



13 December 2022

# API Breaches for H2 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](https://apisecurity.io)

Cyberbroof, Veracode, CA, Deutsche Bank



## Housekeeping Rules

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



# API Security Framework

Framing the discussion



## To err is human

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





# What is “API security”

API governance		API inventory	
API abuse cases	<b>OWASP API Security Top 10</b>		Business logic vulnerabilities
	OWASP API Security Top 10	OWASP Top 10	
	<b>A01:2019 Broken Object Level Authorization</b>	A01:2021-Broken Access Control	
	A02:2019 Broken User Authentication	A02:2021-Cryptographic Failures	
	<b>A03:2019 Excessive Data Exposure</b>	A03:2021-Injection	
	<b>A04:2019 Lack of Resources &amp; Rate Limiting</b>	A04:2021-Insecure Design	
	<b>A05:2019 Broken Function Level Authorization</b>	A05:2021-Security Misconfiguration	
	<b>A06:2019 Mass Assignment</b>	A06:2021-Vulnerable and Outdated Components	
	A07:2019 Security Misconfiguration	A07:2021-Identification and Authentication Failures	
	A08:2019 Injection	A08:2021-Software and Data Integrity Failures	
Coding flaws		Lack of skills	Secrets management
Vulnerable components/libraries		OS vulnerabilities	





# A layered approach to API security

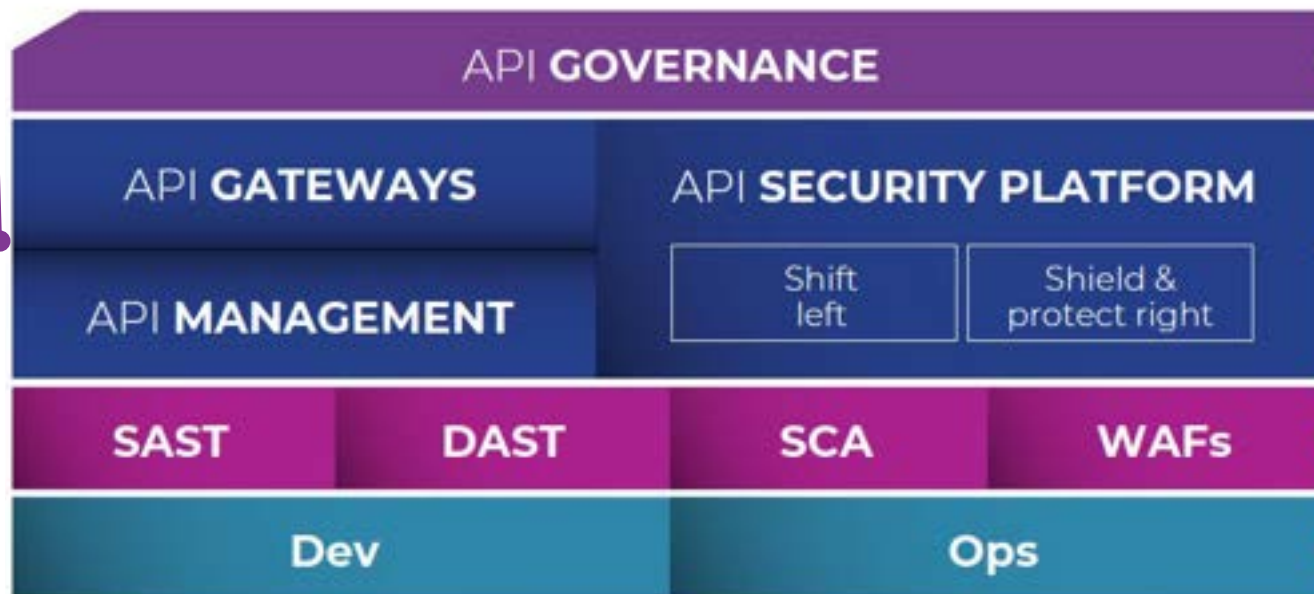
API management portals and gateways are essential to the operation of APIs at scale, and **security features should be leveraged to provide another layer of protection.**

A dedicated API Security platform is essential as the ultimate layer of defense allowing for:

- **Validation** of OAS specifications
- **Verification** of API implementation
- **Runtime protection** of APIs
- **API discovery** and inventory
- **Integration** into IDEs and CI/CD

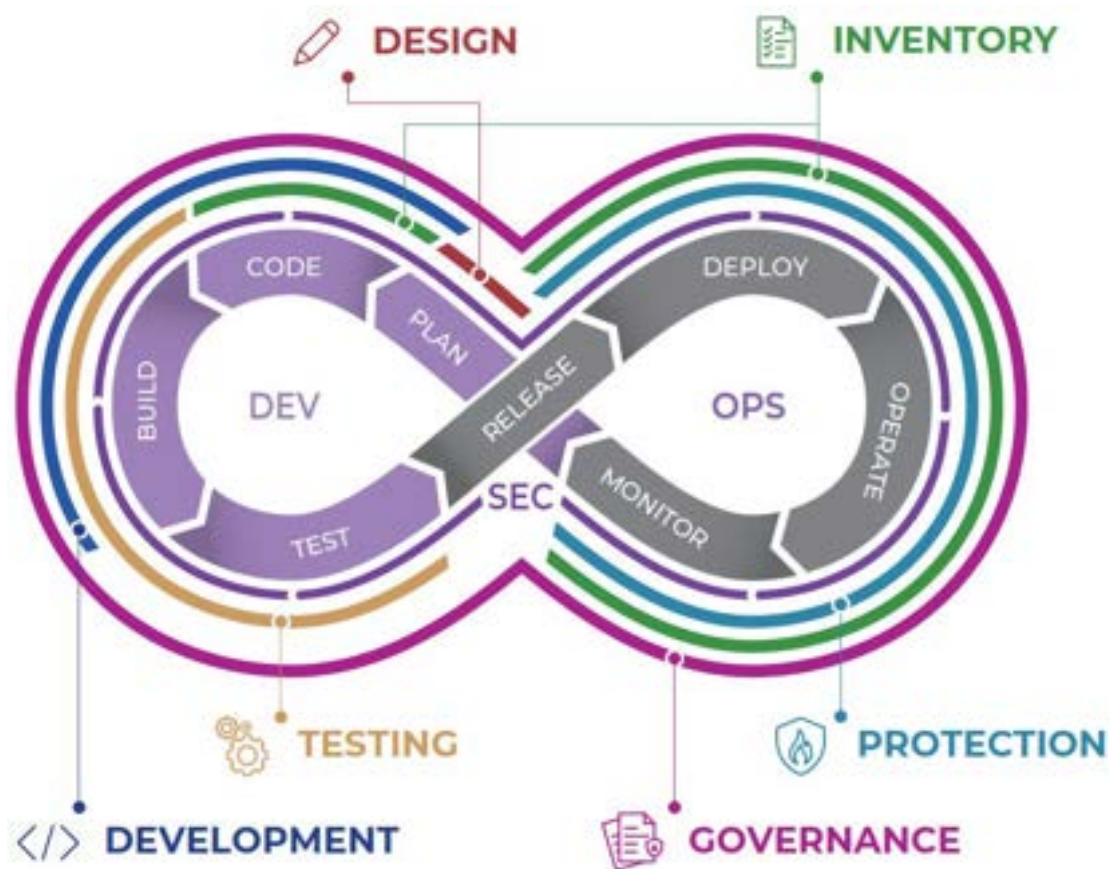
A well established AppSec process is vital to ensure that basic coding and implementation errors are detected in the build process. Traditional security tools (WAF, SAST, DAST) are not specifically tailored for API development and are likely to lead to **gaps in coverage.**

DevOps emphasizes collaboration between Development and Operations teams to ensure **high-velocity delivery of quality applications in a repeatable, automated manner.**





# The six pillars of API security



## API INVENTORY

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

## API TESTING

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

## API DESIGN

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

## API PROTECTION

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

## API DEVELOPMENT

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

## API GOVERNANCE

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





# Vulnerabilities in review

Our favorite vulnerabilities in the second-half of 2022



# #1: Australian telecommunications company

## What happened?

Attackers were able to discover a publicly exposed endpoint which also did not require any authentication. This API exposed highly sensitive customer information, including driver's license numbers, phone numbers, dates of birth, and home addresses. The API also used sequential identifiers, allowing easy enumeration.

