Fullstack Development

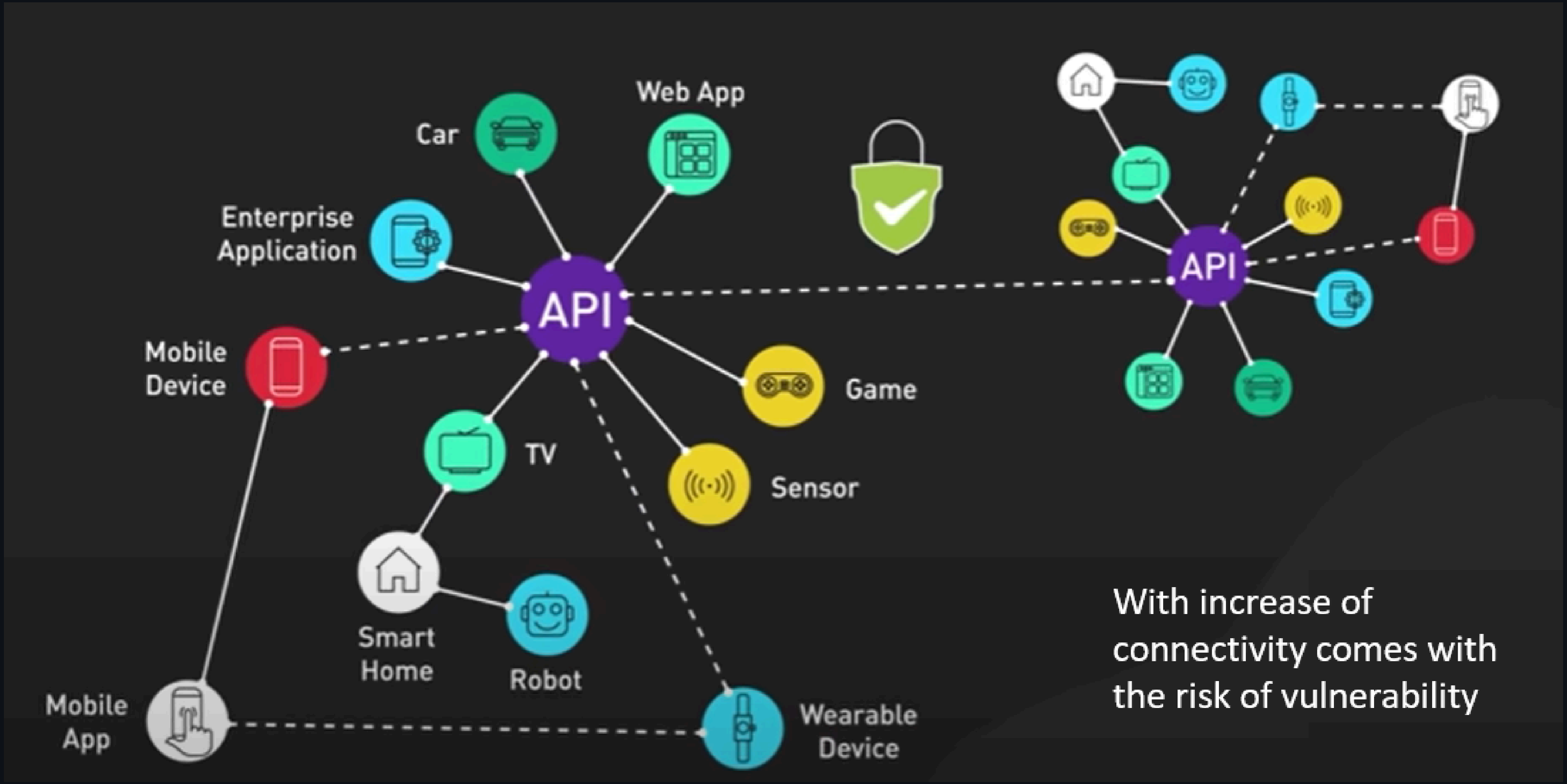
API Architectures and Design #3

Content

- What is API?
- API Architecture Styles
- RESTful API design
- **API Security**
- API Testing

