API Security

Erez Yalon

OWASP Singapore | 13 July 2021

@ErezYalon


IBM
Google
Azure
GitLab
DigitalOcean
CISCO
okcupid
Clubhouse
Drop-in Audio
zoom
facebook
vmware
DigiLocker
Your documents anytime, anywhere
Question: What do all this logos have in common?

Who am I?

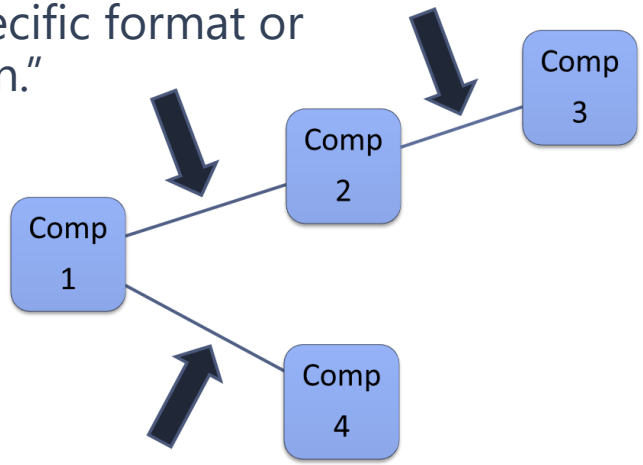
Erez Yalon, Head of Security Research, Checkmarx

- Previous independent security researcher and developer
- Better at breaking than building
- Responsible for maintaining Checkmarx's top notch vulnerability detection technology
- Lead several OWASP projects including the API Security and CN Projects
- Founder of AppSec Village in DEF CON

What is an API?

"An application programming interface (API) is an interface or communication protocol between a client and a server intended to simplify the building of client-side software. It has been described as a "contract" between the client and the server, such that if the client makes a request in a specific format, it will always get a response in a specific format or initiate a defined action."

Wikipedia



But what is API Security?

Checkmarx

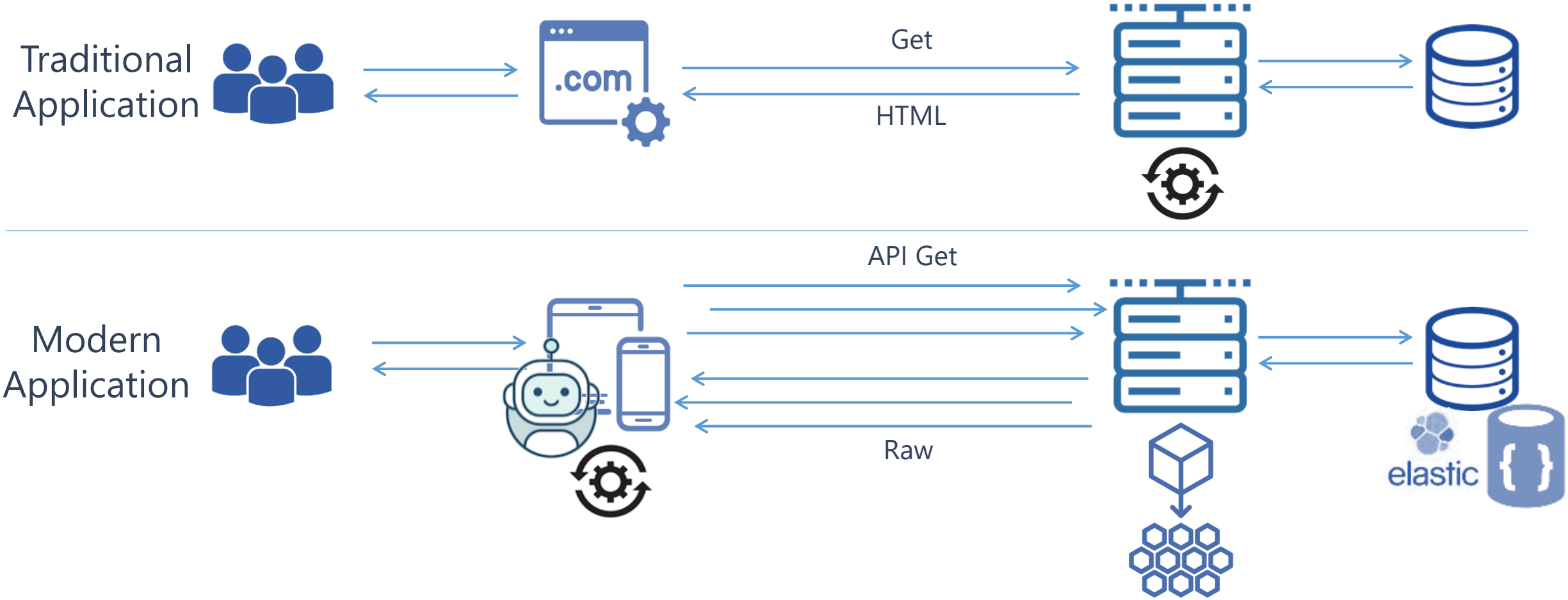
What Uses APIs?

- Microservices
- Mobile
- IoT
- B2B
- Serverless
- Cloud
- Single Page Application

Every Modern Application

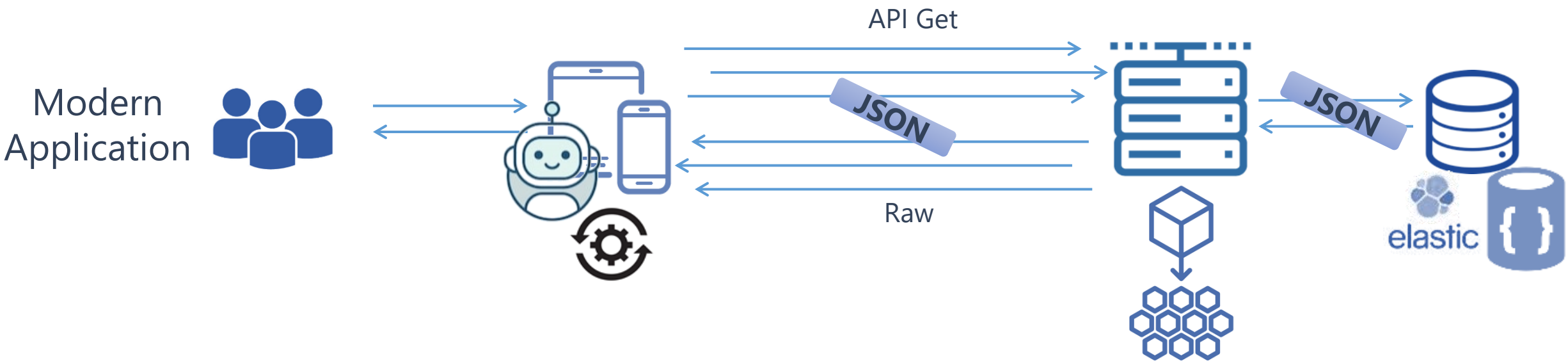
API Security

Traditional vs. Modern Applications



Traditional vs. Modern Applications

- Less abstraction layers
- Client and server (and DB) speak the same JSON language



