21 June 2022

API Breaches for H1 2022

Colin Domoney

API Security Research Specialist & Developer Advocate

Introduction

About the Speaker

Colin Domoney

API Security Research Specialist & Developer Advocate

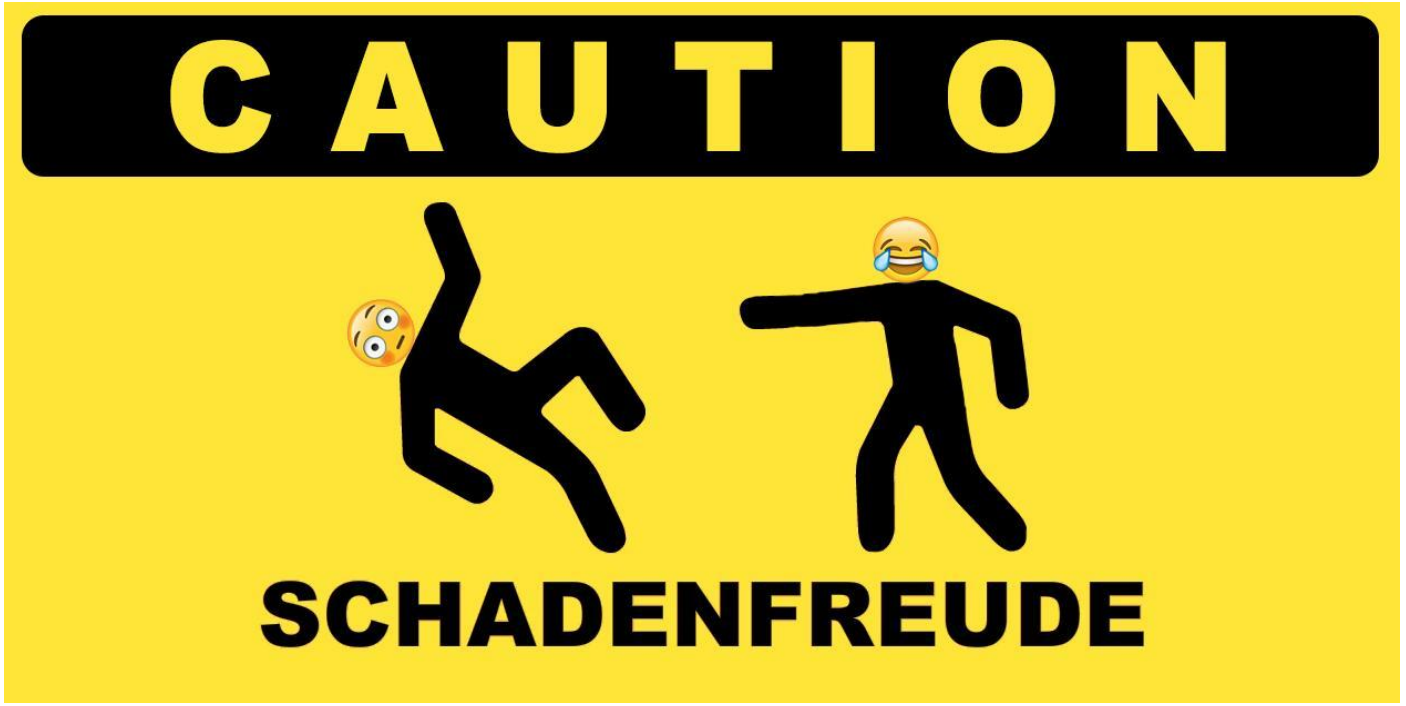
Editor of APISecurity.io
Cyberbroof, Veracode, CA, Deutsche Bank

Housekeeping Rules

- All attendees muted
- Questions via Q&A
- Recording will be shared

To err is human

- Mistakes happen to **everyone**
- Mistakes can be **very costly**
- Mistakes are a **learning opportunity**
- Mistakes can often **easily be detected**



#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.