## Impact:

Massive reputational and financial loss to the company. Loss of PII data of 10 million customers (including passport and driver's license details). Possibly up to A\$ 140 million in breach costs.

## Cause:

- Lack of authentication.
- Excessive information exposure.
- Easily guessable object identifiers.

## Lessons learned:

- Numerous and painful.



# #1: Australian telecommunications company

## Design:

- Failure to under PII requirements - leaked customer details
- Failed to capture any security requirements
- Relied on obscurity - hoping the API would not be discovered

## Development:

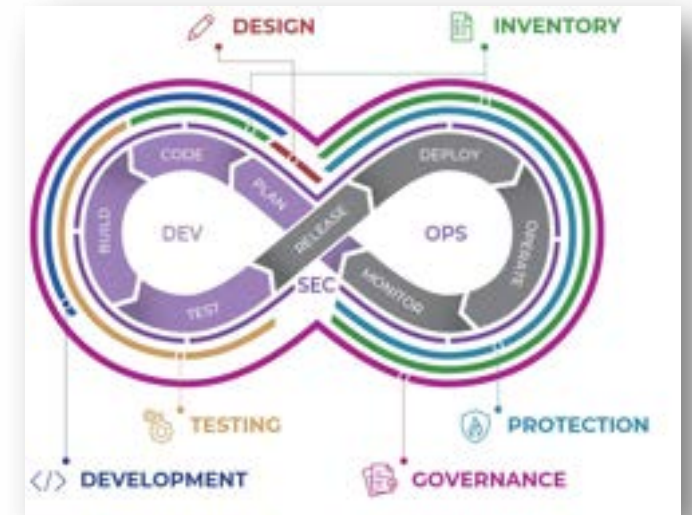
- Very basic security or privacy implementation errors

## Testing:

- Did not detect leakage of customer information
- Did not detect unauthenticated endpoint

## Governance:

- Failure to prevent deployment of totally insecure API
- No control of sensitive customer information





## #2: Popular social media platform

### What happened?

Cybercriminals traded user data of over 5.4 million users in a hacking forum. Attack had been leveraged over a period of a year and resulted from abuse of a “search users” API endpoint.

### Impact:

Cybercriminals were able to perform spear phishing attacks on users known to have accounts. Reputational damage.

### Cause:

- Exposing “helpful” APIs that can be abused.
- Excessive information exposure (returning inconsistent error messages).

### Lessons learned:

- Beware of unauthenticated APIs, regardless of how benign they seem.
- Use a consistent minimal implementation of an error handler.



## #2: Popular social media platform

### Design:

- Failure to consider the implications of providing “search users” function.
- Failure to protect sensitive user data.

### Development:

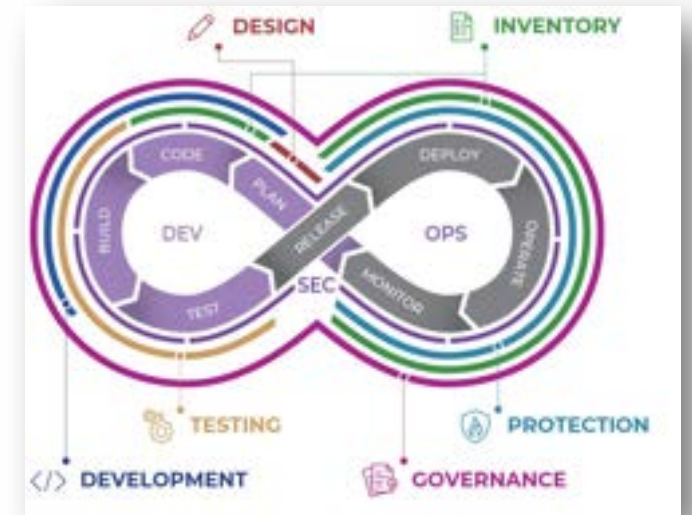
- Excessive information exposure in error handler.

### Testing:

- Did not detect leakage of user information.

### Protection:

- Lack of rate limiting on public API allowing mass exfiltration.







## #3: Crypto trading platform(s)

### What happened?

Seemingly a rogue cryptocurrency bot could perform high volumes of automated trades on various platforms, resulting in unconfirmed losses and a cessation of withdrawals in some cases.

### Impact:

Although unconfirmed, the financial impact could be massive. Loss of investor faith in the cryptocurrency market, scrutiny of regulators.

### Cause:


- Leakage of 3<sup>rd</sup> party keys.
- Theft of 3<sup>rd</sup> party keys.

### Lessons learned:

- Be wary of delegated key access.
- Review key access frequently, and revoke unused keys.




## #3: Crypto trading platform(s)


 **Rodion Longa**  
@LongaRodion · Follow

@cz\_binance @BinanceRussian My account was just exploited using 3commas API leak similar to this case [twitter.com/WuBlockchain/s...](https://twitter.com/WuBlockchain/s...)


Please help. 450k busd lost

 **Wu Blockchain** @WuBlockchain

A new method of stealing coins is emerging: contra trade. On October 19th, a user suddenly found that his FTX account using the 3commas API was trading DMG more than 5,000 times, stealing nearly \$1.6 million such as BTC, ETH, FTT, etc. from his account.



4:18 AM · Dec 9, 2022

 [Read the full conversation on Twitter](#)

175 Reply Copy link

[Read 30 replies](#)



Market News News Other

# Update on investigation into API keys and attacks on exchanges

From  Yuriy Sorokin

DATE PUBLISHED: DEC 10, 2022 · 22 MIN

DATE UPDATED: DEC 11, 2022

Response to frequent questions from our Twitter community



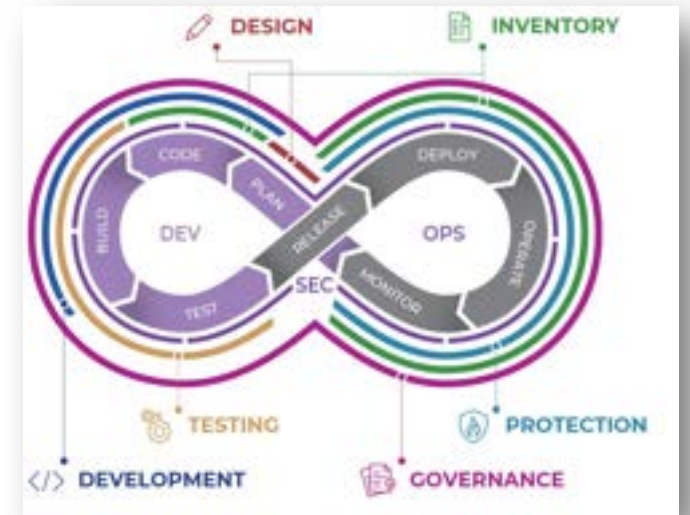
## #3: Crypto trading platform(s)

### Design:

- Possibly a failure to implement a robust token management policy.

### Governance:

- Failure to manage 3<sup>rd</sup> party tokens from theft by internal resources.
- Lack of audit of token use.





## #4: Helpdesk ticketing system

### What happened?

Researchers discovered that an internal GraphQL API endpoint was sensitive to SQL Injection attacks. They demonstrated a proof-of-concept that showed a leakage of user data via this injection.

### Impact:

Minimal to responsible disclosure and rapid patching.

### Cause:

- SQL Injection – 21 years and counting !

### Lessons learned:

- Sanitize all external input in the context of your data sink.
- GraphQL is complex – make sure you understand the security implications.



## #4: Helpdesk ticketing system

### Design:

- GraphQL SQL injection threats could have been identified earlier.