Traditional vs. Modern Applications

The differences we see in Modern Apps

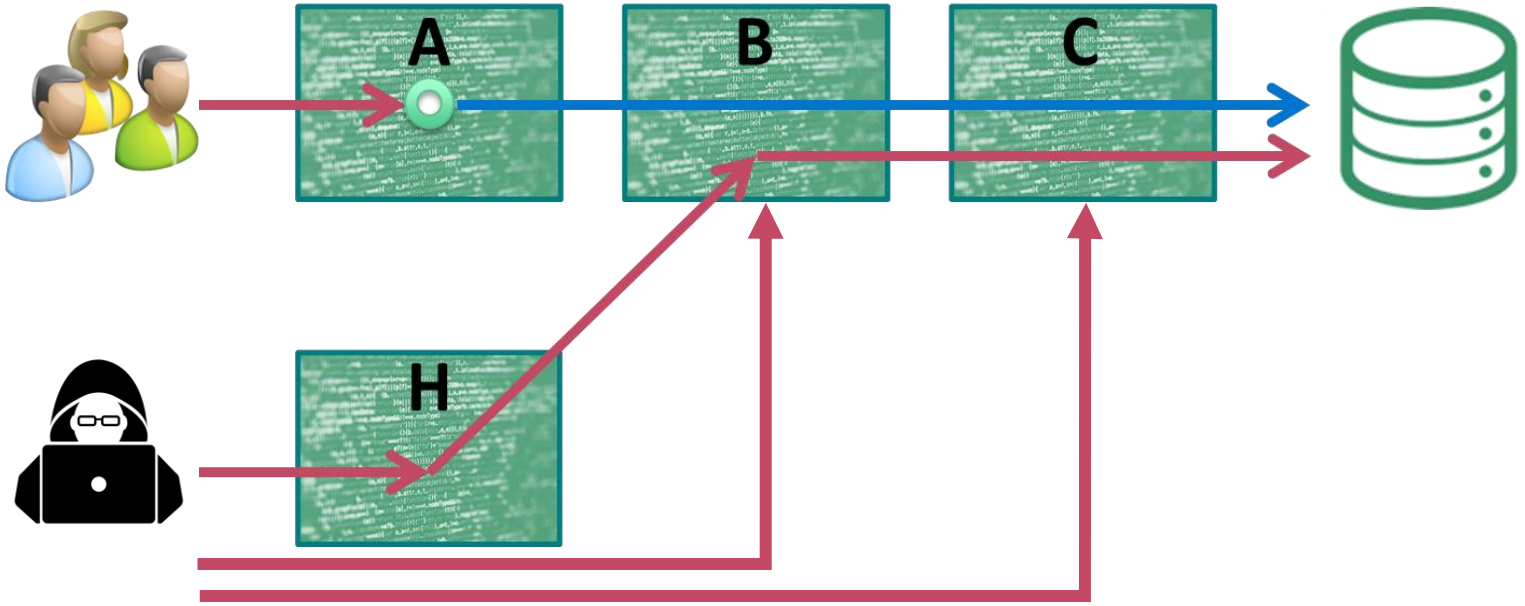
- The server is used more as a proxy for data
- The rendering component is the client, not the server
- The user's state is usually maintained and monitored by the client

- Clients consume raw data
- More parameters are sent in each HTTP request (object ID's, values, filters)

- APIs expose the underlying implementation of the app

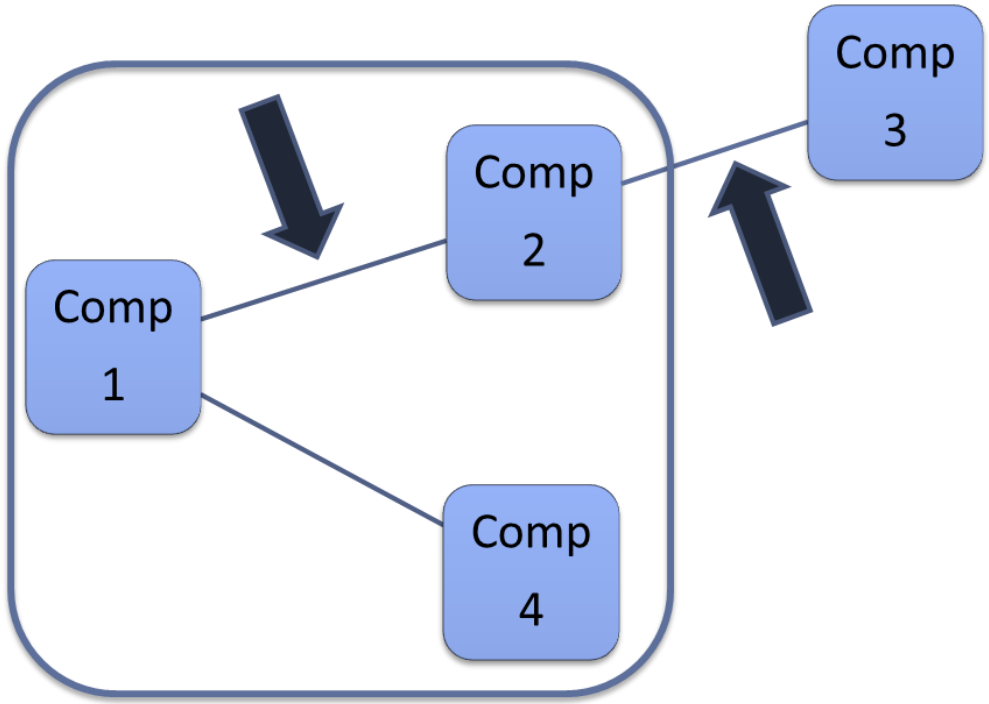
What Makes APIs Vulnerable?

1. The abundancy of API endpoints makes the attack surface bigger



What Makes APIs Vulnerable?