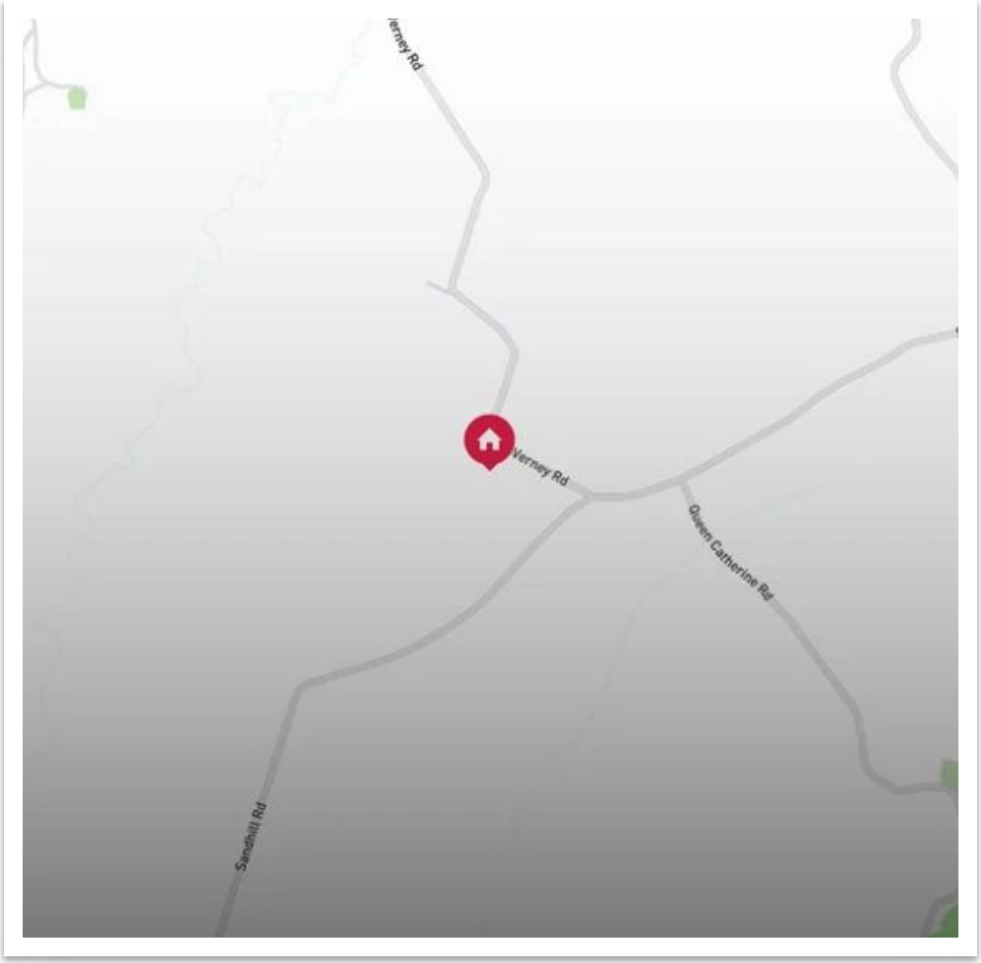
Cause:

- 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



JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

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

#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.

#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]uk/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://[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"}})};
```

Response

Pretty Raw Hex Render \n Select extension...