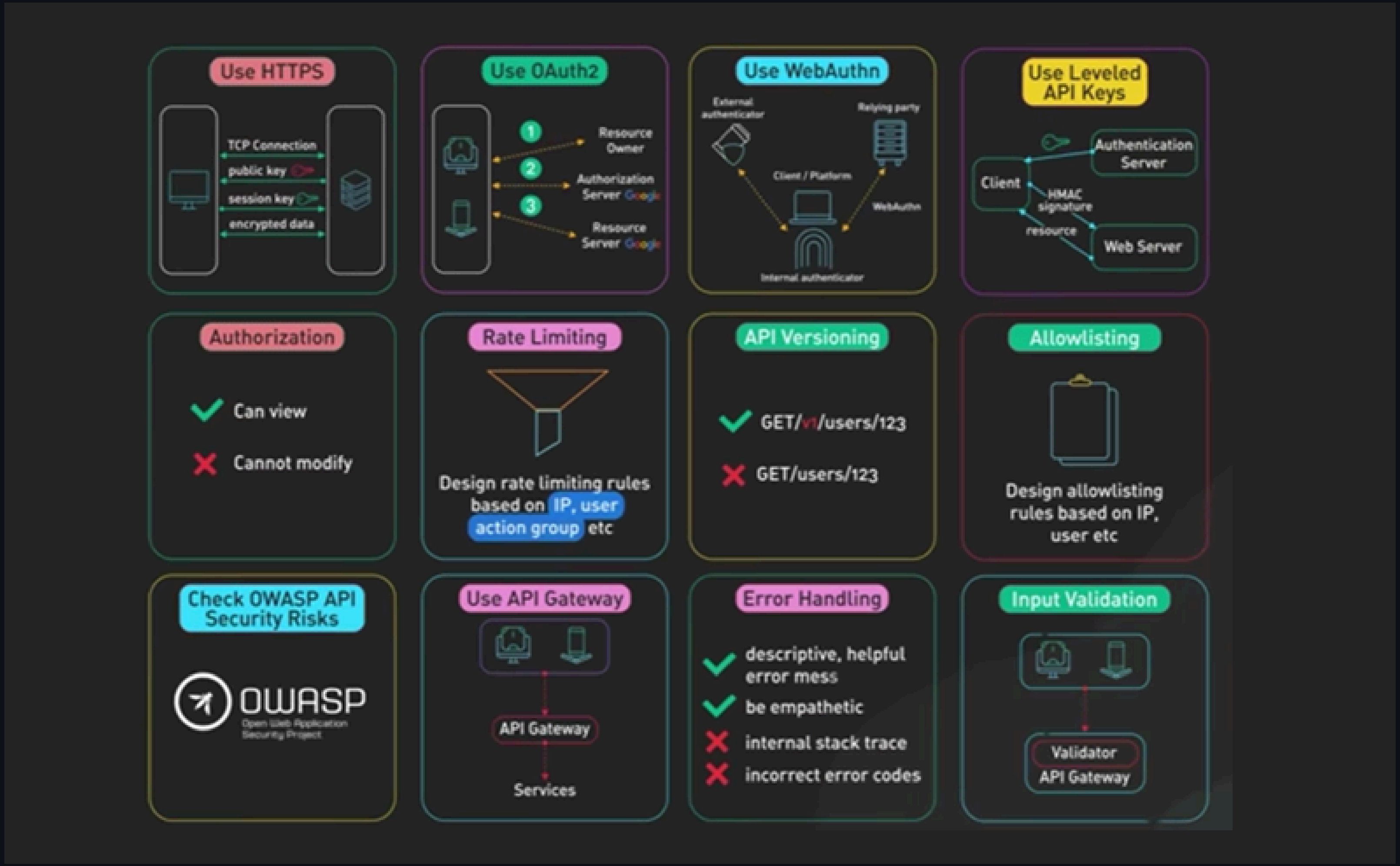
API Security

12 Tips for API Security



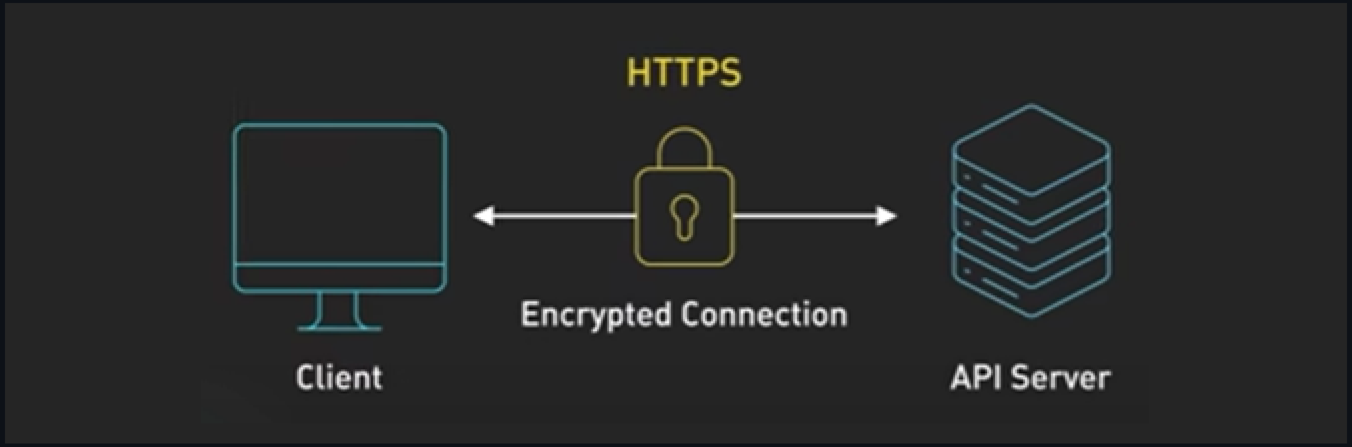
With increase of connectivity comes with the risk of vulnerability

12 Tips for API Security



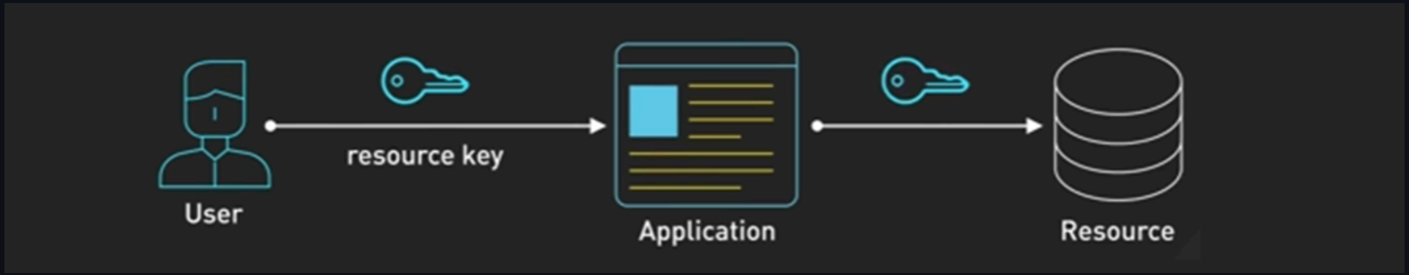
1. Use HTTPS

- `Encrypt data` transmitted between client and server
- Prevent `eavesdropping` and `man-in-the-middle-attack`
- Protect `API keys`, `session tokens`, and user data

