Architecting API security

Dr. Philippe De Ryck

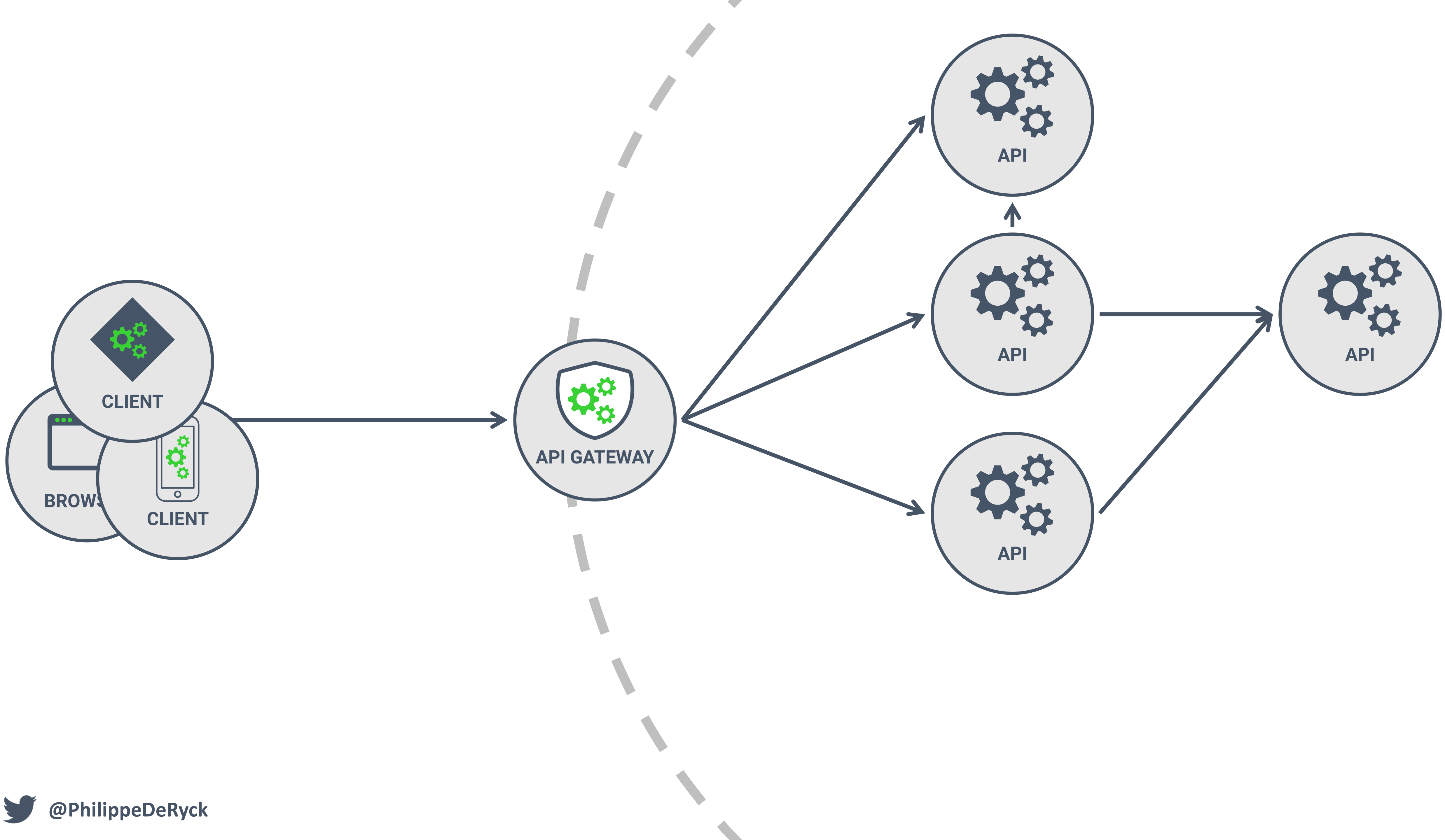
https://Pragmatic Web Security.com

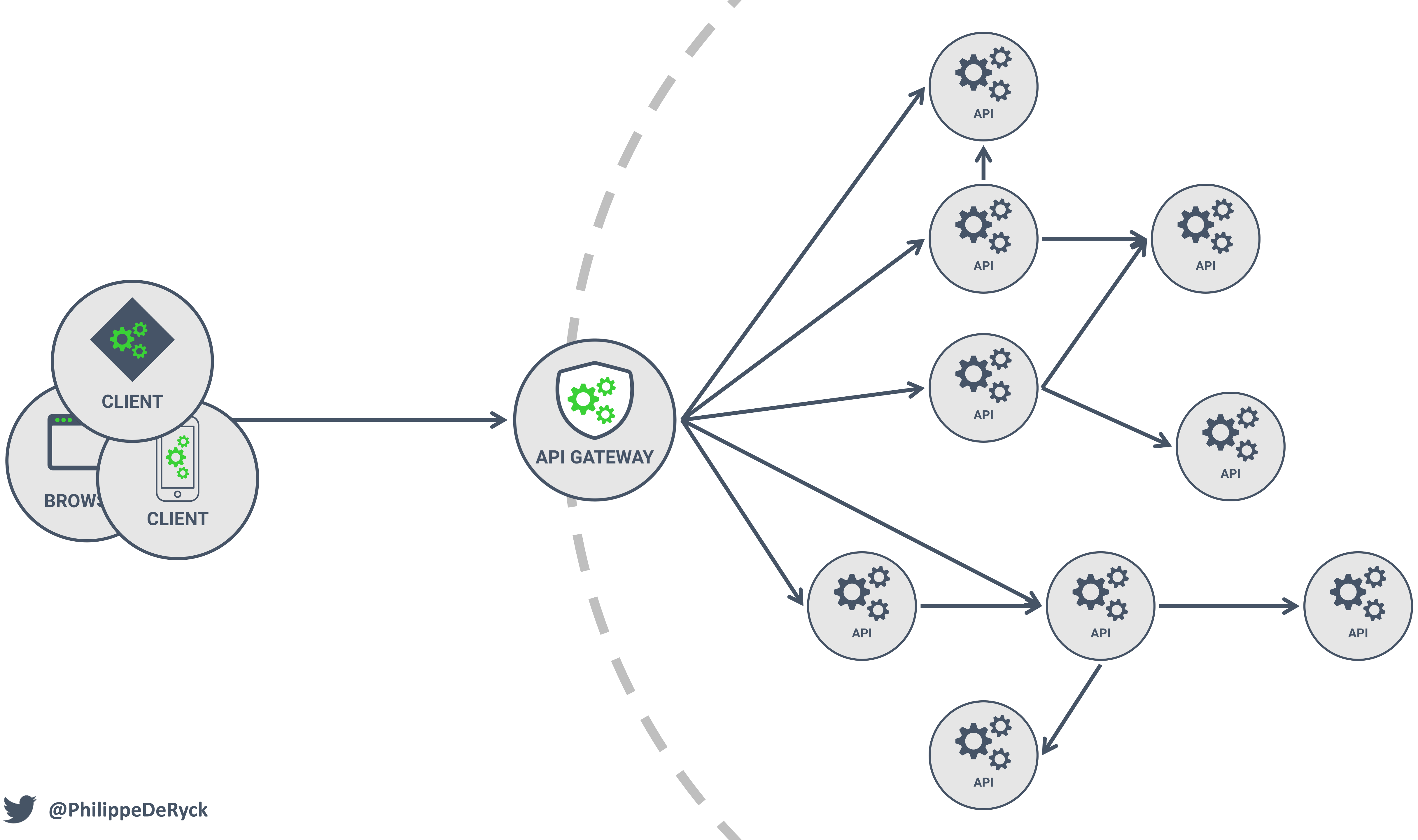
1	Broken object level authorization
2	Broken user authentication
3	Excessive data exposure
4	Lack of resources & rate limiting
5	Broken function level authorization
6	Mass assignment
7	Security misconfiguration
8	Injection
9	Improper assets management
10	Insufficient logging & monitoring

