

You'll need Java 1.8 on your machine. If you don't already have it, I recommend getting it from <https://adoptopenjdk.net/>

Download WebGoat 7.1:

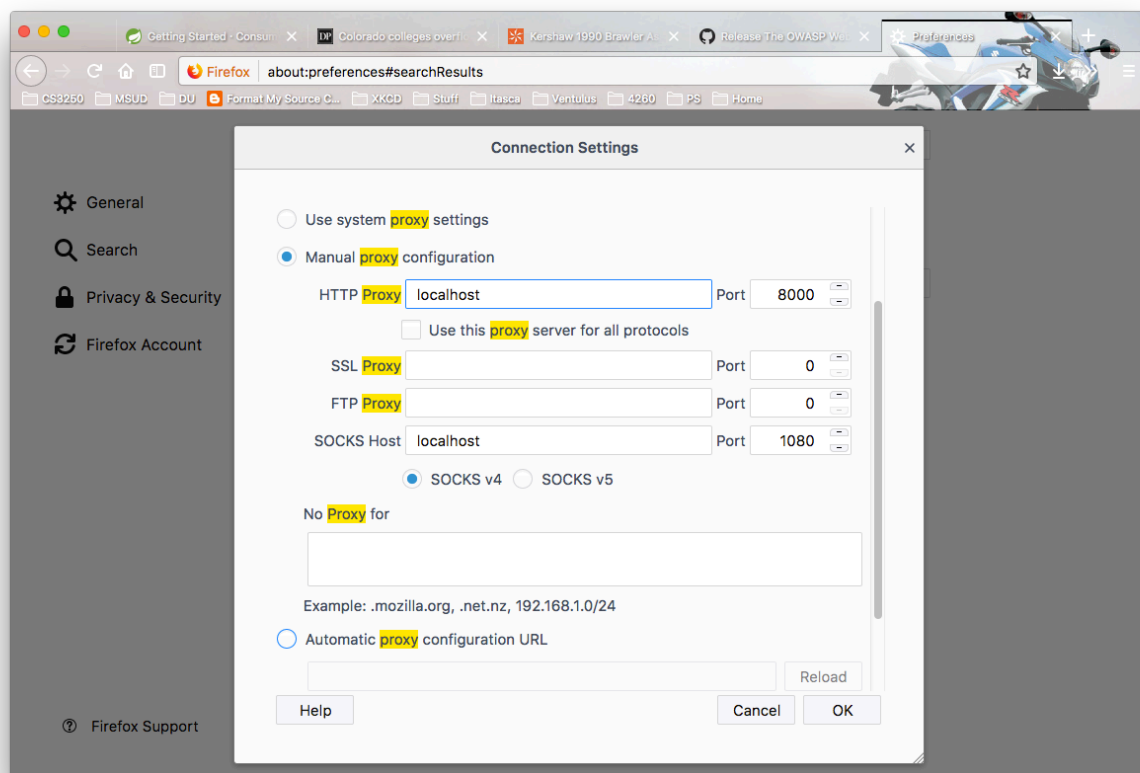
<https://github.com/WebGoat/WebGoat/releases/download/7.1/webgoat-container-7.1-exec.jar>

Then run it:

