TCM://
THE CODING MACHINE

Point du vendredi

Web Security (2023 edition)

31/03/2023

THB


Security
T-Mobile says hacker accessed personal data of 37 million customers
@lorenzofb / 11:36 PM GMT+1 • January 19, 2023
MEDIA INDUSTRY SEPTEMBER 22, 2016 / 1:27 PM / UPDATED 7 YEARS AGO
Yahoo says hackers stole data from 500 million accounts in 2014
MICROSOFT / TECH / SECURITY
Huge Microsoft exploit allowed users to manipulate Bing search results and access Outlook email accounts
Tech
Twitch confirms massive data breach
6 October 2021
Cybersecurity
Microsoft data breach "BlueBleed" exposes 2.4TB of customer data
HOME > TECH
533 million Facebook users' phone numbers and personal data have been leaked online

Introduction

"60% of small companies go out of business within six months of falling victim to a data breach or cyber attack." (Cisco / National Center for the Middle Market, 2017)

- Security is a cornerstone of every project… but it isn't the main focus of our projects
- This presentation will summarize the goals of security handling of our projects and give you tips and tools to apprehend this whatever your experience is

SOMMAIRE

01

What, When, Where ?

What?

- Web application security is the process of protecting every aspect of a website : its functionalities, its data, its files and even its source code
- “Security” implies “reliability”!

When? Where?

- Whenever. Wherever. Period.
- No project can afford to neglect security implementation
- You should always question your client or yourself about the way this feature, this data etc. is available or not

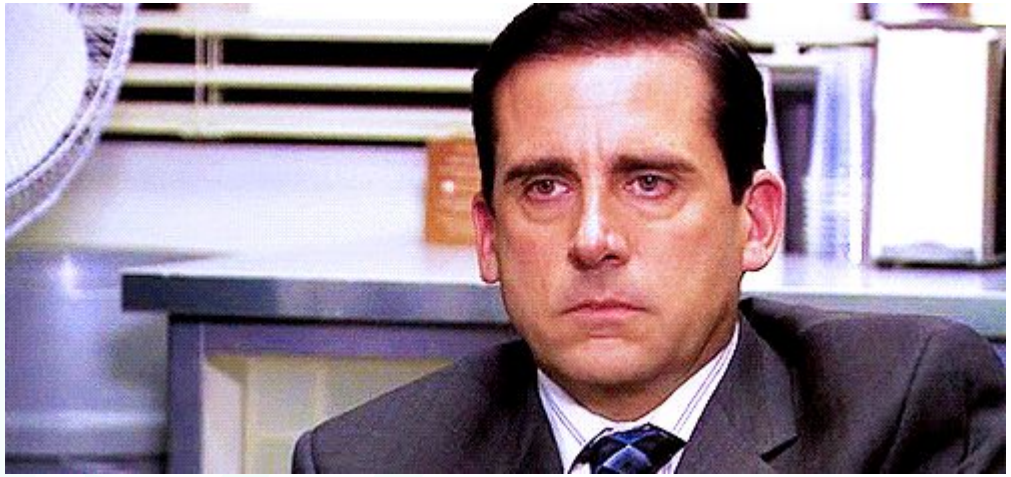
“Which kind of users can do that?”

“When should this action be possible or not?”

“Should this document be accessible when I’m not logged in?”

But...

"But, I don't have a budget for this!"
"But, I don't have time for that!"



There is not "but" with security

- Security developments will cost you less time (and less money) if implemented directly at the beginning of your developments

- As it may impact your data model, the security of your application should be considered as soon as your developments start.

There is not “later“ with security

- “I’ll do this later” means that you probably won’t do it (or won’t do it correctly)

- Your security handling must be seen as a ‘standard’ when working on your project (especially when working with multiple people)

02

Back to basics

Back to basics

- Hard to identify what the “basics” of security are (what is mandatory, what is optional), especially when everyone at any level is concerned
- Is there a list of risks to start with?

OWASP TOP 10

"A good first step toward more secure coding"

- Open Web Application Security Project
- Online community that has been delivering tools and methodologies regarding web app security since 2003
- Its "Top 10" identify the most critical risks facing organizations
- Data collected frequently : 2017, 2021...

OWASP TOP 10: 2021 EDITION

Code	Risk	Comment
A1:2021	Broken access control	Incorrect enforced restrictions on what authenticated users are allowed to do
A2:2021	Cryptographic failures	Data that requires special precautions when exchanged with the browser
A3:2021	Injection	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection
A4:2021	Insecure design	Flaws created by the design of the architecture, of the project
A5:2021	Security misconfiguration	Insecure or incomplete default configurations, misconfigured HTTP headers...
A6:2021	Vulnerable and Outdated components	Vulnerable libraries, frameworks and modules
A7:2021	Identification and Authentication failures	Incorrect authentication and session management implementation
A8:2021	Software and Data integrity failures	Deserialization that leads to remote code execution
A9:2021	Security logging and monitoring failures	Allows attackers to further attack system
A10:2021	Server-side request forgery (SSRF)	Get around firewall and allows scan of server (e.g)

A1:2021 - BROKEN ACCESS CONTROL

Vulnerability:

- Bypassing access control checks by modifying the URL, the HTML, or using a API tool.
- Elevation of privilege: access to something without having the requirements

How to prevent it:

- With the exception of public resources, deny by default.
- Unique application business limit requirements should be implemented specifically
- Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present within web roots.
- Log repeated access control failures, alert admins when appropriate.

