Assignment 4: WebGoat

CS390P: Web Application Architecture

John Samson

November 11th, 2018

Abstract

In our previous work with Ruby on Rails we learned the importance of creating web applications that are *secure* as well as easy to navigate and pleasing to the eye. We've only begun to learn about *defensive security* and *validation* of the input fields on our website. Here we seek a more complete understanding of web security; only by understanding the kinds of attacks we see on the web can we expect to defend against them. For this we will create an environment for *penetration testing*, using the Docker virtualization tool to create a *container* for the "intentionally vulnerable" web server we'll be using for our mock attacks. The WebGoat web server from OWASP is a learning tool which allows us to safely perform various forms of web-based attacks. We'll rely heavily on another OWASP tool, Zed Attack Proxy (ZAP, for short), to listen to and tamper with incoming/outgoing HTTP requests. We'll learn a variety of attacks here, from basic HTTP request manipulation, breaking into files we shouldn't have access to, and more advanced *injection* of code into SQL server database queries. Finally, we'll hijack our user's browser itself and execute a simple JavaScript using a *Cross-Site Scripting* (XSS) attack.

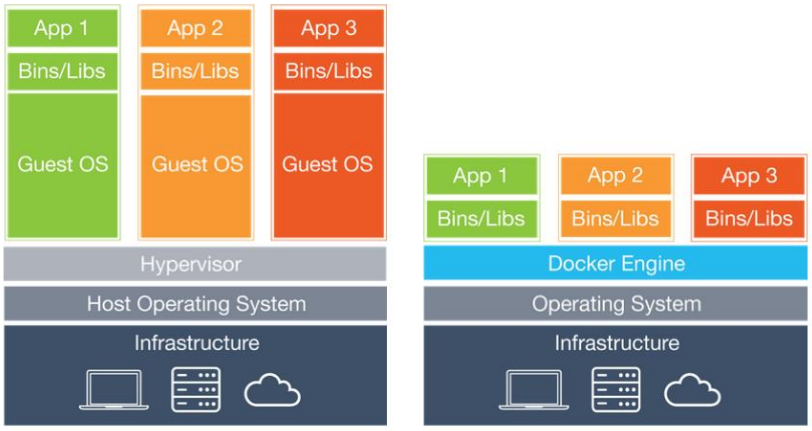
Introduction

We have some previous experience with WebGoat and penetration testing, having completed CS49AU Computer Penetration Testing and Defense here at MSU Denver. For this reason, we will select more 'intermediate' WebGoat exercises, and in general seek to expand upon the knowledge and work we did before.

We'll need to setup an environment for WebGoat. We have a few options:

1. (Windows 10 64-bit Professional only) Enable virtualization within the BIOS and install Docker to create a *container* to run the WebGoat server. Download Docker for Windows here: https://docs.docker.com/docker-for-windows/install/. Mac OSX Users can Install Docker for Mac: https://docs.docker.com/docker-for-mac/.
2. (Older Windows and Mac versions) Install Docker Toolbox to create a WebGoat container, available here: https://docs.docker.com/toolbox/toolbox_install_windows/
3. Install Oracle's VirtualBox virtual machine software to create a virtual Kali Linux machine to run the WebGoat server. Download VirtualBox is available here: https://www.virtualbox.org/wiki/Downloads, then download and install the latest Kali Linux disc image in VirtualBox, available here: https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-hyperv-image-download/. Then download the WebGoat 8 server file 'webgoat-server-8.0.0.M21.jar', available here: https://github.com/WebGoat/WebGoat/releases.

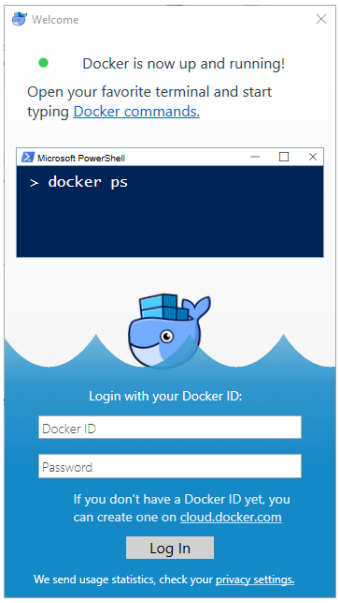
As an alternative to using virtual machines for virtualization, Docker creates a relatively lightweight *container* file for housing on WebGoat server. The advantage is that we don't need a virtual operating system in order to run our server, which frees up many valuable system resources. The following diagram illustrates the architecture of a virtual machine vs. Docker containers (Gyurko, 2018):



Part 1: Setup

Setup Docker

For most users, setting up Docker itself is a relatively painless process. First create a free account with Docker using your email, then download and install an appropriate Docker installation for your system (as described above). Launch the Docker or Docker Toolbox app, and login using the credentials you created.



