42 crunch

Pragmatic Web Security
Security for developers

NEW YEAR'S
FITNESS RESOLUTION

OWASP API SECURITY TOP 10

Find, Fix and Secure your APIs

JAN 25, FEB 17 & MAR 24
3-PART WEBINAR SERIES

Dr. Philippe de Ryck
Web Security Expert
Pragmatic Web Security

Colin Domoney
Security Researcher
& Developer Advocate
42Crunch

Introduction

About our Speakers

Colin Domoney

API Security Research Specialist & Developer Advocate

Editor of APISecurity.io

42Crunch

Dr. Philippe De Ryck

Web Security Expert

Pragmatic Web Security

Housekeeping Rules

- All attendees muted
- Questions via chat window
- Recording will be shared on-demand
- Polling questions

1	Broken object level authorization
2	Broken user authentication
3	Excessive data exposure
4	Lack of resources & rate limiting
5	Broken function level authorization
6	Mass assignment
7	Security misconfiguration
8	Injection
9	Improper assets management
10	Insufficient logging & monitoring

OWASP

API Security

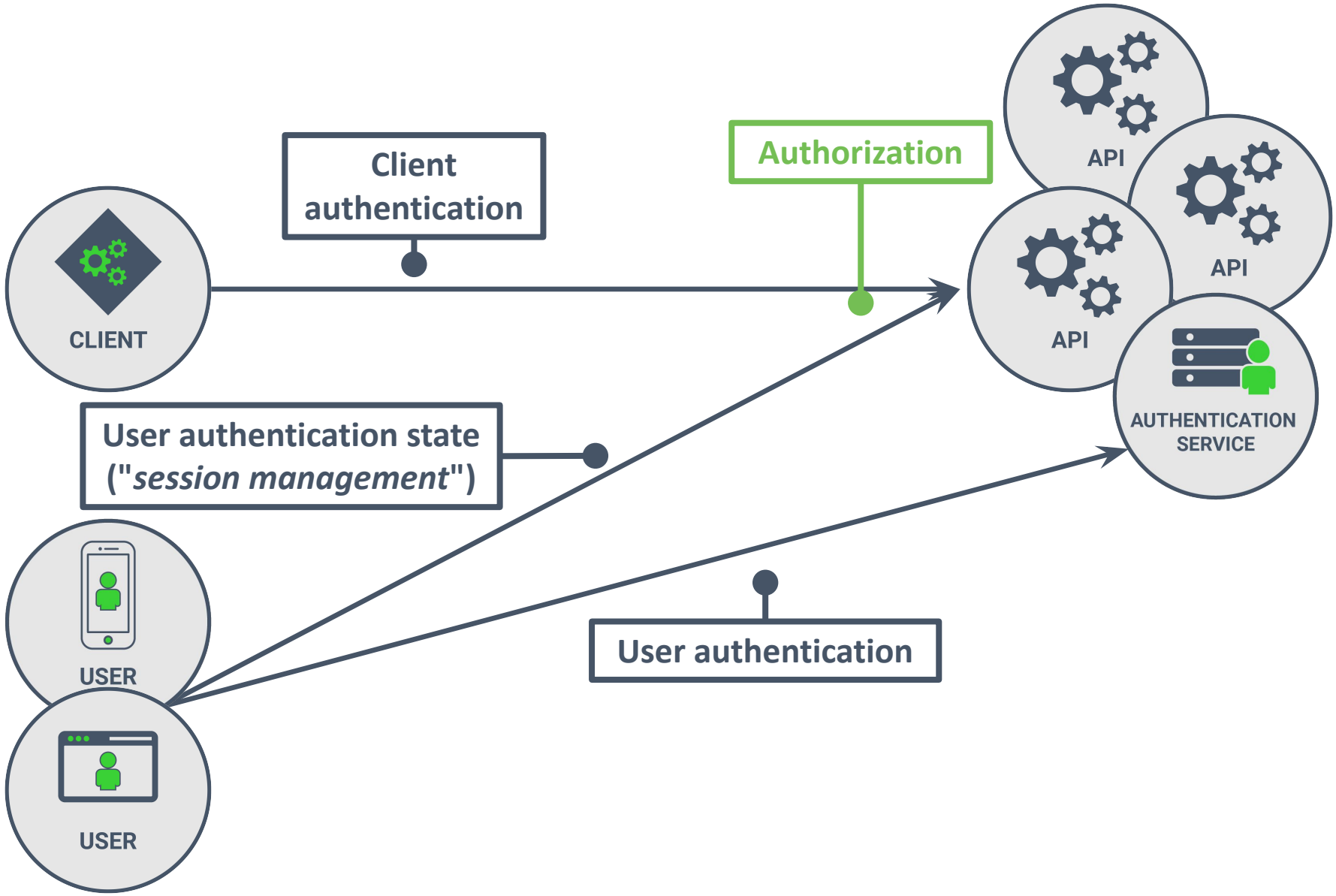
TOP 10

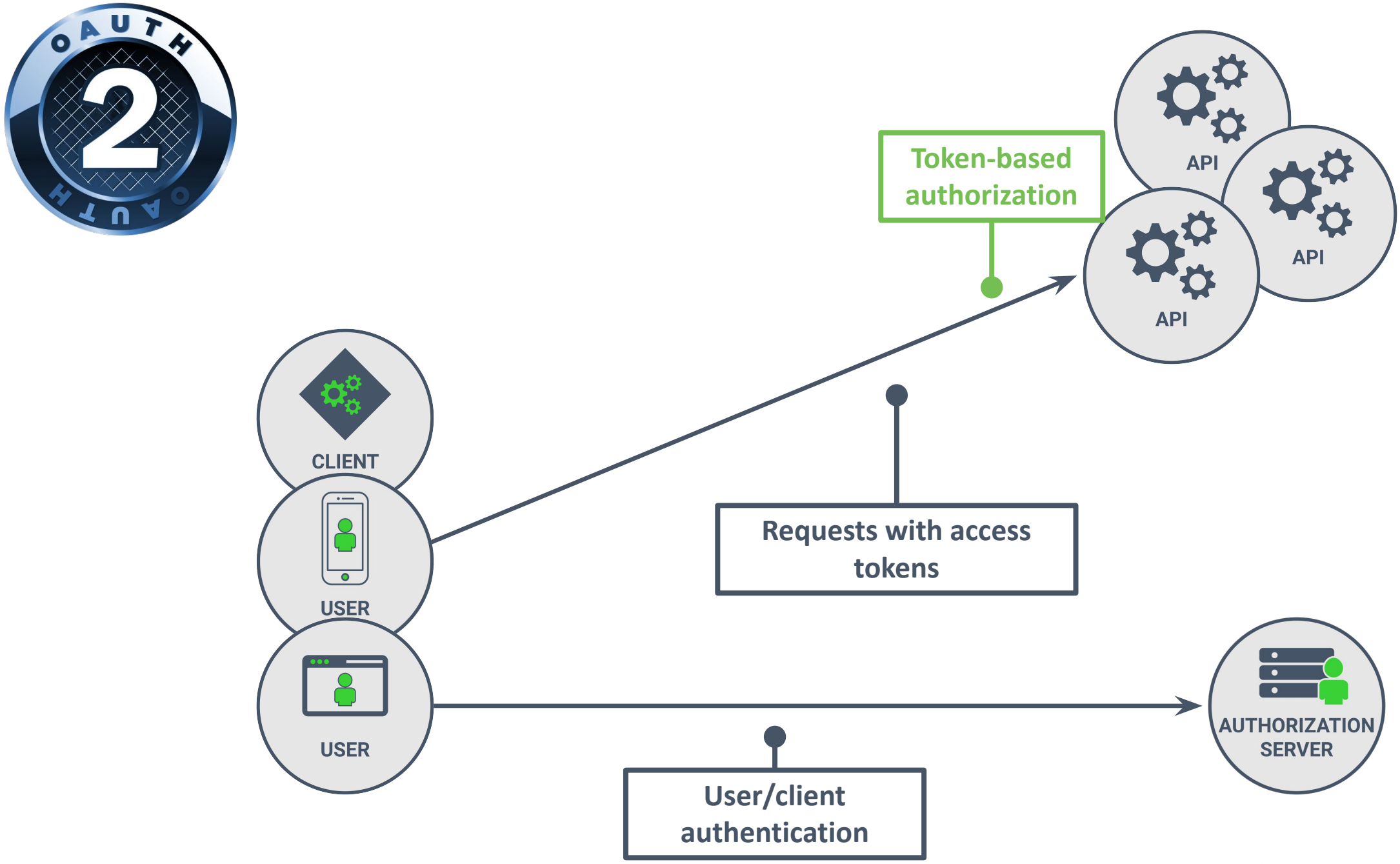
1	Broken object level authorization
2	Broken user authentication
3	Excessive data exposure
4	Lack of resources & rate limiting
5	Broken function level authorization
6	Mass assignment
7	Security misconfiguration
8	Injection
9	Improper assets management
10	Insufficient logging & monitoring