OWASP

API Security

TOP 10


CLIENT
BROWS
CLIENT
API GATEWAY
API
API
API
API



@PhilippeDeRyck

I am *Dr. Philippe De Ryck*

Founder of Pragmatic Web Security

Google Developer Expert

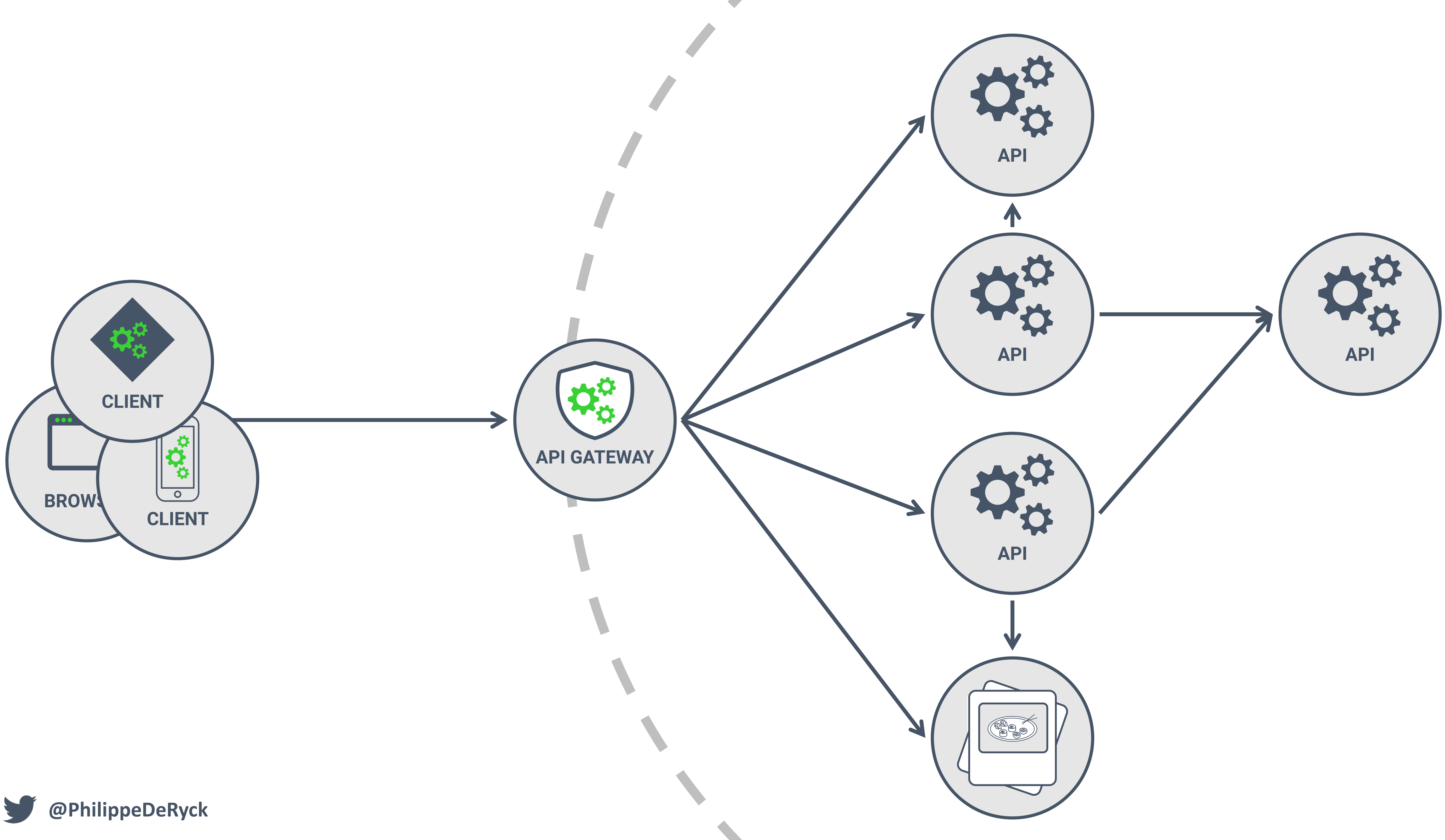
Auth0 Ambassador

SecAppDev organizer

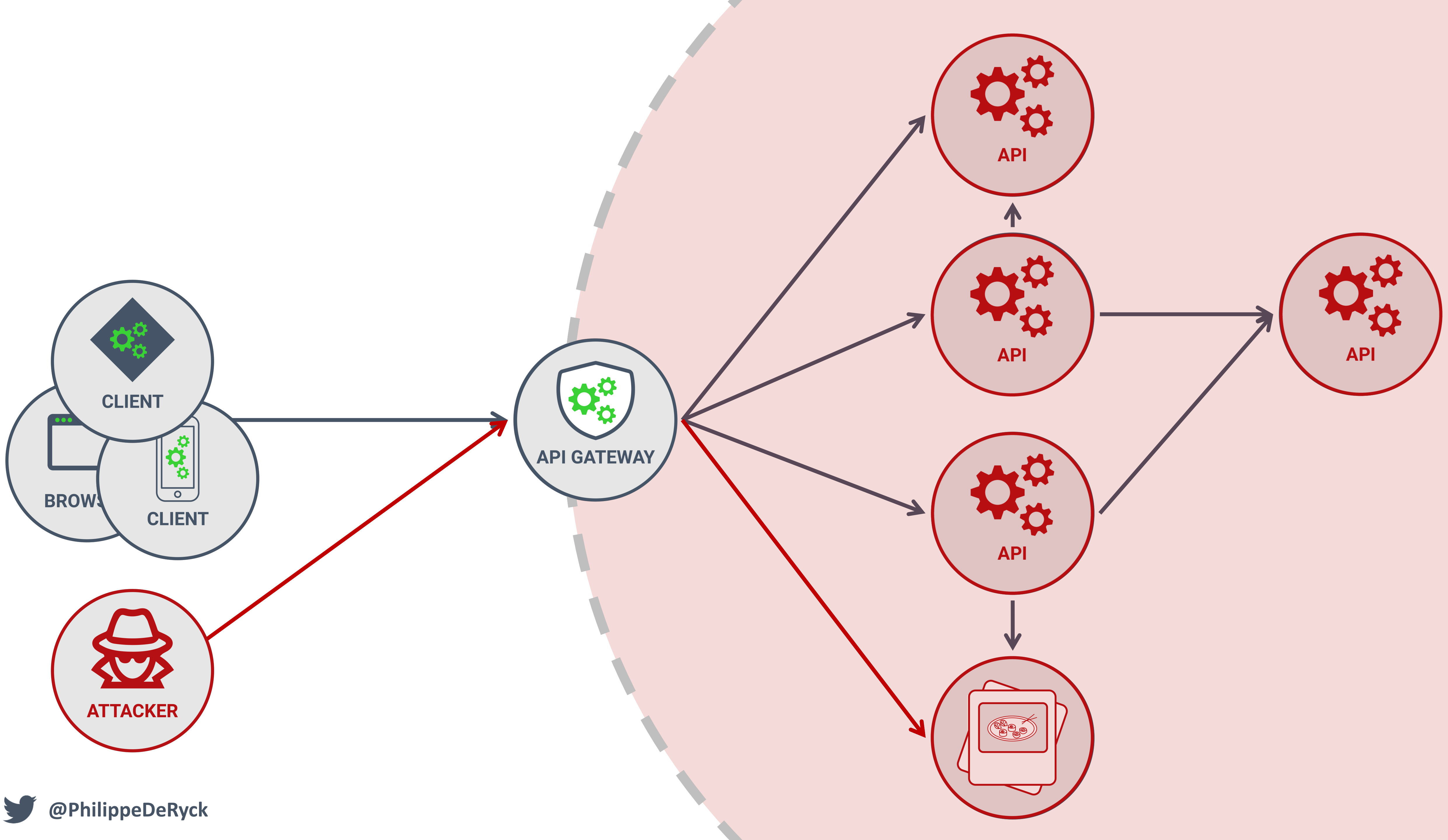
I help developers with security

- Hands-on in-depth security training
- Advanced online security courses
- Security advisory services

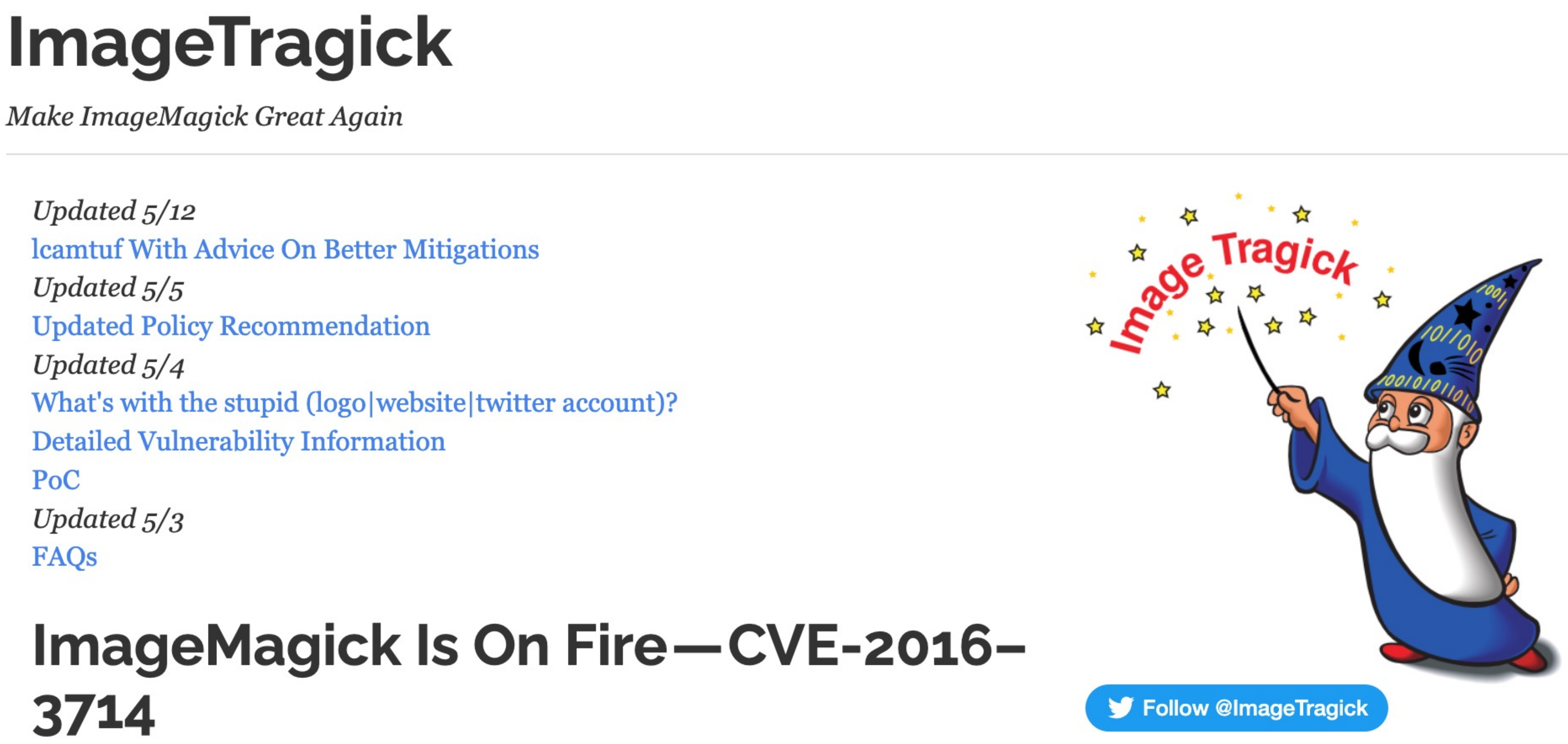
https://pragmaticwebsecurity.com


CLIENT
BROWS
CLIENT
API GATEWAY
API
API
API
API

What happens when 💩 goes wrong?



@PhilippeDeRyck



ImageTragick

Make ImageMagick Great Again

Updated 5/12
lcamtuf With Advice On Better Mitigations
Updated 5/5
Updated Policy Recommendation
Updated 5/4
What's with the stupid (logo|website|twitter account)?
Detailed Vulnerability Information
PoC
Updated 5/3
FAQs

