Mitigating at Design Time

InfoSeCon 2019 - Shaun Lamb

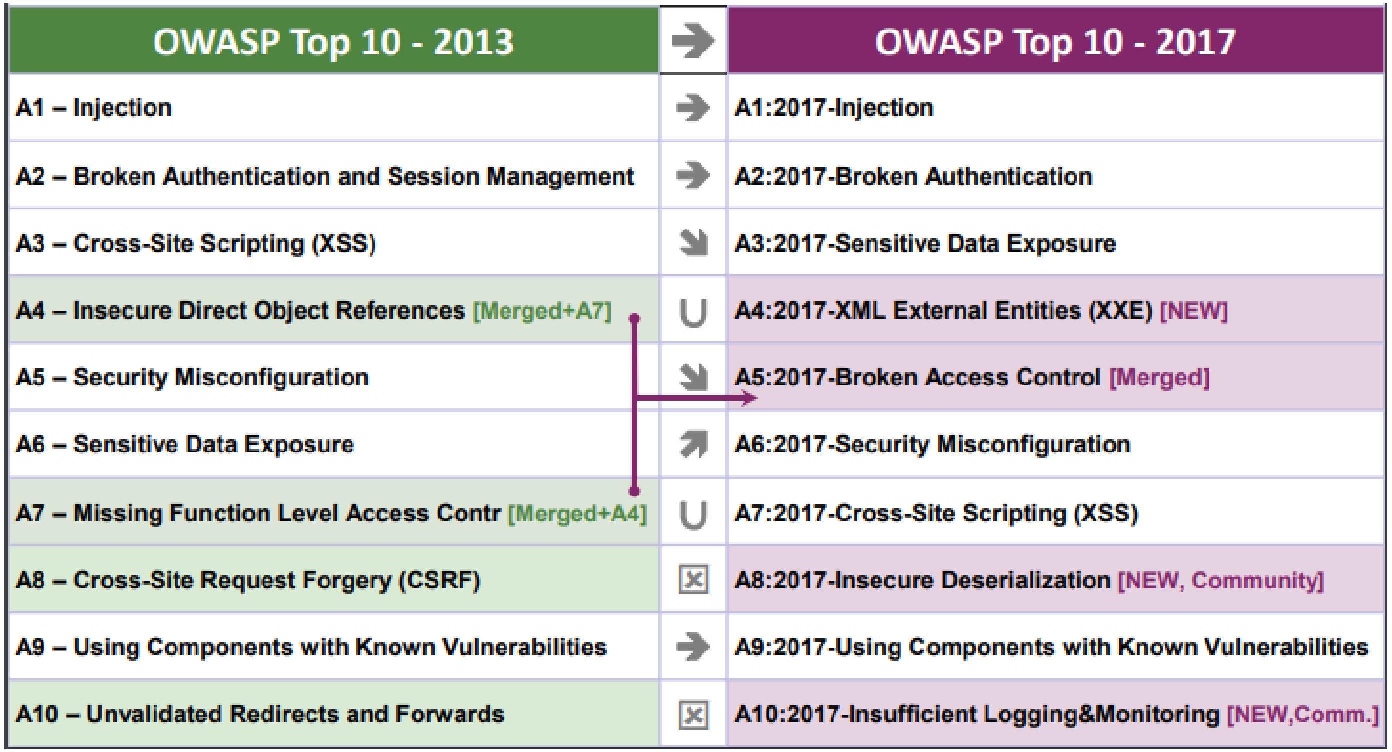
Vulnerability Management Whac-a-Mole

What is a Secure Application Architecture?

Imagine Security Made all decisions for a new application

- A secure application architecture not only prevents vulnerabilities in the initial release but also reduces the frequency of security issues being introduced into subsequent release candidates.

Scope



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.]

Mitigation Strategies

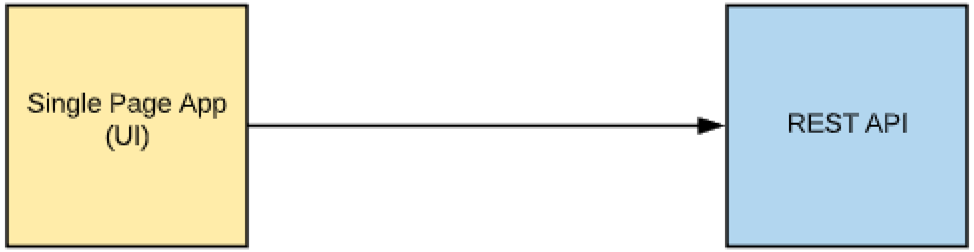
- Security without thinking
  - Secure by default for developers
  - Technology Stack Choices for automatic mitigations
- Avoidance as a mitigation strategy
  - Alternative Approaches
- DevSecOps in Kubernetes (production self-service for devs but with guardrails)
  - Containing the Containers
  - Gating APIs
  - Continuous Security Feedback

Security without Thinking

Secure By Default – No nagging required

XSS

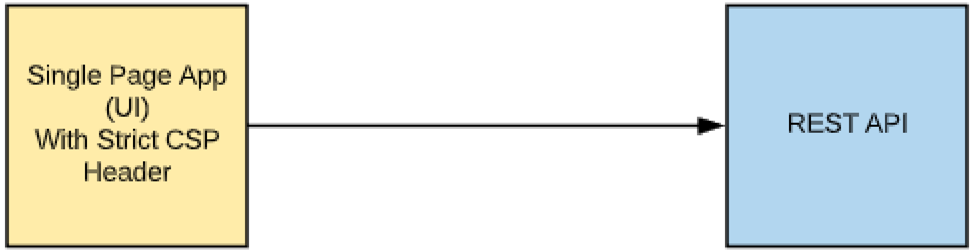
- Traditionally, manually output encode every single JSP/ASP file
- Traditional back up option: input validate (weak)
- Two tiered architecture
  - Angular static UI
  - Output encodes automatically



