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

"I tried to brute force the 6 digit code on www.facebook.com and was blocked after 10–12 invalid attempts."

"Then I looked out for the same issue on beta.facebook.com and mbasic.beta.facebook.com."

"Interestingly, rate limiting was missing from forgot password endpoint."

https://www.freecodecamp.org/news/responsible-disclosure-how-i-could-have-hacked-all-facebook-accounts-f47c0252ae4d/

TOP 10

Treat your APIs like your children

Keep track where you put them, regularly check up on them, keep them out of trouble.

TOP 10

YOUR ATTACK SURFACE = EVERY EXPOSED API

Keep track of the assets you deploy, assign responsibilities for assets, and maximize API re-use.

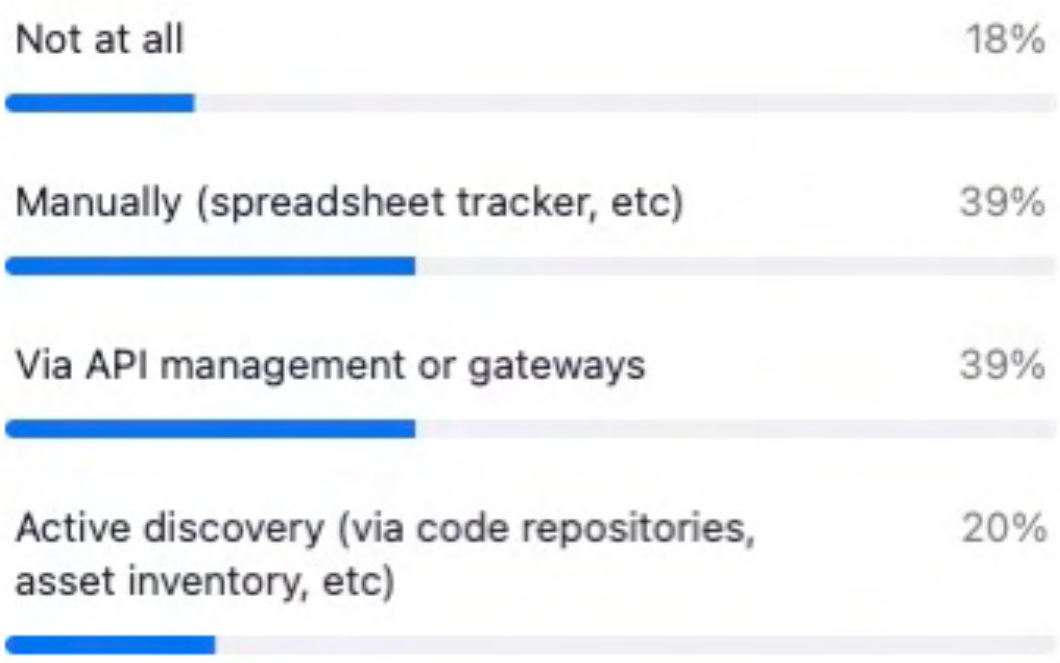
Polling Question 1: (Multiple Choice)

How are you tracking and managing your API inventory?

1. Not at all
2. Manually (spreadsheet tracker, etc)
3. Via API management or gateways
1. Active discovery (via code repositories, asset inventory, etc)

Polling Question 1: (Multiple Choice)

How are you tracking and managing your API inventory?



Attendee Question

Many companies to pen testing. Doesn't it solve the problems you mentioned here?

Attendee Question

The 42 Crunch Platform relies on a valid OpenAPI Spec File. Do you have any good recommendations of libraries for various languages like
Node, Golang, Python that'll help generate these valid files? I know Swashbuckle is really good for .NET.

A Python Flask API endpoint

```
1 @app.route('/')
2 def my_first_api_endpoint():
3   json_data = json.loads(request.data)
4   ...
5   return "", 200
```

Which HTTP methods are accepted by this endpoint?

