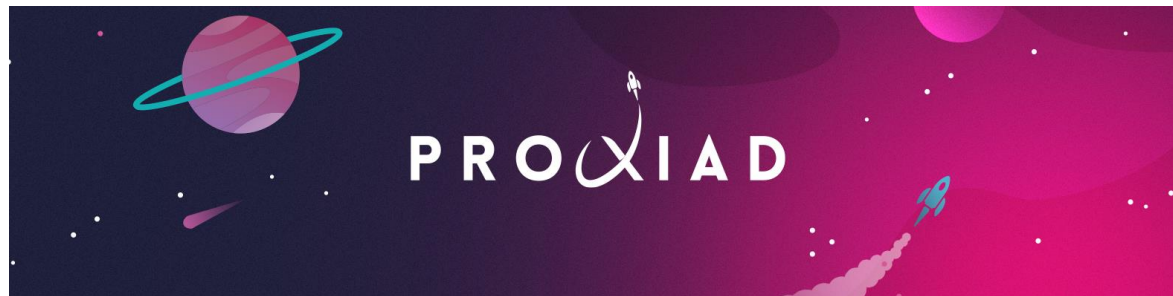


Front-end security with HATEOAS

Plovdiv, 01.04.2023



Software Security

Software security is the protection of software applications and digital solutions from unauthorized access, use, or destruction.

Software security is set of measures, practices and techniques incorporated in the software development life cycle (SDLC) and testing processes.

Authentication

Who you are? - verifying identity

Relies on a factor to establish trust:

- Something you know (like passwords)
- Something you have (using your phone for OTPs)
- Something you are (biometrics, such as your face).



