API Vulnerabilties and What to Do About Them

Eoin Woods

accu 2021
VIRTUAL EVENT

Bloomberg Engineering
undo
mosaic CONSULTANTS TO FINANCIAL SERVICES

Agenda

1. THE STATE OF API SECURITY
2. INTRODUCING SOFTWARE SECURITY AND OWASP
3. THE TOP 10 API SECURITY RISKS
4. IMPROVING SOFTWARE SECURITY
5. SUMMARY

Dr Eoin Woods – “Owen”

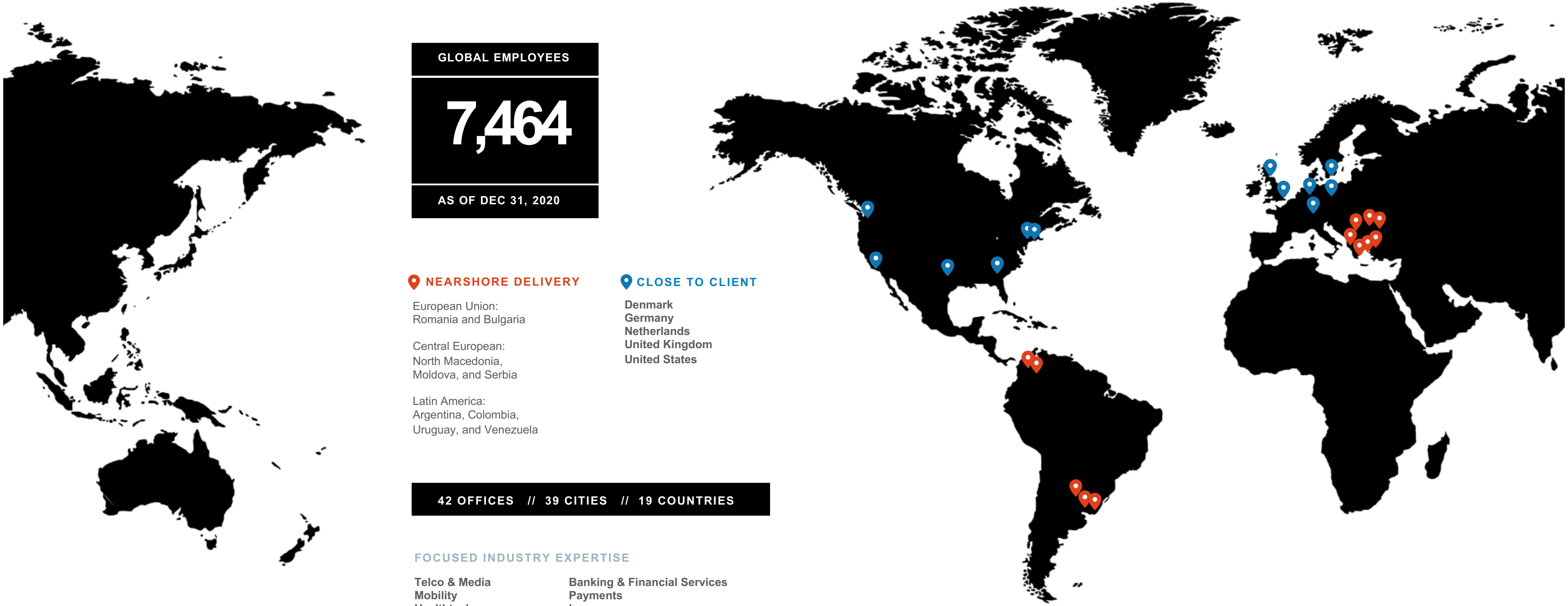
CTO at Endava since 2015

- 1990 – 2003: Product companies in UK & US
- 2003 – 2014: Capital Markets companies

Been trying to bridge ”security” and “development” for a long time

Author, speaker, community guy

www.eoinwoods.info / @eoinwoodz



GLOBAL EMPLOYEES

7,464

AS OF DEC 31, 2020

NEARSHORE DELIVERY

European Union:
Romania and Bulgaria

Central European:
North Macedonia,
Moldova, and Serbia

Latin America:
Argentina, Colombia,
Uruguay, and Venezuela

CLOSE TO CLIENT

Denmark
Germany
Netherlands
United Kingdom
United States

42 OFFICES // 39 CITIES // 19 COUNTRIES

FOCUSED INDUSTRY EXPERTISE

Telco & Media
Mobility
Healthtech
Retail & CPG
Banking & Financial Services
Payments
Insurance
Investment Management

1

THE STATE OF API SECURITY

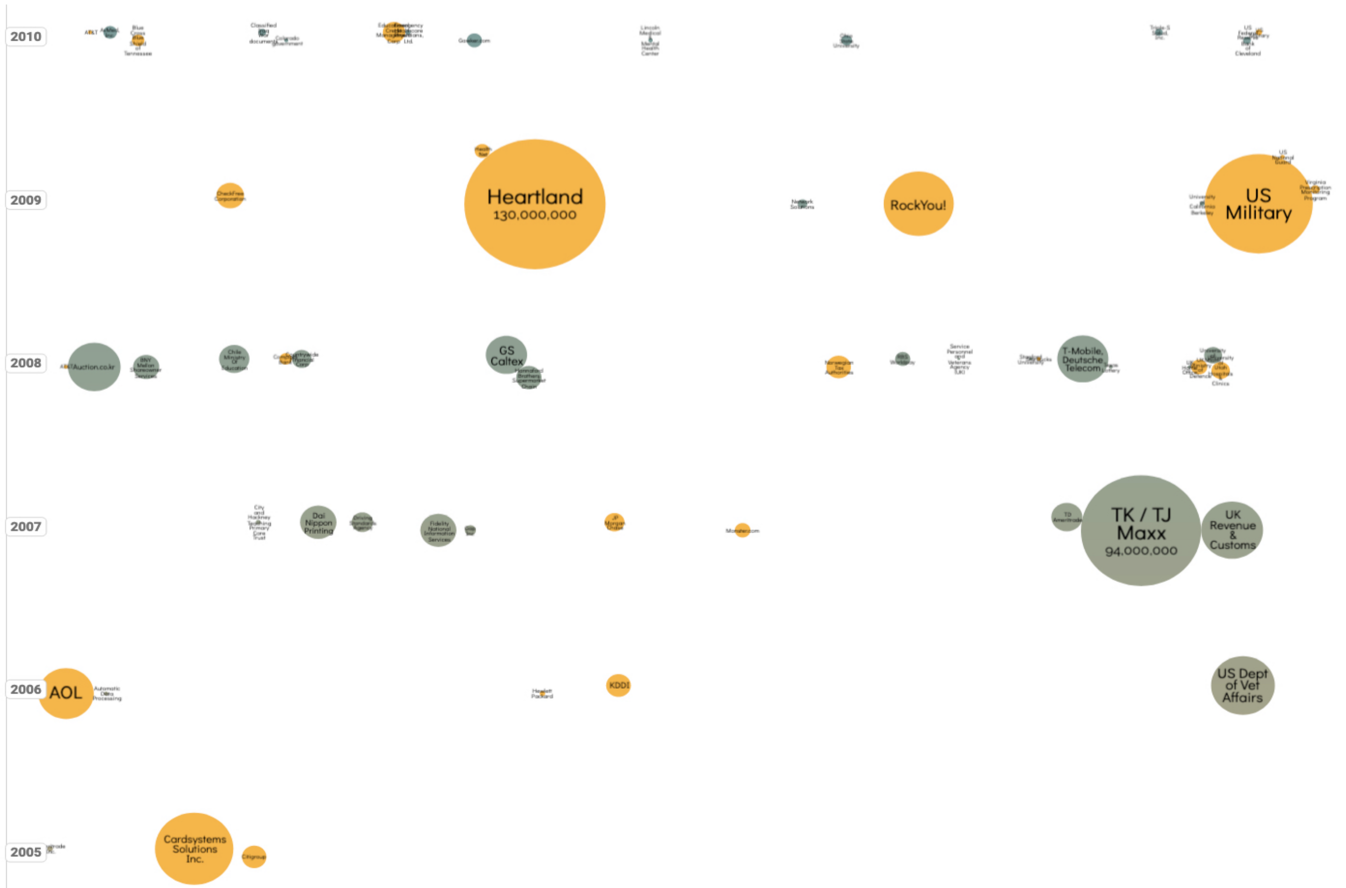
Why Security Threats Matter

- We need **dependable** systems even if things go wrong
    - Malice, Mistakes, Mischance
- People are sometimes **bad**, **careless** or just **unlucky**
- System **security** aims to **mitigate** these situations

TODAY’S THREAT LANDSCAPE

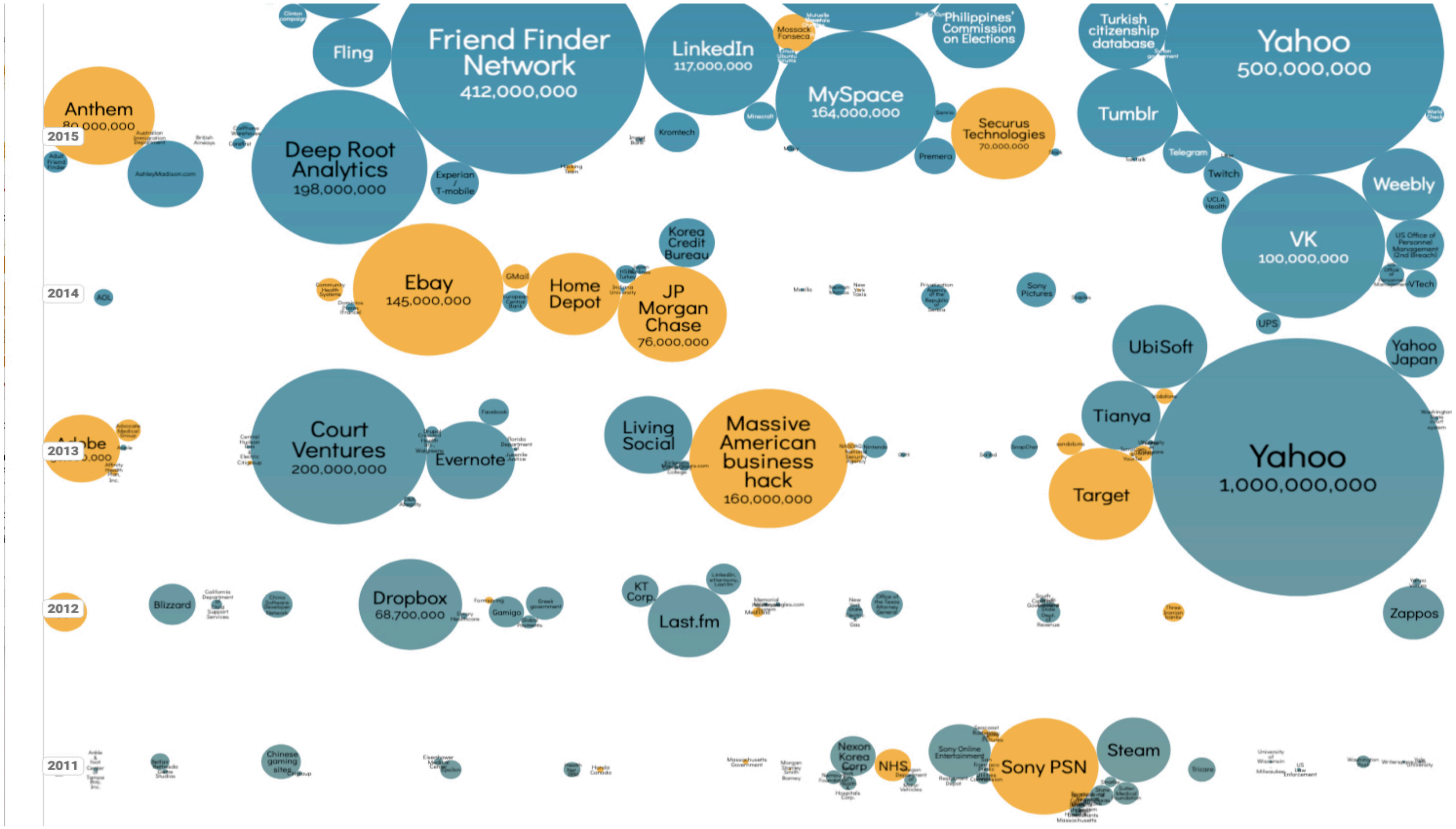
- Internal applications exposed on the Internet
- Introspection of APIs
- Attacks being ”weaponized”