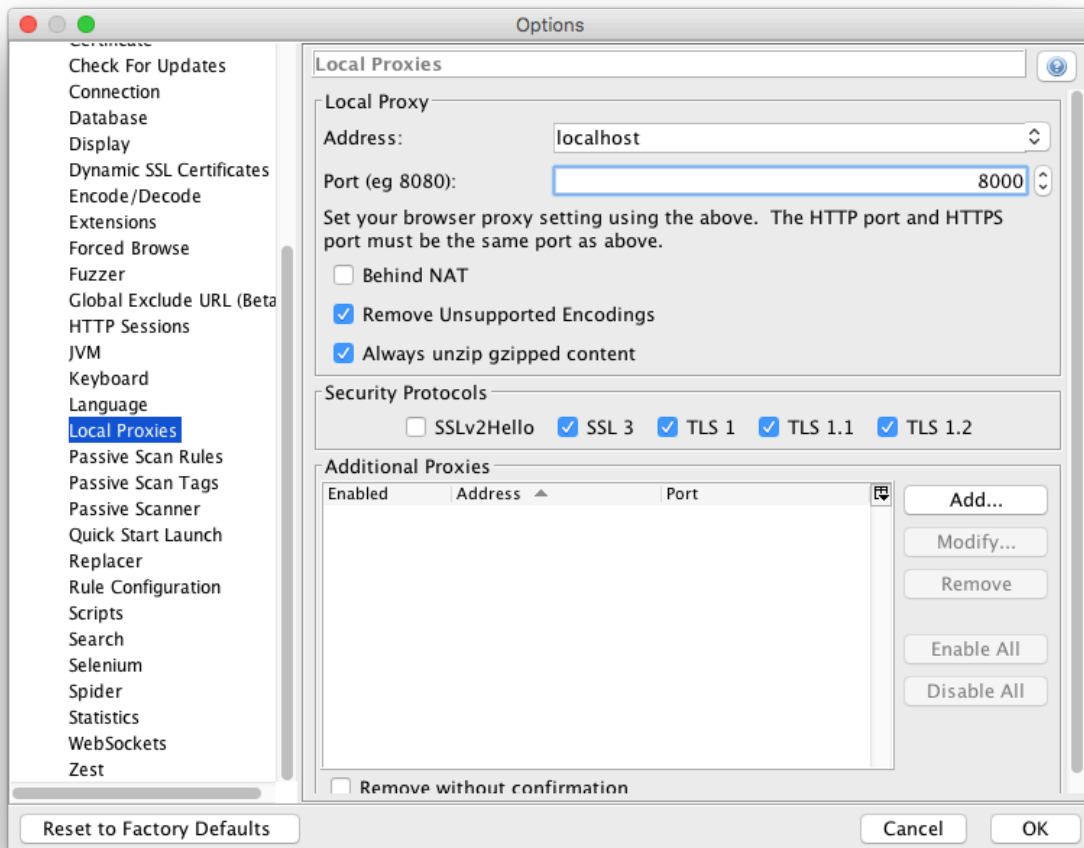
```
java -jar webgoat-container-7.1-exec.jar
```

Make sure you can connect to it: <http://localhost:8080/WebGoat>

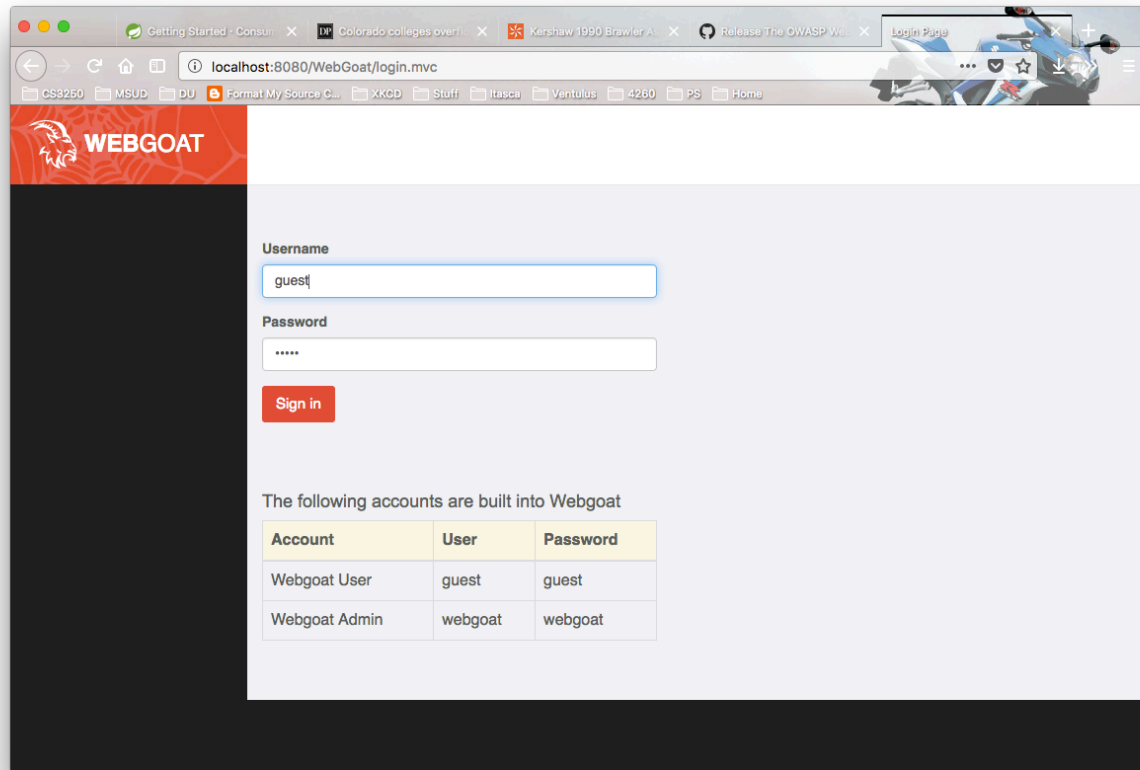
Set up a proxy in your browser. Make sure you remove localhost and 127.0.0.1 from the “No proxy for” textbox as that is exactly what you want to proxy for. Here's an example from Firefox.