ImageMagick Is On Fire—CVE-2016-3714

https://imagetragick.com/

Perimeter security is dead

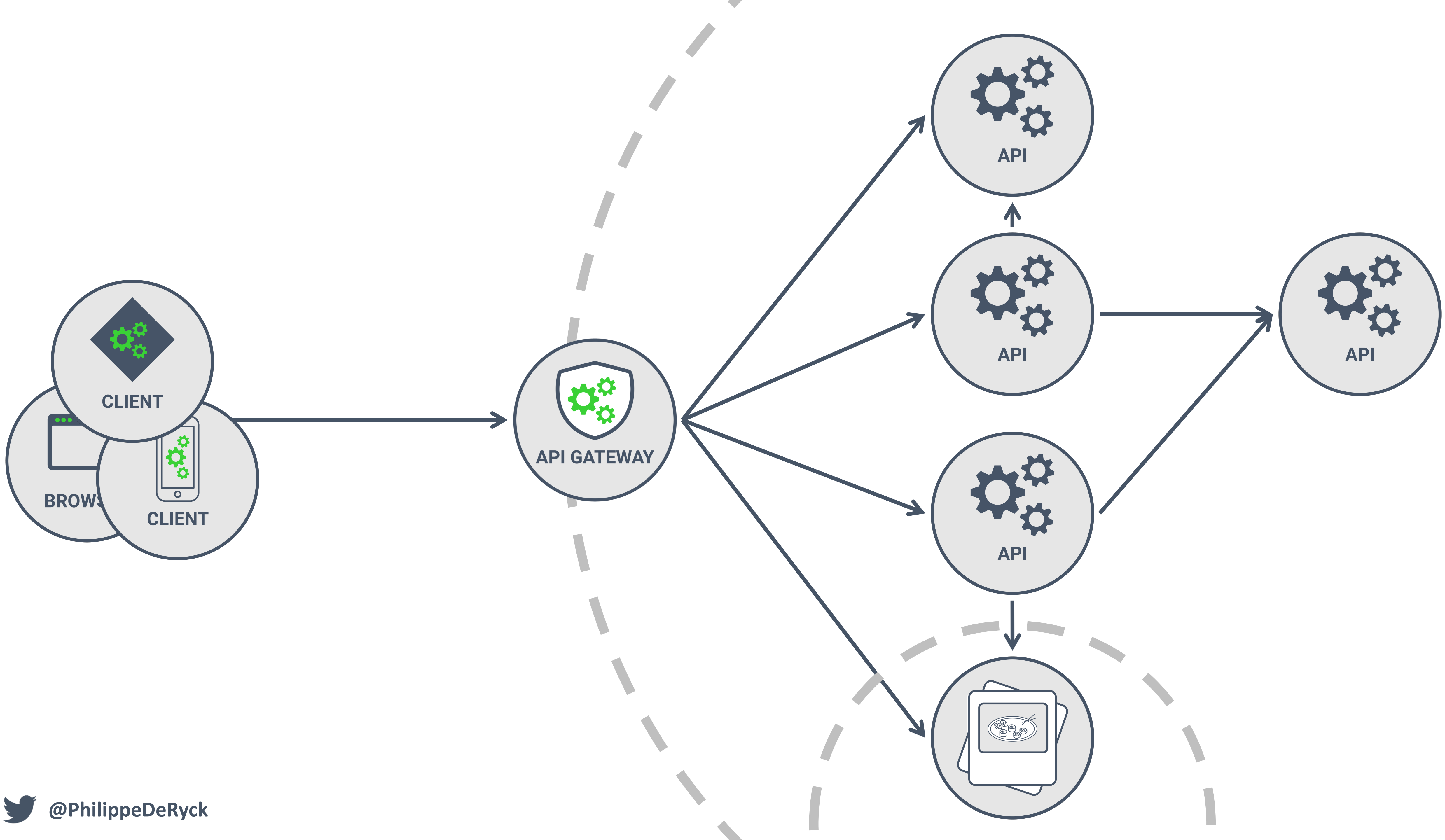
The traditional security boundary at the perimeter can no longer be maintained.

Your perimeter will be breached eventually, so design your systems for that scenario.

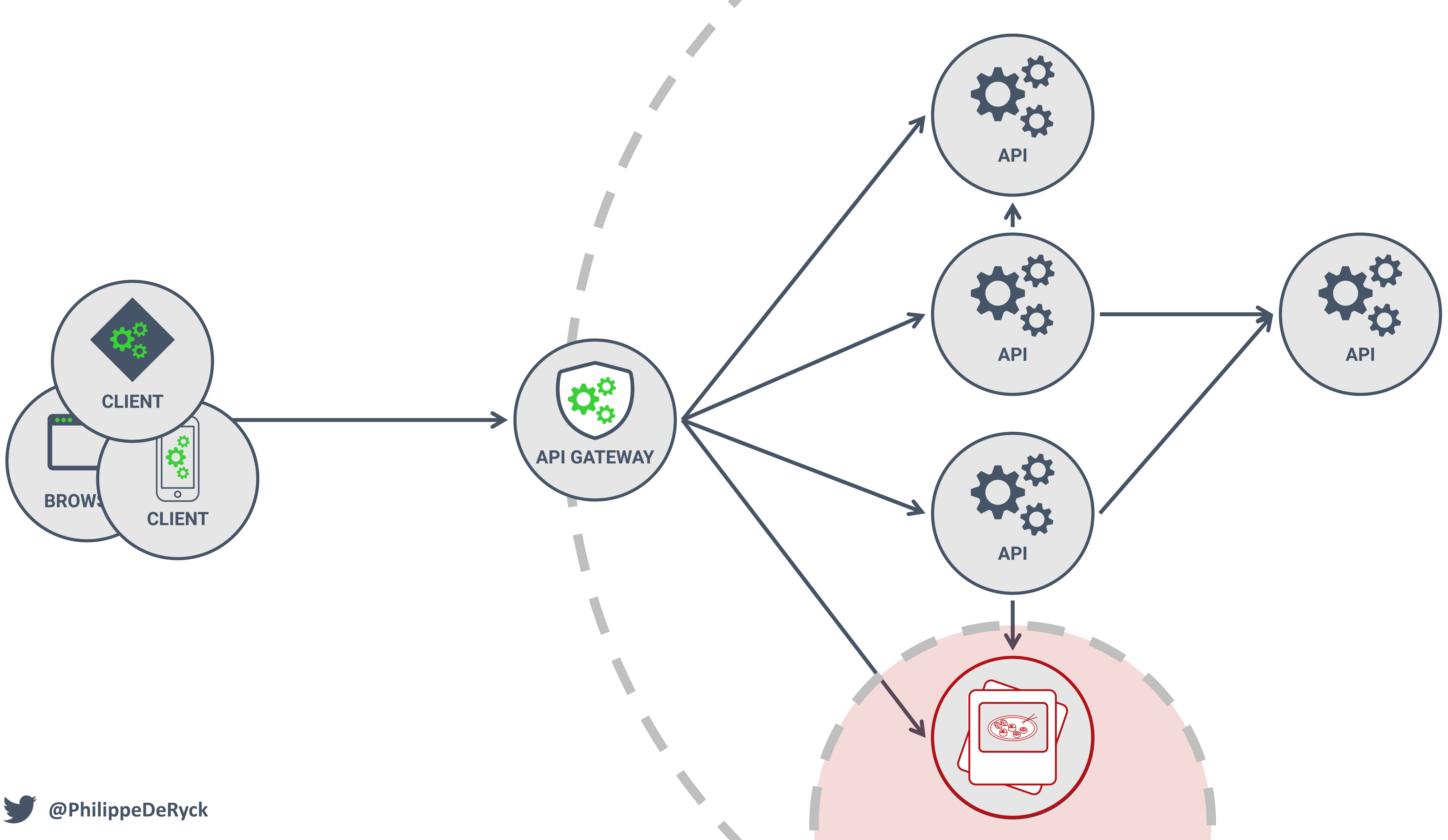
Photo by Richard Clark on Unsplash

Perimeter security is a good primary defense

Stopping attackers at the perimeter is a great defense, as long as it is not the only defense.



@PhilippeDeRyck


CLIENT
BROWS
CLIENT
API GATEWAY
API
API
API
API

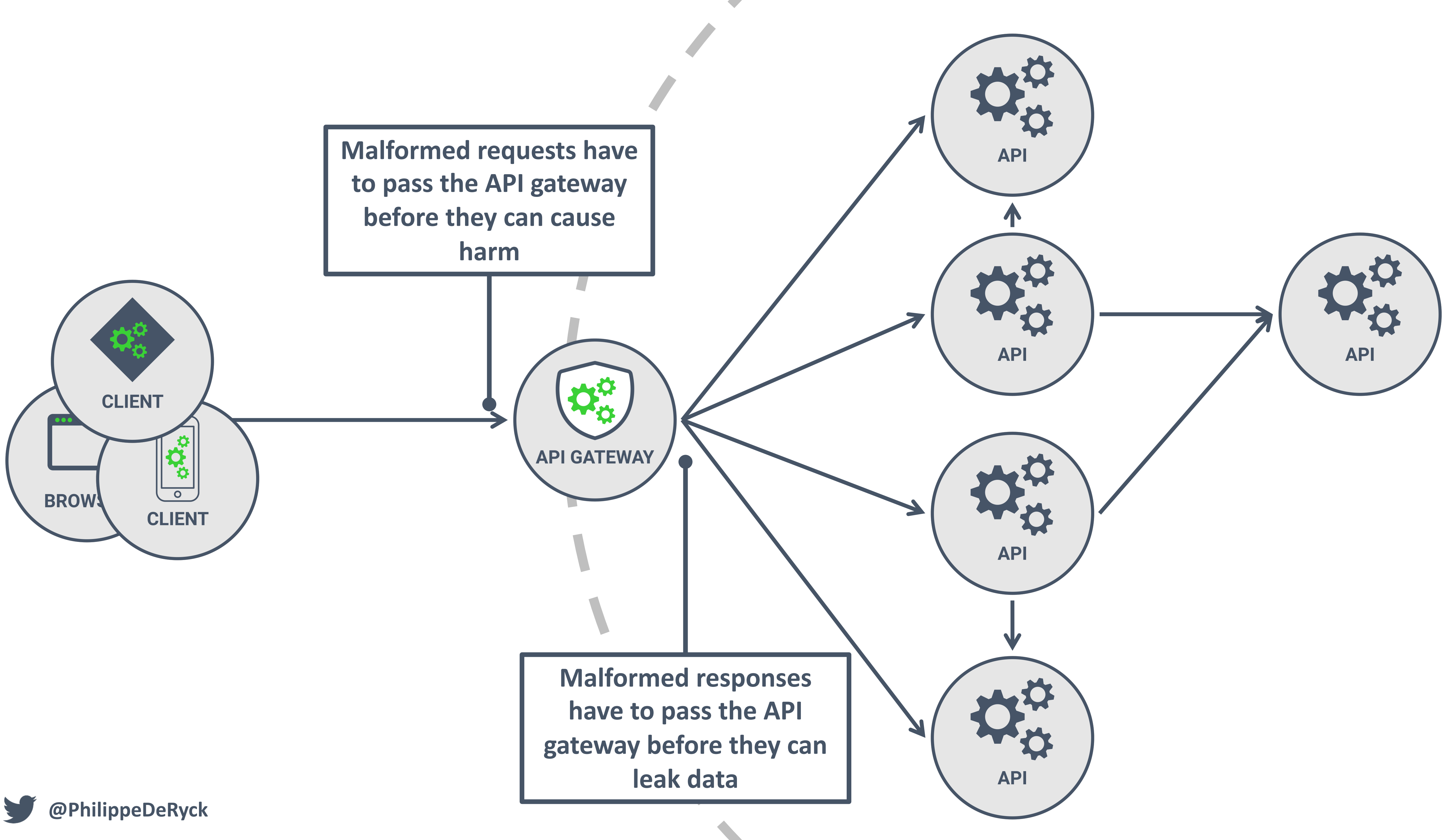
Compartmentalization is crucial

Compartmentalizing the application into different trust zones helps contain the impact of a breach.