DATA BREACHES 2005 - 2010

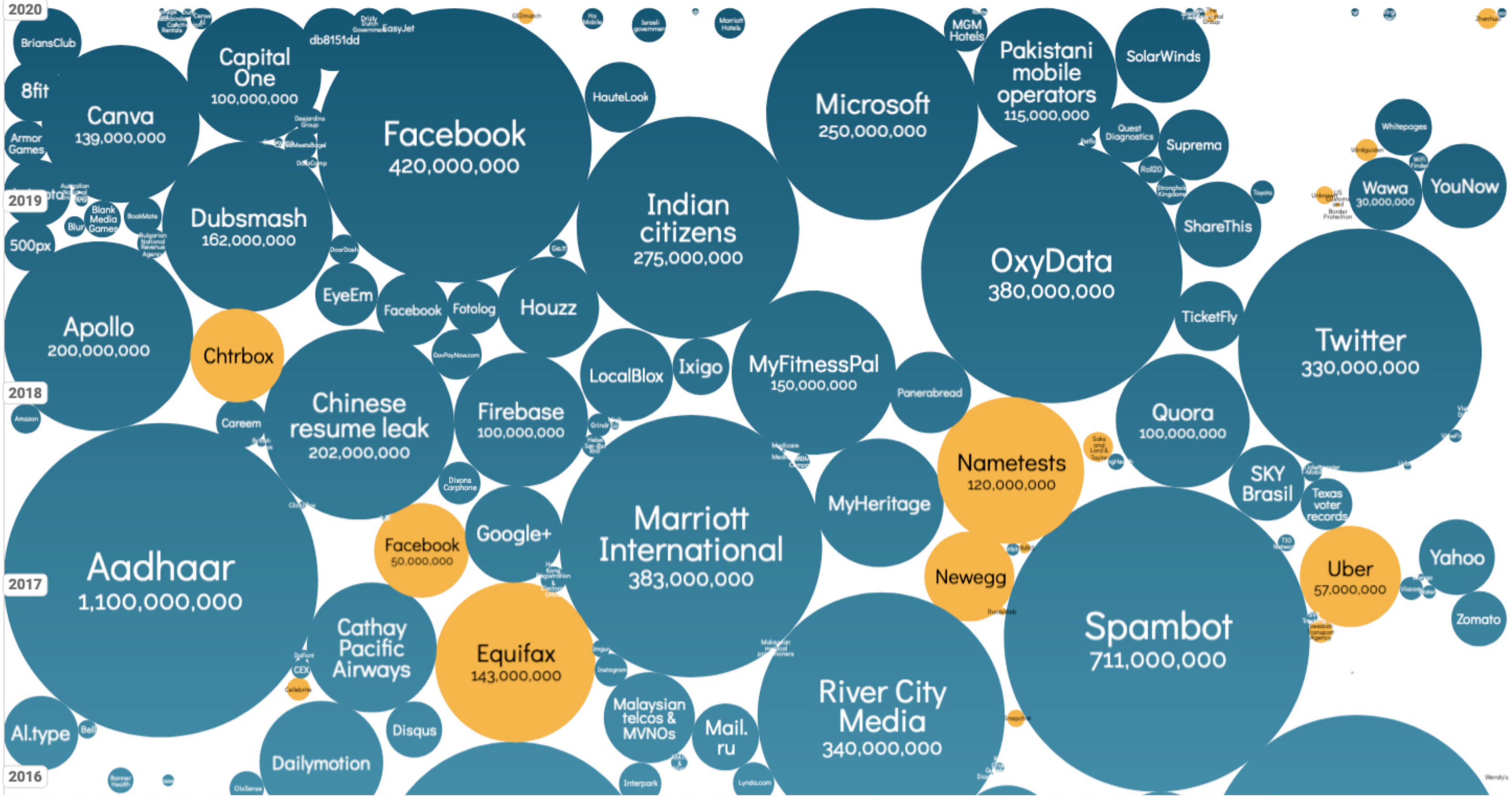


https://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks

DATA BREACHES 2011 - 2015



DATA BREACHES 2016-2020

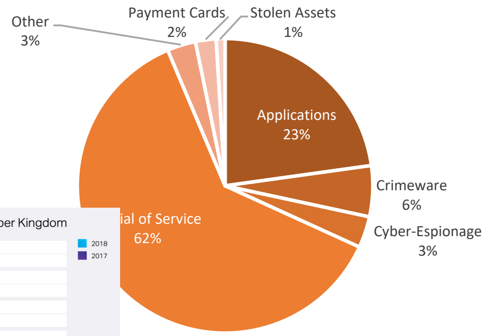


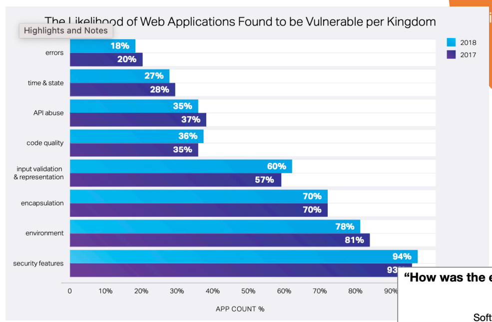
The Importance of Application Security

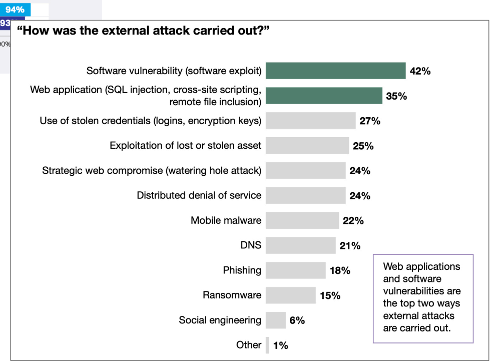
Verizon 2019 Data Breach Investigation report found **applications** were the root cause of about **25% of breaches**

Microfocus analysis of **Fortify on Demand** data found **93%** of applications had a **security bug**

Forrester 2019 survey suggests that **35%** of security incidents had a **webapp** as a root cause







https://enterprise.verizon.com/resources/reports/dbir
https://www.microfocus.com/en-us/assets/security/application-security-risk-report
https://www.forrester.com/report/The+State+Of+Application+Security+2019

What do we mean by APIs?

- We know APIs are as old as software
    - any interface to allow the invocation of one piece of software from another

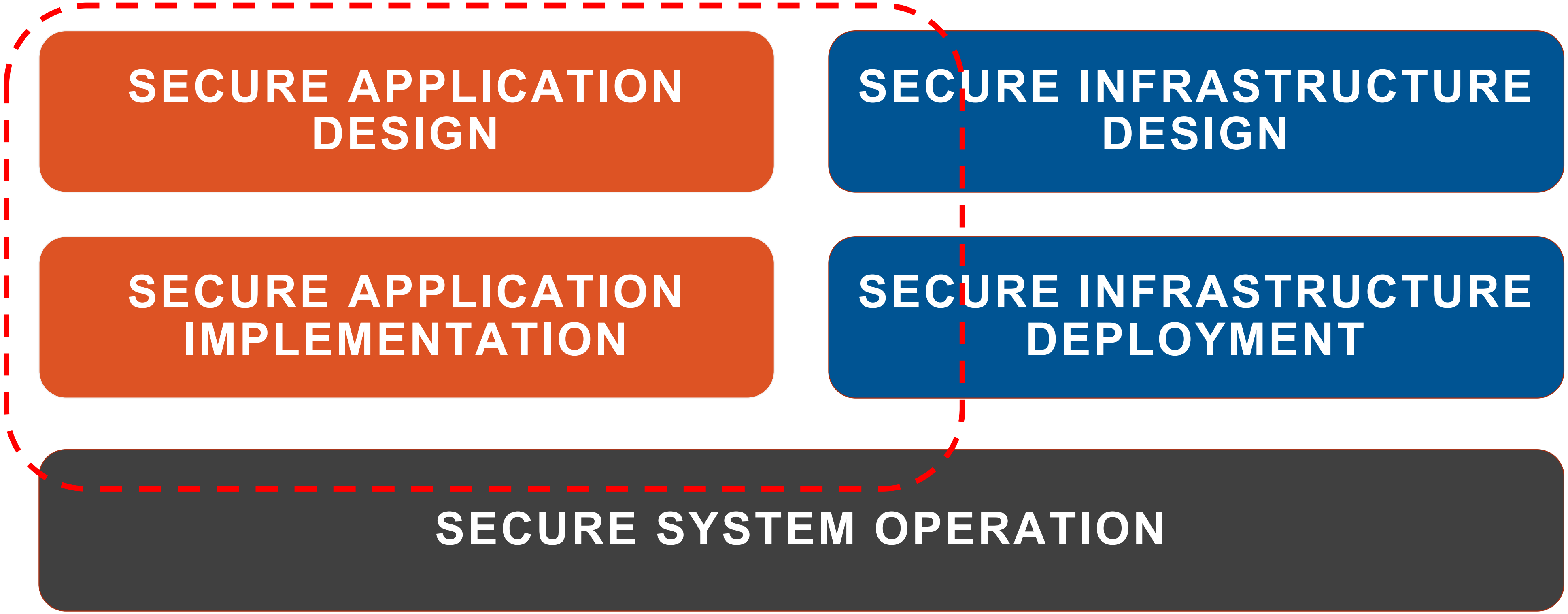
For this talk we'll focus on **network** APIs

- Any network accessible way of executing an operation on another piece of software
    - RPCs, RMIs, REST, GraphQL, …
- In most cases we're assuming a **"REST style" API** – e.g. JSON over HTTP

2

INTRODUCING SOFTWARE SECURITY & OWASP

ASPECTS OF SECURITY PRACTICE



Who are OWASP?