XSS Mitigation with Content Security Policy

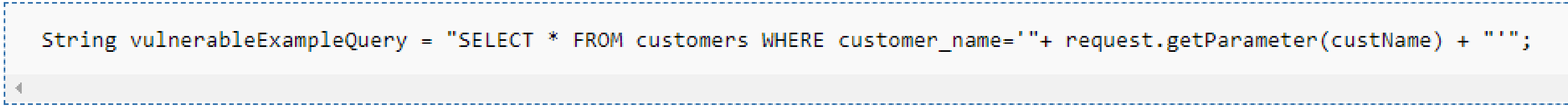
Single Page Apps

- Http Response header that whitelists allowed behavior for web page
- Enforced by browser
- Unobtrusive Javascript
  - No inline JS
  - Html goes in .html files, Javascript goes in .js files, css in .css files
  - Allows restrictive Content Security Policy Header

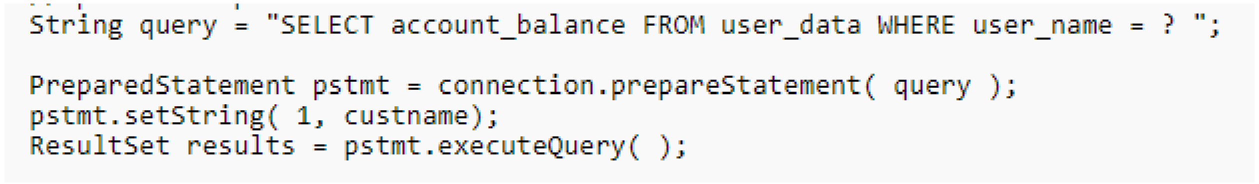


SQL Injection

- Traditional mitigation: parameterize queries
  - String concatenation in query creation (red flag)
- Traditional backup mitigation option: Encode or Validate



```
String vulnerableExampleQuery = "SELECT * FROM customers WHERE customer_name='"+ request.getParameter(custName) + "'";
```

