OWASP Top 10 2017

David Johansson

Principal Consultant, Synopsys

Presentation material contributed by Andrew van der Stock

David Johansson

- Security consultant with 10 years in AppSec
- Helping clients design and build secure software
- Develop and deliver security training
- Based in London, working for Synopsys

(@securitybits)

OWASP Top 10 RC1

Road to release

WE CAN'T STOP HERE.
THIS IS BAT COUNTRY.
ICANHASCHEEZBURGER.COM

Criticism – valid and invalid

- “Not OWASP like”
  - The new additions *A7 Insufficient Attack Protection* and *A10 Underprotected APIs* boiled down to “failure to buy a tool”
  - From a vendor who sets the standard
  - From a vendor who owns the tool type market
- John Steven and others had ontological issues with the mix of both controls and vulnerabilities (“Define vulnerability. Is that a vulnerability?”)
- Others had problems with the data quality
- Showed us people really care about the OWASP Top 10!

The community called for change!

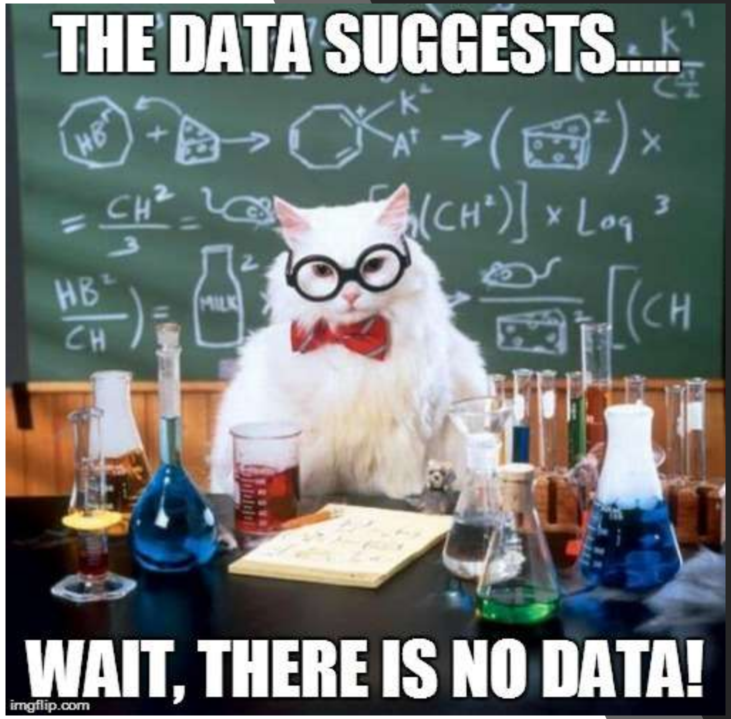
"Make the OWASP Top 10 Great Again"
MAKE THE OWASP TOP TEN GREAT AGAIN
OWASP
Recasting the Problem from Politics to Technical Challenges

AppSec USA 2017 Keynote by Jim Manico and John Steven

Leadership

- Dave Wichers and Jeff Williams stood down
- Handed it over to Andrew van der Stock
- Immediately appointed co-leaders
  - Neil Smithline (participated since 2004)
  - Torsten Gigler (German translator since 2010)
  - And the team added … Brian Glas (data geek)

Data Call

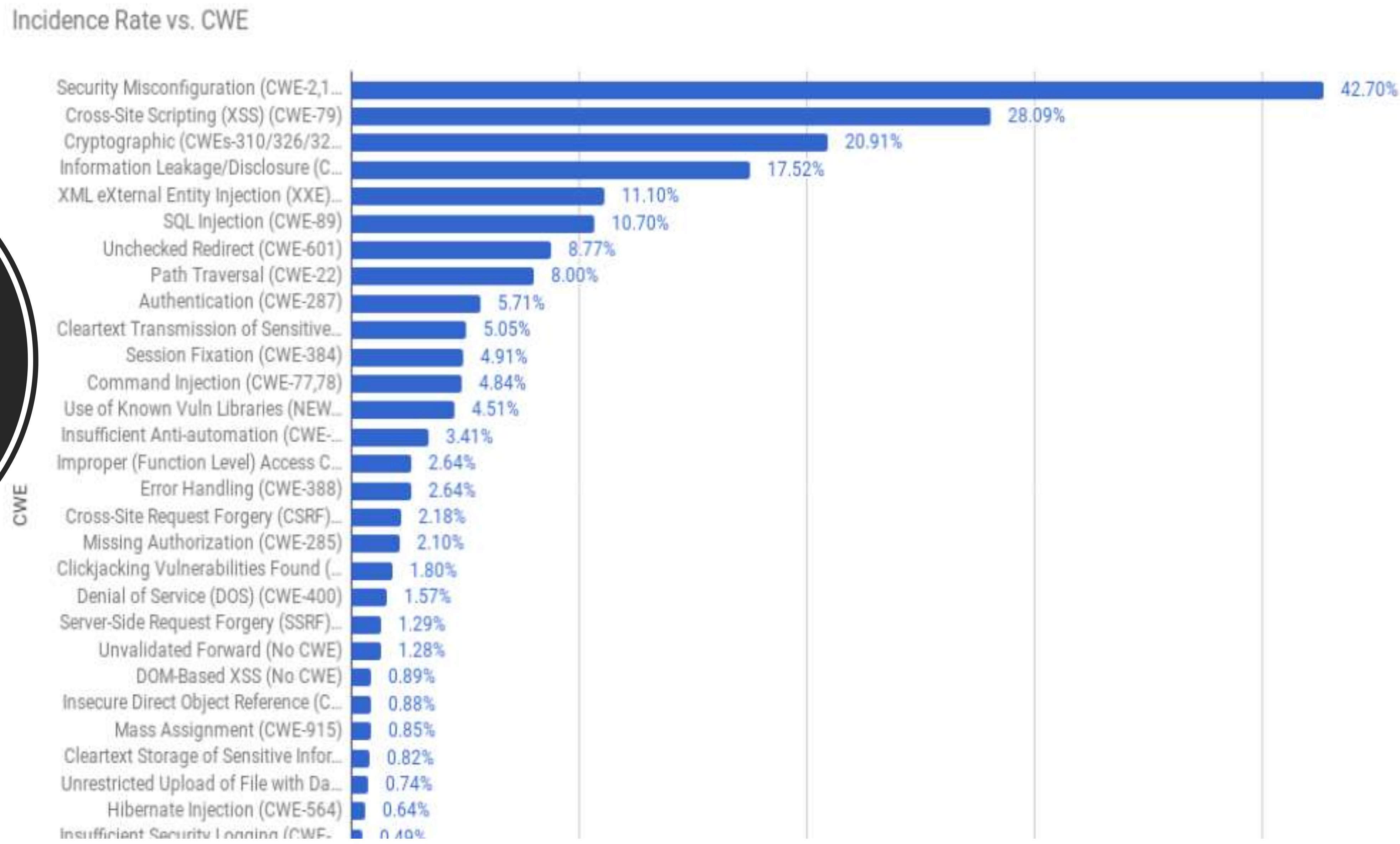


- Need data for 2016
- Need qualitative survey data for two replacements for A7 and A10