@PhilippeDeRyck

Compartmentalization goes both ways

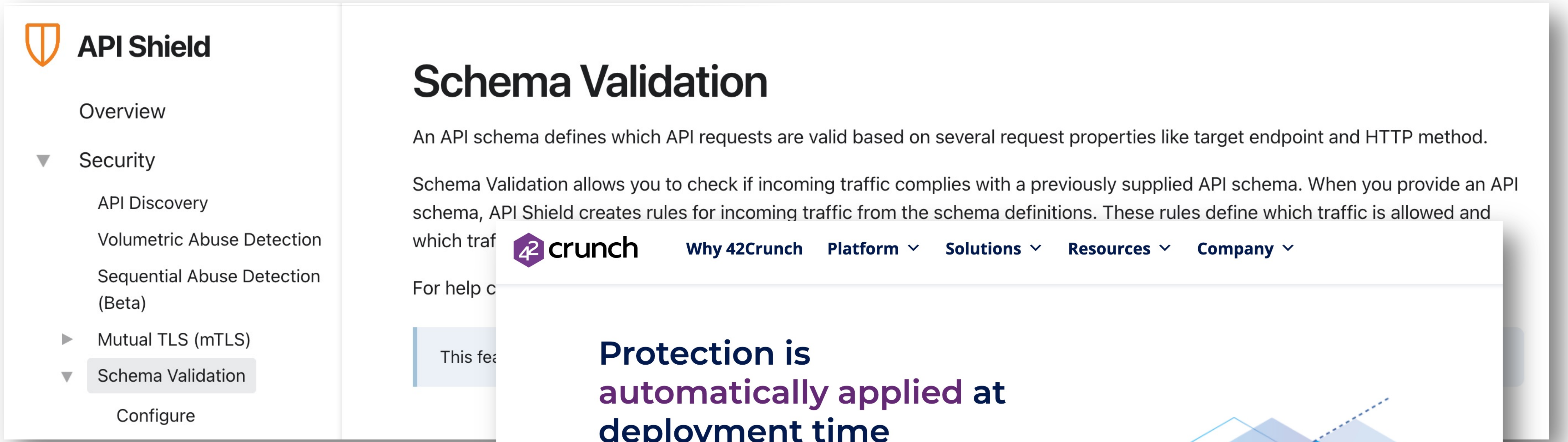
Compartmentalization can be used to isolate untrusted services (sandboxing), or to shield extremely sensitive services.