2. Clients consume raw data
   More parameters are sent in each HTTP request (object ID's, values, filters)



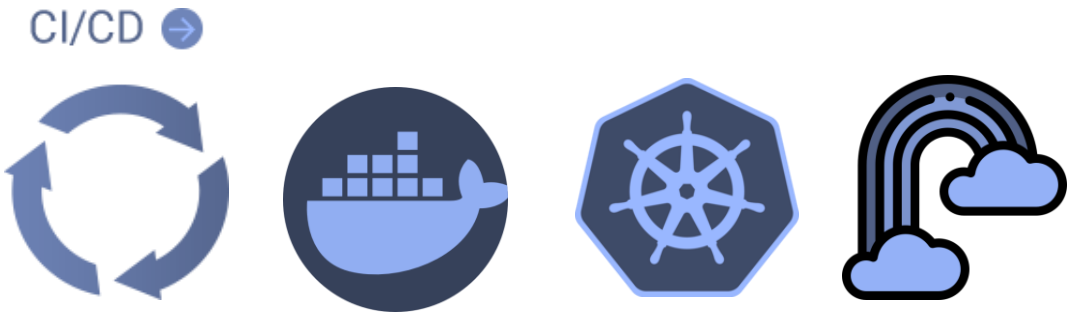
What Makes APIs Vulnerable?

3. The flexibility of CI/CD processes today, and the effortless deployment of new microservices, containers, and cloud infrastructure.
   It takes just a few clicks to spin up new APIs (hosts).

The rate of updates and changes in APIs may be too fast to handle.

APIs Become hard to track:

- **Shadow APIs**
- **Old Exposed APIs**



It’s Not All Bad News

- Traditional vulnerabilities are less common in API-based apps:
  - ↓ SQLi – due to increasing use of frameworks/ORMs
  - ↓ CSRF – due to authorization headers instead of cookies
  - ↓ Path Manipulations – due to cloud-based storage
  - ↓ Classic IT security issues - SaaS

Bridging The Gap

Bridging The Gap

Checkmarx

T10 OWASP Top 10 Application Security Risks – 2017

A1:2017-Injection	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2:2017-Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
A3:2017-Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
A4:2017-XML External Entities (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
A5:2017-Broken Access Control	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.
A6:2017-Security Misconfiguration	Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.
A7:2017-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A8:2017-Insecure Deserialization	Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
A9:2017-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
A10:2017-Insufficient Logging & Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

Bridging The Gap

OWASP API Security Project

Checkmarx

OWASP API Security Top 10

OWASP API Security Top 10

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API3: Excessive Data Exposure
- API4: Lack of Resources & Rate Limiting
- API5: Broken Function Level Authorization
- API6: Mass Assignment
- API7: Security Misconfiguration
- API8: Injection
- API9: Improper Assets Management
- API10: Insufficient Logging & Monitoring

OWASP Top 10

Checkmarx

API8 – Injection

Why drop from A1 to A8?

- “Injection” is #1 because of SQL Injections.
- SQL Injection are not very common in modern APIs, because:
  - Use of ORMs
  - Increasing use of NoSQL
- NoSQL injections are a thing, but are usually not as common / severe

OWASP Top 10 - 2017
A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE)
A5:2017-Broken Access Control
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

API10 - Insufficient Logging & Monitoring

Same as OWASP Top 10

Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident.

Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected.

Checkmarx

OWASP API Security Top 10

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API3: Excessive Data Exposure
- API4: Lack of Resources & Rate Limiting
- API5: Broken Function Level Authorization
- API6: Mass Assignment
- API7: Security Misconfiguration
- API8: Injection
- API9: Improper Assets Management
- API10: Insufficient Logging & Monitoring

OWASP Top 10

Access Control

Checkmarx

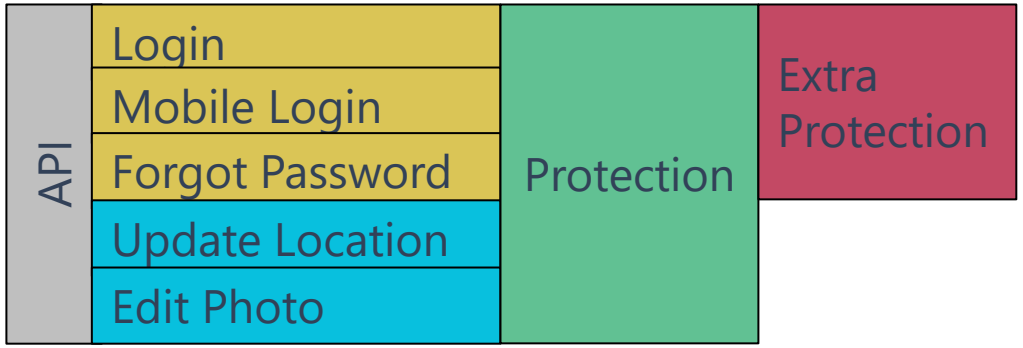
Access Control
API Security's Biggest Challenge

Access Control

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API5: Broken Function Level Authorization

API2: Broken Authentication

Lack of protection



- Captcha
- Account lockout mechanism
- Credentials Stuffing Protection

Misimplementation

- JWT Supports {“alg”:”none”}
- No validation of authentication provider
- Passwords stored without salt
- Etc…

API2: Broken Authentication

Why is it so common in APIs?

- Authentication endpoints are exposed to anyone by design.
- Software/security engineers have misconceptions.

  API keys should not be used for user's authentication

  Authorization != Authentication
- Multiple authentication flows in modern apps

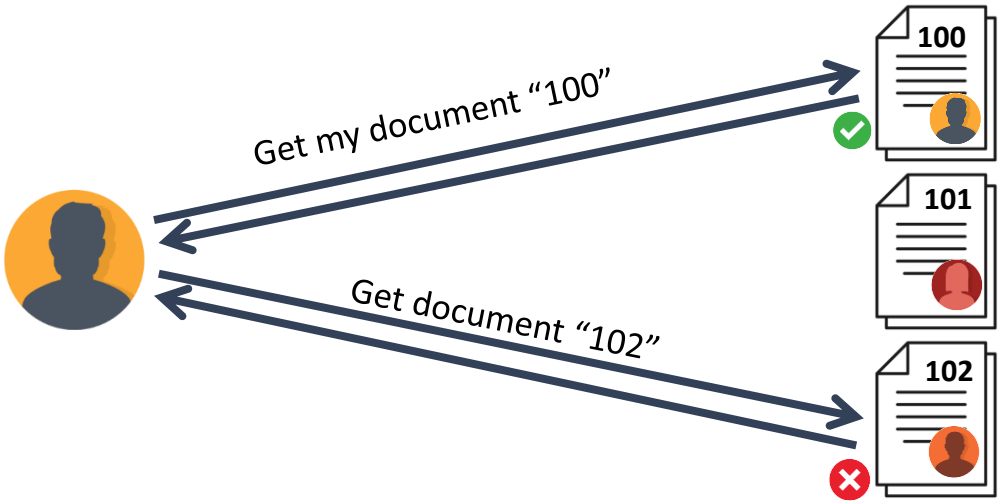
  IoT, Mobile, Legacy, Deep links with credentials

  etc…

Access Control

- **API1: Broken Object Level Authorization**
- API2: Broken Authentication
- API5: Broken Function Level Authorization