Fire up Zap and set the port it listens to 8000 (or anything other than 8080 as that is what WebGoat listens to).



Point your browser to WebGoat.



Go to the numeric SQL injection page.

The screenshot shows the WebGoat application running in a browser. The address bar indicates the URL is `localhost:8080/WebGoat/start.mvc#attack/101829144/1100`. The page title is "Numeric SQL Injection".

Left Sidebar (Navigation):

- Introduction
- General
- Access Control Flaws
- AJAX Security
- Authentication Flaws
- Buffer Overflows
- Code Quality
- Concurrency
- Cross-Site Scripting (XSS)
- Improper Error Handling
- Injection Flaws
 - Command Injection
 - Numeric SQL Injection
 - Log Spoofing
 - XPATN Injection
 - String SQL Injection
 - LAB: SQL Injection
 - Stage 1: String SQL Injection
 - Stage 2: Parameterized Query #1
 - Stage 3: Numeric SQL Injection
 - Stage 4: Parameterized Query #2
 - Database Backdoors

Main Content Area:

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will be used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view weather data. Try to inject an SQL string that results in all the weather data being displayed.

Select your local weather station:

`SELECT * FROM weather_data WHERE station = [station]`

Right Sidebar (Cookies / Parameters):

Cookies / Parameters

Cookie/s

name	value
testing_session	ShVvcDl0WmQOU3Ys3pLaVvVOWoel01zR2NexdBWmfocDJ1TTdFTBWTThWkh2azhLNV0XMspQRTNC6odUS1dCFWmdBVkcxV0R2WUZWESjWm00MipUcUUwrZGNJK2RFL1gwRGVpPcKzovWocKzNmM2pZa0hua2R3PTOLtdSj02ZXJldWEJJeJhWnhc9PQ%3D%3D--2b4ce0c091c47734166a58ef0ace777f55a

comment
domain
maxAge
path
secure
version
httpOnly

Parameters

scr	101829144
menu	1100
stage	
num	

Hit "Go!" so that the submission goes through Zap.

Find the POST, right click, and choose Break.

The screenshot shows the OWASP ZAP 2.7.0 interface. The 'History' tab is active, displaying a list of requests. The selected request is a POST to `http://localhost:8080/WebGoat/attack?Screen=101829144&menu=1100`. The 'Request' pane shows the raw HTTP request details, including headers and body. The body contains the text `station=101&SUBMIT=Go!`.