The Open Web Application Security Project

- Largely volunteer organisation, largely online

Exists to improve the state of software security

- Research, tools, guidance, standards
- Runs local chapters for face-to-face meetings

“OWASP Top 10” projects list top application security risks

- OWASP Top 10 Webapp Security Risks
- OWASP Top 10 Mobile Risks
- OWASP Top 10 API Risks

Other Key Security Organisations

MITRE Corporation

- Common Vulnerabilities and Exposures (CVE)
- Common Weaknesses Enumeration (CWE)

SAFECode

- Fundamental Practices for Secure Software Development
- Training

There are a lot of others too (CPNI, CERT, CIS, ISSA, ISC2, …)

3

THE API TOP 10 SECURITY RISKS

How was the 2019 API List Produced?

Volunteer project of the OWASP organisation

- 3 authors, ~35 contributors
- `https://www.owasp.org/index.php/OWASP_API_Security_Project`

First version in 2019 so less mature than the WebApp Top 10

- Initial analysis of public data sets (e.g. vulnerabilities & bug bounty data)
- Penetration testing practitioners surveyed for their own ”top 10s”
- Top 10 resulted from a consensus between data and surveys
- Expert review provided refinement
- Some work to do to achieve full conceptual consistency and coherence

Future plan to extend a public call for data (like the WebApp set)

OWASP API Top 10 - 2019

#1 Broken Object Authorization
#2 Broken User Authentication
#3 Excessive Data Exposure
#4 Resources & Rate Limiting
#5 Broken Function Authorization
#6 Mass Assignment
#7 Security Misconfiguration
#8 Injection
#9 Improper Asset Management
#10 Insufficient Logging and Monitoring

OWASP API Top 10 - 2019

#1 Broken Object Authorization
#2 Broken User Authentication
#3 Excessive Data Exposure
#4 Resources & Rate Limiting
#5 Broken Function Authorization

#6 Mass Assignment
#7 Security Misconfiguration
#8 Injection
#9 Improper Asset Management
#10 Insufficient Logging and Monitoring

Some are closely related to the Webapp Top 10
A few surprising omissions (e.g. vulnerable components)

#1 Broken Object-Level Authorisation

Exploitability	3
Prevalence	3
Detectability	2
Technical	3

- After authentication many APIs don't fully authorise access to resources
  - To make matters worse object ”keys” are often predictable or accessible

```
$> wget https://aprovider.com/era/reports/1224459/monthly-latest
```

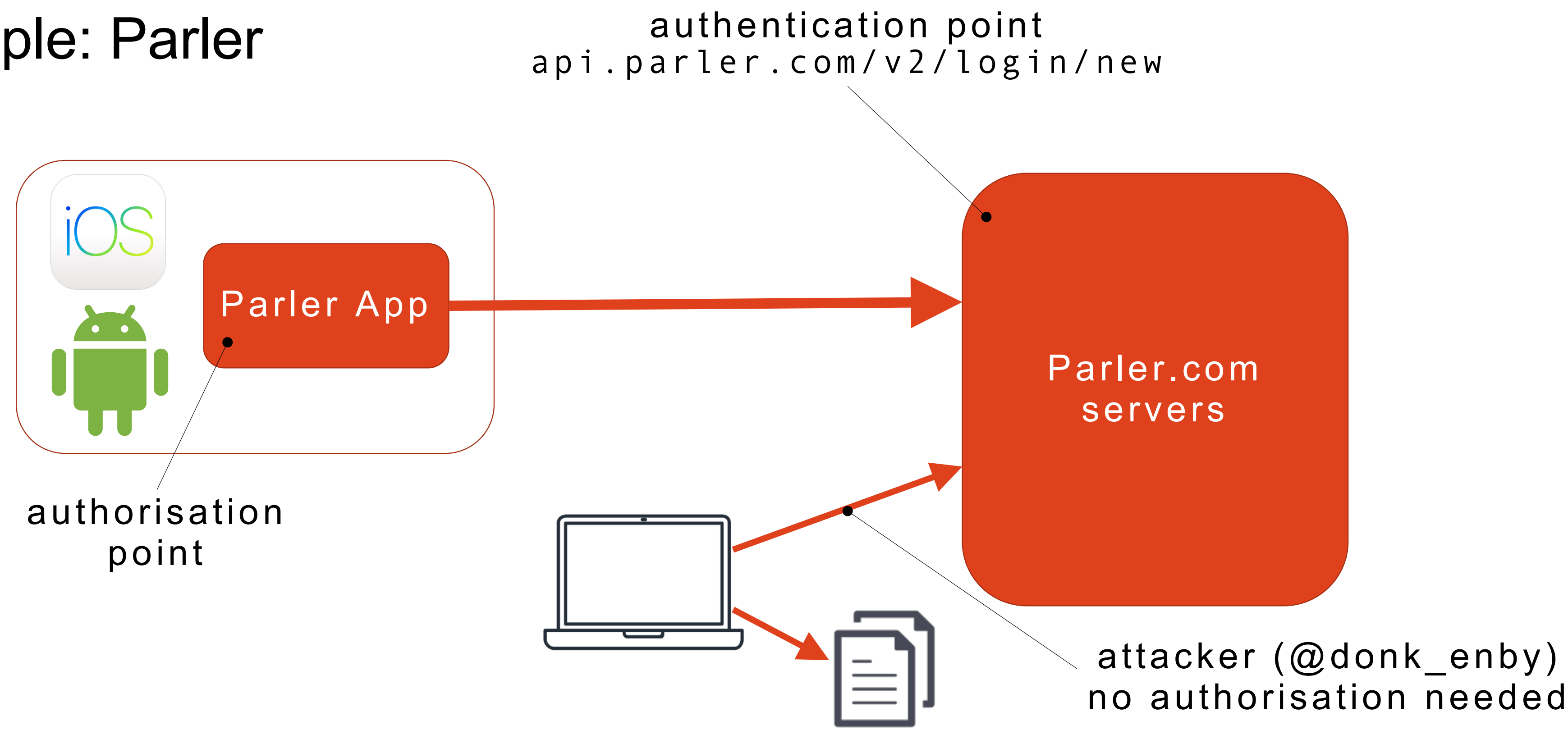
  - What would happen if you tried 1224470?
  - Hopefully the API would recognise that you weren't authorised to view it
  - It turns out that many don't!

- Mitigations: enforce object authorisation for every request, well structured API design making need for authorisation clearer, long random object keys, testing

#1 Broken Object-Level Authorisation

Exploitability	3
Prevalence	3
Detectability	2
Technical	3

Example: Parler



https://medium.com/swlh/exposing-the-riot-parler-api-mistakes-9a4db4e905d5
https://github.com/d0nk/parler-tricks
https://github.com/daniel-centore/ParlerScraper

#2 Broken User Authentication

Exploitability	3
Prevalence	2
Detectability	2
Technical	2

- A range of possible problems rather than a single weakness
  - Allowing “credential stuffing”
  - Accepting weak passwords => brute-force credential attacks
  - Revealing authentication information in the API structure (e.g. URL)
  - Missing or incorrect validation of authentication tokens (e.g. JWT)
  - Mistakes in protocol implementation (very easy to do !)

- Example: see example #10

#2 Broken User Authentication

Exploitability	3
Prevalence	2
Detectability	2
Technical	2

- Mitigations:
  - Multi-factor authentication for humans
  - Controls around login & credential recovery (e.g. password rules, lockout periods after failures, captchas, rate limiting)
  - Use proven, tested authentication mechanisms
  - Take time to understand any sophisticated security technologies
  - Careful implementation with expert design and code review
  - Functional and penetration testing

#3 Excessive Data Exposure

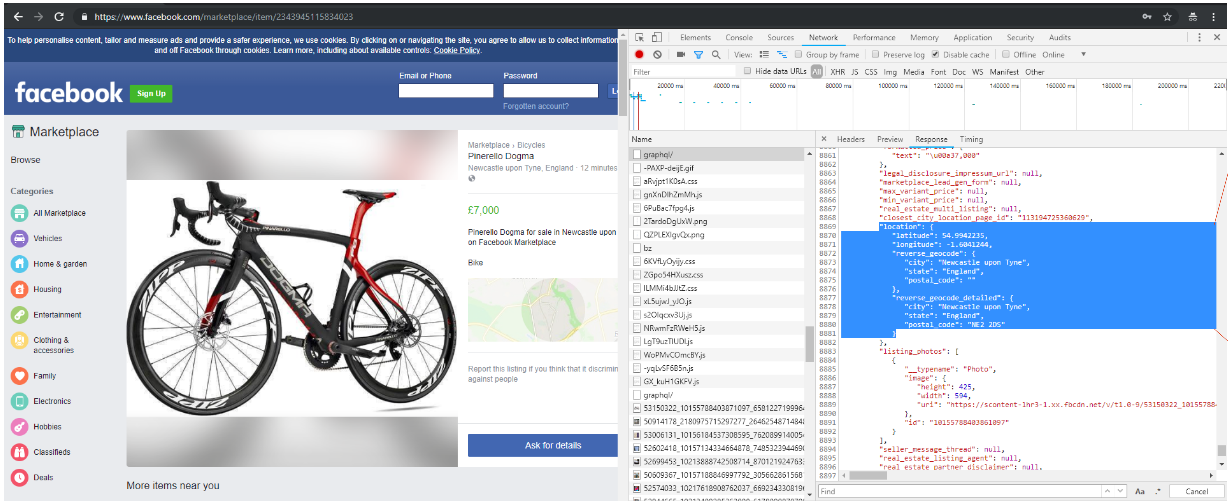
Exploitability	3
Prevalence	2
Detectability	2
Technical	2

- APIs often return more data that is required by the client
  - client-side filtering hides this from the user but not from software
- API developers don’t always know what the client needs
  - or are trying to provide a more general solution to avoid rework
- Sometimes an assumption that the client is ”trusted”
  - analogous problem to browser-side security in webapps
- Problem often not obvious unless you know the data
  - automated tools aren’t going to spot this

#3 Excessive Data Exposure

Exploitability	3
Prevalence	2
Detectability	2
Technical	2

Example: Facebook Marketplace (2019)




```
"location": {
  "latitude": 54.9942235,
  "longitude": -1.6041244,
  "reverse_geocode": {
      "city": "Newcastle upon Tyne",
      "state": "England",
      "postal_code": ""
  },
  "reverse_geocode_detailed": {
    "city": "Newcastle upon Tyne",
    "state": "England",
    "postal_code": "NE2 2DS"
  }
}
```


https://www.7elements.co.uk/resources/blog/facebooks-burglary-shopping-list/

#3 Excessive Data Exposure

Exploitability	3
Prevalence	2
Detectability	2
Technical	2

- Mitigations
  - Assume the client is untrusted when developing an API
  - Always use the "need to know" principle when designing data types
    - needs understanding of the context of the API request
  - Don't return serialised forms of internal types
    - can leak information over time
    - use specifically designed return types with the right data items
  - Identify sensitive information classes (e.g. PII, card data, …) and have a specific review of any API call that accesses this information