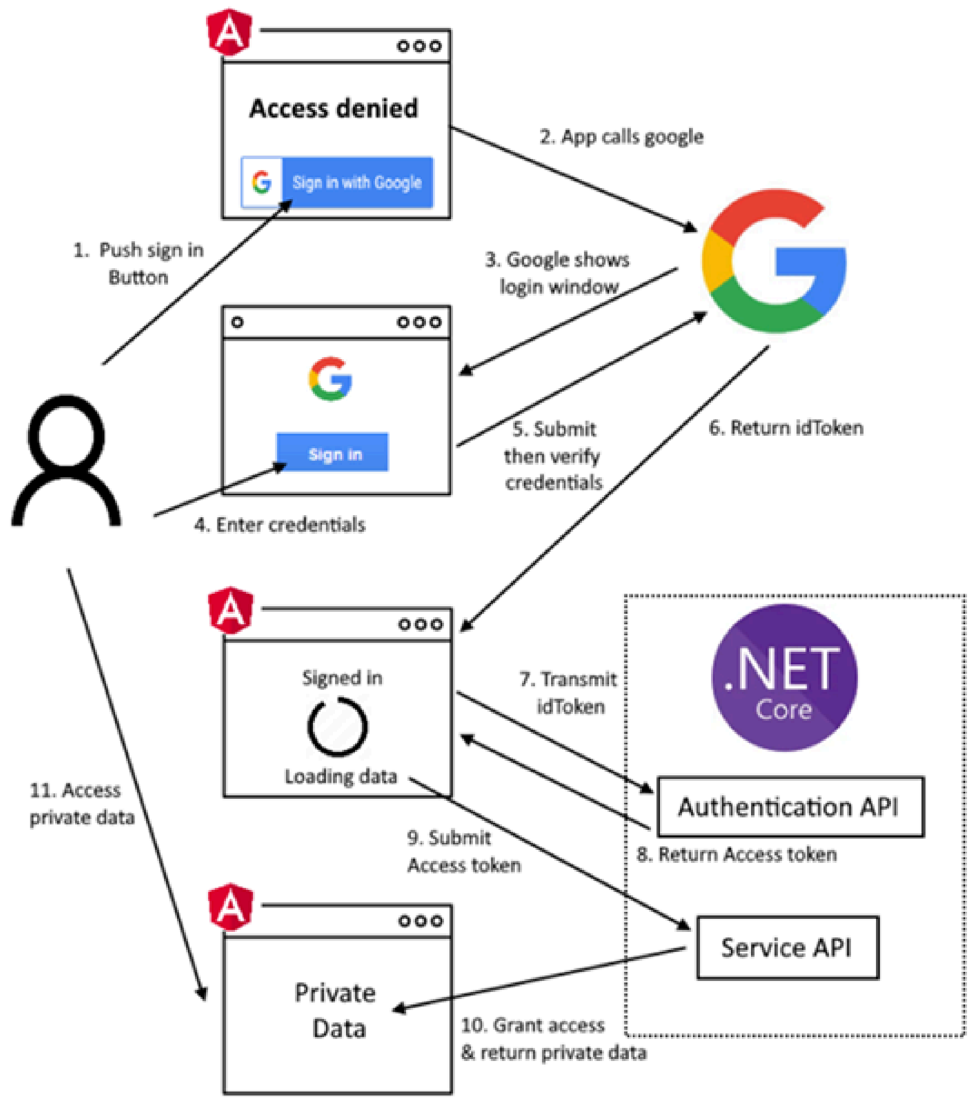


2. Use OAuth2

- Modern industry standard `authorization` protocol
- Allow a user to grant `3rd-party` app (i.e., our application) limited access to its resources
    - Without sharing user's credential
    - `Authorization server` generates and return a `temporary token` to client
    - `Client` uses the token to access 3rd-party app resources

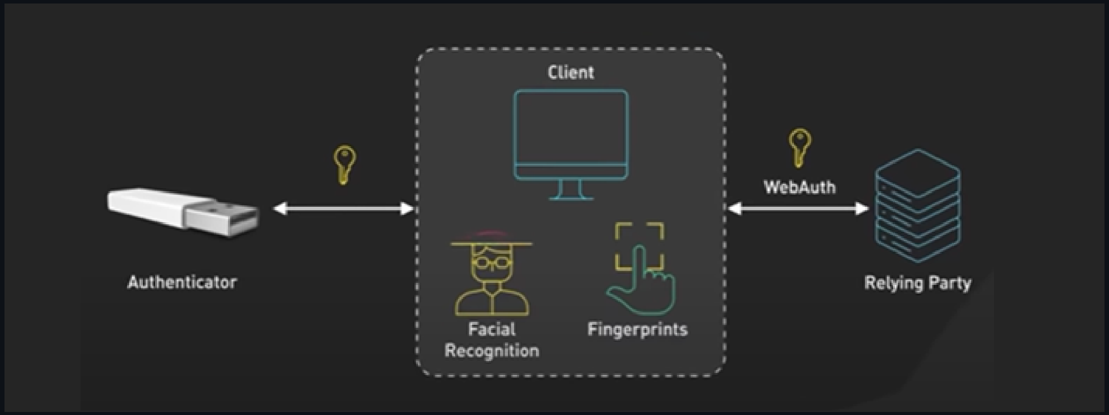


Google OAuth2

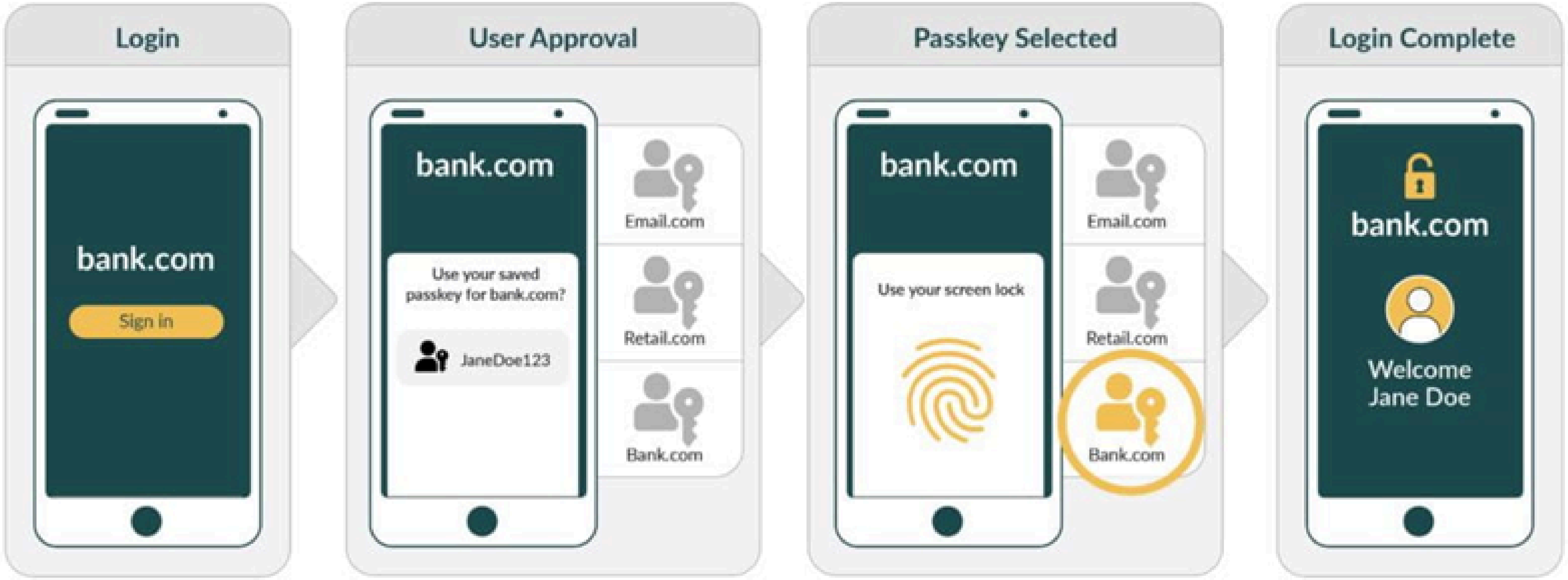


3. Use WebAuthn (and Passkey)

- `WebAuthn` is technical standard for `passwordless` authentication
  - Using `public-key` cryptography
- `Passkey` is a user-friendly implementation of WebAuthn
  - User needs to register `Passkey` for each device (laptop, mobile, ...)
  - For each `Passkey`, a pair of `public-private` key is generated

