channel by using a secure transport layer such as HTTPS via SSL. This also applies to mobile applications and communication with third-party services. By using this approach we ensure the safe data exchange by encrypting during transit and decrypting on arrival on any end.

Session hijacking

This vulnerability allows the attacker to act as any user with an open session by stealing the cookie that applications commonly use to recover the current session. In order to prevent this concern, we encrypt all of the cookies by using HTTPS, disabling any attacker from getting the plain value of them.

Cross-site request forgery

The attacker can build a script to make the application think the request is coming from a legit source. For example, the attacker can gain access to parameters or massively submit forms and perform actions on the system. To prevent this, we include on every page and form loaded, a security token that is validated on every request and by the server itself.

Password encryption

In some cases, we have encountered applications that save the user passwords without encryption, which enables the attacker to know all the access from every user and act like them. At Icalia Labs we encrypt passwords using the bcrypt algorithm, which creates an obfuscated string from the input password making it irreversible and not understandable by humans. We talk more about this further on the document.

Dependencies automated security checks

Every modern application is built on top of libraries, plugins, or frameworks which enhance the development experience, but as the security attacks grow, some of the dependencies may outdate really fast, whether for performance patches, but most importantly security concerns. To deal with this, we have bots that perform automated dependencies checks on every project keeping the code healthy and safe.

Injection attacks

This attacking technique happens when untrusted or unsanitized data is sent to an interpreter to work as part of a query or command. The most common injection attacks are SQL, NoSQL, OS, and LDAP injection. In our case and due to the nature of most of the applications we build, we focus only on the first and third one, which can be mitigated by:

1. Sanitize every input that comes from forms before passing it to the SQL interpreter.
2. Encode or remove SQL reserved characters.
3. Run automated and continuous checks to solve this while developing.
4. Usage of object-relational mapping (ORM) tools, like the one included in most of the modern frameworks.

Docker enabled pipeline

Since 2016 we have adopted Docker as a platform to build our whole development process, in order to enable each team to deliver software with confidence and securely. By implementing Docker within the core of our development process:

1. We enforce developers to care about the code they are building, but also the delivery of it.

2. We can easily migrate from one cloud to another, in case of an attack or emergency from an outrage

3. Homogenizes the development environment, for engineers and designers.

4. Allow us to have a zero downtime during deployment, and easily rollback in case of emergency, while updating the whole infrastructure.

Logging & monitoring

The exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident.

Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected. The most common concerns are:

- Auditable events, such as logins, failed logins, and high-value transactions are not logged.

- Warnings and errors generate no, inadequate, or unclear log messages.

- Logs of applications and APIs are not monitored for suspicious activity.

- Logs are only stored locally.

In order to prevent this, we at Icalia Labs:

- Integrate *New Relic* to track events on every application, primary for performance issues.

- Integrate *Sentry* for error monitoring, so the whole team is always aware if something happened after performing a deployment.

- Centralize all the logs with *DataDog* for later usage, analysis or contingency measures

- Filter sensitive information from the logs, such as passwords, credit card numbers, social security numbers and more.

- Implementation of *Castle* to monitor the users activity on the site, to detect suspicious access for example.

Key considerations for every application

When building a web-based application, we take into account several considerations that do not directly impact the final product, but they certainly help our engineers to better design applications for the long run.

Unit and system automated testing are the core of every application we build at Icalia Labs. This enables the team to integrate code with the confidence no previous functionality has been compromised with the integration of new code. **We strive for 70% of test coverage.**

Set up the infrastructure to deliver new features **on-demand**, by integrating a continuous integration and delivery approach. Tools such as Semaphore CI allow us to manage multiple environments, workflows, and delivery of the code, this way we keep a seamless deployment for every application.

We take code quality very seriously at Icalia Labs, and that is why we have dedicated engineers in charge of performing pair programming with team members as well as continuously review the code we are delivering, to ensure we have another pair of eyes watching for the code health.