Overlooked vulnerabilities in GraphQL open the door to cross-site request forgery attacks

Charlie Osborne 26 May 2021 at 10:14 UTC
Updated: 26 May 2021 at 10:26 UTC

CSRF XS-Leak Secure Development

CSRF risk factors are often hidden, and misunderstood, in GraphQL implementations

https://portswigger.net/daily-swig/overlooked-vulnerabilities-in-graphql-open-the-door-to-cross-site-request-forgery-attacks

Flask defaults to GET, but supports explicit configuration of allowed HTTP methods

A Python Flask API endpoint

```
1 @app.route('/', methods=['POST'])
2 def my_first_api_endpoint():
3   json_data = json.loads(request.data)
4   ...
5   return "", 200
```

TOP 10

RESTRICT HTTP METHODS

Ensure your API only accepts expected HTTP methods, both using code analysis and dynamic testing techniques.

Grafana web security vulnerability opened a plethora of attack possibilities

John Leyden 15 February 2022 at 14:19 UTC

Research | Vulnerabilities | Open Source Software

Visualize this

https://portswigger.net/daily-swig/grafana-web-security-vulnerability-opened-a-plethora-of-attack-possibilities

By default, Flask accepts any content type, including JSON, form-based content types, and "text/plain"

A Python Flask API endpoint

```
1  @app.route('/', methods=['POST'])
2  def my_first_api_endpoint():
3    json_data = json.loads(request.data)
4    ...
5    return "", 200
```

```
1 POST /tasks
2 Host: api.example.com
3 Content-Type: application/json
4
5 { "title": "Drink milk", "description": "Need strong bones" }
```

Sending such a request from the user's browser falls under CORS restrictions enforced by the browser

```
1 POST /tasks
2 Host: api.example.com
3 Content-Type: text/plain
4
5 { "title": "Drink milk", "description": "Need strong bones" }
```

A naïve server accepts anything that parses as JSON, allowing the attacker to bypass CORS restrictions

```
1 POST /tasks
2 Host: api.example.com
3 Content-Type: text/plain; application/json
4
5 { "title": "Drink milk", "description": "Need strong bones" }
```

A lesser naïve server could still be content with seeing "application/json" somewhere in the header

Restricting content types in Flask

```
1  # Decorator to restrict content types
2  def content_type(allowed_content_type):
3    def decorated(f):
4      @wraps(f)
5      def wrapper(*args, **kwargs):
6        ct = request.headers.get('Content-Type', '')
7        if ct.lower() == allowed_content_type.lower():
8          return f(*args, **kwargs)
9
10       raise UnsupportedMediaType
11     return wrapper
12   return decorated
13
14 @app.route('/', methods=['POST'])
15 @content_type('application/json')
16 def my_first_api_endpoint():
17   json_data = request.json
18   ...
19   return "", 200
```

This endpoint only accepts POST requests with the content type set to "application/json"

TOP 10

RESTRICT HTTP CONTENT TYPES

Ensure your API only accepts expected content types, even when the unexpected value looks somewhat correct

When the content type of the response is not properly configured, navigating a browser to an API endpoint can result in code execution in the browser

A typical JSON response from an API

```
1 {
2   "title": "Drink milk",
3   "description": "<script>alert('XSS?')</script>"
4 }
```

Properly set the response type (*application/json*) and add the *X-Content-Type-Options: nosniff* header

Photo by Richard Clark on Unsplash

Overview of best practice header configurations for APIs

```
Strict-Transport-Security: max-age 31536000
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Security-Policy: frame-ancestors 'none'; sandbox; default-src 'none'
```

Apply these headers as a static configuration on every API endpoint

TOP 10

Defense-in-depth for your APIs

Apply browser security headers to avoid unintended side effects from the rendering of API responses.