Note: This tutorial will follow the necessary steps to setup Docker (not the Toolbox) for Windows 10 64-bit Professional and enable virtualization in the BIOS. Docker Toolbox setup will not require changing any BIOS settings.

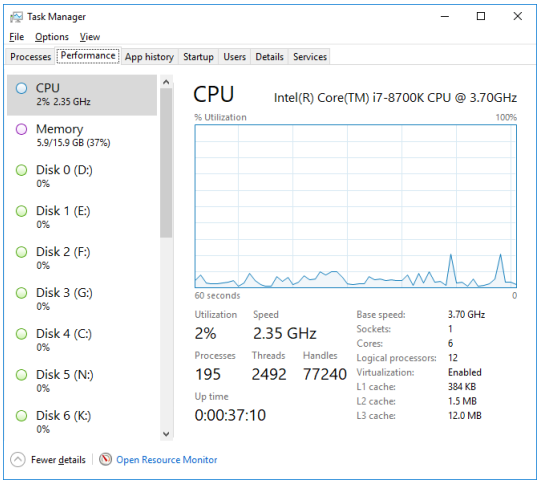
- Verify the Docker installation by entering '`docker --version`' in the Windows command line (Docker Docs, 2018):

```
C:\CS390P-Web-Application-Architecture>docker --version
Docker version 18.06.1-ce, build e68fc7a
```

Enable Virtualization in the BIOS

- We need to enable *virtualization* within the BIOS of our machine in order to run a Docker container for WebGoat. Restart your computer and enter the BIOS settings (usually by pressing 'F2' or 'F11' during startup for most PC's) and look for a setting similar to 'Intel Virtualization Technology'. While your BIOS settings will undoubtedly look different, here's one for reference:

- Set the virtualization settings to ‘Enabled’, then save the BIOS settings and restart your computer. Run Windows Task Manager (Ctrl + Alt + Delete), then click on the ‘Performance’ tab to verify that we have virtualization enabled:



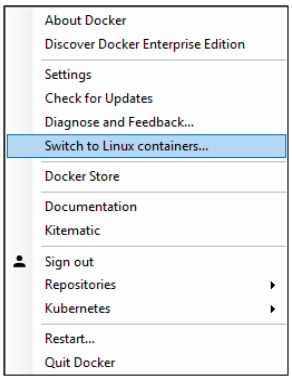
Setup WebGoat 7.1

One great feature of Docker is its ability to quickly download pre-configured containers for a variety of projects and applications.

- Download the WebGoat 7.1 Docker image by entering ‘`docker pull webgoat/webgoat-7.1`’ in the command line interface (Docker Hub, 2016):

```
C:\CS390P-Web-Application-Architecture>docker pull webgoat/webgoat-7.1
Using default tag: latest
latest: Pulling from webgoat/webgoat-7.1
image operating system "linux" cannot be used on this platform
```

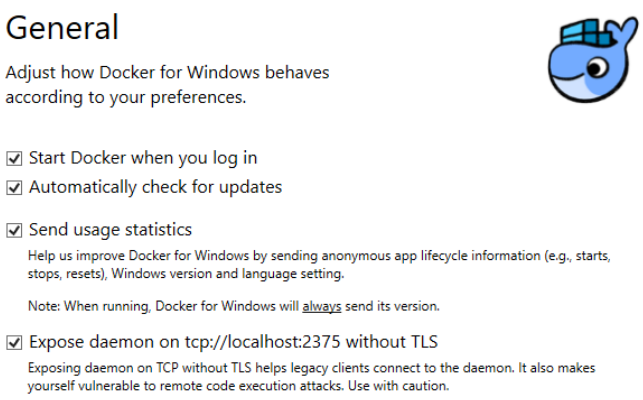
- We can see we need to switch to using Linux containers with Docker. Find the Docker icon in the Windows taskbar, then select ‘Switch to Linux Containers’, then click ‘Switch’ to switch.



