



ISC2[™] Spotlight

A Comprehensive Guide to API Authorization

isc2.org/Events | [#ISC2Events](https://twitter.com/ISC2Events)

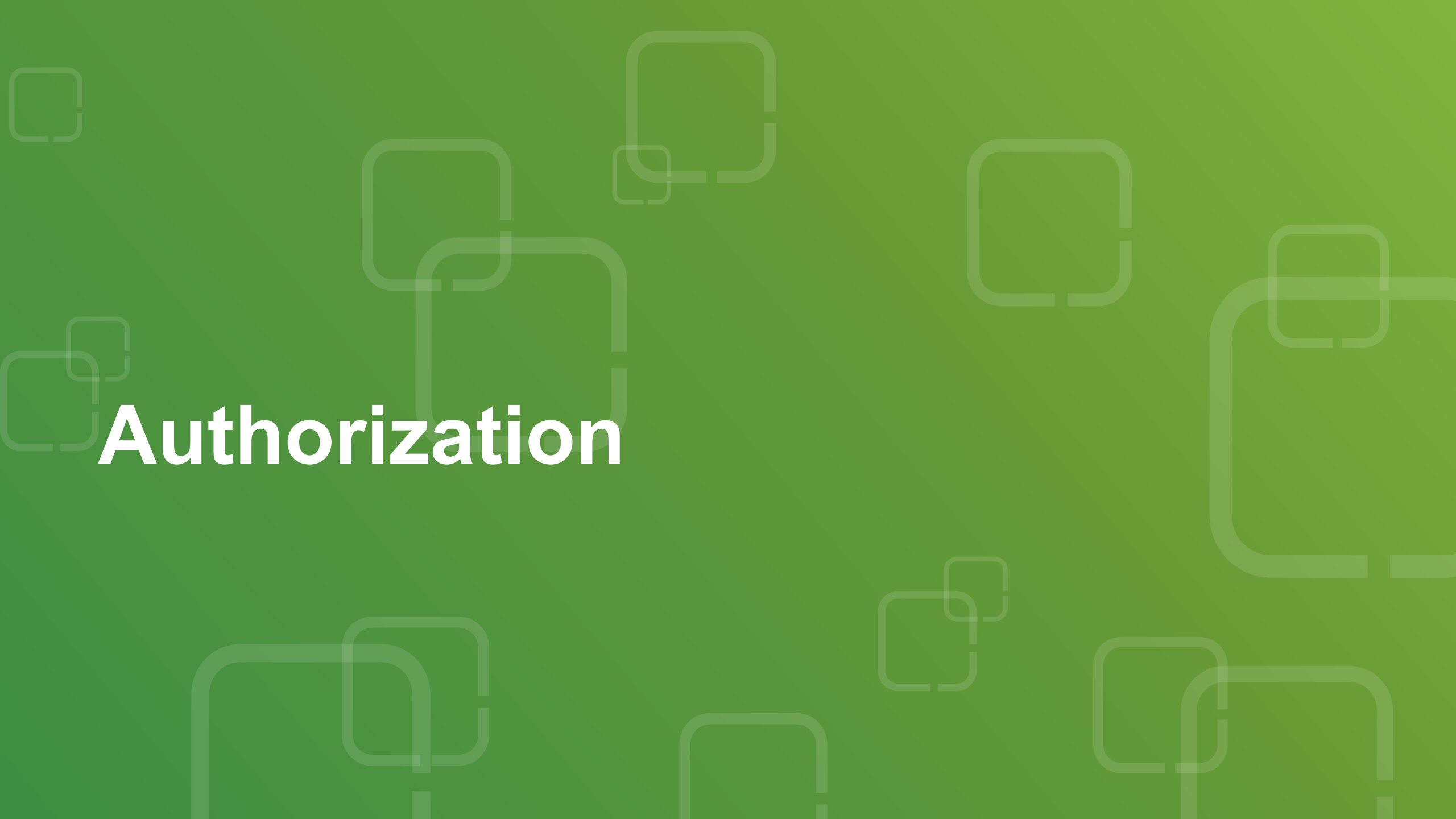
Pradheepa Pullanieswaran

- Sr. Solution Architect
- IAM Enthusiast
- <https://www.linkedin.com/in/pradheepa/>
- www.pradheepa.com