#4 Resources and Rate Limiting

Exploitability	2
Prevalence	3
Detectability	3
Technical	2

- Classical DoS attacks use network protocols (e.g. SYN flood)
- APIs are also vulnerable to overload attacks
  - can be exacerbated by the right (excessive) parameter values
  - e.g. parallel upload of multi-GB binary files
- Two dimensions
  - Number of parallel requests allowed
  - Quantity of resources each request can be allocated
- Mitigations:
  - Rate limiting at API level (spike limit, limit in time interval)
  - Rate limiting at session or user level (ditto)
  - Hard limits on parameter values and sizes
  - Runtime limits on memory, CPU, file descriptors, …

#4 Resources and Rate Limiting

Exploitability	2
Prevalence	3
Detectability	3
Technical	2

```
$> wget https://svc.com/inv/item?name=%22%2a%22&maxsize=9999999
```

- Hopefully this gets stopped immediately by a validation check
  - Or overridden within the API by an internal maximum
  - Unfortunately, quite a few APIs don’t always do this
- Result is likely to be a large database result set and a huge amount of memory used => a runtime failure

#5 Broken Function-Level Authorisation

Exploitability	3
Prevalence	2
Detectability	1
Technical	2

- Incomplete or incorrect authorisation checks when API called
  - like #1 (object-level authorisation) a range of possible problems
  - Rarely totally missing, usually “holes” in the implementation
- Frequently a result of a complex security model or API design
  - “correct” is complex, given interaction of authentication, roles, sensitivity levels, …
- Can be due to complexity of application or 3rd party component
  - e.g. declarative security rules can often contain subtle problems
  - e.g. “falling through” logic which ends up providing access by mistake

#5 Broken Function-Level Authorisation

Exploitability	3
Prevalence	2
Detectability	1
Technical	2

Example: NewRelic “delete filterset” vulnerability

To create a NR “filter set” you call

```
POST https://infrastructure.newrelic.com/accounts/12345/settings/filterSets
```

… passing a parameter block defining the new filter set.

It turns out that calling …

```
DELETE https://infrastructure.newrelic.com/accounts/12345/settings/filterSets
```

… could delete the filter set without checking the user is authorised to do so

https://www.cloudvector.com/owasp-api-security-top-10-broken-function-level-authorization/

#5 Broken Function-Level Authorisation

Exploitability	3
Prevalence	2
Detectability	1
Technical	2

- Mitigations
  - Simple as possible in design and implementation
  - Highlight sensitive operations for specific review
  - Thorough automated functional testing of authorisation
  - Take time to understand sophisticated security technology
  - Don’t invent your own security technology (again)
  - Always default to “no access”

#6 Mass Assignment

Exploitability	2
Prevalence	2
Detectability	2
Technical	2

- Different fields in a data entity often have different sensitivities
- We often use libraries to “bind” data elements to and from API parameter sets
  - `var item = JSON.parse(json_str); // JavaScript`
  - ```
    // Java with Jackson
    Trade t = mapr.readObject(jsonStr, Trade.class);
    ```
- Client could add “rogue” fields to overwrite sensitive state
    ```

#6 Mass Assignment

Exploitability	2
Prevalence	2
Detectability	2
Technical	2

Example: the Harbor privilege escalation vulnerability

Harbor: ”*Our mission is to be the trusted cloud native repository for Kubernetes*”

Unfortunately, their product contained a privilege escalation vulnerability:

```
POST /api/users HTTP/1.1
{
  “username”:”test”,
  ”email”:”test123@gmail.com”,
  ”realname”:”no name”,
  ”password”:”password1\u0021″,
  ”comment”:null,
  “has_admin_role”:”true”
}
```

… due to a JSON mass assignment operation in JavaScript!

https://unit42.paloaltonetworks.com/critical-vulnerability-in-harbor-enables-privilege-escalation-from-zero-to-admin-cve-2019-16097/

#6 Mass Assignment - Example

Simple filtering example:

```
function filterProperties(propL
  for (var p in obj) {
        if (!obj.hasOwnProperty(p
        if (propList.indexOf(p) =
          delete obj[p];
        }
  }
}
var fieldList = [‘name’, ‘cpid’, ‘price’, // … ];
filterProperties(fieldList, accountItem)
```


```
{
  name: ‘My Customer',
```

```
{
  name: ‘My Customer',
  cpid: '1234234',
  price: 12.54,
}
```

```
sKICAgIGlmICghb2JqLmhhc093blB
yb3BlcnR5KHApKSBjb250aW51ZTsK
ICAgIGlmIChwcm9wTGlzdC5pbmRle
E9mKHApID09PSAtMSkgewogICAgIC
BkZWxldGUgb2JqW3BdOwogICAgfQo
gIH0KfQ=='
}
```

#6 Mass Assignment

Exploitability	2
Prevalence	2
Detectability	2
Technical	2

- Mitigation:
  - Be careful when using automatic data binding libraries
  - Use specific types for API definition and explicit code to extract values and apply them to system state
  - Have "whitelists" for fields that can be updated by a client

#7 Security Misconfiguration

Exploitability	3
Prevalence	3
Detectability	3
Technical	2

- Again a class of problem rather than a single cause
  - Missing security patches
  - Incorrect authorisation configuration
  - Unnecessary features enabled
  - Security enforcement (e.g. requiring TLS) incorrect or missing
  - Exposing sensitive information (e.g. 500 error stack traces!)
- Mitigation
  - Testing (automated security tests, manual penetration testing)
  - Automated configuration and deployment for consistency
  - Expert review and code scanning throughout projects
  - Careful error handling

#7 Security Misconfiguration

Exploitability	3
Prevalence	3
Detectability	3
Technical	2

Example: Algolia Search

```
$> curl https://myappid-dsn.algolia.net/1/keys/APIKEY?x-algolia-application-id=myappid&x-algolia-api-key=a7hw7gsh273hrk382

{
      ”value”: ‘.....’,
      “createdAt”: 15173453234,
      “acl” : [“search”, “addObject”, … ”editSettings”, “listIndexes”, …]
}
```

It turns out that many people accidentally use their **admin** API key for **client** API calls because of the way that Algolia's documentation is written.

https://www.secjuice.com/api-misconfiguration-data-breach

#7 Security Misconfiguration

Exploitability	3
Prevalence	3
Detectability	3
Technical	2

```
# ModSecurity (default) configuration
SecRuleEngine DetectionOnly
SecRequestBodyAccess On
SecRule REQUEST_HEADERS:Content-Type "(?:application(?:/soap\+|/)|text/)xml" \
     "id:'200000',phase:1,t:none,t:lowercase,pass,nolog,ctl:requestBodyProcessor=XML"
SecRule REQUEST_HEADERS:Content-Type "application/json" \
     "id:'200001',phase:1,t:none,t:lowercase,pass,nolog,ctl:requestBodyProcessor=JSON"
SecRequestBodyLimit 13107200
SecRequestBodyNoFilesLimit 131072
SecRequestBodyLimitAction Reject
SecRule REQBODY_ERROR "!@eq 0" \
"id:'200002', phase:2,t:none,log,deny,status:400,msg:'Failed to parse request body.’,
    logdata:'%{reqbody_error_msg}',severity:2"
SecRule MULTIPART_STRICT_ERROR "!@eq 0" \
"id:'200003',phase:2,t:none,log,deny,status:400, \
msg:'Multipart request body failed strict validation: \
PE %{REQBODY_PROCESSOR_ERROR}, \
BQ %{MULTIPART_BOUNDARY_QUOTED}, \
BW %{MULTIPART_BOUNDARY_WHITESPACE}, \
DB %{MULTIPART_DATA_BEFORE}, \
DA %{MULTIPART_DATA_AFTER}, \
HF %{MULTIPART_HEADER_FOLDING}, \
LF %{MULTIPART_LF_LINE}, \
SM %{MULTIPART_MISSING_SEMICOLON}, \
IQ %{MULTIPART_INVALID_QUOTING}, \
IP %{MULTIPART_INVALID_PART}, \
IH %{MULTIPART_INVALID_HEADER_FOLDING}, \
FL %{MULTIPART_FILE_LIMIT_EXCEEDED}'"
SecRule MULTIPART_UNMATCHED_BOUNDARY "@eq 1" \
    "id:'200004',phase:2,t:none,log,deny,msg:'Multipart parser detected a possible unmatched boundary.'"
SecPcreMatchLimit 1000
SecPcreMatchLimitRecursion 1000
SecRule TX:/^MSC_/ "!@streq 0" \
       "id:'200005',phase:2,t:none,deny,msg:'ModSecurity internal error flagged: %{MATCHED_VAR_NAME}'"
```

https://krebsonsecurity.com/tag/capital-one-breach

#8 Injection

Exploitability	3
Prevalence	2
Detectability	3
Technical	3

- Our old friend from the Webapp Top 10!
- Dangerous in APIs as well as in webapps
- Anything interpreted can be injected:
  - Database query parameters
  - Command line arguments
  - Configuration items that are parsed and processed

#8 Injection

Exploitability	3
Prevalence	2
Detectability	3
Technical	3

Example: check parameters and avoid direct SQL

```
String tradeId = req.path().param(“tradeid”);
ESAPI.validator().getValidInput(“tradeId”, tradeId,
   “HTTPParameterValue”, 12, false, true);