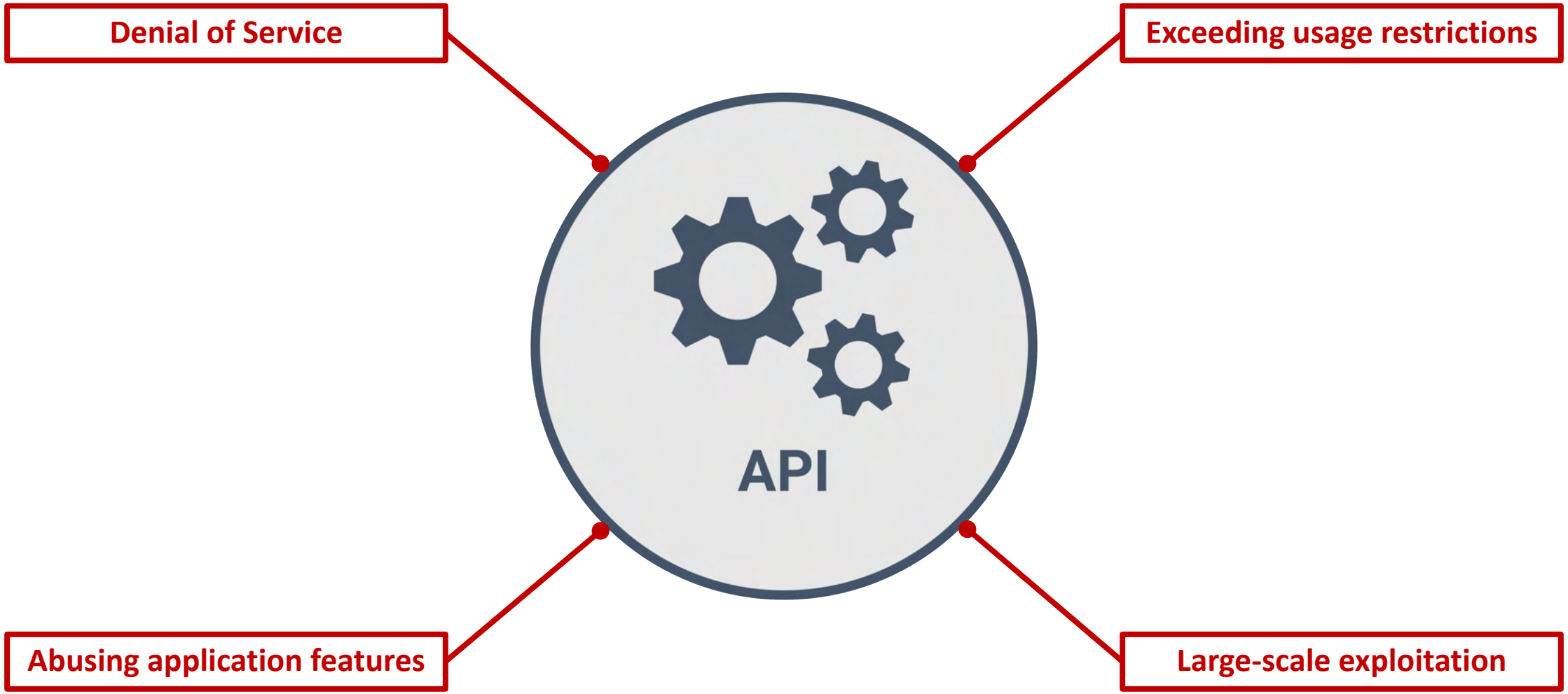
The trade-off between cost and benefit is overwhelmingly positive!