### Development:

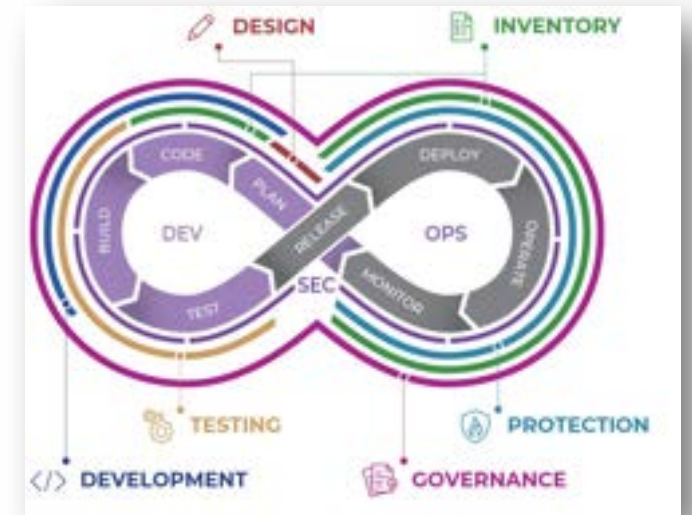
- Failure to use standard SQL injection protections.

### Testing:

- SQLi vulnerabilities can usually be detected with:
  - Penetration testing
  - Static code analysis

### Protection:

- SQL injection payloads can usually be easily filtered with an application-level firewall.







## #5: File hosting service

### What happened?

A file storage service had 150 of its internal GitHub repositories accessed by an attacker using phished API credentials.

### Impact:

Apparently limited since the repositories contained mostly clones of public domain libraries. No customer data was compromised.

### Cause:

- Phishing attack on developers via CI/CD reset email.

### Lessons learned:

- Beware of phishing emails (particularly unprompted password reset emails).
- API tokens are a key part of the supply chain.



## #5: File hosting service

### Design:

- Possibly a failure to implement a robust token management policy.

### Development:

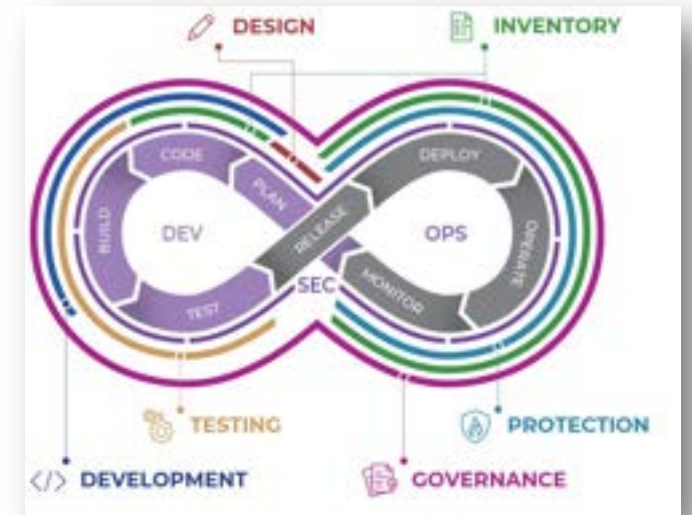
- Failed to detect a phishing attack.

### Governance:

- Failure to manage 3<sup>rd</sup> party tokens from theft by internal resources.
- Lack of audit of token use.

### Protection:

- GitHub did an exemplary job of detecting this API credential abuse.





## #6 Popular social media platform (again)

### What happened?

Security researchers disclosed the details of 3,200 mobile applications potentially leaking the platform's API keys. In most cases keys had been embedded into consumer applications.

### Impact:

No known abuse or impact, but this attack could allow massively distributed bot activity against the platform.

### Cause:

- Hard-coded keys.
- Lack of robust key management policy.

### Lessons learned:

- Never hardcode API keys.
- As a provider, be able to detect and revoke stolen keys.



## #6 Popular social media platform (again)

### Design:

- Failure to design adequate local key handling policy.

### Development:

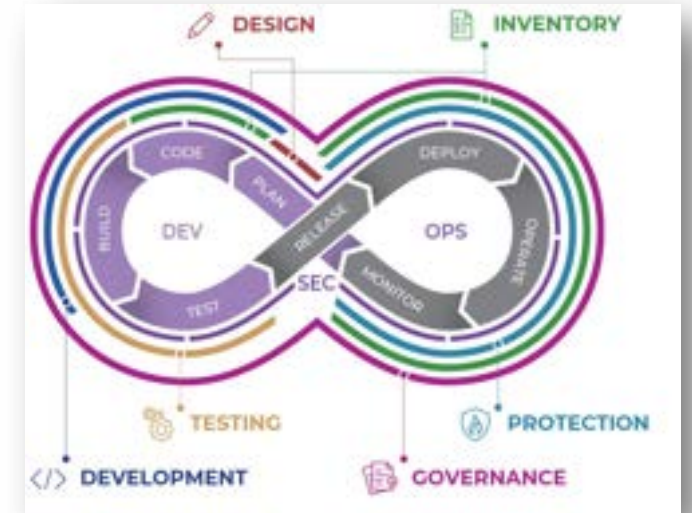
- Developers should have been aware of risks of hardcoded credentials.

### Testing:

- Code scanning would have detected hard-coded credentials easily.

### Protection:

- Platform could have detected use of leaked keys and blocked access.





## #7: Data center management platform

### What happened?

Researchers discovered that they could perform a total compromise of a popular data center management platform by stealing the web interface token and use CSRF methods to attack the underlying API.

### Impact:

Limited in this case due to responsible disclosure and rapid remediation by vendor.

### Cause:

- API that was vulnerable to CSRF based attacks.
- Very skilled attackers ;)

### Lessons learned:

- Beware of the dangers lurking in your (not-so) hidden APIs.
- Make sure you understand the basics of web application request forgery protection.





## Cookieless CSRF #2 – add DB user

### Cookieless CSRF #3 – add FTP user

We managed to add a new FTP user with access to any existing domain we wanted

```

1 POST /api/v2/firmware HTTP/1.1
2 Host: 84.247.23.290:8443
3 Cookie: sat=id=004A.D062.01F6.AF94.BE05.B78D.A212.5463; PL20K00002=
    mcmwsm; fcdm=2f1d07fae7c9caef; sat=sid=005B
4 Content-Length: 596
5 Authorization: Basic YWVkaW9kbWludG9ybWZlIGRpdmlwcm
6 Sec-CH-UA: "Chromium";v="8", "Chrome";v="101"
7 Accept: application/json
8 Content-Type: application/json
9 Sec-CH-UA-Mobile: ?0
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/101.0.5050.56 Safari/537.36
11 Sec-CH-UA-Platform: "linux"
12 Origin: https://84.247.23.290:8443
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer: https://84.247.23.290:8443/api/v2/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9,eng=0.8
19 Connection: close
20
21 {
22   "name": "fortibridge",
23   "password": "changemeIQHW",
24   "home": "/fortibridge",
25   "quota": 1,
26   "permissions": [
27     "read"/bin",
28     "write"/true"
29 ]
30 }

```



## #7: Data center management platform

And that is exactly what we needed. We ended up compromising Plesk via its REST API (CSRF) and change the Admin's password. Game over!

```
Request
Pretty Raw Hex
1 POST /api/v2/cli/admin/call HTTP/1.1
2 Host: 84.247.29.236:8443
3 Content-Length: 72
4 Cache-Control: max-age=0
5 Authorization: Basic YWpPc2V2cllkaW50JyMkRmYkF0dXVw
6 Sec-Ch-Ua: "Not(A:Brand)";v="8", "Chromium";v="105"
7 Sec-Ch-Ua-Mobile: 10
8 Sec-Ch-Ua-Platform: "Linux"
9 Upgrade-Insecure-Requests: 1
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.54 Safari/537.36
11 Origin: null
12 Content-Type: text/plain
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site: cross-site
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-US,en-US;q=0.9,en;q=0.8
20 Connection: close
21
22 {
23   "param": {
24     "...set-admin-password",
25     "password",
26     "wP0rT0r1203198"
27   }
28 }
```

```
Response
Pretty Raw Hex
1 HTTP/1.1 200 OK
2 Server: sw-cp-server
3 Date: Tue, 17 May 2022 08:03:55 GMT
4 Content-Type: application/json
5 Connection: close
6 Access-Control-Allow-Origin: *
7 Access-Control-Allow-Methods: *
8 Access-Control-Allow-Headers: *
9 Content-Length: 113
10
11 {
12   "code": 0,
13   "stdout": "The Plesk administrator's password was successfully updated.",
14   "stderr": ""
15 }
```