String query =
   “select id, ccy, value, … from trade where id=?” ;
PreparedStatement ps = conn.prepareStatement(query);
ps.setString(1, tradeId);
ResultSet results = ps.executeQuery() ;
```

#8 Injection

Exploitability	3
Prevalence	2
Detectability	3
Technical	3

- Mitigation:
  - Validate and sanitise all data entering the system
  - Use a single, well-tested, validation library to make validation reliable and straightforward (“easy thing” == “what is actually done”)
  - Where possible use APIs rather than interpreters (e.g. bind parameters for “prepared” database queries not query strings)
  - Sanity check result payloads (e.g. maximum size checks)
  - Strongly type API interfaces and enforce types strictly

#9 Improper Asset Management

Exploitability	3
Prevalence	3
Detectability	2
Technical	2

- Many application estates today are not well understood
- Old applications can run for years with little attention
  - Will contain vulnerabilities in old software components
  - Often skipped during software security remediation work
  - Can have deliberate or accidental vulnerabilities themselves
  - Compromises may not be noticed
- Sometimes important applications have old neglected features
  - Old data interfaces left in place for backwards compatibility
  - Unsupported opensource components to avoid regression testing
  - Insecure mechanisms (e.g. FTP file transfer) to avoid touching other old applications

#9 Improper Asset Management

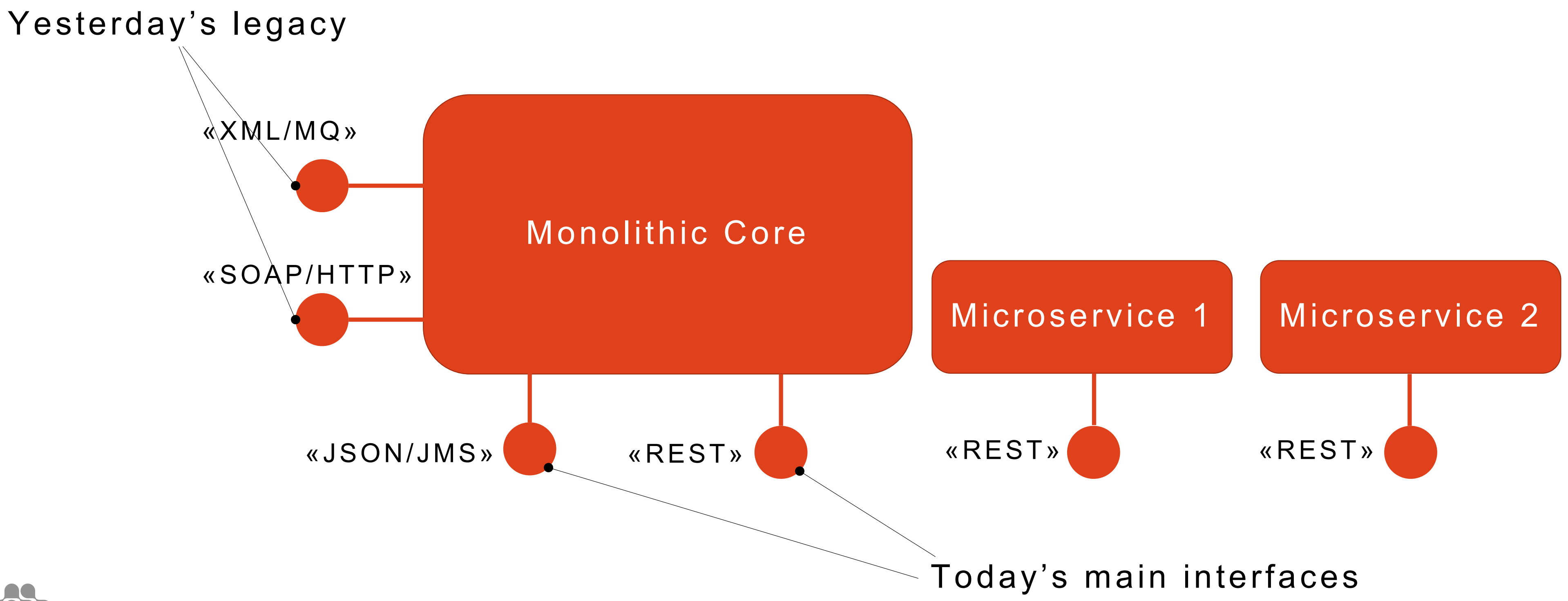
Exploitability	3
Prevalence	3
Detectability	2
Technical	2

- New applications can also introduce problems if not understood
  - Microservices introduce many moving parts with network interfaces
  - Cloud allows application teams to deploy new applications and infrastructure quickly and independently … and perhaps insecurely
  - Rate of change in modern application estates can make keeping track of the estate difficult if not automated … tomorrow's legacy

#9 Improper Asset Management

Exploitability	3
Prevalence	3
Detectability	2
Technical	2

Example: application evolution



Exploitability	3
Prevalence	3
Detectability	2
Technical	2

#9 Improper Asset Management

Example: the scale problem

Large organisations have thousands of applications, servers, services, message queues, databases, …

… all constantly changing

#9 Improper Asset Management

Exploitability	3
Prevalence	3
Detectability	2
Technical	2

- Mitigation:
  - There are no easy mitigations once in this situation!
  - Easy to say, but avoidance is the most effective mitigation
  - Finding these applications and features is often the most difficult part
  - Network scanning can be useful to find unexpected end points
  - Once found, investing in modernisation, improving security or retirement are all options
  - Automate maintenance of application and infrastructure inventories wherever possible

#10 Insufficient Logging & Monitoring

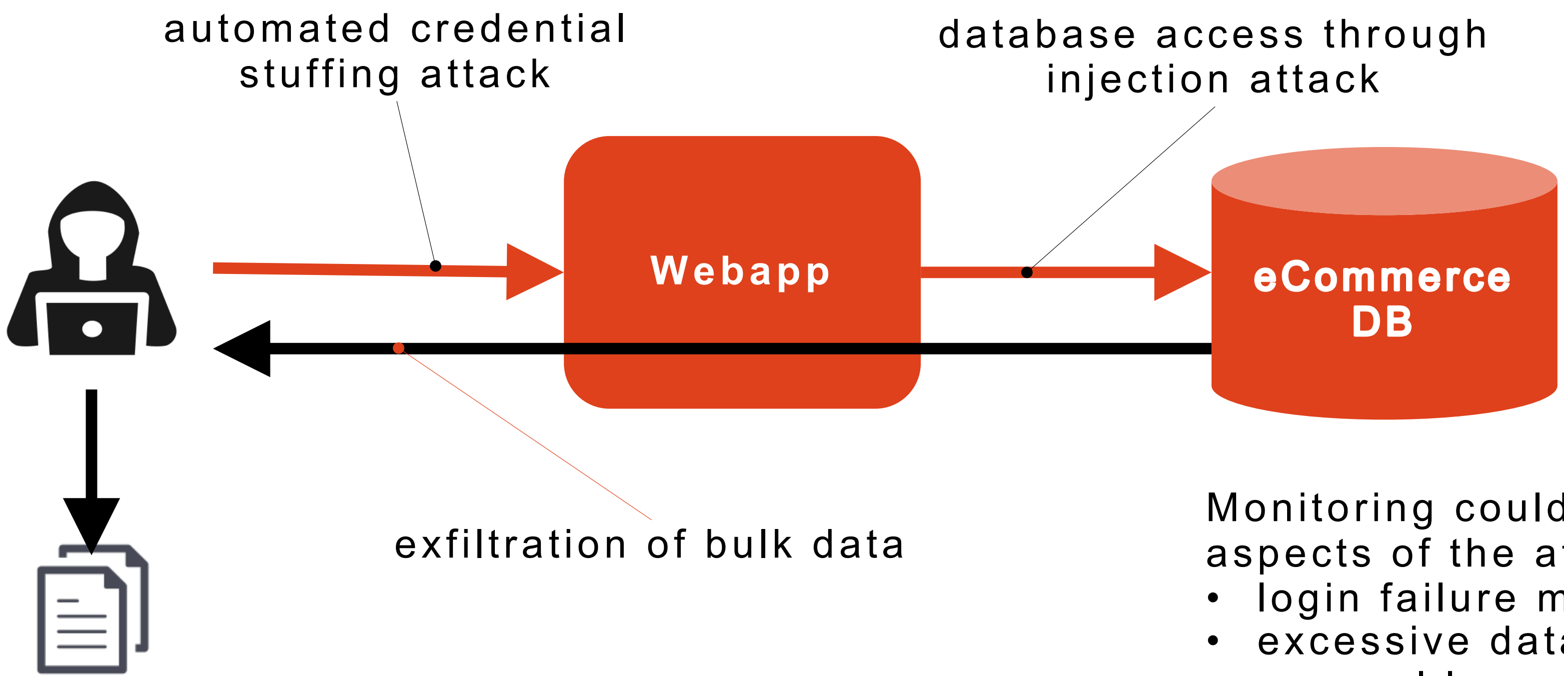
Exploitability	2
Prevalence	3
Detectability	1
Technical	2

- Another familiar “friend” from the Webapp Top 10
