Fachhochschule
Nordwestschweiz

Application Security (apsi)

Lecture at FHNW

Lecture 12, 2021

Arno Wagner, Michael Schläpfer, Rolf Wagner

<arno@wagner.name>, <{michael.schlaepfer,rolf.wagner}@fort-it.ch>

Lecture Evaluation

1215 – 1225 Lecture evaluation apsi

Dates and rooms

- MSP

03.02.2021 (Thursday)	13:15-14:45	6.0D13

- Remaining on-site lectures

20.12.2021	12:15-15:00	5.3A17
10.01.2022	12:15-15:00	5.3A17
17.01.2022	12:15-15:00	5.3A17

- Guest-Lecture

10.01.2022	13:15-14:00	5.3A17

10.01.22: Guest Speaker about WAF/Identity Federation

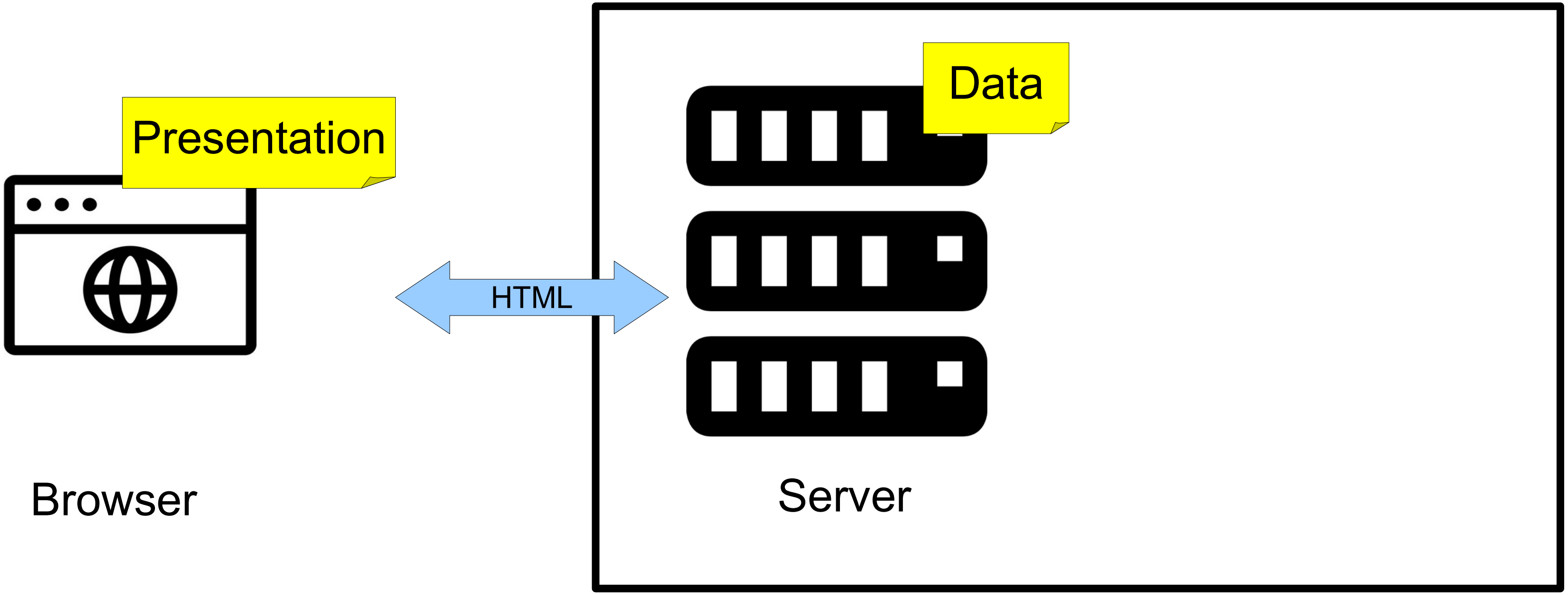
UNITED SECURITY PROVIDERS

- Christoph Schulthess, Teamleader Application Security @ United Security Providers AG
- Corporate Networks and Application Level Security using Web Application Firewalls
- Integration with web applications and consequences on software development/deployment for you