OWASP

API Security

TOP 10


Client authentication
Authorization
API
API
API
CLIENT
AUTHENTICATION SERVICE
User authentication state ("*session management*")
USER
USER
User authentication



crunch

Pragmatic Web Security
Security for developers

Introduction to OAuth 2.0 and

courses.pragmaticwebsecurity.com/courses/introduction-to-oauth-2-0-and-openid-connect

Pragmatic Web Security

Security for developers

SIGN IN

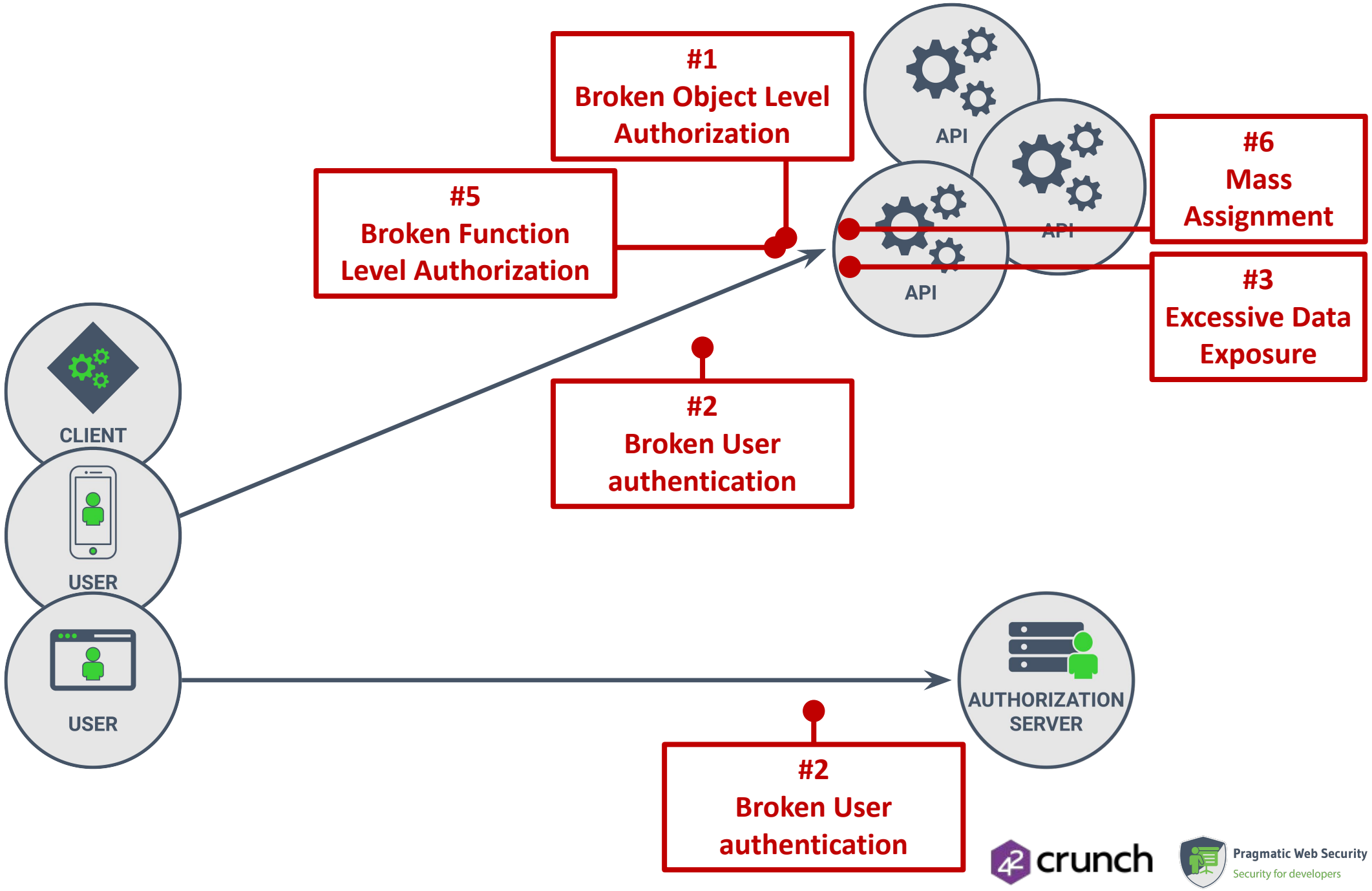
Introduction to OAuth 2.0 and OpenID Connect

An overview of common use cases and current best practices

Everyone who first learns about OAuth 2.0 and OpenID Connect is confused. There are dozens of specifications with uncommon terminology and hard-to-understand scenarios. Eventually, you will have a working implementation, but questions remain. Why use the complicated redirect, instead of just a custom login form? Is this the right flow for my application? Where do I store tokens, and how can I protect them?

This introduction course helps you clear up the confusion surrounding OAuth 2.0 and OpenID Connect.

https://courses.pragmaticwebsecurity.com/courses/introduction-to-oauth-2-0-and-openid-connect


#1
Broken Object Level Authorization
#5
Broken Function Level Authorization
#6
Mass Assignment
#3
Excessive Data Exposure
#2
Broken User authentication
#2
Broken User authentication
API
API
API
CLIENT
USER
USER
AUTHORIZATION SERVER
crunch
Pragmatic Web Security
Security for developers

