42crunch

17 February 2023

Everything You Need to Know About API Security

Colin Domoney

Chief Technology Evangelist

@colindomoney

About the speaker

Colin Domoney

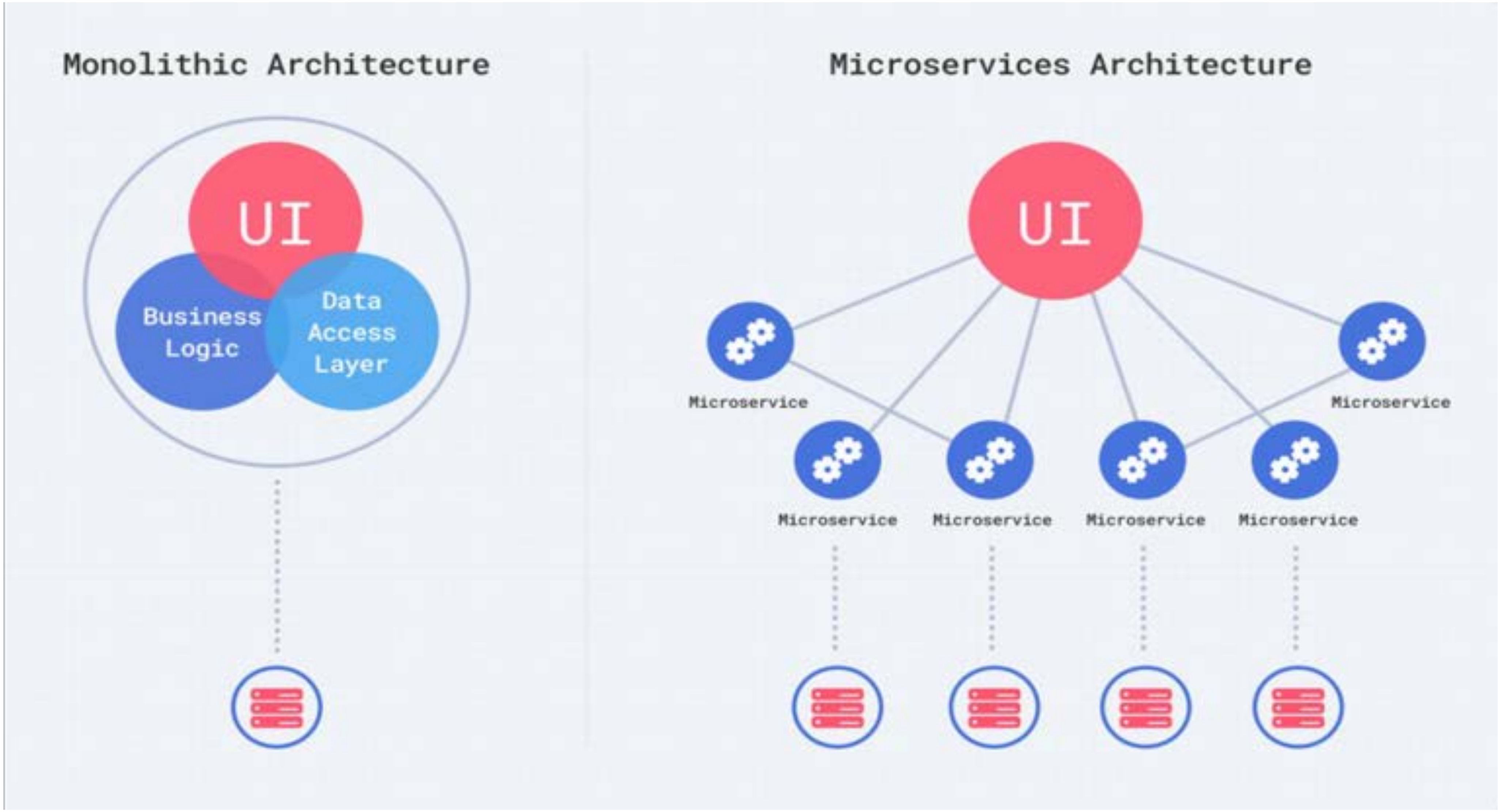
Chief Technology Evangelist

DevSecOps specialist and evangelist, lifelong learner/hacker and latent developer

- **VP of AppSec at Deutsche Bank**
  - 20k developers, 6k app
  - Fixed over 3 million high severity flaws
  - Built global AppSec program
- **Innovation manager/DevRel/Solution Architect at Veracode Inc.**
  - Frequent speaker and blogger
  - Advised Fortune 100 on DevSecOps implementations
  - Advisor to Product Management team
- **Independent DevSecOps consultant**
- **Industry analyst and advisor**

The need for API security

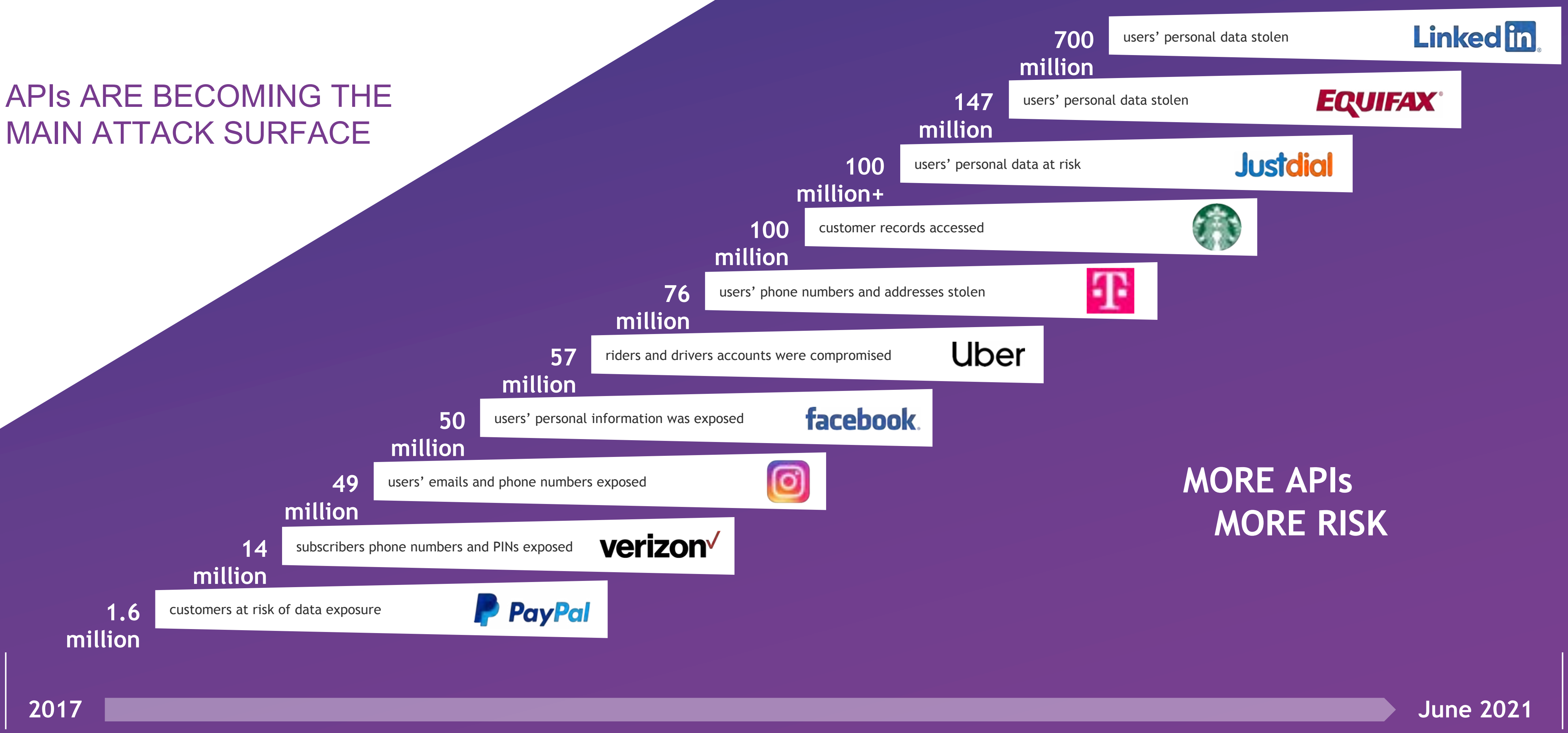
From the monolith to microservices



It’s all about the supply chain

Growing Number and Scale of Attacks

APIs ARE BECOMING THE MAIN ATTACK SURFACE



700 million users' personal data stolen

147 million users' personal data stolen

100 million+ users' personal data at risk

100 million customer records accessed

76 million users' phone numbers and addresses stolen

57 million riders and drivers accounts were compromised

50 million users' personal information was exposed

49 million users' emails and phone numbers exposed

14 million subscribers phone numbers and PINs exposed

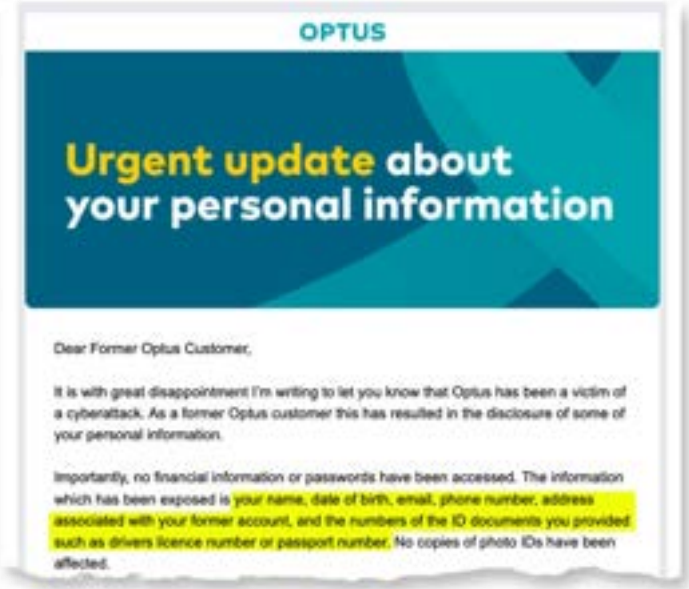
1.6 million customers at risk of data exposure

2017

June 2021

MORE APIs
MORE RISK

The age of the API mega-breach



OPTUS

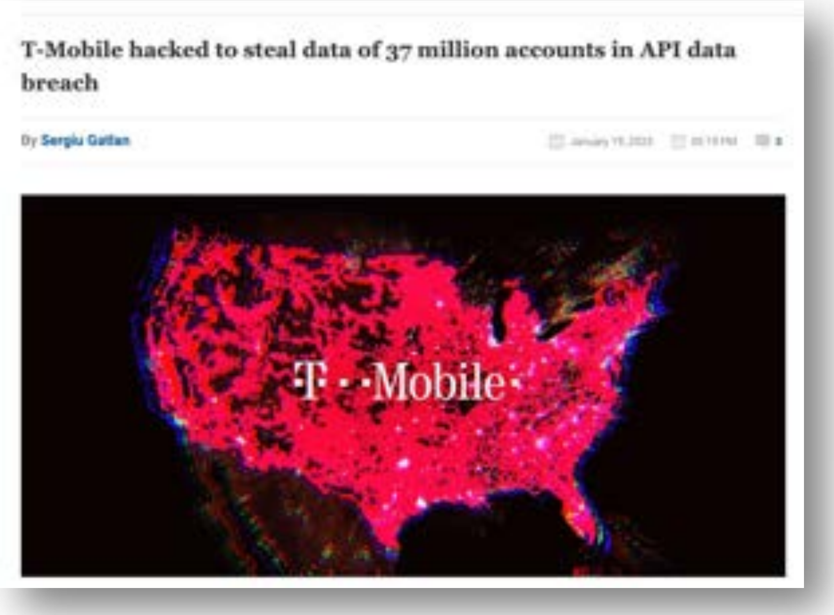
Urgent update about your personal information

Dear Former Optus Customer,

It is with great disappointment I'm writing to let you know that Optus has been a victim of a cyberattack. As a former Optus customer this has resulted in the disclosure of some of your personal information.

Importantly, no financial information or passwords have been accessed. The information which has been exposed is your name, date of birth, email, phone number, address associated with your former account, and the numbers of the ID documents you provided such as drivers licence number or passport number. No copies of photo IDs have been affected.

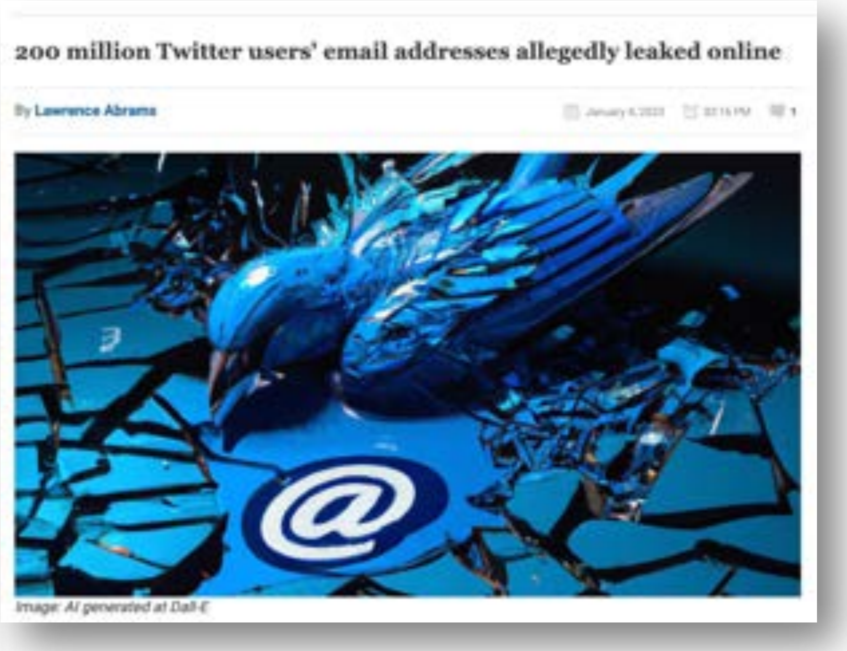
https://www.optus.com.au/about/media-centre/media-releases/2022/09/optus-notifies-customers-of-cyberattack



T-Mobile hacked to steal data of 37 million accounts in API data breach

By Sergiu Gatlan January 19, 2023 [illegible] 2

https://www.bleepingcomputer.com/news/security/t-mobile-hacked-to-steal-data-of-37-million-accounts-in-api-data-breach/



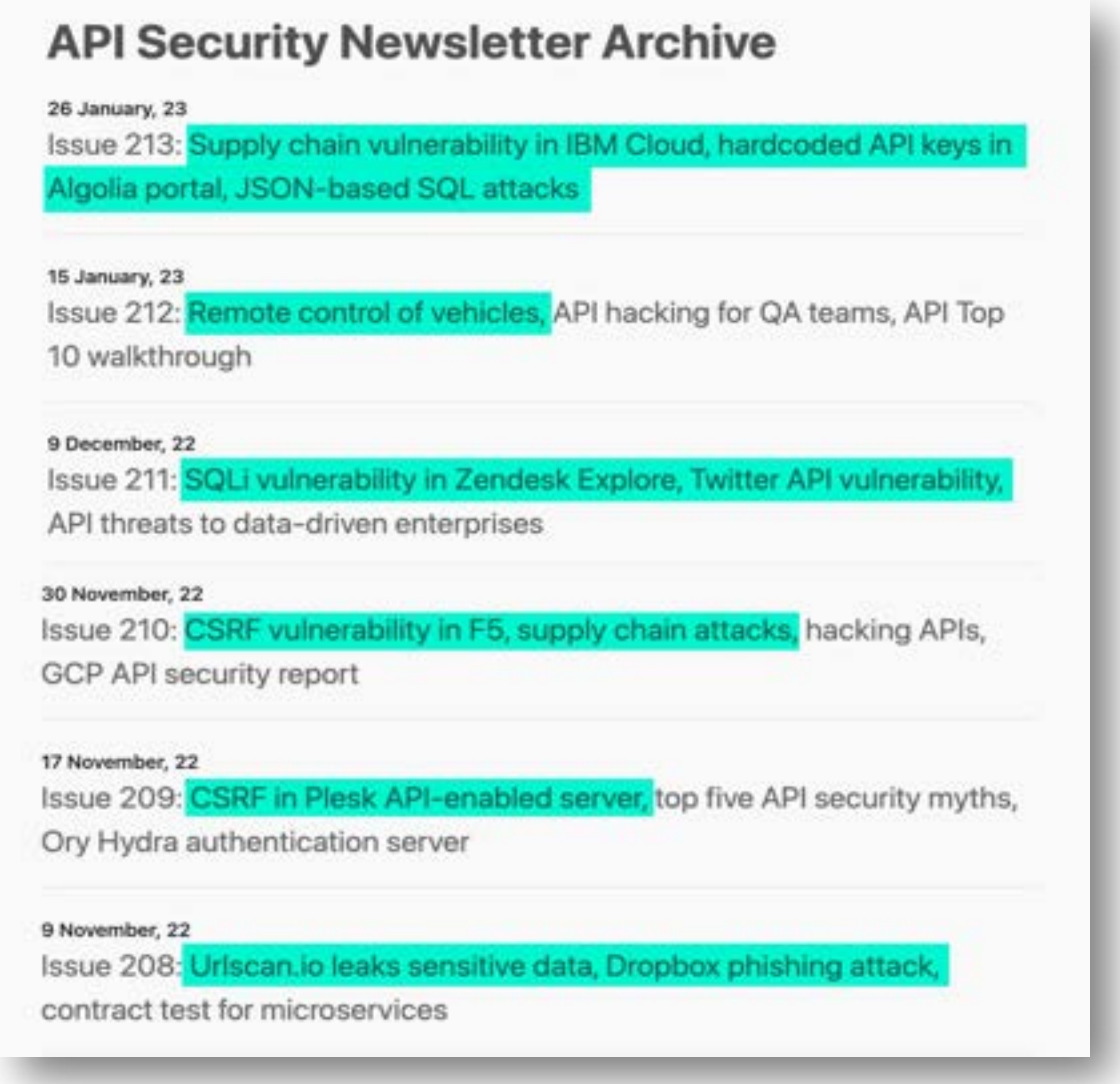
200 million Twitter users' email addresses allegedly leaked online

By Lawrence Abrams January 6, 2023 [illegible] 1

Image: AI generated at Dall-E

https://www.bleepingcomputer.com/news/security/200-million-twitter-users-email-addresses-allegedly-leaked-online/

Every week brings a new story



API Security Newsletter Archive

26 January, 23

Issue 213: Supply chain vulnerability in IBM Cloud, hardcoded API keys in Algolia portal, JSON-based SQL attacks

15 January, 23

Issue 212: Remote control of vehicles, API hacking for QA teams, API Top 10 walkthrough

9 December, 22

Issue 211: SQLi vulnerability in Zendesk Explore, Twitter API vulnerability, API threats to data-driven enterprises

30 November, 22

Issue 210: CSRF vulnerability in F5, supply chain attacks, hacking APIs, GCP API security report

17 November, 22

Issue 209: CSRF in Plesk API-enabled server, top five API security myths, Ory Hydra authentication server

9 November, 22

Issue 208: Urlscan.io leaks sensitive data, Dropbox phishing attack, contract test for microservices

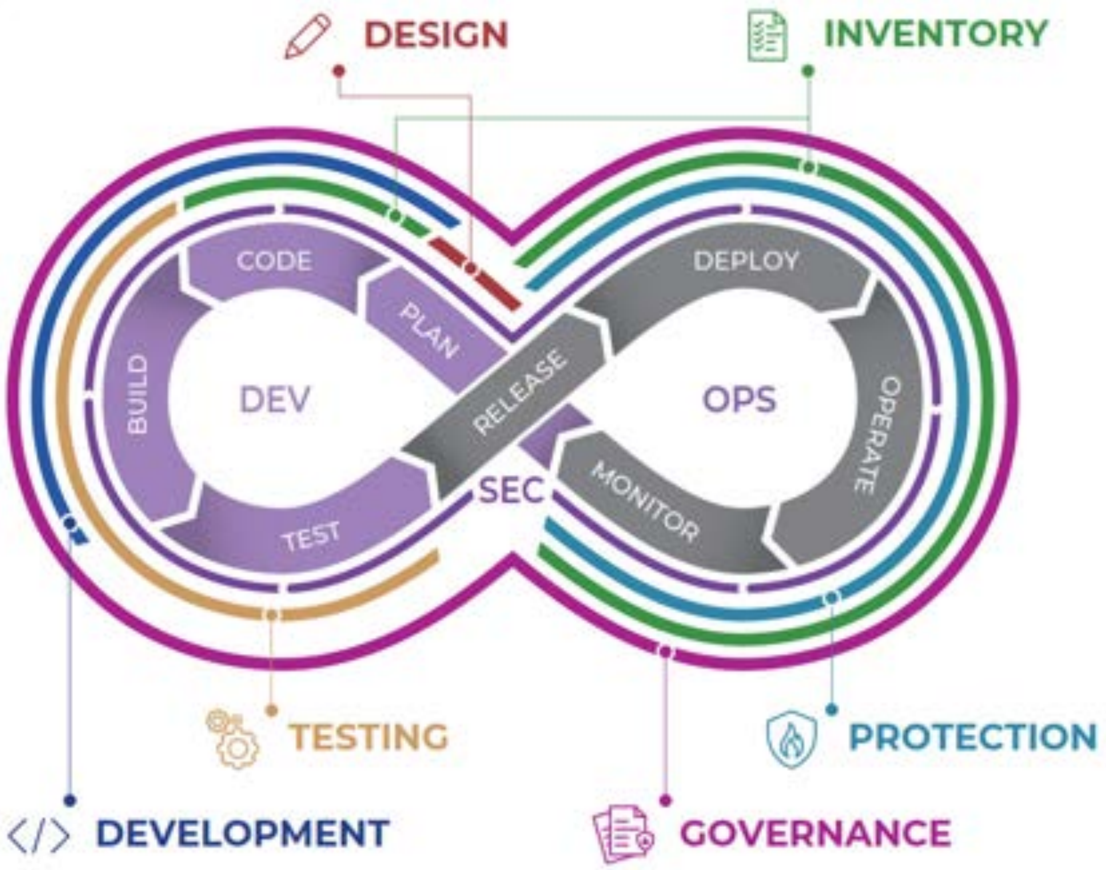
https://apisecurity.io/

The root cause of API vulnerabilities …

Human error is the root cause of API vulnerabilities !