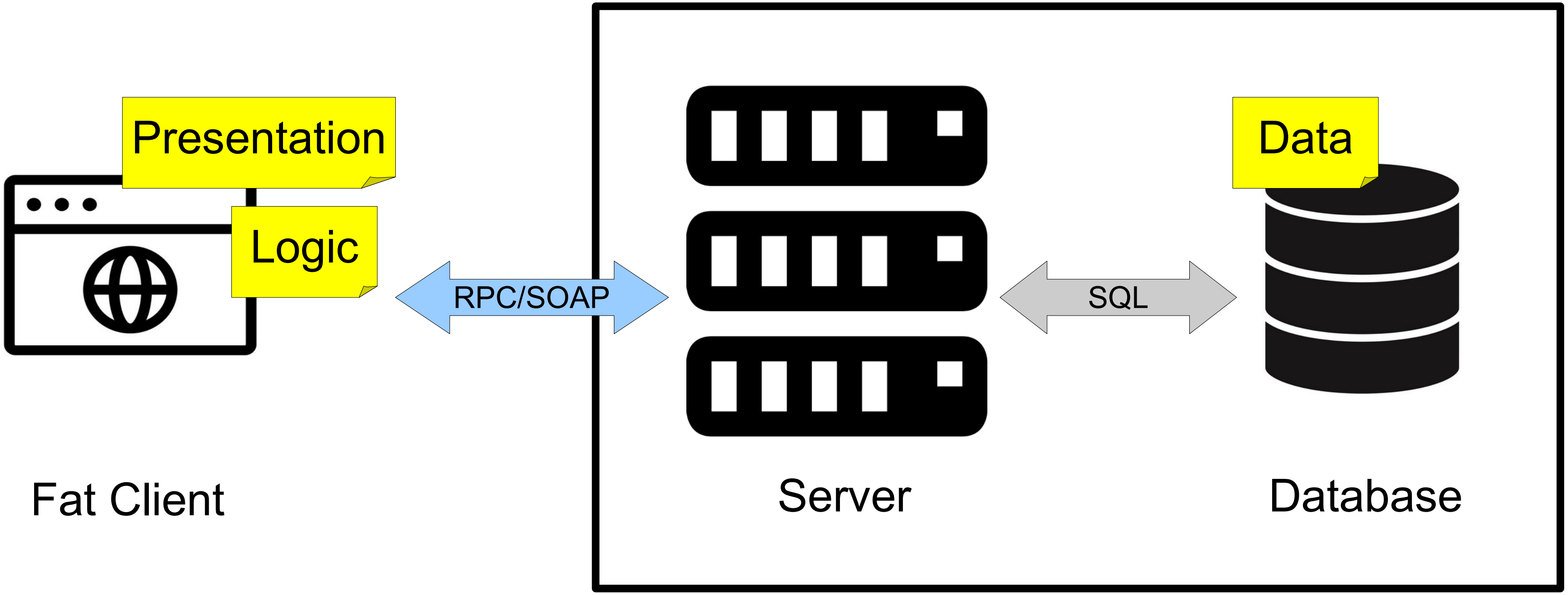
Agenda

- **Brief history on Web Technology**
- Security Architecture of Web Applications
- (Security-) Frameworks
- Example: Spring Security Framework
- Example: MEAN Stack

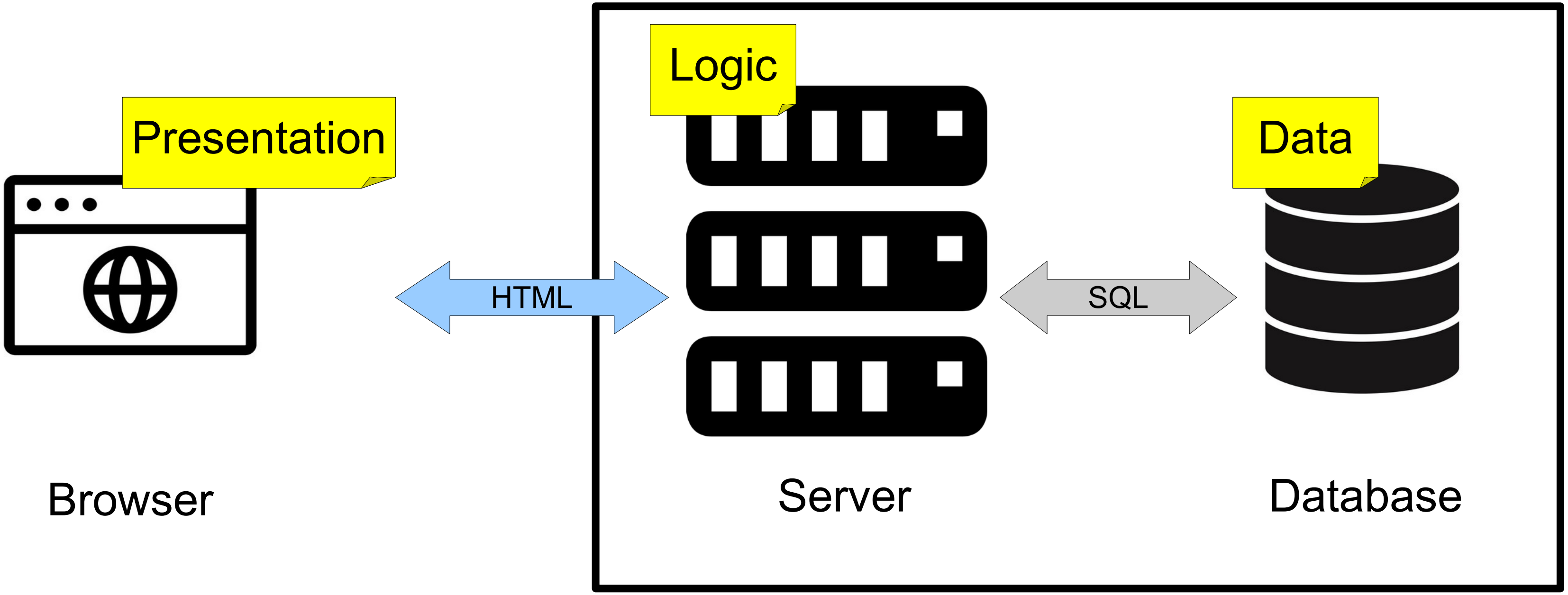
History on Web Technology (~1990)