Why it was #1 in 2021: most of the applications tested had at least one occurence of Common Weakness Enumerations (CWEs) including *CWE-200: Exposure of Sensitive Information to an Unauthorized Actor*, *CWE-201: Exposure of Sensitive Information Through Sent Data*, and *CWE-352: Cross-Site Request Forgery* (318k occurrences)

Real life examples: Snapchat (2013) ; Facebook Business (2015) : possibility to assign admin permissions to himself

A2:2021 - CRYPTOGRAPHIC FAILURES

Vulnerability:

- Hosting of sensitive data (passwords, credit card numbers, health records...) that is not protected as it should be: accessible by the client, unencrypted etc.
- This data falls under privacy laws, e.g. EU's General Data Protection Regulation (GDPR)

How to prevent it:

- Classify data processed by the application. Identify which data is sensitive according to privacy laws or business needs.
- Don't store sensitive data unnecessarily. Discard it as soon as possible.
- Encrypt all data in transit with secure protocols such as TLS. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Disable caching for response that contains sensitive data.

Previously known as A3:2017 - Sensitive Data Exposure (233k occurrences) : *CWE-259: Use of Hard-coded Password*, *CWE-327: Broken or Risky Crypto Algorithm*

Real life example : Cloudbleed (Cloudflare, 2017) : ability to dump memory of Cloudflare servers, containing sensitive data, some of which were cached by search engines (the bug was triggered 1,242,071 times but 0 passwords, credit cards or health records were exposed according to Cloudflare)

A3:2021 - INJECTION

Vulnerability:

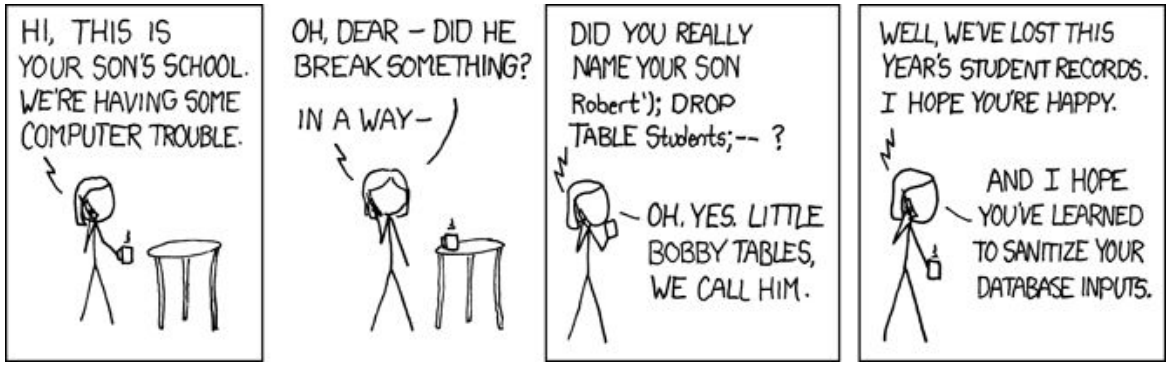
- User-supplied data is not **validated**, **filtered**, or **sanitized** by the application
- SQL injection is the most common injection but there can also be LDAP and OS command injections

How to prevent it:

- Use ORM (correctly!)
- Use positive or “whitelist” server-side input validation.
- For any residual dynamic queries, escape special characters

Includes *CWE-89: SQL Injection*, *CWE-73: External Control of File Name or Path* (274k occurrences) (now includes *CWE-79: Cross-site Scripting* too)

Real life example : The Panama Papers (11.5 million records from Mossack Fonseca regarding financial dealings, 2016) : SQL injection flaw in their Drupal website