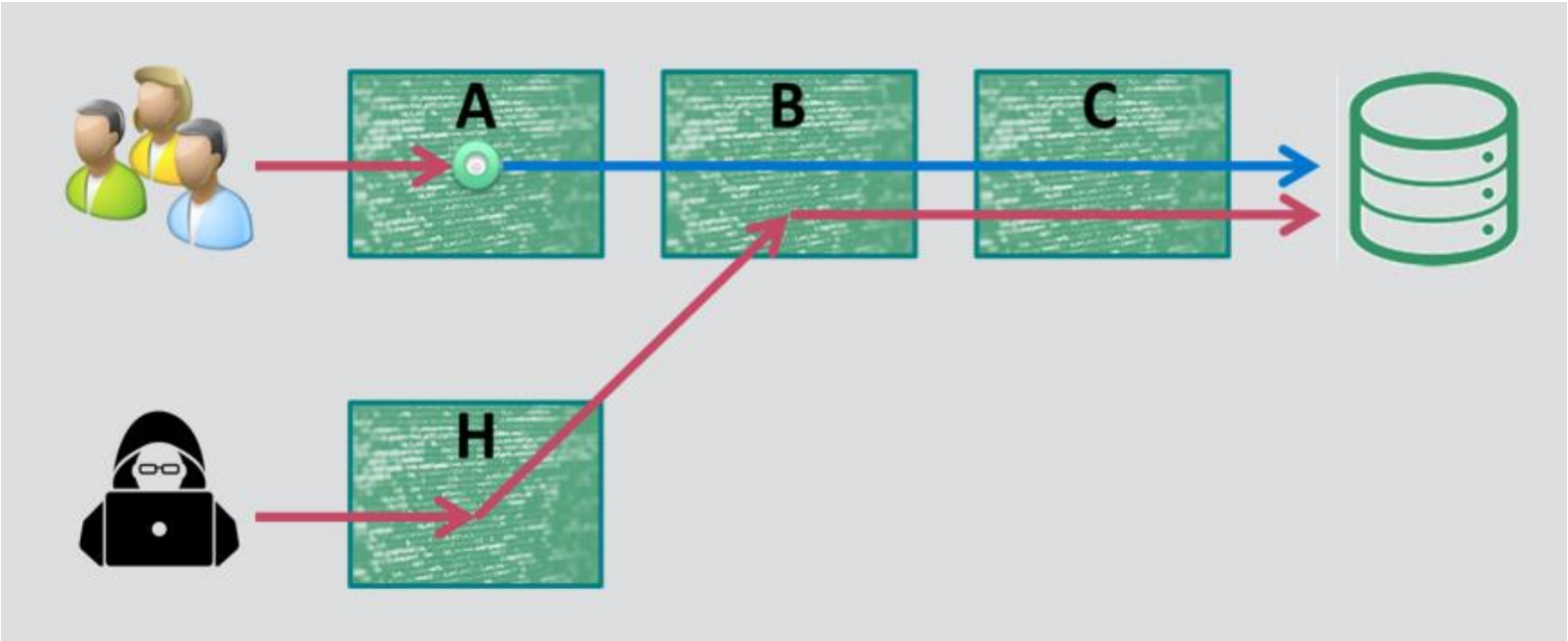
API1: Broken Object Level Authorization (BOLA)



API1: Broken Object Level Authorization (BOLA)

Why is it so common in APIs?

- The attack surface is much wider



- **No security solution exists that solves the problem**

API1: Broken Object Level Authorization (BOLA)

Why not “IDOR”? It's not accurate / indicative enough

- "IDOR" - **I**nsecure **D**irect **O**bject **R**eference
- "I**D**OR" implies that object reference should be indirect

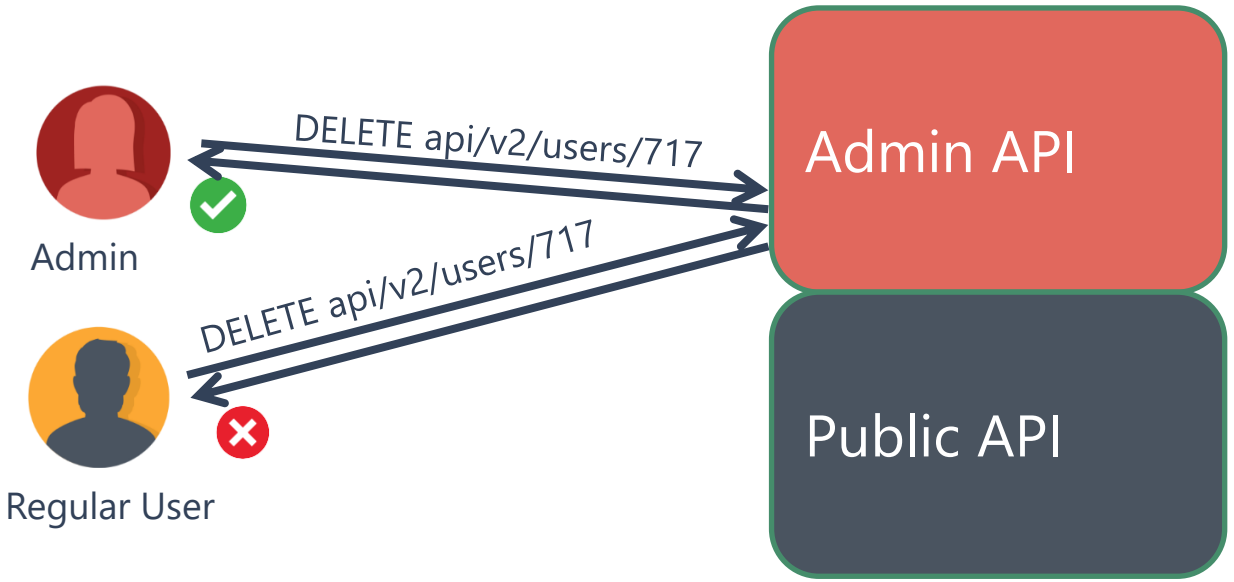
  (salted hash map / random string added to every ID)
- The problem is not the Object Reference, but a lack of authorization

What would happen if you asked your developers to implement “Indirect” mechanism in every place that receives ID?