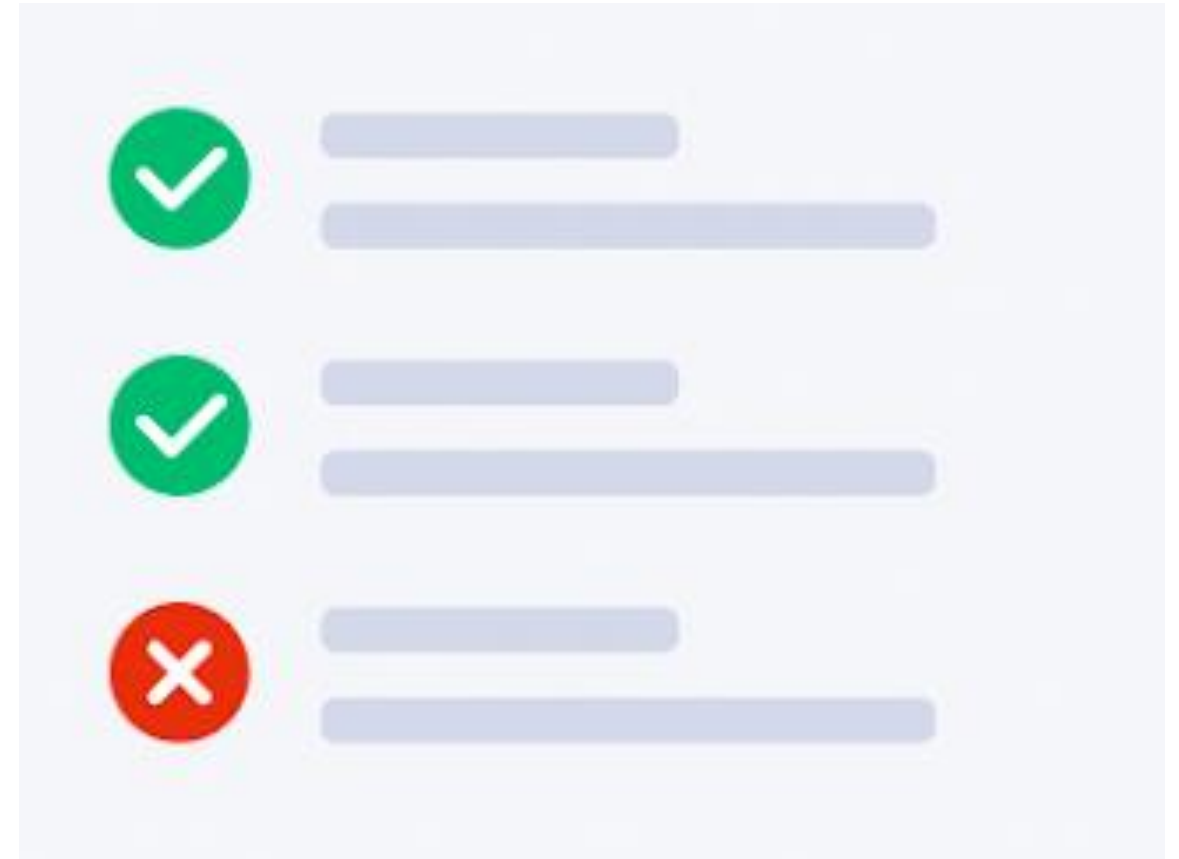
Authorization

Can you do that? verifying access to specific features or resources

Allow or deny the access to a specific resource or action.

Other terms with same meaning:

- access control/management
- client privileges



OWASP

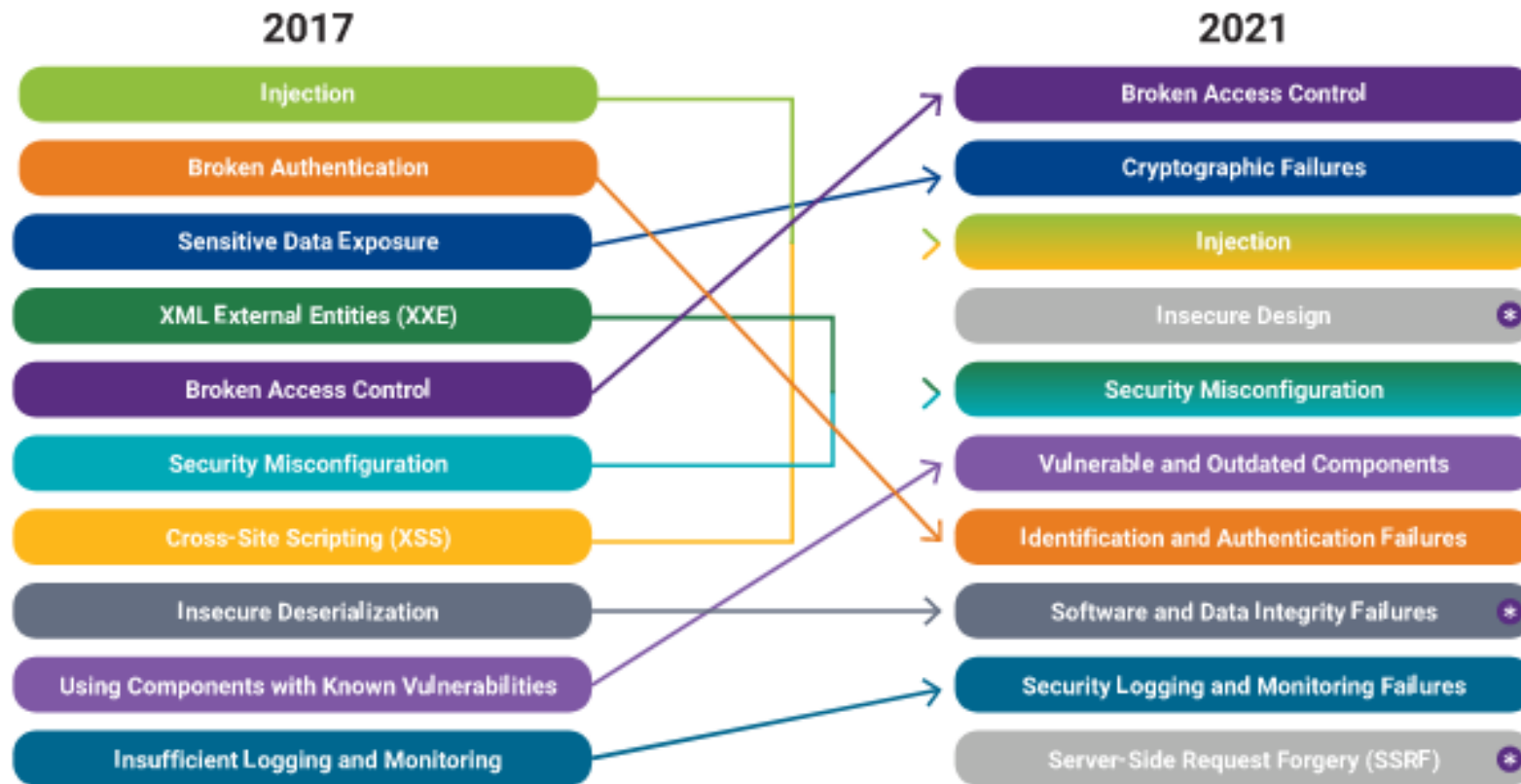
The Open Worldwide Application Security Project® (OWASP) is a nonprofit foundation that works to improve the security of software.