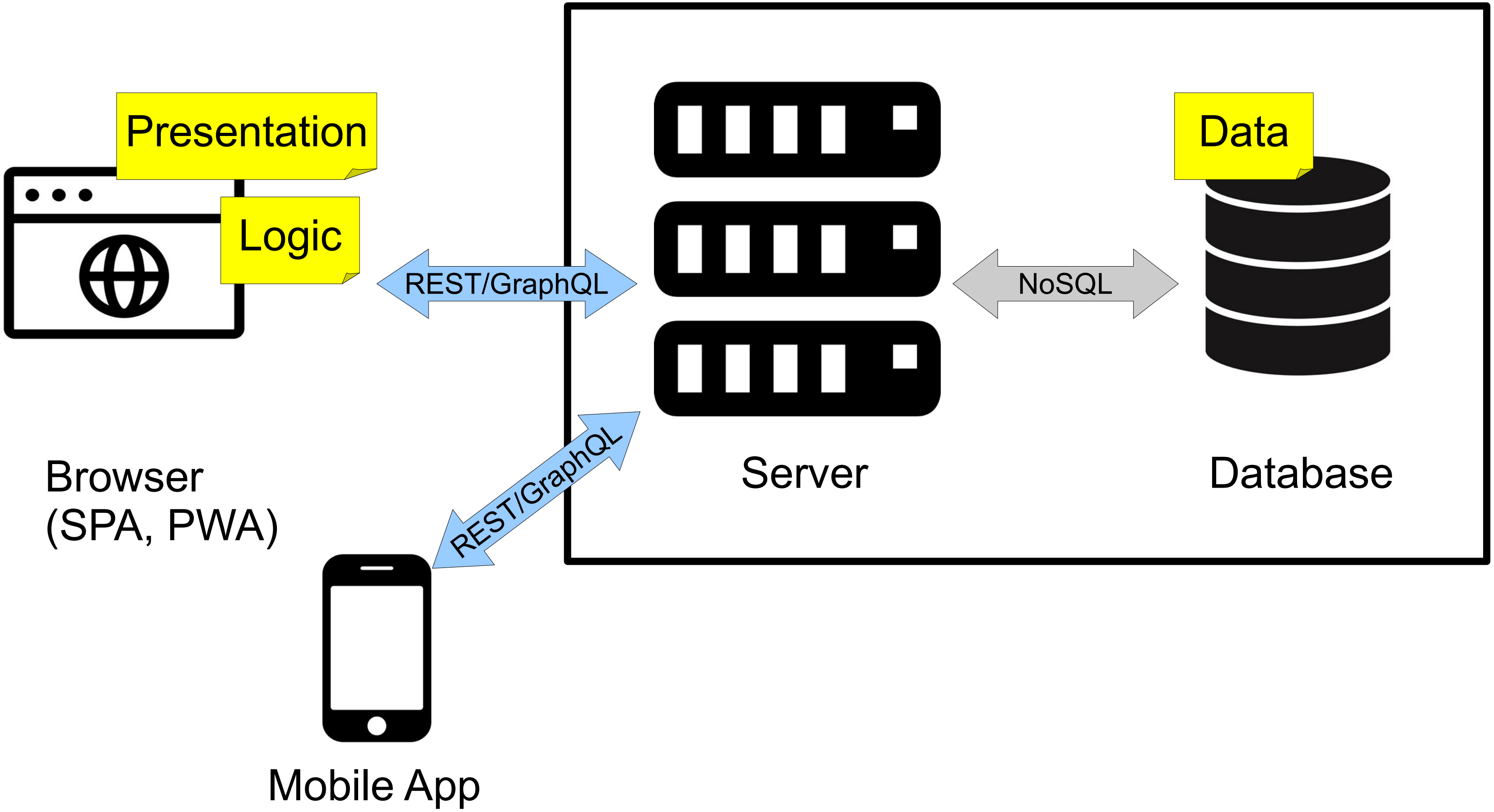
History on Web Technology (~2000)



History on Web Technology (~2010)



History on Web Technology (~2020)



Pitfalls

What could possibly go wrong when we put the logic in the browser?

Amongst many other issues the following are obvious:

- Secrets must not be stored in the application (browser)
- Authentication flows must be validated by the backend (e.g., challenge/ response)
- Every API call's authorization must be validated (principle of complete mediation/ Zero Trust)
- The application's logic cannot be trusted, since the adversary is in control of it's execution (WAFs are useless with respect to ensuring Layer-7-logic)

Agenda

- Brief history on Web Technology
- **Security Architecture of Web Applications**
- (Security-) Frameworks
- Example: Spring Security Framework
- Example: MEAN Stack

Security Architecture of a Web Application

What aspects must it cover?