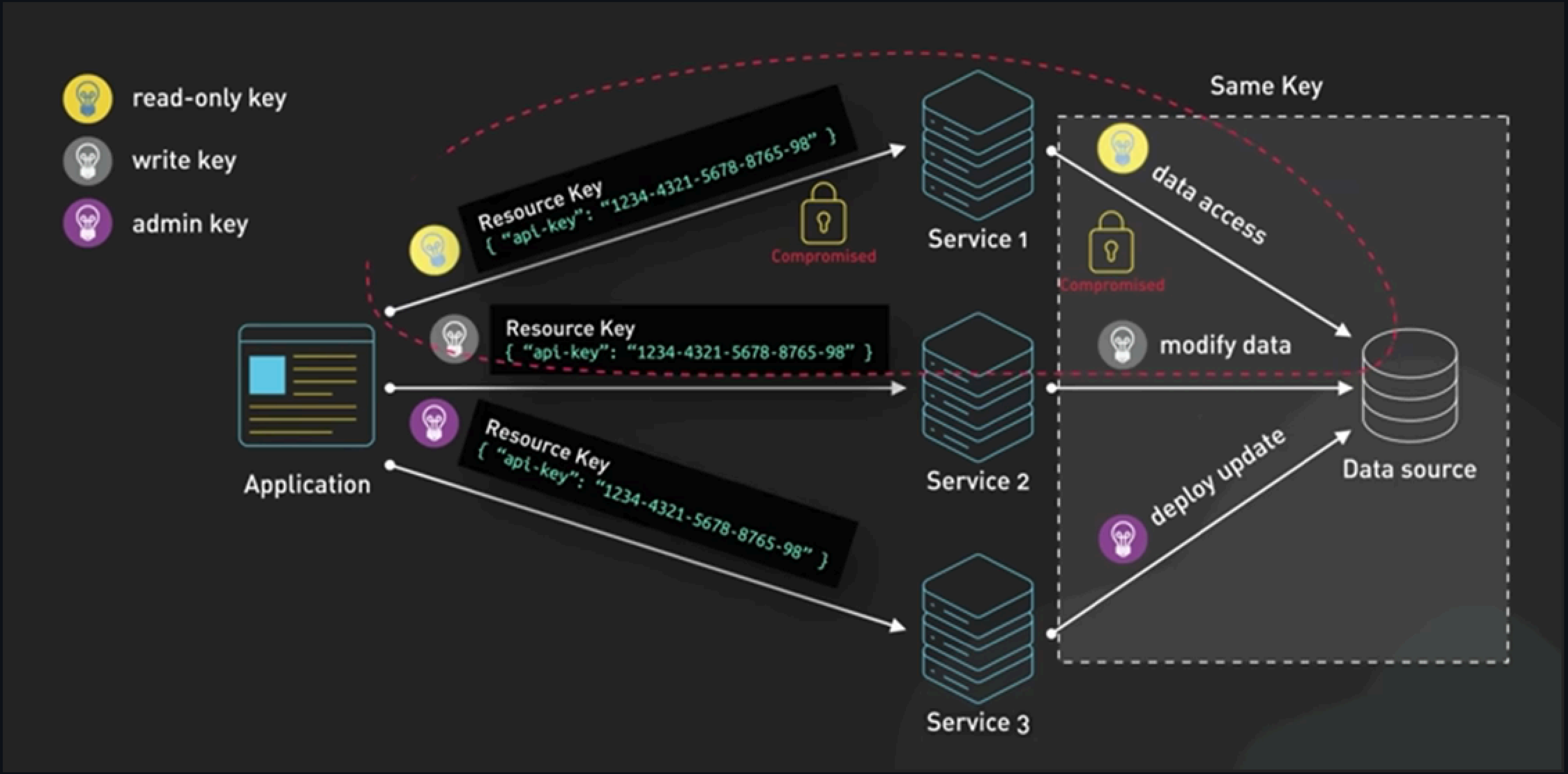


Signing in with `Passkey`



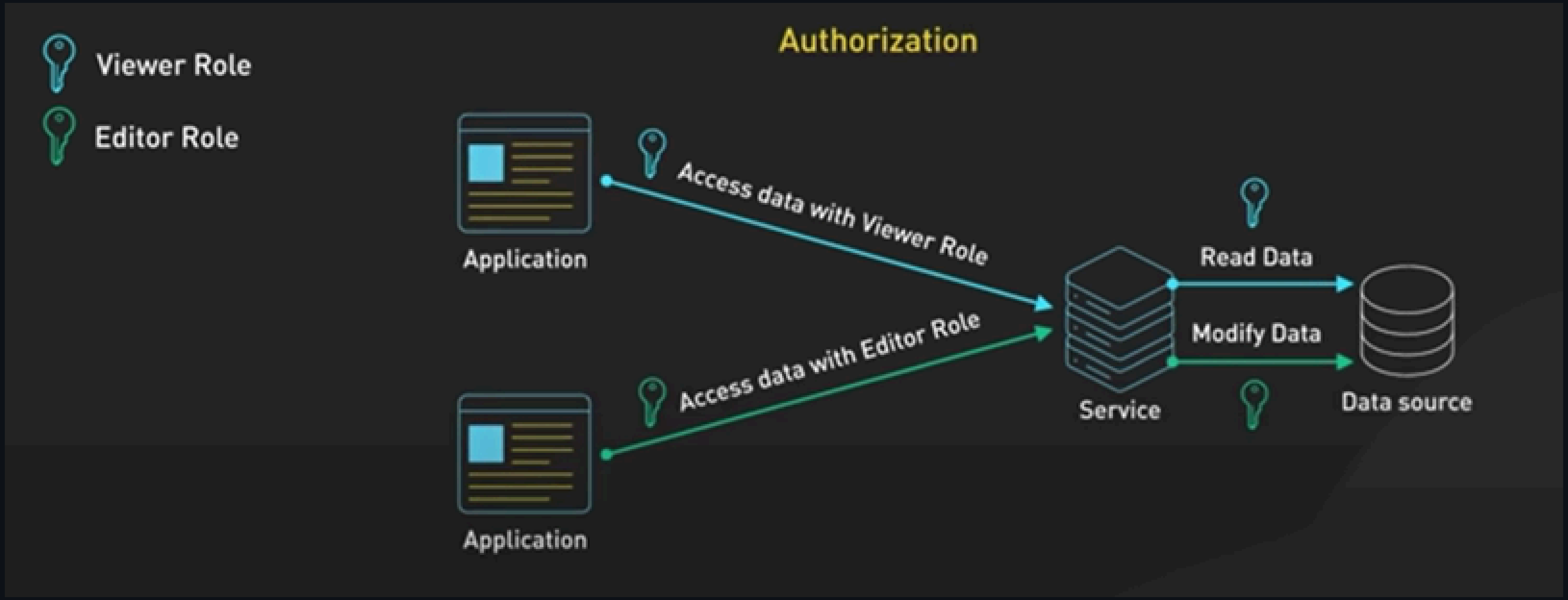
4. Use Leveled API Keys

- Use **Multiple API keys** for different `permissions` / `resources`
  - `Read-only` access: retrieve data
  - `Write` access: modify data
  - `Admin` access: deploy update, ...
- If a key is **compromised**, attacker only have a certain access
  - Minimize **blast radius**


read-only key
write key
admin key
Resource Key
{ "api-key": "1234-4321-5678-8765-98" }
Resource Key
{ "api-key": "1234-4321-5678-8765-98" }
Resource Key
{ "api-key": "1234-4321-5678-8765-98" }