Definitive source of security tools and resources:

- OWASP Top 10
- OWASP Cheat Sheet Series
- OWASP Dependency Check
- ...



OWASP Top 10 (Vulnerabilities)

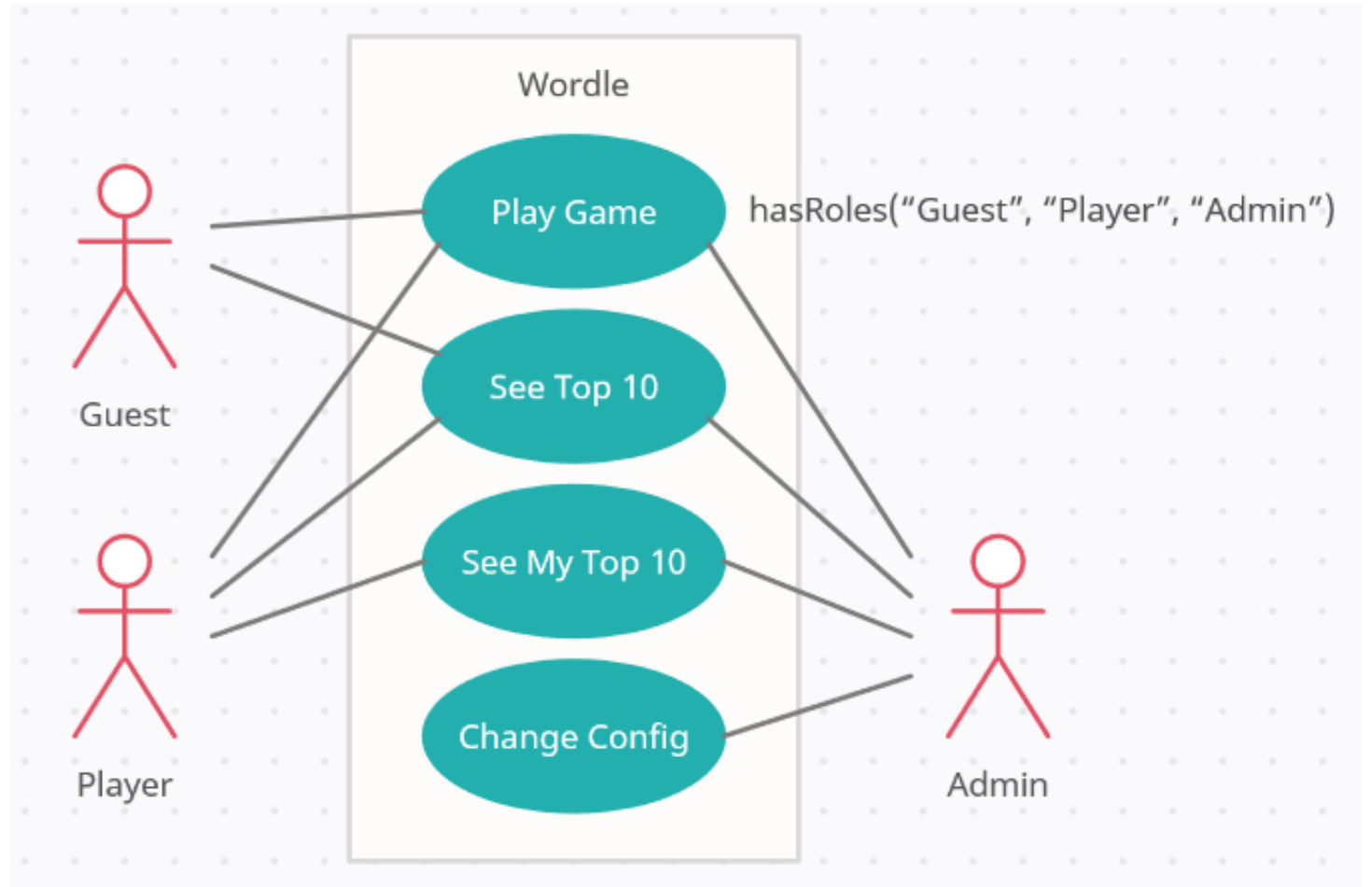


Access Control models

- Role-Based Access Control (RBAC)
- Resource-Based Access Control (RBAC)
- Attribute-Based Access Control (ABAC)
- Relations-Based Access Control (ReBAC)
Suitable for Social Networks -> Only friends can see the posts