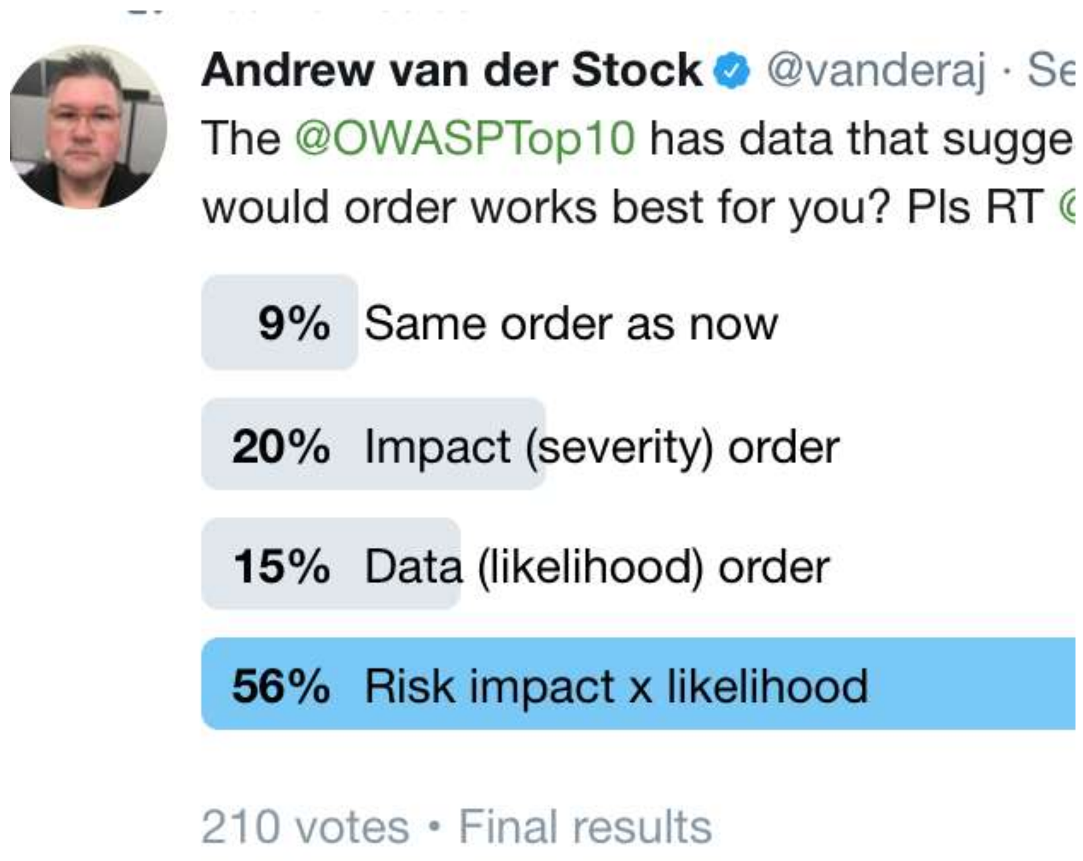
- Brian Glas designed the new survey
- 500+ responses

- Obtained a great deal more data, including from HPE (Fortify), Veracode, CheckMarx, BugCrowd, and Synopsys. Over 114,000 apps form data set.

Data Call Summary



Ordering



Andrew van der Stock @vanderaj · Se

The @OWASPTop10 has data that sugge

would order works best for you? Pls RT @

9%	Same order as now
20%	Impact (severity) order
15%	Data (likelihood) order
56%	Risk impact x likelihood