Malformed requests have to pass the API gateway before they can cause harm

Malformed responses have to pass the API gateway before they can leak data

An OpenAPI contract for an API 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
                      description: The display name of the user
```



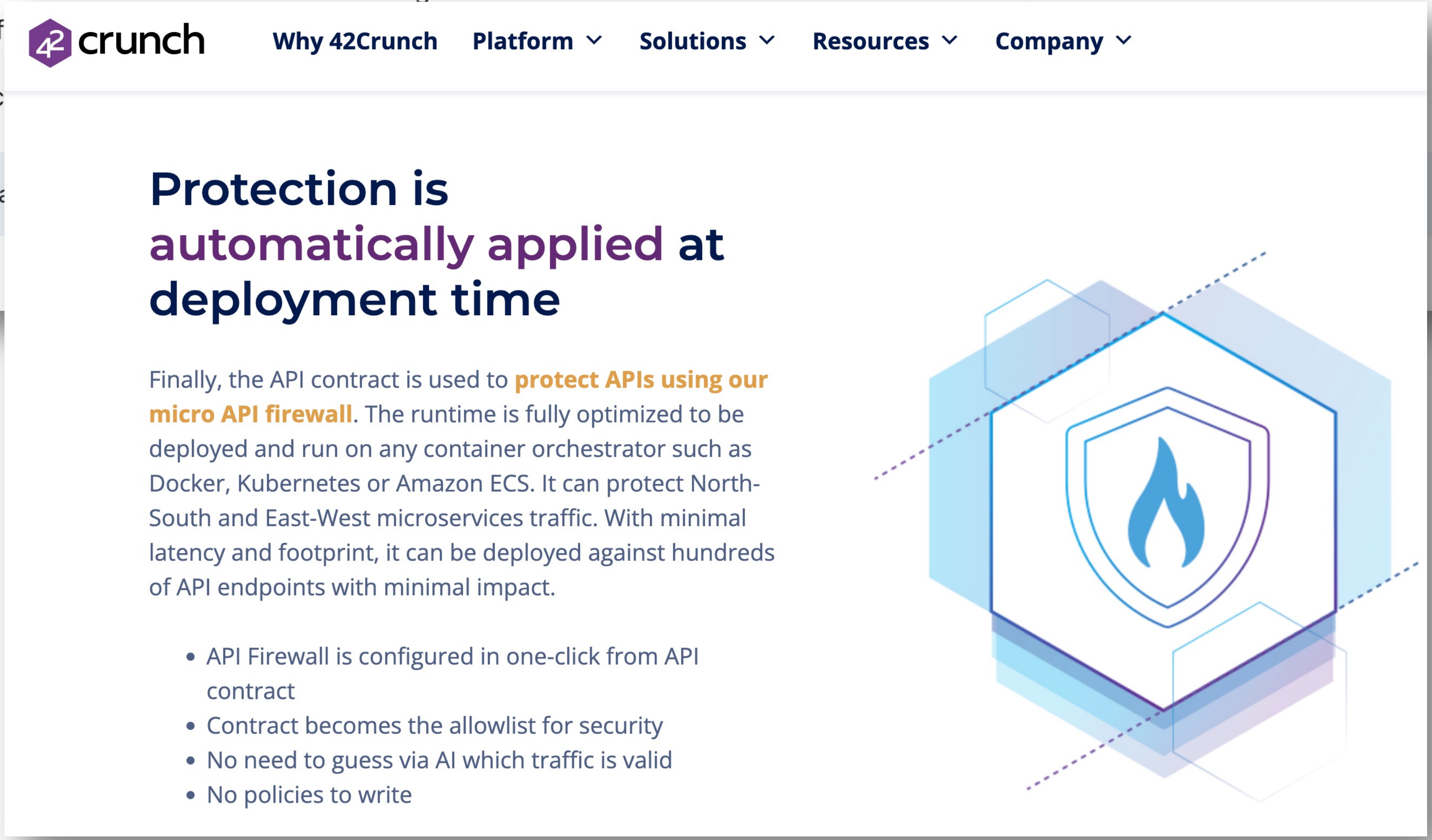
Schema Validation

An API schema defines which API requests are valid based on several request properties like target endpoint and HTTP method.

Schema Validation allows you to check if incoming traffic complies with a previously supplied API schema. When you provide an API schema, API Shield creates rules for incoming traffic from the schema definitions. These rules define which traffic is allowed and which traf

For help c

This fea



Protection is automatically applied at deployment time

Finally, the API contract is used to **protect APIs using our micro API firewall**. The runtime is fully optimized to be deployed and run on any container orchestrator such as Docker, Kubernetes or Amazon ECS. It can protect North-South and East-West microservices traffic. With minimal latency and footprint, it can be deployed against hundreds of API endpoints with minimal impact.

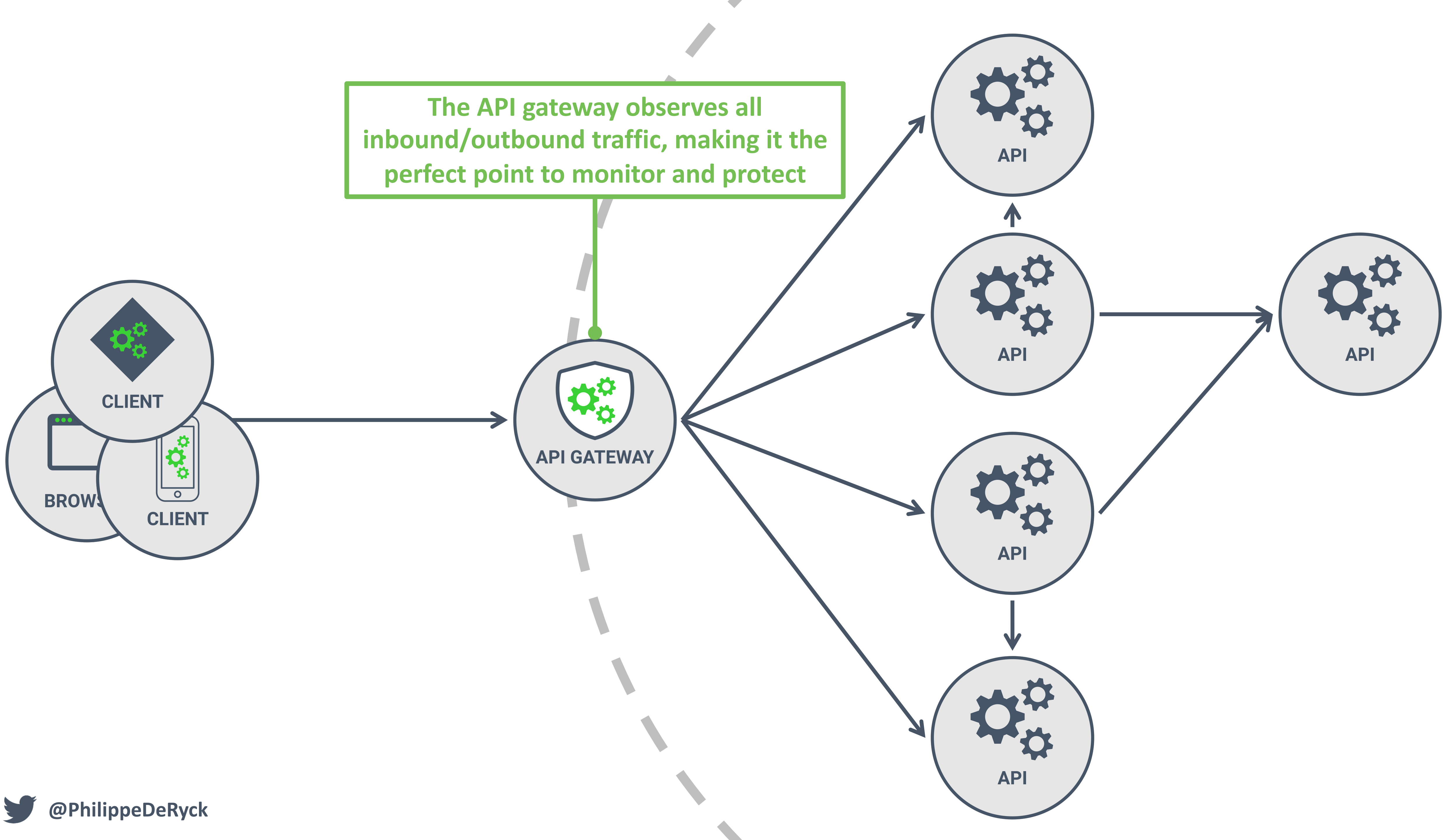
- API Firewall is configured in one-click from API contract
- Contract becomes the allowlist for security
- No need to guess via AI which traffic is valid
- No policies to write

@PhilippeDeRyck

https://42crunch.com/api-security-platform/
https://developers.cloudflare.com/api-shield/security/schema-validation//

Use OpenAPI contracts for security

Use OpenAPI definitions to enforce validity on both requests and responses at the API gateway to avoid sensitive data exposure and mass assignment

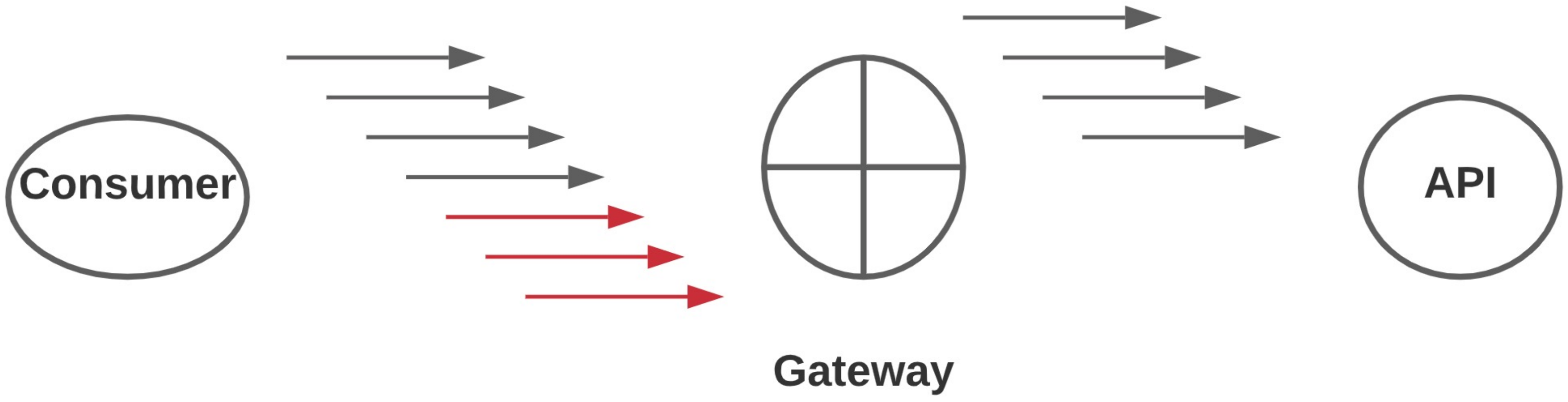

The API gateway observes all inbound/outbound traffic, making it the perfect point to monitor and protect
CLIENT
BROWS
CLIENT
API GATEWAY
API
API
API
API
API

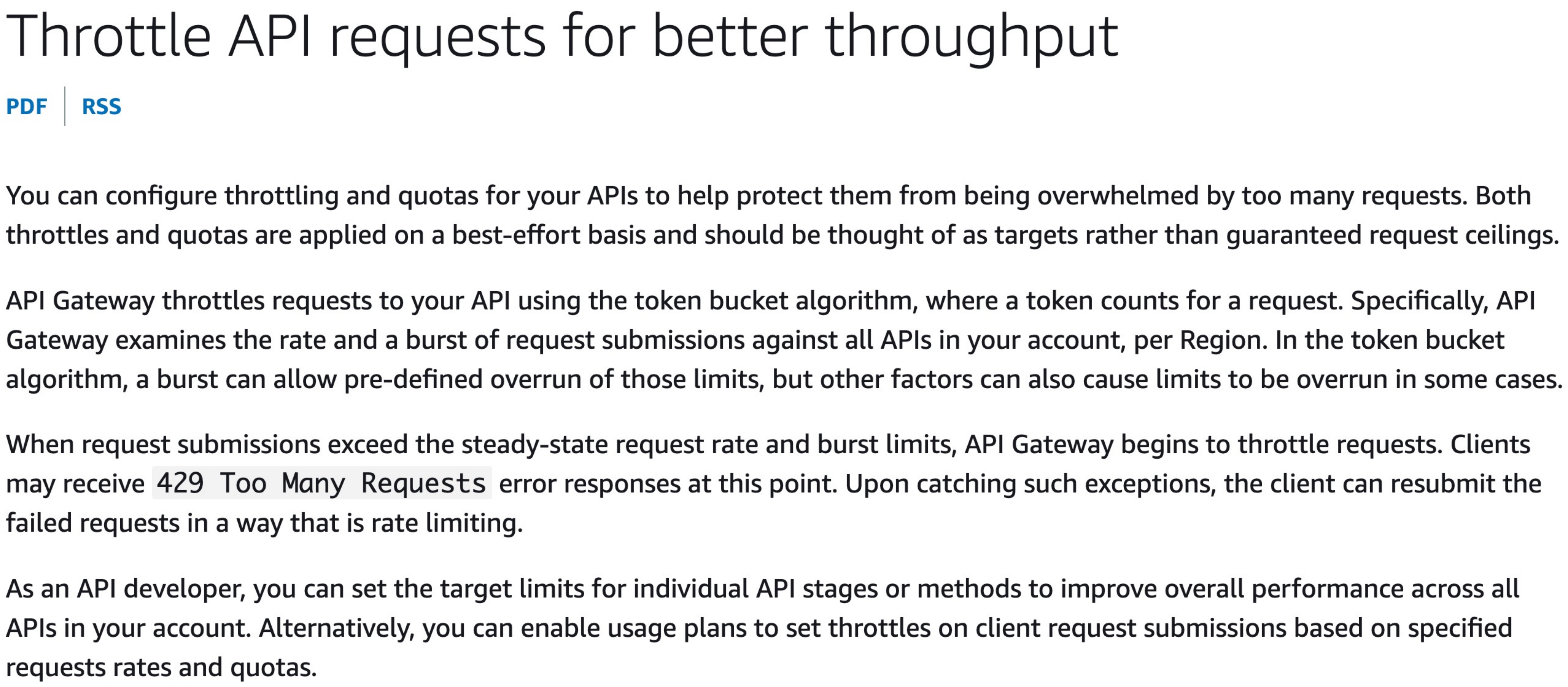
@PhilippeDeRyck

API Rate Limiting with Spring Cloud Gateway

ENGINEERING | HAYTHAM MOHAMED | APRIL 05, 2021 1 COMMENT