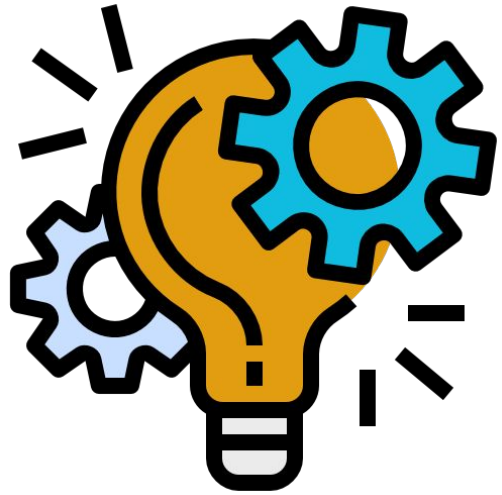
XKDC - Exploits of a Mom

```
private const SORT_COLUMNS = [
    'socialReason' => 'accounts.name',
    'name' => 'folders.name',
    'beneficiary' => 'CONCAT(folders.recipient_firstname, folders.recipient_lastname)',
    'city' => 'CONCAT(folders.recipient_city, folders.recipient_zipcode)',
    'premiumTotalAmount' => 'folders.premium_total_amount',
    'status' => 'folders_status.label',
    'cumacTotalAmount' => 'folders.cumac_total_amount',
    'partnerProStatus' => 'folders_status_pro_user.label',
    'partnerPublicStatus' => 'folders_status_public_user.label',
    'adminProStatus' => 'folders_status_pro_admin.label',
    'adminPublicStatus' => 'folders_status_public_admin.label',
    'assigner' => 'CONCAT(certyshare_app_users.firstname, certyshare_app_users.lastname)',
    'lastUpdateDatetime' => 'folders.last_update_datetime',
];
```

A4:2021 - INSECURE DESIGN

Vulnerability:

- Design and architectural flaws : by design, a vulnerability is exposed
- Presence of an insecure process : credential recovery workflow using “questions and answers” (multiple people can know the answers)
- Error message with sensitive information



How to prevent it:

- Validate your architecture choice by a professional
- Challenge unusual requests made during the conception
- Use library of secure design patterns
- Integrate plausibility checks at each tier of your application (from frontend to backend)

Includes *CWE-209: Generation of Error Message Containing Sensitive Information, CWE-256: Unprotected Storage of Credentials, CWE-501: Trust Boundary Violation, and CWE-522: Insufficiently Protected Credentials (262k occurrences)*

Real life example : Scalpers bots able to purchase entire stocks of GPUs on many e-commerce sites

A5:2021 - SECURITY MISCONFIGURATION

Vulnerability:

- Missing appropriate security hardening across part of the application stack or external services (e.g. Amazon S3...)
- Unnecessary features are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.

How to prevent it:

- Continuous Deployment (CD) to minimize the effort: Development, QA, and production environments configured identically, with different (and strong) credentials.
- Remove or do not install unused features, frameworks or packages.

Includes *CWE-16 Configuration* and *CWE-526 Exposure of Sensitive Information Through Environmental Variables* (208k occurrences)

Real life example: Accenture (2017): Authentication information as well as sensitive customer information hosted in a public Amazon S3 bucket

A6:2021 - VULNERABLE AND OUTDATED COMPONENTS

Vulnerability:

- Bad knowledge of the versions of all components you use (both client-side and server-side, directly or nested ones)
- Vulnerable, unsupported, or out of date software: OS, web server, database management system, APIs and libraries
- If you do not scan for vulnerabilities regularly
- If software developers do not test the compatibility of updated, upgraded, or patched libraries

How to prevent it:

- *composer update*, *yarn upgrade* (don't be afraid!)
- *yarn audit* and SecurityAdvisories that checks for vulnerable components
- Remove unused dependencies and components
- Only obtain components from official sources to reduce the chance of including a malicious component (check number of issues, of stars, date of the latest commit and release, etc. on GitHub)
- Add Dependabot: continuously update the versions of both client-side and server-side components (ask Guillaume !)
- Subscribe to email alerts for security vulnerabilities related to components you use

Includes CWE-1104: Use of Unmaintained Third-Party Components

Real life example: Log4j (open-source library used by major online service providers, 2021) : possibility to execute arbitrary Java code on a server

A7:2021 - IDENTIFICATION AND AUTHENTICATION FAILURES

Vulnerability:

- The authentication process can be brute forced (allow weak passwords)
- Weak credential recovery and forgot-password processes

How to prevent it:

- Do not ship with any default credentials
- Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.
- Limit or increasingly delay failed login attempts. Log all failures and alert administrators when attacks are detected.
- Ensure registration and credential recovery are hardened against account enumeration attacks by using the same messages for all outcomes.
- Where possible, implement multi-factor authentication (MFA) to prevent automated attacks.

Was previously #2 in 2017 (known as A2:20217 - Broken Authentication) : risks are mitigated thanks to the use of standardized frameworks that includes protections against these vulnerabilities.