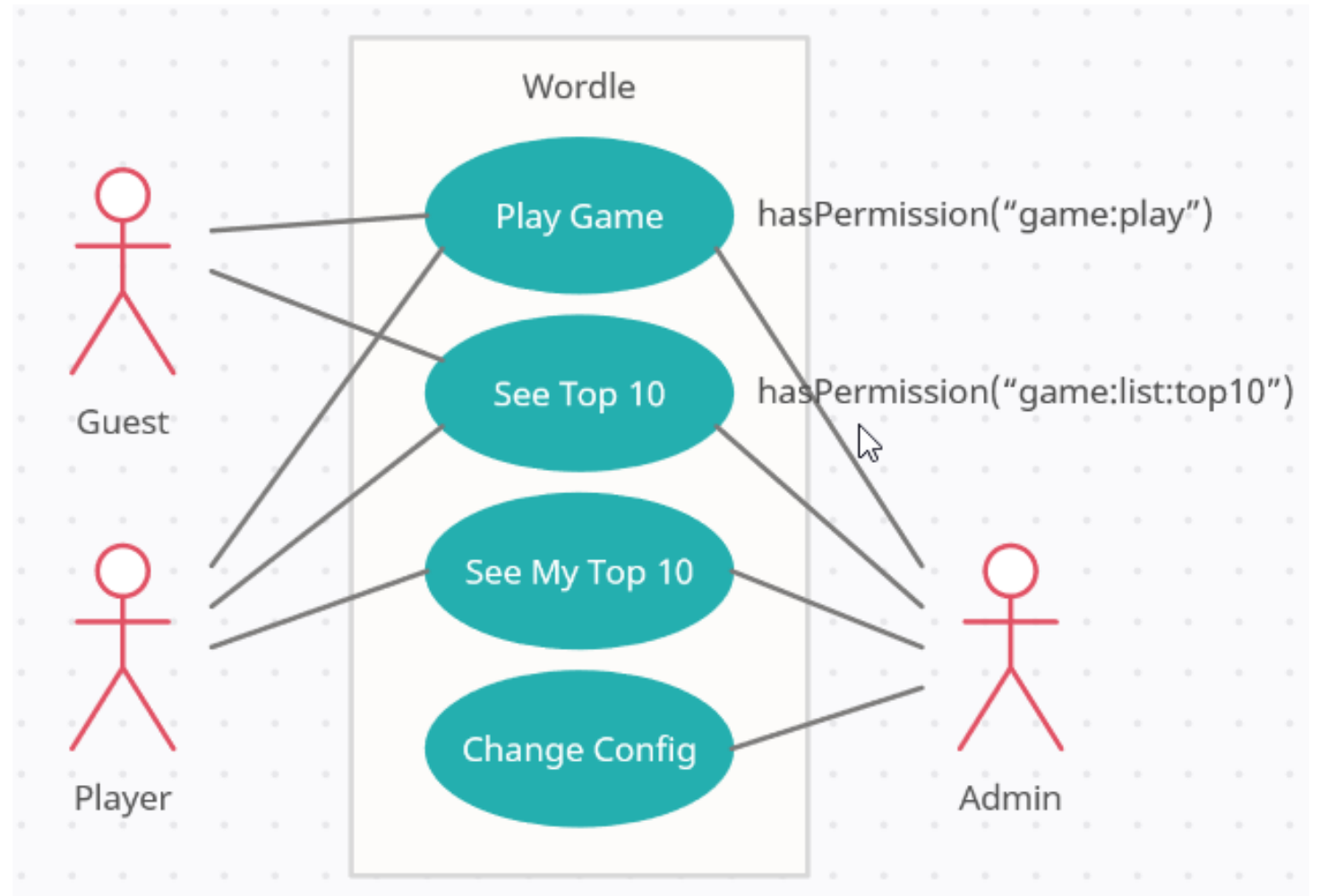
Role-Based Access Control

- Static checks against **hard-coded roles**
- User's roles are defined dynamically
- **Any change in what a role can do requires coding**



Resource-Based Access Control

- Static checks against **hard-coded permissions**
- Role's permissions are defined dynamically
- User's roles are defined dynamically

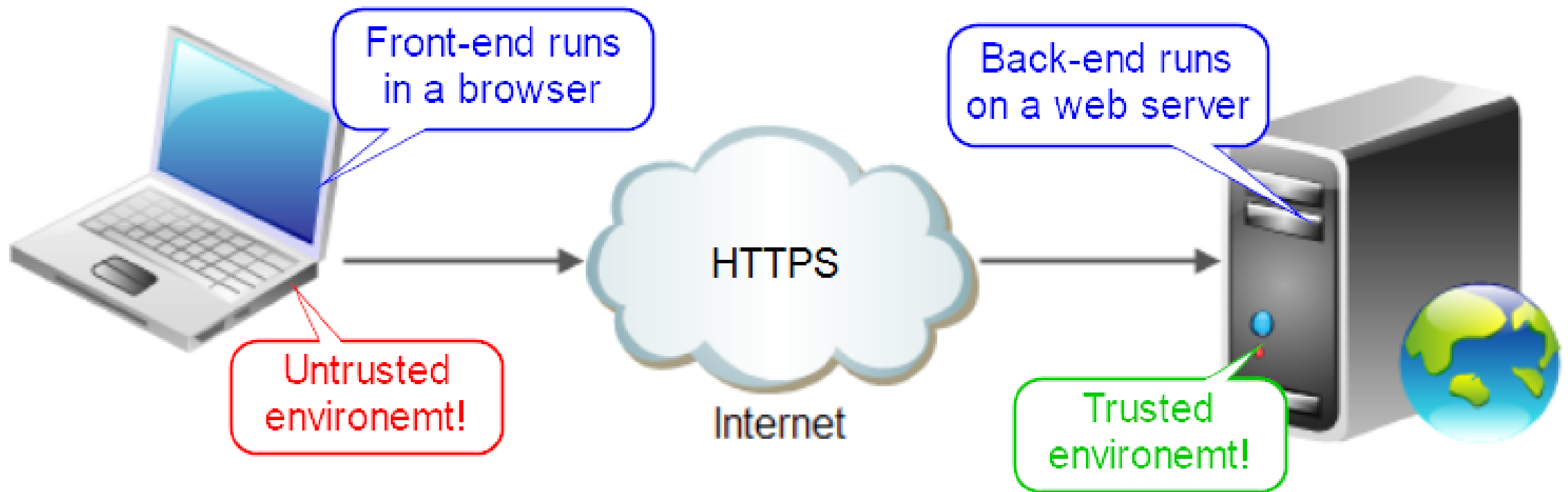


Attribute-Based Access Control (ABAC)

- Every Subject, Resource, Action or Environment could have attributes.
- Access control is defined using policies defining if access is granted/denied for a set of attributes
 - Subject's "job role" = "communications"
 - Subject's "business unit" = "marketing"
 - Action = "edit"
 - Resource "type" = "media strategy document"
 - Resource "business unit" = "marketing"
- More flexible than RBAC, but more complex to implement