One of the imperative architectural concerns is to protect APIs and service endpoints from harmful effects, such as denial of service, cascading failure. or overuse of resources. Rate limiting is a technique to control the rate by which an API or a service is consumed. In a distributed system, no better option exists than to centralize configuring and managing the rate at which consumers can interact with APIs. Only those requests within a defined rate would make it to the API. Any more would raise an HTTP "Many requests" error.





Throttle API requests for better throughput

PDF | RSS

You can configure throttling and quotas for your APIs to help protect them from being overwhelmed by too many requests. Both throttles and quotas are applied on a best-effort basis and should be thought of as targets rather than guaranteed request ceilings.

API Gateway throttles requests to your API using the token bucket algorithm, where a token counts for a request. Specifically, API Gateway examines the rate and a burst of request submissions against all APIs in your account, per Region. In the token bucket algorithm, a burst can allow pre-defined overrun of those limits, but other factors can also cause limits to be overrun in some cases.

When request submissions exceed the steady-state request rate and burst limits, API Gateway begins to throttle requests. Clients may receive `429 Too Many Requests` error responses at this point. Upon catching such exceptions, the client can resubmit the failed requests in a way that is rate limiting.

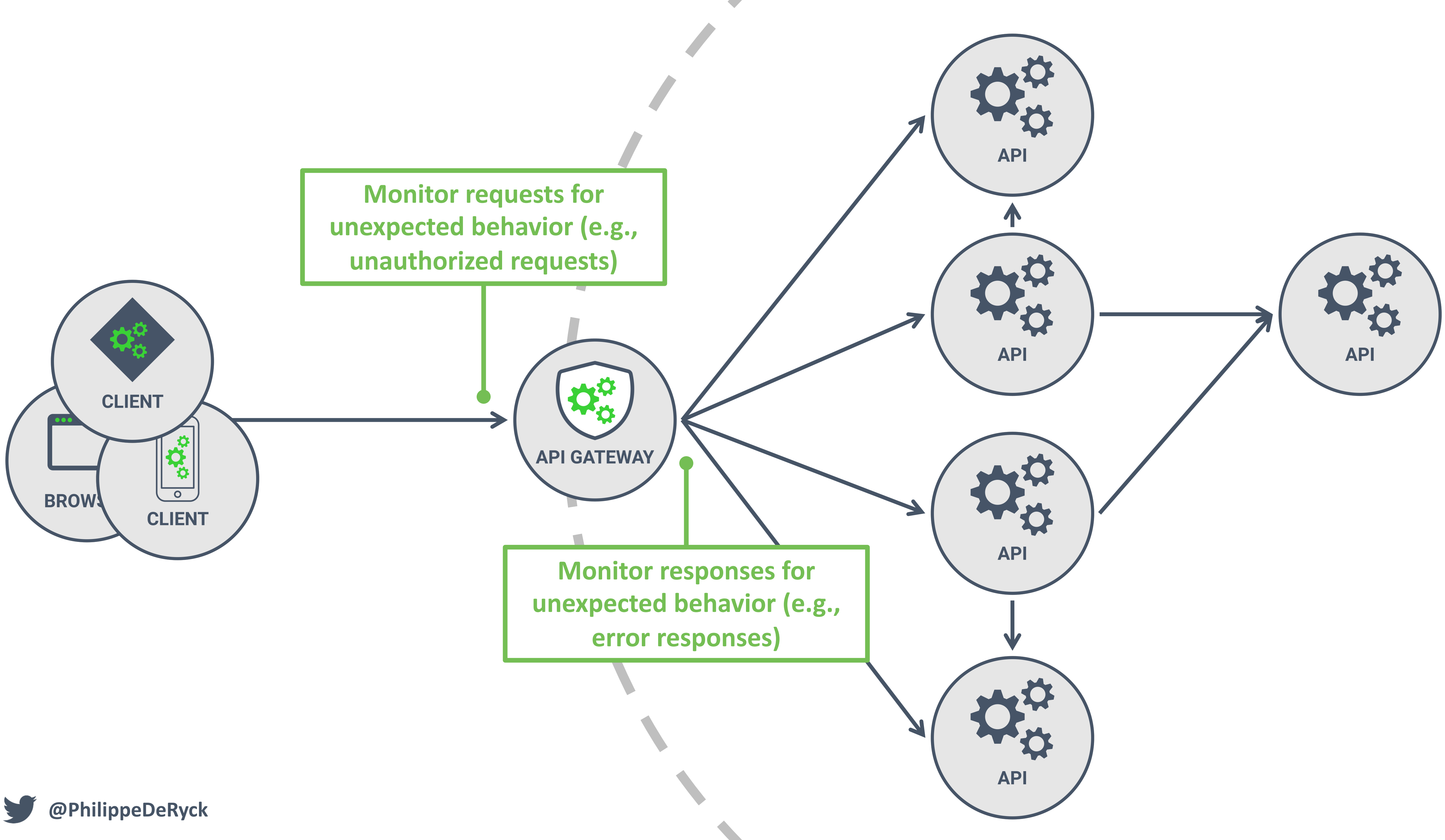
As an API developer, you can set the target limits for individual API stages or methods to improve overall performance across all APIs in your account. Alternatively, you can enable usage plans to set throttles on client request submissions based on specified requests rates and quotas.

@PhilippeDeRyck

https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-request-throttling.html

Apply rate limiting at the gateway

The API gateway can apply generic rate limiting mechanisms, or API-specific rate limiting or usage restriction policies


Monitor requests for unexpected behavior (e.g., unauthorized requests)
CLIENT
BROWS
CLIENT
API GATEWAY
Monitor responses for unexpected behavior (e.g., error responses)
API
API
API
API
API

@PhilippeDeRyck

Apigee > Documentation > Guides

Was this helpful?

About anomaly detection

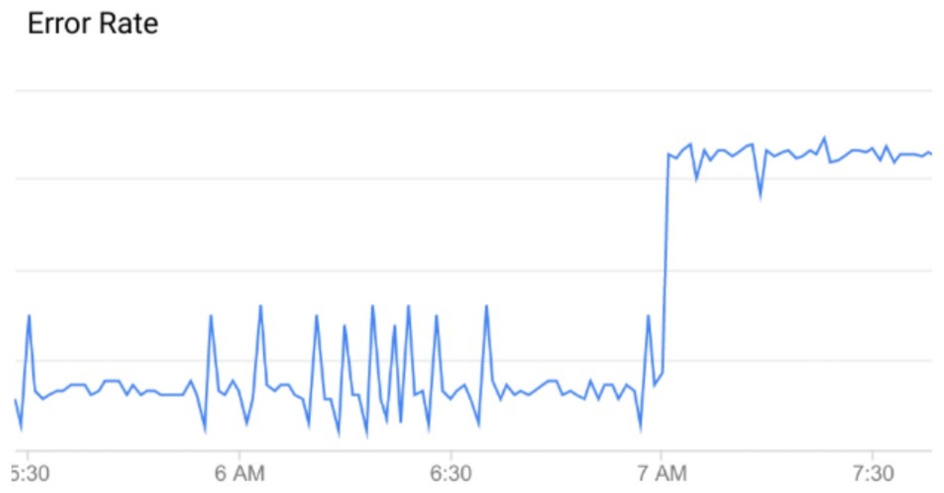
Send feedback

*You're viewing **Apigee X** documentation.*

View Apigee Edge documentation.

What is an anomaly?