Access Control

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API5: Broken Function Level Authorization

API5: Broken Function Level Authorization (BFLA)



API5: Broken Function Level Authorization (BFLA)

Why is it so common in APIs?

- Function Level Authorization can be implemented in different ways:

  Code, Configuration, API Gateway, etc.

- **Easier to detect and exploit in APIs – Endpoints are predictable**

	Get user's profile (Regular endpoint)	Delete user (Admin endpoint)
Traditional	GET /app/users_view.aspx?user_id=1337	POST app/admin_panel/users_mgmt.aspx? action=delete&user_id=1337
Modern	GET /api/v2/users/1337	DELETE /api/v2/users/1337

Hard to Predict

Very Easy to Predict

Checkmarx

Coursera API vulnerabilities disclosed by researchers

Coursera took “prompt ownership” of the bugs, once reported.

By Charlie Osborne for Zero Day | July 8, 2021 -- 13:00 GMT (14:00 BST) | Topic: Security

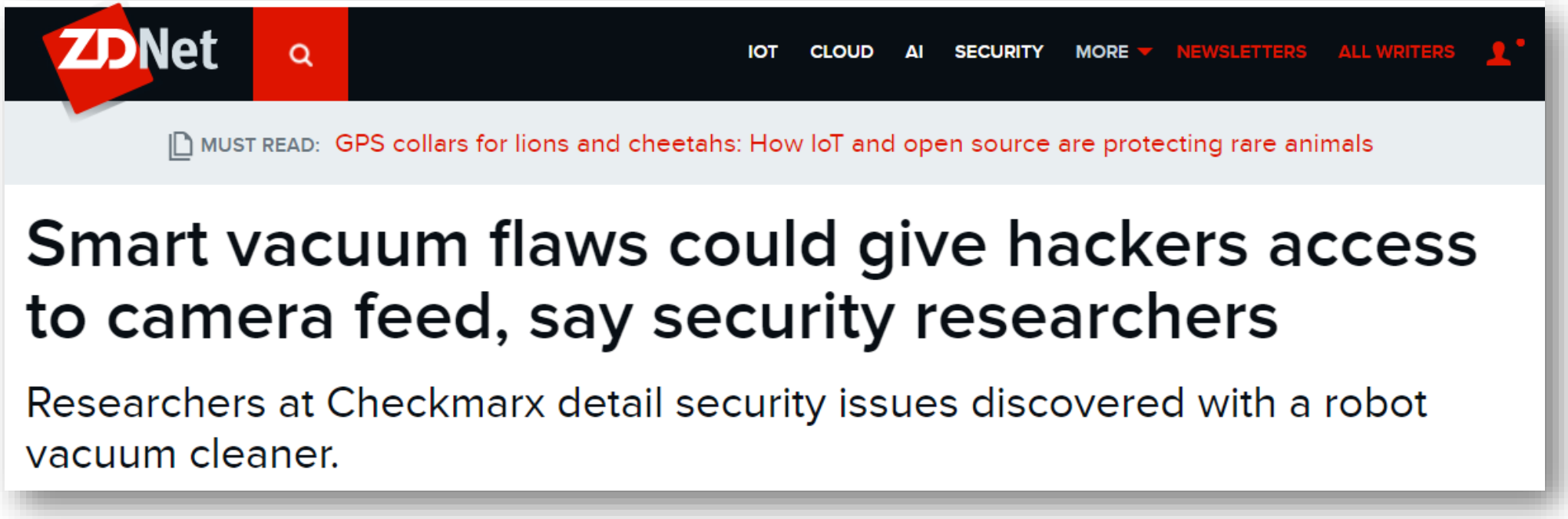
Researchers have disclosed a set of API vulnerabilities in the Coursera platform.

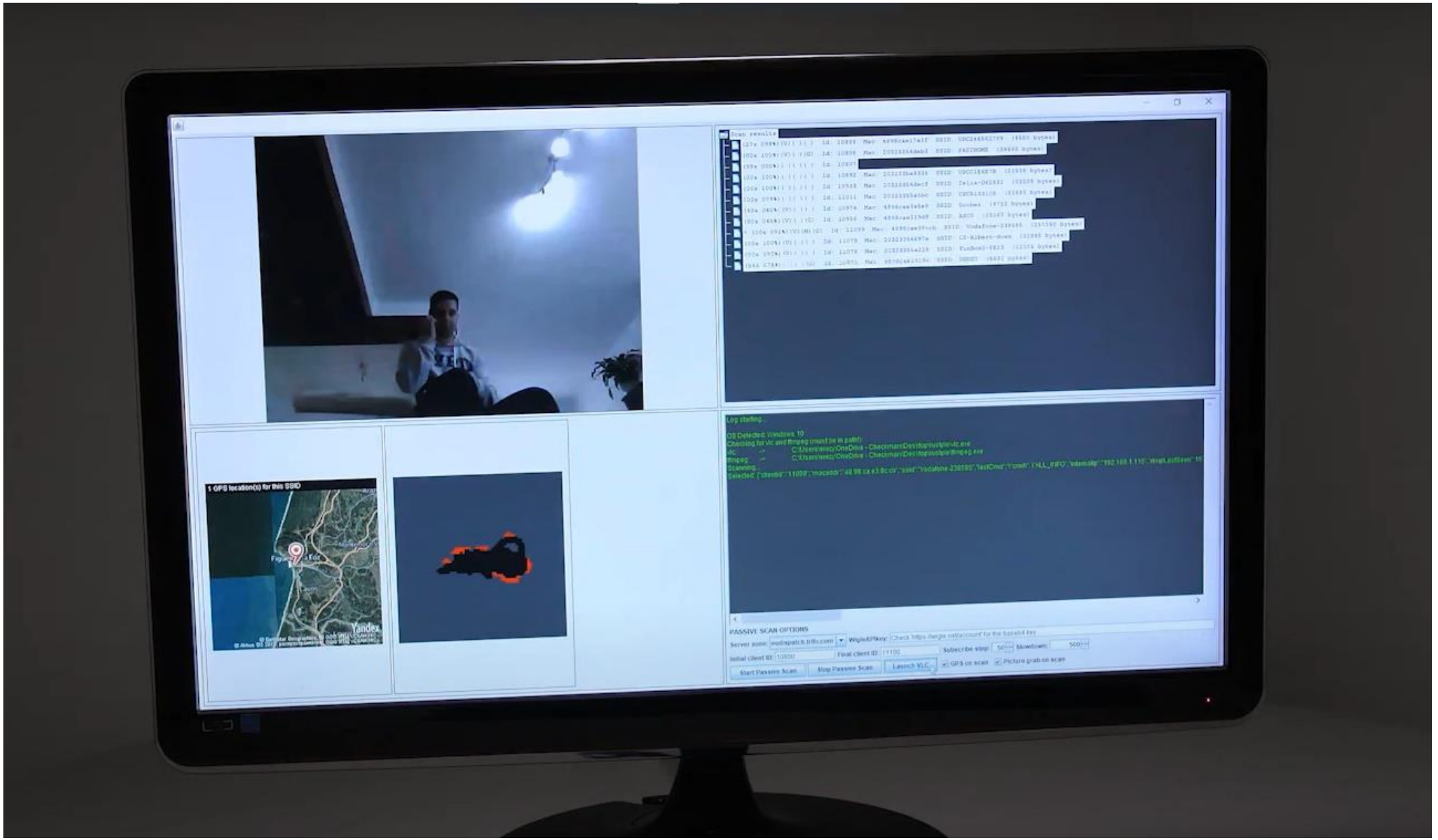
On Thursday, Checkmarx security researcher Paulo Silva revealed the discovery of multiple security failings in the Coursera online learning