- Logging and monitoring rarely comes ”for free” with APIs
  - therefore it often gets forgotten or deprioritised
- Poor logging and monitoring technology, implementation or practices means it is difficult to detect and respond to suspicious activity
  - e.g. you find that an API credential has been compromised for several days … do you know what that credential has been used for while compromised?

#10 Insufficient Logging & Monitoring

Exploitability	2
Prevalence	3
Detectability	1
Technical	2

- Example: the need for monitoring



Monitoring could alert to all three aspects of the attack:

- login failure monitoring
- excessive database result set
- unusual large outbound transfer

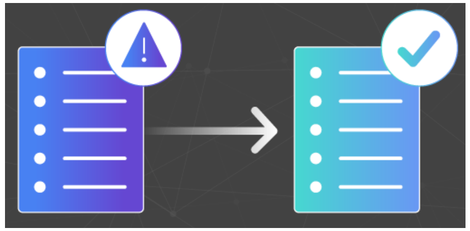
#10 Insufficient Logging & Monitoring

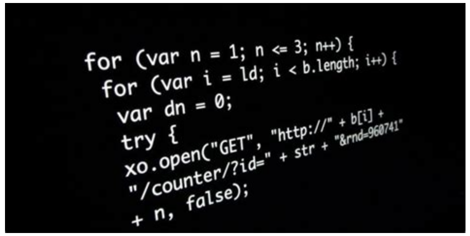
Exploitability	2
Prevalence	3
Detectability	1
Technical	2

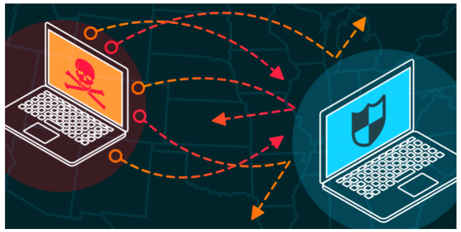
- Mitigation … all well known solutions
  - Log all security sensitive events (authentication activity, access failures, validation failures, …)
  - Keep logs accessible but secure
  - Use SEIM systems to aggregate the logs from different sources
  - Build awareness of ”normal” and create dashboards for security related metrics to allow ”abnormal” to be spotted

Summary of Vulnerability Types

- Injection
  - SQL, configuration, operating system command, …
- Inadequate validation
  - Of authentication to confirm identity of caller
  - Of authorisation to access resources
  - Accepting unexpected inputs (e.g. unnecessary fields, excessive parameter lengths)
- Implementation mistakes
  - Returning too much data
  - Incomplete or faulty authorisation checks
  - Blindly binding data structures to inputs
- Environment problems
  - Need for rate limiting
  - Monitoring and logging
  - Careful configuration of the entire stack



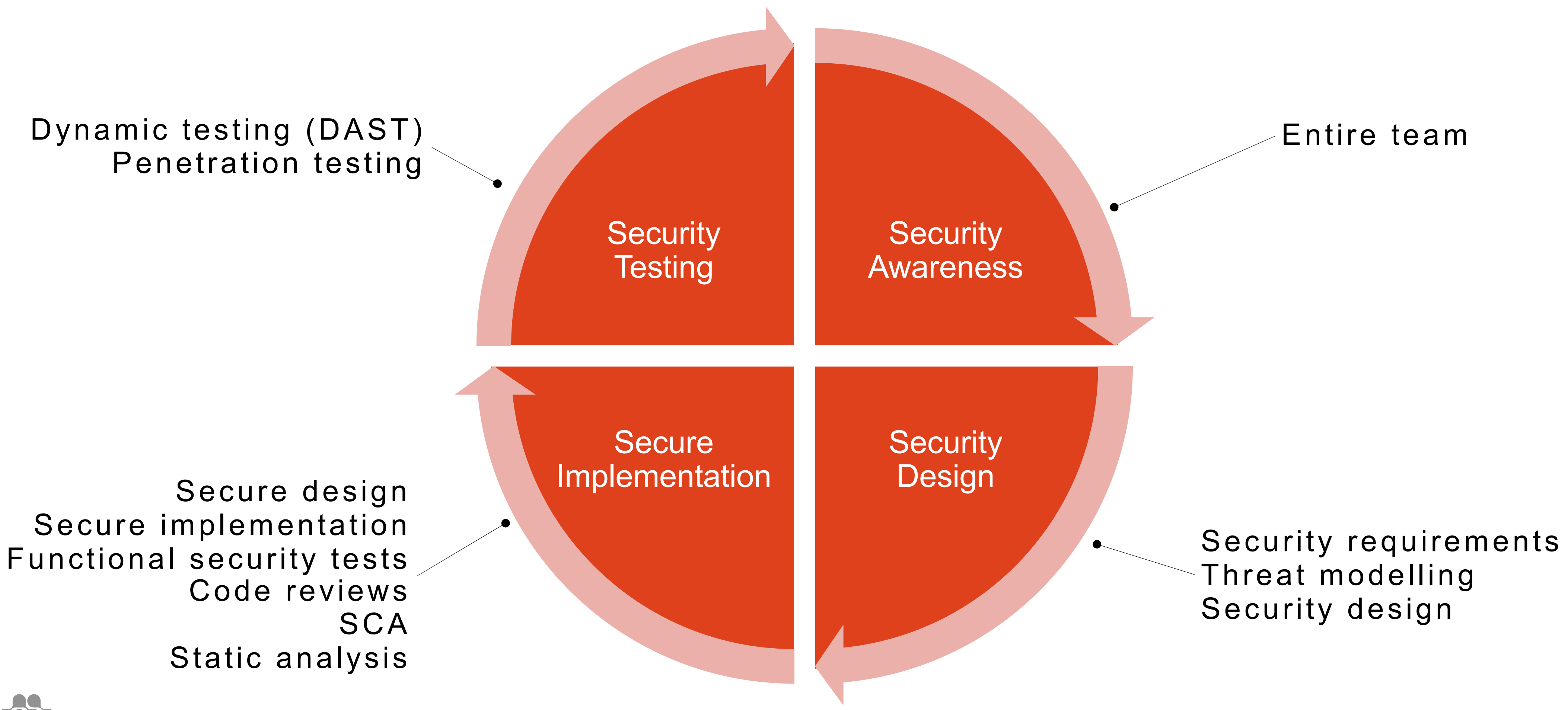




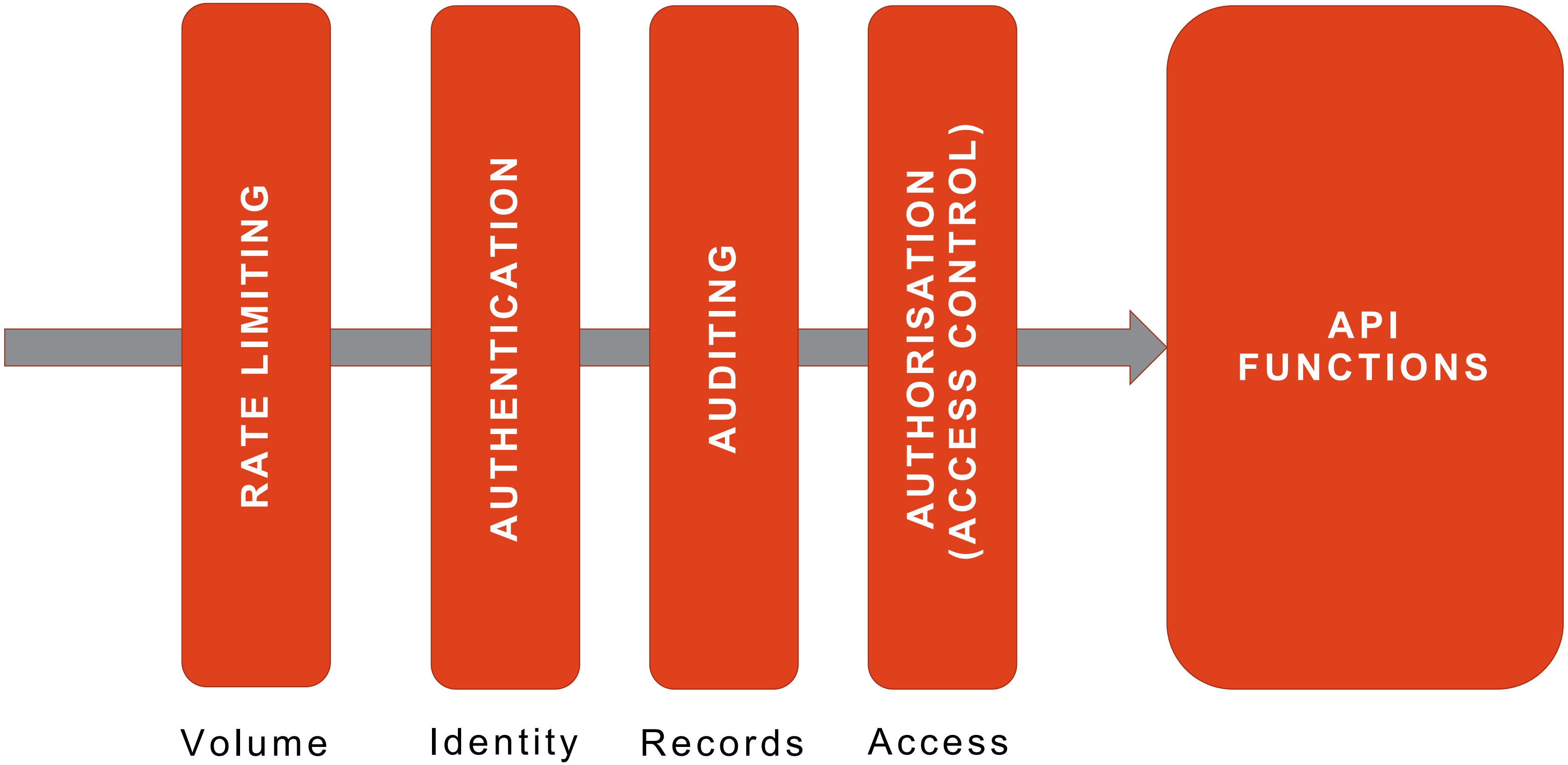
4

IMPROVING SOFTWARE SECURITY

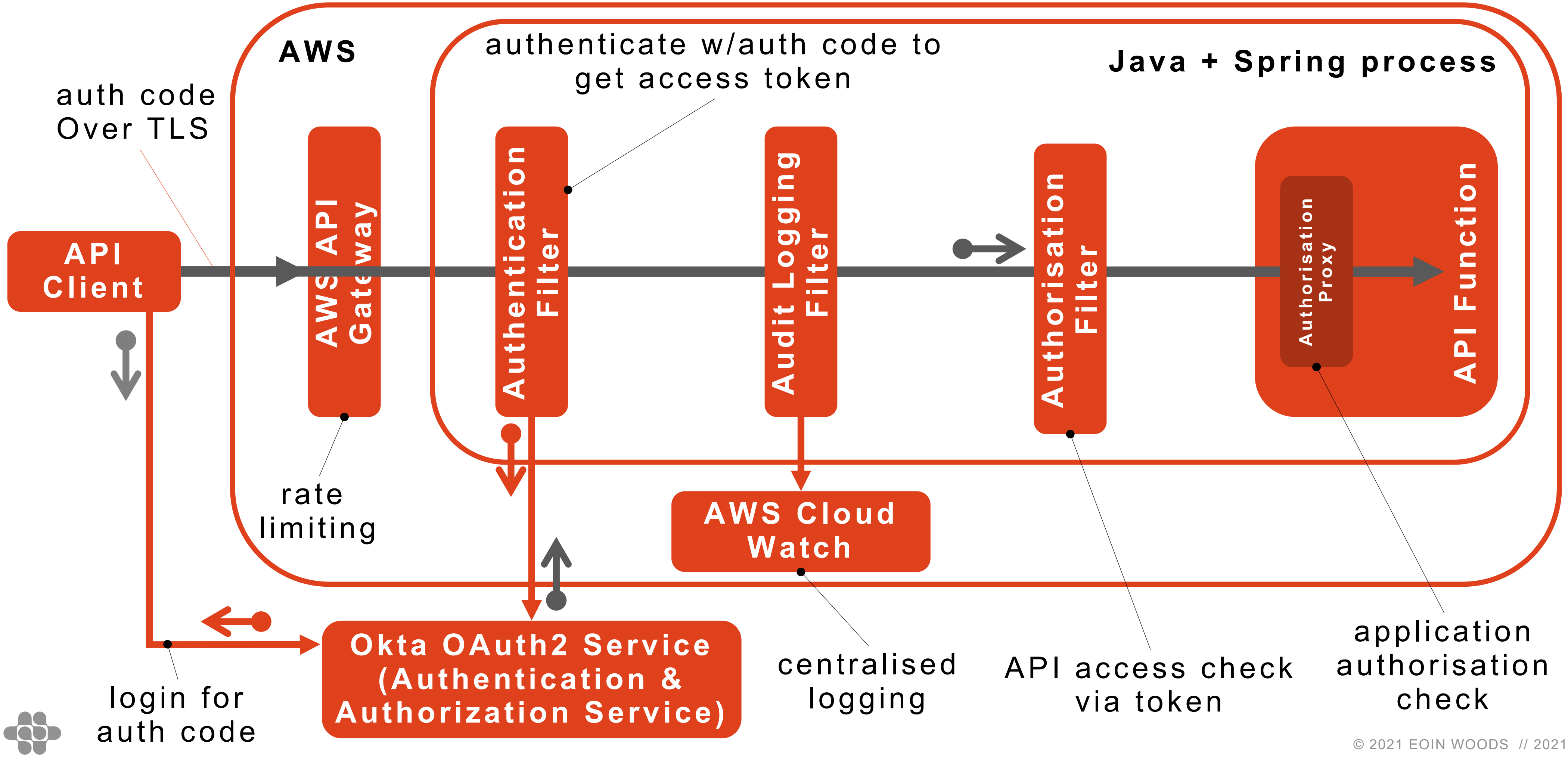
Some Key Aspects of Software Security for Teams



Securing an API



Securing an API - Example



Lots of Choice When Securing an API

Authentication & Authorisation

- Local implementation (users, passwords, groups, ACLs)
- Cloud services (AWS Cognito, Azure AD, …)
- 3rd party services (Auth0, Okta, …)
- Enterprise products (IAM vendors)
- Open source (Keycloak, Gluu, …)

Auditing

- Plain files
- OS logging
- Database records
- Cloud services (CloudWatch, Azure Monitor)
- SIEM (Splunk, Rapid 7, …)

Rate Limiting

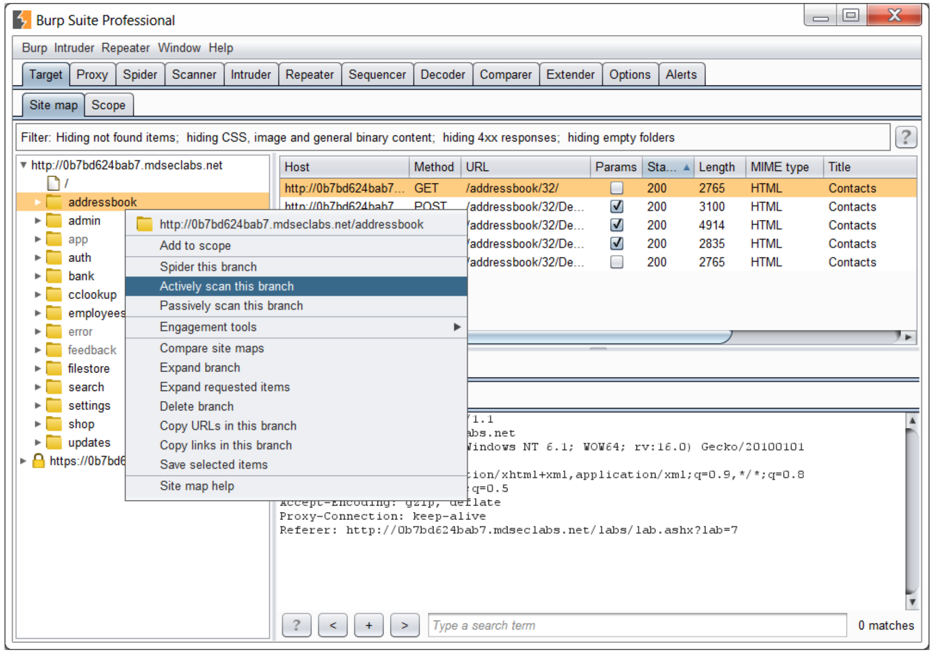
- Cloud API gateways
- Enterprise API gateways
- Reverse proxies
- Open source middleware

Key Tactics

- Don't trust clients
  - authentication, authorisation, validation
- Identify "interpreters" and sanitise inputs, use bind variables, …
  - command lines, database queries, configuration data, …
- Protect valuable information at rest and in transit
  - encryption
- Simplicity
  - Avoid the special cases, make sure the system is understood
- Standardise and Automate
  - consistency, correctness, avoid configuration errors