- ‘`docker pull webgoat/webgoat-7.1`’ now gives us a different warning upon failure.

```
C:\CS390P-Web-Application-Architecture>docker pull webgoat/webgoat-7.1
Using default tag: latest
Warning: failed to get default registry endpoint from daemon (error during connect: Get http://%2F%2F.%2Fpipe%2Fdocke
r_engine/v1.38/info: open //./pipe/docker_engine: The system cannot find the file specified. In the default daemon co
nfiguration on Windows, the docker client must be run elevated to connect. This error may also indicate that the dock
er daemon is not running.). Using system default: https://index.docker.io/v1/
error during connect: Post http://%2F%2F.%2Fpipe%2Fdocker_engine/v1.38/images/create?fromImage=webgoat%2Fwebgoat-7.1&
tag=latest: open //./pipe/docker_engine: The system cannot find the file specified. In the default daemon configurati
on on Windows, the docker client must be run elevated to connect. This error may also indicate that the docker daemon
 is not running.
```

- Right click the Docker taskbar icon, this time click 'Settings' then 'General'. Click the 'Expose daemon on tcp://localhost:3275 without TLS' checkbox. This allows the Docker daemon to communicate with the internet. Note: Use with caution.



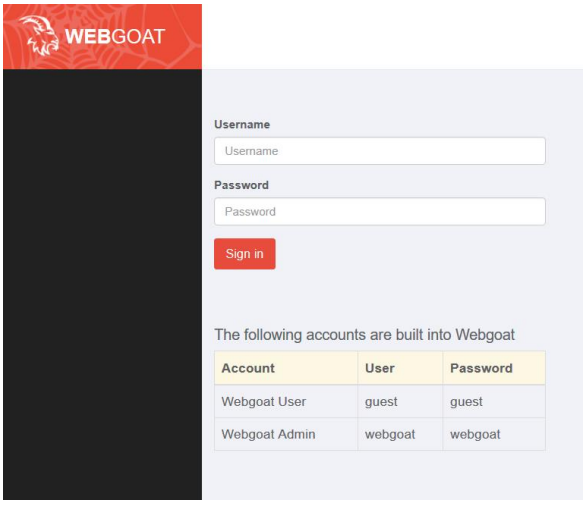
- Finally, we can enter 'docker pull webgoat/webgoat-7.1' to download the WebGoat 7.1 Docker image:

```
C:\CS390P-Web-Application-Architecture>docker pull webgoat/webgoat-7.1
Using default tag: latest
latest: Pulling from webgoat/webgoat-7.1
75a822cd7888: Downloading [========================>                          ]  26.16MB/51.36MB
57de64c72267: Download complete
cd1fc1696ecd: Download complete
34836fffacad: Download complete
4f4f57ee64ee: Download complete
975b9daf71f5: Download complete
6c6cde91351a: Download complete
e9fdf25c5996: Download complete
ec518ea20c44: Download complete
788152abd098: Download complete
```

- Run the WebGoat server using 'docker run -p 8080:8080 -t webgoat/webgoat-7.1' (Docker Hub, 2016):

```
C:\CS390P-Web-Application-Architecture>docker run -p 8080:8080 -t webgoat/webgoat-7.1
```

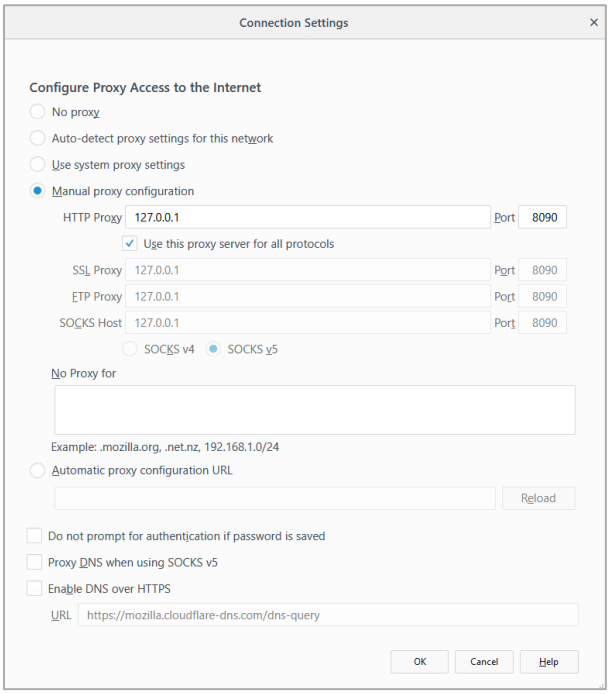
- Navigate your Browser (we'll be using FireFox) to 'localhost:8080/WebGoat'. You should see the WebGoat login page:



Configure Proxy Settings

In order to use OWASP ZAP to perform man-in-the-middle attacks on WebGoat, we need to setup a *proxy server* within FireFox to forward web traffic from one port (the WebGoat service will run via port 8080) to another (we'll use port 8090).