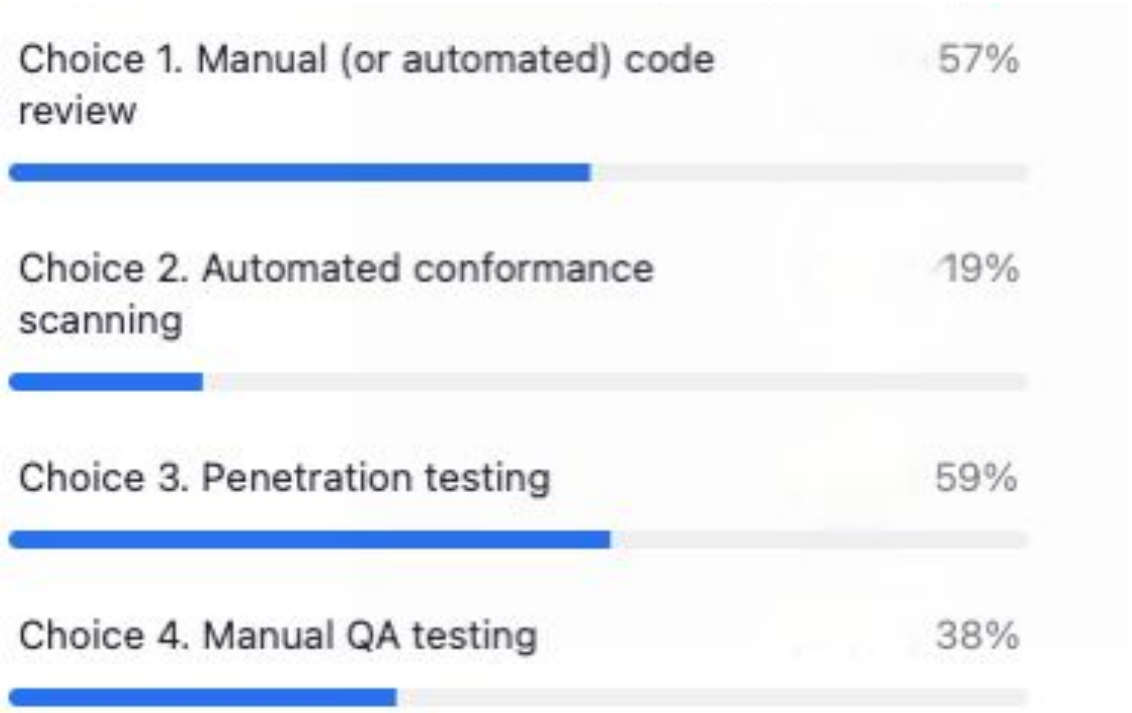
Polling Question 1: Multiple Choice

How are you testing/checking for authentication and/or authorization vulnerabilities?

1. Manual (or automated) code review
2. Automated conformance scanning
3. Penetration testing
4. Manual QA testing

Polling Question 1: Multiple Choice

How are you testing/checking for authentication and/or authorization vulnerabilities?



Olatunde Michael Garuba FOLLOW
Full stack Javascript Developer

Build Node.js RESTful APIs in 10 Minutes

Published Jan 12, 2017 Last updated Aug 18, 2017

https://www.codementor.io/@olatundegaruba/nodejs-restful-apis-in-10-minutes-q0sgsfhbd

A REST API endpoint without any authorization

```
1  app.delete('/tasks/:taskId', function(req, res) {
2    Task.remove({
3      _id: req.params.taskId
4    }, function(err, task) {
5      if (err)
6        res.send(err);
7      res.json({ message: 'Task successfully deleted' });
8    });
9  };
```

A REST API endpoint restricted to authenticated users only

```
1  app.delete('/tasks/:taskId', requiresAuth, function(req, res) {
2    Task.remove({
3      _id: req.params.taskId
4    }, function(err, task) {
5      if (err)
6        res.send(err);
7      res.json({ message: 'Task successfully deleted' });
8    });
9  };
```

Permissions on an endpoint do not suffice to stop *broken object-level authorization* (#1)

Checking permissions helps prevent *broken function-level authorization* (#5)

A REST API endpoint restricted to users with the specific "deleteTask" permission

```
1  app.delete('/tasks/:taskId', auth.hasPermission('deleteTask'), function(req, res) {
2    Task.remove({
3      _id: req.params.taskId
4    }, function(err, task) {
5      if (err)
6        res.send(err);
7      res.json({ message: 'Task successfully deleted' });
8    });
9  };
```

TOP 10

Enforce authorization at the function level

By applying a sensible permission/role check to every endpoint, unauthorized requests can be rejected before they reach the application logic

A REST API endpoint to get a task

```
1  app.get('/tasks/:taskId', function(req, res) {
2    Task.findById(req.params.taskId, function(err, task) {
3      if (err)
4        res.send(err);
5      res.json(task);
6    });
7  };
```

A REST API endpoint to delete a task

```
1  app.delete('/tasks/:taskId', auth.hasPermission('deleteTask'), function(req, res) {
2    Task.remove({
3      _id: req.params.taskId
4    }, function(err, task) {
5      if (err)
6        res.send(err);
7      res.json({ message: 'Task successfully deleted' });
8    });
9  };
```

A REST API endpoint to get a task

```
1  app.get('/tasks/:taskId', auth.allowPublicAccess(), function(req, res) {
2    Task.findById(req.params.taskId, function(err, task) {
3      if (err)
4        res.send(err);
5      res.json(task);
6    });
7  };
```

A REST API endpoint to delete a task

```
1  app.delete('/tasks/:taskId', auth.hasPermission('deleteTask'), function(req, res) {
2    Task.remove({
3      _id: req.params.taskId
4    }, function(err, task) {
5      if (err)
6        res.send(err);
7      res.json({ message: 'Task successfully deleted' });
8    });
9  };
```

TOP 10

Empower auditability

Simplify the auditing of your authorization policy by making authorization logic explicit, even when endpoints have no specific authorization requirements.

#1 Broken Object Level Authorization
#2 Broken User Authentication
#5 Broken Function Level Authorization

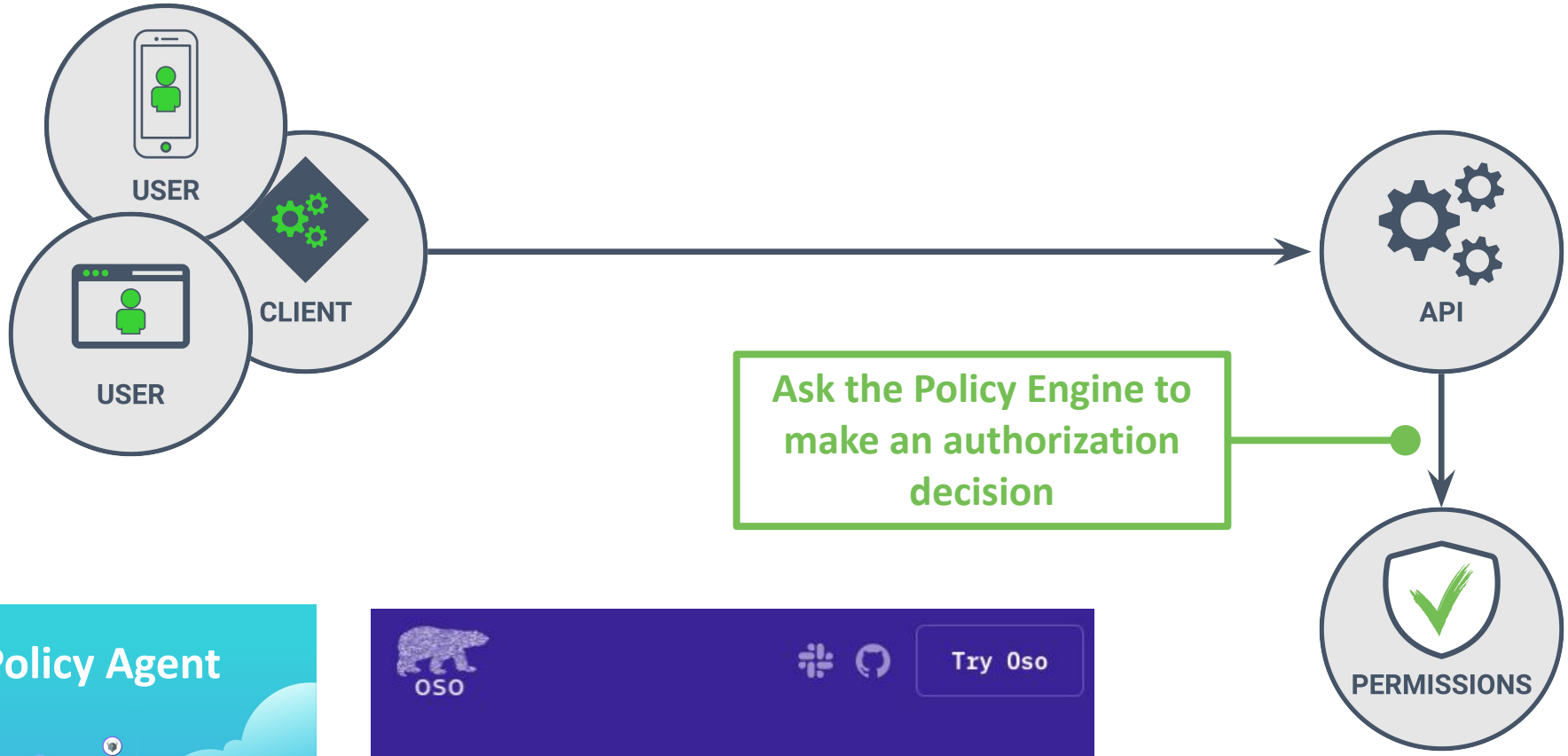
A permission check only allows authorized users to access this endpoint