Compromised
Application
Service 1
Service 2
Service 3
Same Key
Compromised
data access
modify data
deploy update
Data source

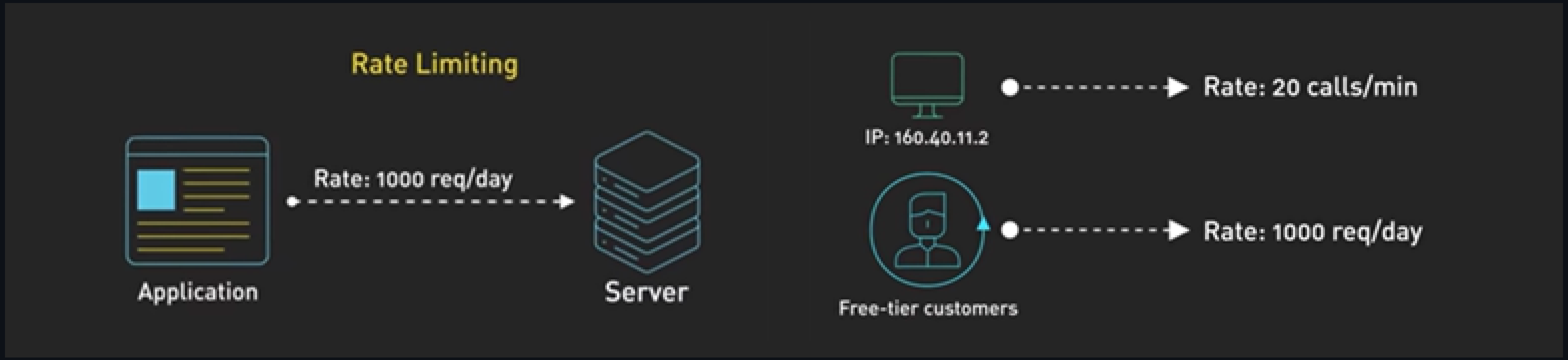
5. RBAC Authorization

- Implement **Role-based Access Control**
- Different `roles` have different group of `permissions`
- A **user** may be assigned with **multiple roles**



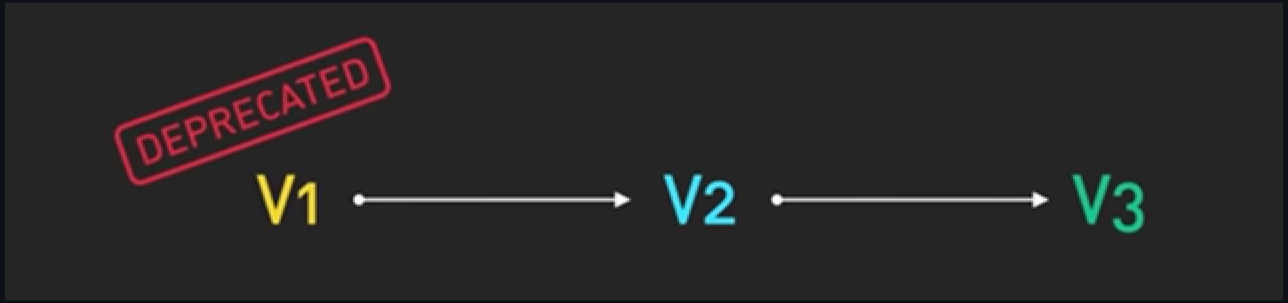
6. Rate limiting

- Controls the **number of requests** in a given period of time (`#hour`, `#days`)
- Improves `security`, `performance`, and `availability`
- Can be based on many factors: `IP Address`, `User ID`, `API Key`, ...