```
"id":[redacted],
"group_id":8,
"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]"
    ],
    "telephone":"+44777[redacted]",
    "postcode":"[redacted]",
    "city":"[redacted]",
    "firstname":"Alan",
    "lastname":"Monie",
    "default_shipping":true
  },
```

#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.

#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:

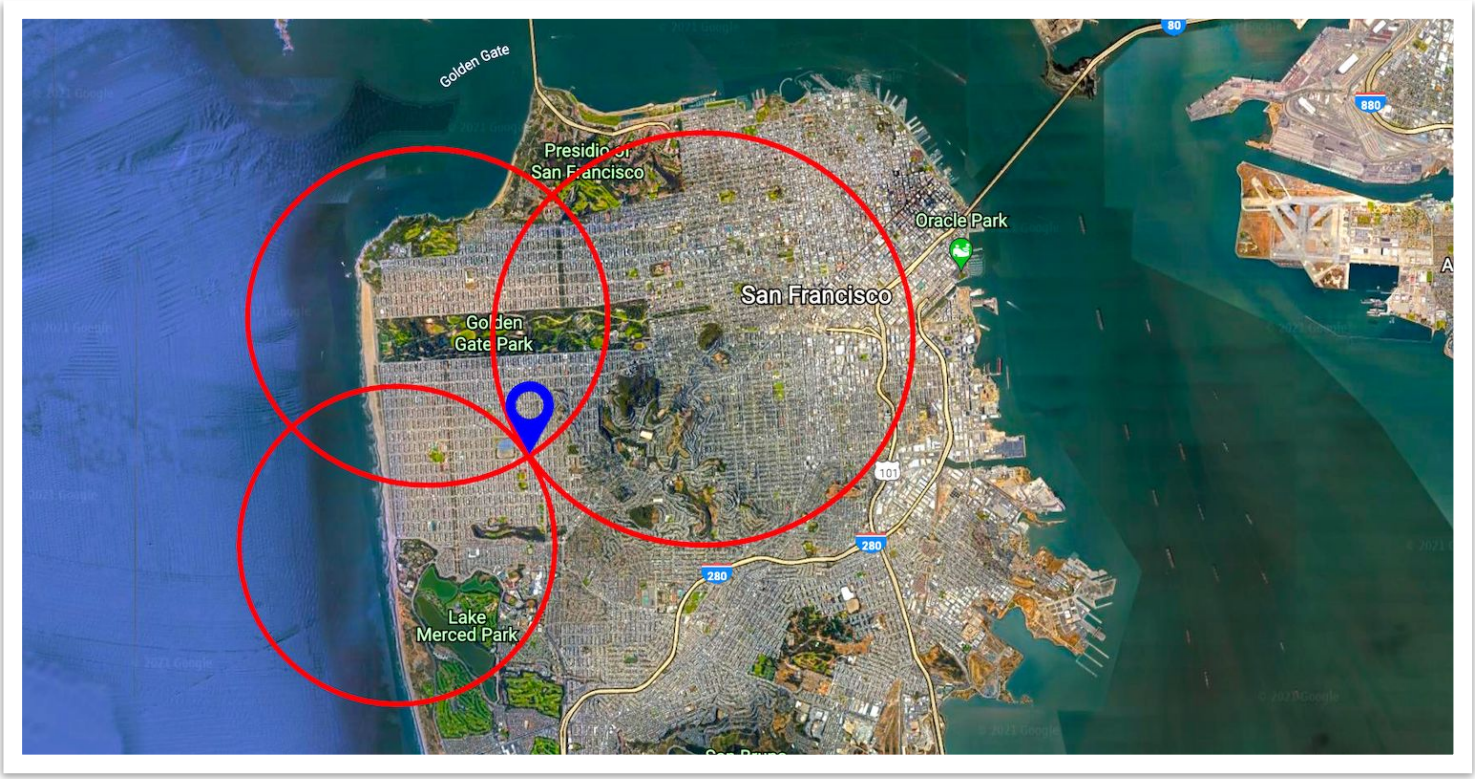
- Another example of broken object-level authorization
- Excessive information exposure allowed inference of user location to a high level of precision.
- Security by obscurity

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...
}
```



#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.

#7 : A case study of API vulnerabilities

What happened?

A researcher discovered an assortment of API vulnerabilities can be chained together to totally compromise an affected platform.

Impact:

Minimal due to immediate, responsible disclosure.

Cause:

- UUID leakage
- Broken object-level authorization
- Broken function-level authorization
- Poor JWT validation
- Secrets committed to source code repositories
- Leakage of API tokens via web UI

Lessons learned:

- Skilled attackers can chain vulnerabilities together to achieve total compromise.

#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.

#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.