- Identities: Anon, pseudonym, clear-names?
- Authentication:
  - How to establish: Local / SSO / ID provider (local LDAP, Open ID connect, ...)?
  - How to maintain? Typically cookies these days, but lifetimes and protection?
  - What about site and possibly client certificates?
- Authorization:
  - What roles/permissions are there?
  - How is access limited?
- Confidentiality: Is there data that needs protection? Passwords? Other?
- Integrity: Is there data that an attacker may want to change?
  - Defacement, Sabotage, …
  - Order and payment data, access data for other services, etc.

Security Architecture of a Web Application

What aspects are not or not necessarily in scope?

- Performance:
  This one is tricky. Denial-of-Service may or may not be an issue.
  Performance may or may not be able to fix it if it is an issue.
- Reliability:
  See above. Also, reliability issues are often also security issues.
- Look & Feel:
  This one is also tricky. Bad decisions can allow attackers to trick users.
- Maintenance:
  In scope only for components providing a security function.
- Backup:
  In scope for disaster-recovery after an attack

User States

Example: Web-Forum

- Unauthenticated:
  - Read access or not? Write access in some places? Write as "anonymous"?
- Authenticated regular user
  - Read access everywhere or not (private boards…)?
  - Write access everywhere or not (private boards, admin messages, …)?
- Moderator
  Read/write: Same as regular user or more/less?
  - Delete access to postings?
  - Approver access in some/all places?
  - Access to "close" discussion threads?
  - Approval of new users?
- Admin
  - Moderator access or not? General or restricted?
  - Can make users moderators? Can make users admins?

State Transitions

- Unauthenticated → Authenticated
  - Typically via log-in

- Authenticated → Unauthenticated
  - Log-out (not always)
  - Session end (browser close)
  - Timeout
  - Forced immediate log-out by admin or moderator

- Authenticated → Moderator/Admin
  - Usually via authorization mechanism
    But: Also as user-initiated change
  - How to handle:
    – check on each request,
    – permission in cookie
    – permission in state

- Moderator/Admin → Authenticated
  - may or may not be possible...

Content-Based Attacks

This is a forum, so users post content

The obvious ones:

- XSS
- CRSF
- Malware in binaries (also pictures linked to)
- …

The less obvious ones

- Insecure links, possibly camouflaged (allow HTML-like markup?)
- Dangerous picture links (attack code in pictures)
- Illegal links, for example to copyright infringing sites
- ...

How to Protect Against User-Generated Content?

- For XSS, CSRF, etc. use well established defense techniques
  => Can be part of a framework, but caveats apply.
- Allow only plain-text or simple markup
  => This needs to be enforced carefully!
- Do not allow links
  What about non-active links? May still cause problems…
- Do not allow binary uploads. But what about pictures, e.g. an user icon?
  Or maybe do a virus-scan?
- Allow posting only after moderation
  => Somebody has to do that…
- Allow binary uploads only after moderation
  => It is basically impossible to check those reliably
-

Agenda

- Brief history on Web Technology
- Security Architecture of Web Applications
- **(Security-) Frameworks**
- Example: Spring Security Framework
- Example: MEAN Stack

Between 70% and 80% of all mobile and cloud breaches are due to misconfigurations.
– Gartner (2019)

Why use a (Security-) Framework?

First and foremost: To save time

Development:

- Security (base-) mechanisms are already there
- No need to actually understand the details to get it working
  => This is a real problem. A lot of application vulnerabilities come from this.

Maintenance:

- Security is "somebody else's responsibility"
  May be good or bad but certainly saves time…
- New mechanisms may become available (or not) in the framework

Bad Reasons to use a (Security-) Framework

- Criteria, if you are looking for staff
  => People with "xyz" on their CV will surely know how that works, right?
- As "quality signal" in advertising
  "We use the well-known framework XYZ".
  => But what if it becomes infamous for being insecure...
- As a way to get it done with cheaper developers
  => You do not need to know how the framework does things, right?
- "Everybody does it"
  => And everybody may just have problems with it…
- "The competition does it"
  => Yes. And what is their reasoning? Good or bad?
- "It is the standard"
  => No, it is not. If you want standards check RFCs, not an implementation

Problems of Using a Security Framework

- Vulnerabilities are much more global
- He who can attack the framework may find many, many targets
  This increases economic incentives to create attacks
  => Patch availability and patching in time becomes critical
- Updates may break functionality
- The developers may just not care about you…
  => Particularly bad if these are updates to fix security problems!
- Security-audits become dependent on the framework as well
  => Should be re-done on framework upgrades
- Quality may vary over time, mechanisms can become obsolete
- Loss of control
- Not considering alternate mechanisms missing in a specific framework

What if the Framework will not let you go?