Agenda

- Authorization - What is it?
- API Authorization Types
- RBAC, ABAC, ReBAC
- API Vulnerabilities



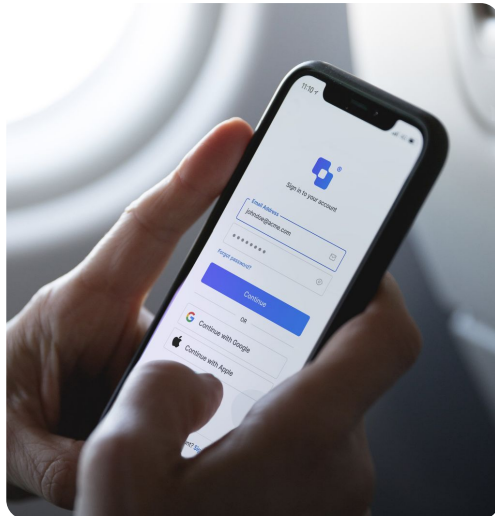


Authorization

Authentication vs Authorization

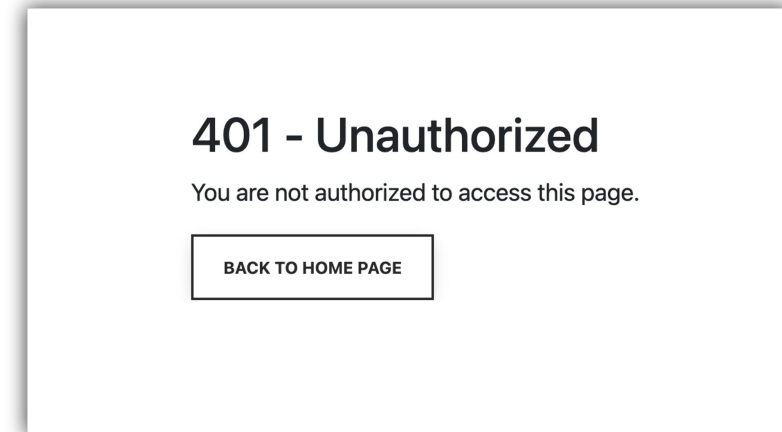
Authentication

- Process of verifying someone's identity
 - E.g. Username, Password



Authorization

- Process of evaluating if a user can perform certain action on a resource
 - Can Anna access this folder?