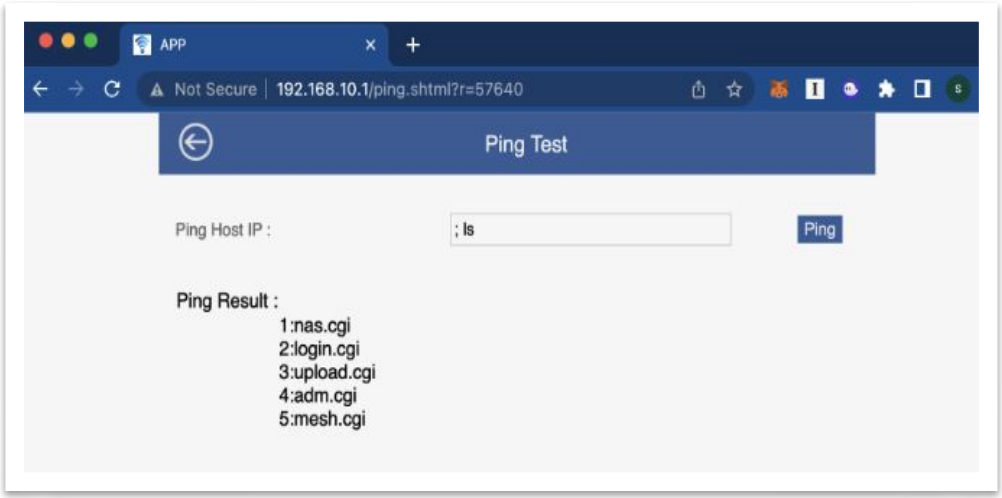
Cause:

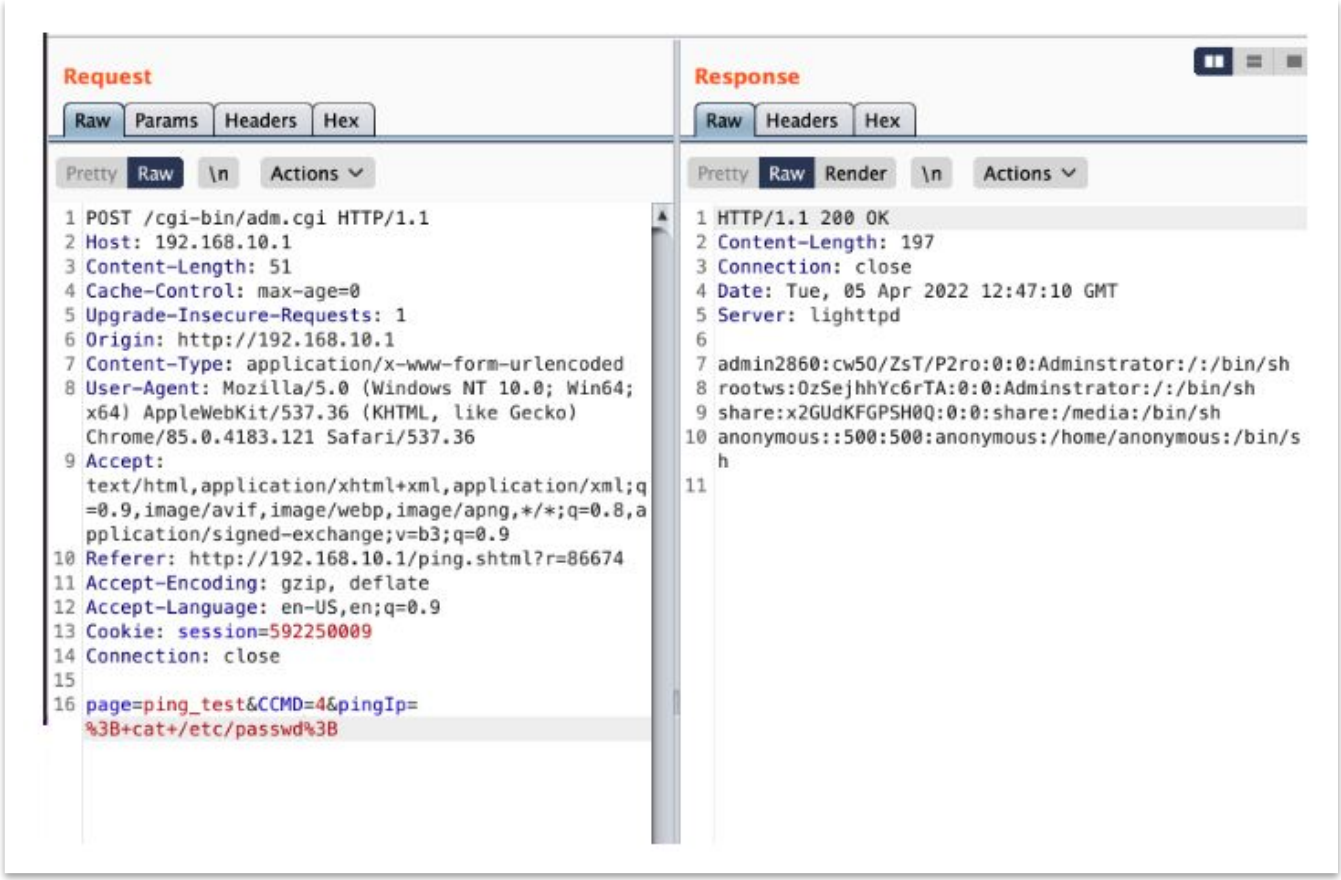
- Broken authentication
- Cross-site request forgery