Includes *CWE-297: Improper Validation of Certificate with Host Mismatch*, *CWE-287: Improper Authentication* and *CWE-384: Session Fixation* (132k occurrences)

Real life example: Department of Revenue (2012) : default password set in the authentication layer, 387 000 credit card numbers and 3.6 million Social Security numbers stolen

A8:2021 - SOFTWARE AND DATA INTEGRITY FAILURES

Vulnerability:

- The application relies upon plugins or libraries from untrusted sources, repositories and content delivery networks
- Insecure deserialization

How to prevent it:

- Ensure libraries and dependencies are consuming trusted repositories
- Ensure that there is a review process for code configuration changes
- Ensure the integrity of the code flowing through the build and deploy processes using CI / CD pipeline

New category : includes *CWE-829: Inclusion of Functionality from Untrusted Control Sphere*, *CWE-494: Download of Code Without Integrity Check* and *CWE-502: Deserialization of Untrusted Data*

Real life example : SolarWinds (2020) : compromised updates sent to IT ressources of numerous companies and government agencies such as the Pentagon

A9:2021 - SECURITY LOGGING AND MONITORING FAILURES

Vulnerability:

- Auditable events, such as logins, failed logins, and high-value transactions are not logged.
- Errors generate no (or unclear) log messages.
- Logs of applications and APIs are not monitored for suspicious activity.

How to prevent it:

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious accounts
- Establish effective monitoring and alerting such that suspicious activities are detected and responded in a short time.
- Handling error like a pro (by @DAN)

Includes *CWE-117 Improper Output Neutralization for Logs, CWE-223 Omission of Security-relevant Information, and CWE-532 Insertion of Sensitive Information into Log File.*

A10:2021 - SERVER-SIDE REQUEST FORGERY (SSRF)

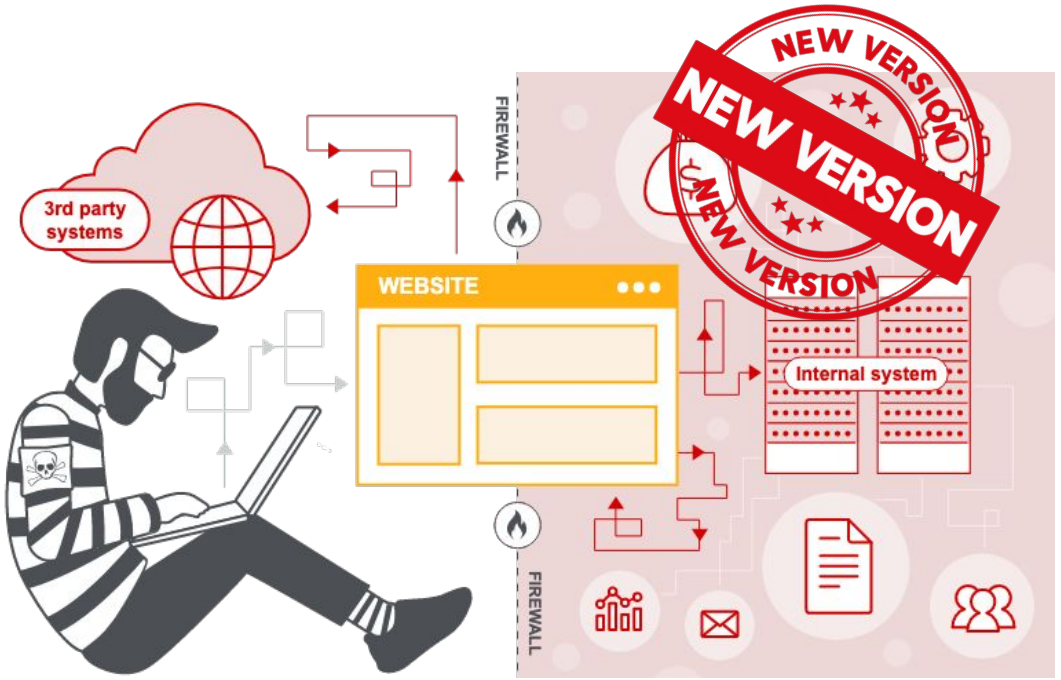
Vulnerability:

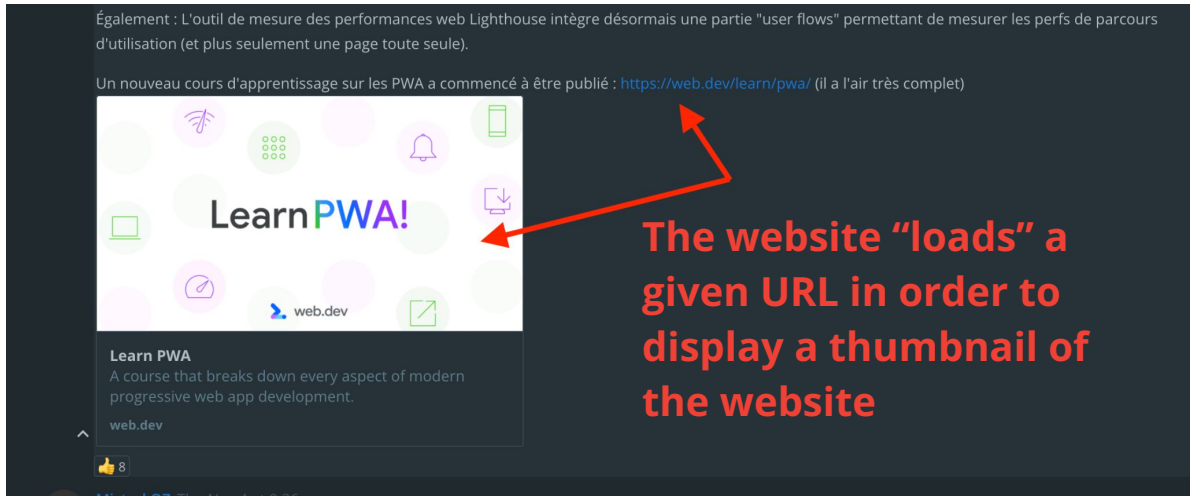
- A given URL is loaded without whitelisting
- Your server exposed internal structure and content
- A 3rd party can send request on behalf of a server