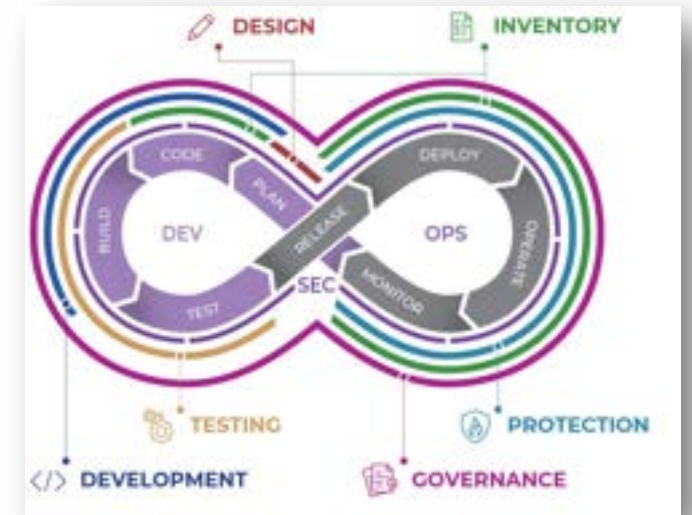
## #7: Data center management platform

### Design:

- Failure to adequately understand the risk of a CSRF-based attack.
- In-depth threat modeling may have identified risks.

### Development:

- Developers could benefit from advanced web security training.
- Failure to implement standard controls correctly i.e. CORS





## #8: Load balancing appliance

### What happened?

Researchers discovered that they could compromise a popular load-balancing appliance by stealing the web interface token and use CSRF methods to attack the underlying API.

### Impact:

Potentially serious due to the widespread use and impact. No reports of a successful compromise as yet.

### Cause:

- API that was vulnerable to CSRF based attacks.

### Lessons learned:

- Beware of the dangers lurking in your (not-so) hidden APIs.
- Make sure you understand the basics of web application request forgery protection.



## #8: Load balancing appliance

### Check in exploit script for CVE-2022-41622 (CSRF into SOAP) #

**Merged** space-r7 merged 2 commits into rapid7:master from rbowes-r7:f5-csrf-cve-2022-41622 25 days ago

Conversation 9 Commits 2 Checks 23 Files changed 2



rbowes-r7 commented 27 days ago

Contributor ...

Check in a module for CVE-2022-41622, which is a CSRF "write a file anywhere" vulnerability that can lead to persistent RCE as root.

Everything is tested against F5 Big-IP 17.0.0.1, which is freely available as a VM image (with a free account). It's also in our "vulnerable software" directory.

#### Verification

This is a CSRF vuln, so it requires a browser in addition to msf:

1. Install the application
2. Start `msfconsole`
3. Do: use `exploit/linux/http/f5_icontrol_soap_csrf_rce_cve_2022_41622`
4. Do `set TARGET_HOST <target> / set LHOST <yourtest>`
5. Do: run
6. You should get a url such as: `http://10.0.0.179:8080/ddgjZ0`
7. Open a browser and visit that URL
8. If you don't already have an HTTP Basic session, it'll ask for your credentials (the `admin` account from earlier works great)





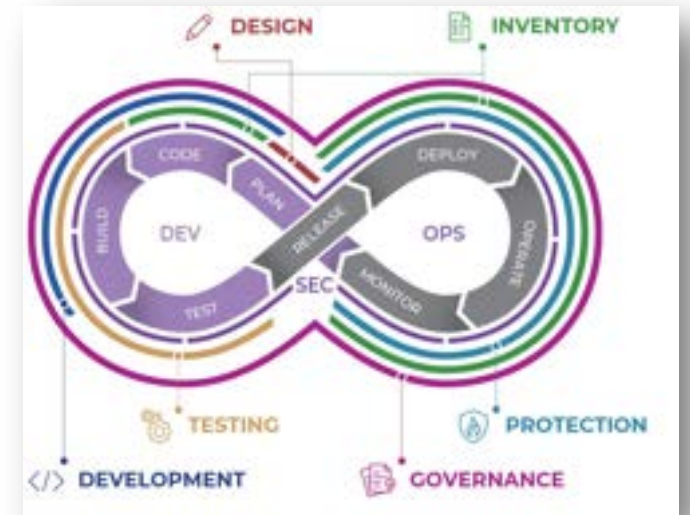
## #8: Load balancing appliance

### Design:

- Failure to adequately understand the risk of a CSRF-based attack.
- In-depth threat modeling may have identified risks.

### Development:

- Developers could benefit from advanced web security training.
- Failure to implement standard controls correctly i.e. CORS





## #9: Community platform

### What happened?

Researchers discovered that they could access the logs of other forum groups by manipulation the request object identifier.

### Impact:

Low due to the relatively insensitive data, and rapid remediation.

### Cause:

- Broken object-level authorization

### Lessons learned:

- Never trust user supplied parameters.
- Always fully validate access rights to requested resources.



## #9: Community platform

Code 510 Bytes

[Wrap lines](#) [Copy](#) [Download](#)

```
1 POST / HTTP/2
2 Host: ggl.███.███.com
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/json
8 Content-Length: 62
9 X-███-Compression: 1
10 Origin: https://www.███.███.com
11 Sec-Fetch-Dest: empty
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Site: same-site
14 Authorization: Bearer ourtoken
15 Referer: https://www.███.███.com/
16 Te: trailers
17
18 {"id":"6243efcbc6ld","variables":{"███ Name":"any-███ it"}}
```



## #9: Community platform

### Design:

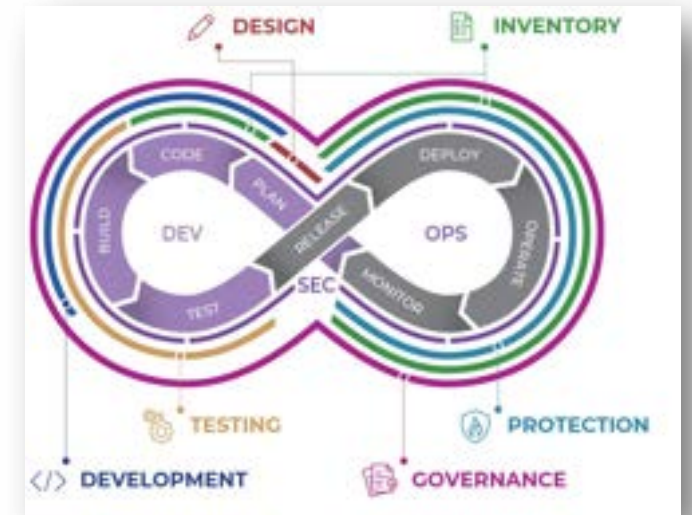
- Threat modeling and/or code review may have identified this weakness.

### Development:

- Developers could benefit from API security training.
- Always consider the origin of input parameters.

### Testing:

- Simple cases of BOLA (such as this) can be identified with automated testing





## #10: AI search tool

### What happened?

Security researchers disclosed the details of 3,200 mobile applications potentially leaking the platform's API keys. In most cases, keys had been embedded into consumer applications.

### Impact:

Limited with no known exploits.