7. API Versioning

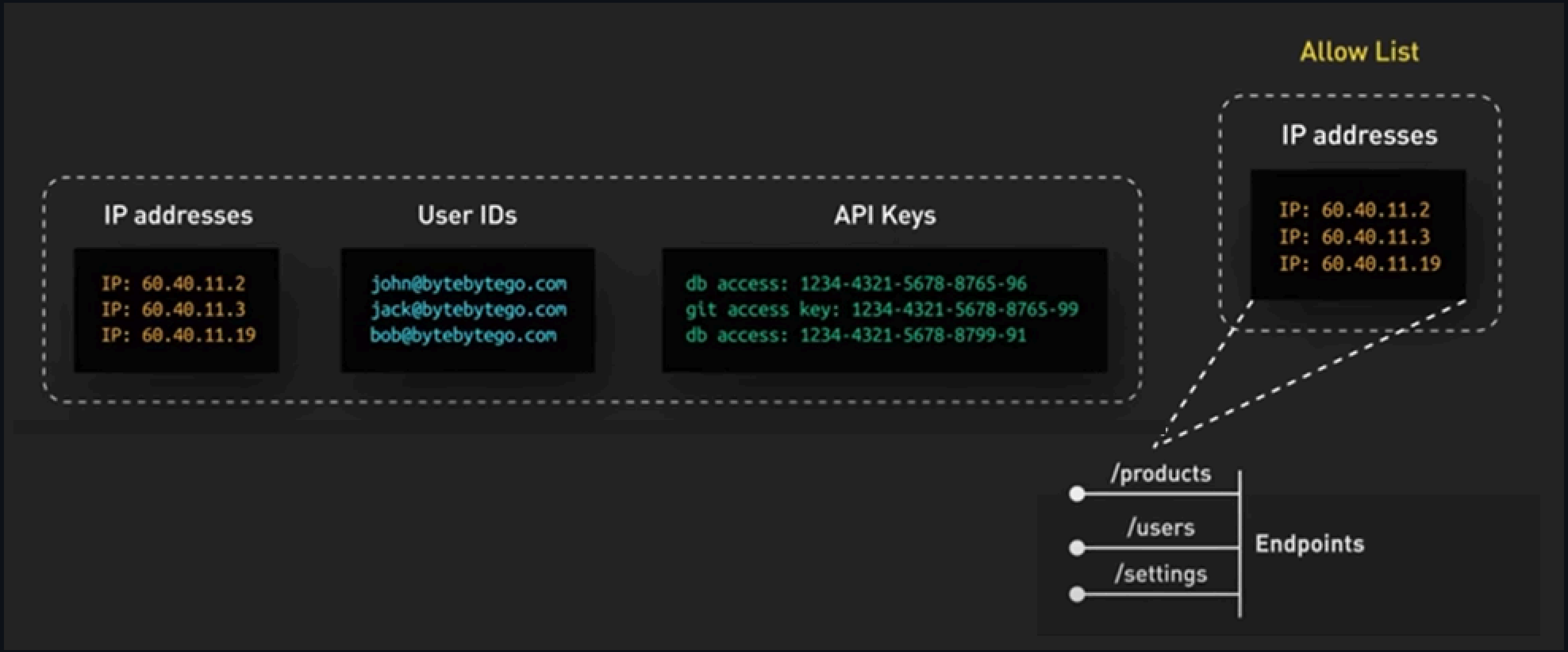
- Allow developer to evolve API over time
- Provide new features without disrupting existing clients
- Help in **change management** and **documentation**



8. Allow Listing

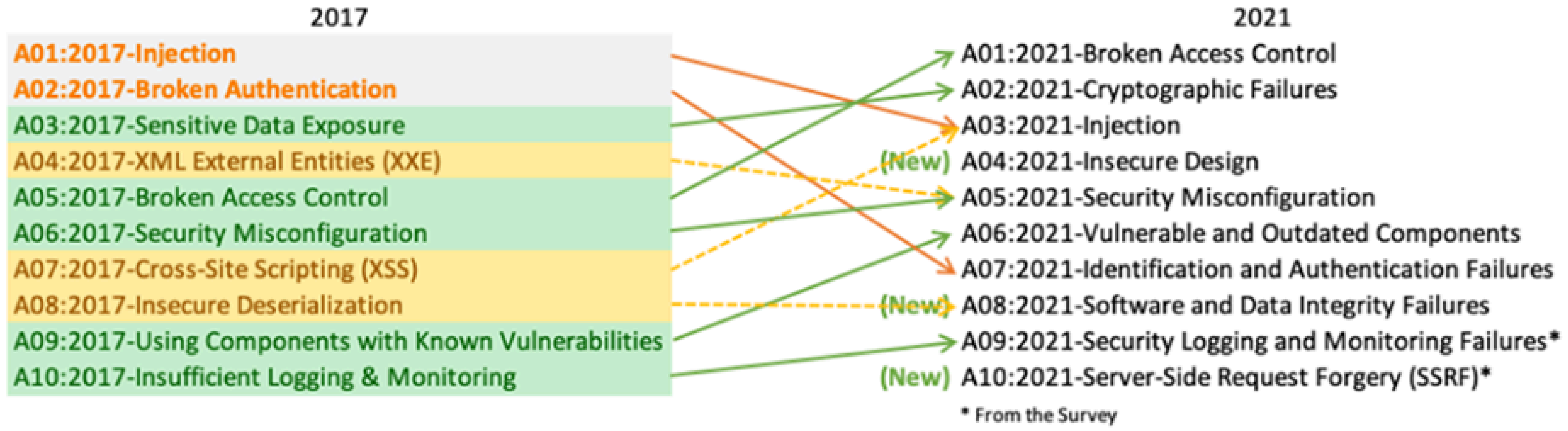
- A list of explicitly **allowed entities** (aka. `Whitelist`)
    - `IP Address` , `User ID` , `API Key` , ...
    - **Deny all**, **Permit some**
- Give limited access to certain resources

Whitelist


Allow List
IP addresses
IP: 60.40.11.2
IP: 60.40.11.3
IP: 60.40.11.19
IP addresses
IP: 60.40.11.2
IP: 60.40.11.3
IP: 60.40.11.19
User IDs
john@bytebytego.com
jack@bytebytego.com
bob@bytebytego.com
API Keys
db access: 1234-4321-5678-8765-96
git access key: 1234-4321-5678-8765-99
db access: 1234-4321-5678-8799-91
/products
/users
/settings
Endpoints

9. Check OWASP API Security Risks

- **OWASP** provides resources for `Web app` and `API` security
- Top 10 most security risks



2017

- A01:2017-Injection
- A02:2017-Broken Authentication
- A03:2017-Sensitive Data Exposure
- A04:2017-XML External Entities (XXE)
- A05:2017-Broken Access Control
- A06:2017-Security Misconfiguration
- A07:2017-Cross-Site Scripting (XSS)
- A08:2017-Insecure Deserialization
- A09:2017-Using Components with Known Vulnerabilities
- A10:2017-Insufficient Logging & Monitoring