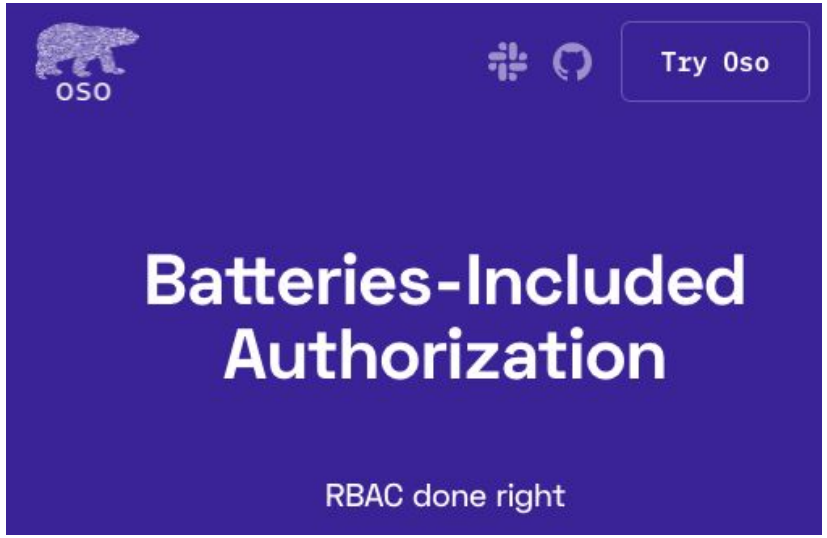
Object-level access control is often challenging to implement

```
1  app.delete('/tasks/:taskId', auth.hasPermission('deleteTask'), async (req, res) => {
2    let origTask = await Task.findById(req.params.taskId)
3
4    if(auth.hasRole('employee') && !origTask.owner.id == auth.currentUser.id)
5      res.status(403).send(
6        { message: 'You are not a manager. You can only delete your own tasks.'});
7
8    // Delete task
9  });
```

Policies scattered throughout the code like this are impossible to audit for security

Certain roles require additional restrictions, such as task ownership





https://www.openpolicyagent.org/
https://www.osohq.com/

TOP 10

Encapsulate complex authorization logic

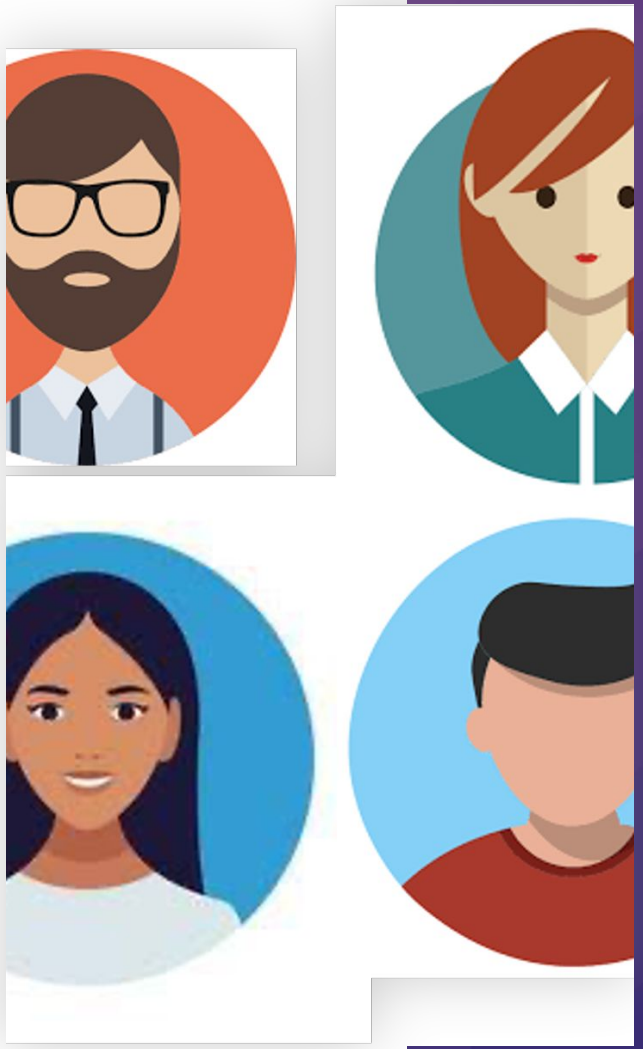
Complex authorization logic should not be scattered throughout the code, but is best defined in a clear and understandable authorization policy

#1 Broken Object Level Authorization
#5 Broken Function Level Authorization

Polling Question 2: Multiple Choice

What Authorization Framework/Library/Stack are you using?

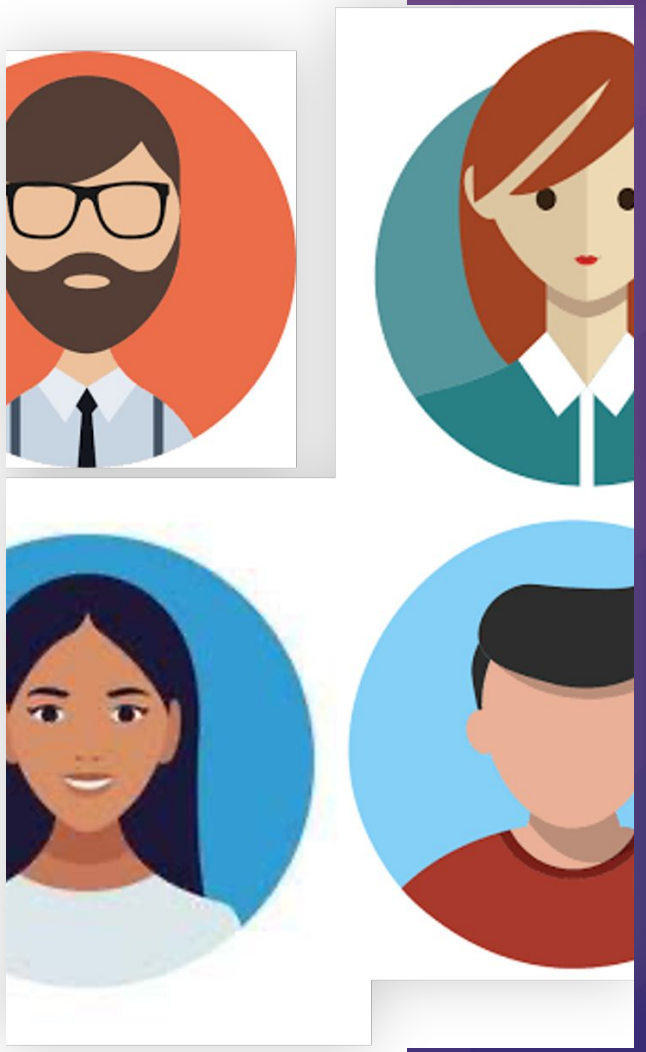
1. Roll our own
2. Framework middleware (ASP.Net Core, Spring, etc)
3. Open Policy Agent (OPA)
4. OSOHq