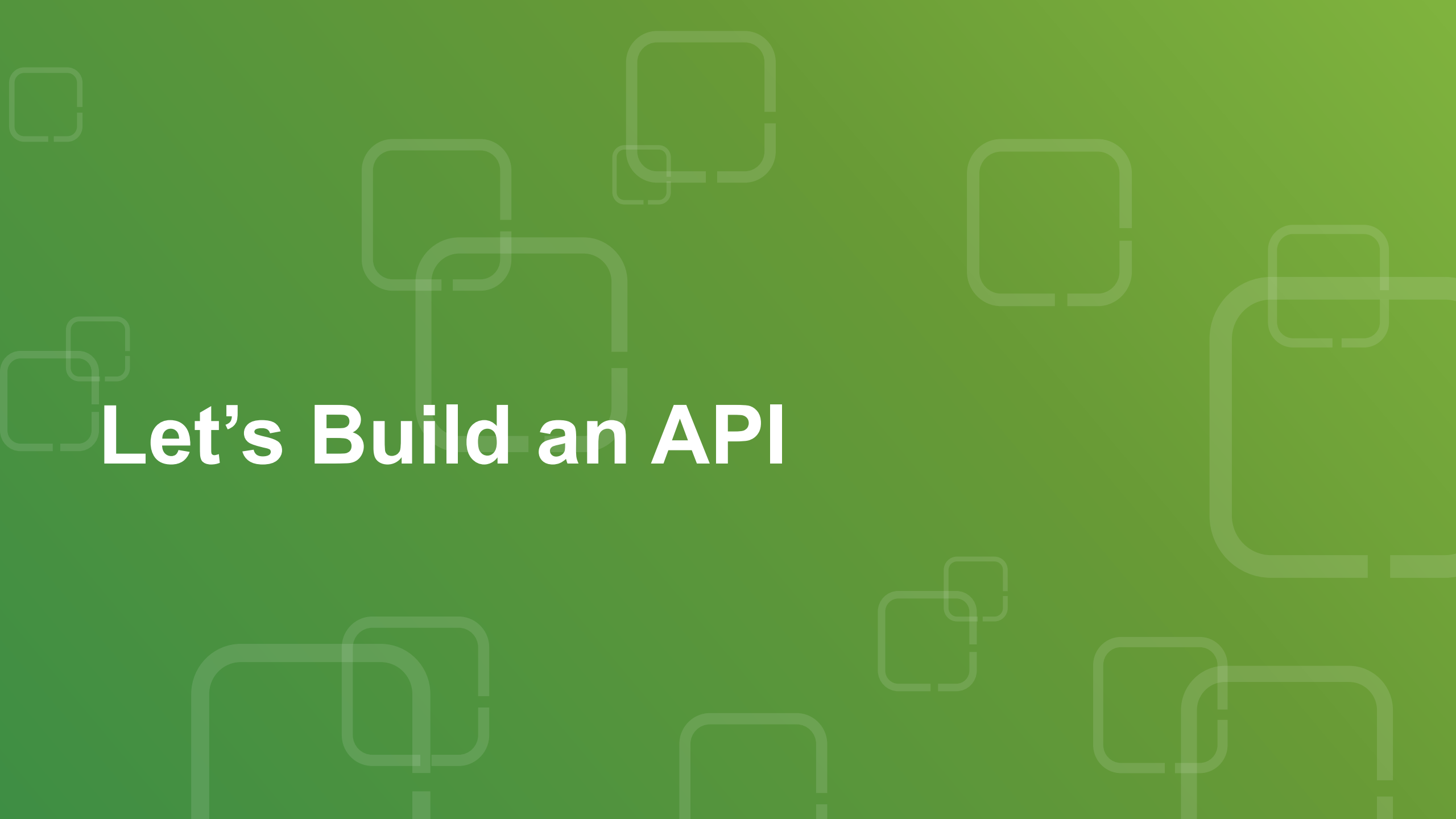


Authorization Types

- API Key
 - Generic secret, No user level information
 - Simple level of Authorization
- OAuth 2.0 - JSON Web Token (JWT)
 - Token body contains user information, called claims
 - Support multiple types of Access Control
- Bearer Token
 - Authorization : Bearer <Token>
 - Often used along with OAuth 2.0

Design Approaches to Access Control

- RBAC: Role-Based Access Control
- ABAC: Attribute-Based Access Control
- ReBAC: Relationship-Based Access Control



Let's Build an API

API Returning a Document

```
Get Documents

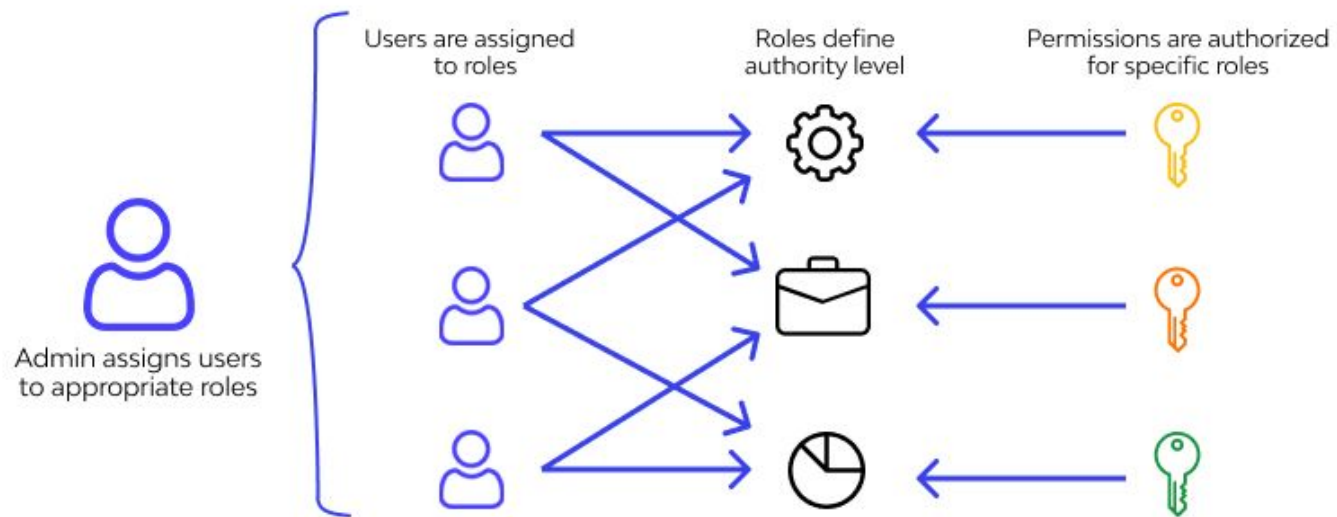
app.get("/documents/:id", async (res, req) => {
  const document = await db.documents.findOne({id: req.params.id})
  res.send({ document});
});
```