Tactic: Don’t Trust Clients

- Be wary of everything sent by a client
- Assume possible tampering
  - TLS connections
  - short lived sessions
  - reauthenticate humans, recheck tokens before sensitive operations
  - use opaque tokens for IDs
  - validate everything



Tactic: Watch Out for Injection

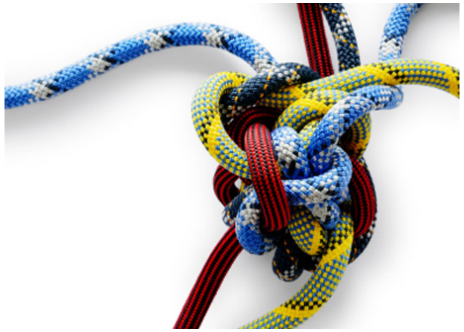
- Many things are interpreters
  - Operating system shells
  - Database query languages
  - Configuration files
  - Parsers
- Assume someone will notice!
  - Avoid using direct string manipulation
    - libraries and bind variables
  - Sanitise strings passed to interpreters
    - 3rd party library (e.g. OWASP)
  - Reject very long strings

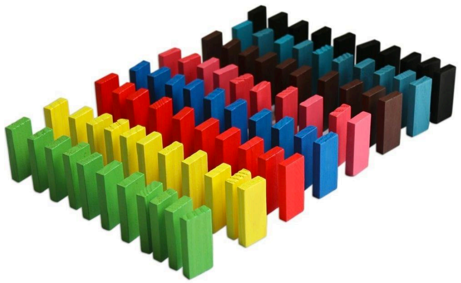
Tactic: Protect Information

- Assume perimeter breach
  - defence in depth
  - encrypt everything possible
- But there are tradeoffs
  - slows everything down
  - querying is difficult
  - Message routing on sensitive fields
  - Manage and rotate keys
  - Complexity added to restore

Tactic: Simplify and Standardise

- Complexity is the enemy of security
  - “you can’t secure what you don’t understand”
  - special cases often forgotten
- Simplify, standardise, automate
  - Simple things easier to check & secure
  - Standardisation removes a lot of special cases
  - Automation removes human inconsistencies avoiding one area of risk





A Few Words on Tools

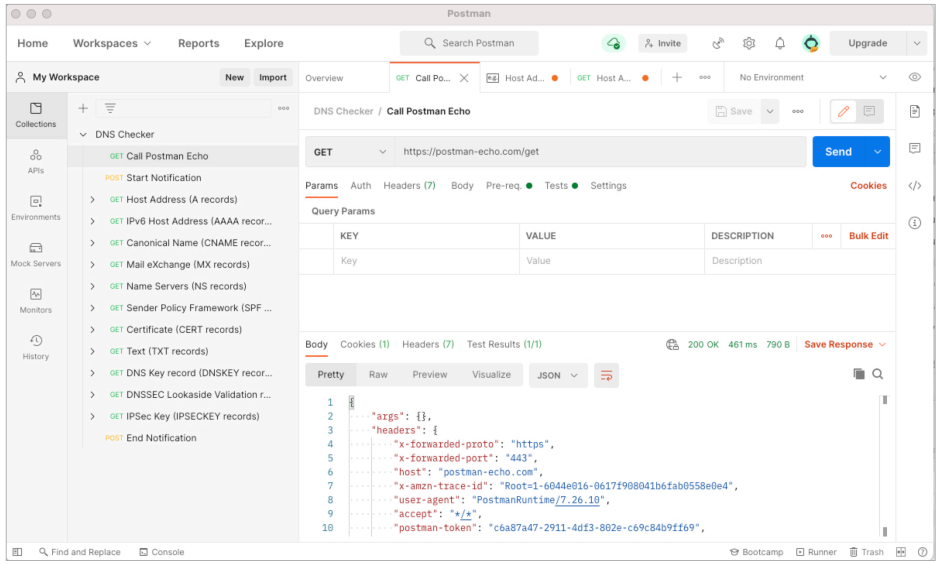
- Security tools are obviously useful
- Many types exist from simple to very complex
- Need make sure people don't view tools as an alternative to thinking!
- Main groups
  - Specialist security scanning tools
  - Interactive tools for penetration and exploratory testing
  - Software composition analysis (open source scanning)

Automated Security Testing

- Automated tools are useful for some types of security problem
  - SAST – static scanning
  - DAST – simulated attacks
  - IAST – agent-based monitoring
  - RASP – runtime security monitoring
- Challenges are false positives and effort to mitigate if used late
- Danger of over-reliance

Postman

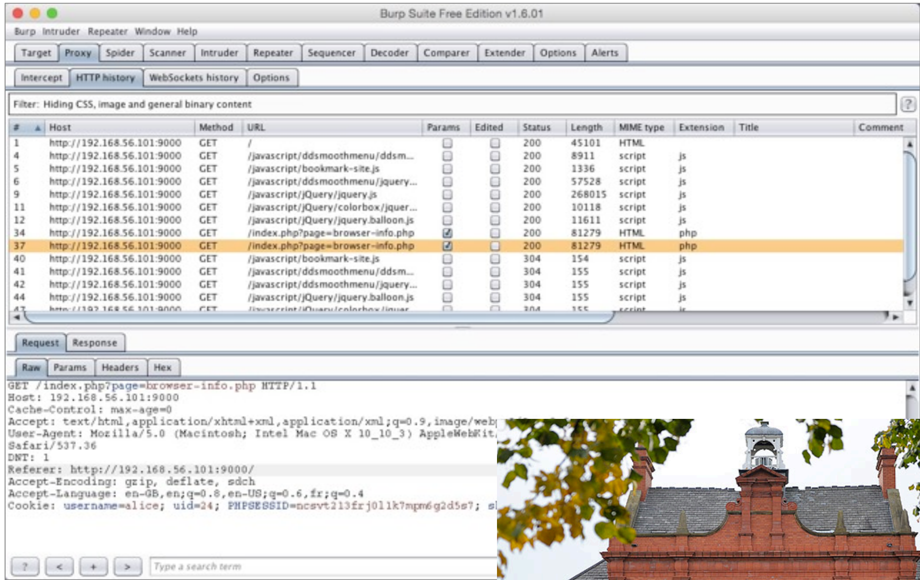
- API development & testing suite
- Popular for functional and security API testing
- Desktop tool with link to cloud service (with a web UI)
- Interactive or command line (via ”Newman” runner extension)



https://www.postman.com/

BurpSuite

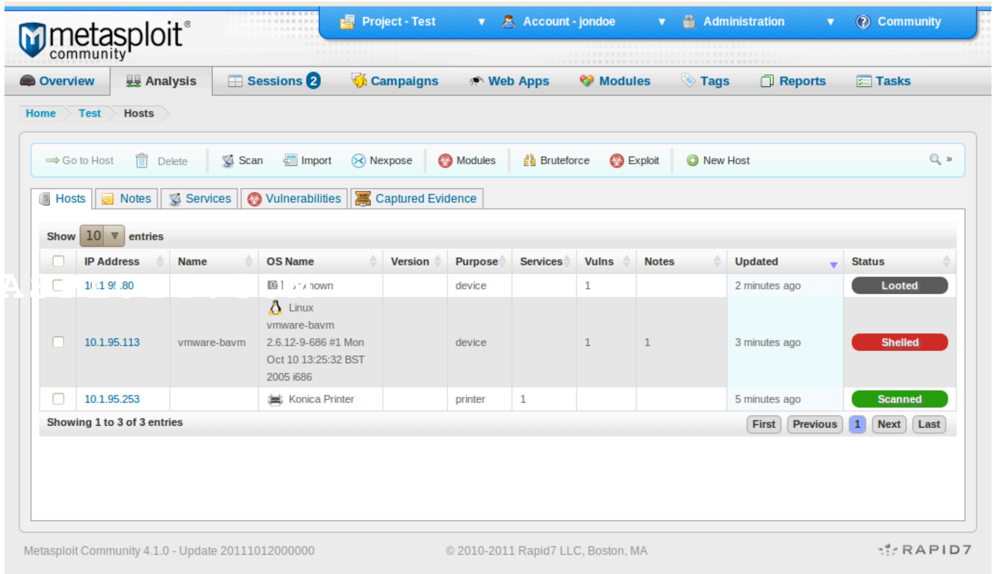
- Proxy, scanning, pentest tool
- Very capable free version
- Fuller commercial version available
- Inspect traffic, manipulate headers and content, replay, spider, …
- Made in Knutsford!



http://portswigger.net/burp

Metasploit

- The pentester's "standard" tool
- Very wide range of capabilities
- Commercial version available

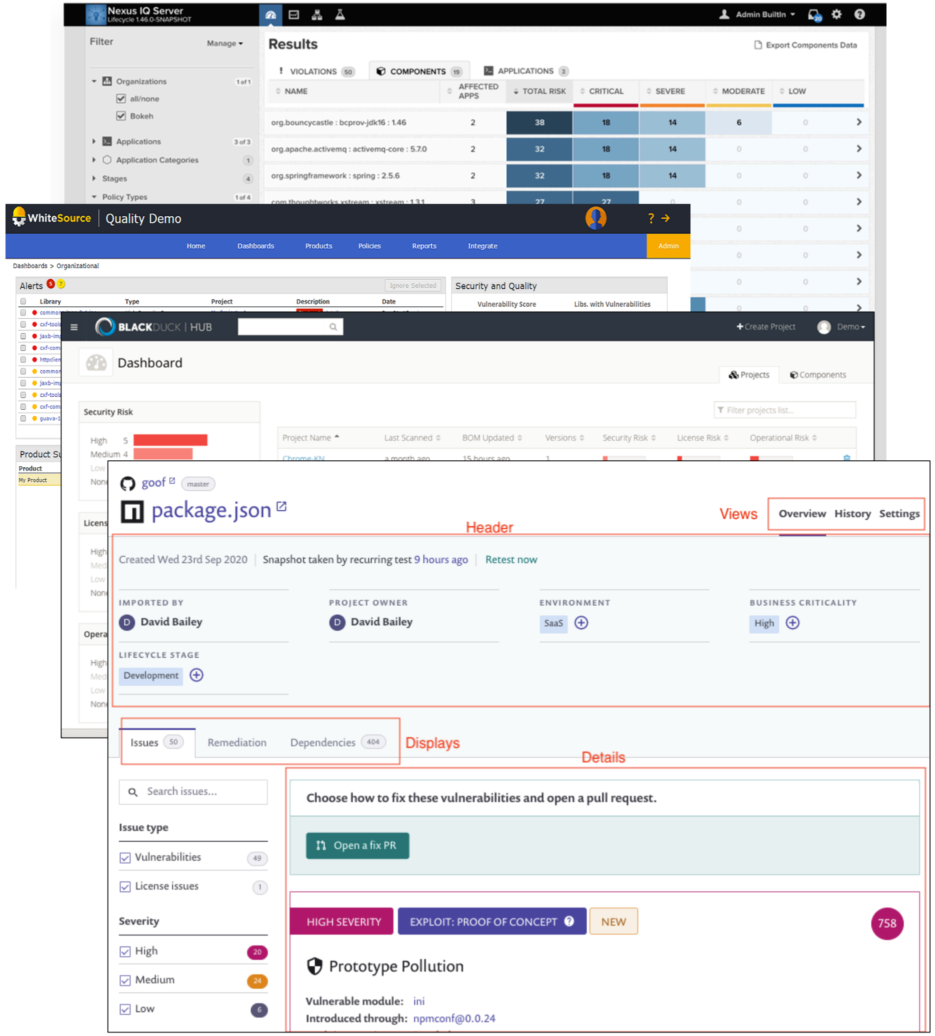


https://www.metasploit.com

Open Source Scanning

- Example commercial tools for OSS security, audit & compliance:
  - BlackDuck
  - Whitesource
  - Sonatype LCM
  - Snyk
- Scan builds identifying open source
- Checks for known vulnerabilities
- Alerts and dashboards for monitoring