How to prevent it:

- Segment resource in separate networks
- Enforce “deny by default” firewall policies to block all but essential intranet traffic
- Sanitize and validate all client-supplied input data
- Enforce the URL schema, port, and destination with a positive allow list
- Do not send raw responses to clients

New category which appears because of an increasing risk linked to this problem





OTHER RISKS THAT YOU NEED TO KEEP IN MIND

These items didn't make the cut in OWASP Top 10 but are well worth the effort to identify and remediate.

Code Quality issues

- Code quality issues include known security defects or patterns, reusing variables for multiple purposes, exposure of sensitive information in debugging output
- Solution (e.g.): use (and configure correctly) a static code analysis tool

Denial of Service

- Always possible given sufficient resources
- Anyone with the link can access a large file, or a computationally expensive transaction occurs on every page
- Solution: Performance test code, monitor memory usage, re-architect, optimize, or cache expensive operations, add access controls for larger objects to ensure that only authorized individuals can access huge files

Memory Management Errors

- The languages we use everyday are written in systems languages that have memory management issues
- Solution: use memory-safe languages such as Rust or Go, update to newer versions of PHP for example. For existing code, use of strict compiler flags, strong typing, static code analysis to identify memory leaks, memory, and array overruns, and more.

03

Advanced access control

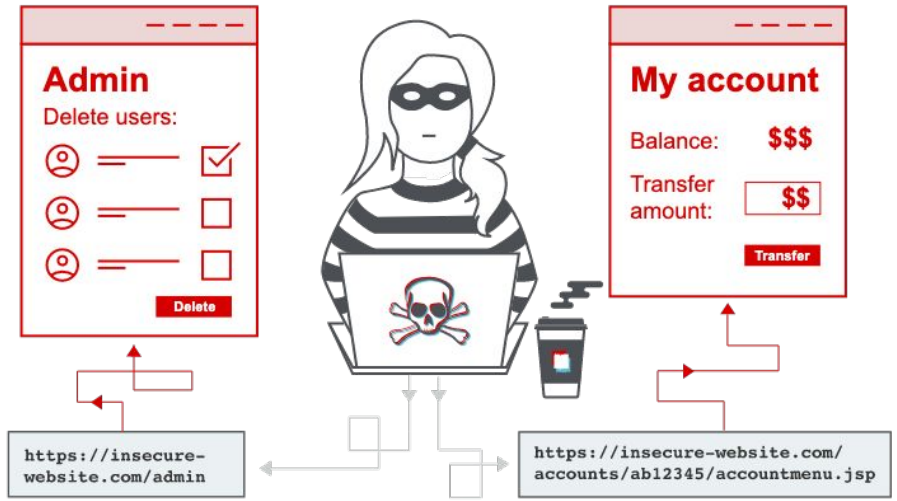
Advanced access control

Access control determines whether the user is allowed to carry out the action that they are attempting to perform.

3 different types of access control exist:

- **Vertical access control**
- **Horizontal access control**
- Context-dependent access control

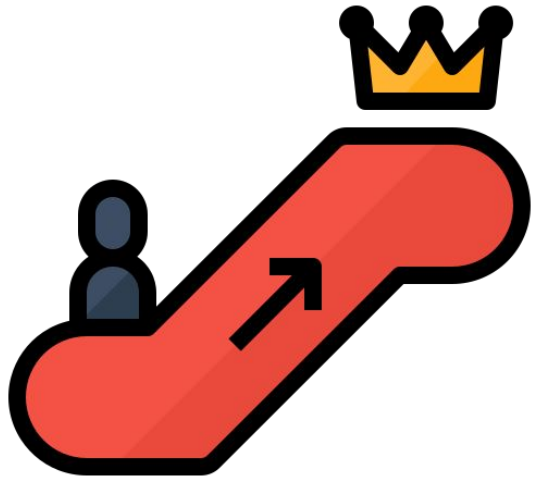
(perform the actions in a wrong order)



Vertical access control: definition

- Vertical access controls are mechanisms that restrict access to sensitive functionality that is not available to other types of users.