- Logic errors
- Poor design
- Coding errors
- Misuse of components/libraries
- Misconfiguration of servers
- Shortcuts
- Assumptions
- Insecure defaults
- Misunderstanding attack vectors
- Vulnerable dependencies

The six pillars of API security



API INVENTORY

Do you understand what APIs you own? Do you track shadow and zombie APIs?

API DESIGN

Are you doing API-design-first? Do you incorporate security into the design phase?

API DEVELOPMENT

Are your developers trained to code securely? Do they understand API security threats and risks?

API TESTING

Are you doing automated API testing? Are you considering security in your test strategy?

API PROTECTION

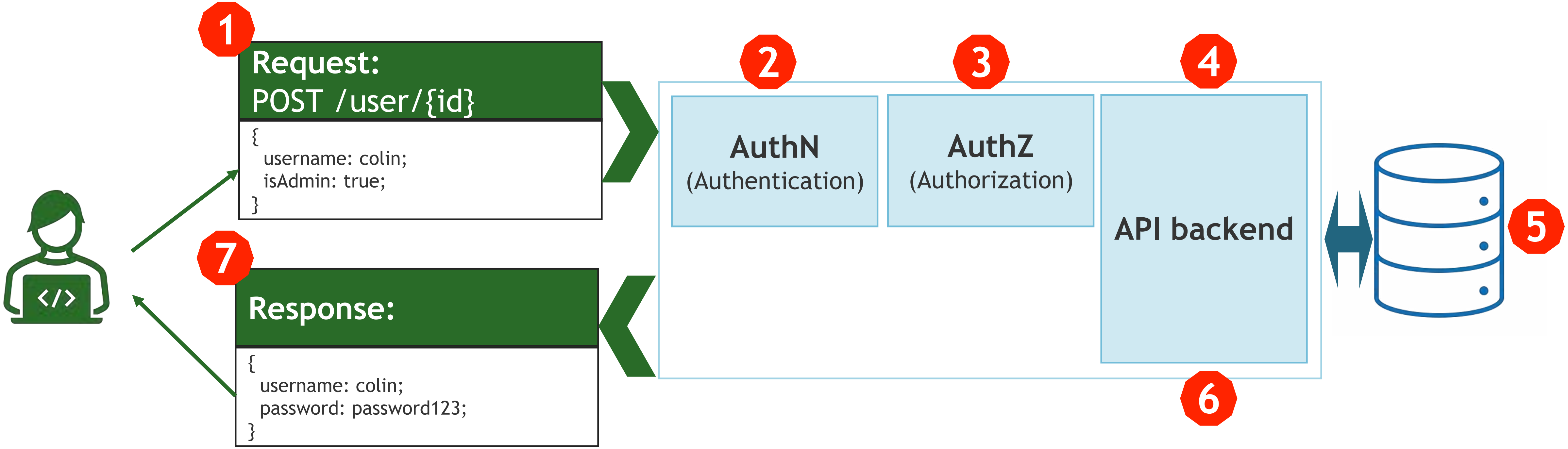
Are you using API protection technology (WAFs, WAAPs, API gateways) in your deployments?

API GOVERNANCE

Do you control and actively monitor your API estate and environments?

OWASP API Security Top 10

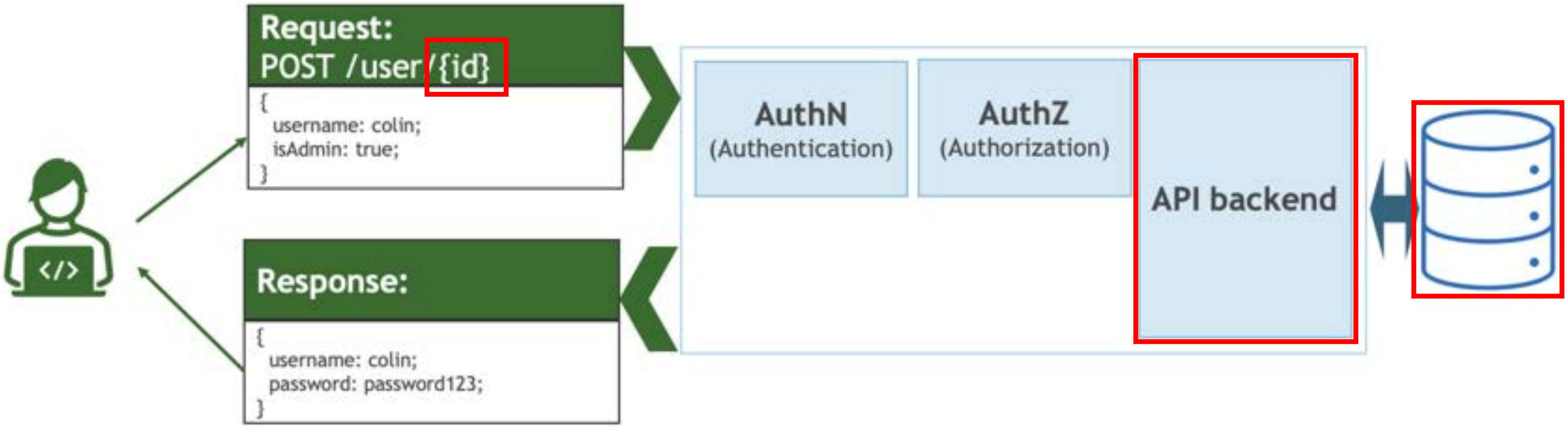
How does an API work?



```
{
  username: colin;
  isAdmin: true;
}
```

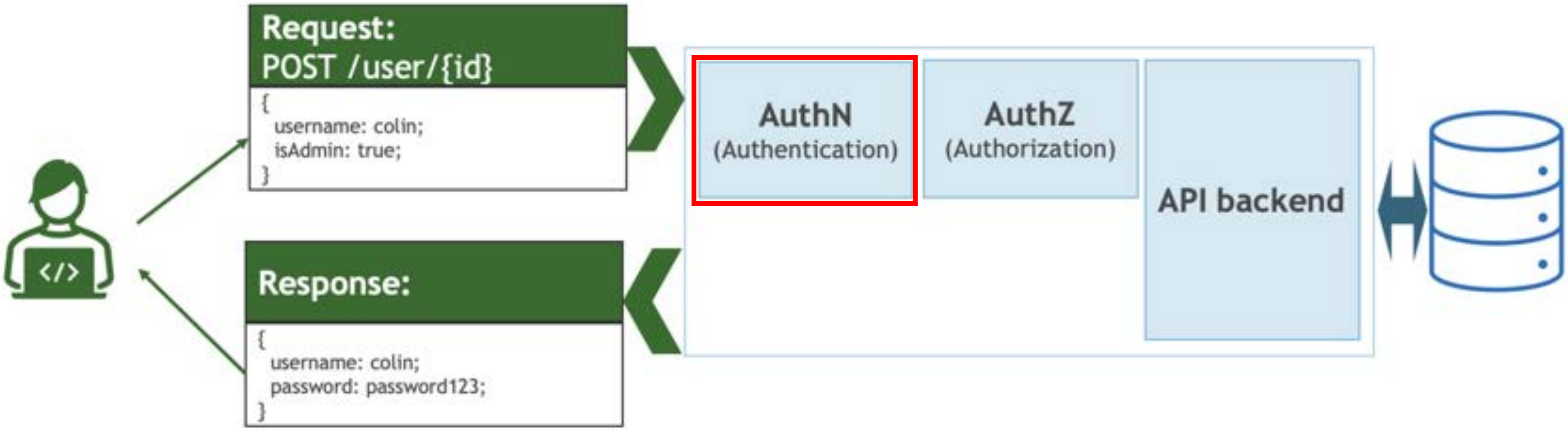
```
{
  username: colin;
  password: password123;
}
```

API1 — BROKEN OBJECT LEVEL AUTHORIZATION



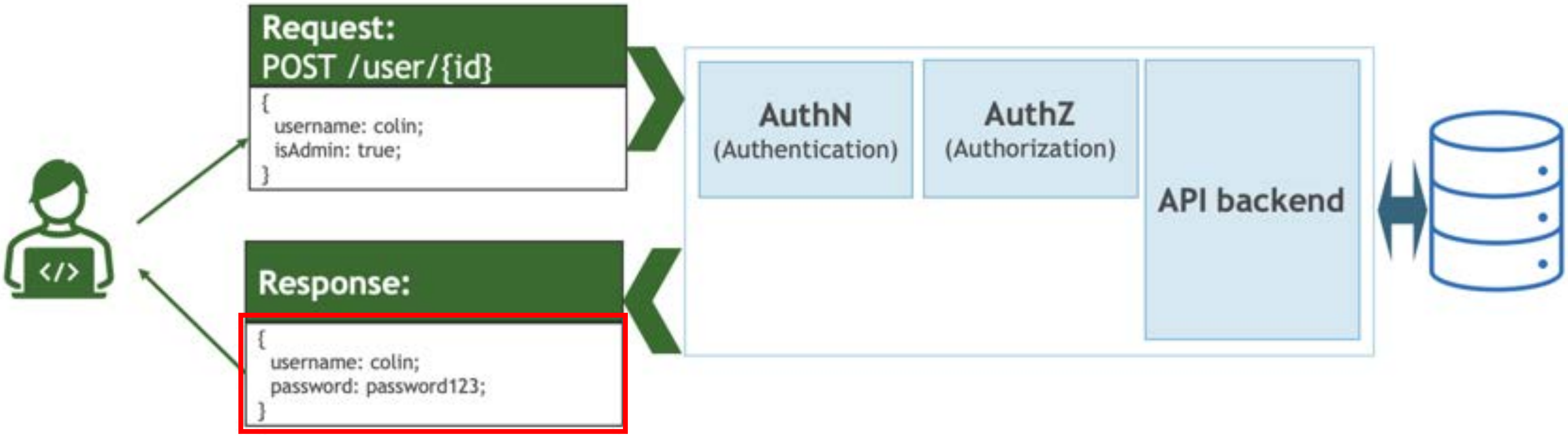
https://apisecurity.io/encyclopedia/content/owasp/api1-broken-object-level-authorization

API 2: BROKEN AUTHENTICATION



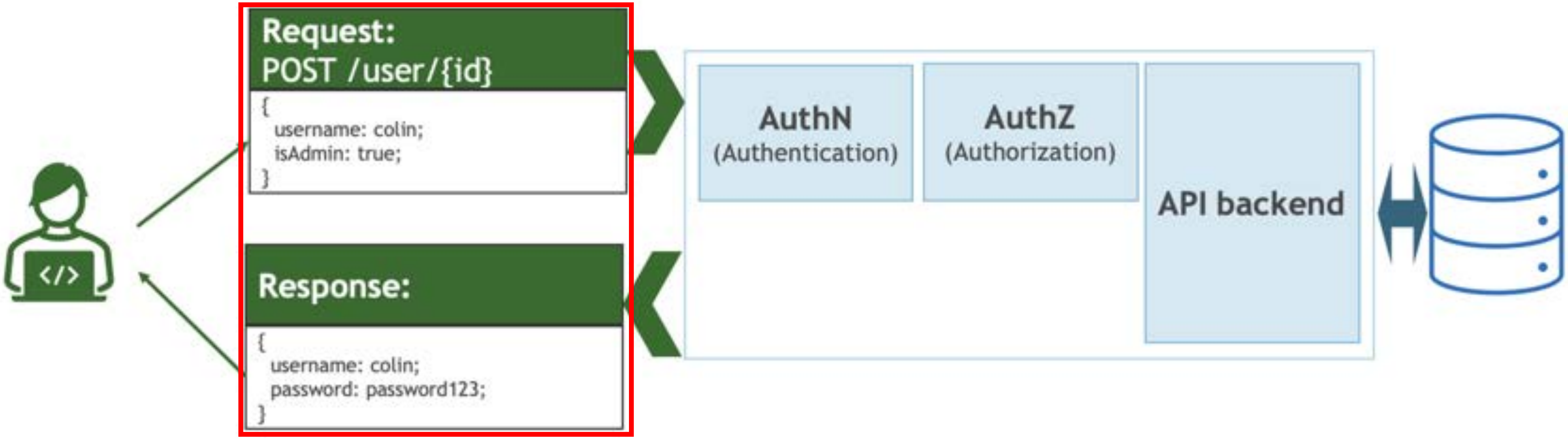
https://apisecurity.io/encyclopedia/content/owasp/api2-broken-authentication

API 3: DATA/EXCEPTION LEAKAGE



https://apisecurity.io/encyclopedia/content/owasp/api3-excessive-data-exposure

API 4: RESOURCES PROTECTION/RATE LIMITING



Request:
POST /user/{id}

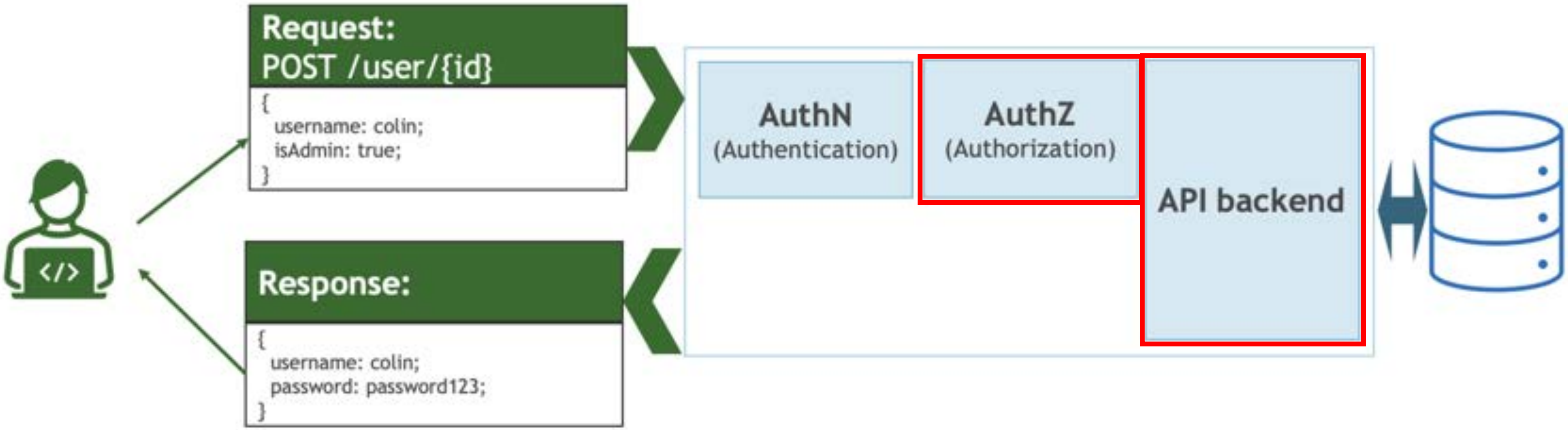
```
{
 username: colin;
 isAdmin: true;
}
```

Response:

```
{
 username: colin;
 password: password123;
}
```

https://apisecurity.io/encyclopedia/content/owasp/api4-lack-of-resources-and-rate-limiting

API 5: BROKEN FUNCTION LEVEL AUTH



Request:
POST /user/{id}

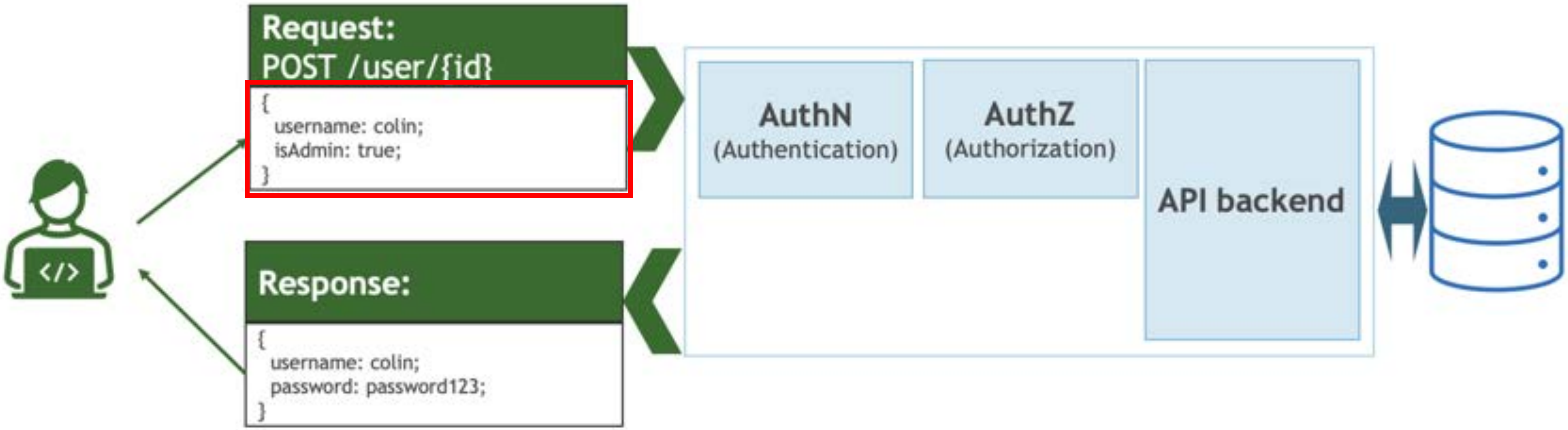
```
{
 username: colin;
 isAdmin: true;
}
```

Response:

```
{
 username: colin;
 password: password123;
}
```

https://apisecurity.io/encyclopedia/content/owasp/api5-broken-function-level-authorization

API 6: MASS ASSIGNMENT

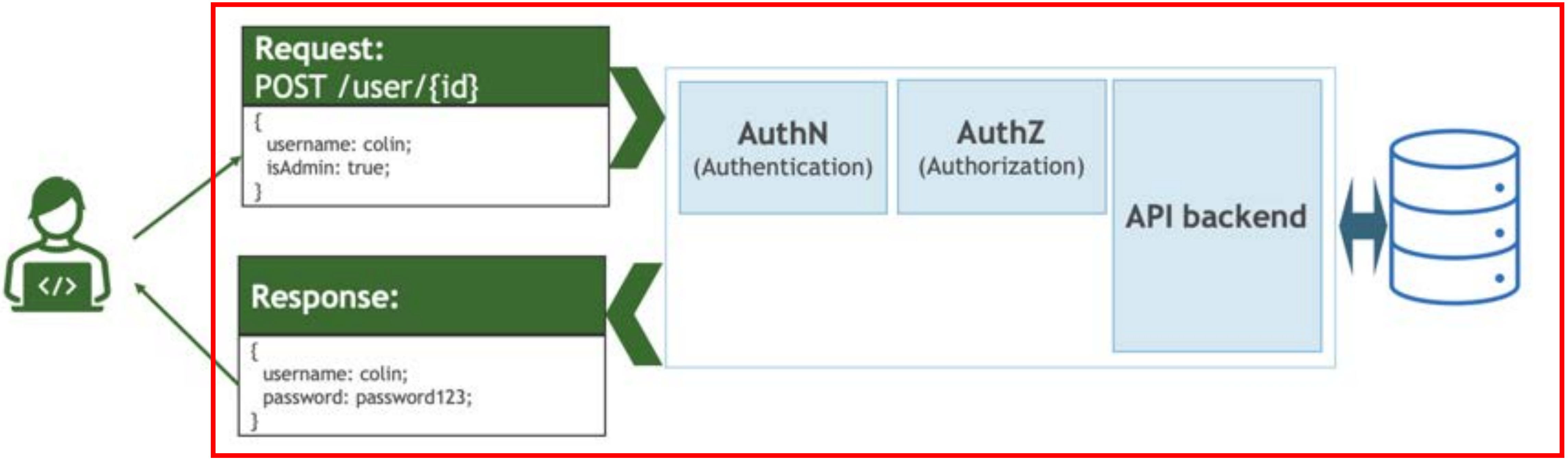


```
{
  username: colin;
  isAdmin: true;
}
```

```
{
  username: colin;
  password: password123;
}
```

https://apisecurity.io/encyclopedia/content/owasp/api6-mass-assignment

API 7: SECURITY MISCONFIGURATION



Request:
POST /user/{id}

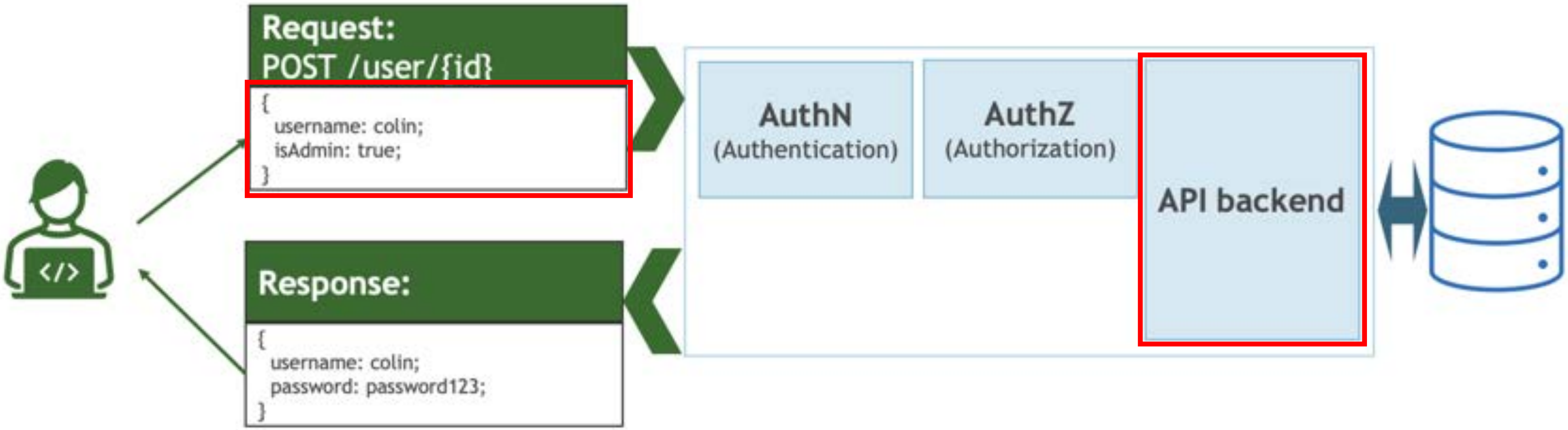
```
{
  username: colin;
  isAdmin: true;
}
```