Role-Based Access Control

RBAC

Role-Based Access Control



Authorization with RBAC

```
Role-Based Access Control

app.get("/documents/:id", async (res, req) => {

  if(!req.user.roles.includes("employee")){
    throw new ForbiddenError();
  }
  const document = await db.documents.findOne({id: req.params.id})
  res.send({ document});
});
```

RBAC - Pros & Cons

Pros

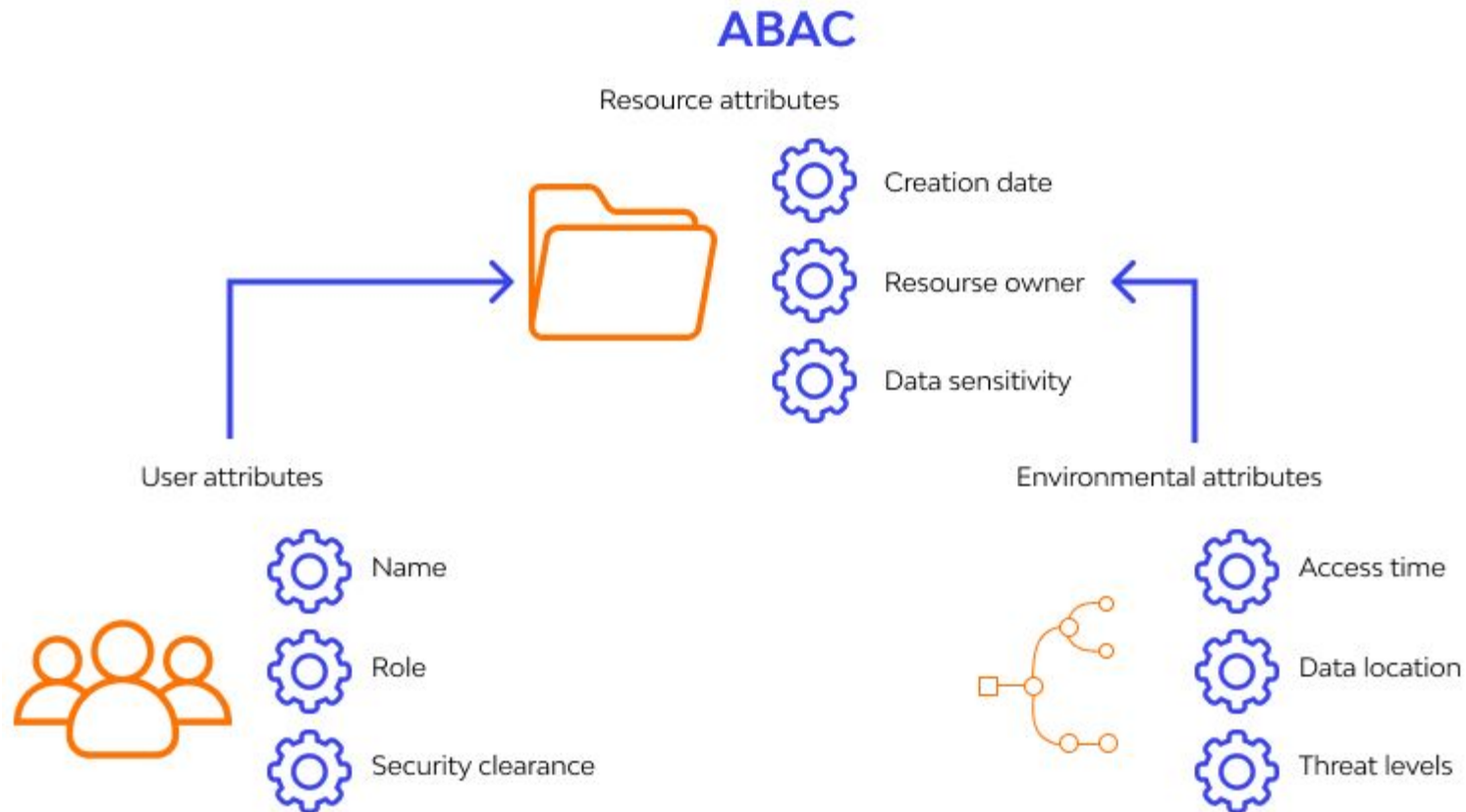
- Simple and Easy to use
- Good for applications where there are defined roles for all the personas