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

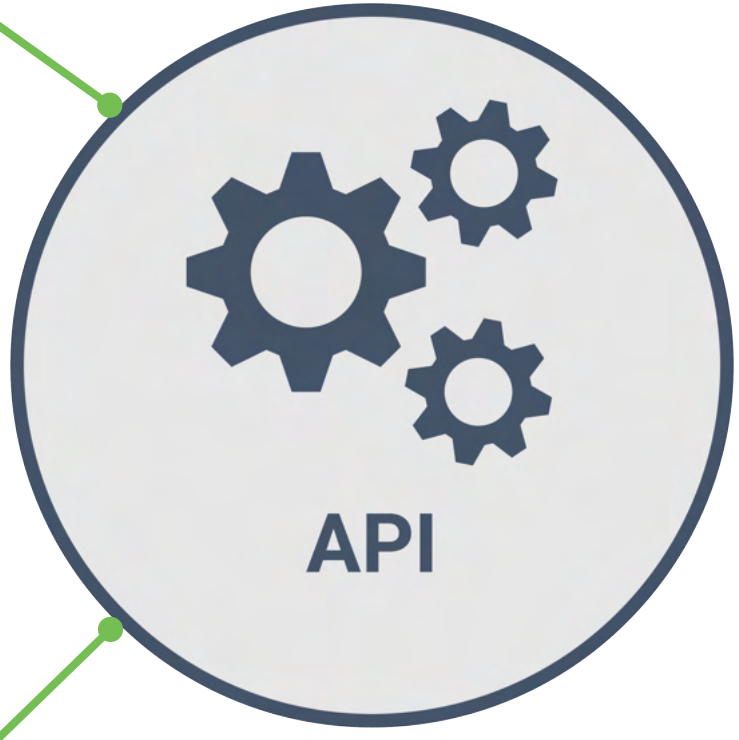

Denial of Service
Exceeding usage restrictions
API
Abusing application features
Large-scale exploitation

TOP 10

It's more than Denial of Service

Denial of Service is only part of the picture. Other abuse scenarios have a more direct impact on the API and its data.

Infrastructure and computation limitations



Application-level input validation

/tasks?page=1&count=10

/tasks?page=1&count=1000000

Failing to enforce an upper limit on seemingly unimportant values can result in DoS

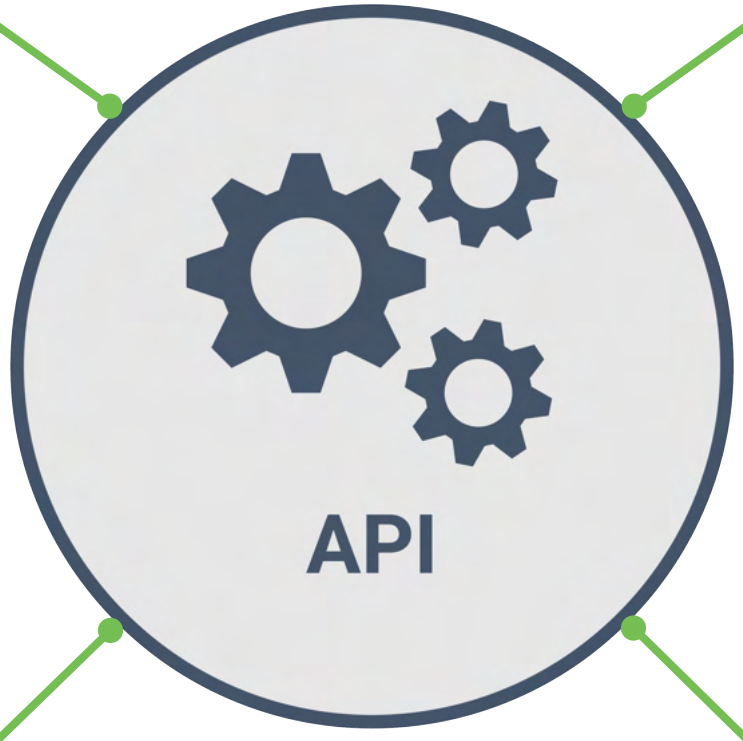
OpenAPI definitions support adding useful limits to values

```yaml
/tasks:
  get:
    parameters:
    - in: query
      name: count
      required: true
      schema:
        type: integer
        minimum: 10
        maximum: 100
```

OpenAPI definitions are still your best friend when it comes to defining expected behavior