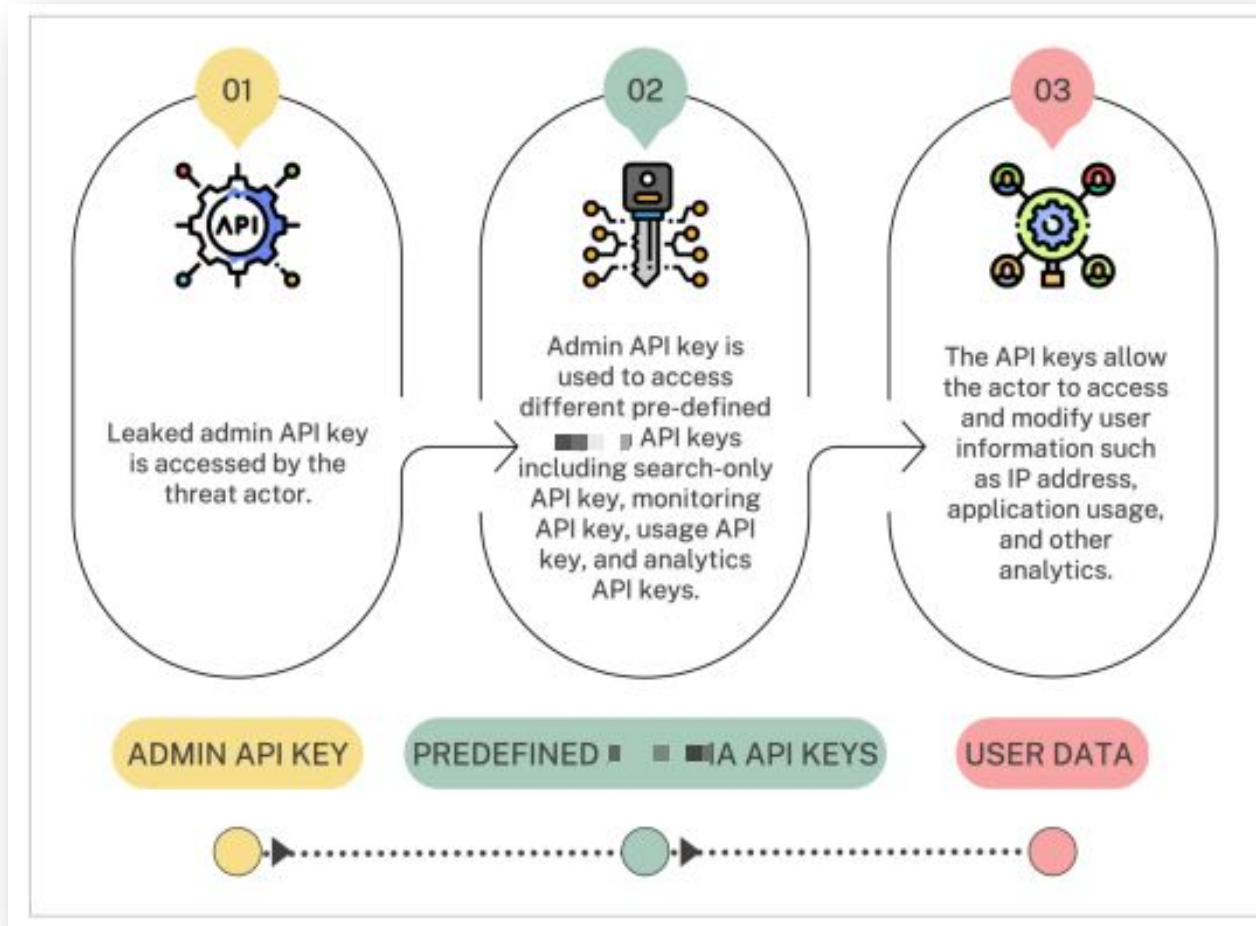
### Cause:

- Hard-coded keys.
- Lack of robust key management policy.

### Lessons learned:

- Never hardcode API keys.
- As a provider, be able to detect and revoke stolen keys.

## #10: AI search tool



App	No. of Downloads
Shopping app	2306000
News & Magazines app	300000
Food & Drink app	200000
Education app	110000
Health & Fitness app	105000
Photography app	100000
Lifestyle app	80000
Business app	11000
Entertainment app	11000
Medical app	10000
Parenting app	10000
Books & Reference app	5000
Productivity app	5000
Tool app	5000
Auto & Vehicles app	1000



# #10: AI search tool

## Design:

- Failure to design adequate local key handling policy.
- Failure to scope token access adequately.

## Development:

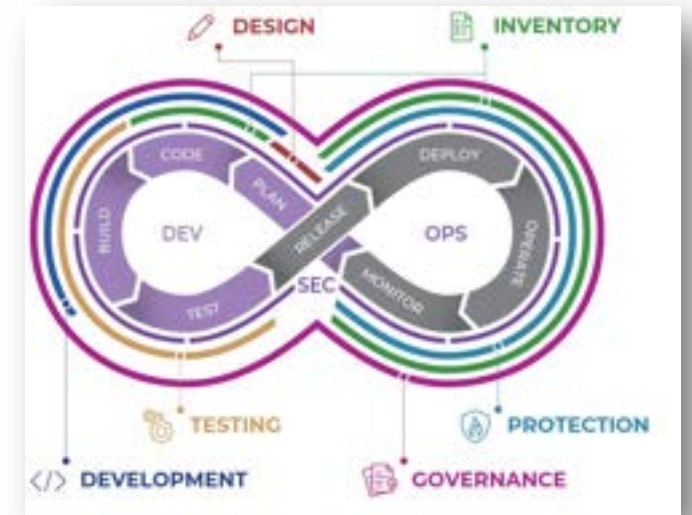
- Developers should have been aware of risks of hardcoded credentials.
- Provide a dedicated API endpoint for sensitive operations.
- Use a tightly scoped token to access this dedicated endpoint (avoiding the use of global scope admin tokens)

## Testing:

- Code scanning would have detected hard-code credentials easily.

## Protection:

- Platform could have detected use of leaked keys and blocked access.







# #11: Two popular vehicles

## What happened?

A security researcher discovered who could access vehicle computers on two popular brands of vehicles and perform remote commands such as sounding the horn or starting the vehicle!

## Impact:

Fortunately, in this case, very limited but unimaginable if exploited by a bad actor.

## Cause:

- Insufficient input sanitization.
- Poor regular expression design.

## Lessons learned:

- Attacks on vehicles can have dramatic consequences.
- Critical controls such as vehicle ignition should be barred from remote access perhaps?



# #11: Two popular vehicles

Request

1 POST /ac/v2/crm/rhl/11q8 HTTP/1.1  
2 Host: api.telematic1.com  
3 Accept: application/json  
4 AppType: 1  
5 AppVersion: 5.0.7  
6 ContentType: ANDROID  
7 Connection: close  
8 DeviceId: [REDACTED]  
9 DeviceName: lives  
10 DeviceModel: [REDACTED]  
11 FeatureUsageTime: 2022-10-01T01:55:11Z  
12 RequestId: 35  
13 Location: [REDACTED]  
14 From: SPA  
15 Language: 0  
16 ModId: [REDACTED]  
17 Offset: -7  
18 To: ISS  
19 Version: 1.4.7  
20 ClientId: [REDACTED]  
21 Vin: FKHLS  
22 Appcloud-Vin: FKHLS  
23 StandIndicator: 0  
24 Gen: 2  
25 RegistrationId: [REDACTED]  
26 EncryptFlag: false  
27 DeviceName: [REDACTED]  
28 AccessToken: [REDACTED]  
29 Content-Type: application/json  
30 Content-Length: 62  
31 Accept-Encoding: gzip, deflate  
32 User-Agent: curl/7.14.0  
33 Connection: close  
34 {"username": "[REDACTED]@gmail.com", "vin": "FKHLS"}  
35

Response

1 HTTP/1.1 200 OK  
2 Date: Sat, 01 Oct 2022 05:59:48 GMT  
3 Server: Oracle-Application-Server-11g  
4 Access-Control-Allow-Methods: POST, GET, PUT,  
5 Access-Control-Allow-Headers: language, offset, UCID, From,  
6 s-requested-with, Content-Type, origin, authentication, accept,  
7 client-security-token, accessToken, clientId, clientSecret  
8 Access-Control-Allow-Credentials: true  
9 Content-Length: 0  
10 Access-Control-Allow-Headers: s-requested-with  
11 Content-Language: en,en-us  
12 to: ISS  
13 Access-Control-Expose-Headers: transactionId, transactionDate, offset,  
14 language  
15 elapsedTime: 104  
16 Access-Control-Allow-Methods: POST, GET  
17 Access-Control-Allow-Origin: \*  
18 gen: 2  
19 vin: [REDACTED]  
20 timeId: [REDACTED]  
21 from: SPA  
22 guid: [REDACTED]  
23 X-ORACLE-SMS-RID: 011  
24 X-ORACLE-SMS-RID: 011  
25 X-ORACLE-SMS-ECID: [REDACTED]  
26 X-ORACLE-SMS-ECID: [REDACTED]  
27 Access-Control-Allow-Headers: transactionId, transactionDate, offset,  
28 language  
29 Connection: close  
30 Content-Type: application/json  
31 Public-Key-Firm-Report  
32 gin-sha256="7b010e00qg  
33 gin-sha256="7b010e00qg  
34 report-uri="https://  
35