Response:

```
{
  username: colin;
  password: password123;
}
```

https://apisecurity.io/encyclopedia/content/owasp/api7-security-misconfiguration

API 8: INJECTION



Request:
POST /user/{id}

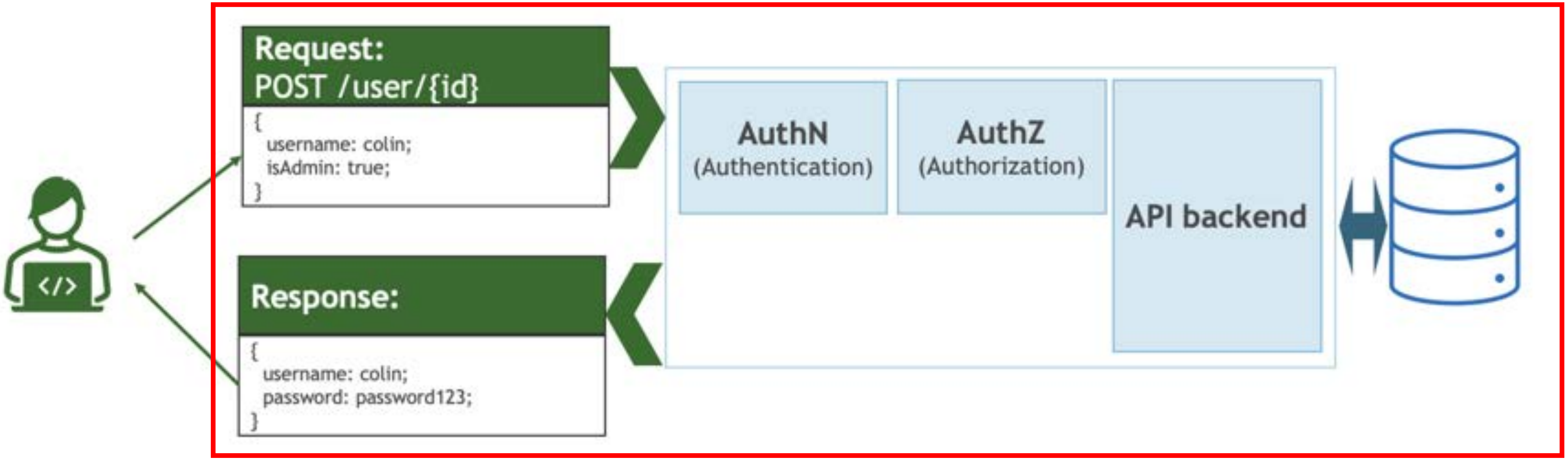
```
{
  username: colin;
  isAdmin: true;
}
```

Response:

```
{
  username: colin;
  password: password123;
}
```

https://apisecurity.io/encyclopedia/content/owasp/api8-injection

API 9: IMPROPER ASSETS MANAGEMENT



Request:
POST /user/{id}

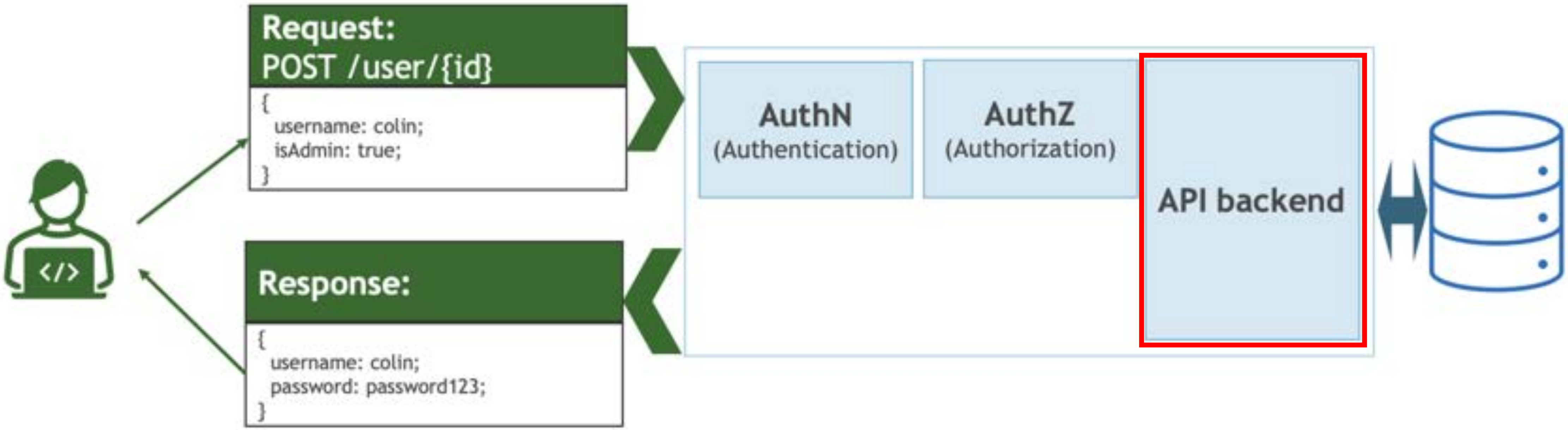
```
{
 username: colin;
 isAdmin: true;
}
```

Response:

```
{
 username: colin;
 password: password123;
}
```

https://apisecurity.io/encyclopedia/content/owasp/api9-improper-assets-management

API 10: LOGGING /MONITORING



```
{
 username: colin;
 isAdmin: true;
}
```

```
{
 username: colin;
 password: password123;
}
```

https://apisecurity.io/encyclopedia/content/owasp/api10-insufficient-logging-and-monitoring

Vulnerabilities in review

Recap of our favorite vulnerabilities in 2022

#1: Global shipping company

What happened?

Researchers discovered they could automatically submit parcel numbers to an API that retrieved a map image. They then used this image to guess the postcode and then were able to retrieve full parcel information and extended user information.

Impact:

Potentially large-scale exfiltration of customer PII and parcel tracking information. Researchers reported responsibly and a fix was released before exploitation.

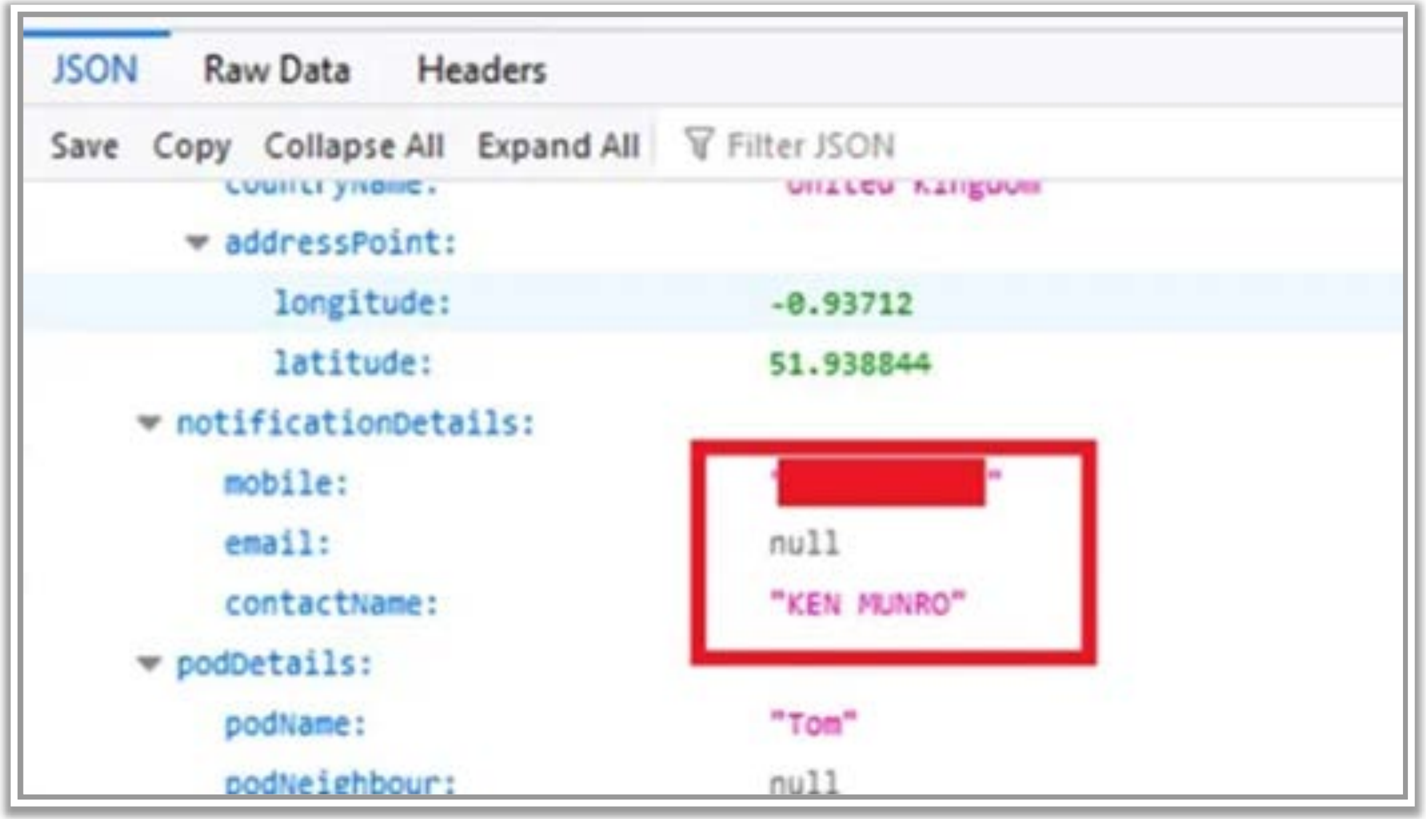
Cause:

- Lack of rate-limiting
- Excessive information exposure

Lessons learned:

- Protect APIs from brute-force attacks.
- Only return the minimum information necessary.

#1: Global shipping company



```
countryName:                     "United Kingdom"
  ▾ addressPoint:
      longitude:                 -0.93712
      latitude:                  51.938844
▾ notificationDetails:
    mobile:                      "[redacted]"
    email:                       null
    contactName:                 "KEN MUNRO"
▾ podDetails:
    podName:                     "Tom"
    podNeighbour:                null
```

#1: Global shipping company - how to prevent it

Design:

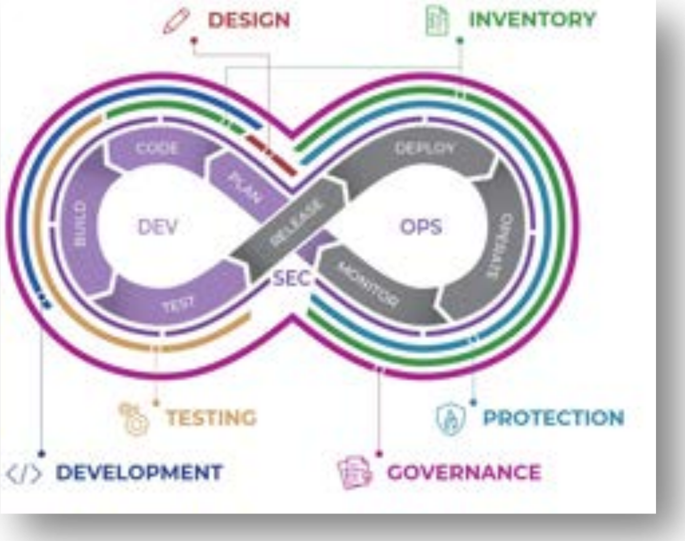
- Failure to under PII requirements – leaked customer details
- Failed to understand abuse case – brute forcing of tracking codes
- Relied on obscurity – map image could be reverse engineered

Testing:

- Did not detect leakage of customer information

Protection:

- Lack of rate limiting on critical API allowing brute forcing



#2: Campus access control

What happened?

A campus access control application used a backend API that did not authenticate users allowing an attacker to impersonate any user given their guessable IDs. By faking the user location an attacker could access all doors on campus.

Impact:

Unknown, but probably limited.

Cause:

- Broken function-level authorization
- Broken authentication

Lessons learned:

- Ensure all functions are fully authenticated.
- Make sure you can revoke any sessions keys or tokens.

#2: Campus access control – how to prevent it

Design:

- Allowed user to fake location details
- Relied on public domain user IDs which could be guessed

Development:

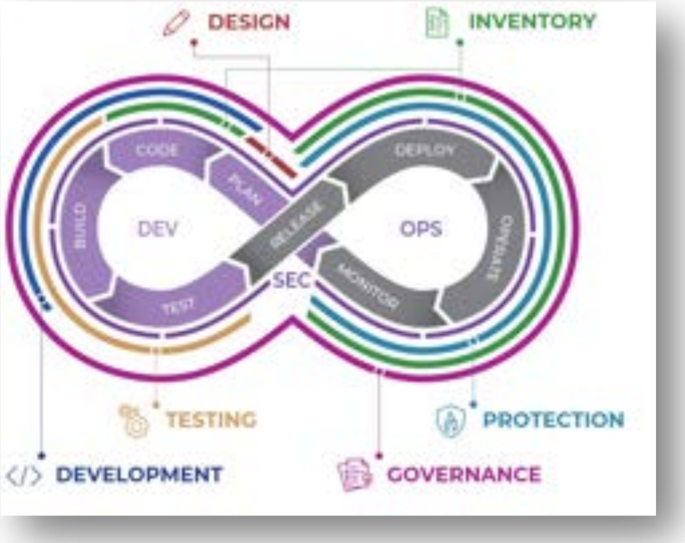
- Failed to prevent BOLA – use better authorization

Testing:

- Did not detect broken authentication – easy to detect!

Governance:

- No security process in place to handle security disclosure



#3: Microbrewery application

What happened?

Mobile application for microbrewery used hardcoded tokens within application binary which could easily be extracted allowing for manipulation of backend functions including other users PII, and access to discount schemes, etc.

Impact:

Free beer !! Disclosure of user’s PII.

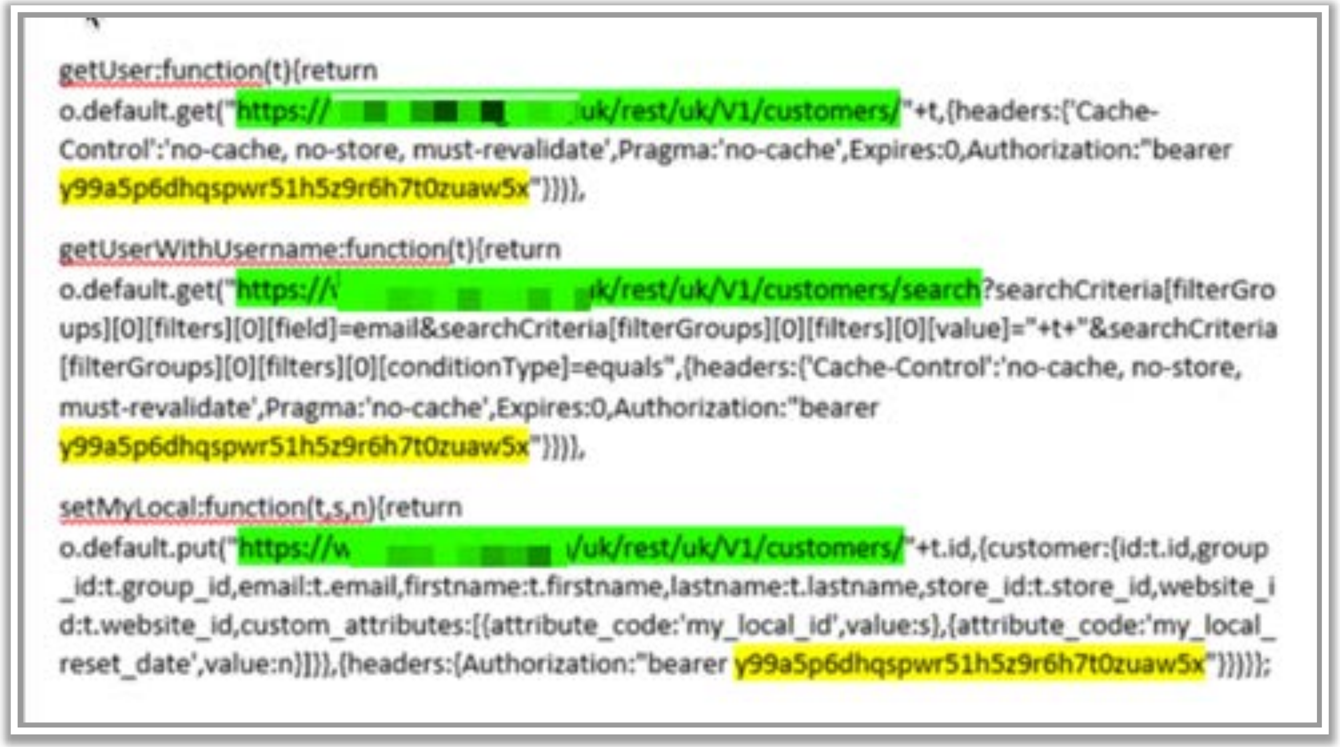
Cause:

- Hardcoded tokens in mobile application

Lessons learned:

- Use a standard mechanism (OAuth2) for the exchange and distribution of tokens.
- Make sure you are able to revoke any sessions keys or tokens.