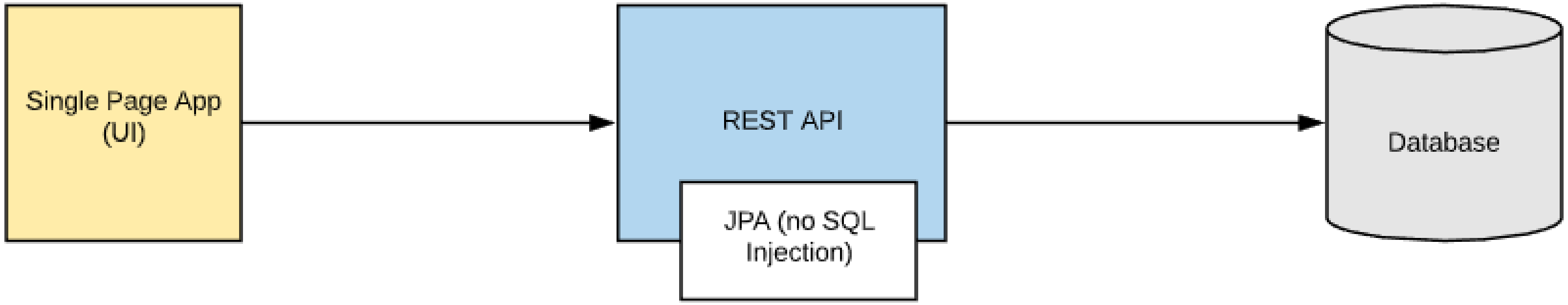


```
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";

PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

SQL Injection

Secure By Default – No nagging required

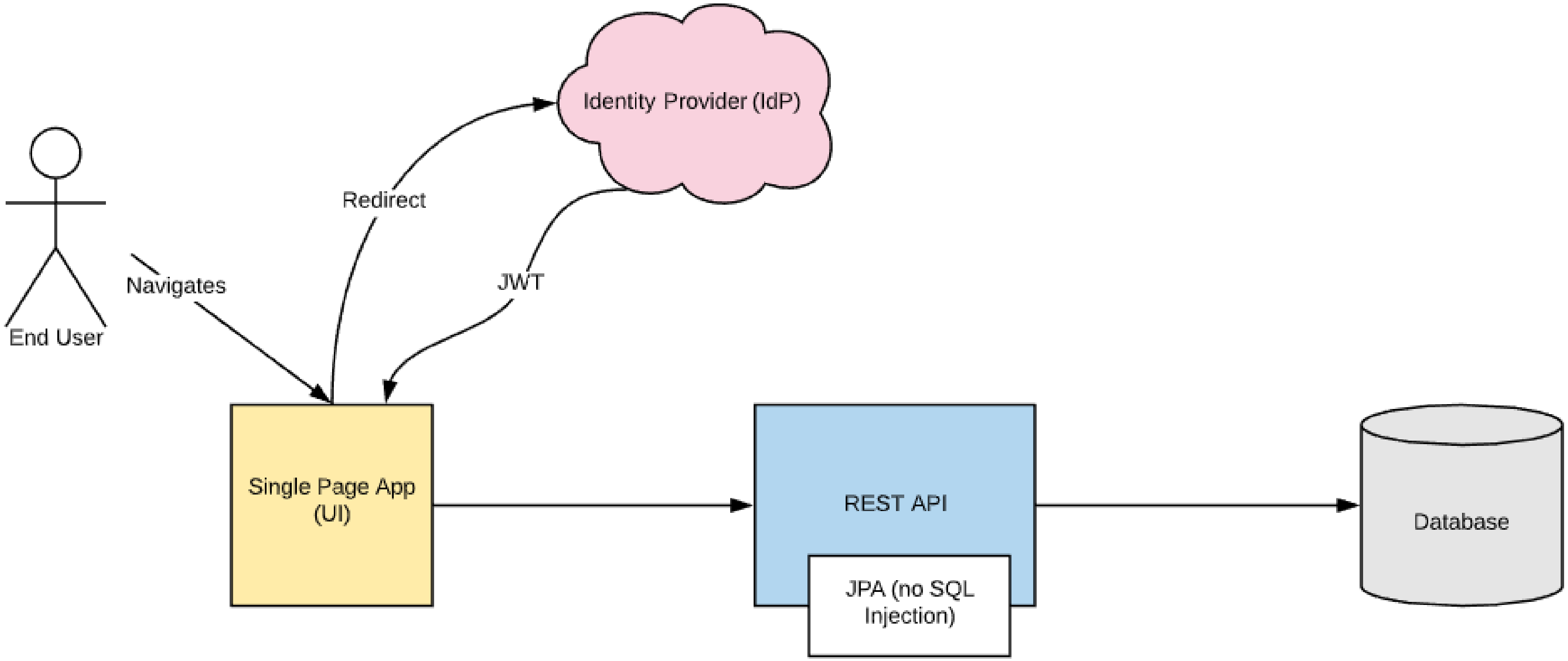
- Automatic SQL Injection prevention libraries
- *Almost* eliminates possibility of SQL Injection



CSRF – Cross Site Request Forgery

- Traditional Mitigation: Synchronizer Token Pattern
- Traditional Alternative: Verify Referer/Origin headers (weak)
- Modern mitigation (in a two tiered architecture)