Web Application Architecture

Nowadays applications use Client-Side Rendering, i.e., JavaScript running in a Browser (aka front-end) calling APIs on the back-end



OWASP Application Security Verification Standard

- Verify that Authorization Checks are Performed in the Right Location
 - 1.4.1 Verify that trusted enforcement points, such as access control gateways, servers, and serverless functions, enforce access controls. Never enforce access controls on the client.
 - 4.1.1 Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed

Back-end authorization



Back-end authz with Shiro

- Organized around permissions, i.e. what a user can do in the application
- A permission depends on 3 parts divided by colon (last 1 is optional): **Resource, Action, Instance**

```
"permissions": [  
    "game:play",  
    "game:query:myTop10",  
    "config:read",  
    "config:update"  
],
```

The diagram illustrates the components of the permissions listed in the code block above. Three callout boxes are present:

- A box labeled "instance" points to the `myTop10` part of the `"game:query:myTop10"` permission.
- A box labeled "resource" points to the `config` part of the `"config:read"` and `"config:update"` permissions.
- A box labeled "action" points to the `read` and `update` parts of the `"config:read"` and `"config:update"` permissions, respectively.

Back-end authz with Shiro

- Declarative way with Java annotations at class or method level
- Programmatic (imperative) way streight in the code

```
@GetMapping  
@RequiresPermissions("config:read")  
public ConfigModel getConfig() {  
    return configService.getConfig();  
}
```

```
@GetMapping()  
public CollectionModel<GameModel> listLast10(@RequestParam String filter) {  
    return switch (filter) {  
        case "myTop10" -> {  
            SecurityUtils.getSubject().checkPermission("game:query:myTop10");  
            yield gameModelAssembler.toCollectionModel(gameService.listLast10());  
        }  
    }  
}
```

Front-end authorisation

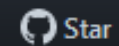
CASL^{v6}



**Isomorphic
Authorization
JavaScript library**

[Get Started](#)

[Source code](#)



Star

4,748

Front-end authz with CASL

- Organized around abilities, i.e. what a user can do in the application
- An ability depends on 4 parameters (last 3 are optional):
User Action, Subject, Fields, Conditions

```
export default defineAbility((can, cannot) => {  
  can('read', 'post');  
  can('update', 'post');  
  can('read', 'comment');  
  cannot('update', 'comment');  
});
```

action

subject
(resource)

Front-end authz with CASL

- CASL React – React components for CASL
- DSL like syntax, easy to read and understand
- Show/hide or pass allowed value

```
<!-- show/hide -->  
<Can I="create" a="post">  
  <button>Save</button>  
</Can>
```

```
<!-- pass allowed value -->  
<Can I="create" a="post" passThrough>  
  {allowed => <button disabled={!allowed}>Save</button>  
</Can>
```

What is wrong?

- If your JavaScript is prepared to handle all use cases then you have **all REST endpoints hard-coded in the front-end**
- The result: front-end source code becomes like Swagger UI for the back-end REST API

Sources Outline

▼ Main Thread

▼ localhost:3000

▼ static/js

bundle.js

▶ Webpack

▼ c://

▼ Dev/puspace/wordle-2023/src/main/front

node_modules

src

JS App.js

JS Can.js

```
45 function App() {
46   _s();
47   const [filter, setFilter] = (0,react__WEBPACK_IMPORTED_MODULE_0___useState)("top10");
48   const {
49     data: last10Model
50   } = (0,swr__WEBPACK_IMPORTED_MODULE_1___default())("/api/games?filter=${filter}");
51   const [last10Model, setLast10Model] = (0,react__WEBPACK_IMPORTED_MODULE_0___useState)(last10Model);
52   const [currentModel, setCurrentModel] = (0,react__WEBPACK_IMPORTED_MODULE_0___useState)(null);
53   const [currentModelId, setCurrentModelId] = (0,react__WEBPACK_IMPORTED_MODULE_0___useState)(null);
54   const [currentModelName, setCurrentModelName] = (0,react__WEBPACK_IMPORTED_MODULE_0___useState)(null);
55   const [currentModelDescription, setCurrentModelDescription] = (0,react__WEBPACK_IMPORTED_MODULE_0___useState)(null);
56   const [currentModelImage, setCurrentModelImage] = (0,react__WEBPACK_IMPORTED_MODULE_0___useState)(null);
57   const [currentModelVideo, setCurrentModelVideo] = (0,react__WEBPACK_IMPORTED_MODULE_0___useState)(null);
58   async function startNewGame() {
59     const response = await fetch("/api/games", {
60       method: "POST"
61     });
```