Cons

- Role Explosion
- Limited to express more complex or contextual access policies

Attribute-Based Access Control

ABAC



Authorization with ABAC

```
Attribute-Based Access Control

const document = await db.documents.findOne({id: req.params.id})

// Get all the folders recursively
const folders = await db.folders.recursivelyFindAllFolders({id:
document.folder_id})

// Get all the groups recursively
const groups = await db.groups.recursivelyFindAllGroups({id: req.user.group_ids})

// Verify if the user has access
boolean hasAccess = await authorize(req.user, "read", {folders, groups});

if (hasAccess) { res.send({ document}); }
});
```


ABAC - Pros & Cons

Pros

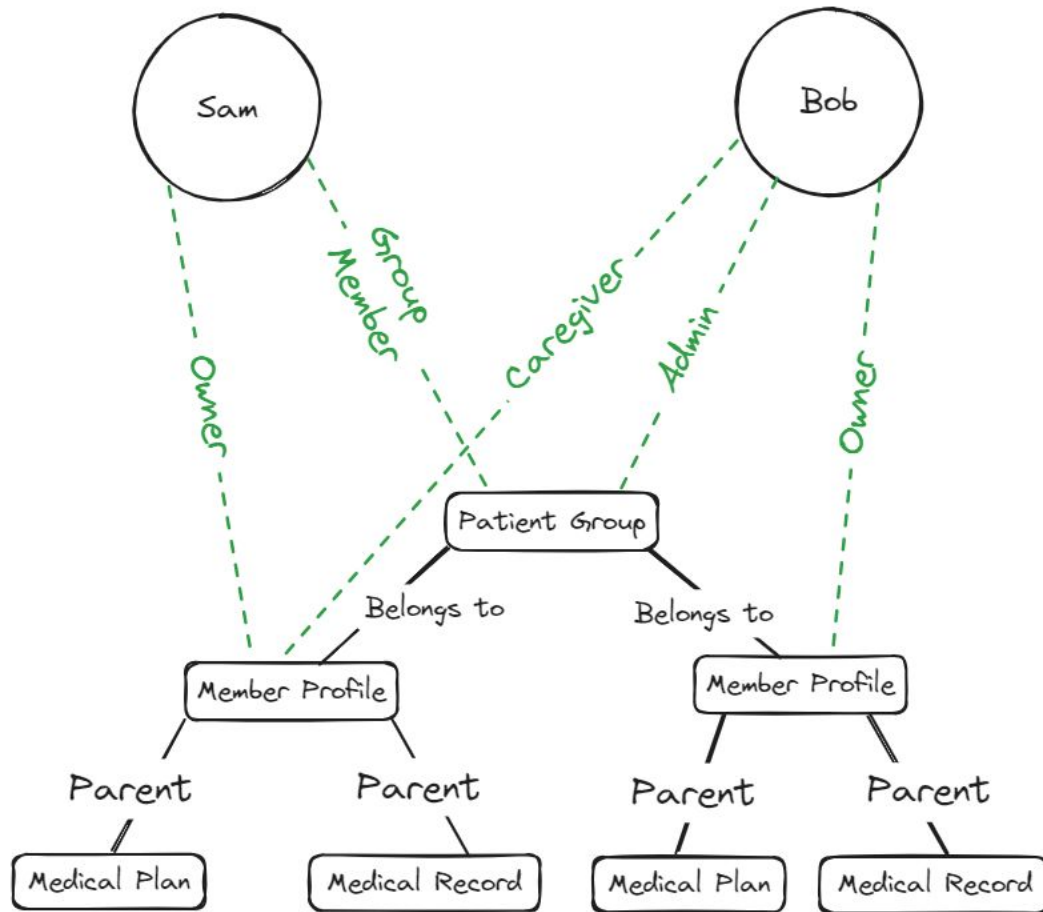
- Fine-Grained Access

Cons

- Need to collect all the data before verification.
- Data needs to be gathered from the product.

Relationship-Based Access Control

ReBAC



Relationships

- User
- Resource
- Relation

Relationships Definition

```
ReBAC - Relationships

model
  schema 1.1

  type doc
    relations
      define can_change_owner: owner
      define can_read: viewer or owner or viewer from parent
      define can_share: owner or owner from parent
      define can_write: owner or owner from parent
      define owner: [user]
      define parent: [folder]
      define viewer: [user, user:*, group#member]

  type folder
    relations
      define can_create_file: owner
      define owner: [user]
      define parent: [folder]
      define viewer: [user, user:*, group#member] or owner or viewer from parent

  type group
    relations
      define member: [user]

  type user
```

Authorization with ReBAC

```
Relationship-Based Access Control

const { rebacClient } = require ('@openfga/sdk');
const apiClient = new OpenFgaClient({'API_HOST', 'STORE_ID', 'CREDS'})

async function getDocument(userId, documentId){

  //Check access
  const hasAccess = apiClient.check(
    {"user": `user:userId`, "relation": "read", "object": `document:documentId` }
  );

  if(!hasAccess){
    throw new ForbiddenError();
  }

  //find the document and return
  return db.documents.find({id: documentId});
}
```

ReBAC - Pros & Cons

Pros

- Fine-Grained Access like Google Drive
- Complex Hierarchy
- Reverse Hierarchy
- Centralized Repository

Cons