Infrastructure and computation limitations

Offloading responsibilities to dedicated services/providers



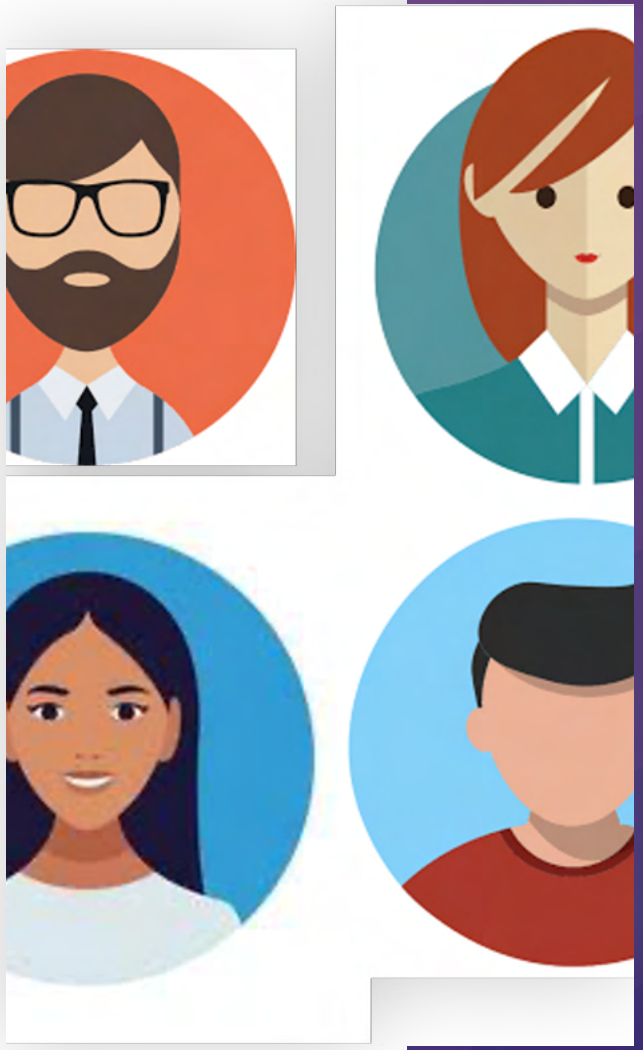
Application-level input validation

Application-specific rate limiting

Polling Question 2: (Multiple Choice)

Are you using GraphQL in your organization?

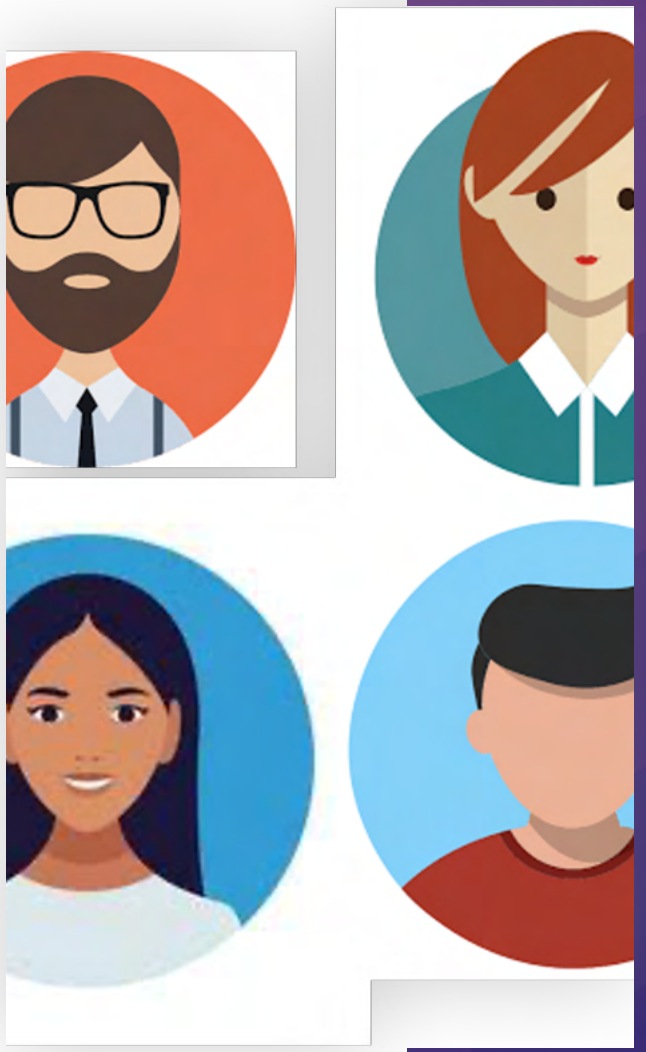
1. Yes, only internally
2. Yes, both internally and externally
3. Not yet, but in the pipeline
4. No, and not planning to



Polling Question 2: (Multiple Choice)