KASEYA ATTACK

Kaseya ransomware supply chain attack: What you need to know

Access Control Vulnerability in Real Life




1 GPS location(s) for this SSID
Yandex
Log starting...
PASSIVE SCAN OPTIONS
Start Passive Scan
Stop Passive Scan
Launch VLC

OWASP API Security Top 10

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API3: Excessive Data Exposure
- API4: Lack of Resources & Rate Limiting
- API5: Broken Function Level Authorization
- API6: Mass Assignment
- API7: Security Misconfiguration
- API8: Injection
- API9: Improper Assets Management
- API10: Insufficient Logging & Monitoring

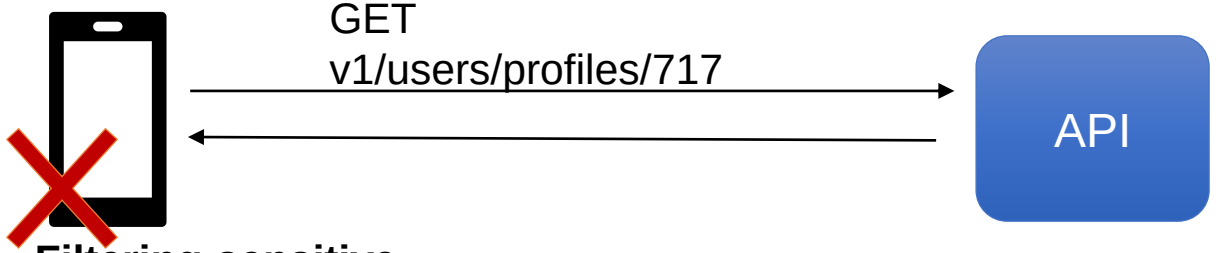
OWASP Top 10

Access Control

Excess Data

API3 – Excessive Data Exposure


```
200 OK
{
    "users": [{
            "profile_pic": "profiles/bob.jpg",
            "user_id": 717,
            "name": "Bob",
            "Role": "Minion",
            "Hobbies": ["Bananas"],
            "address": "Gru's Mansion, 1000 Evil Rd"
        }
    ]
}
```



Filtering sensitive data on the client side is always a bad idea

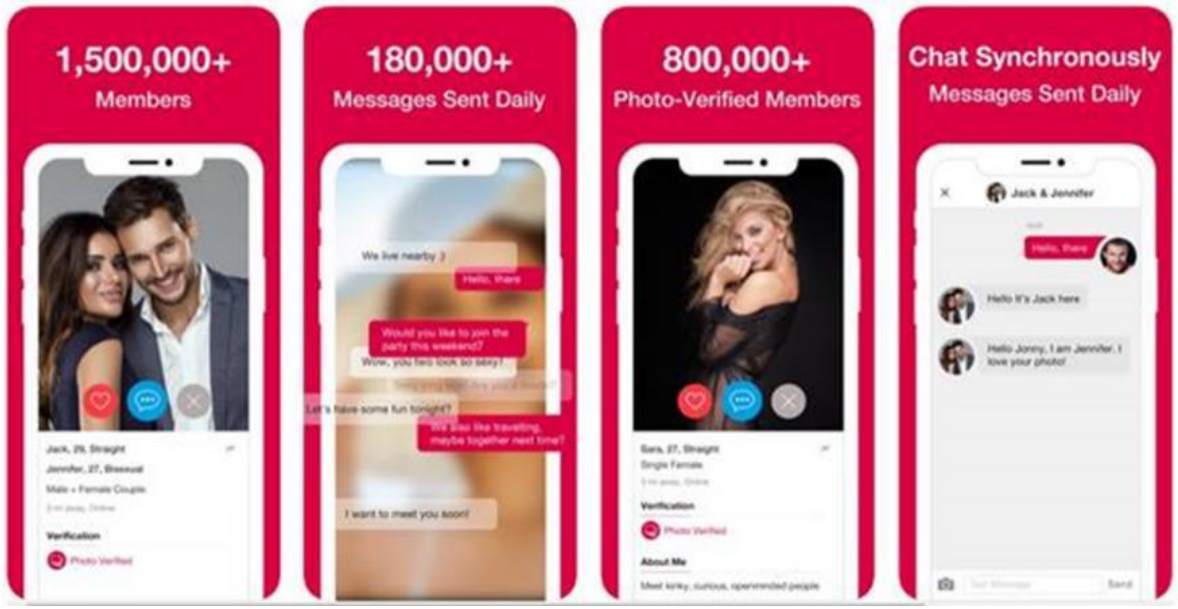
Checkmarx

API3 – Excessive Data Exposure

Why it is so common?