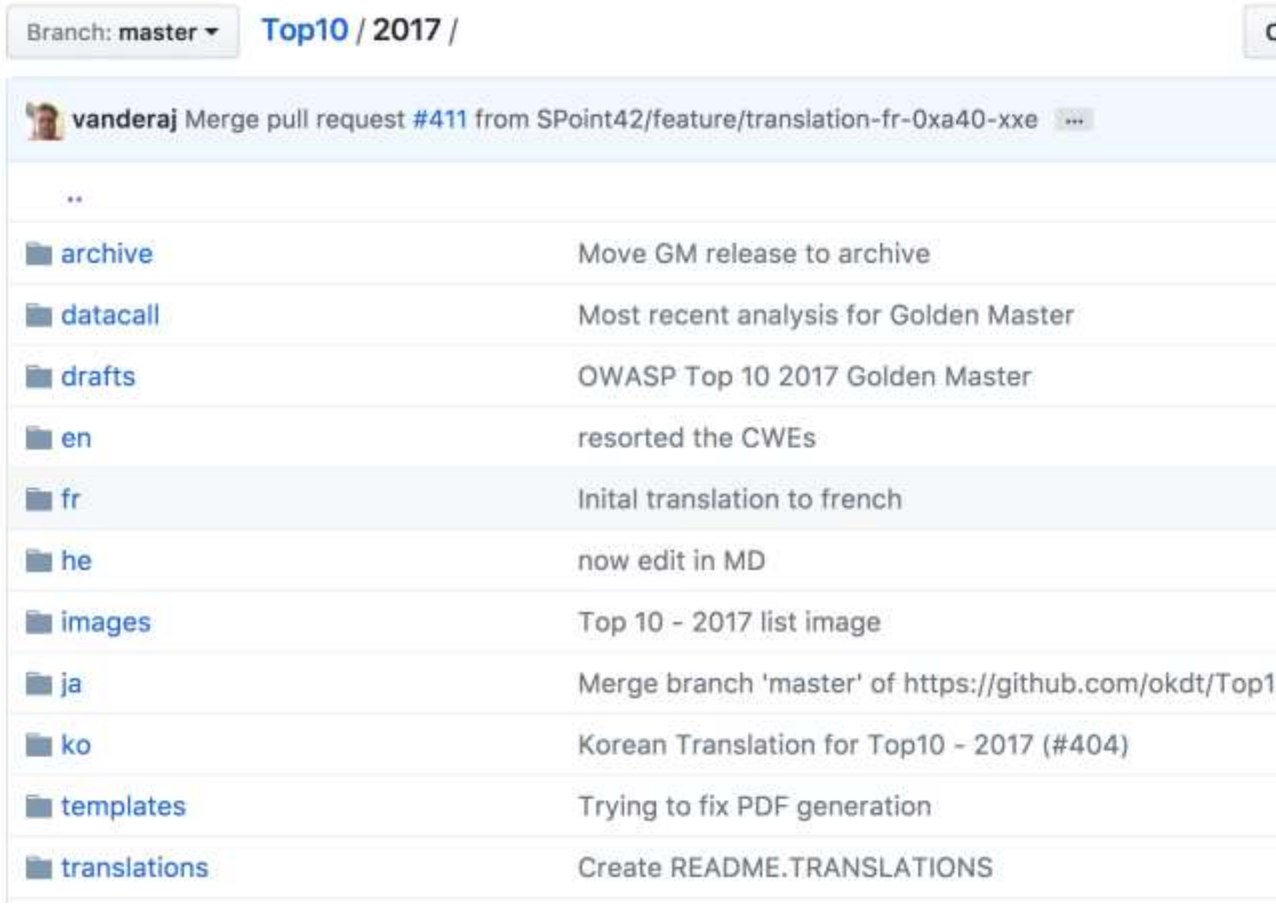
210 votes • Final results

- We ordered in risk (impact x likelihood), which means CVSS x (survey | data)
- Represents our best understanding of 2017 issues

GitHub

- Everything is in GitHub

- Open: Moved to GitHub
- Open: Data and analysis
- Traceable: Issues
- Translatable: Markdown



Branch: master ▾ Top10 / 2017 /

vanderaj Merge pull request #411 from SPoint42/feature/translation-fr-0xa40-xxe …

..

archive	Move GM release to archive
datacall	Most recent analysis for Golden Master
drafts	OWASP Top 10 2017 Golden Master
en	resorted the CWEs
fr	Inital translation to french
he	now edit in MD
images	Top 10 - 2017 list image
ja	Merge branch 'master' of https://github.com/okdt/Top1
ko	Korean Translation for Top10 - 2017 (#404)
templates	Trying to fix PDF generation
translations	Create README.TRANSLATIONS

What’s new

OWASP Top 10 2017

OWASP Top 10 2013 → 2017

OWASP
Open Web Application Security Project

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 - Injection	→	A1:2017-Injection
A2 - Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 - Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 - Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 - Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 - Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 - Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 - Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 - Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 - Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

A8 :2017

Insecure Deserialization

Threat Agents
Attack Vectors
Security Weakness
Impacts

App. Specific	Exploitability: 1	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks to the underlying exploit code.		This issue is included in the Top 10 based on an industry survey and not on quantifiable data. Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as tooling is developed to help identify and address it.		The impact of deserialization flaws cannot be understated. These flaws can lead to remote code execution attacks, one of the most serious attacks possible. The business impact depends on the protection needs of the application and data.	

Is the Application Vulnerable?

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker.

This can result in two primary types of attacks:

How to Prevent

The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.

Is the Application Vulnerable?

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker.

This can result in two primary types of attacks:

- Object and data structure related attacks where the attacker modifies application logic or achieves arbitrary remote code execution if there are classes available to the application that can change behavior during or after deserialization.
- Typical data tampering attacks, such as access-control-related attacks, where existing data structures are used but the content is changed.

Serialization may be used in applications for:

- Remote- and inter-process communication (RPC/IPC)
- Wire protocols, web services, message brokers
- Caching/Persistence
- Databases, cache servers, file systems
- HTTP cookies, HTML form parameters, API authentication tokens

How to Prevent

The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.

If that is not possible, consider one of more of the following:

- Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.
- Enforcing strict type constraints during deserialization before object creation as the code typically expects a definable set of classes. Bypasses to this technique have been demonstrated, so reliance solely on this is not advisable.
- Isolating and running code that deserializes in low privilege environments when possible.
- Logging deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitoring deserialization, alerting if a user deserializes constantly.

Example Attack Scenarios

Scenario #1: A React application calls a set of Spring Boot

References

OWASP

- Caching/Persistence
- Databases, cache servers, file systems
- HTTP cookies, HTML form parameters, API authentication tokens

Example Attack Scenarios

Scenario #1: A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing user state and passing it back and forth with each request. An attacker notices the "R00" Java object signature, and uses the Java Serial Killer tool to gain remote code execution on the application server.

Scenario #2: A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:

```
 a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker changes the serialized object to give themselves admin privileges:

```
 a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

- Logging deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitoring deserialization, alerting if a user deserializes constantly.