  - Use REST APIs with JWTs instead (short lived token lives in browser memory or session Storage)
  - Set CORS headers properly
- If app dev insists on using Cookie then set cookie attribute: SameSite
  - Browser level prevention of sending a cookie from an origin that doesn't match

CSRF – Cross Site Request Forgery



Avoidance as a mitigation strategy

Avoidance as a Mitigation Strategy

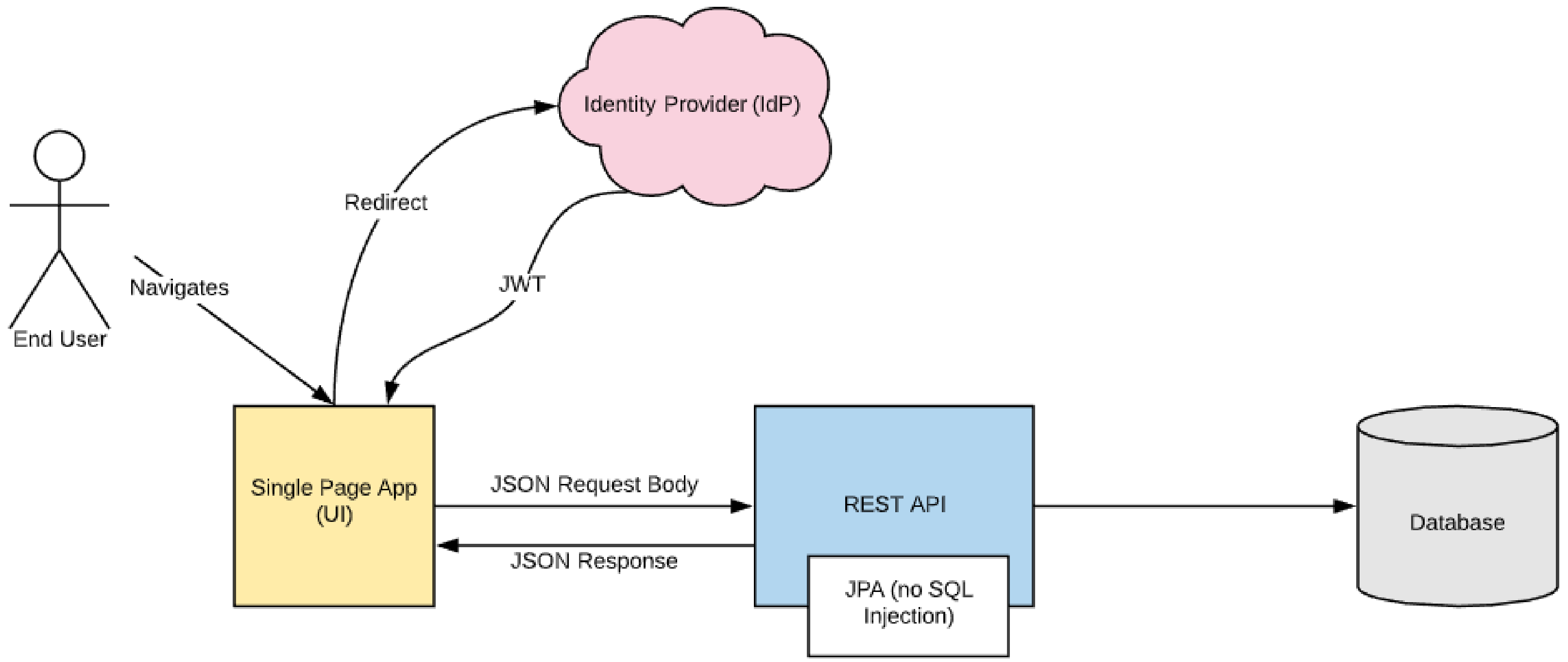
Eat This Not That

- XML Injection
  - Entity Expansion (denial of service)
- CSV Injection
  - Remote Code Injection
- File Upload
  - Malware
  - Path Traversal & Null Byte Injection
  - Site defacement