An *anomaly* is an unusual or unexpected API data pattern. For example, take a look at the graph of API error rate below:



@PhilippeDeRyck

https://cloud.google.com/apigee/docs/aapi-ops/about-anomaly-detection

Anomaly detection helps discover problems

Time-based anomaly detection reduces the manual overhead for monitoring and helps discover functional problems

How can we use anomaly detection to improve the security of our APIs?

Detect mass extraction of data from the application (e.g., typical breach behavior)

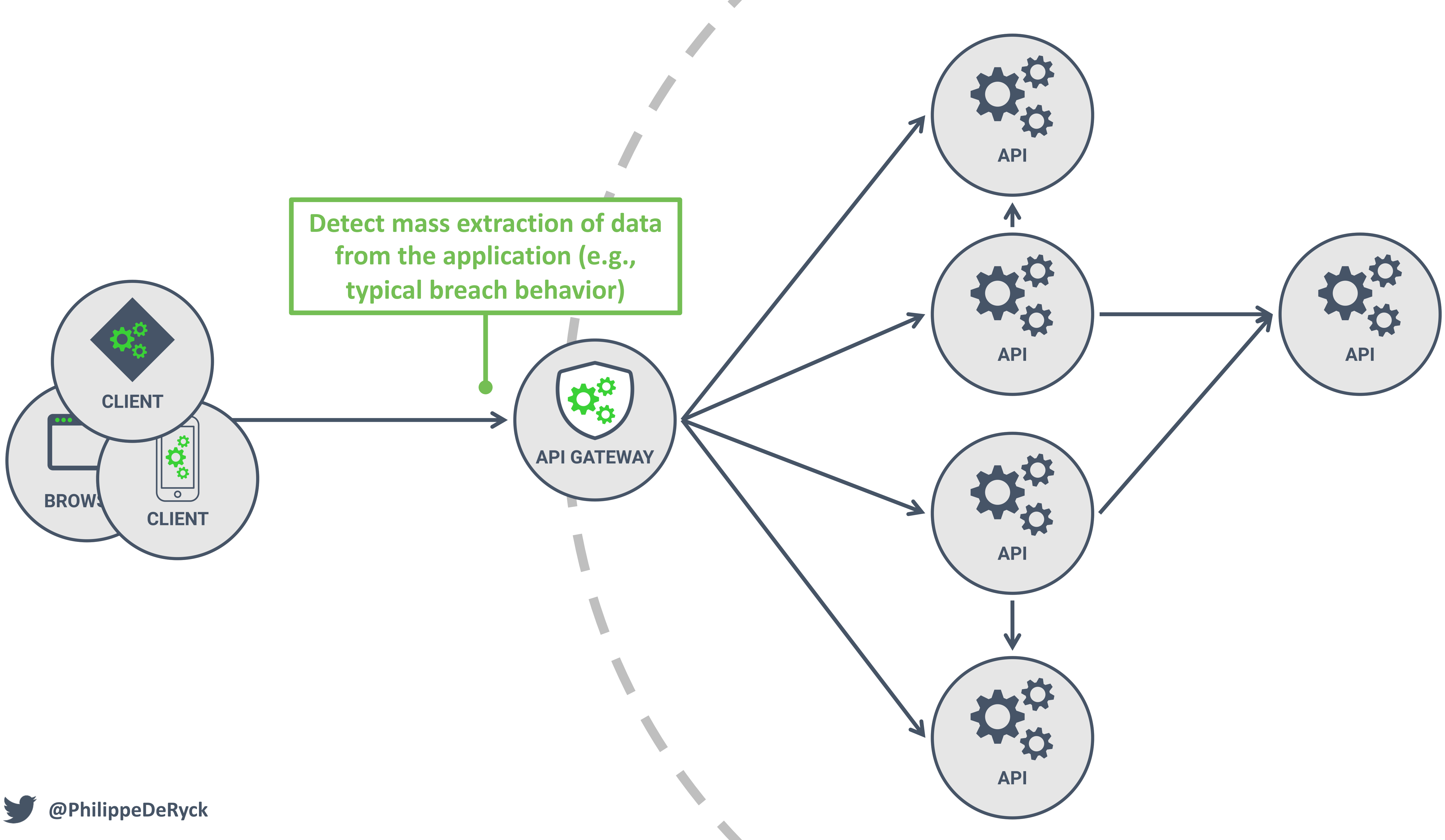


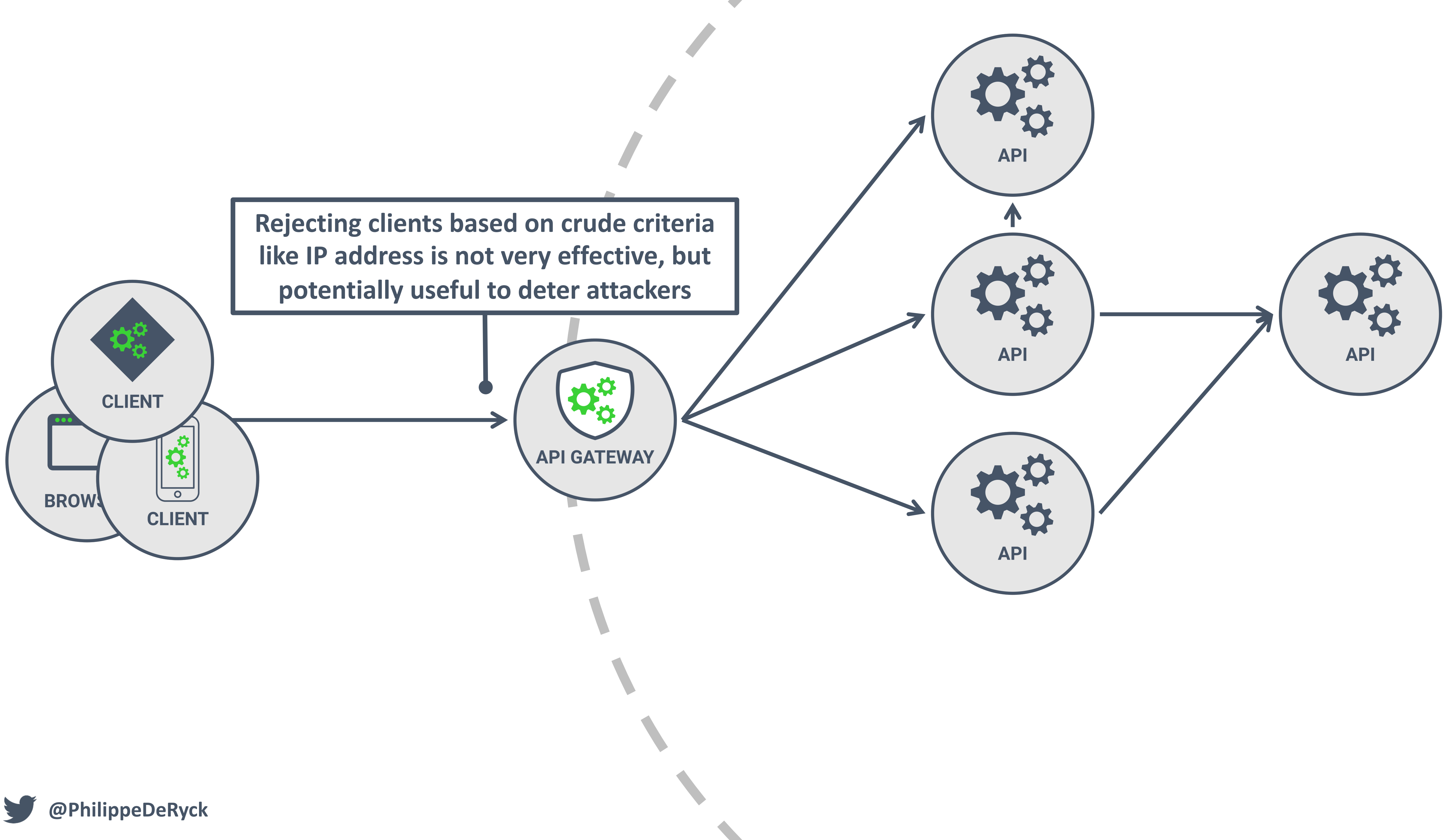
Photo by Mark Olsen on Unsplash

USING MONITORING TO DETECT SECURITY PROBLEMS

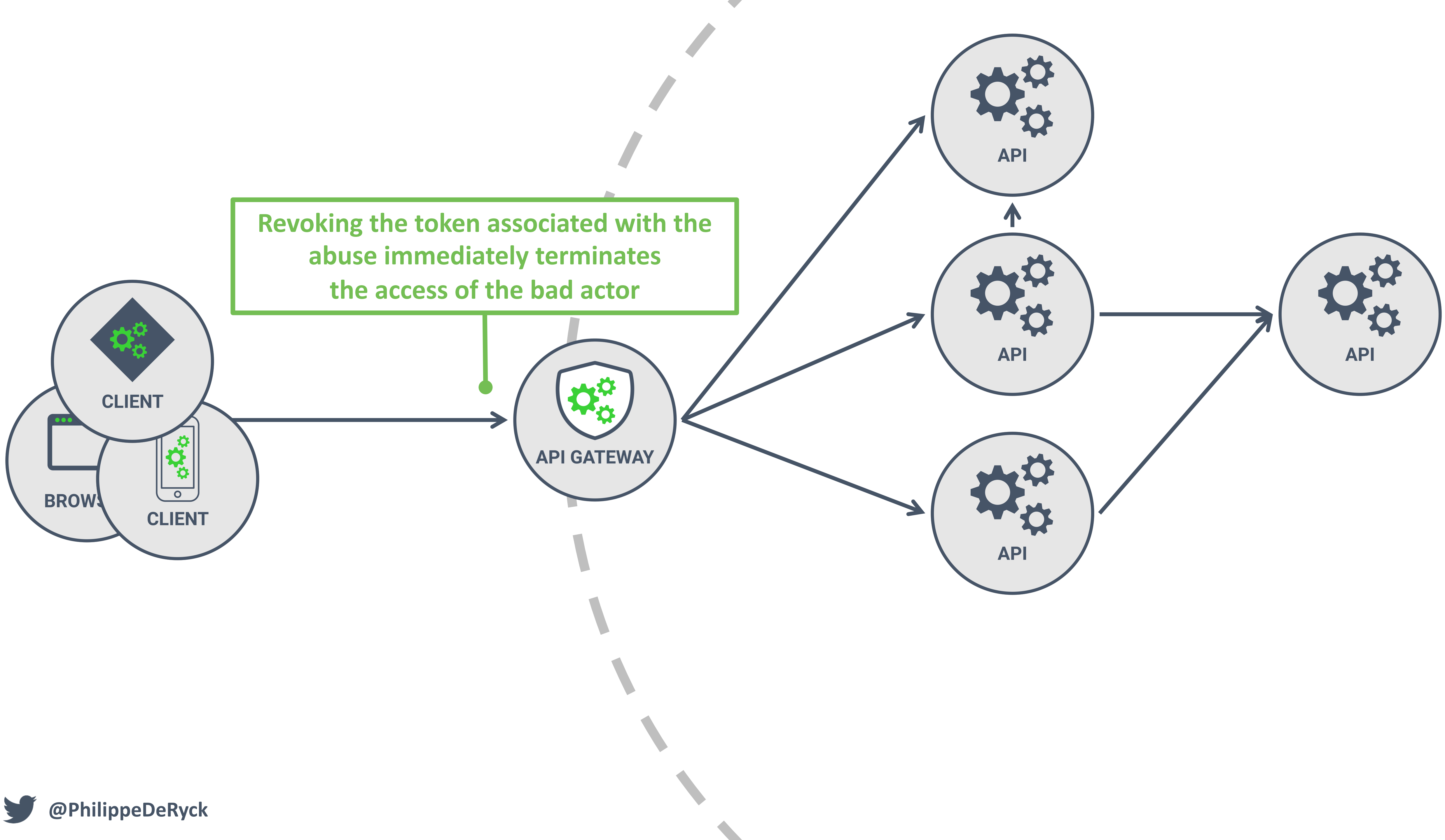
- Data breaches often go undetected until the data surfaces
  - Extracting millions of DB entries or documents makes a lot of noise
  - Monitoring traffic is essential to detect breaches as they are happening
- Traditional traffic analysis can be used to detect high-volume attacks
  - E.g., exploiting a BOLA vulnerability often results in sudden spikes to a specific endpoint
  - Setup alerts when traffic anomalies are detected
  - Focus on avoiding false positives to preserve the value of the alert
- Canaries offer much more reliable signals to detect a breach
  - Include "canary data" in the database that is never used by the legitimate application
    - E.g., a user profile with a sequential ID that belongs to a non-existent user
  - Setup monitoring to detect the canary and sound alarm bells

Setup monitoring for security purposes

Use traffic monitoring and canary data to detect a potential data breach as soon as possible.

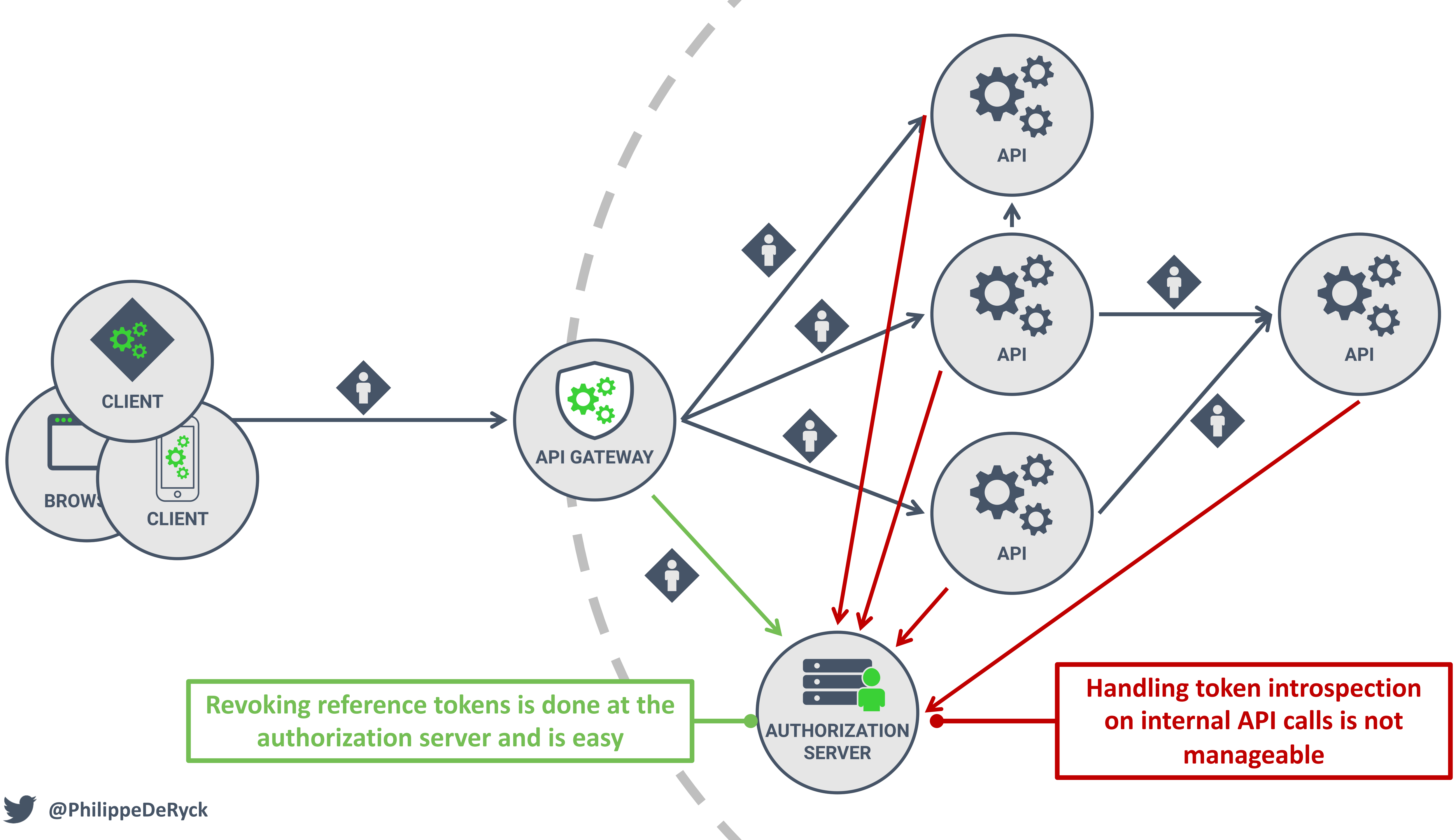

Rejecting clients based on crude criteria like IP address is not very effective, but potentially useful to deter attackers

@PhilippeDeRyck



Revoking the token associated with the abuse immediately terminates the access of the bad actor

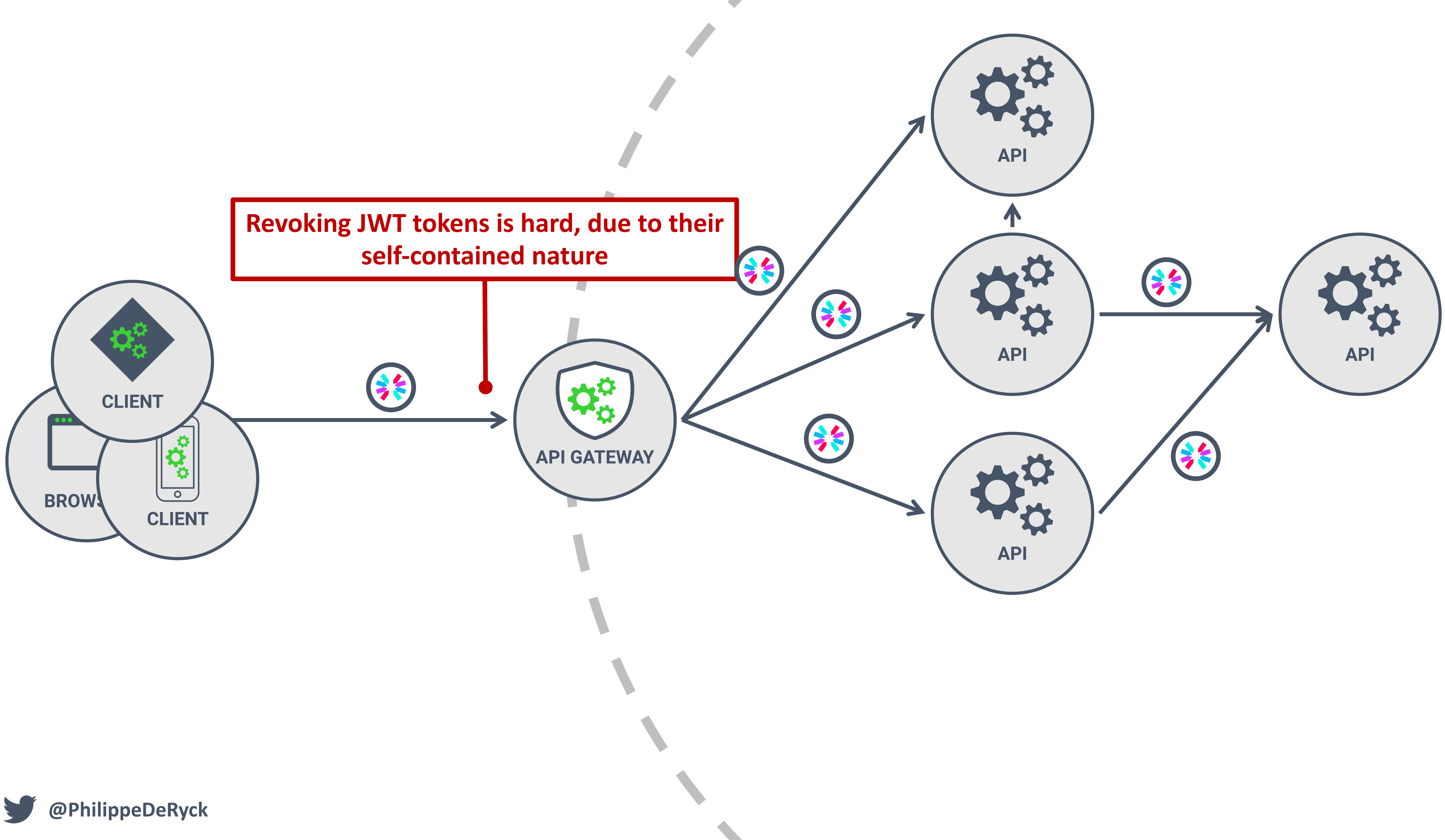
@PhilippeDeRyck



Revoking reference tokens is done at the authorization server and is easy

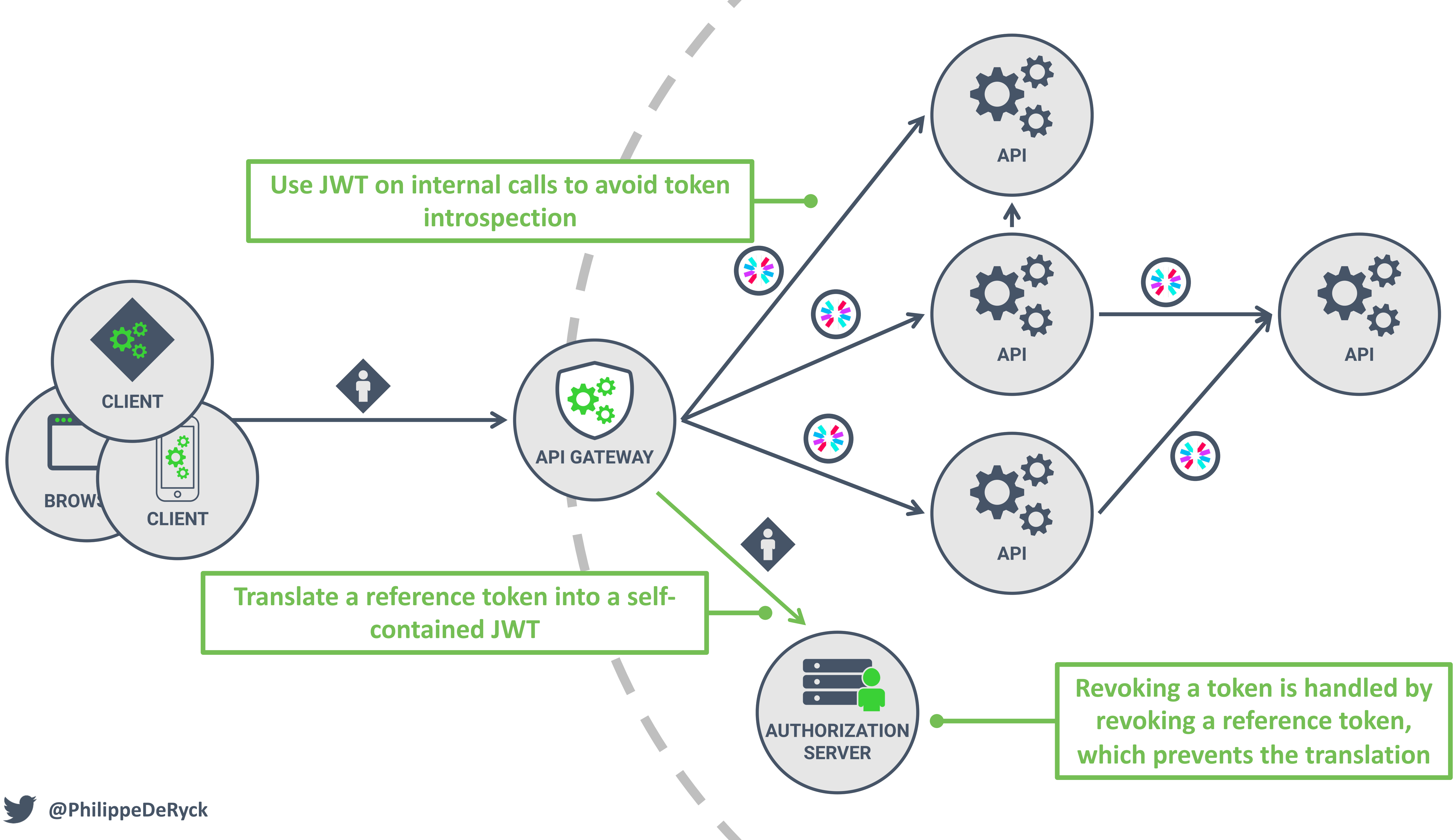
Handling token introspection on internal API calls is not manageable

@PhilippeDeRyck



Revoking JWT tokens is hard, due to their self-contained nature

@PhilippeDeRyck



Use JWT on internal calls to avoid token introspection