References

OWASP

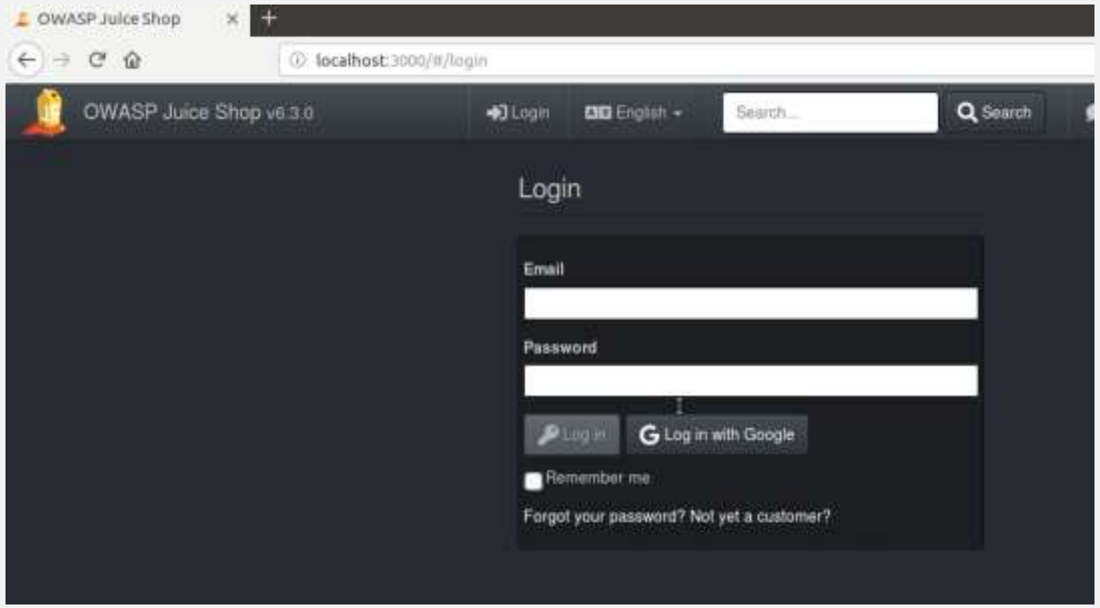
- OWASP Cheat Sheet: Deserialization
- OWASP Proactive Controls: Validate All Inputs
- OWASP Application Security Verification Standard
- OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse
- OWASP AppSecUSA 2017: Friday the 13th JSON Attacks

External

- CWE-502: Deserialization of Untrusted Data
- Java Unmarshaller Security
- OWASP AppSec Cali 2015: Marshalling Pickles

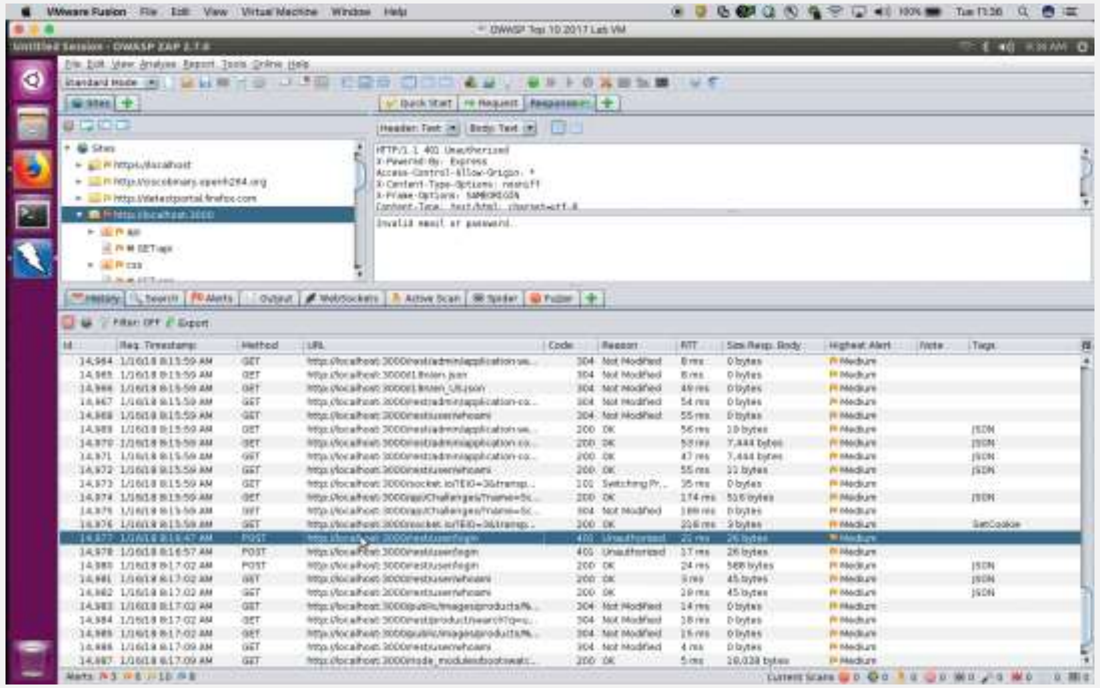
A1:2017 Injections

- SQL injection
- NoSQL injection
- OS Command Injection
- LDAP Injection
- <insert injection here>



A2:2017 Insecure Authentication

- NIST 800-63 Alignment
- Two factor authentication
- Anti-automation
- Credential Stuffing
- Brute forcing and dictionary attacks



Sites
- https://localhost
- http://ocsp.digicert.com
- http://detectportal.firefox.com
- http://localhost:3000
  - api
  - GET:api
  - css

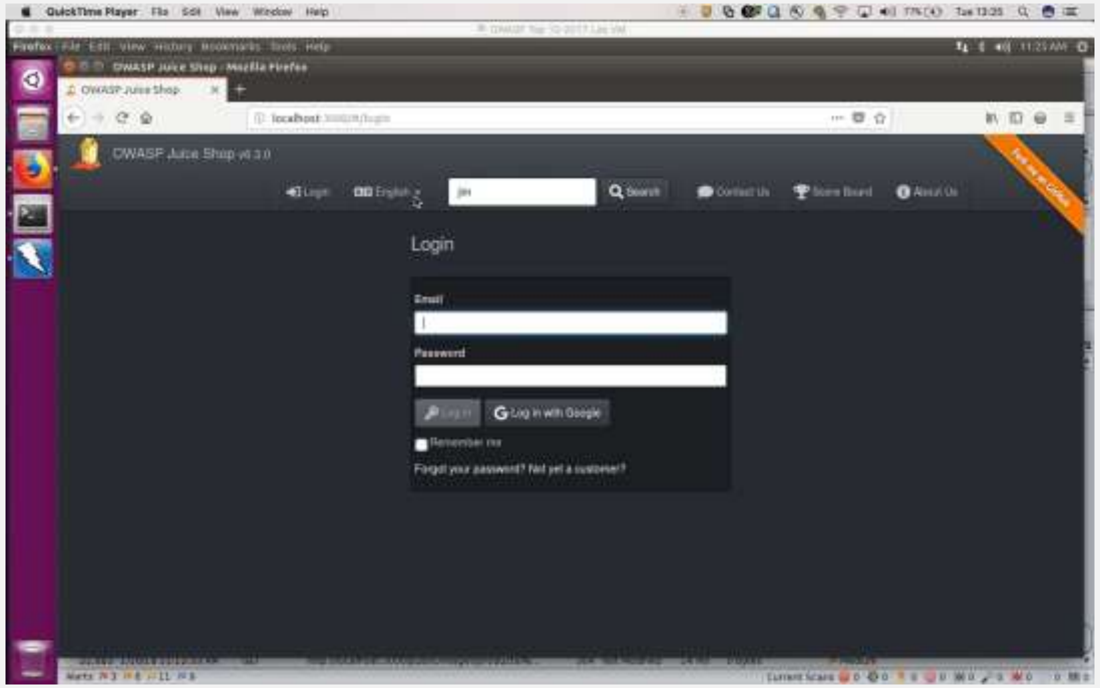
```
HTTP/1.1 401 Unauthorized
X-Powered-By: Express
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Content-Type: text/html; charset=utf-8

Invalid email or password.
```