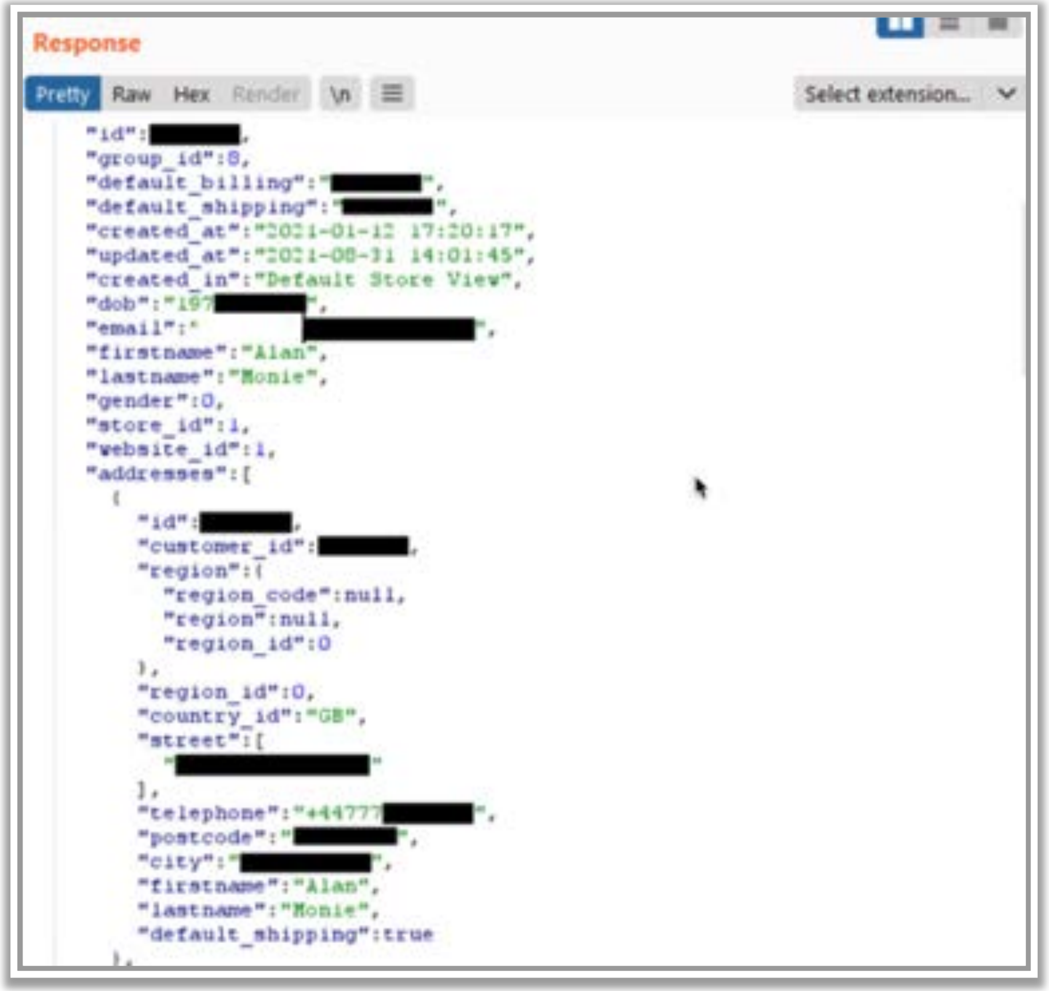
#3: Microbrewery application



```
getUser:function(t){return
o.default.get("https://[redacted]uk/rest/uk/V1/customers/"+t,{headers:{'Cache-
Control':'no-cache, no-store, must-revalidate',Pragma:'no-cache',Expires:0,Authorization:"bearer
y99a5p6dhqspwr51h5z9r6h7t0zuaw5x"}})},

getUserWithUsername:function(t){return
o.default.get("https://\[redacted]ik/rest/uk/V1/customers/search?searchCriteria[filterGro
ups][0][filters][0][field]=email&searchCriteria[filterGroups][0][filters][0][value]="+t+"&searchCriteria
[filterGroups][0][filters][0][conditionType]=equals",{headers:{'Cache-Control':'no-cache, no-store,
must-revalidate',Pragma:'no-cache',Expires:0,Authorization:"bearer
y99a5p6dhqspwr51h5z9r6h7t0zuaw5x"}})},

setMyLocal:function(t,s,n){return
o.default.put("https://w[redacted]/uk/rest/uk/V1/customers/"+t.id,{customer:{id:t.id,group
_id:t.group_id,email:t.email,firstname:t.firstname,lastname:t.lastname,store_id:t.store_id,website_i
d:t.website_id,custom_attributes:[{attribute_code:'my_local_id',value:s},{attribute_code:'my_local_
reset_date',value:n}]}},{headers:{Authorization:"bearer y99a5p6dhqspwr51h5z9r6h7t0zuaw5x"}})}};
```



```
"id":[redacted],
"group_id":0,
"default_billing":"[redacted]",
"default_shipping":"[redacted]",
"created_at":"2021-01-12 17:20:17",
"updated_at":"2021-08-31 14:01:45",
"created_in":"Default Store View",
"dob":"197[redacted]",
"email":"[redacted]",
"firstname":"Alan",
"lastname":"Monie",
"gender":0,
"store_id":1,
"website_id":1,
"addresses":[
  {
    "id":[redacted],
    "customer_id":[redacted],
    "region":{
      "region_code":null,
      "region":null,
      "region_id":0
    },
    "region_id":0,
    "country_id":"GB",
    "street":[
      "[redacted]"
    ],
    "telephome":"+44777[redacted]",
    "postcode":"[redacted]",
    "city":"[redacted]",
    "firstname":"Alan",
    "lastname":"Monie",
    "default_shipping":true
  },
```

#3: Microbrewery application - how to prevent it

Design:

- No attempt to use standard authorization framework
- Failure to under PII requirements – leaked customer details

Testing:

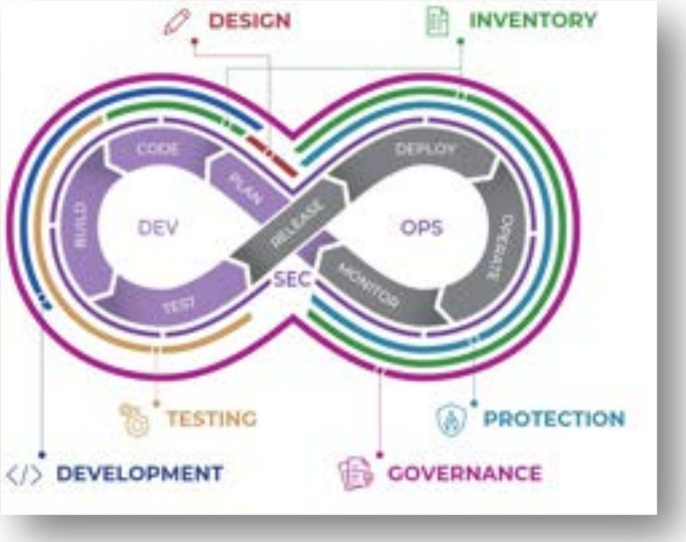
- Hardcoded tokens can be detected trivially – did they even try?

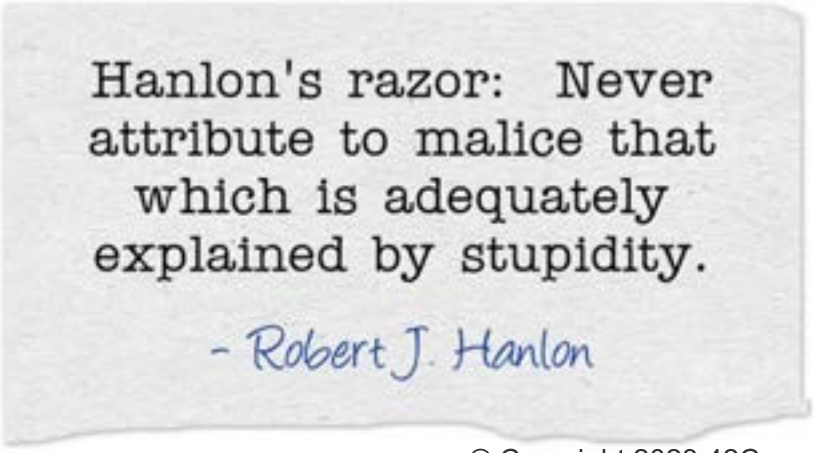
Protection:

- No attempt to monitor abuse – did users actually abuse the vulnerability?

Governance:

- No security process in place to handle security disclosure
- Excessively long time to remediate




Hanlon's razor: Never attribute to malice that which is adequately explained by stupidity.

- Robert J. Hanlon

#4: Cryptocurrency portal

What happened?

A researcher discovered an issue in a cryptocurrency trading platform whereby he could trade between two different accounts. The platforms failed to validate the account details and allowed purchases from accounts with insufficient funds. The exploit could be triggered by manipulating API request parameters.

Impact:

Limited due to responsible disclosure and immediate response.

Cause:

- A text-book case of broken-object level authorization allowing manipulation via an API parameter.

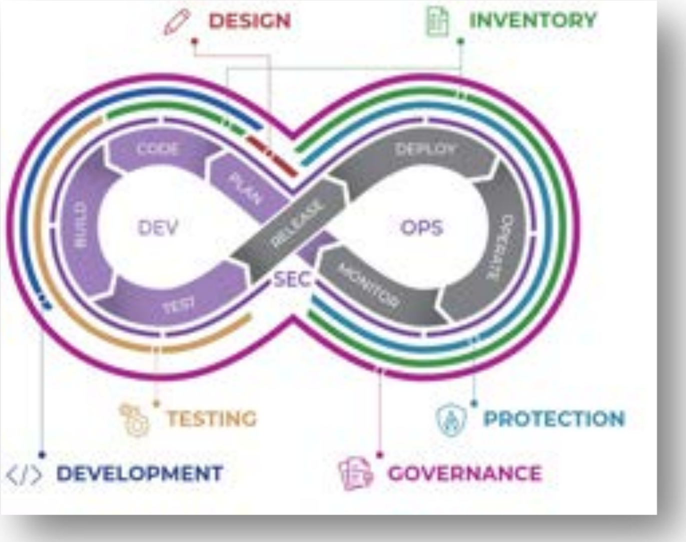
Lessons learned:

- Broken object-level authorization is the number one API security issue — always ensure you fully validate access to objects for all requests.
- Bug bounties can be profitable — this was worth $250,000.

#4: Cryptocurrency portal – how to prevent it

Development:

- Threat modelling could have identified attack vector
- Security training could have raised awareness of BOLA
- Code reviews could have identified issue



#5: Dating application

What happened?

A researcher discovered he could use trilateration techniques to determine the precise location of users. It was also possible to access the PII information of connected users.

Impact:

Minimal although caused embarrassment for the application affected.

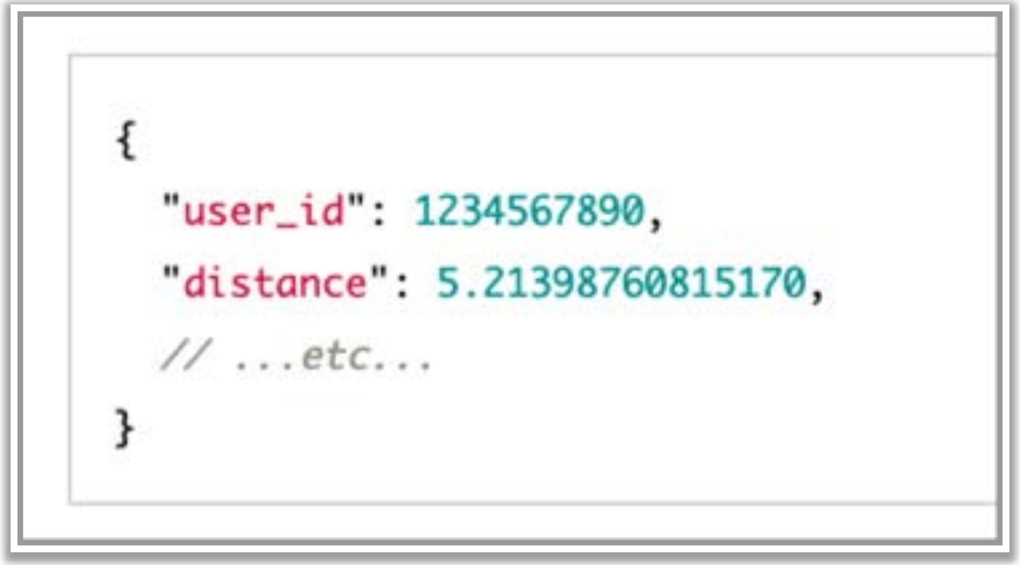
Cause:

- Another example of broken object-level authorization
- Excessive information exposure allowed inference of user location to a high level of precision.
- Security by obscurity

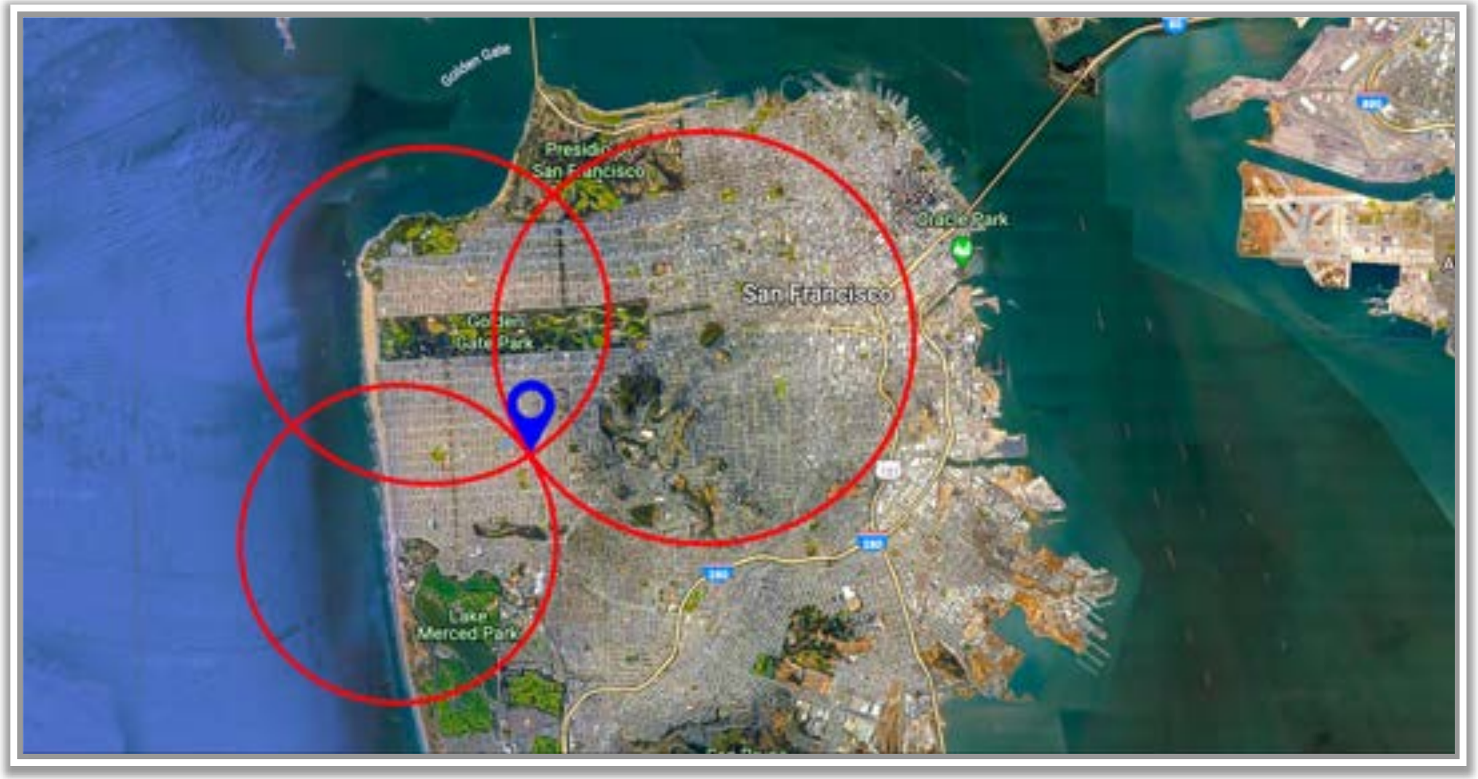
Lessons learned:

- Only disclose the minimum of information necessary via API calls as attackers may infer other useful user data.
- Never rely on client-side protections to protect your user data since this can easily be circumvented by using the API directly.

#5: Dating application



```
{
  "user_id": 1234567890,
  "distance": 5.21398760815170,
  // ...etc...
}
```



#5: Dating application - how to prevent it

Design:

- Threat modelling would have revealed issues with geolocation usage

Development:

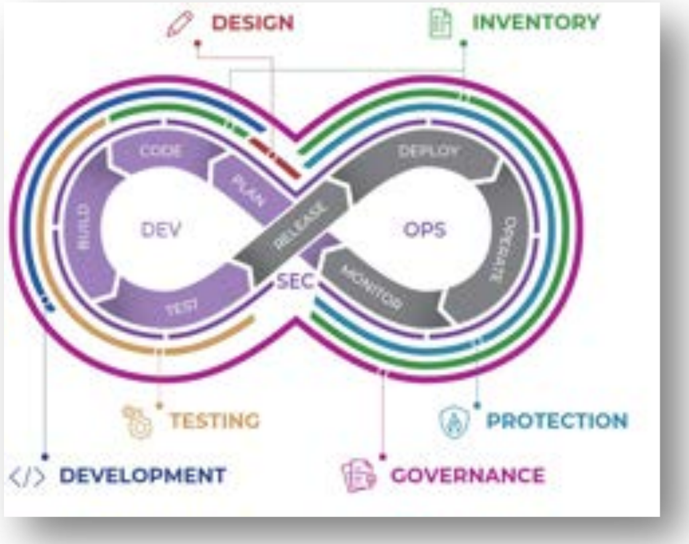
- Reliance of client-side protections
- Failed to protect against BOLA

Testing:

- Excessive information disclosure can easily be tested

Protection:

- Excessive information disclosure can easily be prevented



*“As one of the trailblazers of location-based online dating, T****r was inevitably also one of the trailblazers of location-based security vulnerabilities.”*

#6: All in One SEO WordPress plugin

What happened?

A popular WordPress plugin had a broken API endpoint which allowed any authenticated users to have full access (effectively admin access) to affected sites.

Impact:

Full takeover of affected WordPress sites, immediate patch released.

Cause:

- Broken function-level authorization and poor default settings in API endpoint handler.

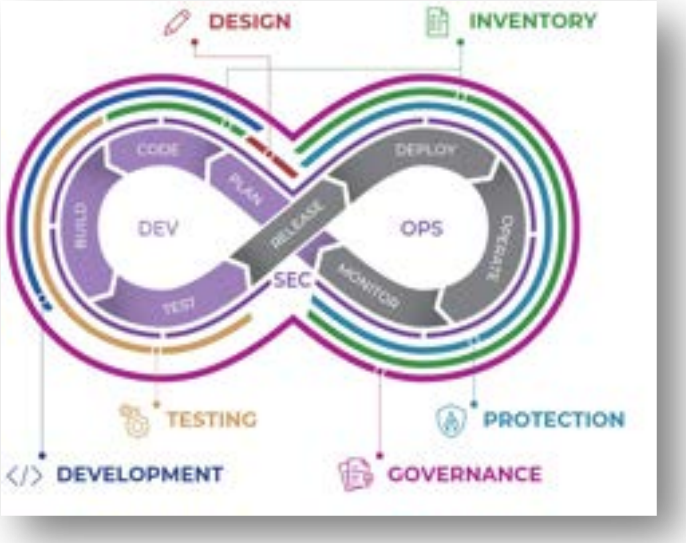
Lessons learned:

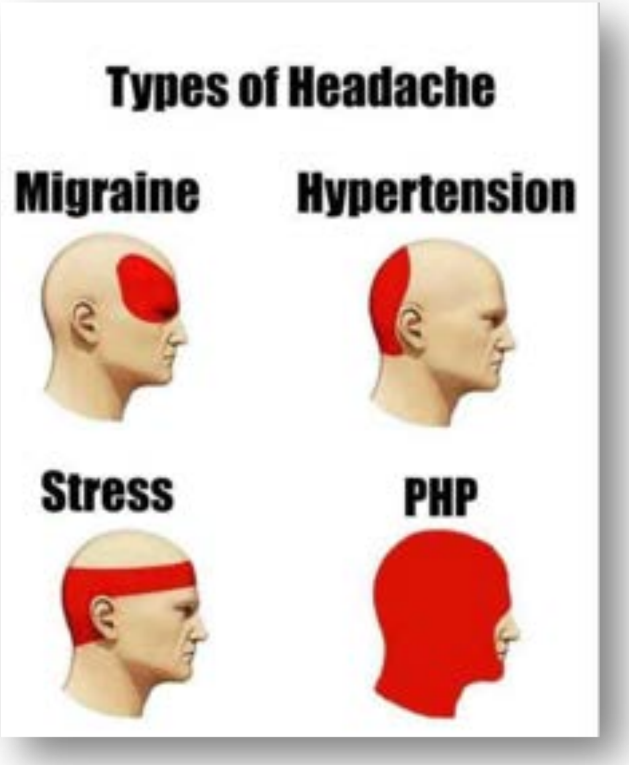
- Ensure that all functions and endpoints are fully authenticated and authorized.
- Defensive coding techniques should be used to prevent access in the case of failure.

#6: All in One SEO WordPress plugin – how to prevent it

Development:

- Poorly implemented authorization handler
  - insecure default used
  - case-sensitivity issues
- Lack of code review of critical code paths





#7: Twitter email/phone number mass leakage

What happened?

Hackers were able to abuse a Twitter API designed for mobile device login flows in order to confirm whether emails or phone numbers were associated with a Twitter account.

Impact:

Very high volumes (estimated at 200 million) of verified Twitter accounts and their associated email and phone numbers were available for sale on the black market. Such databases would aid spear-phishing attacks.