A sign advertising the Impossible Burger. *(Credit: NTL Photography)*

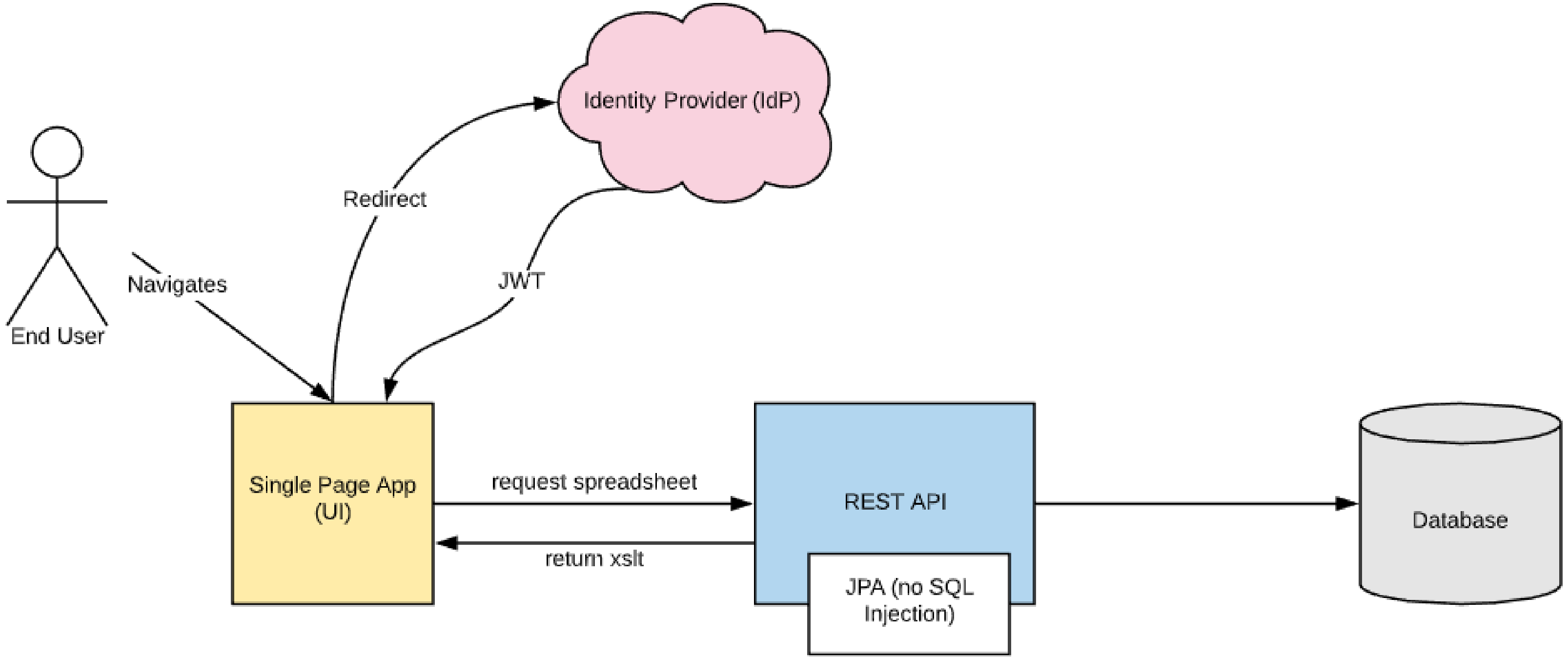
XML Injection

Use JSON not XML



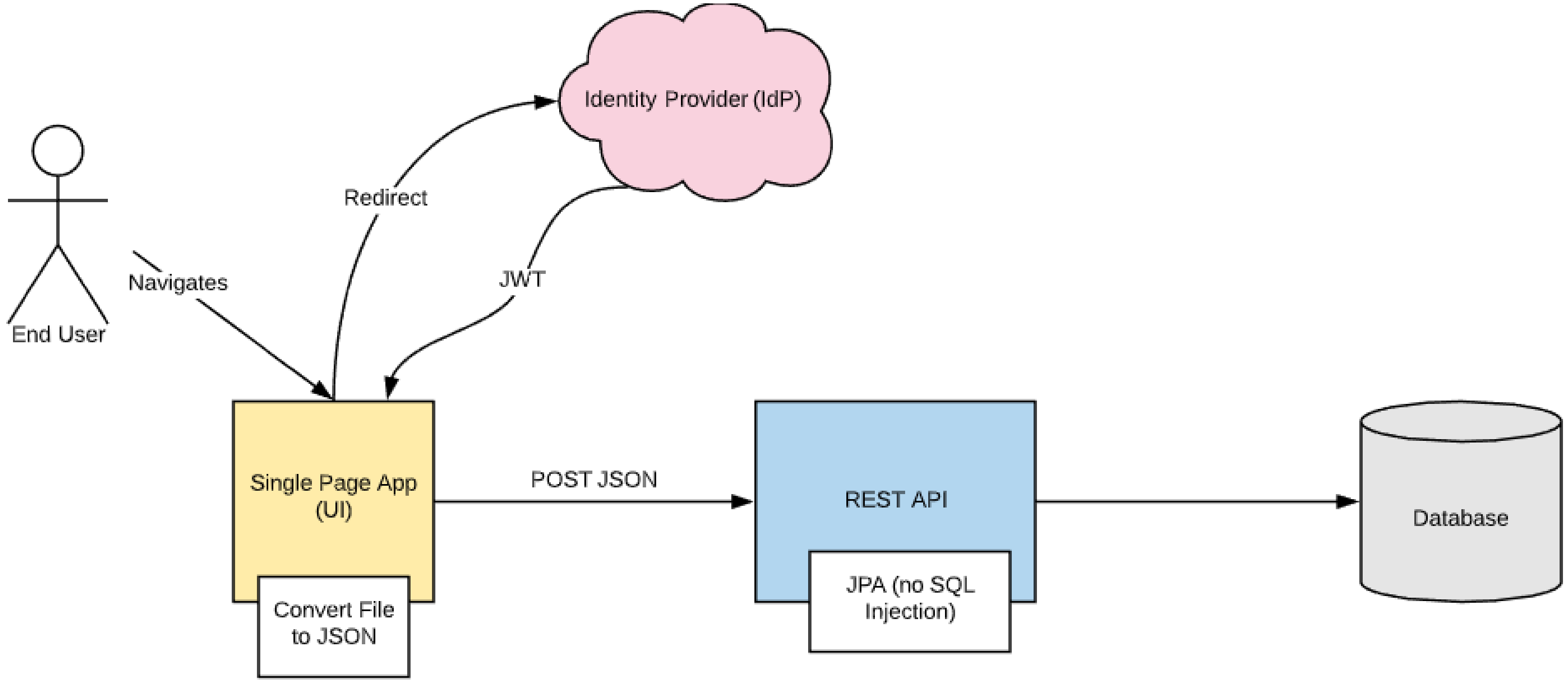
CSV Mitigation

Use XSLT instead of CSV



File Upload

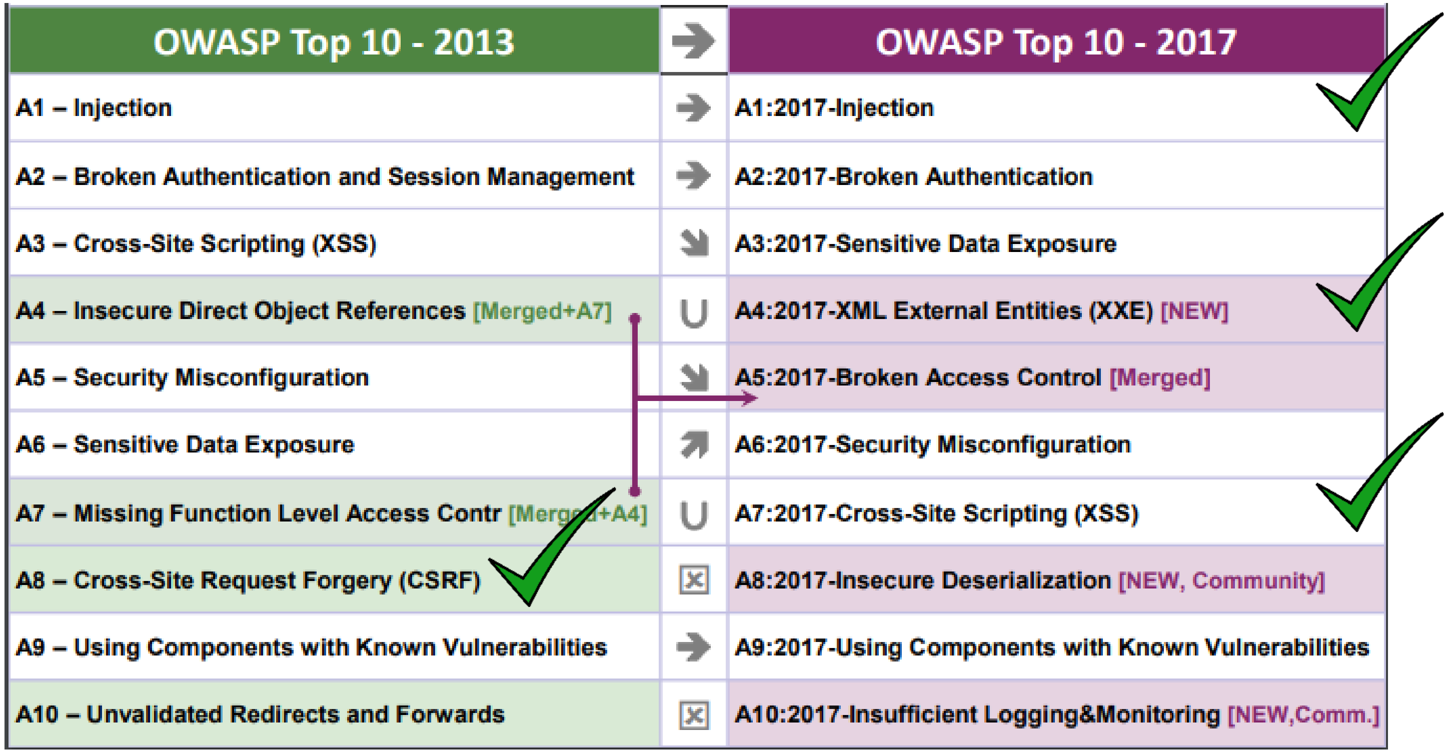
UI converts file to JSON so backend doesn't handle multi-part



DevSecOps

Production Self-Service with Guardrails

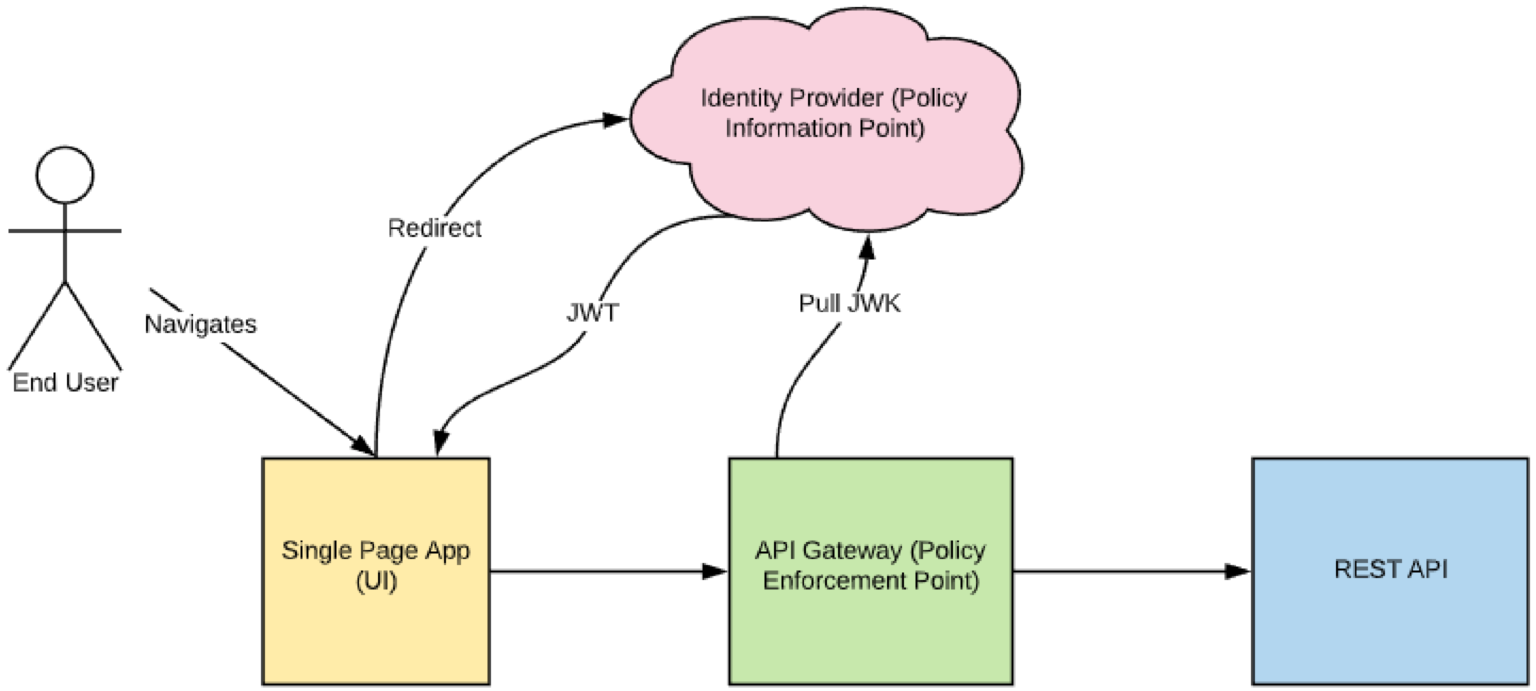
Checkpoint – What's Left?



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.]	

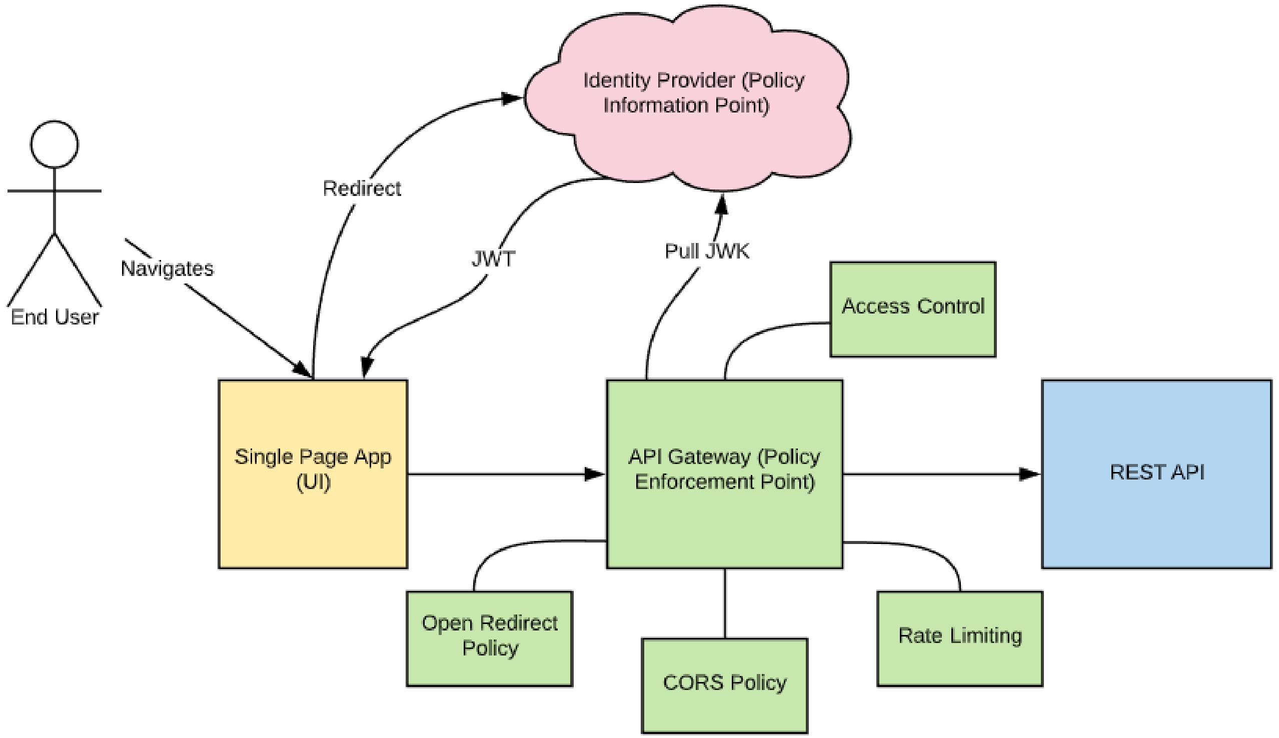
Broken Access Control