Id	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Body	Highest Alert	Note	Tags
14,964	1/16/18 8:15:59 AM	GET	http://localhost:3000/rest/admin/application-ve...	304	Not Modified	8 ms	0 bytes	Medium		
14,965	1/16/18 8:15:59 AM	GET	http://localhost:3000/i18n/en.json	304	Not Modified	8 ms	0 bytes	Medium		
14,966	1/16/18 8:15:59 AM	GET	http://localhost:3000/i18n/en_US.json	304	Not Modified	49 ms	0 bytes	Medium		
14,967	1/16/18 8:15:59 AM	GET	http://localhost:3000/rest/admin/application-co...	304	Not Modified	54 ms	0 bytes	Medium		
14,968	1/16/18 8:15:59 AM	GET	http://localhost:3000/rest/user/whoami	304	Not Modified	55 ms	0 bytes	Medium		
14,969	1/16/18 8:15:59 AM	GET	http://localhost:3000/rest/admin/application-ve...	200	OK	56 ms	19 bytes	Medium		JSON
14,970	1/16/18 8:15:59 AM	GET	http://localhost:3000/rest/admin/application-co...	200	OK	58 ms	7,444 bytes	Medium		JSON
14,971	1/16/18 8:15:59 AM	GET	http://localhost:3000/rest/admin/application-co...	200	OK	47 ms	7,444 bytes	Medium		JSON
14,972	1/16/18 8:15:59 AM	GET	http://localhost:3000/rest/user/whoami	200	OK	55 ms	11 bytes	Medium		JSON
14,973	1/16/18 8:15:59 AM	GET	http://localhost:3000/socket.io/?EIO=3&transp...	101	Switching Pr...	35 ms	0 bytes	Medium		
14,974	1/16/18 8:15:59 AM	GET	http://localhost:3000/api/Challenges/?name=Sc...	200	OK	174 ms	516 bytes	Medium		JSON
14,975	1/16/18 8:15:59 AM	GET	http://localhost:3000/api/Challenges/?name=Sc...	304	Not Modified	189 ms	0 bytes	Medium		
14,976	1/16/18 8:15:59 AM	GET	http://localhost:3000/socket.io/?EIO=3&transp...	200	OK	206 ms	3 bytes	Medium		SetCookie
14,977	1/16/18 8:16:47 AM	POST	http://localhost:3000/rest/user/login	401	Unauthorized	20 ms	26 bytes	Medium		
14,978	1/16/18 8:16:57 AM	POST	http://localhost:3000/rest/user/login	401	Unauthorized	17 ms	26 bytes	Medium		
14,980	1/16/18 8:17:02 AM	POST	http://localhost:3000/rest/user/login	200	OK	24 ms	588 bytes	Medium		JSON
14,981	1/16/18 8:17:02 AM	GET	http://localhost:3000/rest/user/whoami	200	OK	9 ms	45 bytes	Medium		JSON
14,982	1/16/18 8:17:02 AM	GET	http://localhost:3000/rest/user/whoami	200	OK	19 ms	45 bytes	Medium		JSON
14,983	1/16/18 8:17:02 AM	GET	http://localhost:3000/public/images/products/%...	304	Not Modified	14 ms	0 bytes	Medium		
14,984	1/16/18 8:17:02 AM	GET	http://localhost:3000/rest/product/search?q=u...	304	Not Modified	18 ms	0 bytes	Medium		
14,985	1/16/18 8:17:02 AM	GET	http://localhost:3000/public/images/products/%...	304	Not Modified	15 ms	0 bytes	Medium		
14,986	1/16/18 8:17:09 AM	GET	http://localhost:3000/rest/user/whoami	304	Not Modified	4 ms	0 bytes	Medium		
14,987	1/16/18 8:17:09 AM	GET	http://localhost:3000/node_modules/bootstrap...	200	OK	5 ms	19,028 bytes	Medium		