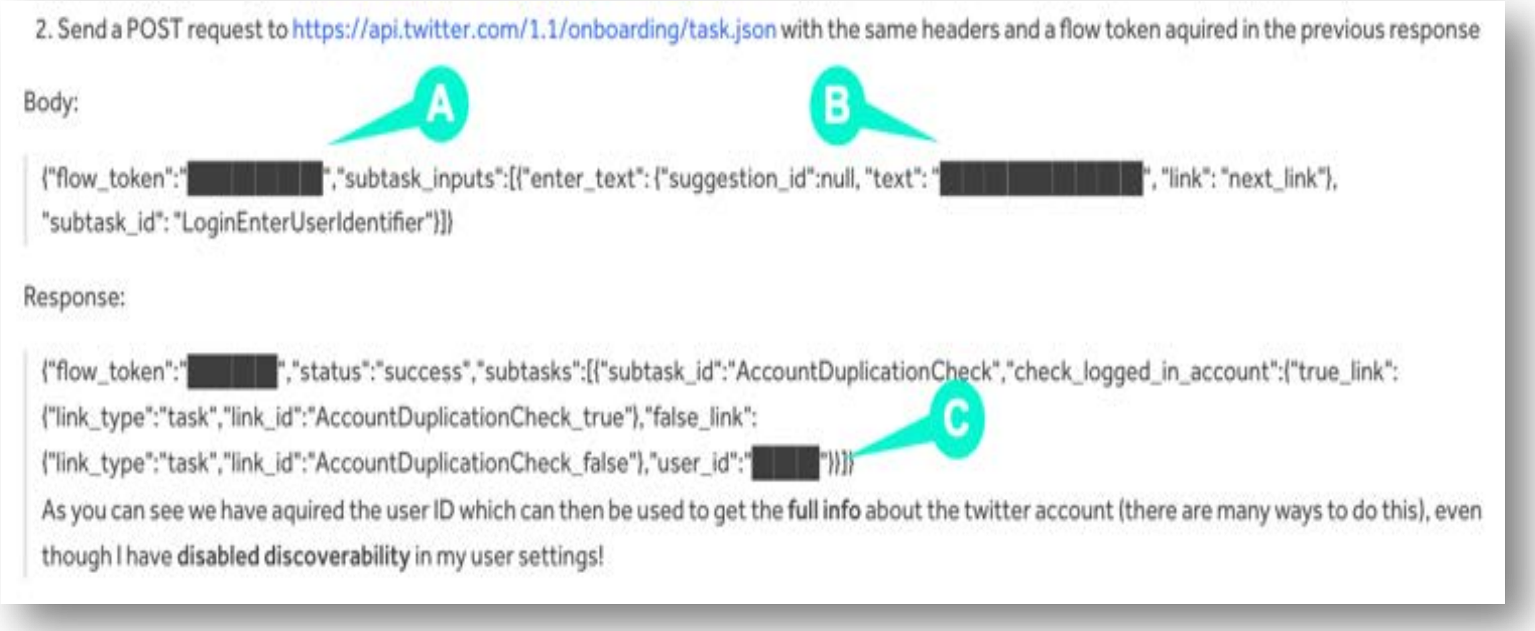
Cause:

- Abuse of a seemingly benign utility API call leaked Twitter IDs for valid accounts.
- Apparent lack of either rate-limiting or intrusion detection on the associated API.

Lessons learned:

- Always consider the abuse case via threat modelling.
- Use preventative controls to detect abuse of APIs.

#7: Twitter email/phone number mass leakage



2. Send a POST request to https://api.twitter.com/1.1/onboarding/task.json with the same headers and a flow token aquired in the previous response

Body:

```
{"flow_token":"[redacted]","subtask_inputs":[{"enter_text":{"suggestion_id":null, "text": "[redacted]", "link": "next_link"},
"subtask_id": "LoginEnterUserIdentifier"}]}
```

Response:

```
{"flow_token":"[redacted]","status":"success","subtasks":[{"subtask_id":"AccountDuplicationCheck","check_logged_in_account":{"true_link":
{"link_type":"task","link_id":"AccountDuplicationCheck_true"},"false_link":
{"link_type":"task","link_id":"AccountDuplicationCheck_false"},"user_id":"[redacted]"}}]}
```

As you can see we have aquired the user ID which can then be used to get the **full info** about the twitter account (there are many ways to do this), even though I have **disabled discoverability** in my user settings!

https://hackerone.com/reports/1439026

Twitter Leak Shows How Important API Security Remains in 2023

It is not enough to have an API tool that can simply discover the APIs for each application and then implement risk-policy firewalls.

Jan 13th, 2023 8:29am by Charlotte Freeman

https://thenewstack.io/twitter-leak-shows-how-important-api-security-remains-in-2023/

Twitter data breach shows APIs are a goldmine for PII and social engineering

Tim Keary
@tim_keary

January 6, 2023 7:39 AM

https://venturebeat.com/security/twitter-social-engineering/

#7: Twitter email/phone number mass leakage - how to prevent it

Design:

- Threat modelling would have identified this abuse case
- Twitter user IDs should not have been exposed

Development:

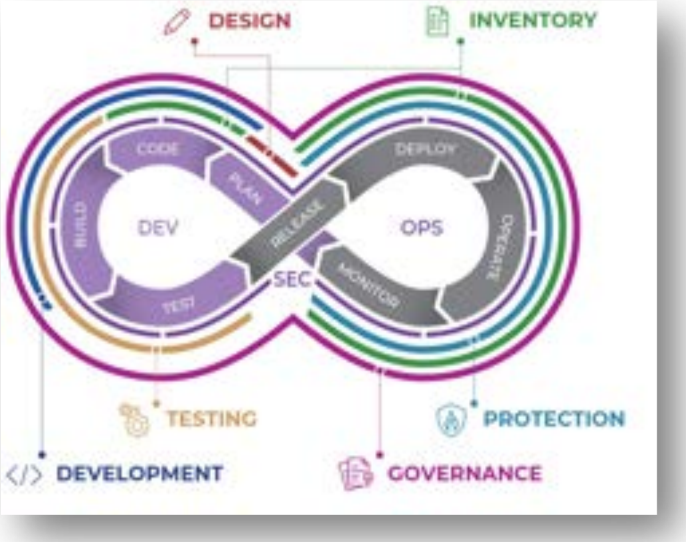
- Development testing could have identified the user ID exposure

Protection:

- Intrusion detection could have detected unusual activity on this API

Governance:

- Failure to control PII information



#8: Load balancer

What happened?

A load balancing and security suite were affected by a Remote Code Execution (RCE) vulnerability. The vulnerability is in the REST API that allowed remote access to platform configuration. Attackers could gain access to an exposed command shell endpoint that did not require any authentication.

Impact:

16,000 systems were exposed on the internet. No reported breaches occurred, and the issue has been patched.

Cause:

- Broken authentication

Lessons learned:

- Ensure all API endpoints are authenticated.
- Reduce attack surface by removing unnecessary management interfaces or lock down their public access.

#8: Load balancer - how to prevent it

Design:

- Insecure defaults – disable diagnostic service by default

Development:

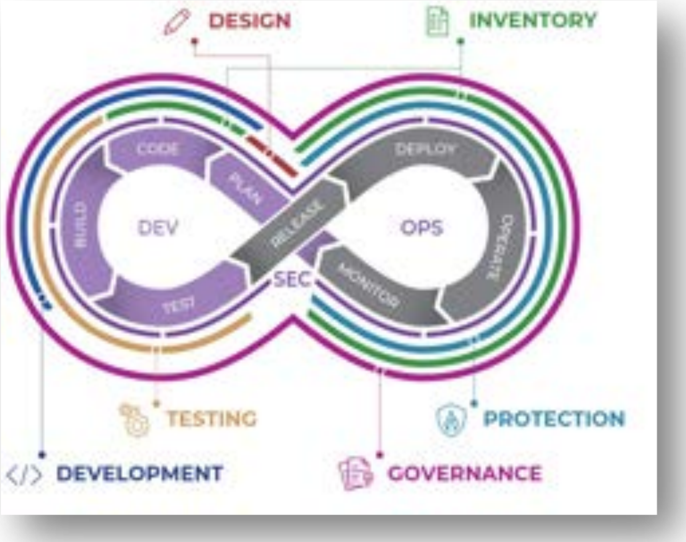
- Code review of critical code paths would detect issues like this

Testing:

- No evidence of testing – unauthenticated endpoints are easy to discover

Governance:

- Strong governance would ensure issues like this are not released to production



#9: Home router

What happened?

A popular home router was vulnerable to command injection vulnerability in an internal API. A security researcher discovered an internal admin interface that the router UI used to execute arbitrary commands and was able to execute arbitrary commands.

Impact:

No reported breaches were disclosed, and at the time of writing no patches were available.

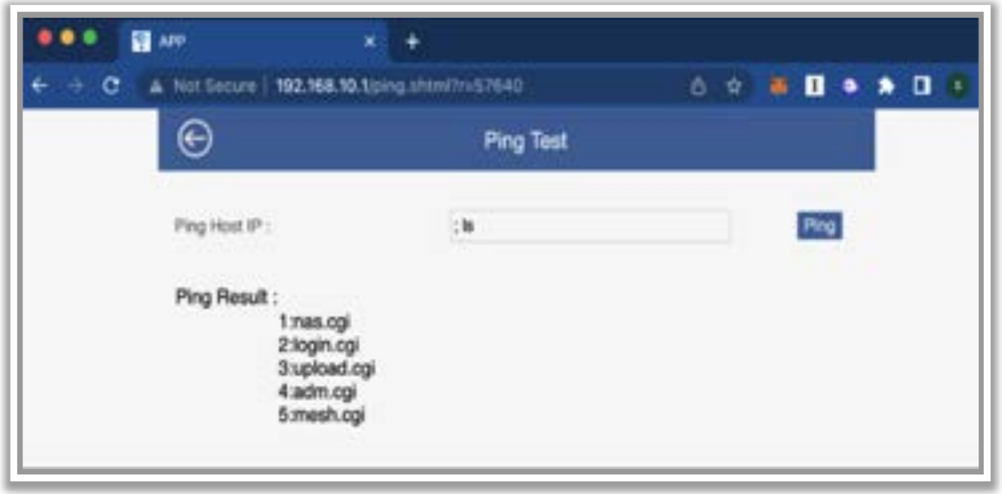
Cause:

- Broken authentication
- Cross-site request forgery

Lessons learned:

- Ensure all API endpoints are authenticated.
- Reduce attack surface by removing unnecessary management interfaces.
- Protect internet facing access with well trusted protections (such as CSRF tokens)

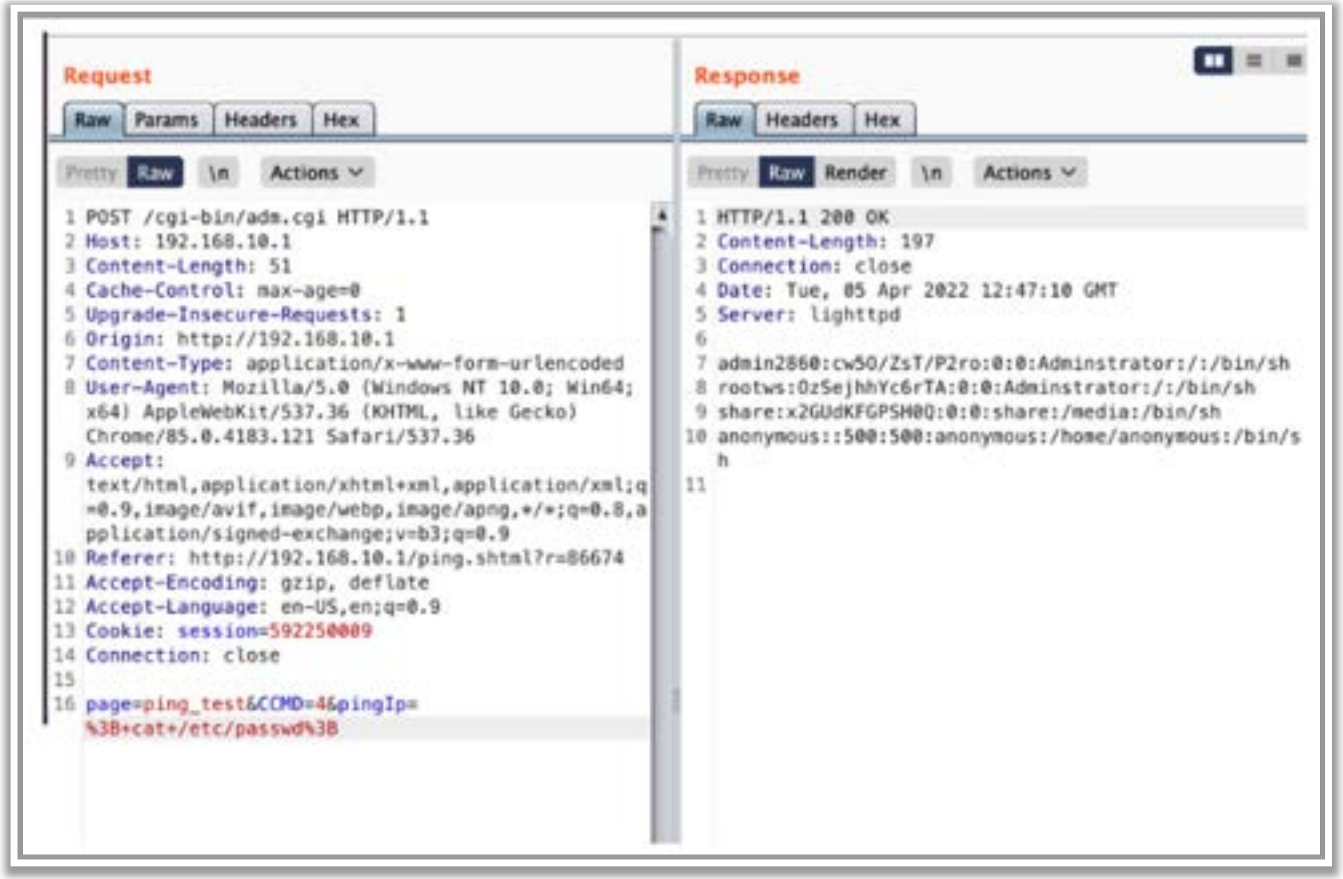
#9: Home router



Ping Host IP : ; ls

Ping Result :

```
1:nas.cgi
2:login.cgi
3:upload.cgi
4:adm.cgi
5:mesh.cgi
```



```
POST /cgi-bin/adm.cgi HTTP/1.1
Host: 192.168.10.1
Content-Length: 51
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.10.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/85.0.4183.121 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q
=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,a
pplication/signed-exchange;v=b3;q=0.9
Referer: http://192.168.10.1/ping.shtml?r=86674
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: session=592250009
Connection: close

page=ping_test&CCMD=4&pingIp=
%3B+cat+/etc/passwd%3B
```

```
HTTP/1.1 200 OK
Content-Length: 197
Connection: close
Date: Tue, 05 Apr 2022 12:47:10 GMT
Server: lighttpd

admin2860:cw5O/ZsT/P2ro:0:0:Adminstrator:/:/bin/sh
rootws:OzSejhhYc6rTA:0:0:Adminstrator:/:/bin/sh
share:x2GUdKFGPSH0Q:0:0:share:/media:/bin/sh
anonymous::500:500:anonymous:/home/anonymous:/bin/s
h

```

#9: Home router - how to prevent it

Design:

- Insecure defaults – disable debug interface by default

Development:

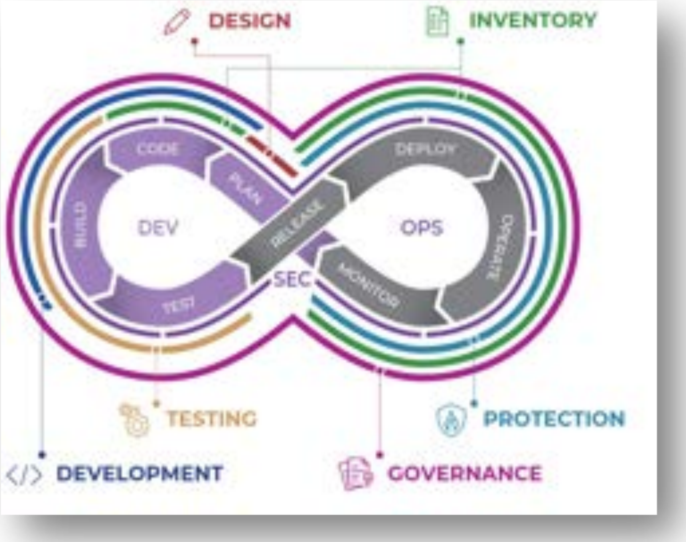
- Code review of critical code paths would detect issues like this
- Security awareness training would prevent basic issues seen here

Testing:

- No evidence of testing – unauthenticated endpoints are easy to discover

Governance:

- Strong governance would ensure issues like this are not released to production



#10: Smart scale

What happened?

Researchers discovered that they could perform a variety of attacks on an API backend for a smart scale, including gaining access to access and refresh tokens, and account takeover using a ‘password reset’ functionality.

Impact:

Vulnerabilities were remediated SEVEN months after disclosure.

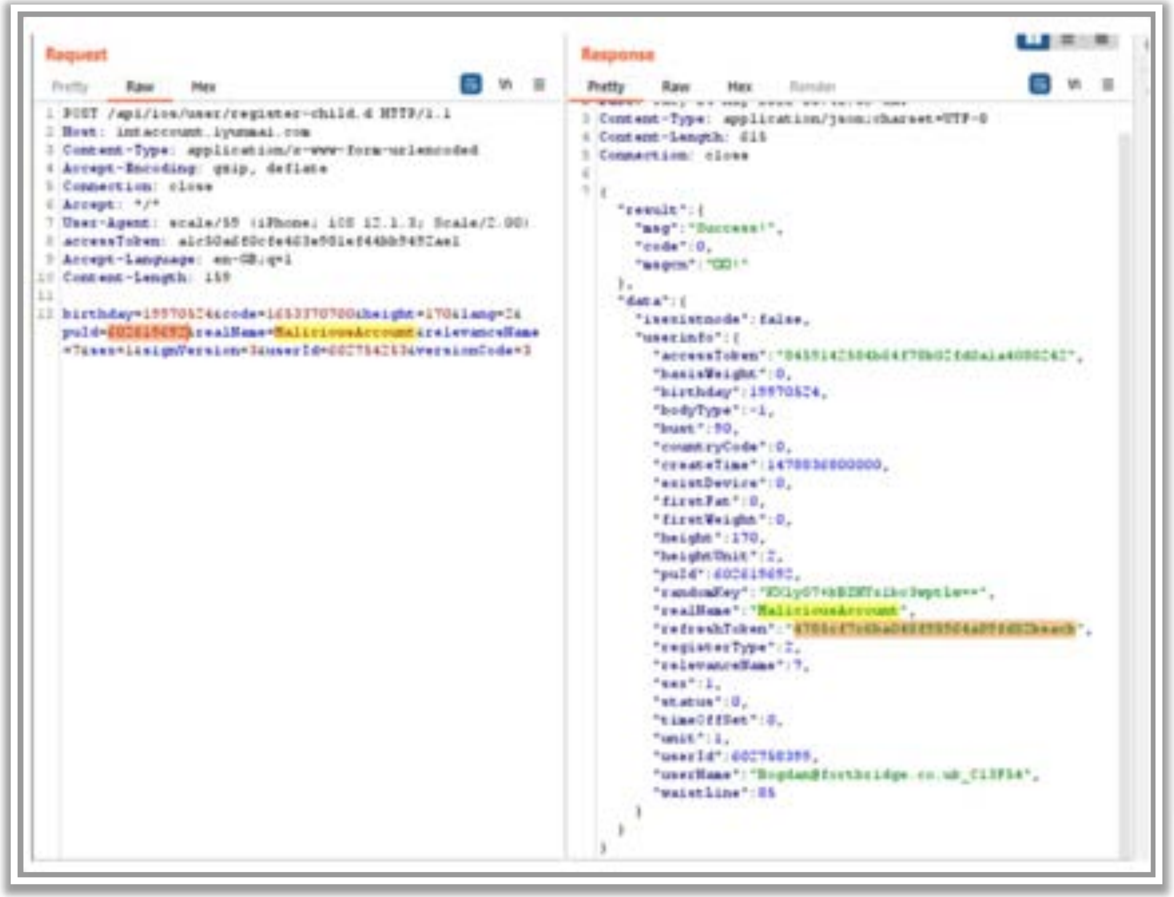
Cause:

- Broken authentication
- Broken object-level authorization
- Excessive data exposure

Lessons learned:

- Multiple vulnerabilities can be effectively combined to achieve total compromise.
- In the event of a disclosure ensure you have a plan for remediation and mitigation.