API Gateway for Standard Authorization Strategy


Identity Provider (Policy Information Point)
Redirect
JWT
Pull JWK
End User
Navigates
Single Page App (UI)
API Gateway (Policy Enforcement Point)
REST API

API Gateway

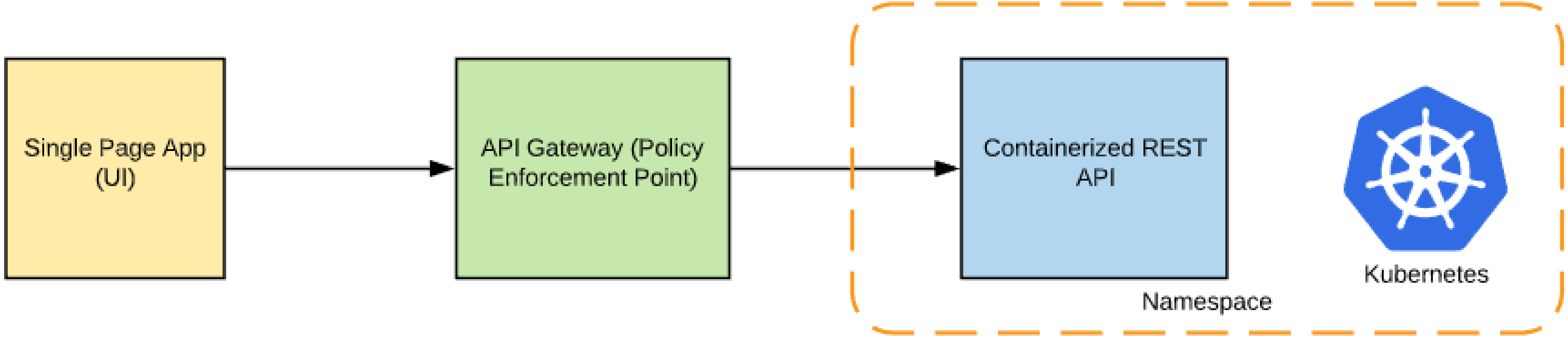
Policies for Defense in Depth



Security Misconfigurations

Docker and Kubernetes for Transparency & DevSecOps

- Configuration as code
- Security as code
- All config goes to Git



Security Misconfigurations

Docker and Kubernetes for Transparency

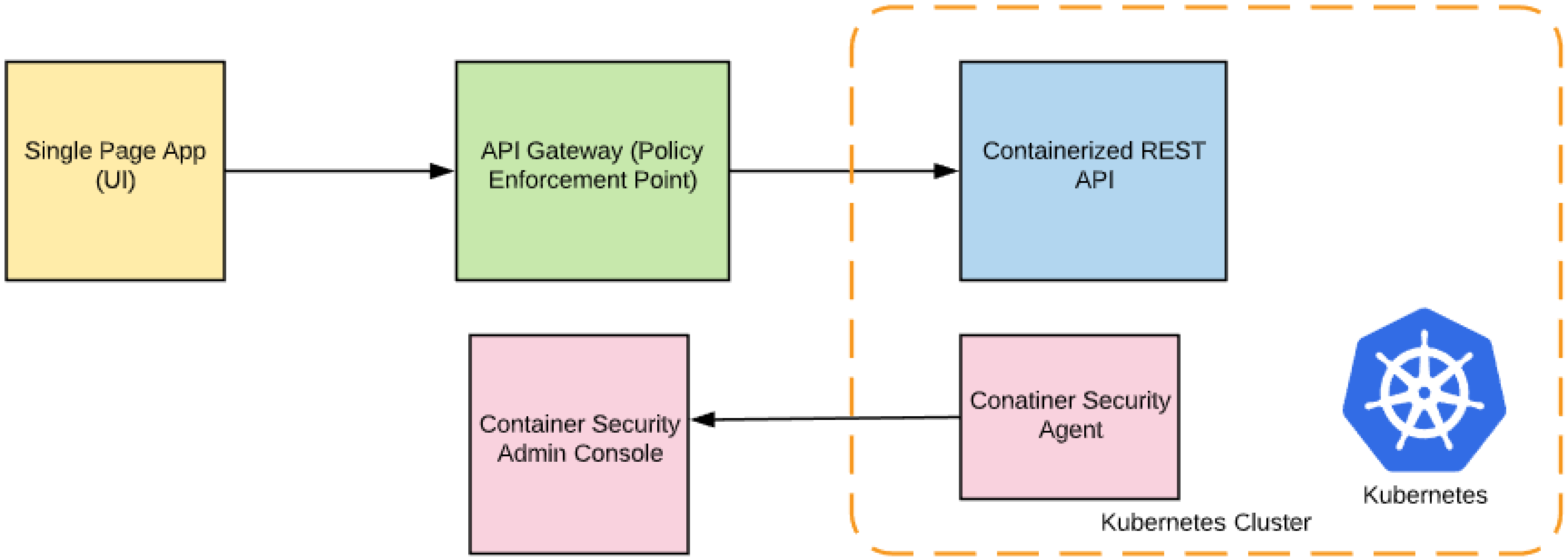
- Configuration as code / Security as code