Alerts 3 8 10 8 Current Scans 0 0 0 0 0 0 0 0

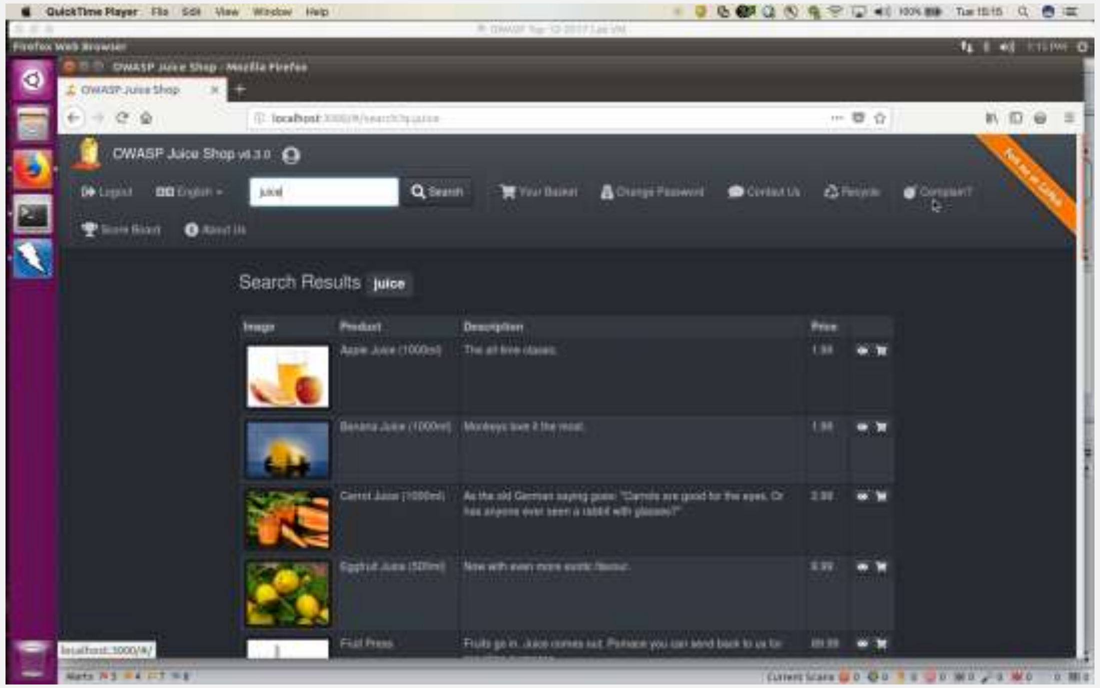
A3:2017 Sensitive Data Exposure

- Focuses on data breaches
- Sensitive, private, health, financial data
- Aligned with GDPR and privacy laws


OWASP Juice Shop - Mozilla Firefox
OWASP Juice Shop
Login
Email
Password
Log in with Google
Remember me
Forgot your password? Not yet a customer?

A4:2017 XXE

- NEW FOR 2017!
- One of the most under-tested issues
- … but only new issue that had sufficient data
- Learn, test, report, fix!

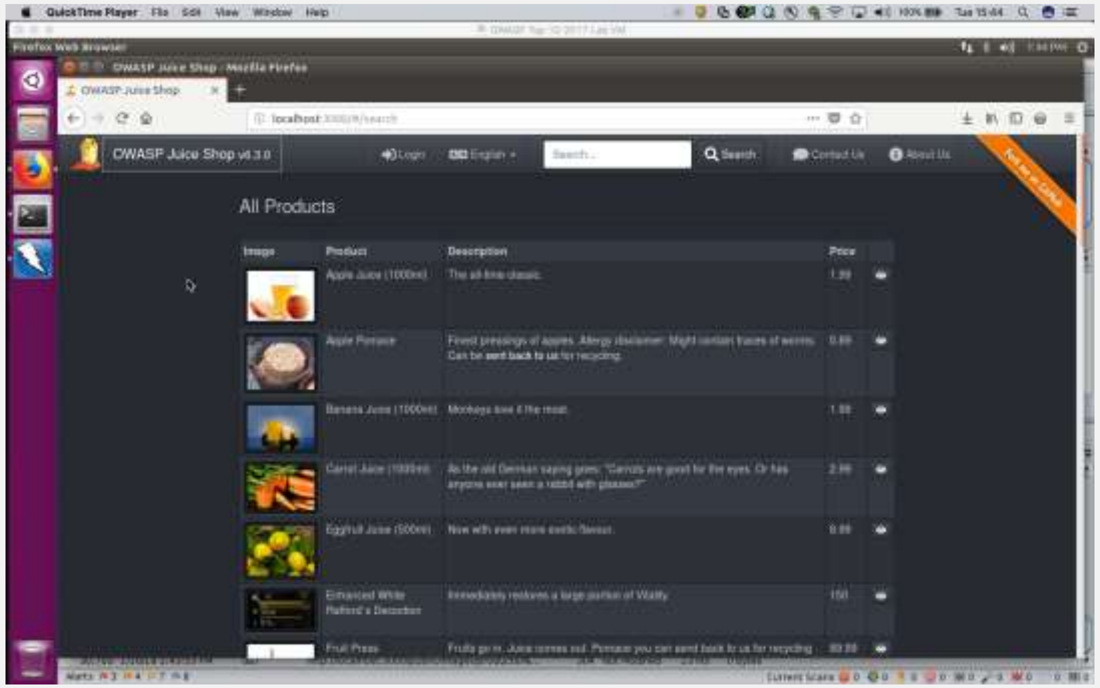


Search Results juice

Image	Product	Description	Price	
	Apple Juice (1000ml)	The all time classic.	1.99	
	Banana Juice (1000ml)	Monkeys love it the most.	1.99	
	Carrot Juice (1000ml)	As the old German saying goes: "Carrots are good for the eyes. Or has anyone ever seen a rabbit with glasses?"	2.99	
	Eggfruit Juice (500ml)	Now with even more exotic flavour.	8.99	
	Fruit Press	Fruits go in. Juice comes out. Pomace you can send back to us for	89.99	

A5:2017 Broken Access Control

- Insecure Direct Object Reference (IDOR)
- Force browsing
- Presentation layer access control
- Controller (“function”) layer access control
- Model layer access control
- Domain access control – business logic