Translate a reference token into a self-contained JWT

Revoking a token is handled by revoking a reference token, which prevents the translation

@PhilippeDeRyck

Follow up on detection with automatic defenses

Detecting ongoing problems is useful, but automatically taking defensive action is even better.

Use block lists or revocation mechanisms to reject malicious traffic.

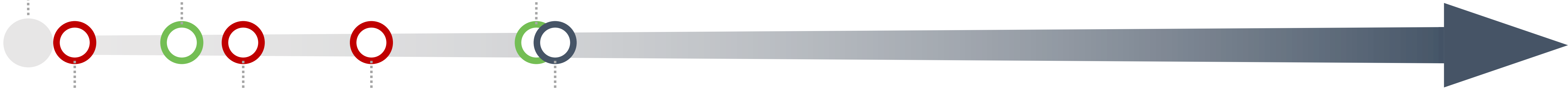
Equifax uses *Apache Struts 2* to build applications

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

@PhilippeDeRyck

Run fire drills

Regularly imitate a security incident to ensure that the detection mechanisms, defenses, and processes all work as expected

Now it is up to you ...

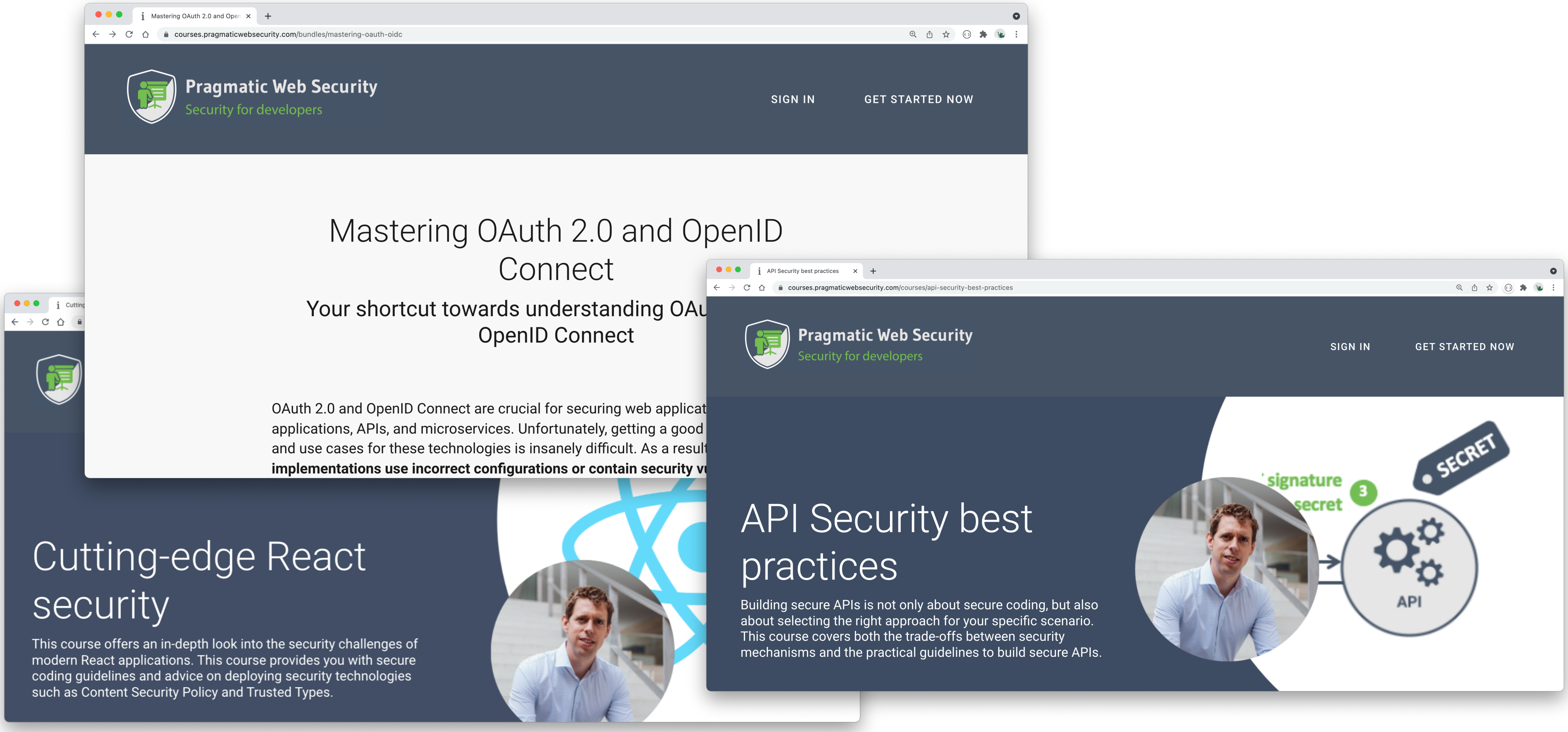
Hope for the best, plan for the worst

Use the API gateway to shield internal details from clients

Rely on the API gateway to protect requests and responses

...

Want more in-depth security content?



Mastering OAuth 2.0 and Oper

courses.pragmaticwebsecurity.com/bundles/mastering-oauth-oidc

Pragmatic Web Security
Security for developers

SIGN IN GET STARTED NOW

Mastering OAuth 2.0 and OpenID Connect

Your shortcut towards understanding OAu
OpenID Connect

OAuth 2.0 and OpenID Connect are crucial for securing web applicat
applications, APIs, and microservices. Unfortunately, getting a good
and use cases for these technologies is insanely difficult. As a result
implementations use incorrect configurations or contain security v

Cutting

Cutting-edge React security

This course offers an in-depth look into the security challenges of modern React applications. This course provides you with secure coding guidelines and advice on deploying security technologies such as Content Security Policy and Trusted Types.

API Security best practices

courses.pragmaticwebsecurity.com/courses/api-security-best-practices

Pragmatic Web Security
Security for developers

SIGN IN GET STARTED NOW

API Security best practices

Building secure APIs is not only about secure coding, but also about selecting the right approach for your specific scenario. This course covers both the trade-offs between security mechanisms and the practical guidelines to build secure APIs.

HTTPS://COURSES.PRAGMATICWEBSECURITY.COM

Thank you!

Connect on social media
to stay in touch on security

@PhilippeDeRyck

/in/PhilippeDeRyck