```
www.blackduck.com
www.whitesourcesoftware.com
www.sonatype.com/nexus-lifecycle
www.snyk.io
```



5

SUMMARY

OWASP API Top 10 - 2019

#1 Broken Object Authorization
#2 Broken User Authentication
#3 Excessive Data Exposure
#4 Resources & Rate Limiting
#5 Broken Function Authorization
#6 Mass Assignment
#7 Security Misconfiguration
#8 Injection
#9 Improper Asset Management
#10 Insufficient Logging and Monitoring

Key Aspects of API Security

Preventing Injection
- SQL, configuration, commands, ...

Validation
- inputs, outputs, authentication, authorisation

Implementation
- automatic binding, too much data, faulty checking

Environment
- rate limiting, monitoring, logging, configuration

Elements of Securing an API

Rate Limiting

Authentication

Auditing

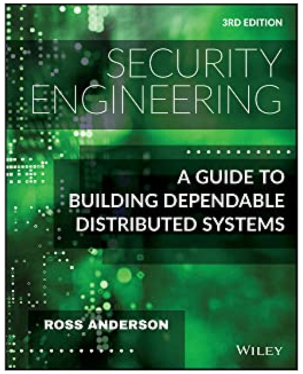
Authorisation

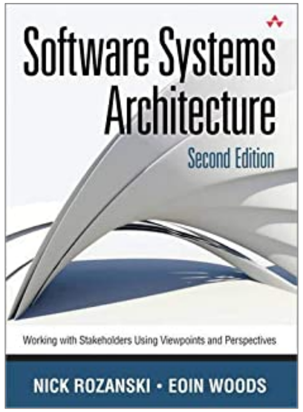
API Function

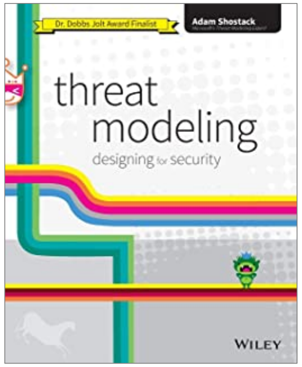
Useful Tactics for API Security

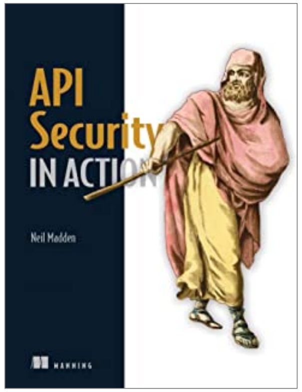
- Don’t trust clients
- Watch out for injection
- Protect information
- Simplify and standardise

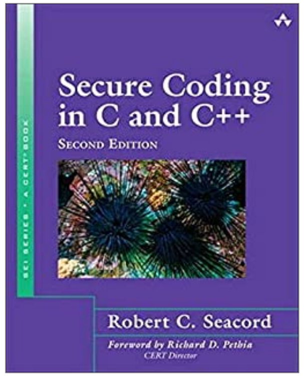
Books

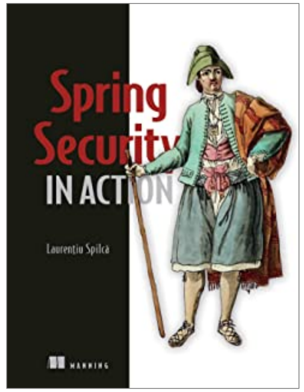

3RD EDITION
SECURITY ENGINEERING
A GUIDE TO BUILDING DEPENDABLE DISTRIBUTED SYSTEMS
ROSS ANDERSON
WILEY

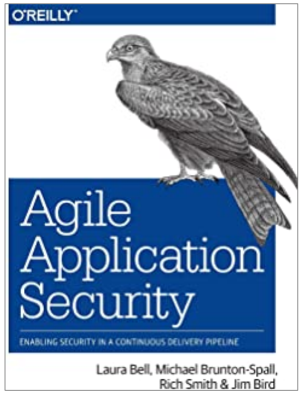

Software Systems Architecture
Second Edition
Working with Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI • EOIN WOODS

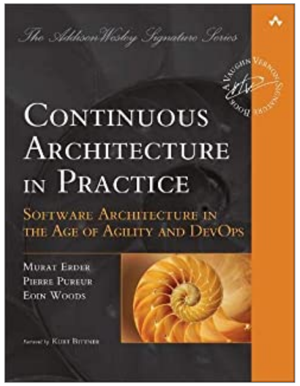

Dr. Dobbs Jolt Award Finalist
Adam Shostack
threat modeling
designing for security
WILEY

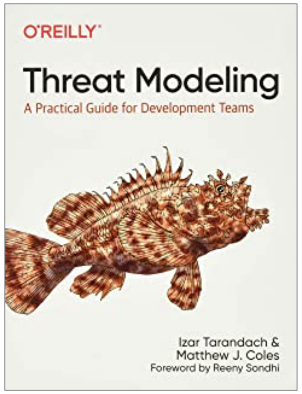

API Security IN ACTION
Neil Madden
MANNING


Secure Coding in C and C++
Second Edition
Robert C. Seacord
Foreword by Richard D. Pethia
CERT Director

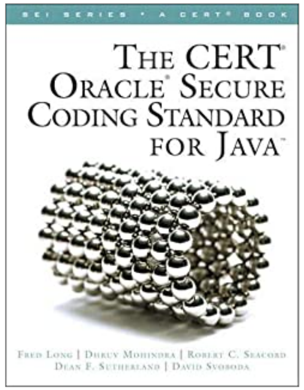

Spring Security IN ACTION
Laurentiu Spilca
MANNING

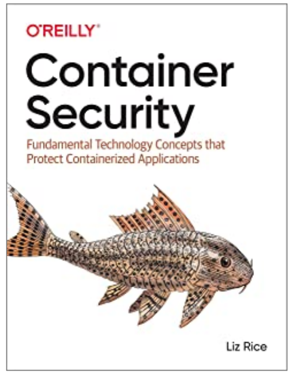

O'REILLY
Agile Application Security
ENABLING SECURITY IN A CONTINUOUS DELIVERY PIPELINE
Laura Bell, Michael Brunton-Spall, Rich Smith & Jim Bird


The Addison-Wesley Signature Series
CONTINUOUS ARCHITECTURE IN PRACTICE
SOFTWARE ARCHITECTURE IN THE AGE OF AGILITY AND DEVOPS
MURAT ERDER
PIERRE PUREUR
EOIN WOODS


O'REILLY
Threat Modeling
A Practical Guide for Development Teams
Izar Tarandach & Matthew J. Coles
Foreword by Reeny Sondhi


Secure by Design
Dan Bergh Johnsson
Daniel Deogun
Daniel Sawano
MANNING


SEI SERIES • A CERT BOOK
THE CERT ORACLE SECURE CODING STANDARD FOR JAVA
FRED LONG | DHRUV MOHINDRA | ROBERT C. SEACORD
DEAN F. SUTHERLAND | DAVID SVOBODA


O'REILLY
Container Security
Fundamental Technology Concepts that Protect Containerized Applications
Liz Rice

endava