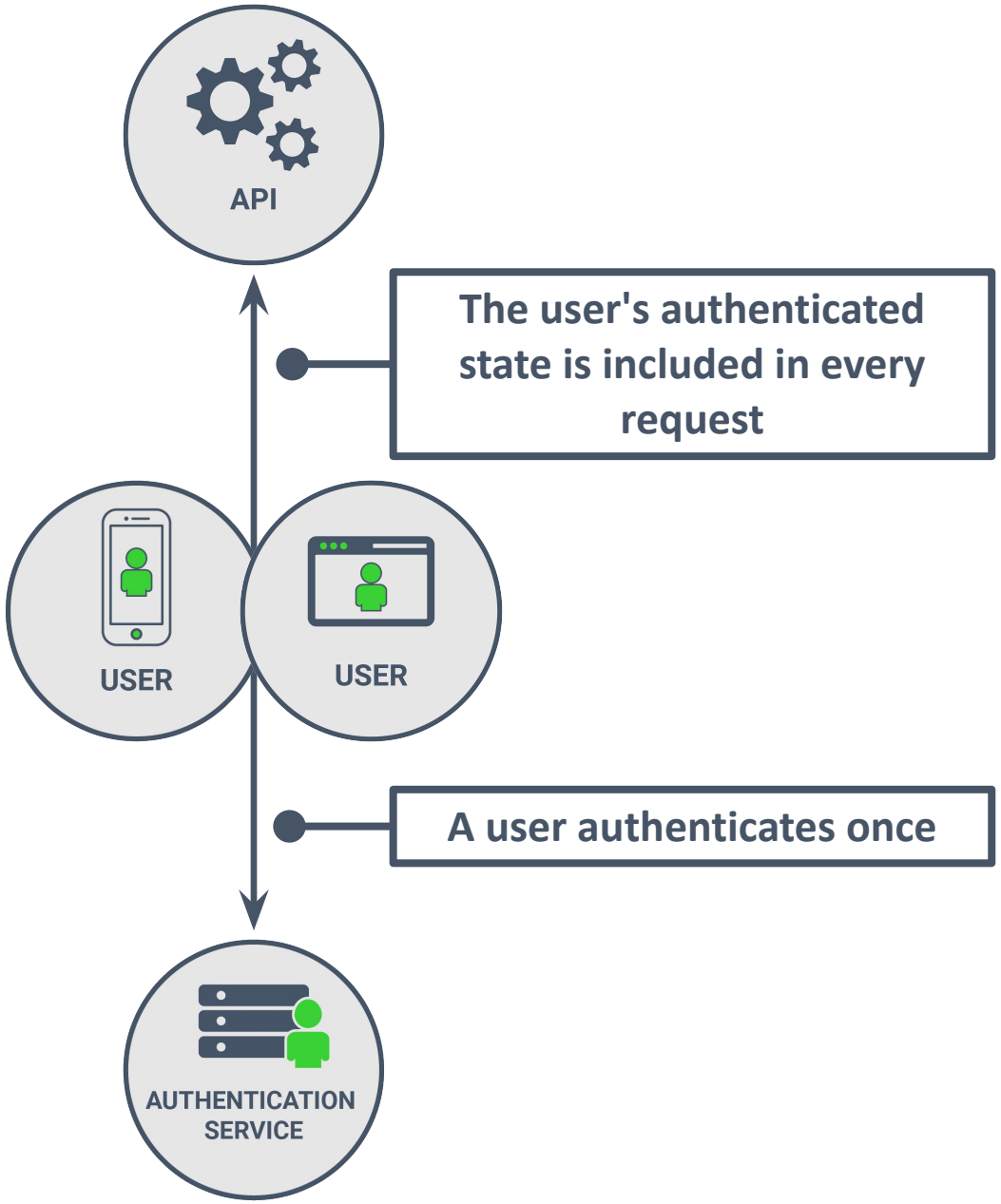


Polling Question 2: Multiple Choice

What Authorization Framework/Library/Stack are you using?

Choice 1. Roll our own	35%
Choice 2. Framework middleware (ASP.Net Core, Spring, etc)	76%
Choice 3. Open Policy Agent (OPA)	13%
Choice 4. OSOHq	3%




API
The user's authenticated state is included in every request
USER
USER
A user authenticates once
AUTHENTICATION SERVICE

Cookie:

JSESSIONID=DCDA…3C06

Java Session Handling on Heroku

English — 日本語に切り替える

Last updated December 16, 2019

Table of Contents

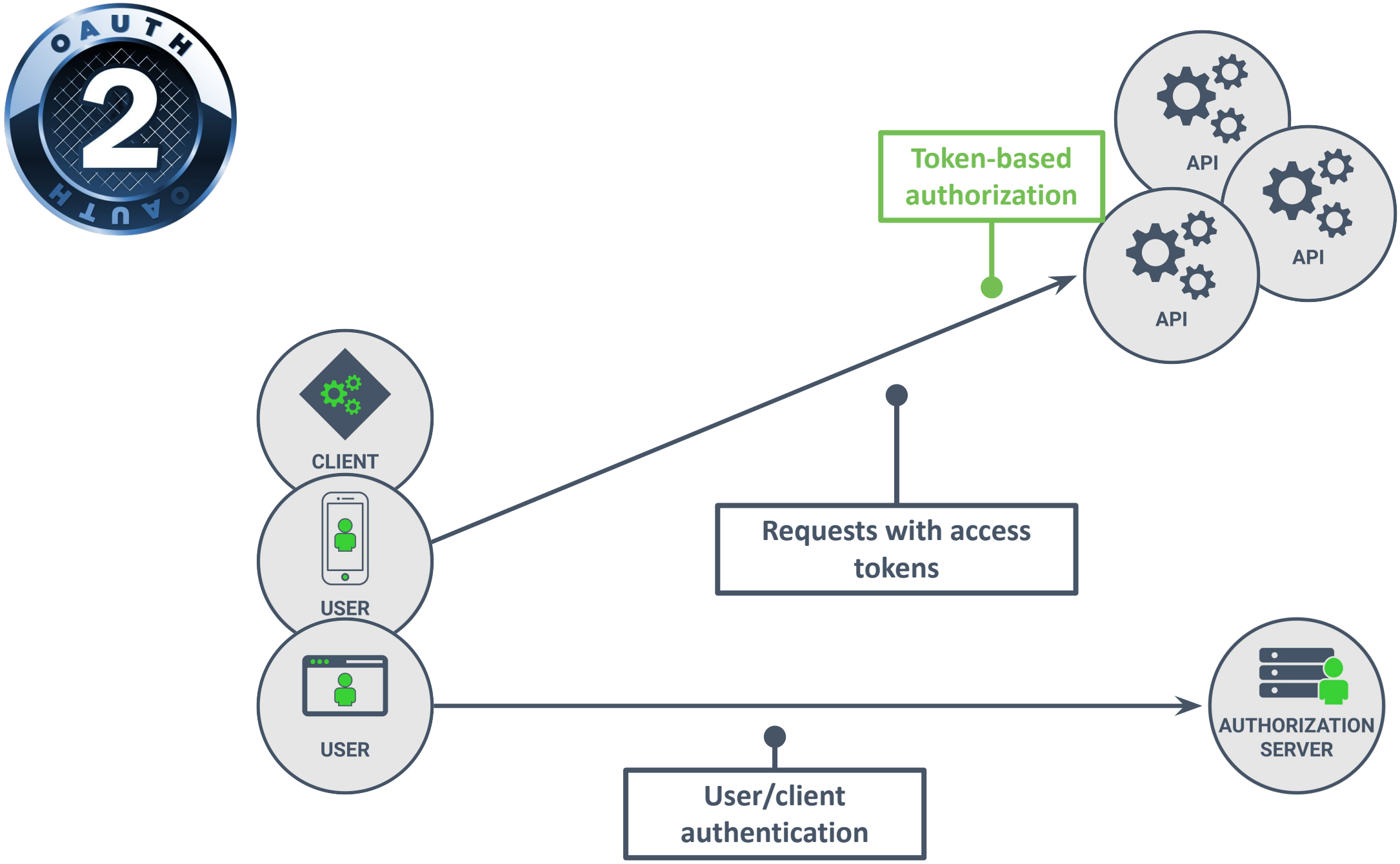
HTTP is a stateless protocol, but most applications need to preserve certain information across requests, such as login state or the contents of a shopping cart. This kind of state is usually stored in a session.

- https://devcenter.heroku.com/articles/java-session-handling-on-heroku

TOP 10

Traditional web sessions are still valid

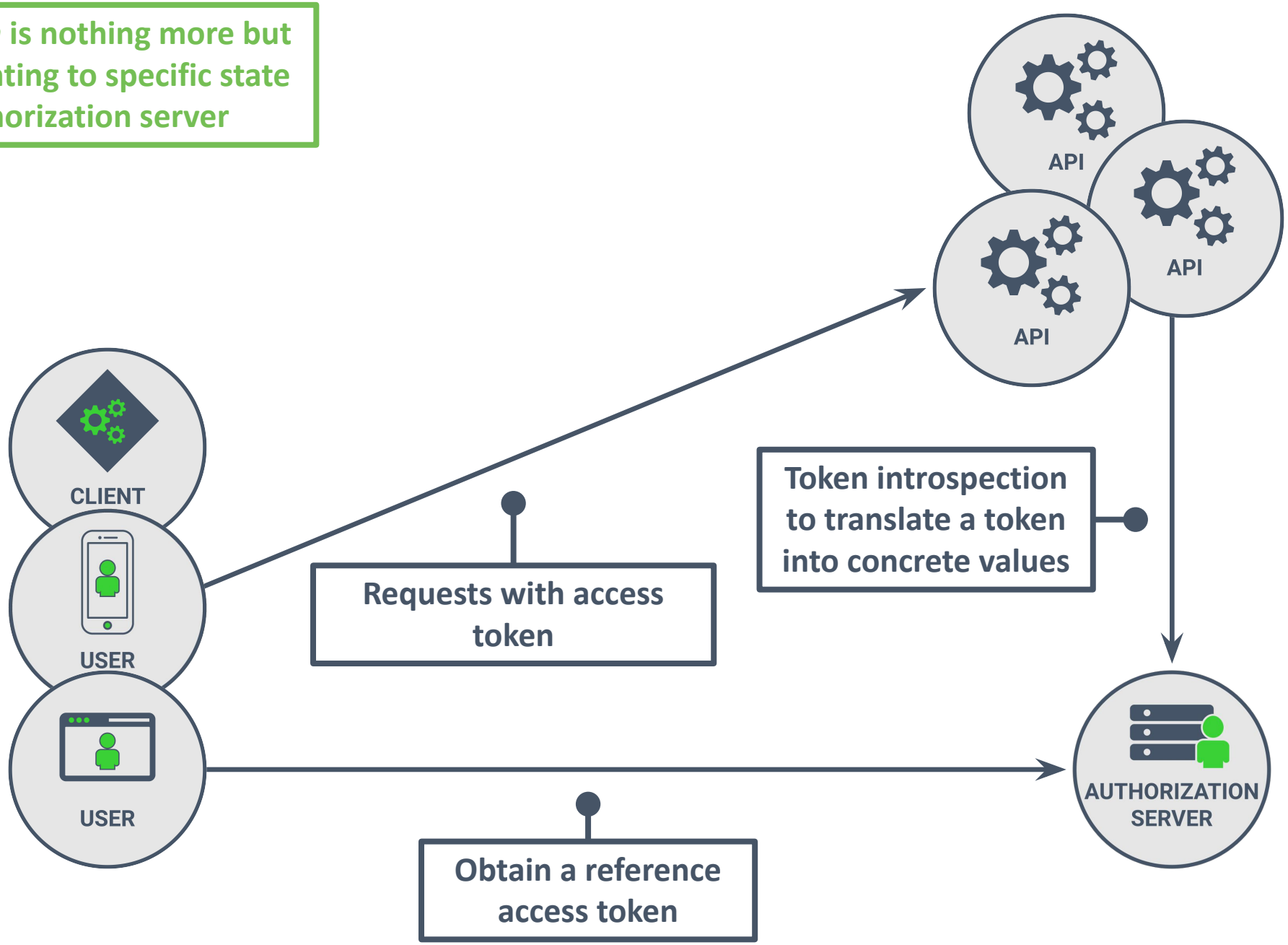
Browser-based applications without extreme scalability needs can rely on traditional cookies to propagate authentication state.