- REST Standards encourage developers to implement APIs in a generic way
- Use of generic functions as "to_json" from the Model / ORM, without thinking about who's the consumer

API3 - 3Fun Hack



Found by Alex Lomas, [Pen Test Partners]

#	Host	Method	URL	Params	Edited	Status	Length	MIME type
322	https://www.go3fun.co	POST	/account_kit_reg	✓		200	447	JSON
325	https://www.go3fun.co	POST	/user/device_token	✓		200	198	JSON
326	https://www.go3fun.co	POST	/user/update	✓		200	265	JSON
327	https://www.go3fun.co	POST	/reset_push_badge			200	198	JSON
329	https://www.go3fun.co	GET	/match_users?from=0&latitude=51.[redacted]	✓		200	23807	JSON
331	https://www.go3fun.co	GET	/user/refresh			200	788	JSON

Request | Response

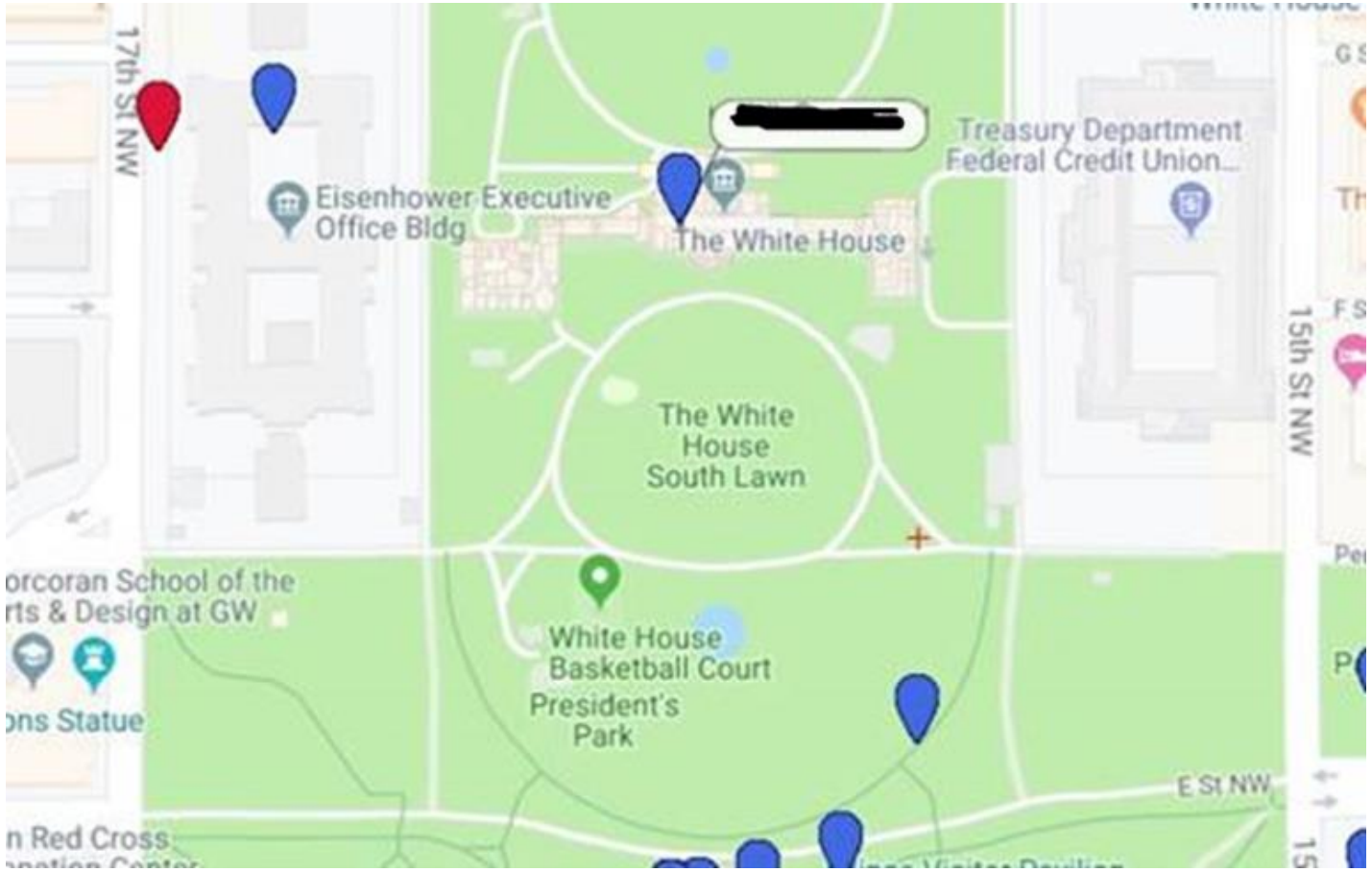
Raw | Headers | Hex | JSON Beautifier


```
    ],
    "latitude": "51.[redacted]",
    "membership": "2",
    "birthday": "1977-[redacted]",
    "sex_orient": "4",
    "gender": "1",
    "longitude": "-0.1[redacted]",
    "photo_verified_status": "1",
    "active": "0",
    "partner_sex_orient": "0",
    "liked_me": "0",
    "settings": {
      "show_online_status": "1",
      "show_distance": "1"
    },
    "username": "[redacted]",
    "usr_id": "17[redacted]",
    "about_me": "Kinky and attractive french financier open to many things ..."
  },
  {
    "last_login": "2019-06-24 20:21:12",
    "private_photos": [
      {
        "icon": "https://s3.amazonaws.com/3fun/821/[redacted]/[redacted]-small.jpg",
        "photo_id": "38[redacted]",
        "py": "500",
        "px": "750",
        "photo": "https://s3.amazonaws.com/3fun/821/[redacted]/[redacted]-big.jpg",
        "descr": null
      }
```


Found by Alex Lomas, [Pen Test Partners]

Checkmarx

API3 - 3Fun Hack



Found by Alex Lomas,

Pen Test Partners

API6 – Mass Assignment

Modern frameworks encourage developers to use “Mass Assignment” functions

NodeJS:

```
var user = new User(req.body);
user.save();
```

Rails:

```
@user = User.new(params[:user])
```

Legit Request

POST /api/users/new

{“username”:”Bob”, ”pass”:”123456”}

Malicious Request

POST /api/users/new

{“username”:”Bob”, ”pass”:”123456”, ”role”:”admin”}

Might contain sensitive params that the user should not have access to

OWASP API Security Top 10

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API3: Excessive Data Exposure
- API4: Lack of Resources & Rate Limiting
- API5: Broken Function Level Authorization
- API6: Mass Assignment
- API7: Security Misconfiguration
- API8: Injection
- API9: Improper Assets Management
- API10: Insufficient Logging & Monitoring