OWASP Juice Shop v6.3.0

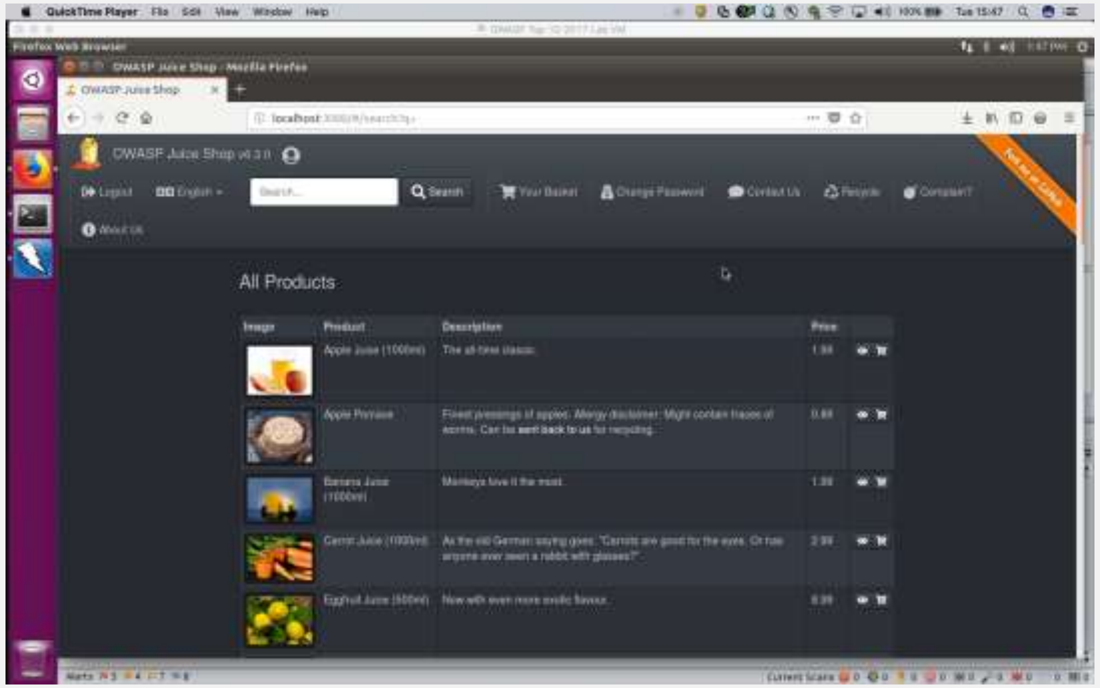
All Products

Image	Product	Description	Price
	Apple Juice (1000ml)	The all-time classic.	1.99
	Apple Pomace	Finest pressings of apples. Allergy disclaimer: Might contain traces of worms. Can be **sent back to us** for recycling.	0.89
	Banana Juice (1000ml)	Monkeys love it the most.	1.99
	Carrot Juice (1000ml)	As the old German saying goes: "Carrots are good for the eyes. Or has anyone ever seen a rabbit with glasses?"	2.99
	Eggfruit Juice (500ml)	Now with even more exotic flavour.	8.99
	Enhanced White Rafford's Decoction	Immediately restores a large portion of Vitality.	150
	Fruit Press	Fruits go in. Juice comes out. Pomace you can send back to us for recycling	89.99

A6:2017 Security Misconfiguration

- S3 buckets, MongoDB, etc.
- Directory listings
- All the passive findings

- Risk rate as per the sensitivity!



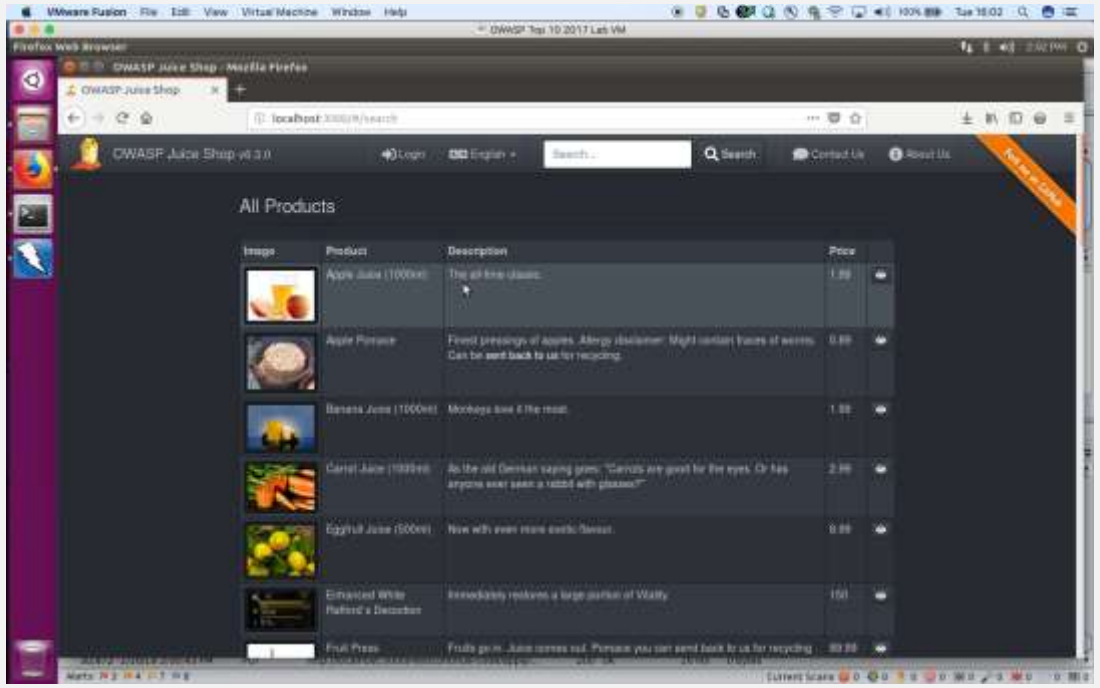
All Products

Image	Product	Description	Price
	Apple Juice (1000ml)	The all-time classic.	1.99
	Apple Pomace	Finest pressings of apples. Allergy disclaimer: Might contain traces of worms. Can be sent back to us for recycling.	0.89
	Banana Juice (1000ml)	Monkeys love it the most.	1.99
	Carrot Juice (1000ml)	As the old German saying goes: "Carrots are good for the eyes. Or has anyone ever seen a rabbit with glasses?"	2.99
	Eggfruit Juice (500ml)	Now with even more exotic flavour.	8.99

A7:2017 XSS

- All your old favorites!