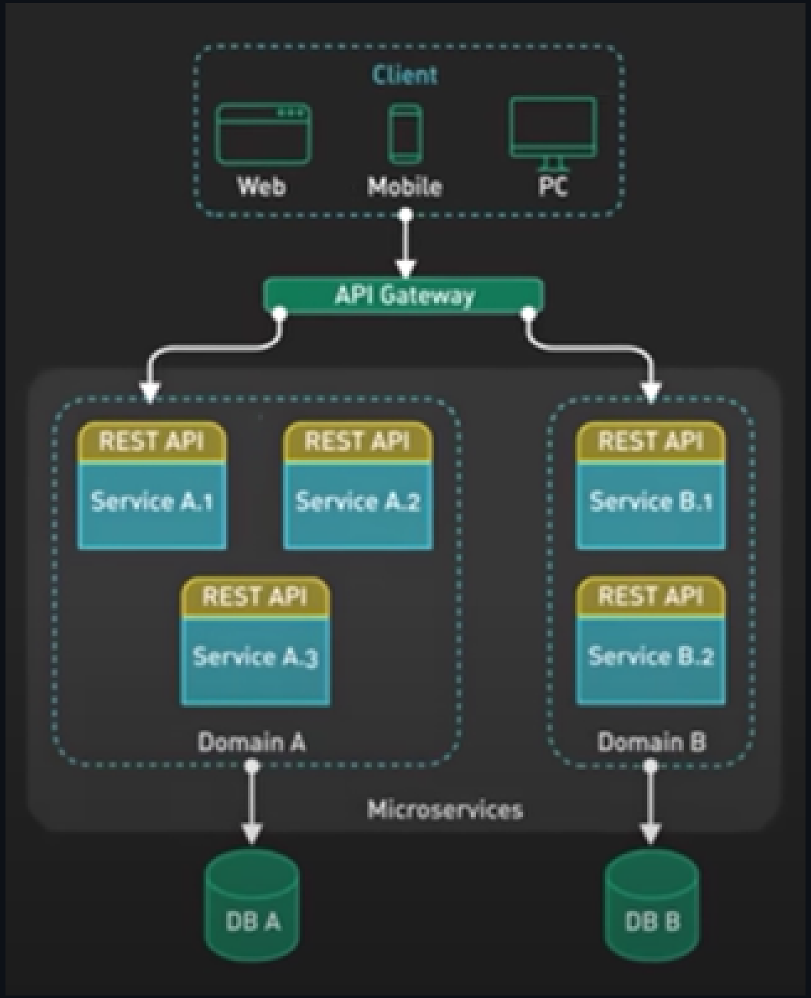
2021

- A01:2021-Broken Access Control
- A02:2021-Cryptographic Failures
- A03:2021-Injection
- (New) A04:2021-Insecure Design
- A05:2021-Security Misconfiguration
- A06:2021-Vulnerable and Outdated Components
- A07:2021-Identification and Authentication Failures
- (New) A08:2021-Software and Data Integrity Failures
- A09:2021-Security Logging and Monitoring Failures*
- (New) A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

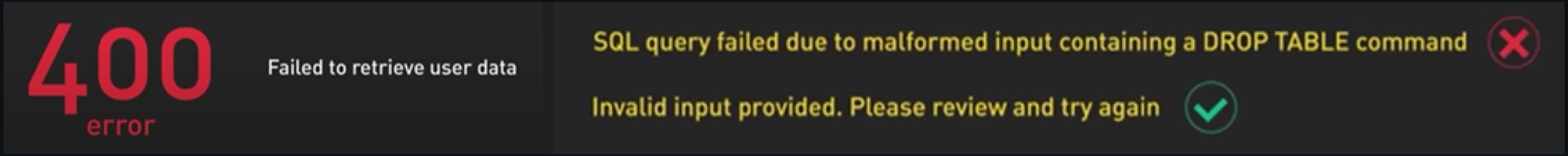
10. Use API Gateway

- A single **entry-point** to backend services
- Security policy enforcement
  - Authentication
  - Traffic management
  - Rate limiting
  - Caching
  - Logging / Monitoring



11. Error Handling

- Crucial for API security and user experience
- Avoid **Internal Server Error (500)**
  - `Failed to retrieve data`
  - `Please check that you are authenticated and have sufficient permissions`
- Avoid exposing sensitive data



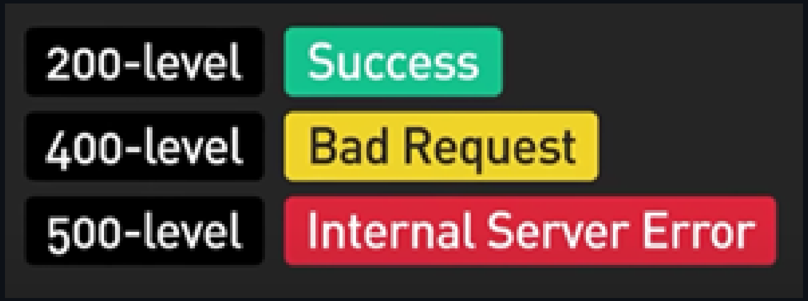
Failed to retrieve user data

SQL query failed due to malformed input containing a DROP TABLE command ✗

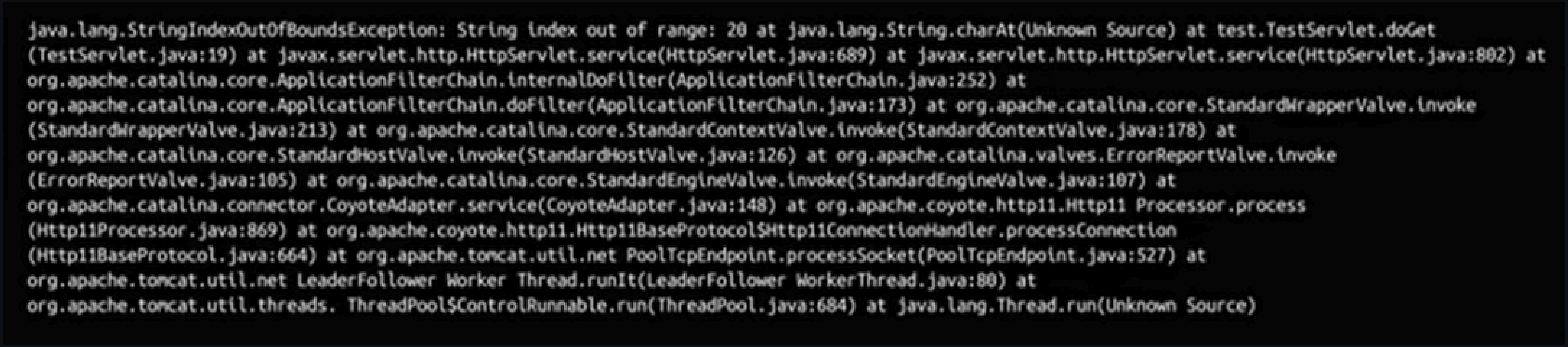
Invalid input provided. Please review and try again ✓