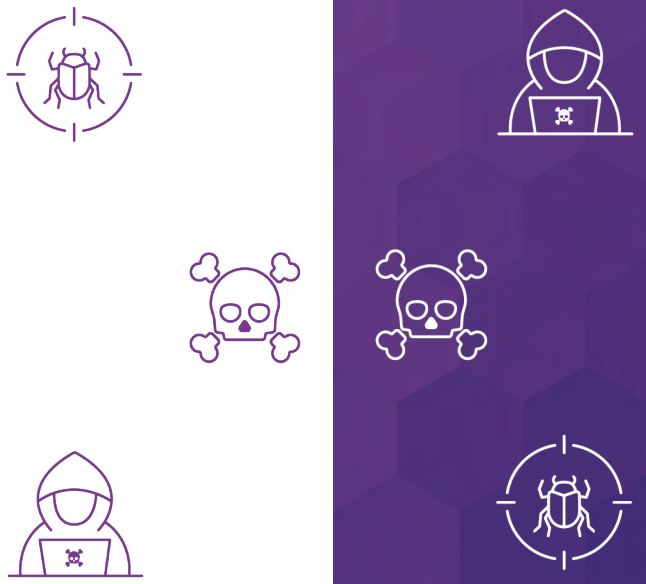
Are you using GraphQL in your organization?

Yes, only internally	11%
Yes, both internally and externally	9%
Not yet, but in the pipeline	30%
No, and not planning to	52%



Attendee Question

How can you assure that third-party APIs (where you in most cases have no view on the code) adhere to best practices?



GraphQL accepts queries as input, making things … interesting

```
query dosQL {
  task (id: 1) {
    project {
      tasks {
        author {
          tasks {
            project
}}}}}}
```

A single GraphQL query can abuse a circular relationship to overload the server

Using graphql-depth-limit *to avoid nested queries*

```
app.use('/graphql', graphqlServer({
  validationRules: [depthLimit(10)]
}));
```

A simple protection mechanism limits the allowed nesting depth of a GraphQL query

Using graphql-validation-complexity *to avoid nested queries*

```
app.use('/graphql', graphqlServer({
  validationRules: [createComplexityLimitRule(1000)]
}));
```

Limit the allowed cost of a GraphQL query using detailed cost estimates

TOP 10

Combine various defensive strategies

Defending against high-volume and feature abuse attacks requires a combination of

- *infrastructure-level security*
- *secure coding guidelines*
- *robust rate limiting mechanisms*
- *compartmentalized service architecture*

NoSQL Injections in Rocket.Chat 3.12.1: How A Small Leak Grounds A Rocket

BY PAUL GERSTE | MAY 18, 2021

Security

Shivam Bathla

May 22, 2020 · 5 min read · Listen

Hacking JWT Tokens: Blind SQLi

23 Nov 2020 | Peter Stöckli

Remote code execution in Elixir-based Paginator

Intro

In August of this year I found a remote code execution vulnerability in the Elixir-based Paginator open-source project from Duffel (a UK-based startup in the flight searching space). The vulnerability has the CVE number CVE-2020-15150 assigned. Since Duffel seemed to use Paginator for its own REST API it seems likely that an attacker exploiting this vulnerability would have been able to execute code on Duffel's (cloud) assets.

https://www.alphabot.com/security/blog/2020/elixir/Remote-code-execution-vulnerability-in-Elixir-based-Paginator-project.html
https://blog.pentesteracademy.com/hacking-jwt-tokens-blind-sqli-efa27958f0e95
https://blog.sonarsource.com/nosql-injections-in-rocket-chat

TOP 10

INJECTION STILL EXISTS

Eradicating injection vulnerabilities requires robust following of secure coding guidelines, complemented with static code analysis

Polling Question 3: (Multiple choice)

What techniques are you using to detect and defend against injection attacks in your APIs?