All the REST endpoint used in the code could be reverse engineered

What is wrong?

- If your JavaScript is prepared to handle all use cases then you have **all abilities listed there**
- The result: front-end source code becomes like **documentation for ALL permissions used in the back-end**

The screenshot shows a web browser's developer tools interface. The 'Sources' panel on the left lists the file structure, with 'bundle.js' selected under the 'static/js' directory. The 'Outline' panel is empty. The code editor on the right displays the contents of 'bundle.js', showing a React component with various permissions like 'read' and 'config' highlighted. A speech bubble points to these permissions with the text 'All the abilities used in the code could be reverse engineered'.

```
77 }, this), /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_6___jsxDEV)("div",
78   className: "d-flex justify-content-center gap-2",
79   children: [/*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_6___jsxDEV)(r
80     onClick: startNewGame,
81     children: "Start New Game"
82   ), void 0, false, {
83     fileName: _jsxFile
84     lineNumber: 39,
85     columnNumber: 9
86   }, this), /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_6___jsxDEV)(_C
87     I: "read",
88     a: "config",
89     children: /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_6___jsxDEV)(I
90     onClick: () => setShowConfig(true),
91     children: [/*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_6___jsxDEV
92       class: "bi bi-toggles"
93     ), void 0, false, {
94       fileName: _jsxFileName,
95       lineNumber: 43,
```

How to prevent exposing
sensitive information to the
browser that is not intended
for the current user?

HATEOS

Hypertext As The Engine Of Application State

На 31 март през...

- 1889 г. - Айфеловата кула (на снимката) е официално отворена
- 1916 г. - В 24 часа България обявява война от Юлиански календар към Германска империя и календар и вместо 1 април настъпва 1 април.
- The links in JSON say what could be done next
- 1953 г. - Делото срещу лекарите в СССР
- 1966 г. - Изстрелян е съветският Лунен апарат, влязъл в орбита около Луната

The links in HTML say what could be seen next

The links in JSON say what could be done next

s in HTML say
d be seen next

```
{  
  "id": "9d92-36da15064e55",  
  "score": "8:43:44.316607",  
  "alphabetMatches": "\u0434\u0444\u0433\u0445\u0438\u043a\u043b\u044c\u043f\u044c\u0437\u044c\u044b\u0446\u0436\u0431\u043d\u043c\u044e",  
  "maxGuesses": 11,  
  "guesses": [],  
  "state": "ONGOING",  
  "_links": {  
    "self": {  
      "href": "http://localhost:8080/api/games/4ecc5fae-23ef-4205-9d...",  
    },  
    "makeGuess": {  
      "href": "http://localhost:8080/api/games/4ecc5fae-23ef-4205-9d...",  
      "templated": true  
    }  
  }  
}
```

Authorization with HATEOS

Replace passing the granted permissions with passing links to the resources with granted access