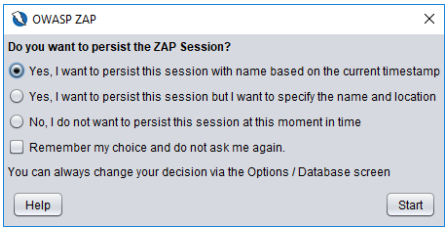
- In Firefox, click the Menu button, then 'Options', then scroll down to 'Network Settings' at the very bottom, then click 'Settings' to open Network Connection Settings. Select 'Manual proxy configuration' and set the 'HTTP Proxy' to '127.0.0.1' and the port to 8090. Then click the 'Use this proxy server for all protocols' checkbox. Click 'OK'.



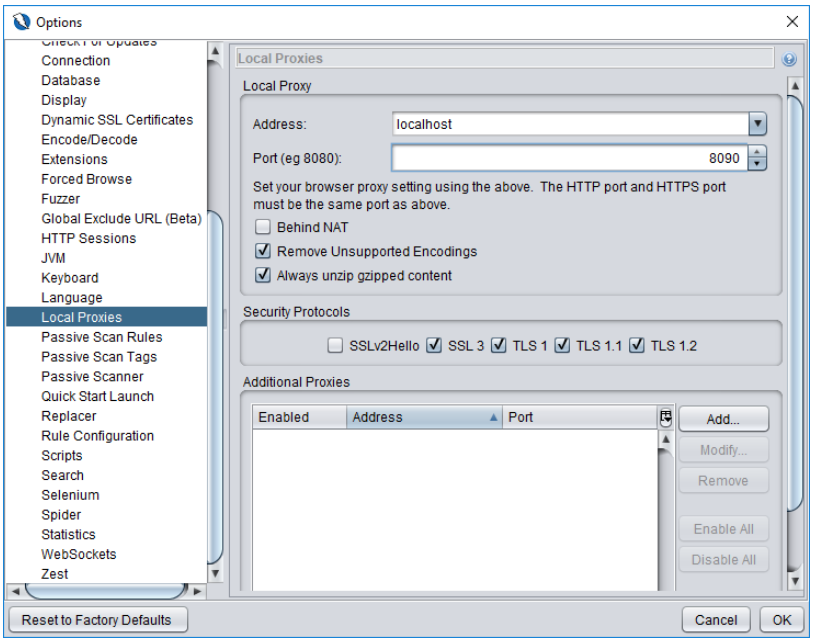
Setup OWASP ZAP

OWASP Zed Attack Proxy (ZAP) is an excellent tool for web penetration testing that allows us to listen in on all incoming and outgoing HTTP requests from specific ports or URLs. ZAP allows us to specify a *break point* on the execution of those requests, so we may view (and tamper with) them and perform a *man-in-the-middle* attack on WebGoat.

- Download OWASP ZAP here: https://www.owasp.org/inex.php/OWASP_Zed_Attack_Proxy_Project
- Run ZAP, you should be prompted with this screen. Click 'Yes, I want to persist this session with name based on the current timestamp, then click 'Start'.



- Select ‘Tools’ from the Menu Bar, then ‘Options’ to setup ZAP for the proxy server we created in FireFox previously. Make sure ‘Address:’ contains ‘localhost’ and that the Port is set to 8090:



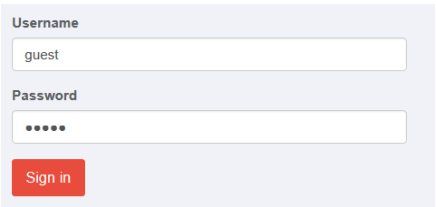
Click ‘OK’ to save ZAP proxy options. We are now ready to explore WebGoat and use ZAP to intercept HTTP requests. First, we’ll perform some simple exploits, then move onto more advanced exercises such as SQL query injection and executing JavaScripts within the browser.