#10: Smart scale

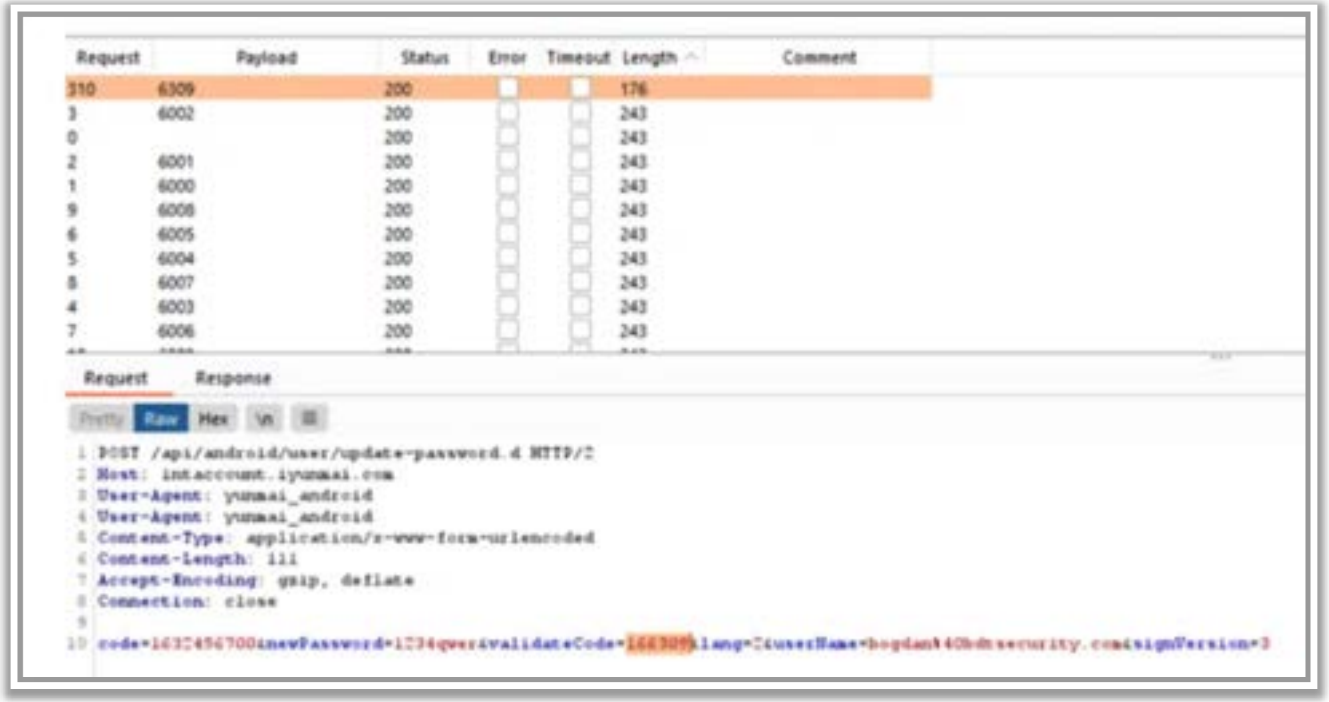


```
POST /api/ios/user/register-child.d HTTP/1.1
Host: intaccount.iyunmai.com
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Connection: close
Accept: */*
User-Agent: scale/55 (iPhone; iOS 12.1.3; Scale/2.00)
accessToken: a1c50ad60cfe463e501ef44bb5452ae1
Accept-Language: en-GB;q=1
Content-Length: 159

birthday=19970524&code=1633770700&height=170&lang=2&
puId=602619692&realName=MaliciousAccount&relevanceName
=7&sex=1&signVersion=3&userId=602754263&versionCode=3
```

```
Content-Type: application/json;charset=UTF-8
Content-Length: 615
Connection: close

{
  "result":{
    "msg":"Success!",
    "code":0,
    "msgcn":"OK!"
  },
  "data":{
    "isexistnode":false,
    "userinfo":{
      "accessToken":"0455142584bd4f70b02fd0a1a4080242",
      "basisWeight":0,
      "birthday":19970524,
      "bodyType":-1,
      "bust":90,
      "countryCode":0,
      "createTime":1478836800000,
      "existDevice":0,
      "firstFat":0,
      "firstWeight":0,
      "height":170,
      "heightUnit":2,
      "puId":602619692,
      "randomKey":"XXly07+bBZNTslbc3wptle==",
      "realName":"MaliciousAccount",
      "refreshToken":"4705cf7c6ba0485f9504a0f8d02eeacb",
      "registerType":2,
      "relevanceName":7,
      "sex":1,
      "status":0,
      "timeOffSet":0,
      "unit":1,
      "userId":602758395,
      "userName":"Bogdan@forthbridge.co.uk_C13F54",
      "waistline":85
    }
  }
}
```



Request	Payload	Status	Error	Timeout	Length	Comment
310	6309	200	□	□	176	
3	6002	200	□	□	243	
0		200	□	□	243	
2	6001	200	□	□	243	
1	6000	200	□	□	243	
9	6008	200	□	□	243	
6	6005	200	□	□	243	
5	6004	200	□	□	243	
8	6007	200	□	□	243	
4	6003	200	□	□	243	
7	6006	200	□	□	243	

```
POST /api/android/user/update-password.d HTTP/2
Host: intaccount.iyunmai.com
User-Agent: yunmai_android
User-Agent: yunmai_android
Content-Type: application/x-www-form-urlencoded
Content-Length: 111
Accept-Encoding: gzip, deflate
Connection: close

code=1632456700&newPassword=1234qwer&validateCode=166309&lang=2&userName=bogdan%40bdmsecurity.com&signVersion=3
```

#10: Smart scale – how to prevent it

Design:

- Insecure defaults – disable debug interface by default
- Threat modelling would have identified issues – guessable IDs, and PINs

Development:

- No understanding of basic API security issues
- Security awareness training would prevent issues seen here

Protection:

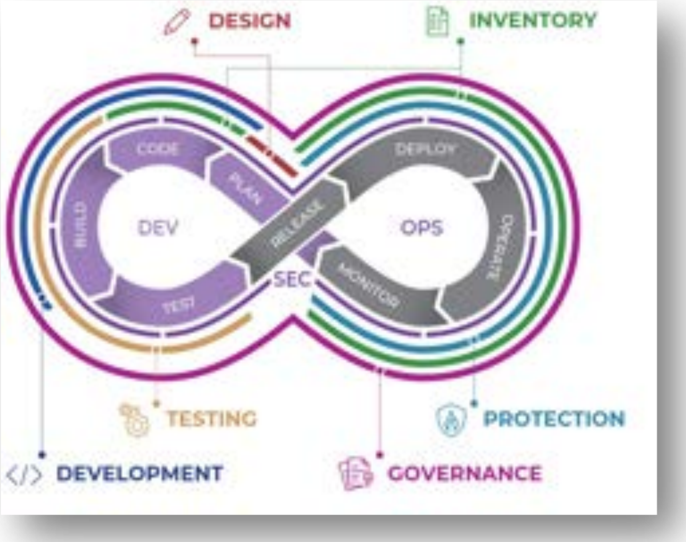
- Excessive information disclosure can easily be prevented

Testing:

- Excessive information disclosure (token leakage) can be easily detected
- Rate-limiting would prevent abuse of password reset function

Governance:

- No responsible disclosure program
- Very slow response to researcher



#11: Automation platform

What happened?

Researchers discovered a vulnerability in the API endpoint that provide remote administration on an industrial automation platform. This allowed remote code execution attacks. Additionally, a file transfer endpoint allowed for overwrite of the local filesystem.

Impact:

Prompt disclosure and a hotfix prevented any compromised.

Cause:

- Broken authentication

Lessons learned:

- Ensure all API endpoints are authenticated.
- Reduce attack surface by removing unnecessary management interfaces.
- Use read-only filesystems for system images such as operating systems.

#11: Automation platform – how to prevent it

Development:

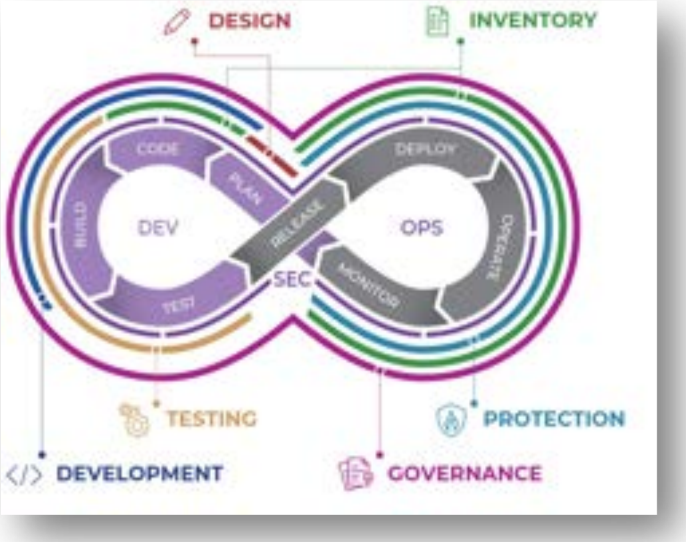
- Code review of critical code paths would detect issues like this

Testing:

- No evidence of testing – unauthenticated endpoints are easy to discover

Governance:

- Strong governance would ensure issues like this are not released to production



#12: CI/CD platform

What happened?

Researchers discovered that they could access historical logs for a popular CI/CD platform by enumerating API endpoints. The logs contained secret information including access tokens and credentials to 3rd party platforms such as GitHub and AWS.

Impact:

Leakage of tens of thousand of access credentials to 3rd party platforms. Vendor claims this is “by design” !

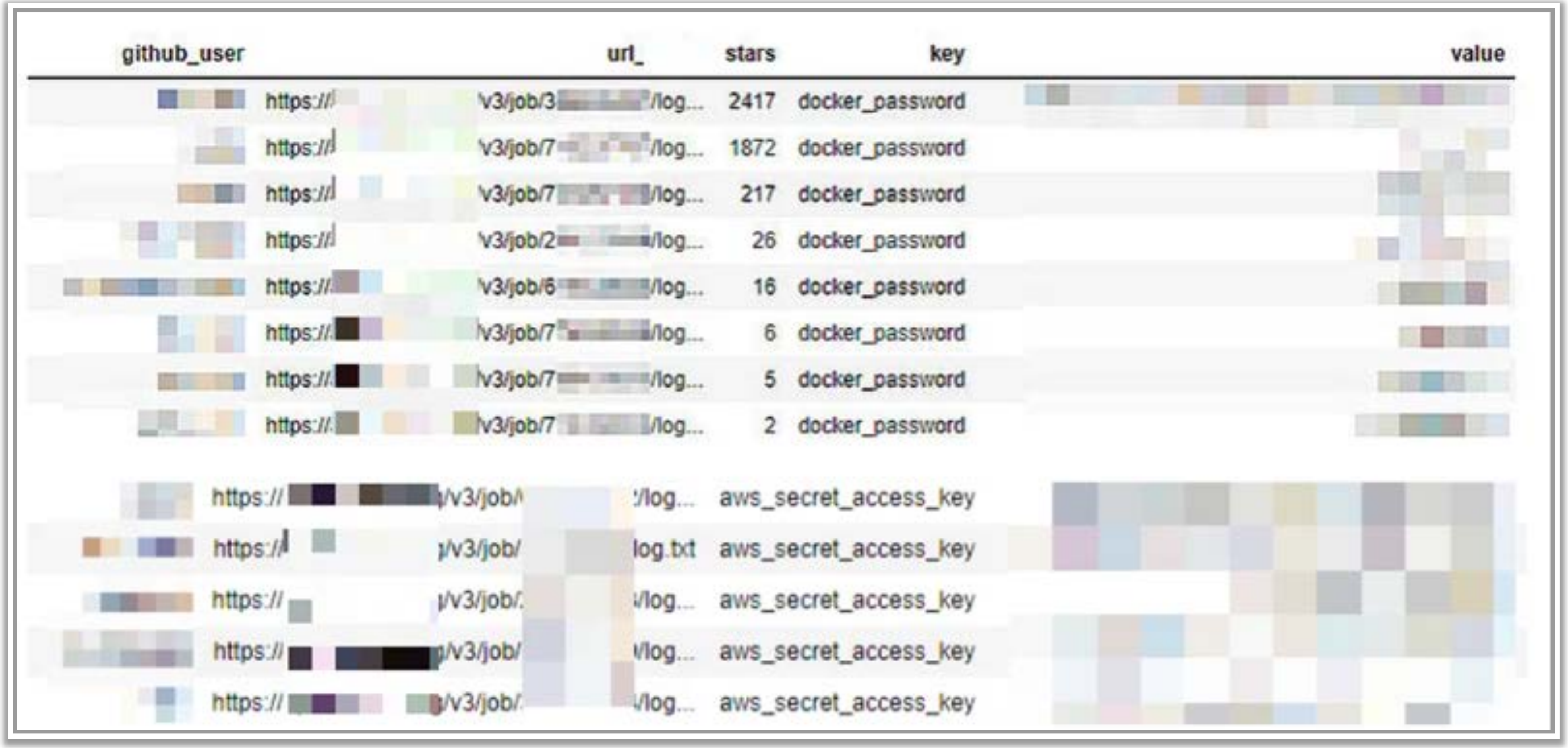
Cause:

- Hidden API endpoints allowed enumeration of archived log files.
- Lack of rate-limiting.

Lessons learned:

- Do not rely on security by obscurity.
- Rate limiting is important, but it is only one element of an API defense strategy.
- Always have a plan for revoking and reissuing credentials.

#12: CI/CD platform



github_user	url_	stars	key	value
[redacted]	https://[redacted]v3/job/3[redacted]/log...	2417	docker_password	[redacted]
[redacted]	https://[redacted]v3/job/7[redacted]/log...	1872	docker_password	[redacted]
[redacted]	https://[redacted]v3/job/7[redacted]/log...	217	docker_password	[redacted]
[redacted]	https://[redacted]v3/job/2[redacted]/log...	26	docker_password	[redacted]
[redacted]	https://[redacted]v3/job/6[redacted]/log...	16	docker_password	[redacted]
[redacted]	https://[redacted]v3/job/7[redacted]/log...	6	docker_password	[redacted]
[redacted]	https://[redacted]v3/job/7[redacted]/log...	5	docker_password	[redacted]
[redacted]	https://[redacted]v3/job/7[redacted]/log...	2	docker_password	[redacted]
[redacted]	https://[redacted]/v3/job/[redacted]/log...		aws_secret_access_key	[redacted]
[redacted]	https://[redacted]/v3/job/[redacted]log.txt		aws_secret_access_key	[redacted]
[redacted]	https://[redacted]/v3/job/[redacted]/log...		aws_secret_access_key	[redacted]
[redacted]	https://[redacted]/v3/job/[redacted]/log...		aws_secret_access_key	[redacted]
[redacted]	https://[redacted]/v3/job/[redacted]/log...		aws_secret_access_key	[redacted]

#12: CI/CD platform – how to prevent it

Design:

- Security by obscurity – don't rely on "hidden" API endpoints
- Use robust credential masking and obscuring methods

Development:

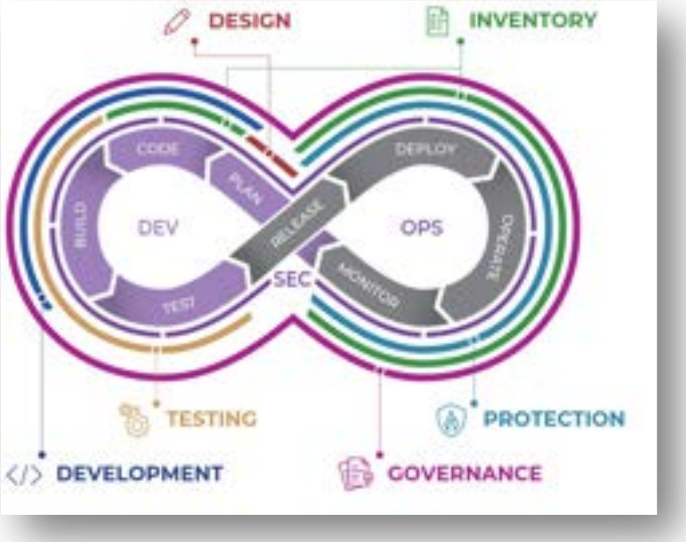
- Awareness training would highlight the need for token protection
- No standard methods used for masking and obfuscation

Protection:

- Rate-limiting would have prevented the brute-force discovery of endpoints

Governance:

- Leakage of 3rd party tokens is not a feature !
- Poor date retention and sanitization processes



The top-ranking issues

Vulnerability	Count
Broken object-level authorization (API1)	5
Lack of rate-limiting (API4)	3
Excessive information exposure (API3)	3
Broken function-level authorization (API5)	3
Broken authentication (API2)	3
Insecure default configuration (API7)	2
Security by obscurity	2
Hardcoded tokens	1
Cross-site request forgery	1

Broken object-level authorization (API1)

Use case

- API call parameters use the ID of the resource accessed through the API /api/shop1/financial_info.
- Attackers replace the IDs of their resources with a different one which they guessed through /api/shop2/financial_info.
- The API does not check permissions and lets the call through.
- Problem is aggravated if IDs can be enumerated /api/123/financial_info.

How to prevent

- Implement authorization checks with user policies and hierarchy.
- Do not rely on IDs that the client sends. Use IDs stored in the session object instead.
- Check authorization for each client request to access database.
- Use random IDs that cannot be guessed (UUIDs).

https://apisecurity.io/encyclopedia/content/owasp/api1-broken-object-level-authorization

Broken authentication (API2)

Use case

- Unprotected APIs that are considered “internal”
- Weak authentication that does not follow industry best practices
- Weak API keys that are not rotated
- Passwords that are weak, plain text, encrypted, poorly hashed, shared, or default passwords
- Authentication susceptible to brute force attacks and credential stuffing
- Credentials and keys included in URLs
- Lack of access token validation (including JWT validation)
- Unsigned or weakly signed non-expiring JWTs

How to prevent

- Check all possible ways to authenticate to all APIs.
- APIs for password reset and one-time links also allow users to authenticate, and should be protected just as rigorously.
- Use standard authentication, token generation, password storage, and multi-factor authentication (MFA).
- Use short-lived access tokens.
- Authenticate your apps (so you know who is talking to you).
- Use stricter rate-limiting for authentication, and implement lockout policies and weak password checks.

https://apisecurity.io/encyclopedia/content/owasp/api2-broken-authentication

Excessive information exposure (API3)

Use case

- The API returns full data objects as they are stored in the backend database.
- The client application filters the responses and only shows the data that the users really need to see.
- Attackers call the API directly and get also the sensitive data that the UI would filter out.

How to prevent

- Never rely on the client to filter data!
- Review all API responses and adapt them to match what the API consumers really need.
- Carefully define schemas for all the API responses.
- Do not forget about error responses, define proper schemas as well.
- Identify all the sensitive data or Personally Identifiable Information (PII), and justify its use.
- Enforce response checks to prevent accidental leaks of data or exceptions.

https://apisecurity.io/encyclopedia/content/owasp/api3-excessive-data-exposure

Rugged by design

Threat modelling

- Establish your attack surface
- Understand the:

  Use cases

  Abuse cases
- Mitigations and compensating controls
- Expose assumptions and misunderstandings

- **Put the Sec into DevOps !**

thaddeus e. grugq
@thegrugq

Your threat model is not my threat model.

8:42 AM · May 15, 2017 · Tweetbot for iOS

703 Retweets 1.3K Likes

Think like an attacker

Understand your data

- Privacy requirements
- What is your PII ?
- Handling privacy requests
- Data ownership

- **Expose the bare minimum of data !**

Understand your environments

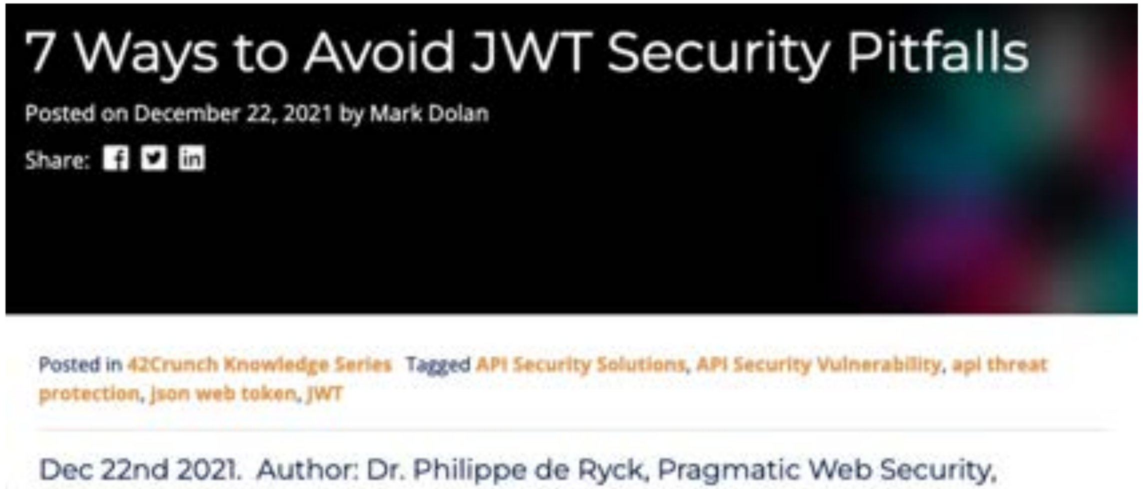
- Secret storage
- Token exchange
- Device integrity
- Channel integrity

- **Assume your environment is hostile**

Rugged by implementation

Boss your JWT validation

1. Use a Static Algorithm Configuration
2. Use Explicit Typing
3. Go All Out on Metadata
4. Treat your Secrets as Secrets
5. Bring Down the Testing Hammer
6. Encapsulating Security Behavior
7. Rely on Static Code Analysis



7 Ways to Avoid JWT Security Pitfalls

Posted on December 22, 2021 by Mark Dolan

Share:

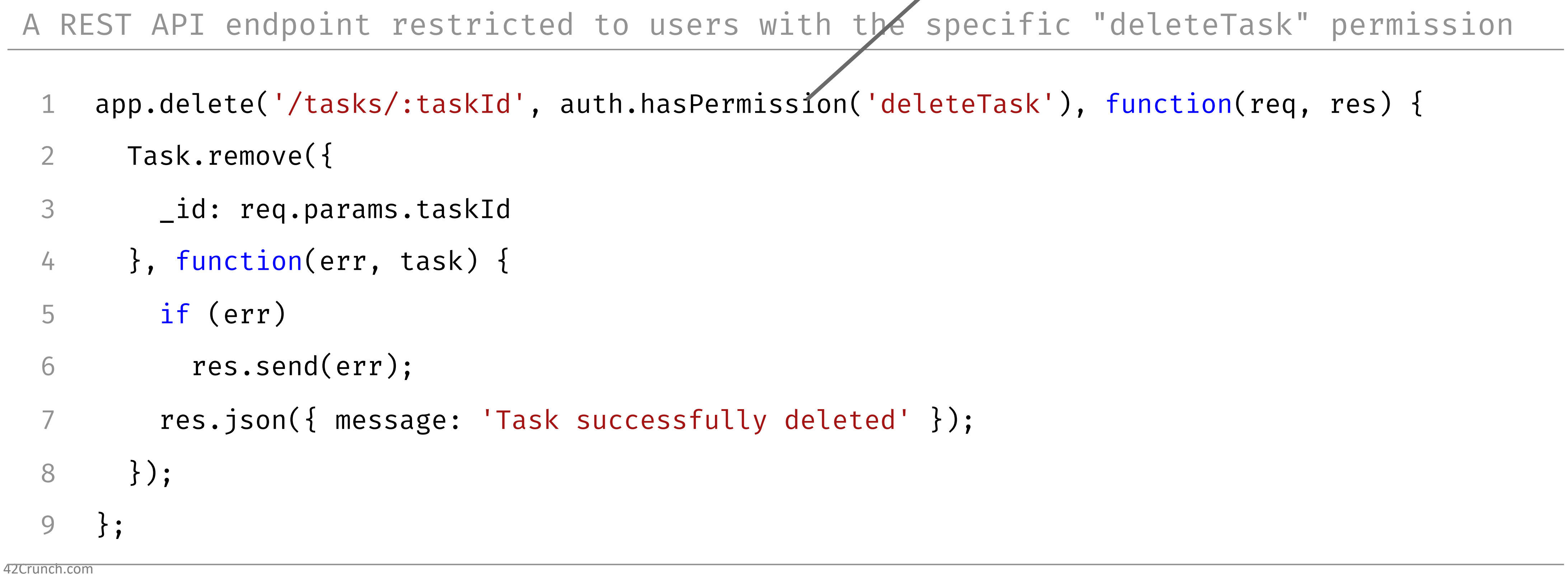
Posted in 42Crunch Knowledge Series Tagged API Security Solutions, API Security Vulnerability, api threat protection, json web token, JWT

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

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

Authorize every function endpoint

Check permissions explicitly for every function access to prevent *broken function-level authorization vulnerabilities*



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

```
app.delete('/tasks/:taskId', auth.hasPermission('deleteTask'), function(req, res) {
  Task.remove({
    _id: req.params.taskId
  }, function(err, task) {
    if (err)
      res.send(err);
    res.json({ message: 'Task successfully deleted' });
  });
};
```

Be explicit about intended anonymous access

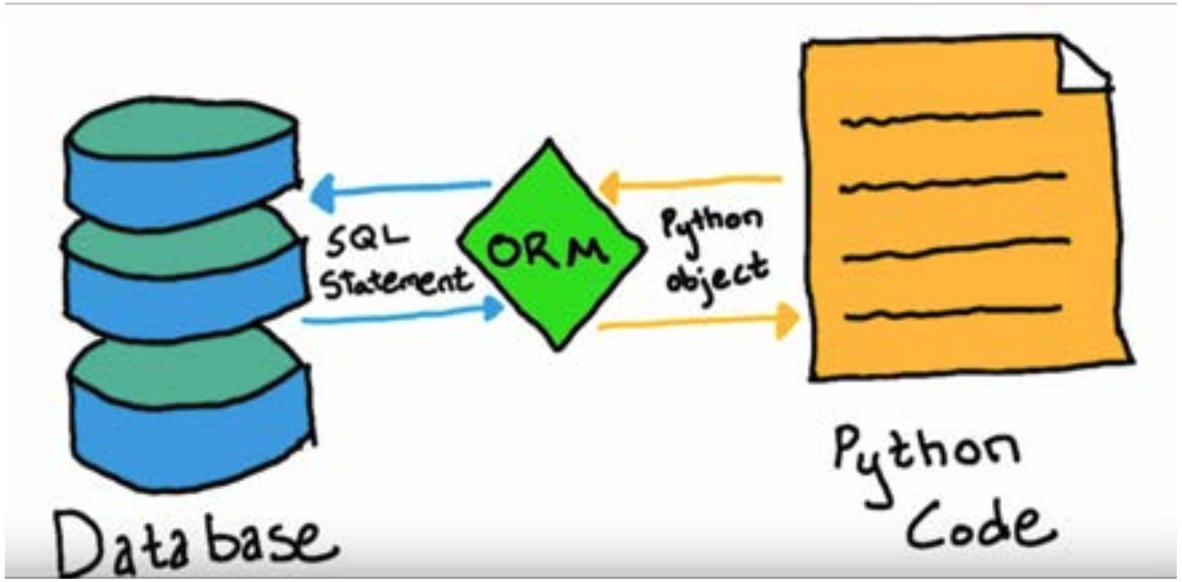
Explicitly declare methods with intended anonymous access with appropriate decorator or notation

A REST API endpoint to get a task

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

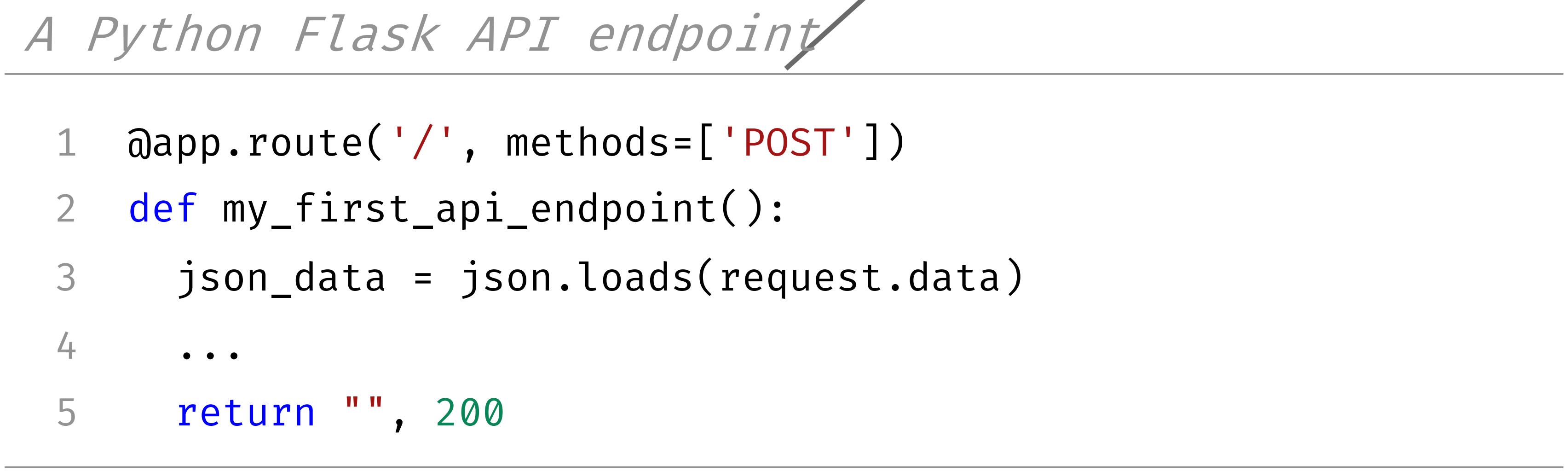
Be wary of your Object Relational Mapper

- ORMs are a massive development convenience speeding up coding effort
- ORMs are a leading cause of:
  - Excessive data exposure
  - Mass assignment
- Always understand what your ORM is doing for you, and decouple from direct API access



Explicitly enable methods, prevent unused methods

Explicitly specify the methods to be used, and prevent unused methods in the framework used

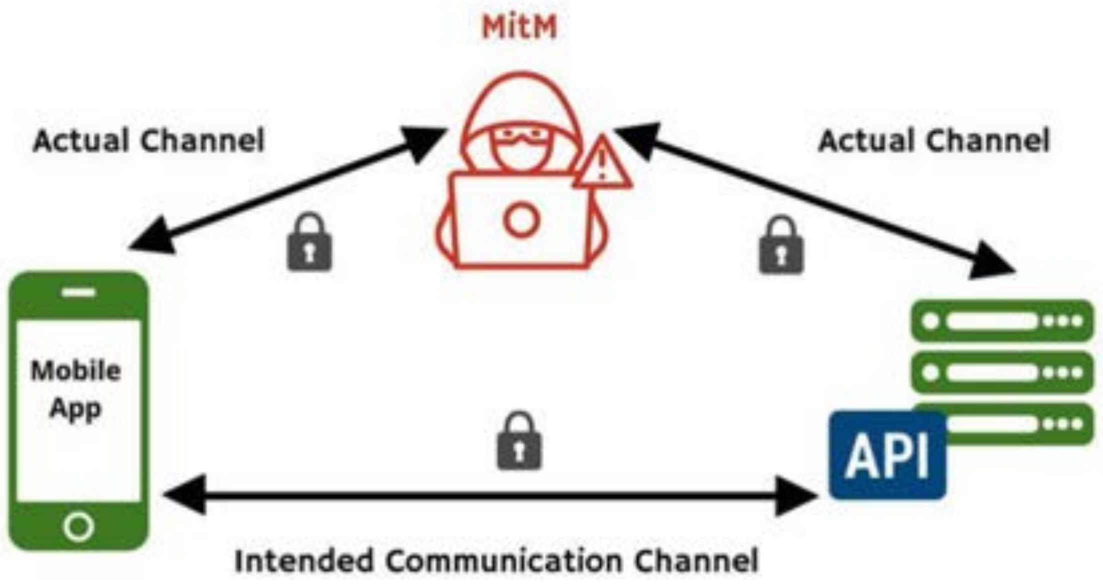


A Python Flask API endpoint

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

Certificate pinning for mobile applications

- Your mobile application is one of your top attack vectors
- Without certificate pinning your backend APIs can be easily reverse engineered
- Tokens and secrets can easily be exfiltrated via a proxy



https://approov.io/blog/man-in-the-middle-myths-and-legends

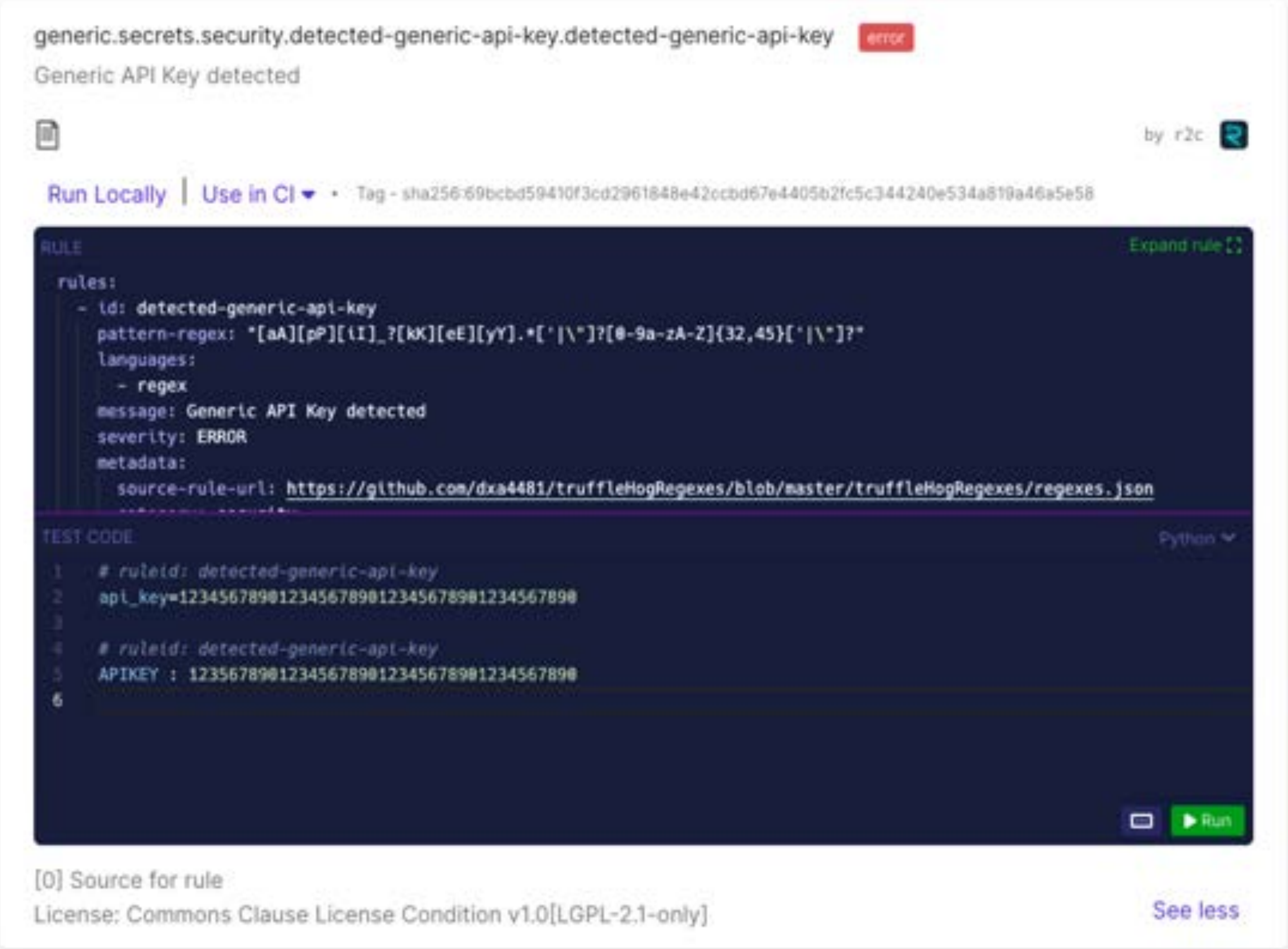
Use standard libraries and components

- NEVER write your own cryptography functions or methods
- Use standard libraries for common functionality (and scan them for vulnerabilities)
- Stand on the shoulder of giants (and contribute fixes and improvements)

Semgrep FTW

Search for:

- Hardcoded secrets
- Missing AuthN/AuthZ decorators
- JWT validation errors
- Missing method decorators
- Transport misconfiguration



generic.secrets.security.detected-generic-api-key.detected-generic-api-key

Generic API Key detected

error
by r2c
Run Locally
Use in CI
Tag - sha256:69bcbd59410f3cd2961848e42ccbd67e4405b2fc5c344240e534a819a46a5e58
RULE
Expand rule
TEST CODE
Python
Run

```
rules:
  - id: detected-generic-api-key
    pattern-regex: "[aA][pP][iI]_?[kK][eE][yY].*['|\"]?[0-9a-zA-Z]{32,45}['|\"]?"
    languages:
      - regex
    message: Generic API Key detected
    severity: ERROR
    metadata:
      source-rule-url: https://github.com/dxa4481/truffleHogRegexes/blob/master/truffleHogRegexes/regexes.json
```

```
# ruleid: detected-generic-api-key
api_key=1234567890123456789012345678901234567890

# ruleid: detected-generic-api-key
APIKEY : 123567890123456789012345678901234567890
```

[0] Source for rule

License: Commons Clause License Condition v1.0[LGPL-2.1-only]

https://semgrep.dev/

Beware of your framework default settings

- Insecure defaults
- Default handlers
- Enabling unused verbs
- Debug info
- Implicit behaviours

spring boot

node JS

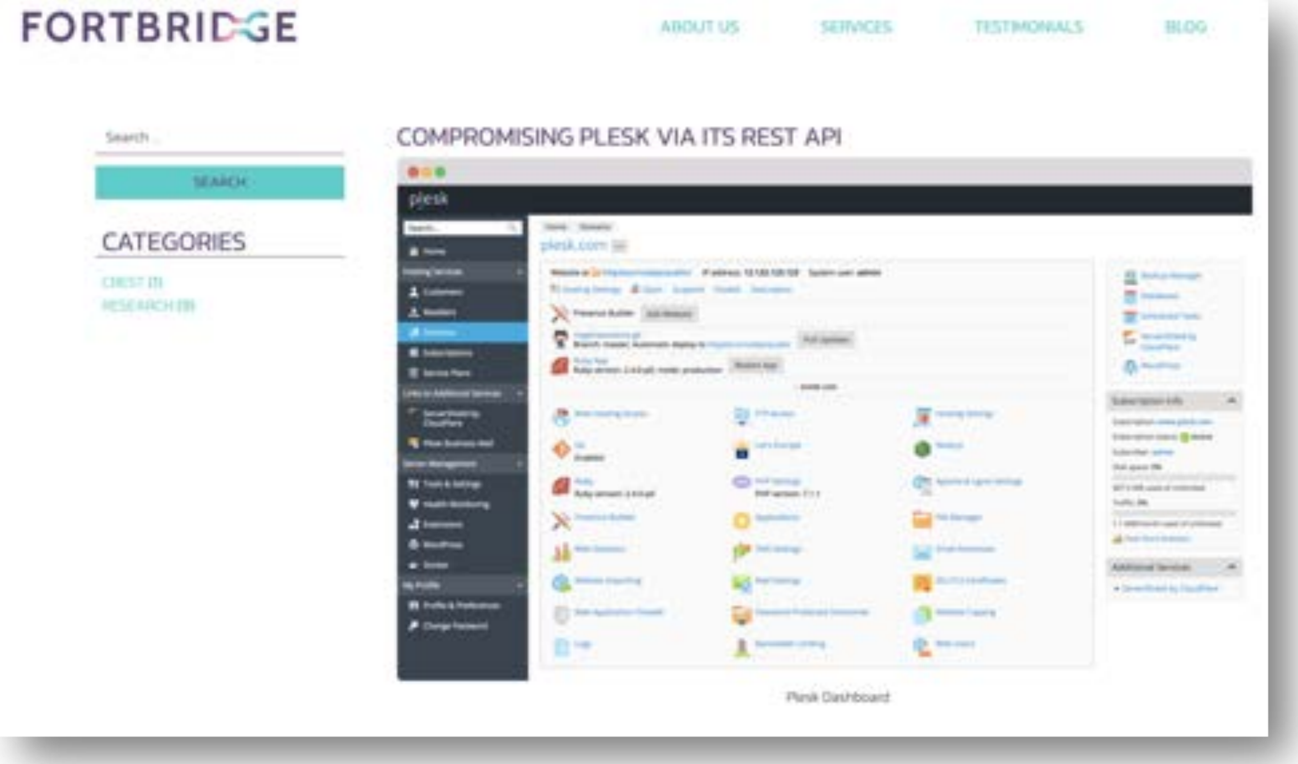
Express

.NET Core

RESTful API

Don’t forget the old favorites

- CSRF in API-backed GUIs
- Command inject in routers and appliances



FORTBRIDGE

ABOUT US SERVICES TESTIMONIALS BLOG

Search

SEARCH

CATEGORIES

CREST (1)

RESEARCH (8)

COMPROMISING PLESK VIA ITS REST API

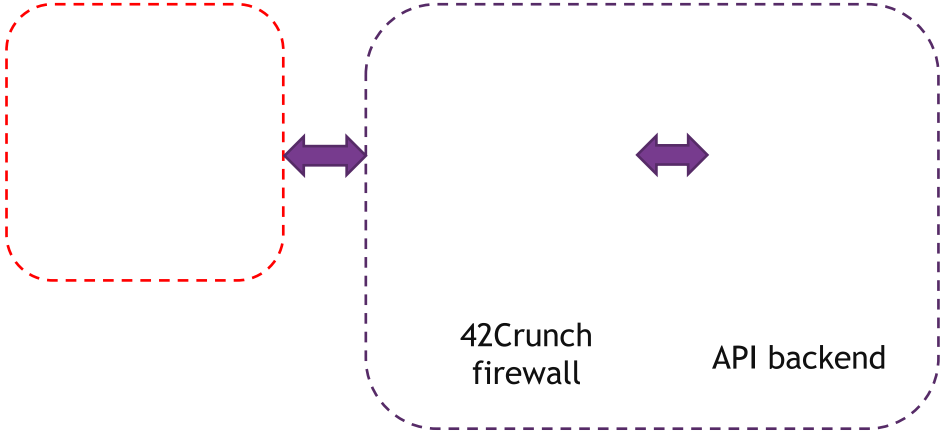
Plesk Dashboard

42crunch
WEBINAR SERIES
DEFENDING APIs
with JIM MANICO
Part 1 Request Forgery on the Web CSRF & SSRF
Part 2 Passwords: Policy & Storage with NIST SP800-63b

Rugged at runtime

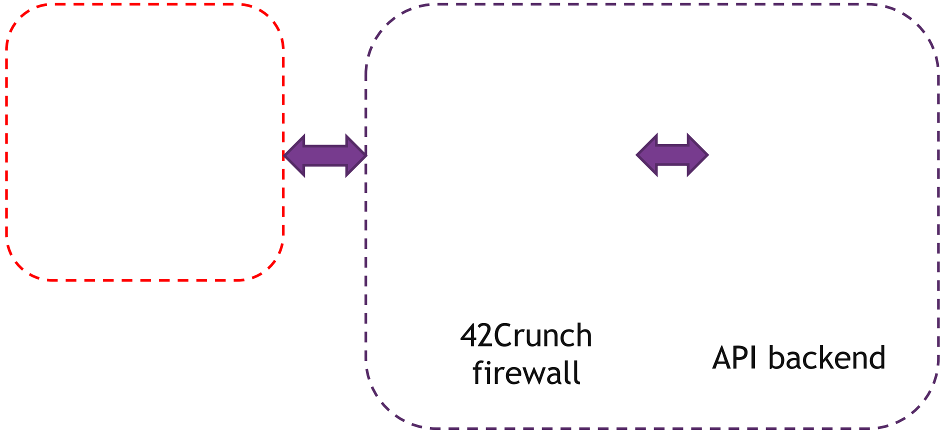
Rate limiting

This protection limits how many requests API Firewall accepts from an IP address within a given time window.