- Complexity
- Difficult to Audit
- Learning Curve to model the correct the relationship
- Should be highly available

Vulnerabilities

Authorization done wrong

OWASP Top 10 API Security Risks – 2023

Risk	Description
API1:2023 - Broken Object Level Authorization	APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface of Object Level Access Control issues. Object level authorization checks should be considered in every function that accesses a data source using an ID from the user.
API2:2023 - Broken Authentication	Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising a system's ability to identify the client/user, compromises API security overall.
API3:2023 - Broken Object Property Level Authorization	This category combines API3:2019 Excessive Data Exposure and API6:2019 - Mass Assignment , focusing on the root cause: the lack of or improper authorization validation at the object property level. This leads to information exposure or manipulation by unauthorized parties.
API4:2023 - Unrestricted Resource Consumption	Satisfying API requests requires resources such as network bandwidth, CPU, memory, and storage. Other resources such as emails/SMS/phone calls or biometrics validation are made available by service providers via API integrations, and paid for per request. Successful attacks can lead to Denial of Service or an increase of operational costs.
API5:2023 - Broken Function Level Authorization	Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers can gain access to other users' resources and/or administrative functions.

The background is a solid green color with a pattern of overlapping, rounded squares in a lighter shade of green. These squares are of various sizes and are arranged in a way that they appear to be floating or layered, creating a modern, geometric aesthetic.

Thank You!

ISC2[™]Spotlight

© 2023 ISC2. All rights reserved. This presentation's images are subject to copyright protection and used under license from third parties. Do not use images from this presentation in other presentations or documents without first consulting with the creative team. The use of copyrighted images outside the licensed scope constitutes copyright infringement and subjects the user to monetary damages and other penalties.