Request

1 POST /bin/common/MyAccountServlet HTTP/1.1  
2 Host: owners.com  
3 Cookie: AMCV\_C3BCED0154FA24300A4C98A1A40AdobeOrg=  
4 -40860457147CMCIDT847C1932647CMCNIDA7C866070475059627107614940279516976  
5 0656417CMCAANLH-167034963447C747CMCAAMB-167034963447C6GlynTcLPuiQxYRzms  
6 \_pkqt1G9yMCEpbZzX5dvJdYQJzPXImdJ0y47CMCOPTOUT-1669751034s47CMCNE47CvVer  
7 sion47C4.4.OV7CMCAIDA7CMCNE; mbox=  
8 PCNae2174e40fa743E5aCb67a10ba74403b.34\_0W1732273042|session8958b55eb4hf  
9 f4d908e64edaded37dfc#1669749329; \_qcl\_aw=1.1.842493504.1664394025; \_ga  
10 =0A1.2.978E03647.1664394026; dtm\_token\_sc=AQEL9Jst8hN1BgF-AdXAAQEBACQ;  
11 dtm\_token=AQEL9Jst8hN1BgF-AdXAAQEBACQ; utag\_main=  
12 v\_id:010193421afe001772f3eab5402505050002000d00fb04\_en:24\_se:54\_ss:04\_s  
13 t:16690300519045vap1\_domain=,.,.,.comfms\_id:166902023027743Bexp-s  
14 emion4\_pm:243Bexp-session; s\_acid=  
15 MCNIDA7C86607047505962710761494027951697606564;  
16 AMCV\_C3BCED0154FA24300A4C98A1A40AdobeOrg=  
17 158554013547CMCNIDA7C8660704750596271076149402795169760656447CMCIDT847C  
18 1931847CMCAIDA7CMCNE47CMCOPTOUT-1669035441s47CMCNE47CvVersion47C4.4.0;  
19 \_tt\_enable\_cookie=1; \_ttp=6e53098d-6220-4334-82ed-92912484233d; \_scid=  
20 910bdf1d-1fc0-430e-8d85-c950393d2a74; \_sctr=1|1664600400000;  
21  
22 Content-Length: 113  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35

Response

1 HTTP/1.1 200 OK  
2 Server: Apache/2.4.51 (Ubuntu) Communicque/4.3.2 OpenSSL/1.0.2k-fips  
3 X-Frame-Options: SAMEORIGIN  
4 X-Content-Type-Options: nosniff  
5 X-Frame-Options: SAMEORIGIN  
6 Content-Length: 0  
7 X-XSS-Protection: 1; mode=block  
8 Date: Tue, 29 Nov 2022 19:19:56 GMT  
9 Connection: keep-alive  
10 Strict-Transport-Security: max-age=86400; includeSubDomains  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35





# #11: Two popular vehicles

## Design:

- Failure to provide a full set of security requirements (MFA, out-of-band, email verification)
- Threat modeling is essential for such scenarios.

## Development:

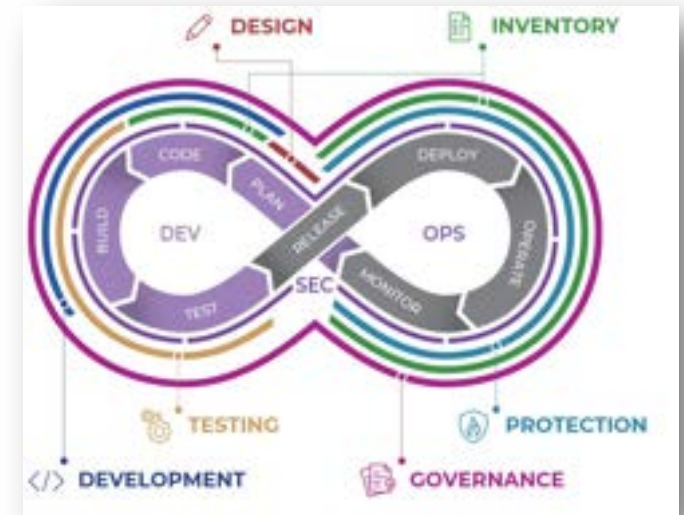
- Use robust input sanitization methods.
- Beware of hidden risks with regular expressions.

## Testing:

- Sensitive code (such as vehicle control) to be peer-reviewed by security experts.
- Static code analysis would likely have detected the poor input sanitization.

## Protection:

- Critical operations should be verified by out-of-band authorization or multi-factor access methods.





# The top-ranking issues

Vulnerability	Count
Excessive information exposure (API3)	4
Hardcoded tokens	4
Security by obscurity	2
Cross-site request forgery	2
Broken object-level authorization (API1)	1
Lack of rate-limiting (API4)	1
SQL Injection	1
Broken authentication (API2)	1
Insecure default configuration (API7)	1





# Preventing API vulnerabilities

How to avoid featuring in the next webinar !



# Take 1



how do I protect against BOLA?



BOLA, or the Bias from Off-License Annotations, is a type of bias that can occur when training natural language processing models. To protect against BOLA, it's important to carefully curate and evaluate the training data that is used to build the model. This can include checking the data for any potential biases, as well as using techniques like data augmentation and regularization to prevent the model from overfitting to the training data. Additionally, it can be helpful to test the model on a diverse set of inputs to ensure that it is able to generalize well and avoid making biased predictions.







## Take 2



how do I protect against IDOR?



IDOR, or Insecure Direct Object References, is a type of vulnerability that can occur in web applications. To protect against IDOR, it's important to properly validate user input and ensure that users are only able to access objects that they are authorized to access. This can include implementing proper authentication and access control mechanisms, as well as regularly testing the application for vulnerabilities. Additionally, it can be helpful to use frameworks and libraries that provide built-in security features to help prevent IDOR vulnerabilities.





## Take 3



how I protect against cross site request forgery?



Cross-site request forgery, or CSRF, is a type of attack that occurs when a malicious website tricks a user's web browser into making unauthorized requests to another website. To protect against CSRF attacks, it's important to implement proper CSRF prevention measures in your web application. This can include using unique, unpredictable tokens to verify each request, as well as implementing measures to prevent the attackers from being able to read or guess the value of these tokens. Additionally, it can be helpful to use frameworks and libraries that provide built-in CSRF protection, as well as regularly testing your application for vulnerabilities.