```
FROM openjdk:alpine

RUN adduser -D -s /bin/sh 65534
USER 65534

VOLUME /tmp
ARG JAR_FILE
ADD ${JAR_FILE} app.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Dependencies with Known Vulnerabilities

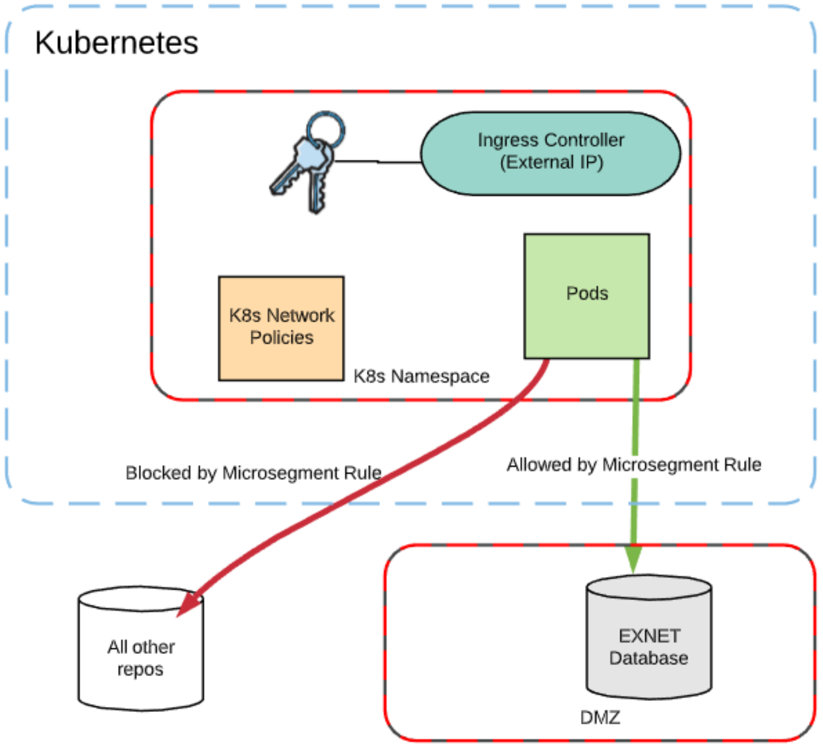
Minimal Base Images & Continuous Scanning