Can be because of "business reasons" or developer ego…

Possible solutions:

- Add a "Framework Abstraction Layer " and only use generic functions
  → This may be (too) much effort, but is the "clean" approach
  Note: It may be pretty difficult designing a future-proof FAL...
- Do not use a framework
  → May also be (too) much effort
- Evaluate other frameworks and design the application for all of them (e.g. Spring Security, JAAS, Apache Shiro, …)
  → If possible, this is a good solution

Of course, this all depends on planned application lifetime.
Beware: Temporary solutions have a tendency to become permanent!

Agenda

- Brief history on Web Technology
- Security Architecture of Web Applications
- (Security-) Frameworks
- **Example: Spring Security Framework**
- Example: MEAN Stack

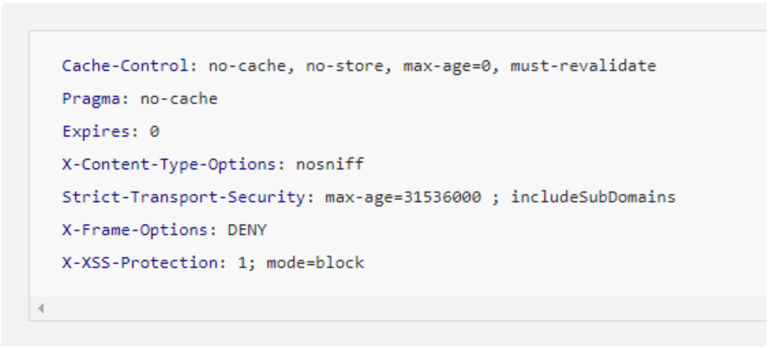
The Spring Security Framework

- https://spring.io/projects/spring-security (currently version 5.6.0)
- Primary developer: Pivotal Software (commercial enterprise, now VMware)
- Spring Security provides comprehensive support for authentication, authorization, and protection against common exploits

Features:

- Authentication & Authorization
- Password Storage (e.g. Argon2): Typically PasswordEncoder is used for storing a password that needs to be compared to a user provided password at the time of authentication
- Protection Against Exploits: CSRF, several security http response headers as for cache control, referrer policy, X-Frame, …
- …

Example 41. Default Security HTTP Response Headers



```
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
```

Source: spring.io

The Spring Security Framework

- Mechanisms supported:
  - OAuth 2.0, OpenID Connect 1.0 (e.g. login with a Google account)
  - SAML 2.0
  - Kerberos
  - Session management
  - “Remember Me”
    - Identify the user across multiple sessions
    - base64(username + ":" + expirationTime + ":" + md5Hex(username + ":" + expirationTime + ":" password + ":" + key))
    - What’s the problem here?
  - …

Versioning

- We talked about the challenges of using frameworks.
  Among others – about the “dependency hell”
- Spring Security uses the widely used semantic versioning approach:
  MAJOR.MINOR.PATCH
- MAJOR versions may contain breaking changes. Typically, these are done to provide improved security to match modern security practices
  □ incompatible changes
- MINOR versions contain enhancements but are considered passive updates
  □ added functionality in a backwards compatible manner
- PATCH level should be perfectly compatible, forwards and backwards, with the possible exception of changes that fix bugs.

Project Integration

- Spring Boot provides a spring-boot-starter-security starter that aggregates Spring Security-related dependencies together.

Spring Boot with Maven

```
<dependencies>
    ...
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
    ...
</dependencies>
```

Spring Boot with Gradle

```
dependencies {
    ...
    compile("org.springframework.boot:spring-boot-starter-security")
    ...
}
```

Characteristics of the External Dependency

Negative:

- External code may change at build-time without warning
  → This can happen even with specifically specified versions
- No way to delay patches or changes
- Code may stop to build at any time, in particular later

=> Enterprise-Environment will need local copy and archival of same

Positive:

- Security fixes are harder to overlook
- No local code repository to establish and maintain

Attacks Via External Dependency

There can be a lot of direct and indirect external dependencies

- Hard to monitor, something may "slip by" the maintainers
- Example: Somebody recently compromised node.js: "event-stream" 3rd party module steals cryptocurrency wallets
- This type of software supply chain attack is possible because in the open source world it is harder to discriminate between good and bad actors.
- Node.js (npm) and Python (PyPI) repositories are thought to be among the most commonly targeted by attackers, as malicious code can be easily triggered during package installation.
- There were 929 attacks recorded between July 2019 and May 2020, according to Sonatype’s annual State of the Software Supply Chain report.

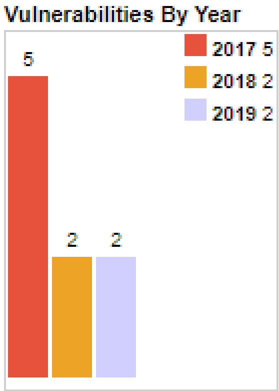
Attacks Via External Dependency

Attack path of the "event-stream" attack:

- Longtime event-stream developer no longer had time to provide updates
- Accepted the help of an unknown developer several months ago
- Attacker carefully injected attack code:
  - Added it in small steps.
  - Code Only became active for Copay (Open Source “secure” Bitcoin and Bitcoin Cash wallet)
  - Attack code is encrypted
    => Apparently nobody noticed this
- Attack code only found because a developer investigated a build warning….
- There was no measure in place to prevent this attack at all!
- … and remember this is node.js, not some small obscure project….