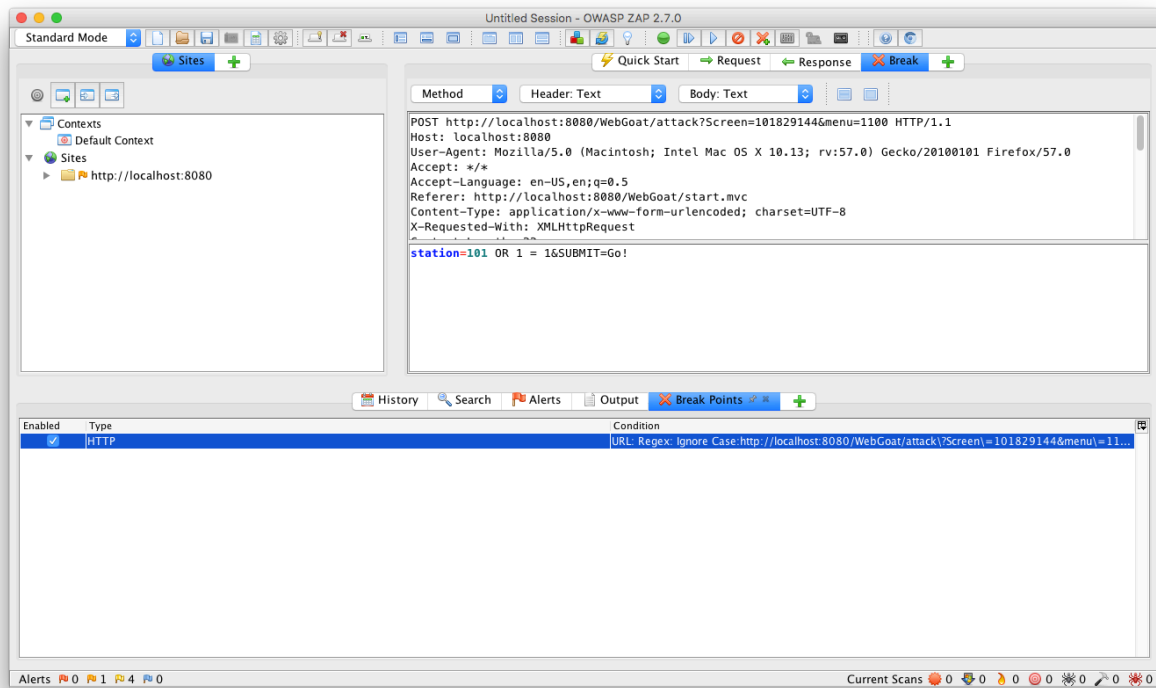
Id	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Body	Highest Alert	Note	Tags
122	11/30/17 9:10:48 AM	GET	http://localhost:8080/WebGoat/service/source.mvc	200	OK	48 ms	11,413 bytes	Low		Comment
123	11/30/17 9:10:48 AM	GET	http://localhost:8080/WebGoat/service/lessoninf...	200	OK	106 ms	127 bytes	Low		JSON
124	11/30/17 9:10:48 AM	GET	http://localhost:8080/WebGoat/service/lessonm...	200	OK	98 ms	10,287 bytes	Low		JSON
125	11/30/17 9:11:07 AM	POST	http://localhost:8080/WebGoat/attack?Screen=1...	200	OK	51 ms	1,701 bytes	Medium		Form, Comment
126	11/30/17 9:11:07 AM	GET	http://localhost:8080/WebGoat/service/lessonpl...	200	OK	11 ms	1,049 bytes	Medium		Comment
127	11/30/17 9:11:07 AM	GET	http://localhost:8080/WebGoat/service/cookie.mvc	200	OK	21 ms	1,974 bytes	Low		JSON
128	11/30/17 9:11:07 AM	GET	http://localhost:8080/WebGoat/service/hint.mvc	200	OK	21 ms	707 bytes	Low		JSON
129	11/30/17 9:11:07 AM	GET	http://localhost:8080/WebGoat/service/solution...	200	OK	29 ms	30,799 bytes	Medium		Comment
130	11/30/17 9:11:07 AM	GET	http://localhost:8080/WebGoat/service/source.mvc	200	OK	34 ms	11,413 bytes	Low		Comment
131	11/30/17 9:11:07 AM	GET	http://localhost:8080/WebGoat/service/lessonpr...	200	OK	10 ms	106 bytes	Low		JSON
132	11/30/17 9:11:07 AM	GET	http://localhost:8080/WebGoat/service/lessoninf...	200	OK	188 ms	127 bytes	Low		JSON
133	11/30/17 9:11:07 AM	GET	http://localhost:8080/WebGoat/service/lessonm...	200	OK	120 ms	10,287 bytes	Low		JSON

Run the request again. This time Zap will stop the submission.

The screenshot shows the OWASP ZAP 2.7.0 interface with a break point configured. The 'Break Points' tab is active, showing a table with one entry: a checked 'Enabled' checkbox, 'Type' set to 'HTTP', and a 'Condition' of `URL: Regexp: Ignore Case: http://localhost:8080/WebGoat/attack?Screen=101829144&menu=11...`. The 'Request' pane shows the same POST request as in the previous screenshot.

Enabled	Type	Condition
<input checked="" type="checkbox"/>	HTTP	URL: Regexp: Ignore Case: http://localhost:8080/WebGoat/attack?Screen=101829144&menu=11...

Inject your SQL and hit play at the top twice.



This should be the result.

number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will be used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view weather data. Try to inject an SQL string that results in all the weather data being displayed.

*** Bet you can't do it again! This lesson has detected your successful attack and has now switched to a defensive mode. Try again to attack a parameterized query.**

Select your local weather station:

```
SELECT * FROM weather_data WHERE station = 101 OR 1=1
```

STATION	NAME	STATE	MIN_TEMP	MAX_TEMP
101	Columbia	MD	-10	102
102	Seattle	WA	-15	90
103	New York	NY	-10	110
104	Houston	TX	20	120
10001	Camp David	MD	-10	100
11001	Ice Station Zebra	NA	-60	30

Session Details:

name	value
testing_session	SHVvcDl0WmQOU3YyY3pLaVlVOWhoel01zR2NwkdBWmfceDJ1TTdFTBWTThWkh2azhLNV0XMspQRtNC60dUS1dCFWmdBVkcxV0R2WUZWESjWm00MipUcUUwrZGNJK2RFL1gwRGVpCkZvVWo3KzNmM2p2a0hua2R3PT0LTdSj2ZXJldWEJJeJhWNC9PQ%3D%3D--2b4ce0c091c47734166a59ef0ce777155a

Parameters:

scr	menu	stage	num
101829144	1100		