## Take 4



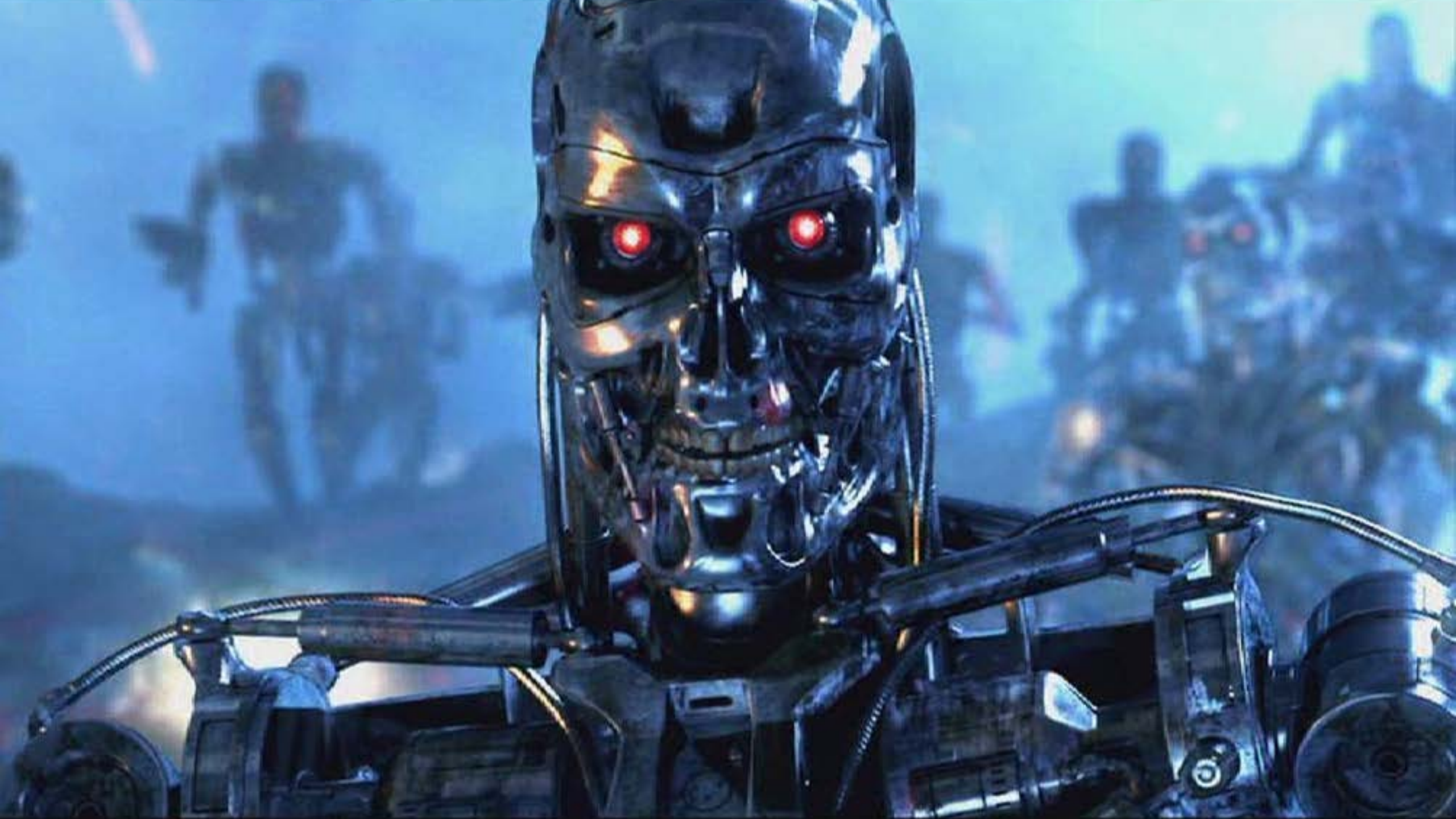
how do I protect against SQL injection



SQL injection is a type of attack that occurs when an attacker inserts malicious SQL code into a web application in order to gain access to sensitive information or to manipulate the data in the database. To protect against SQL injection attacks, it's important to properly validate user input and ensure that it does not contain any malicious SQL code. This can include using prepared statements and parameterized queries, which help to separate the user input from the SQL code and prevent the attacker from being able to insert their own code. Additionally, it can be helpful to use frameworks and libraries that provide built-in SQL injection protection, as well as regularly testing your application for vulnerabilities.











# Computer says no !



will you do my next webinar for me please?



I'm sorry, but I'm not able to do that for you. I'm a large language model trained by OpenAI and I don't have the ability to create or participate in webinars. I'm only able to provide answers to questions and offer general information on a wide range of topics. Is there something else I can help you with?  



## Threat modelling

- Establish your attack surface
- Understand the:
  - Use cases
  - Abuse cases
- Mitigations and compensating controls
- Expose assumptions and misunderstandings





## Use narrow scoped tokens rather than keys

- Supply chain security is increasingly important as apps are assembled rather than written.
- **Hyper automation** requires connectivity between systems using APIs and their keys/tokens.
- These API keys/tokens now form part of the overall supply chain.
- Limit their scope and duration

### The Next Generation of Supply Chain Attacks Is Here to Stay

With the proliferation of interconnected third-party applications, new strategies are needed to close the security gap.



**Alon Jackson**

CEO & Co-Founder, Astrix Security

November 18, 2022



<https://www.darkreading.com/attacks-breaches/the-next-generation-of-supply-chain-attacks-is-here-to-stay>





## Use standard libraries and components

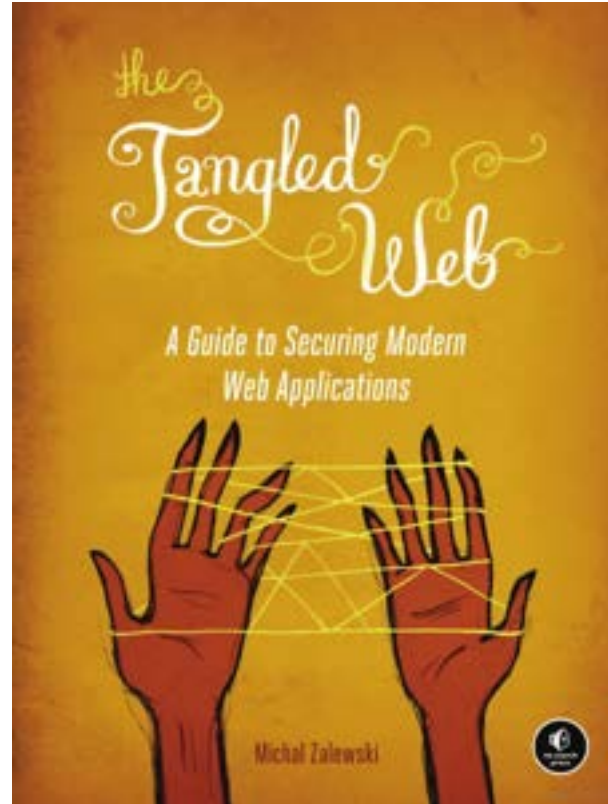
- NEVER write your own cryptography functions or methods
- Use standard libraries for common functionality (and scan them for vulnerabilities)
- Stand on the shoulder of giants (and contribute fixes and improvements)





## Remember the 'old skool' stuff

- CORS/Same-Origin-Policy
- CSRF
- SSRF
- SQL Injection



<https://nostarch.com/tangledweb>



<https://42crunch.com/defending-apis-with-jim-manico-episode-1/>



## 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 error

Generic API Key detected

by r2c

Run Locally | Use in CI • Tag - sha256:69bcbd59410f3cd2961848e42ccbd67e4405b2fc5c344240e534a819a46a5e58

**RULE** Expand rule

```
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
```

**TEST CODE** Python

```
1 # ruleid: detected-generic-api-key
2 api_key=1234567890123456789012345678901234567890
3
4 # ruleid: detected-generic-api-key
5 APIKEY : 123567890123456789012345678901234567890
6
```

[0] Source for rule

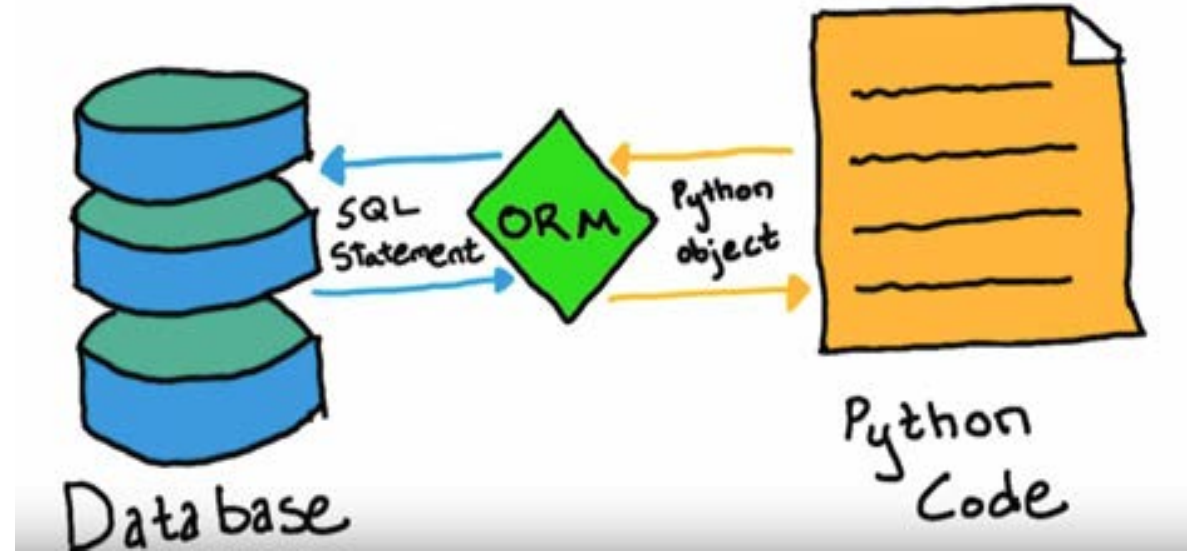
License: Commons Clause License Condition v1.0[LGPL-2.1-only] See less