Security Track-Record of Spring Security

CVEs:



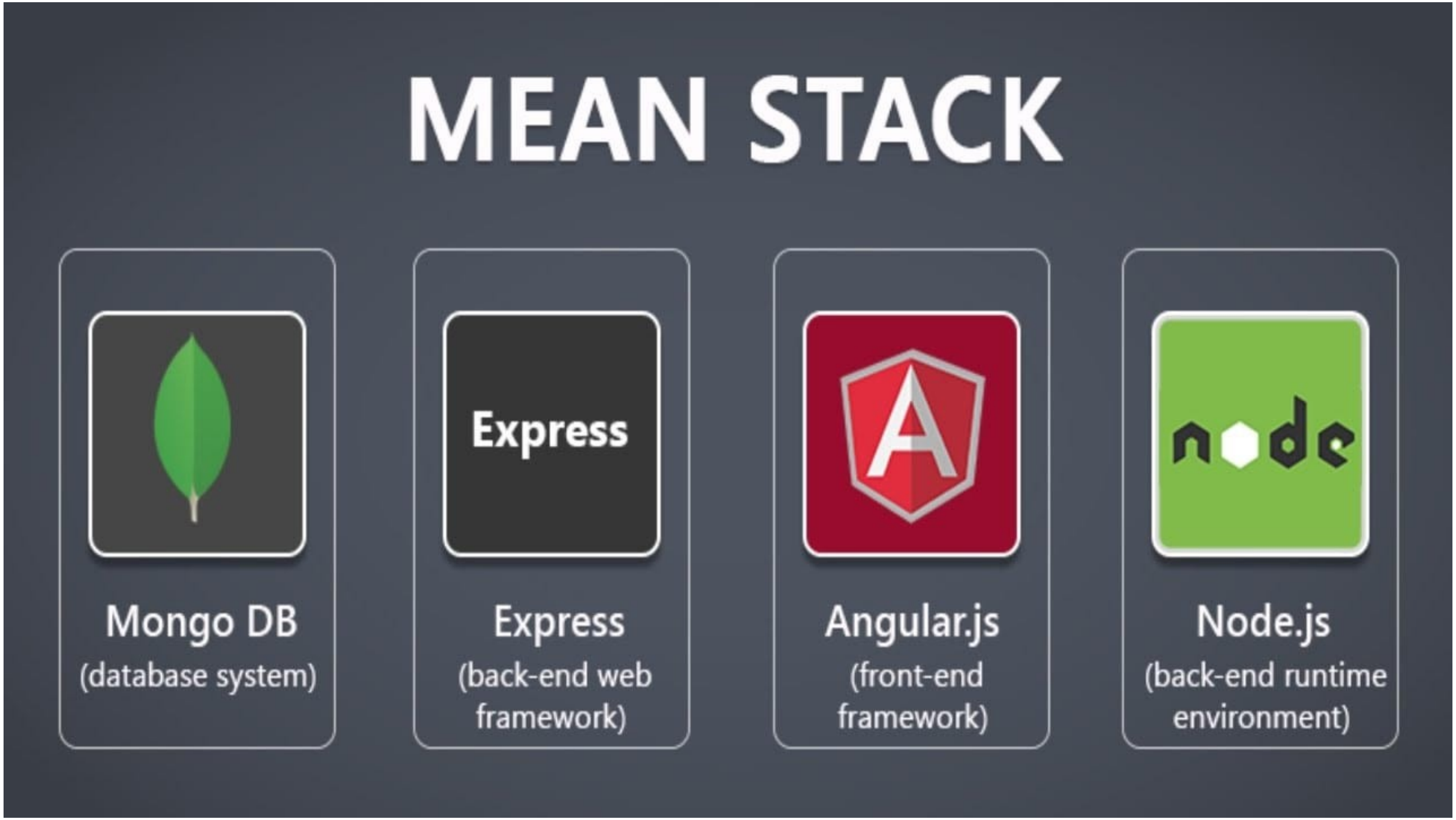
- CVE-2020-5408: Dictionary attack with Spring Security queryable text encryptor
- CVE-2019-11272: If an application using an affected version of Spring Security PlaintextPasswordEncoder and a user has a null encoded password, a malic attacker) can authenticate using a password of "null".
- CVE-2019-3795: Insecure Randomness When Using a SecureRandom Instanc by Spring Security
- CVE-2018-1258: Unauthorized Access with Spring Security Method Security
- CVE-2018-1199: By adding a URL path parameter with special encodings, an attacker may be able to bypass a security constraint.

Security quality estimate: Reasonable but needs careful attention.