The diagram illustrates the transformation of a permissions list into HATEOS links. On the left, a JSON object for a user named 'Wordle Admin' has a 'permissions' array containing four items: 'game:play', 'game:query:myTop10', 'config:read', and 'config:update'. On the right, the same user object is shown, but the 'permissions' array is replaced by a '_links' object. This object contains four entries, each mapping a permission to a specific API endpoint: 'logout' to 'http://localhost:8080/logout', 'startNewGame' to 'http://localhost:8080/api/games', 'top10' to 'http://localhost:8080/api/games?filter=top10', and 'myTop10' to 'http://localhost:8080/api/games?filter=myTop10'. Additionally, a 'self' link is provided for the user profile. Red arrows indicate the mapping from the permissions on the left to the corresponding links on the right.

```
1 {  
2   "id": 2,  
3   "name": "Wordle Admin",  
4   "permissions": [  
5     "game:play",  
6     "game:query:myTop10",  
7     "config:read",  
8     "config:update"  
9   ]  
10 }  
11
```

```
1 {  
2   "id": 2,  
3   "name": "Wordle Admin",  
4   "_links": {  
5     "logout": { "href": "http://localhost:8080/logout" },  
6     "startNewGame": { "href": "http://localhost:8080/api/games" },  
7     "top10": { "href": "http://localhost:8080/api/games?filter=top10" },  
8     "myTop10": { "href": "http://localhost:8080/api/games?filter=myTop10" },  
9     "readConfig": { "href": "http://localhost:8080/api/config" },  
10    "self": {  
11      "href": "http://localhost:8080/api/users/{userId}",  
12      "templated": true  
13    }  
14  }  
15 }  
16
```

Authorization with HATEOS

Get rid of all hard-coded URI in the front-end, except one entry point

```
12 function App() {
13   const [filter, setFilter] = useState("top10");
14   const { data: user } = useSWR("/api/users/current");
15   const { data: top10Model } = useSWR(`/api/games?filter=${filter}`);
16   const top10 = top10Model?._embedded.gameModelList;
17
18   const [showSignIn, setShowSignIn] = useState(false);
19   const [showConfig, setShowConfig] = useState(false);
20
21   const navigate = useNavigate();
22   async function startNewGame() {
23     const response = await fetch("/api/games", { method: "POST" });
24     const game = await response.json();
25     navigate(`/games/${game.id}`);
26   }
27
28   async function signOut() {
29     await fetch("/logout");
30     mutate("/api/users/current");
31   }
```

```
11 function App() {
12   const [filter, setFilter] = useState("top10");
13   const { data: user } = useSWR("/api/users/current");
14   const { data: top10Model } = useSWR(user?._links[filter].href);
15   const top10 = top10Model?._embedded.gameModelList;
16
17   const [showSignIn, setShowSignIn] = useState(false);
18   const [showConfig, setShowConfig] = useState(false);
19
20   const navigate = useNavigate();
21   async function startNewGame() {
22     const response = await fetch("/api/games", { method: "POST" });
23     const game = await response.json();
24     navigate(`/games/${game.id}`);
25   }
26
27   async function signOut(link) {
28     await fetch(link.href);
29     mutate("/api/users/current");
30   }
```


Authorization with HATEOS

Get rid of @casl/react and all hard-coded

```
<Can I="..." a="..." />
```

```
38
39 <Can I="read" a="config">
40   <>
41     <Button onClick={() => setShowConfig(true)}>
42       <i class="bi bi-toggles"></i> Config
43     </Button>
44     <ConfigPopup
45       show={showConfig}
46       onHide={() => setShowConfig(false)}
47     />
48   </>
49 </Can>
50
74 <Can I="query" a="game" field="myTop10">
75   <Nav.Item>
76     <Nav.Link eventKey="myTop10">My Top 10</Nav.Link>
77   </Nav.Item>
78 </Can>
```

```
37 {user?._links.readConfig && (
38   <>
39     <Button onClick={() => setShowConfig(true)}>
40       <i class="bi bi-toggles"></i> Config
41     </Button>
42     <ConfigPopup
43       show={showConfig}
44       onHide={() => setShowConfig(false)}
45       link={user?._links.readConfig}
46     />
47   </>
48   </>
49 )}
50
86 {user?._links.myTop10 && (
87   <Nav.Item>
88     <Nav.Link eventKey="myTop10">My Top 10</Nav.Link> Link<
89   </Nav.Item>
90 )}
```

More secure

- **No hard-coded permissions**
Nobody can reverse engineer what functionality your application provide over what he is granted to see
- **No hard-coded links**
Nobody can reverse engineer your application APIs over the APIs he is granted to use
- **The decision what the front-end can see is taken entirely in the trusted back-end**

Q & A