OWASP Top 10
Access Control
Excess Data
Dev(Sec)Ops

Checkmarx

API4 - Lack of Resources & Rate Limiting

When a resource (memory, CPU, DB, file, etc.) is exposed to the web, there should be defined use limit

- Requests
  - Number, Frequency
- Files
  - Size
- Strings
  - Length

API4 - Lack of Resources & Rate Limiting

Several consequences for not having a limit:

- DoS – Denial of Service
- Brute-force attacks
  - Credential Stuffing

API7 – Security Misconfiguration

- Weak encryption
- Unnecessary exposed HTTP methods
- CSRF protection turned off
- Detailed errors
- Improper CORS

API9 – Improper Assets Management

Two similar housekeeping Issues

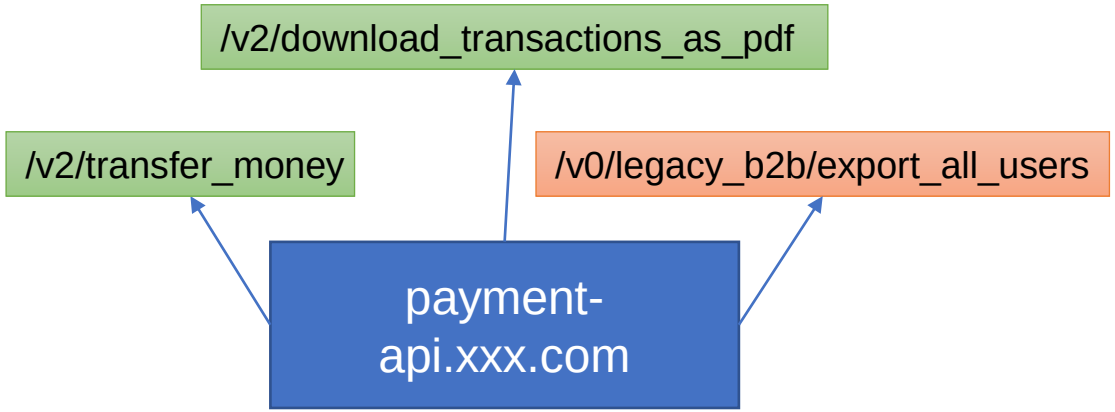
Lack of documentation

Exposed Risky APIs

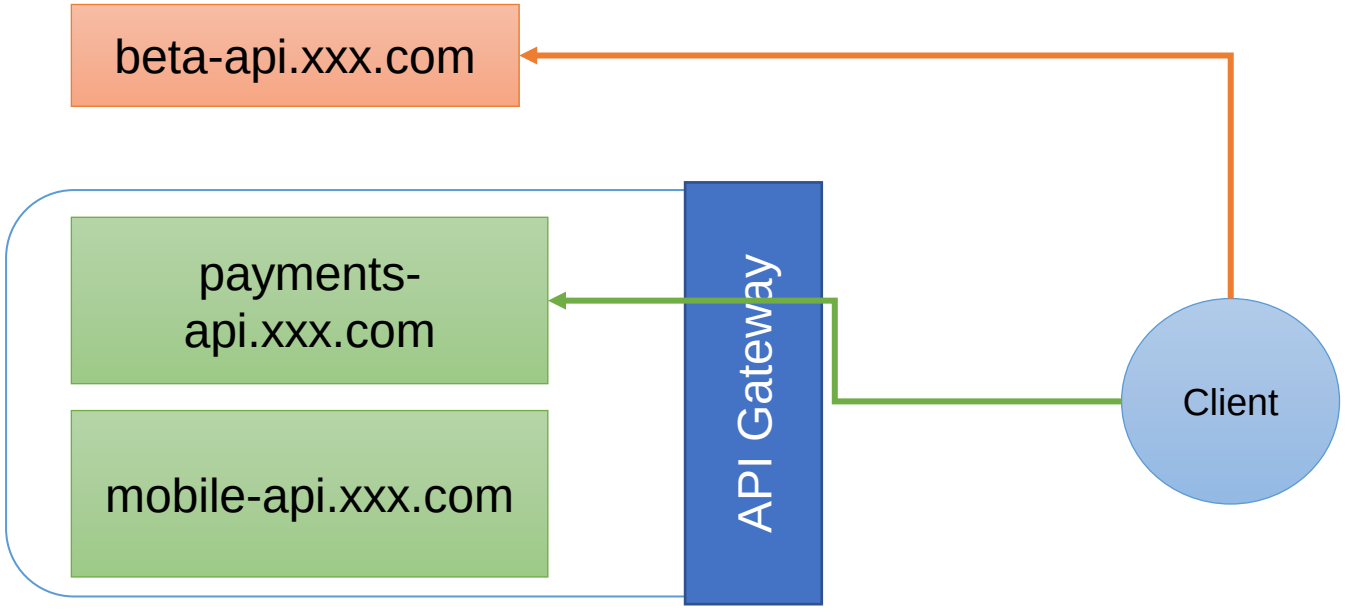
API9 – Improper Assets Management

Two similar housekeeping Issues

Lack of documentation



Exposed Risky APIs



Checkmarx

API9 – Improper Assets Management

Why is it such a big issue?

- APIs change all the time because of **CI/CD**. Developers are focused on delivering and not documenting

- With cloud & deployment automation it is way too easy to spin up new APIs and machines
  - API hosts that have been forgotten
  - Complete environments that have been forgotten
    (what the heck is qa-3-old.app.com ?)

OWASP API Security Top 10

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API3: Excessive Data Exposure
- API4: Lack of Resources & Rate Limiting
- API5: Broken Function Level Authorization
- API6: Mass Assignment
- API7: Security Misconfiguration
- API8: Injection
- API9: Improper Assets Management
- API10: Insufficient Logging & Monitoring

OWASP Top 10

Access Control

Excess Data

Dev(Sec)Ops

Checkmarx

Checkmarx

Summary

What You Need to Remember

- Modern API-based apps are different
- Being different, they have their own security issues
- The attack surface is much wider
- There is more data moving between components
- Access Control is a real challenge

/ Thank you

www.checkmarx.com