Agenda

- Brief history on Web Technology
- Security Architecture of Web Applications
- (Security-) Frameworks
- Example: Spring Security Framework
- **Example: MEAN Stack**

Examples of webframeworks



MEAN STACK

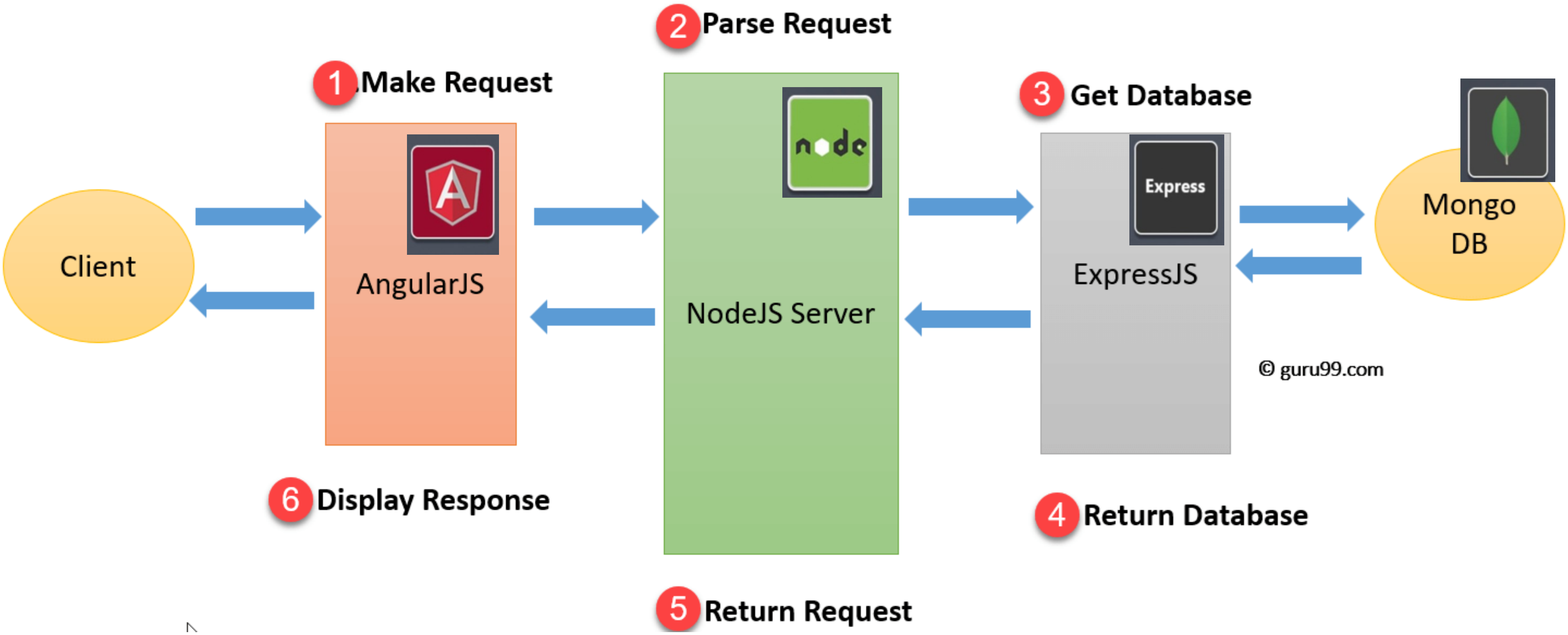
Mongo DB
(database system)

Express
(back-end web framework)

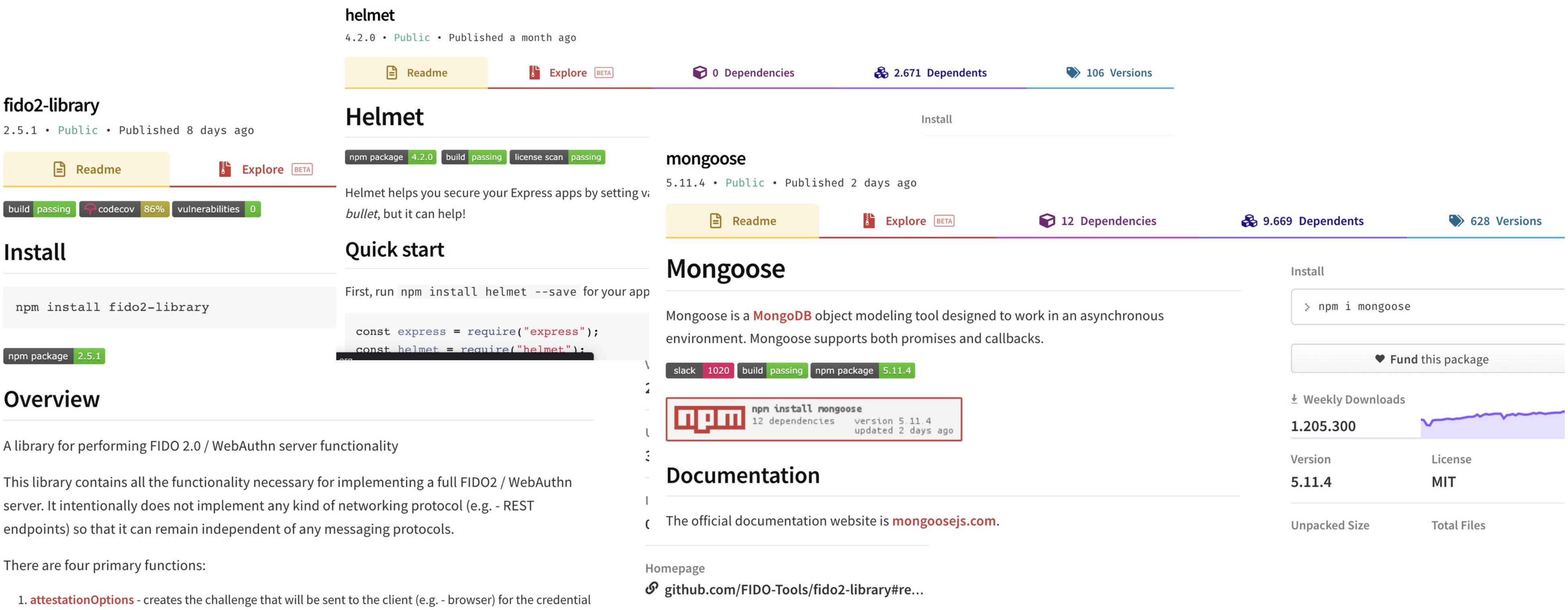
Angular.js
(front-end framework)