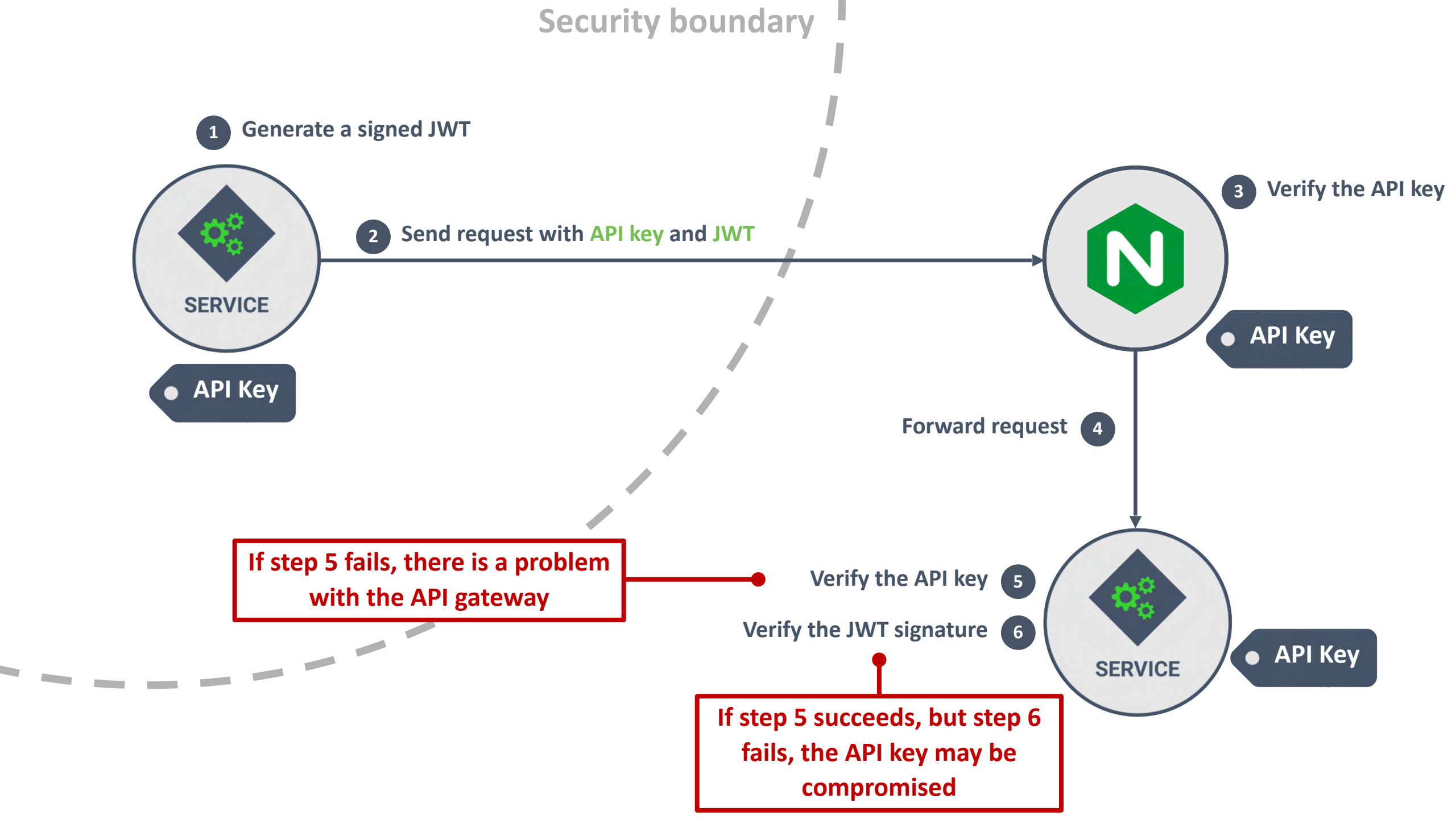
1. SAST
2. DAST
3. Pen-testing
4. Code reviews and inspections

Polling Question 3: (Multiple choice)

What techniques are you using to detect and defend against injection attacks in your APIs?