- Continuously Scans Containers and Worker Nodes for Vulnerability & Compliance issues

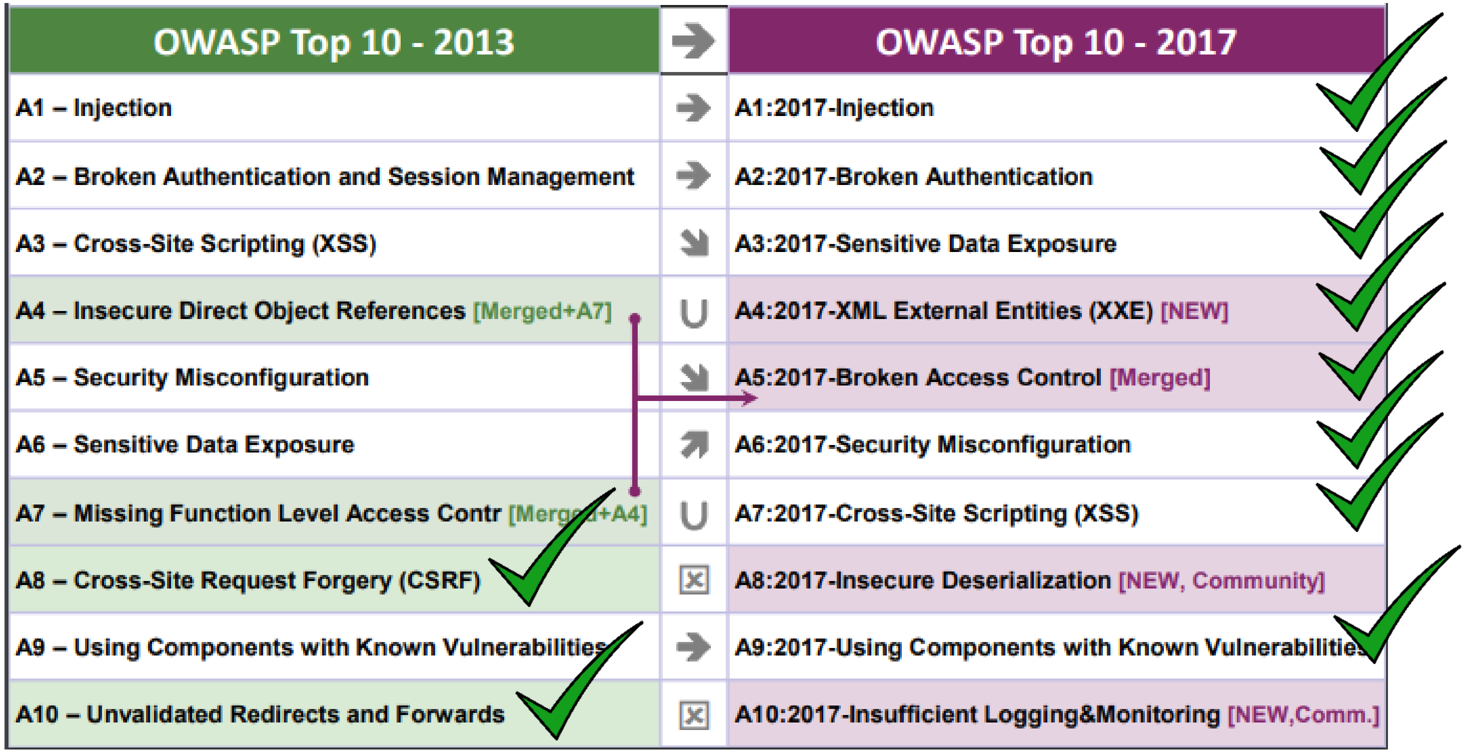


Sensitive Data Exposure

Network Microsegmentation in Kubernetes


Kubernetes
Ingress Controller (External IP)
K8s Network Policies
Pods
K8s Namespace
Blocked by Microsegment Rule
Allowed by Microsegment Rule
All other repos
EXNET Database
DMZ

Checkpoint – What's Left?



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.]

DevSecOps - Guardrails

Enabling Production Access

- Container Security Tool
  - block images with malware
- Pod Security Policy
  - Blocking containers from running as root
  - Require SecComp Profile
- Network Microsegmentation
  - Default Deny All outbound network connections
- Resource Quotas
  - Limit Resources such as memory and CPU

Wrap Up

- Secure by Default Technology Stack
  - Two Tiered Architecture (XSS, CSRF)
  - Choosing libraries that prevent vulns (Angular for output encoding)
- Avoidance as a mitigation strategy
  - Do this Not That – XML and CSV Injection prevention
- Embracing DevSecOps
  - API Gateway (Authorization strategy, defense in depth)
  - DevSecOps K8s/Docker (Reducing misconfigurations, Preventing data exfiltration, Continuous Security Testing of whole stack)

Questions?