- For example, an administrator might be able to modify or delete any user's account, while an ordinary user has no access to these actions.



Vertical access control: implementation in Symfony

Use annotation @Security (with method “**hasRight**” / “**hasRole**” for example):

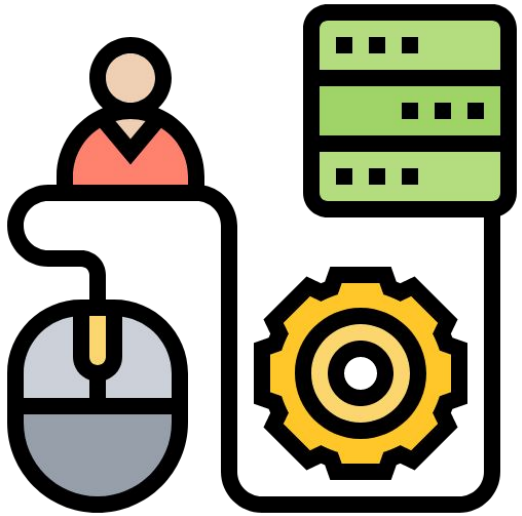
- `Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;` ***(REST controllers)***
- `TheCodingMachine\GraphQLite\Annotations\Security;` ***(GraphQLite controllers)***
- `TheCodingMachine\GraphQLite\Annotations\Right('ROLE_MYROLE');` ***(GraphQLite)***

```php
/**
 * @param Folder[] $folders
 *
 * @throws StringsException
 *
 * @Mutation()
 * @Logged()
 * @Security("user.hasRight('EDIT_FOLDER_ADMIN_USER')")
 */
public function assignFoldersToAdminUser(array $folders, AdminUser $adminUser): Success
```

Horizontal access control

- Horizontal access controls are mechanisms that restrict access to resources to the users who are specifically allowed to access those resources.

- For example, a banking application will allow a user to view transactions and make payments from their own accounts, but not the accounts of any other user.



Horizontal access control: implementation in Symfony

- **Voters** : @Security with **is_granted**
- You can mix the conditions, specify the status code and the message (cf. Symfony Doc.)
- Each voter will be evaluated to find the one matching your criterias
- Two methods to implement: supports() and voteOnAttribute()
- Very useful with tools that directly type your entrypoint parameters (GraphQLite)

```php
/**
 * @throws StringsException
 *
 * @Mutation()
 * @Logged()
 * @Security("is_granted('CAN_ACCESS_FOLDER', folder)")
 */
public function assignToAdminUser(Folder $folder, AdminUser $adminUser): ?CertyshareAppUser
```

supports()

Decides if the"Voter" is the correct one (true or false):

- Does it support the **$attribute**?
- Is the **$subject** correctly typed?

```php
<?php

declare(strict_types=1);

namespace App\Security;

use ...

class FolderVoter extends Voter
{
    public const CAN_ACCESS_FOLDER = 'CAN_ACCESS_FOLDER';
    public const CAN_ACCESS_DOCUMENT = 'CAN_ACCESS_DOCUMENT';
    public const CAN_ACCESS_SUPPORTING_DOCUMENT = 'CAN_ACCESS_SUPPORTING_DOCUMENT';
    public const CAN_ACCESS_FOLDER_INSTRUCTION  = 'CAN_ACCESS_FOLDER_INSTRUCTION';

    /**
     * phpcs:disable
     * @param string $attribute
     * @param mixed  $subject
     * @return bool
     */
    protected function supports($attribute, $subject)
    {
        // if the attribute isn't one we support, return false
        if (! in_array($attribute, [self::CAN_ACCESS_FOLDER, self::CAN_ACCESS_DOCUMENT, self::CAN_ACCESS_SUPPORTING_DOCUMENT,self::CAN_ACCESS_FOLDER_INSTRUCTION])) {
            return false;
        }
        if($subject instanceof FolderInstruction){
            $subject = $subject->getFolder();
        }
        // only vote on Folder objects inside this voter
        if (! $subject instanceof Folder
            && !$subject instanceof SupportingDocumentUploadedFile
            && !$subject instanceof SupportingDocument
            && !$subject instanceof SupportingDocumentType
        ) {
            return false;
        }
        return true;
        // phpcs:enable
    }
```

voteOnAttribute()