11. Error Handling (2)

- Never expose **internal error messages**
    - Can be valuable information for attackers



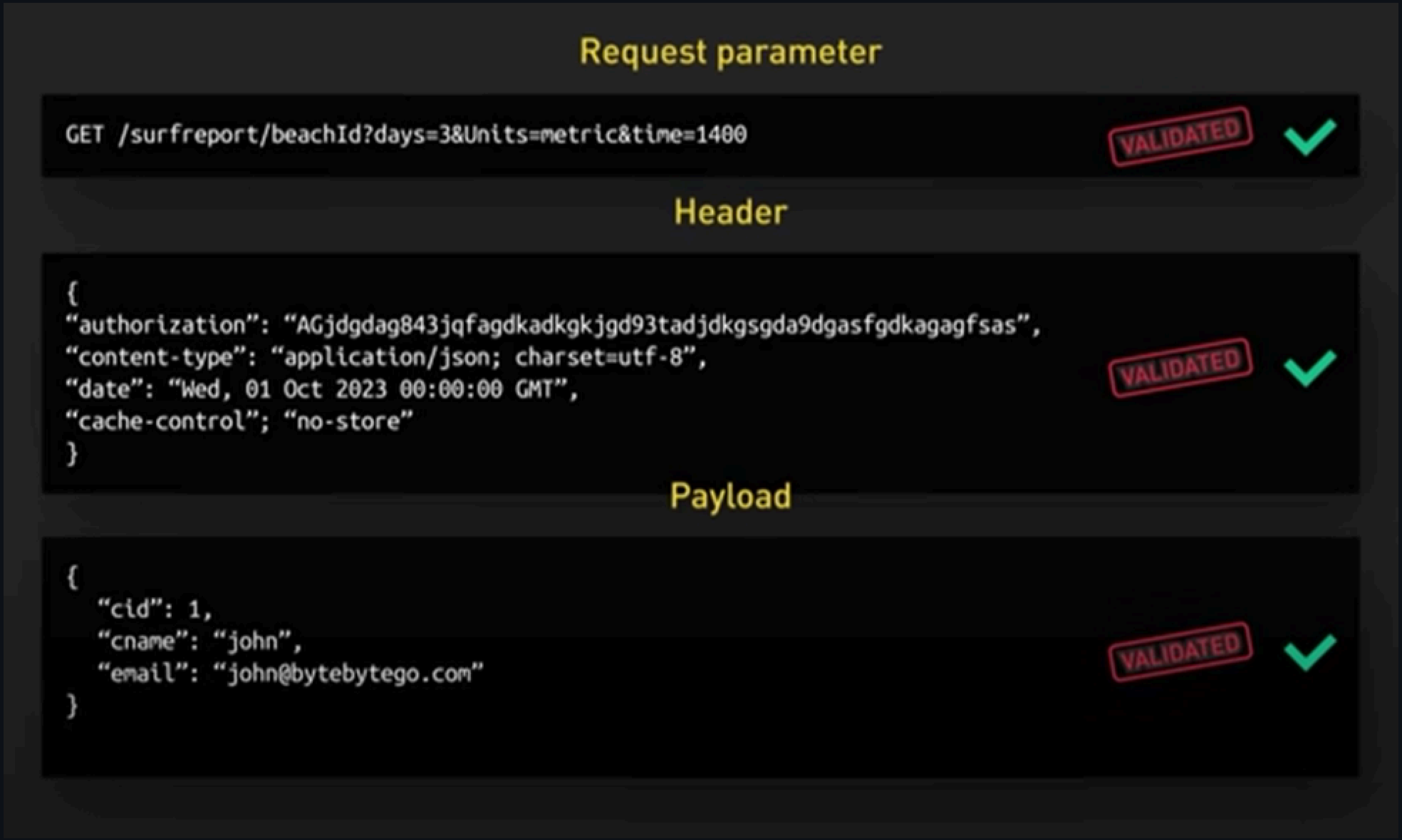
200-level	Success
400-level	Bad Request
500-level	Internal Server Error



```
java.lang.StringIndexOutOfBoundsException: String index out of range: 20 at java.lang.String.charAt(Unknown Source) at test.TestServlet.doGet
(TestServlet.java:19) at javax.servlet.http.HttpServlet.service(HttpServlet.java:689) at javax.servlet.http.HttpServlet.service(HttpServlet.java:802) at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:252) at
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173) at org.apache.catalina.core.StandardWrapperValve.invoke
(StandardWrapperValve.java:213) at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:178) at
org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:126) at org.apache.catalina.valves.ErrorReportValve.invoke
(ErrorReportValve.java:105) at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:107) at
org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:148) at org.apache.coyote.http11.Http11 Processor.process
(Http11Processor.java:869) at org.apache.coyote.http11.Http11BaseProtocol$Http11ConnectionHandler.processConnection
(Http11BaseProtocol.java:664) at org.apache.tomcat.util.net PoolTcpEndpoint.processSocket(PoolTcpEndpoint.java:527) at
org.apache.tomcat.util.net LeaderFollower Worker Thread.runIt(LeaderFollower WorkerThread.java:80) at
org.apache.tomcat.util.threads. ThreadPool$ControlRunnable.run(ThreadPool.java:684) at java.lang.Thread.run(Unknown Source)
```

12. Input Validation

- Validate user supplied **inputs**
  - Request parameters
  - Header
  - Payload
- **Invalidated input** can lead to problems
  - SQL injection
  - Cross-site scripting (XSS)
- Validation should be done on both **client** and **server**



Request parameter

```
GET /surfreport/beachId?days=3&Units=metric&time=1400
```

VALIDATED ✓

Header

```
{
"authorization": "AGjdgdag843jqfagdkadkgkjgd93tadjdkgsgda9dgasfgdkagagfsas",
"content-type": "application/json; charset=utf-8",
"date": "Wed, 01 Oct 2023 00:00:00 GMT",
"cache-control"; "no-store"
}
```

VALIDATED ✓

Payload

```
{
  "cid": 1,
  "cname": "john",
  "email": "john@bytebytego.com"
}
```

VALIDATED ✓

References

- Top 12 Tips for API Security