As open source contributors and consumers, we love open standards-based technology and integrations, this includes using standards using HTML, CSS, but also linters for Ruby, Javascript, just to name a few.

Automated checks being done by bots, for instance:

- Static code analysis to reduce technical debt
- Security checks on application dependencies
- SQL injection vulnerabilities checks
- SQL performance concerns, such as N+1 problem

A layered architecture using MVC. This pattern provides a clear separation between the components that conform the application, losing coupling, separation of concerns, which helps to maintain the source code in the future.

An automated process to provide new features. We have a bot that helps the development team to build or scaffold recurring functionalities, enabling them to focus on the product core, but also will profoundly impact on the developer productivity, happiness, time-to-market, and quality increase.

Performance is tackled since day one of development as we provide lean Docker images and continuous checks to provide immediate feedback for the developer to take action. Depending on the requirements for the application being built, we can take some other actions for this concern, such as caching, data structures, and so on.

Scalability is something we build with time, in other words, we start as lean as possible, and from there we scale the platform with many different techniques, such as serverless, clustering, caching, load balancing scheduling, background jobs, etc. This approach helps the client to pay no more than the infrastructure needed.

User management on applications

We understand that managing users and authorization is always a concern for our clients, so that is why we dedicate the next section only to provide a comprehensive guide on how we achieve this for every single application.

Authentication

Authentication is merely allowing the user to access an application based on their username/email and password. But a lot happens on this process, and we do not just validate for access credentials.

Authorization

For every application, we establish access policies for each action on every controller, for every user role, but also where necessary we provide fine-grained access control to other resources, such as view components, methods, and pages.

- We have a centralized access manager that out of the box blacklists all the application resources for every role, this way if a new role is added we enforce all the policies and rules

- In some cases we also define authorization from the routes dispatcher, to provide an extra layer of resource access from the applications, an API endpoint namespace, the back office module, and authenticated users routes.

- Access to third-party services, such as the database, email provider, or any other is handled by the application using environment variables, and the users don't have access to them.

Password Policy

We will guide you into how encryption works, and how we implement it at Icalia Labs to protect and ensure passwords are properly obfuscated from the naked eyes. SHA and MD5 are some of the most common algorithms to encrypt passwords and save them to the database. It is important to recognize that for now, it is impossible to decrypt the encrypted passwords from these algorithms, but there are always ways for attackers to exploit them. The most simple one is the brute force attack, which is based on a trial-error method to decode the encrypted data. *Rainbow tables* is a recurrent tool for attackers to perform an automated match and gain access.

Even though this vulnerability seems simple to crack, by creating strong password policies for your site users, it becomes harder for hackers to crack encrypted credentials. Some measures to increase the score for securing passwords are:

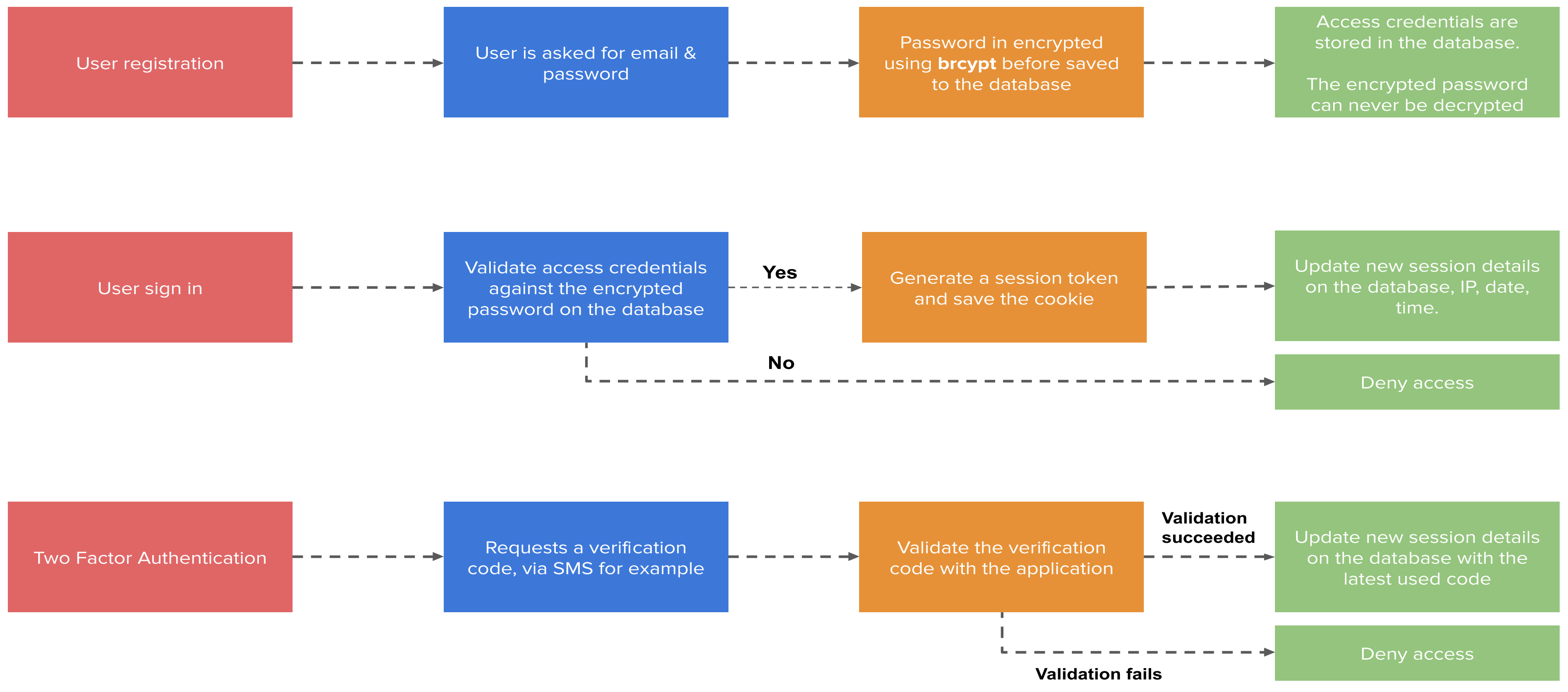
- Usage of special characters such as !,@,?,&
- Minimum of 8 characters
- At least one capital letter
- Usage of at least 1 number

Depending on the security concern for our clients, we build combinations from these options to meet the desired security level.

At Icalia Labs we salt passwords using **bcrypt** as mentioned earlier which encodes the data by generating different encryption values from the same password input in this case. For more information on how this works, you can check this *link*.

Two-factor authentication is playing a primary role in providing users an additional layer of security to guarantee access to almost any modern application, in some cases this feature is mandatory to perform some data sensitive data transactions. This security concern can be implemented by several mechanisms, such as:

- **OTP (One Time Password) -** this is one of the most common ways out there. The application server requests an auto-generated password (commonly digits) to an SMS gateway to send the OTP to the user's device. Once the password is input, it is validated by the application, and only if the OTP matches, the user is allowed into the system.
- **Security questions -** In some cases, some applications ask several security questions that only the user can answer, so once they are answered correctly and validated, the user is allowed into the system.
- **Authenticator app -** a more modern approach would be to use an authenticator app, which continuously generates new tokens to grant access to the application, so when the user is prompt to provide a new token, he/she just needs to open the application and type in the current token. The token is validated and the user is allowed to perform the action.


User registration
User is asked for email & password
Password in encrypted using brcypt before saved to the database
Access credentials are stored in the database.
The encrypted password can never be decrypted
User sign in
Validate access credentials against the encrypted password on the database
Yes
Generate a session token and save the cookie
Update new session details on the database, IP, date, time.
No
Deny access
Two Factor Authentication
Requests a verification code, via SMS for example
Validate the verification code with the application
Validation succeeded
Update new session details on the database with the latest used code
Validation fails
Deny access