Run



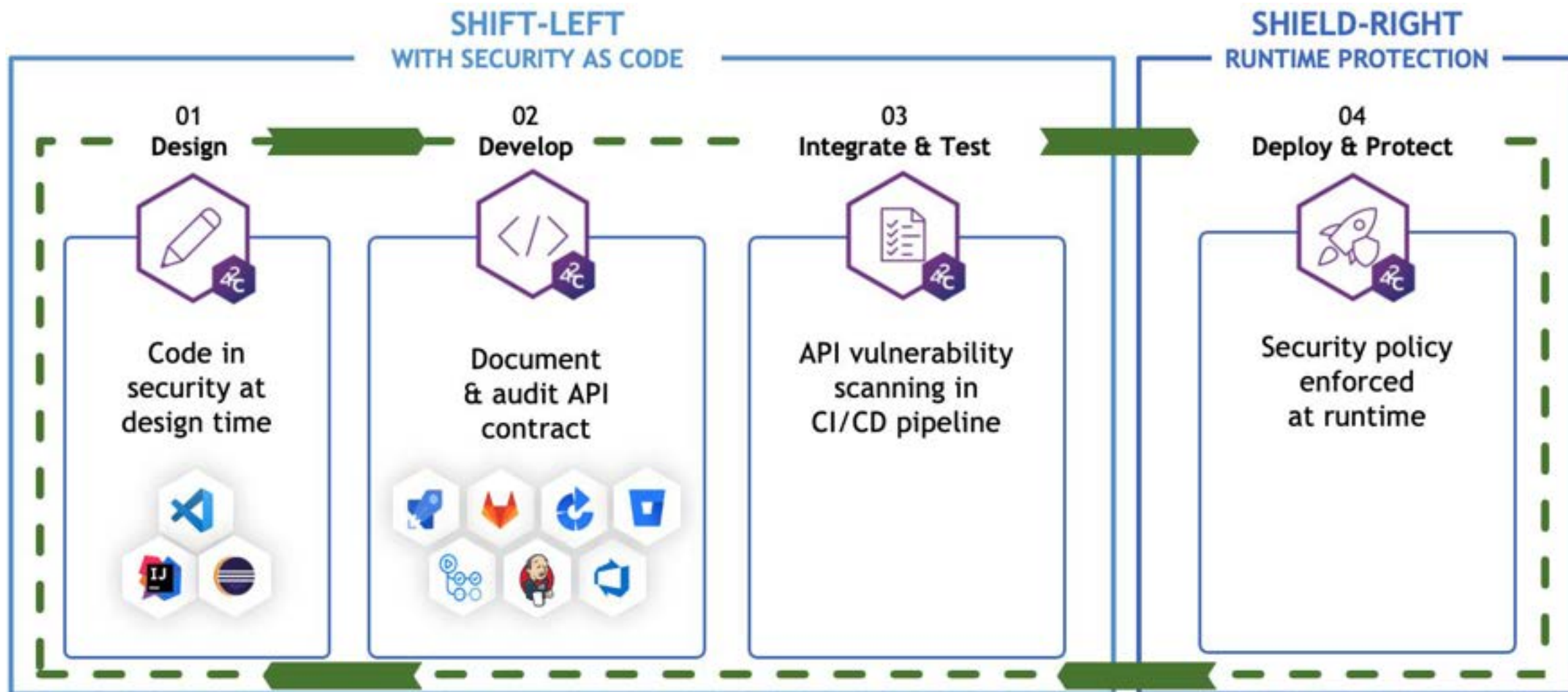
# Be wary of your Object Relational Mapper

- 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





# 42Crunch - shift-left, shield-right





## My API security predictions for 2023

- Greater developer awareness of API security
- Training courses
- Supply chain security for APIs
- Automated API security testing
- API threat hunting





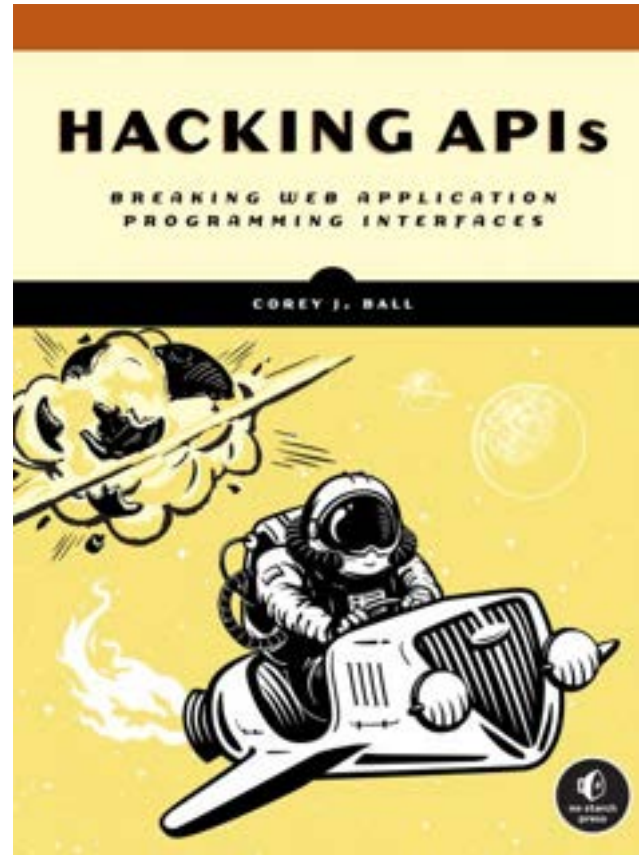
## Learning more

APISecurity.io



<https://apisecurity.io/>

“Hacking APIs” - Corey Ball



<https://nostarch.com/hacking-apis>

“Defending APIs against Cyber Attack” - Colin Domoney



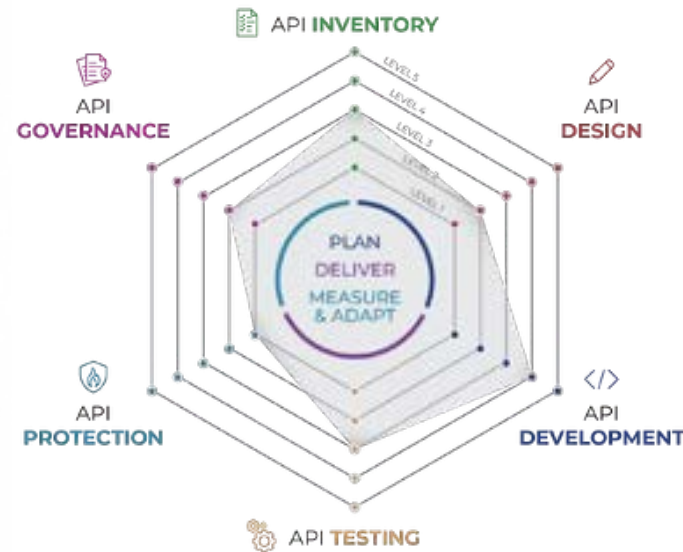
<https://amzn.to/3fHp8Mz>





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/>



Upcoming News

Further Activities

apidays  
10 YEAR ANNIVERSARY



APISecurity.io Weekly Newsletter

<https://apisecurity.io/>



OpenAPI Editor - Free Download

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