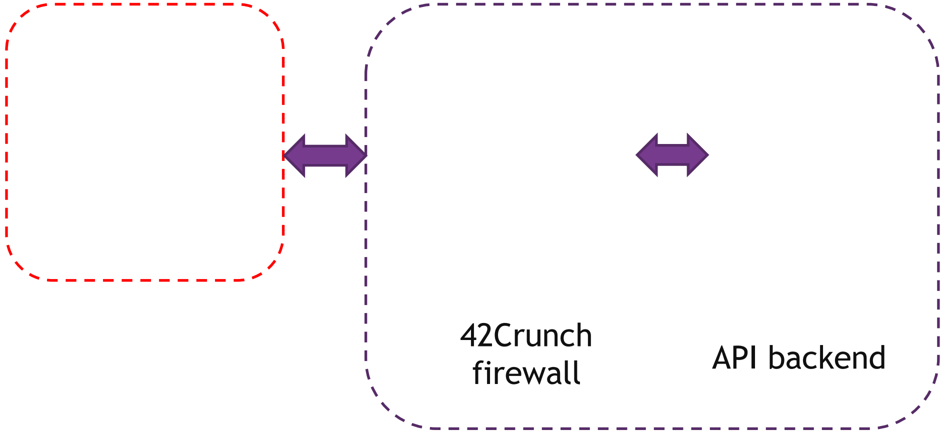
JWT token validation

JWT token validation performs a variety of checks on request tokens and blocks invalid requests

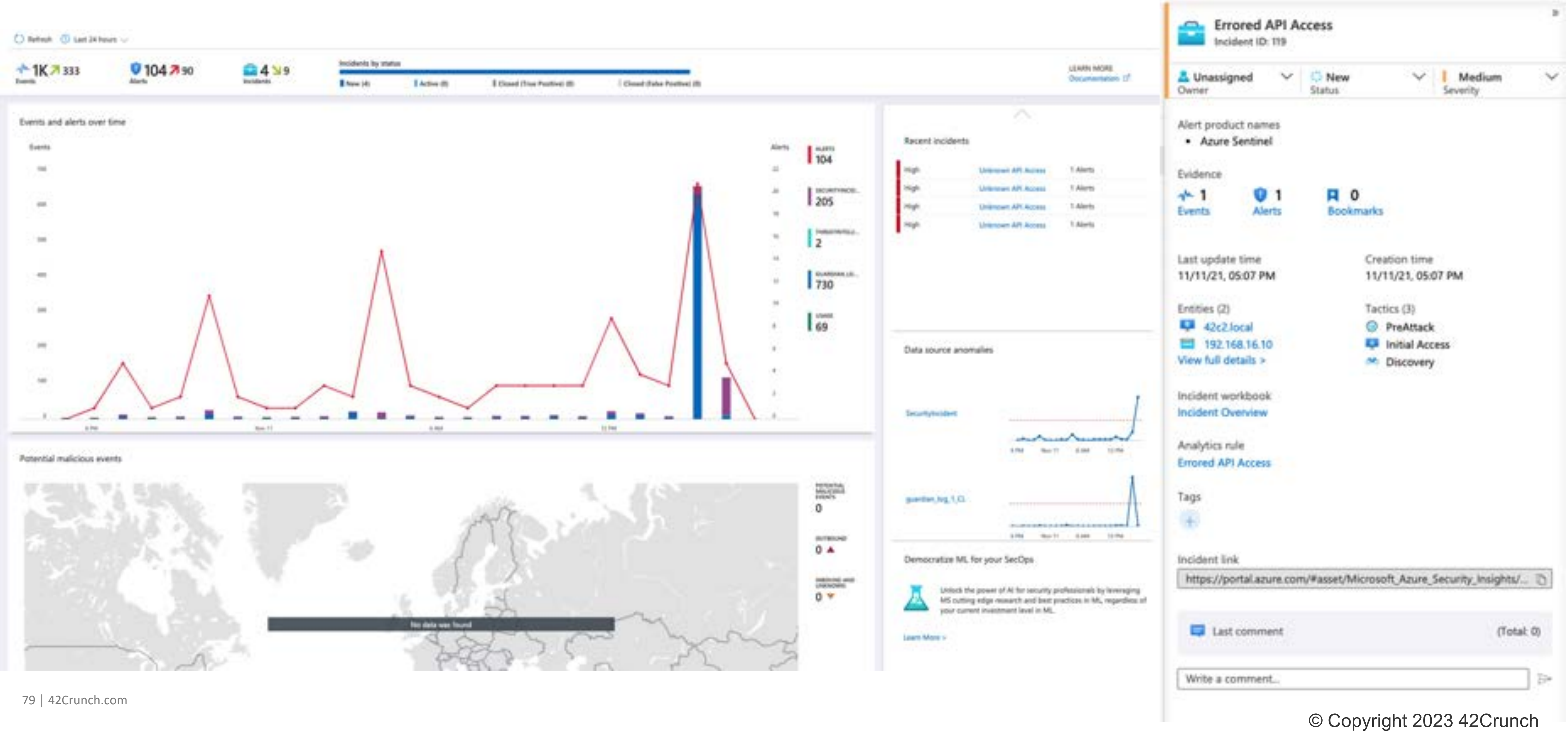


Security headers

These protections on APIs control security headers either locally to specific paths, operations, responses, or alternatively to all incoming requests or outgoing responses.



API monitoring in your SIEM – Azure Sentinel



Recent incidents

High	Unknown API Access	1 Alerts
High	Unknown API Access	1 Alerts
High	Unknown API Access	1 Alerts
High	Unknown API Access	1 Alerts

Unlock the power of AI for security professionals by leveraging MS cutting edge research and best practices in ML, regardless of your current investment level in ML.

Errored API Access
Incident ID: 119

Unassigned
Owner

New
Status

Medium
Severity

Alert product names
- Azure Sentinel

Evidence
1 Events
1 Alerts
0 Bookmarks

Last update time
11/11/21, 05:07 PM

Creation time
11/11/21, 05:07 PM

Entities (2)
42c2.local
192.168.16.10
View full details >

Tactics (3)
PreAttack
Initial Access
Discovery

Incident workbook
Incident Overview

Analytics rule
Errored API Access

Tags

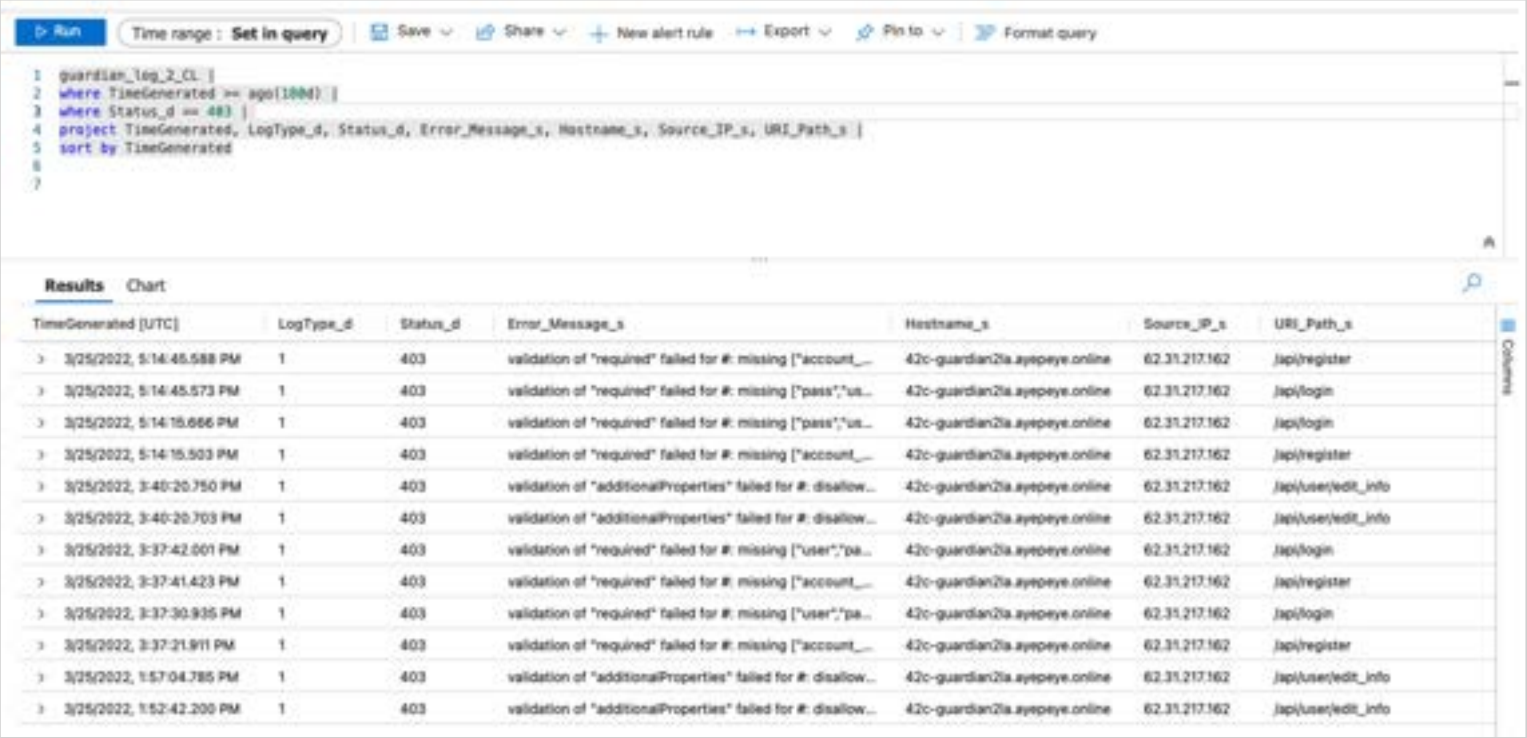
Incident link
https://portal.azure.com/#asset/Microsoft_Azure_Security_Insights/...

Last comment (Total: 0)

Write a comment...

Continuous threat detection

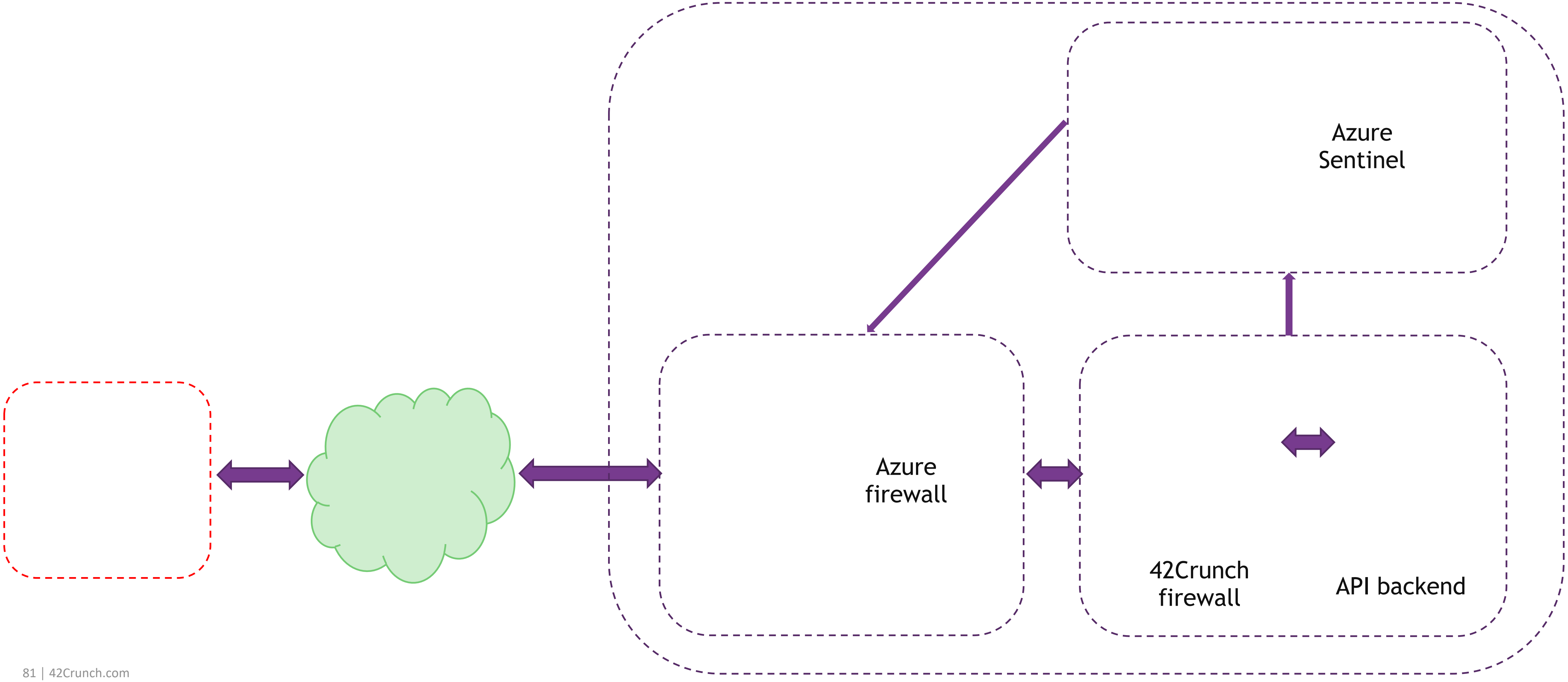
- Excessive AuthZ errors
- Path enumeration
- Rate limiting
- Excessive requests
- Tool detection
- Fuzzing detection
- Risky IP address detection



```
guardian_log_2_CL |
where TimeGenerated >= ago(100d) |
where Status_d == 403 |
project TimeGenerated, LogType_d, Status_d, Error_Message_s, Hostname_s, Source_IP_s, URI_Path_s |
sort by TimeGenerated
```

TimeGenerated [UTC]	LogType_d	Status_d	Error_Message_s	Hostname_s	Source_IP_s	URI_Path_s
3/25/2022, 5:14:45.588 PM	1	403	validation of "required" failed for #: missing ["account_...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/register
3/25/2022, 5:14:45.573 PM	1	403	validation of "required" failed for #: missing ["pass","us...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/login
3/25/2022, 5:14:15.666 PM	1	403	validation of "required" failed for #: missing ["pass","us...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/login
3/25/2022, 5:14:15.503 PM	1	403	validation of "required" failed for #: missing ["account_...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/register
3/25/2022, 3:40:20.750 PM	1	403	validation of "additionalProperties" failed for #: disallow...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/user/edit_info
3/25/2022, 3:40:20.703 PM	1	403	validation of "additionalProperties" failed for #: disallow...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/user/edit_info
3/25/2022, 3:37:42.001 PM	1	403	validation of "required" failed for #: missing ["user","pa...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/login
3/25/2022, 3:37:41.423 PM	1	403	validation of "required" failed for #: missing ["account_...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/register
3/25/2022, 3:37:30.935 PM	1	403	validation of "required" failed for #: missing ["user","pa...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/login
3/25/2022, 3:37:21.911 PM	1	403	validation of "required" failed for #: missing ["account_...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/register
3/25/2022, 1:57:04.785 PM	1	403	validation of "additionalProperties" failed for #: disallow...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/user/edit_info
3/25/2022, 1:52:42.200 PM	1	403	validation of "additionalProperties" failed for #: disallow...	42c-guardian2la.ayepeye.online	62.31.217.162	/api/user/edit_info

Leveraging your Cloud platform protections

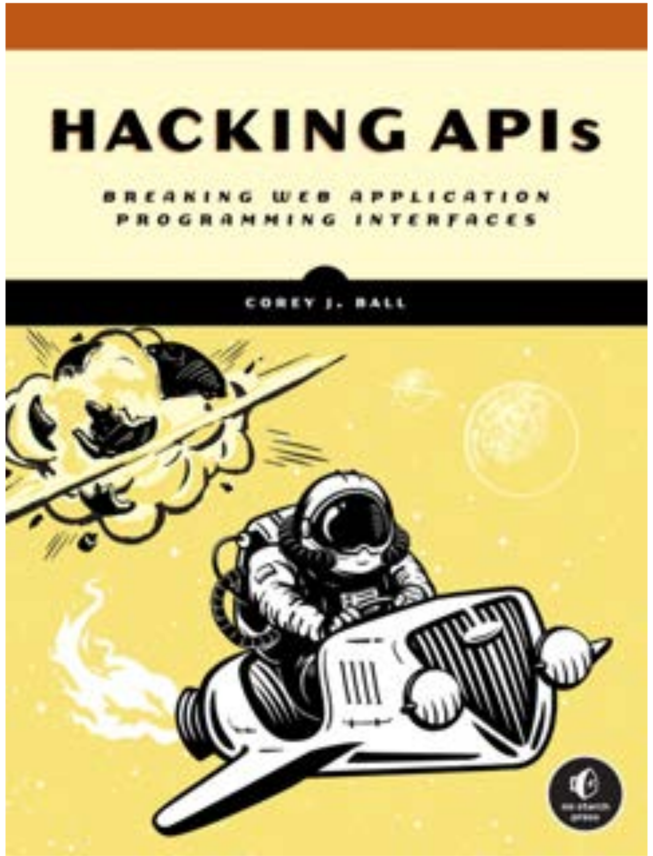


Learning more

APISecurity.io

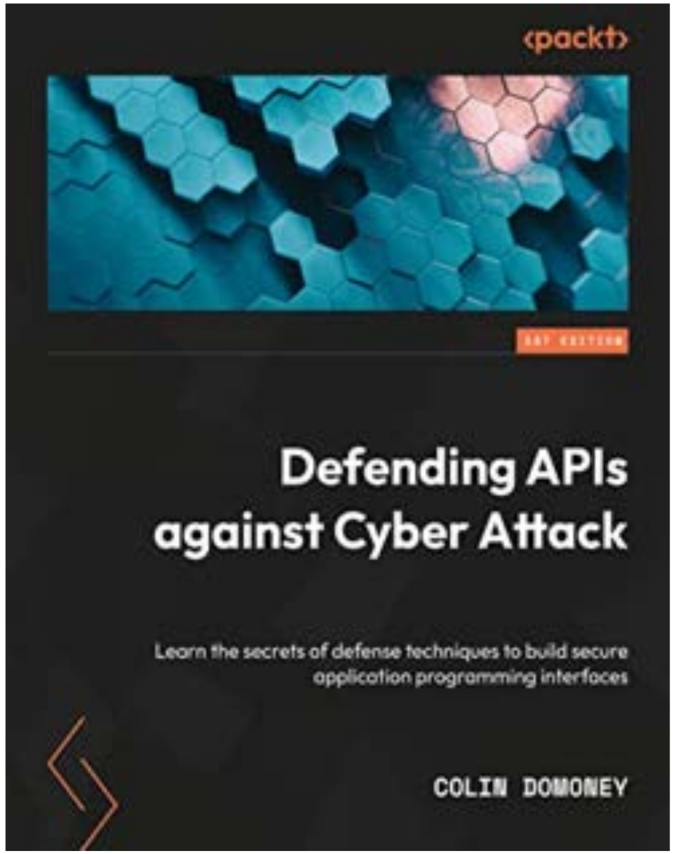
https://apisecurity.io/

“Hacking APIs” – Corey Ball



https://nostarch.com/hacking-apis

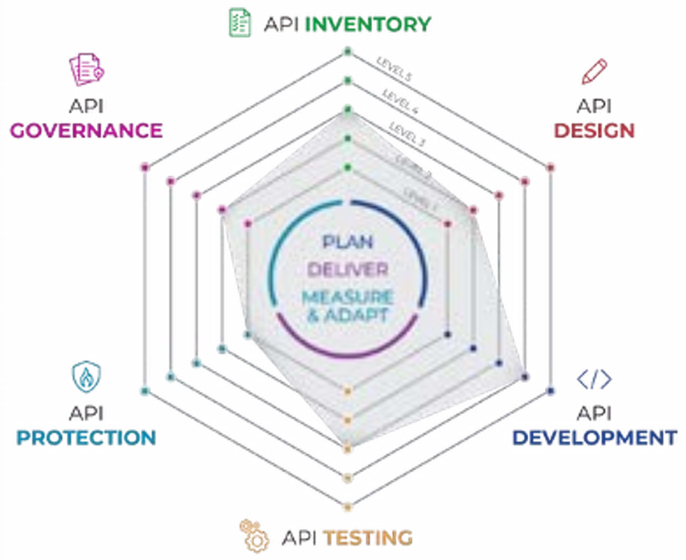
“Defending APIs against Cyber Attack” – Colin Domoney


<packt>
1ST EDITION
Defending APIs against Cyber Attack
Learn the secrets of defense techniques to build secure application programming interfaces
COLIN DOMONEY

https://amzn.to/3fHp8Mz

eBook

API Security: A Blueprint for Success



- *Practical Guide on an API Security program.*
- *Map your enterprise's API security posture against 6 key domains.*
- *Champion the case for API Security.*

Download here: https://42crunch.com/ebook-api-security-blueprint/

APIsecurity.io Community Newsletter: https://apisecurity.io/