Node.js
(back-end runtime environment)

MEAN Stack - Overview



Example: npm Repository ("Node Package Manager")



fido2-library

2.5.1 • Public • Published 8 days ago

Readme | Explore BETA

build passing | codecov 86% | vulnerabilities 0

Install

```
npm install fido2-library
```

npm package 2.5.1

Overview

A library for performing FIDO 2.0 / WebAuthn server functionality

This library contains all the functionality necessary for implementing a full FIDO2 / WebAuthn server. It intentionally does not implement any kind of networking protocol (e.g. - REST endpoints) so that it can remain independent of any messaging protocols.

There are four primary functions:

1. attestationOptions - creates the challenge that will be sent to the client (e.g. - browser) for the credential

Homepage

github.com/FIDO-Tools/fido2-library#re...

helmet

4.2.0 • Public • Published a month ago

Readme | Explore BETA | 0 Dependencies | 2.671 Dependents | 106 Versions

Install

Helmet

npm package 4.2.0 | build passing | license scan passing

Helmet helps you secure your Express apps by setting v
bullet, but it can help!

Quick start

First, run `npm install helmet --save` for your app

```
const express = require("express");
const helmet = require("helmet");
```

mongoose

5.11.4 • Public • Published 2 days ago

Readme | Explore BETA | 12 Dependencies | 9.669 Dependents | 628 Versions

Mongoose

Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment. Mongoose supports both promises and callbacks.

slack 1020 | build passing | npm package 5.11.4

npm install mongoose
12 dependencies version 5.11.4
updated 2 days ago

Documentation

The official documentation website is mongoosejs.com.

Install

```
> npm i mongoose
```

♥ Fund this package

Weekly Downloads

1.205.300

Version	License
5.11.4	MIT

Unpacked Size | Total Files

MEAN Stack - MongoDB

- Disable JavaScript execution
- Use a framework to access your database (e.g., mongoose) but know how it works
- Properly set up Access Control (i.e., separate credentials, RBAC)
- Restrict and limit access to the database
- Encrypt data (at rest, in transit, additionally sensitive data)
- Auditing and logs
- As usual: stay up-to-date with security fixes

MEAN Stack - Express

- Always use TLS
- Use “Helmet” for setting proper security HTTP headers (e.g., CSP): https://helmetjs.github.io/
- Make sure all dependencies are up-to-date and secure (npm audit)

MEAN Stack - Angular

- Use interpolation ({{ }}) to safely encode and escape HTML/CSS expressions withing templates
- Prevent using [innerHTML]
- Prevent using templates concatenated with potential user input
- Do not manipulate the DOM on your own (e.g., node.appendChild() or using the document object), instead use Angular’s APIs to manipulate the DOM
- Hold all packages up-to-date and regularly scan for vulnerabilities (npm audit)

MEAN Stack – Node.js

- Use parameterized inputs only to prevent injection attacks
- Sanitize all user input to prevent XXS (Cross-site-scripting) attacks
- Use MFA to prevent automated attacks
- Discard sensitive data after use
- Patch old XML processors to prevent XXE (XML external entity) attacks
- Enforce access control on every request
- Keep all packages up-to-date
- Regularly scan your application for vulnerabilities (npm audit)
- Enable auditing and logging

Take Home Message

- Webframeworks help to develop web applications fast, reuse existing work, and avoid common development errors (also security errors)
- Ease of use comes with a price:
  - You have to trust the framework
  - You have to trust the components' developers and the repositories
  - Maintenance, security fixing, and future development of the used components is not assured / neither you have much control over it
  - Not the same deep understanding of how things work
  - Devs learn frameworks instead of security principles