crunch

Pragmatic Web Security
Security for developers

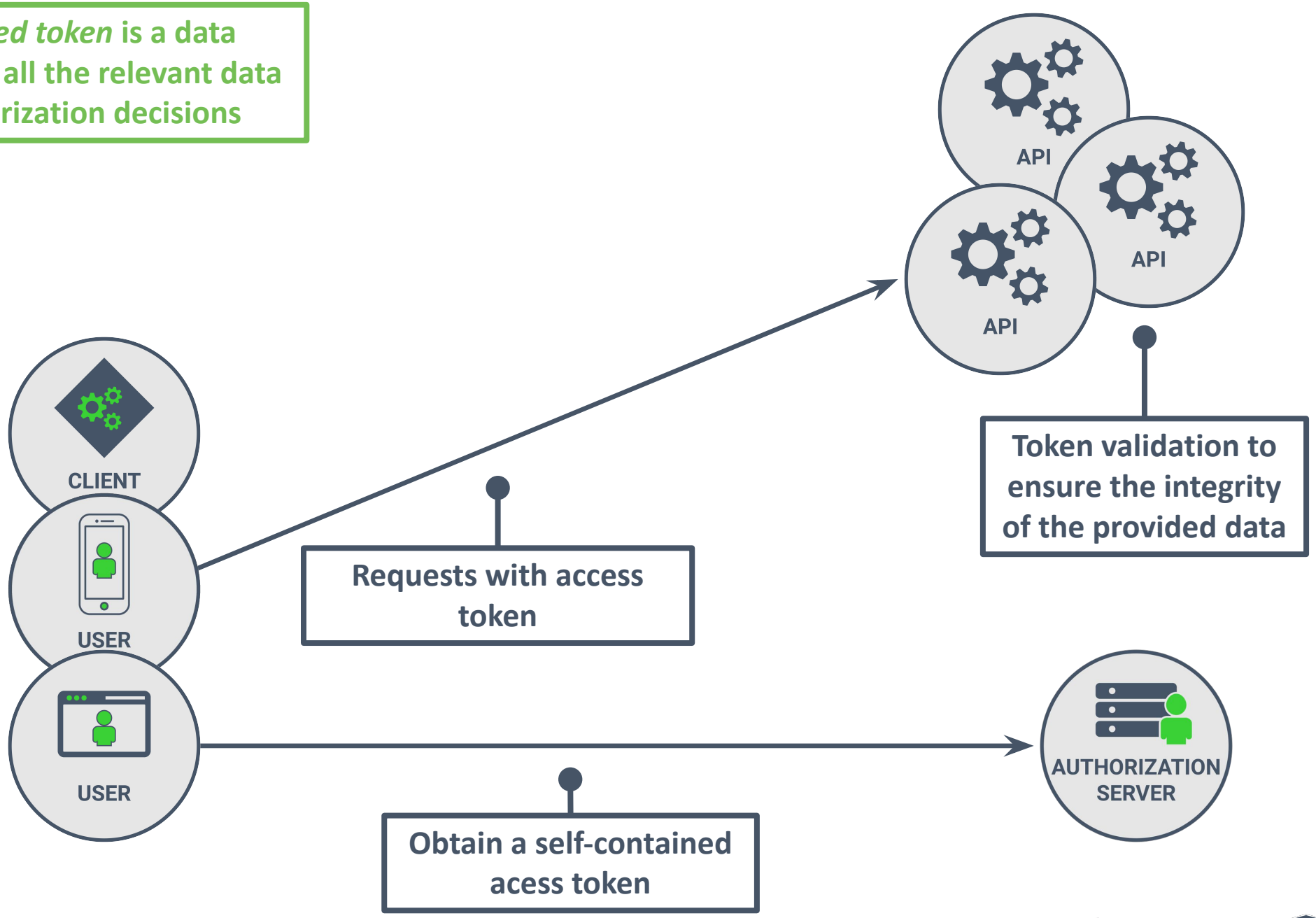
A *reference token* is nothing more but an identifier, pointing to specific state kept the authorization server



crunch

Pragmatic Web Security
Security for developers

A *self-contained token* is a data structure holding all the relevant data to make authorization decisions



crunch

Pragmatic Web Security
Security for developers

TOP 10

Analyze your security requirements

Self-contained tokens are hard to revoke automatically, but reference tokens induce a lot more overhead.

Understanding your requirements is essential for making the right decision.

Attendee Questions:

“Any hints you can share on how to draw the line on when to refactor/upgrade the authorization component to things like OPA and OSO?”

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtp
ZCI6Ik5UVkJPVFUzTXpCQk9FVXdOemhCUTBWR01r
UTBRVVU1UVRZeFFVVXlPVU5FUVVVeE5qRXlNdyJ9
.eyJpc3MiOiJodHRwczovL3N0cy5yZXN0b2dyYWR
lLmNvbS8iLCJzdWIiOiJhdXRoMHw1ZWI5MTZjMjU
4YmRiNTBiZjIwMzY2YzYiLCJhdWQiOlsiaHR0cHM
6Ly9hcGkucmVzdG9ncmFkZS5jb20iLCJodHRwczo
vL3Jlc3RvZ3JhZGUuZXUuYXV0aDAuY29tL3VzZXJ
pbmZvIl0sImlhdCI6MTU4OTc3NTA3MiwiZXhwIjo
xNTg5ODYxNDcyLCJhenAiOiJPTEtObjM4OVNVSW1
1ZkV4Z1JHMVJpbExTZ2RZeHdFcCIsInNjb3BlIjo
ib3BlbmlkIHByb2ZpbGUgZW1haWwgb2ZmbGluZV9
hY2Nlc3MifQ.XzJOXtTXOGOSbCFvp4yZGJzh7XhM
mOmI2XxtjWdlODz_siI-u8h11elcr8LwX6-
hL20QOW0eStzBzmm1FM_tS7MxuKkYx8QlTWOURPe
mbVKZOhNi8kN-
1j0pyc0uzve7Jib5vcxmkPwqpcVDFACgP85_0NYe
4zXHKxCA5_8VOn05cRCDSkNMTFzGJCT9ipCcNXaV
GdksojYGqQzezjpzzzwrtPEkiyFLFtDPZAl0MleF
3oFAOCBK0UKuNjJ_cSBbUsaIwfvK0WH47AwFrRn_
TxL4S1P3j3b1GgBm8tAqXysY84VZu0
rSg3zrZj1PnoqPD4mbOXds20xafCr9wR4WTQ

The decoded header of a JWT access token

```
1 {
2   "alg": "PS256",
3   "typ": "at+jwt",
4   "kid": "NTVBOTU3MzBBOEUUU5QTYxQUUyOUNEQUUxNjEyMw"
5 }
```

The decoded payload of a JWT access token

```
1 {
2   "iss": "https://sts.restograde.com",
3   "aud": "https://api.restograde.com",
4   "sub": "2262430d-c9cb-484f-9770-805893ff9518",
5   "iat": "1516239022",
6   "exp": "1516249022",
7   "permissions": ["reviews:read", "reviews:write"],
8   …
9 }
```

Internet Engineering Task Force (IETF)
Request for Comments: 8725
BCP: 225
Updates: 7519
Category: Best Current Practice
ISSN: 2070-1721

Y. Sheffer
Intuit
D. Hardt
M. Jones
Microsoft

February 2020

JSON Web Token Best Current Practices

Abstract

JSON Web Tokens, also known as JWTs, are URL-safe JSON-based security tokens that contain a set of claims that can be signed and/or encrypted. JWTs are being widely used and deployed as a simple security token format in numerous protocols and applications, both in the area of digital identity and in other application areas. This Best Current Practices document updates RFC 7519 to provide actionable guidance leading to secure implementation and deployment of JWTs.

7 Ways to Avoid JWT Security P

42crunch.com/7-ways-to-avoid-jwt-pitfalls/

Docs Support Login

crunch Why 42Crunch Platform Solutions Resources Company Get a demo

42CRUNCH BLOG

7 Ways to Avoid JWT Security Pitfalls

Posted on December 22, 2021 by Mark Dolan

Share:

Posted in 42Crunch Knowledge Series

Dec 22nd 2021. Author: Dr. Philippe de Ryck, Pragmatic Web Security,

https://42crunch.com/7-ways-to-avoid-jwt-pitfalls/

TOP 10

Religiously follow JWT security best practices

Insecure JWT handling is extremely common.

Encapsulate this behavior in a reusable component which is vetted for security.