Part 2: Simple WebGoat Exploits

- Enter ‘Ctrl + C’ while the WebGoat server is running to stop the WebGoat service. Then enter ‘`docker run -p 8080:8080 -t webgoat/webgoat-7.1`’ to restart the WebGoat server.

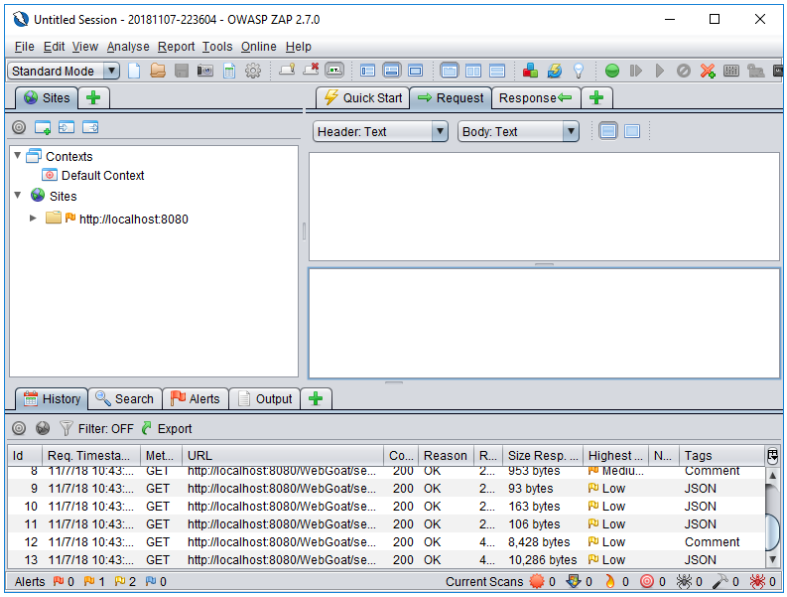
```
C:\CS390P-Web-Application-Architecture>docker run -p 8080:8080 -t webgoat/webgoat-7.1
```

- Login to Webgoat with the username ‘guest’ and password ‘guest’:



- The WebGoat 7.1 Homepage. Take some time to look through the various exercises:

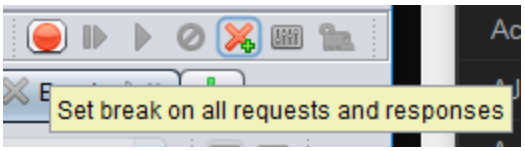
- Make sure that ZAP is listening in on web traffic as you navigate through WebGoat's various lessons:



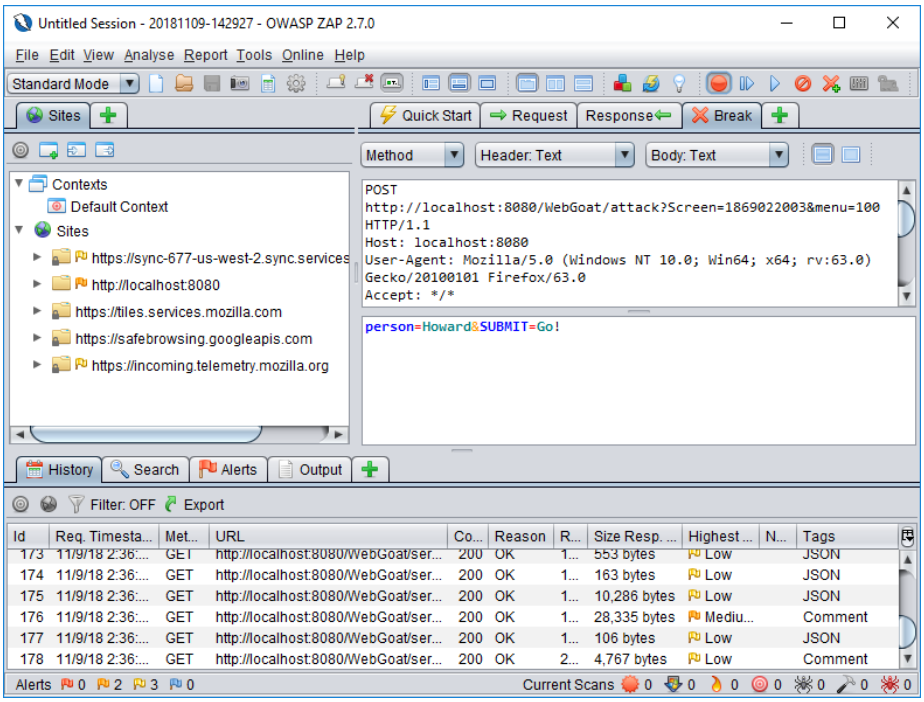
HTTP Basics

- Our first lesson is a simple example. Here we have a simple text input field, which reverses a person's name:

- Let's set a break point in ZAP and see if we can change the name we enter while its in transit. Click the green circle to set a break point, then press 'Go!' in WebGoat.



- Here we can see a POST HTTP request in ZAP:

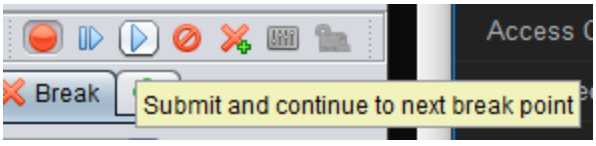


- Now let's change 'Howard' to 'Beverly'

```
person=Howard&SUBMIT=Go!

person=Beverly&SUBMIT=Go!
```

- Click the 'Play' button to resume WebGoat:



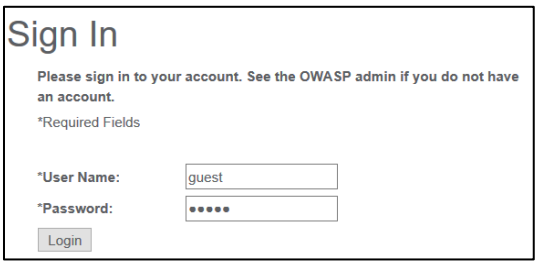
- We can see that 'Beverly', in reverse, is displayed by WebGoat. Our first successful exploit.

Enter your Name: ylreveB Go!

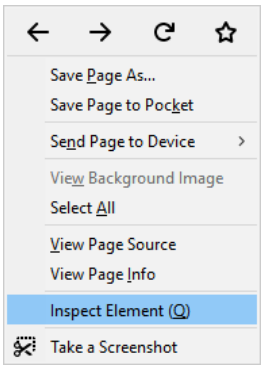
Code Quality: Discover Clues in the HTML

We can exploit poor code quality and find things ordinary users simply wouldn't look for. Lazy web administrators may leave privileged information within HTML comments.

- Here we have a Sign In page, let's see if we can inspect the HTML to find anything useful.

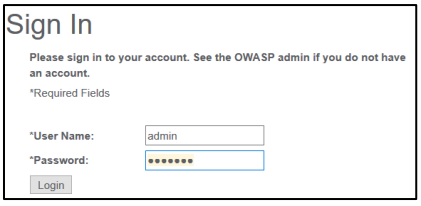


- Right click on the Sign In page and select 'Inspect Element':

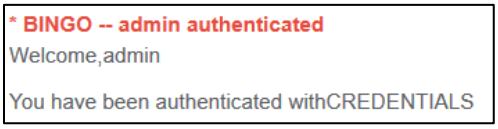


- Here we can see the WebGoat admin has foolishly left a comment with admin login credentials:

- We can use this login information, Sign In with the username 'admin' and password 'adminpw':



- We've logged in successfully:

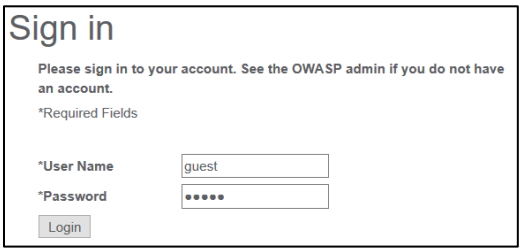


Improper Error Handling: Fail Open Authentication Scheme

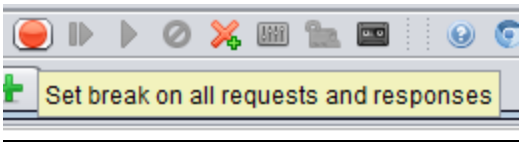
In some cases, we can bypass an authentication scheme by simply deleting the required parameter.

- Now let's use ZAP to intercept a web packet.