- The logic that defines your specific rule about the horizontal access control of your **$subject**
- Use the **TokenInterface $token** to retrieve the logged user and check his/her access on your $subject
- Use specific code for each case supported by your voter

```php
/**
 * phpcs:disable
 * @param string $attribute
 * @param mixed  $subject
 * @return bool
 */
protected function voteOnAttribute($attribute, $subject, TokenInterface $token)
{
    $user = $token->getUser();
    \assert( assertion: $user instanceof SerializableUser);
    $certyshareAppUser = $user->getCertyshareAppUser();
    if (! $certyshareAppUser instanceof CertyshareAppUser) {
        // the user must be logged in; if not, deny access
        return false;
    }

    switch ($attribute) {
        case self::CAN_ACCESS_FOLDER_INSTRUCTION:
            /**
             * Folder $folder
             */
            $folder = $subject;

            return $this->canAccessFolderInstruction($folder, $certyshareAppUser);
        case self::CAN_ACCESS_FOLDER:
            $folder = $subject;
            \assert( assertion: $folder instanceof Folder);

            return $this->canAccessFolder($folder, $certyshareAppUser);
        case self::CAN_ACCESS_DOCUMENT:
            $supportingDocumentUploadedFile = $subject;
            \assert( assertion: $supportingDocumentUploadedFile instanceof SupportingDocumentUploadedFile);

            return $this->canAccessDocument($supportingDocumentUploadedFile, $certyshareAppUser);
        case self::CAN_ACCESS_SUPPORTING_DOCUMENT:
            /**
             * @var SupportingDocument $supportingDocumentUploadedFile
             */
            $supportingDocument = $subject;

            return $this->canAccessSupportingDocument($supportingDocument, $certyshareAppUser);
    }

    throw new LogicException( message: 'This code should not be reached!');
}
```

```php
private function canAccessFolder(Folder $folder, CertyshareAppUser $currentUser): bool
{
    //If Admin and VIEW_FOLDERS
    if ($currentUser instanceof AdminUser) {
        return $folder->checkCanAdminAccessFolder($currentUser);
    }

    //If User and folder.account.id in the family ids
    return $currentUser instanceof User && $folder->checkCanPartnerAccessFolder($currentUser);
}
```

04

Challenge your security

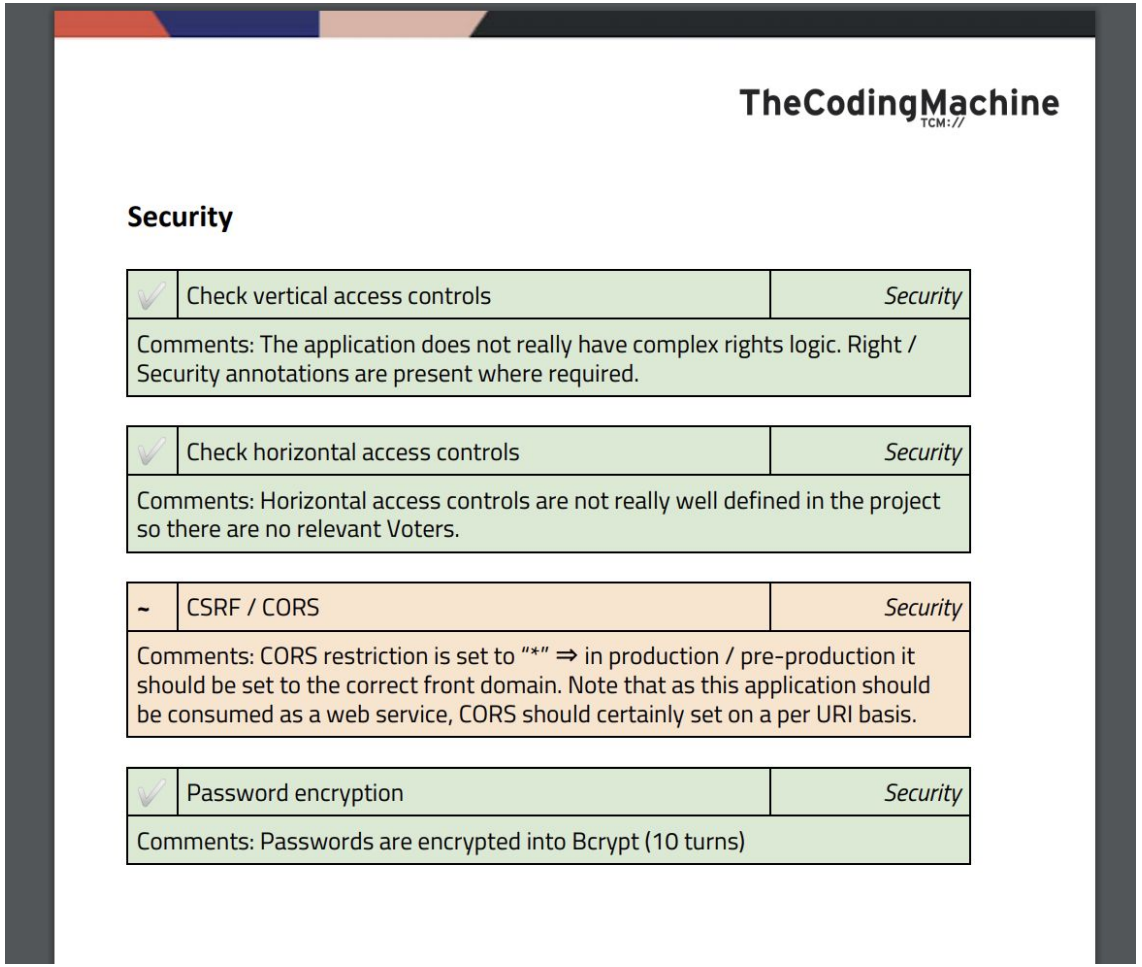
ACTIONS TAKEN TO IMPROVE THE SECURITY OF OUR TOOLS

- Enabling 2FA on our Gitlab
- Enabling 2FA on our Google Accounts
- Audit by the Technical Direction
- Zoho Vault for password management (*request an access if you need it*) (RIP the LoginMachine)