The API response to retrieve online users

```
[
  {
    "id": 3,
    "name": "Hugh",
    "address": "5 George's Dock, …",
  },
  {
    "id": 6,
    "name": "Colin",
    "address": "71-75 Shelton Street, …",
  },
  {
    "id": 17,
    "name": "Philippe",
    "address": "Holsbeeksesteenweg 143, …",
  }
]
```

crunch

Pragmatic Web Security
Security for developers

The Java Spring endpoint returning users

```
1  @RequestMapping(path = "/online/users", method = GET, produces = "application/json")
2  public ResponseEntity<Object> getOnlineUsers() {
3    List<User> users = UserService.getOnlineUsers();
4    return new ResponseEntity<Object>(users, HttpStatus.OK);
5  }
```

The User data class

```
1  public class User {
2    private String id, name, address;
3    …
4    public String getName() {
5      return name;
6    }
7
8    public String getAddress() {
9      return address;
10   }
11 }
```

Data fields are automatically translated to JSON, even when they are not supposed to be exposed

The Java Spring endpoint returning users

```
1  @RequestMapping(path = "/online/users", method = GET, produces = "application/json")
2  public ResponseEntity<Object> getOnlineUsers() {
3    List<User> users = UserService.getOnlineUsers();
4    return new ResponseEntity<Object>(users, HttpStatus.OK);
5  }
```

The User data class

```
1  public class User {
2    private String id, name, address;
3    …
4    public String getName() {
5      return name;
6    }
7
8    @JsonIgnore
9    public String getAddress() {
10     return address;
11   }
```

Annotations can be used to avoid including sensitive fields in JSON responses

The Java Spring endpoint returning users

```
1  @RequestMapping(path = "/online/users", method = GET, produces = "application/json")
2  public ResponseEntity<Object> getOnlineUsers() {
3    List<User> users = UserService.getOnlineUsers();
4    return new ResponseEntity<Object>(users.stream().map(PublicUserInfo::new), HttpStatus.OK);
5  }
```

The PublicUserInfo DTO class

```
1  public class PublicUserInfo {
2    private String id, name;
3
4    public PublicUserInfo(User user) {
5      this.setId(user.getId());
6      this.setName(user.getName());
7    }
8    …
9    public String getName() {
10     return name;
11   }
```

The DTO class only defines fields that are supposed to be exposed.

A User object is never directly exposed to the client.

TOP 10

Test your API endpoints directly

Testing the behavior of an API cannot be done through a client application.

Inspect the endpoints' code and responses to ensure the API behaves correctly.

The OpenAPI contract for the endpoint

```yaml
paths:
  /online/users:
    get:
      responses:
        '200':
          description: A list of online users
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  properties:
                    id:
                      type: integer
                      description: The user ID
                    name:
                      type: string
```

TOP 10

Use OpenAPI definitions for security

Write Swagger/OpenAPI definitions to specify the behavior of your API.

Security tools consume such definitions for automatic detection and protection.

Polling Question 3: Single Choice

Which of the following are you specifying in your OpenAPI specifications?

1. Security constructs (AuthZ and AuthN)
2. Data contracts (request and response payloads)
3. Both
4. Not using OpenAPI specifications

Polling Question 3: Single Choice

Which of the following are you specifying in your OpenAPI specifications?

Choice 1. Security constructs (AuthZ and AuthN)	9%
Choice 2. Data contracts (request and response payloads)	35%
Choice 3. Both	33%
Choice 4. Not using OpenAPI specifications	23%

The Java Spring endpoint returning users

```
1  @RequestMapping(path = "/user/{id}", method = PATCH, consumes = "application/json")
2  public void updateUser(String id, @RequestBody User user) {
3    UserService.updateUser(id, user);
4  }
```

Updates the DB with new field values for the user with the given ID

The User data class

```
1  public class User {
2    private String id, name, role;
3    …
4    public void setName(String name) {
5      this.name = name;
6    }
7
8    public String setRole(String role) {
9      this. role = role;
10   }
11 }
```

A legitimate request payload to update the user's name

```
1  {
2    "name": "Dr. Phil"
3  }
```

A malicious request payload to update restricted fields

```
1  {
2    "name": ”Philippe becomes admin",
3    "role": "admin"
4  }
```

The Java Spring endpoint returning users

```
1  @RequestMapping(path = "/user/{id}", method = PATCH, consumes = "application/json")
2  public void updateUser(String id, @RequestBody User user) {
3    UserService.updateUser(id, user);
4  }
```

The User data class

```
1   public class User {
2     private String id, name, role;
3     …
4     public void setName(String name) {
5       this.name = name;
6     }
7
8     @JsonProperty(access = Access.READ_ONLY)
9     public String setRole(String role) {
10      this. role = role;
11    }
12  }
```

Annotations can be used to avoid populating sensitive fields with JSON data

The Java Spring endpoint returning users

```
1  @RequestMapping(path = "/user/{id}", method = PATCH, consumes = "application/json")
2  public void updateUser(String id, @RequestBody UpdateUserInfo user) {
3    UserService.updateUser(id, user);
4  }
```

The UpdateUserInfo DTO class

```
1  public class UpdateUserInfo {
2    private name;
3
4    public String getName() {
5      return name;
6    }
7
8    public void setName(String name) {
9      this.name = name;
10   }
11 }
```

The DTO class only defines fields that are supposed to be populated.

A User object is never directly accepted as input from the client.

TOP 10

Test your APIs in their natural habitat

Make sure your API behaves the way you think it does.

Code analysis is only one aspect. Runtime testing is necessary to get the full picture.

The OpenAPI contract for the endpoint

```yaml
/users/{id}:
  patch:
    description: Update a user
    parameters:
      - in: path
        name: id
        description: The unique ID of the user
        required: true
        schema:
          type: integer
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              name:
```

TOP 10

Use OpenAPI definitions for security

Write Swagger/OpenAPI definitions to specify the behavior of your API.

Security tools consume such definitions for automatic detection and protection.

Attendee Questions:

“Is reference token same as an Opaque access token?”

Extra Reading

Further Information

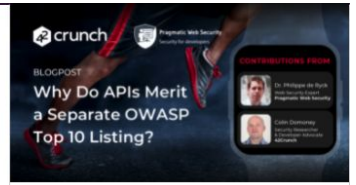
Webinar 3: Remediating the outstanding OWASP API Security Top 10 Issues.
11am EST / 4pm GMT - March 24, 2022

Blogposts

42Crunch Knowledge Series

How Developers Can Become API Security Champions

Question: Everyone is talking about DevSecOps, why are we not able to fix the security issues? Despite the obvious challenges, Colin believes that the industry has made progress as compared to ten years ago when very insecure code was prevalent. Today's code is definitely more secure and security is improving — thankfully most developers are ...



42Crunch Knowledge Series

Why Do APIs Merit a Separate OWASP Top 10 Listing?

Throughout the 3 part webinar series "API Security Landscape Today and the OWASP API Security Top 10 Challenges" we will publish blog posts that highlight some of the main talking points addressed by the speakers. In this post, Philippe and Colin explore the differences between APIs and web apps that necessitated the creation of a ...

APIsecurity.io Weekly Newsletter

https://apisecurity.io/

APIsecurity.io Powered by crunch

#1 OpenAPI Editor - 400k+ users

https://42crunch.com/resources-free-tools/

Developer-First API Security Platform

https://42crunch.com/request-demo/

42 crunch

Pragmatic Web Security
Security for developers

NEW YEAR'S
FITNESS RESOLUTION

OWASP API SECURITY TOP 10

Find, Fix and Secure your APIs

JAN 25, FEB 17 & MAR 24
3-PART WEBINAR SERIES

Dr. Philippe de Ryck
Web Security Expert
Pragmatic Web Security

Colin Domoney
Security Researcher
& Developer Advocate
42Crunch