- Now with extra focus on DOM XSS
- Reflected XSS
- Stored XSS

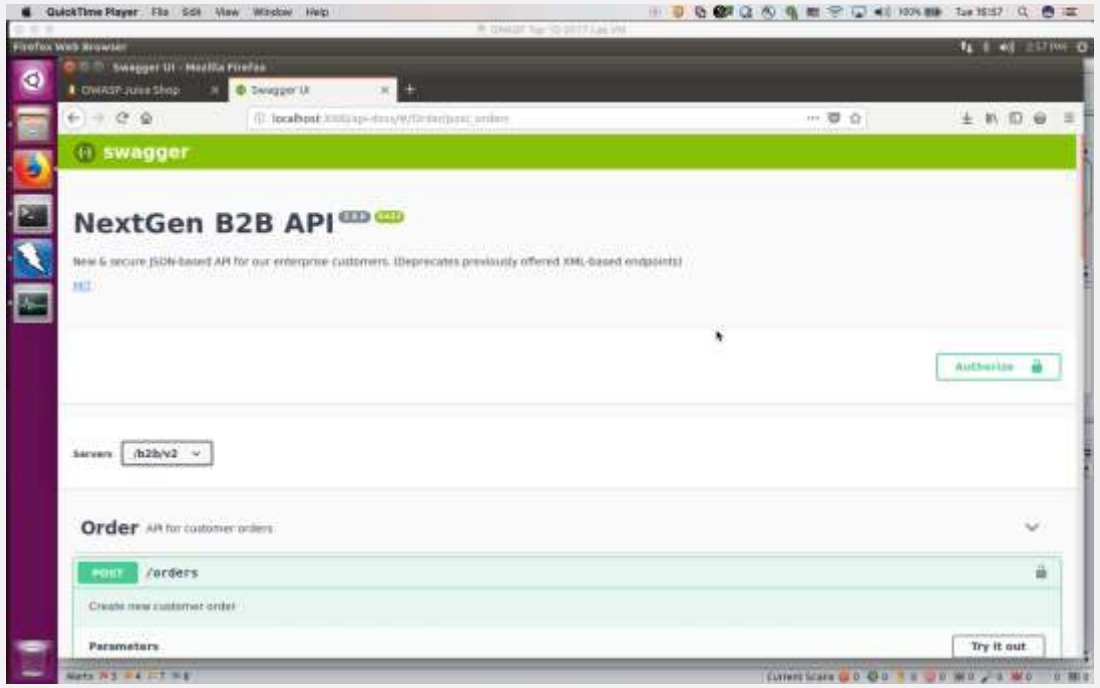


All Products

Image	Product	Description	Price	
	Apple Juice (1000ml)	The all-time classic.	1.99	
	Apple Pomace	Finest pressings of apples. Allergy disclaimer: Might contain traces of worms. Can be **sent back to us** for recycling.	0.89	
	Banana Juice (1000ml)	Monkeys love it the most.	1.99	
	Carrot Juice (1000ml)	As the old German saying goes: "Carrots are good for the eyes. Or has anyone ever seen a rabbit with glasses?"	2.99	
	Eggfruit Juice (500ml)	Now with even more exotic flavour.	8.99	
	Enhanced White Rafford's Decoction	Immediately restores a large portion of Vitality.	150	
	Fruit Press	Fruits go in. Juice comes out. Pomace you can send back to us for recycling.	89.99	

A8:2017 Insecure deserialization

- New for 2017 (Community)
- Deserialization discovered – and this section written - by Chris Frohoff and Gabe Lawrence (!!)
- Learn, Test, Report, Fix!



NextGen B2B API

New & secure JSON-based API for our enterprise customers. (Deprecates previously offered XML-based endpoints)

Servers /b2b/v2

Order API for customer orders

POST /orders

Create new customer order

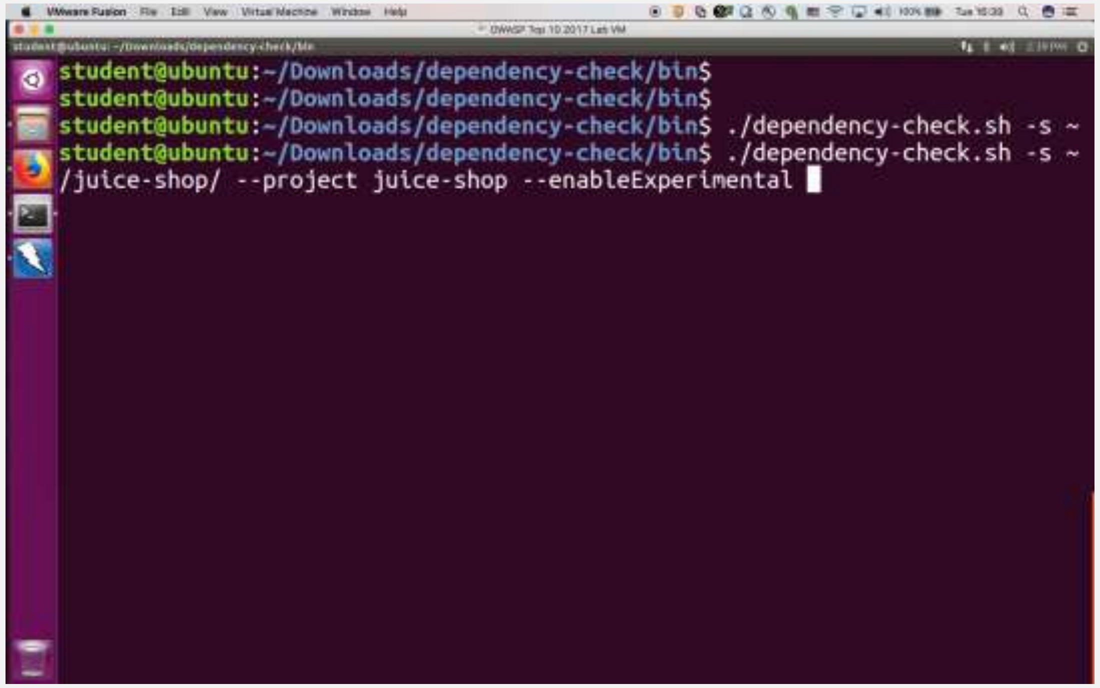
Parameters

A9:2017 Known vulnerable components

- It's still here!

- Automatically report with CI/CD dependency checkers
- Warn if outdated, break if vulnerable

- Let's see if we can retire this in 3 years!



```
student@ubuntu:~/Downloads/dependency-check/bin$
student@ubuntu:~/Downloads/dependency-check/bin$
student@ubuntu:~/Downloads/dependency-check/bin$ ./dependency-check.sh -s ~
student@ubuntu:~/Downloads/dependency-check/bin$ ./dependency-check.sh -s ~
/juice-shop/ --project juice-shop --enableExperimental
```

A10:2017 Insufficient Logging and Monitoring

- NEW for 2017 (Survey)
- Average time to discover pwnage: > 190 days
- Usually reported by external third party
- Usually costs > $1m and often a lot more

- This **is** a missing or ineffective control.

- Testing for this is pretty straight forward – talk to your operations team or look in your SIEM

- Detected?
- Would action be taken?
- Would escalation have occurred?

- Minimize time to detect and respond

Time to upskill and continuously improve

- OWASP Top 10 2017 is different
  - Update skills
  - Update test plans
  - Update tools
  - Update scan policies

In particular, A3, A8 and A10 are very different. No tool can adequately capture all 10 risks

Get ready for 2020

- Get ready for the next release!

- Look at the data we collected this time around
- It’s 2018 already. Start automating that data collection!

- Please consider donating your stats to OWASP!

David Johansson

@securitybits

David.Johansson@synopsys.com

Thank you!

And thanks to Andrew van der Stock who contributed the presentation material.