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





#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

Request

Pretty Raw Hex

```
1 POST /api/ios/user/register-child.d HTTP/1.1
2 Host: intaccount.iyunmai.com
3 Content-Type: application/x-www-form-urlencoded
4 Accept-Encoding: gzip, deflate
5 Connection: close
6 Accept: */*
7 User-Agent: scale/59 (iPhone; iOS 12.1.3; Scale/2.00)
8 accessToken: a1c50a6f0cfe463e981ef44bb9492ae1
9 Accept-Language: en-GB;q=1
10 Content-Length: 159
11
12 birthday=19970524&code=1653370700&height=170&lang=2&
puId=602619692&realName=MaliciousAccount&relevanceName
=7&sex=1&signVersion=3&userId=602754253&versionCode=3
```

Response

Pretty Raw Hex Render

```
3 Content-Type: application/json;charset=UTF-8
4 Content-Length: 615
5 Connection: close
6
7 {
  "result":{
    "msg":"Success!",
    "code":0,
    "msgcn":"□□!"
  },
  "data":{
    "isexistnode":false,
    "userinfo":{
      "accessToken":"8459142584b64f78b02fd0a1a4080242",
      "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":"KX1y07+kBZHYx1bc3wptLw==",
      "realName":"MaliciousAccount",
      "refreshToken":"4785cf7c6ba048f98964a89fd82beacb",
      "registerType":2,
      "relevanceName":7,
      "sex":1,
      "status":0,
      "timeOffSet":0,
      "unit":1,
      "userId":602758399,
      "userName":"Bogdan@fortbridge.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	

Request Response

Pretty Raw Hex \n

```
1 POST /api/android/user/update-password.d HTTP/2
2 Host: intaccount.iyunmai.com
3 User-Agent: yunmai_android
4 User-Agent: yunmai_android
5 Content-Type: application/x-www-form-urlencoded
6 Content-Length: 111
7 Accept-Encoding: gzip, deflate
8 Connection: close
9
10 code=1632496700&newPassword=1234qwer&validateCode=166309&lang=2&userName=bogdan%40bdtsecurity.com&signVersion=3
```

#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.

#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
https://	v3/job/3 /log...	2417	docker_password	
https://	v3/job/7 /log...	1872	docker_password	
https://	v3/job/7 /log...	217	docker_password	
https://	v3/job/2 /log...	26	docker_password	
https://	v3/job/6 /log...	16	docker_password	
https://	v3/job/7 /log...	6	docker_password	
https://	v3/job/7 /log...	5	docker_password	
https://	v3/job/7 /log...	2	docker_password	
https://	/v3/job/ /log...		aws_secret_access_key	
https://	/v3/job/ log.txt		aws_secret_access_key	
https://	/v3/job/ /log...		aws_secret_access_key	
https://	/v3/job/ /log...		aws_secret_access_key	
https://	/v3/job/ /log...		aws_secret_access_key	

Protecting your APIs

Best practices

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

Learning more

APISecurity.io

https://apisecurity.io/

Episode Two - Remediation and Prevention

https://42crunch.com/api-breaches-h1-2022/

Upcoming News

Further Activities

Amsterdam

Charlotte

Dallas

San Jose

Part 2: API Breaches in H12022. July 21, 2022

Demo of Remediating the key API Breaches

APISecurity.io Weekly Newsletter

https://apisecurity.io/

OpenAPI Editor - Free Download

https://42crunch.com/resources-free-tools/