WHAT YOU CAN DO FOR THE SECURITY AT TCM

- Update your operating system
- Update your browser
- Don’t forget to lock your computer when you don’t use it

- Client Security audit (Penetration tests...)
- Confront with OWASP Top 10
  - SecurityHeaders.com
- TheCodingMachine audit by the Technical Team (10%, 50%, post mortem)
- Be adventurous: test your own application!



TheCodingMachine

Security

✓	Check vertical access controls	*Security*
Comments: The application does not really have complex rights logic. Right / Security annotations are present where required.		
✓	Check horizontal access controls	*Security*
Comments: Horizontal access controls are not really well defined in the project so there are no relevant Voters.		
~	CSRF / CORS	*Security*
Comments: CORS restriction is set to "*" ⇒ in production / pre-production it should be set to the correct front domain. Note that as this application should be consumed as a web service, CORS should certainly set on a per URI basis.		
✓	Password encryption	*Security*
Comments: Passwords are encrypted into Bcrypt (10 turns)		

05

Feedback

Feedback

- My first “big” project from scratch with
- Ambitious project for the client: revamp their customer relationship management tools
- Have to deal with sensible data: personal information, identity document, tax notice…
- GraphQLite, Symfony: discovery of the voters

Penetration tests

- Pentests done by the IT department of the parent company
- Some relevant security issues (XSS, GraphQLite API exposure…)
- Some smart remarks (Time of response of the “forgotten password” call)
- Some compliments about things operated by the framework (Protection of the session cookie)

06

Horror Stories

Horror Stories

- Unprotected routes in configuration of the framework
- Unprotected pages in the back office
- No @Logged annotations in the whole project
- Database access from the open ports of your Docker
- Projects leaks using Git (Hi WordPress!)
- .env file commited with login information : impossible to delete it as Git stores the history
- “Script kiddies” use scripts to scan the web, find websites with known vulnerabilities and deface them: hacked WordPress projects : keep your project dependencies up to date!

Any other “horror story” to confess ?

Conclusion

WHAT YOU NEED TO REMEMBER

- Web security concerns everyone and may have an impact on many aspects of your application. It's the **cornerstone of every project**
- The security of an application **is not only a checklist but also a cursor to adjust** according to the nature of your application
- Security works **in layers**. Don't hesitate to add security even if you think it is useless (for instance strong passwords in dev environments). It might one day save you
- Frameworks and packages implement **solutions to facilitate** the implementation of security measures (Symfony voters, @Security annotations...)
- Don't be afraid to challenge your habits on the subject ; Don't be afraid to discuss about security with your client : **Security** implies **reliability**
- Every new project will be audited internally: **at the start and in the middle of the project** (ask if you need to audit your 'old' application)

Thank you !

Any questions?

contact@thecodingmachine.com
www.thecodingmachine.com

TheCodingMachine
56 rue de Londres - 75008 - Paris


TCM://
PARIS
56 rue de Londres
75008 Paris


TCM://
LYON
35 Rue de Marseille
69007 LYON


TCM://
HONG-KONG
Level 15
The Lee Garden Two 28 Yun Ping Road
Causeway Bay, Hong Kong


TCM://
LISBOA
Rua da Palma, 219, 3ºEsq
1100-391 Lisboa

TCM://
THE CODING MACHINE

Point du vendredi - Add on

TechPoints

~~Peer codings noob~~

New process, new frame

- ➔ Follow-up / support on internship / junior team mates technical progress
- ➔ Define objectives
- ➔ Peer Coding
- ➔ Transversal milestones during internship / trial period

Objectives

TCM://

Global Process

➔ 3 Milestones

✓ M+1 [TechPoints]

✓ M+3 [TechPoints]

✓ Pre-hiring [Global overview] new

TCM://

- ➔ New Frame (easier feedback)
- ➔ @DP/CdP TechPoints to be prepared d-1
- ➔ Now include some PM soft skills evaluation
- ➔ @DP/CdP Feedback to be performed afterwards
  (M+3 will be included in mid-internship feedback)

TechPoints

Whats' new ?

- ➔ Required soft skills overview
- ➔ PM skills discussion
- ➔ Main objectives for the coming months / year

Pre-hiring review

TCM://