Different levels of logging serve different purposes

- Logging is often used for debugging or informative purposes
- Make sure you can use your logs as audit trails (*what did a user do during a specific session?*)
- Security-relevant events should be logged as critical log messages


Security boundary
1 Generate a signed JWT
SERVICE
API Key
2 Send request with API key and JWT
3 Verify the API key
API Key
Forward request 4
Verify the API key 5
Verify the JWT signature 6
SERVICE
API Key
If step 5 fails, there is a problem with the API gateway
If step 5 succeeds, but step 6 fails, the API key may be compromised

Different levels of logging serve different purposes

- Logging is often used for debugging or informative purposes
- Make sure you can use your logs as audit trails (*what did a user do during a specific session?*)
- Security-relevant events should be logged as critical log messages

Monitoring turns data gathering into a detective security measure

- Analyze logs and act on critical security-relevant events
- Analyze system behavior (e.g., logs, traffic) and trigger alerts when anomalies are detected
- Setup procedures to follow when abuse scenarios are detected

Photo by Mark Olsen on Unsplash

TOP 10

LOG AND MONITOR

Make logging useful, create high-security alerts of failures that should never occur, and make sure someone follows up on the logs!

Equifax uses *Apache Struts 2* to build applications

a patched version of *Struts2* fixes a remote code execution vulnerability

March 7th, 2017

Renewal of the expired certificate on the monitoring device

July 29th, 2017

July 29th, 2017

Equifax discovers the breach of their systems

May 2017

attackers escalate the attack to full-scale data exfiltration

March 10th, 2017

attackers start probing *Equifax* systems using the *Struts* vulnerability

December 2015

a certificate used by a network monitoring device expires

crunch

Pragmatic Web Security

Security for developers

TOP 10

RUN FIRE DRILLS

Regularly imitate a security incident to ensure that all defenses are working properly

Key takeaways for API security

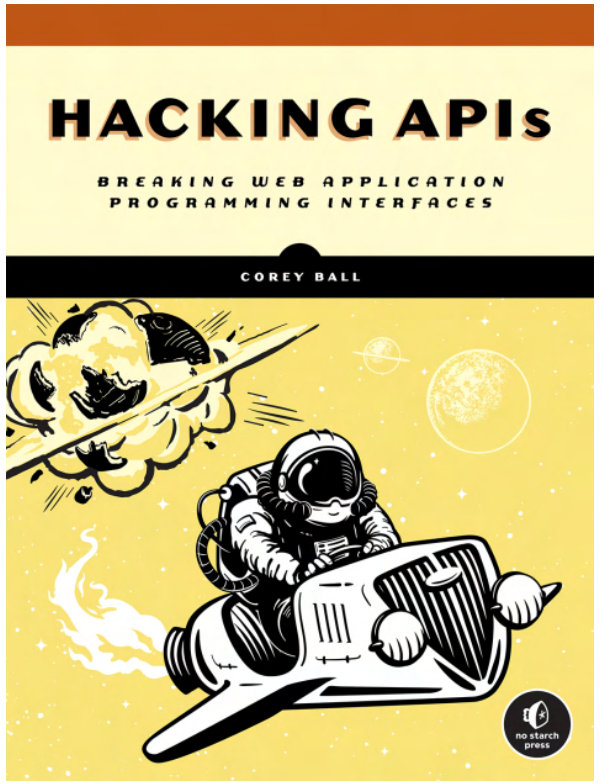
- Specify expected behavior with OpenAPI definitions
- Verify expected behavior on running APIs in their natural habitat
- Readability and auditability enable security

Further Information

APISecurity.io

https://apisecurity.io/

“Hacking APIs”



https://nostarch.com/hacking-apis

Awesome API Security

https://github.com/arainho/awesome-api-security

Upcoming Activity

Further Information

API CONFERENCE

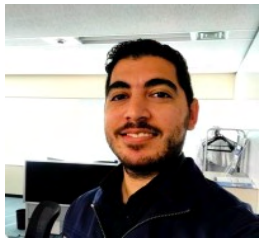
Cisco & 42Crunch
“Adopting a Positive Security Model”


apisecure

42Crunch
“Are your APIs Rugged?”

CyberProof®
A UST Global Company

42Crunch & CyberProof
"How to put the Dev and the Sec into your DevSecOps".

#1 API Security Community Weekly Newsletter

APIsecurity.io

#1 OpenAPI Editor - 400k+ users

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