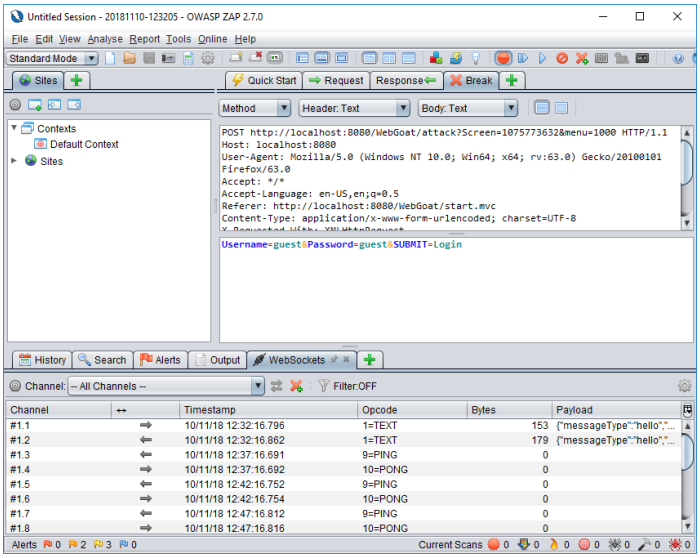
  Again, we have a Sign In screen:



- Let's set a break point and begin tampering with WebGoat's HTTP requests:



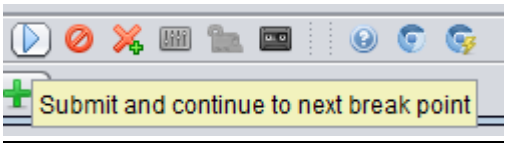
- Click 'Login' in WebGoat to see the packet contents within ZAP.



- Here we can delete the 'Password' field as shown:

```
Username=guest&SUBMIT=Login
```

- Now let's allow WebGoat to continue:

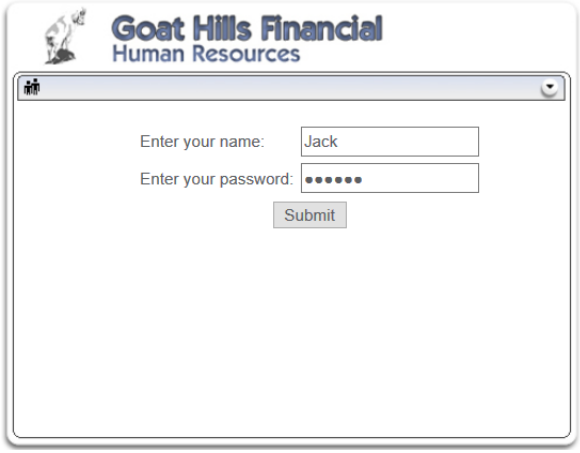


- Another successful login, again without having credentials:

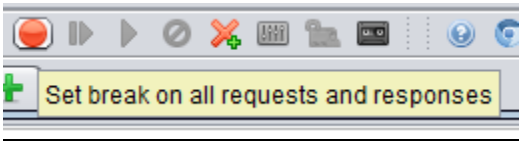
Insecure Communication: Insecure Login

For this exercise, we will use ZAP to *sniff* out a password using an unsecure cleartext communication.

- Again, we have a Sign In screen:



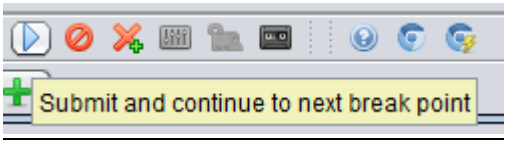
- Now set a break point in ZAP:



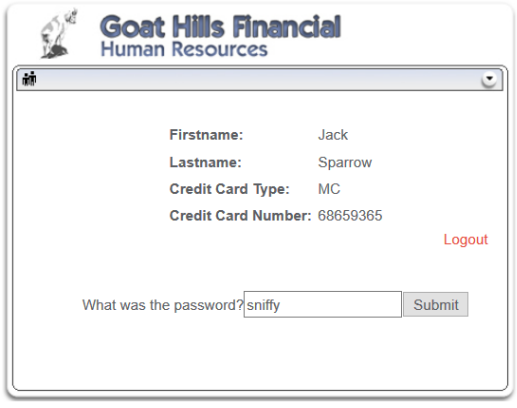
- ZAP shows us the following password 'sniffy' in cleartext:

```
clear_user=Jack&clear_pass=sniffy&Submit=Submit
```

- Let ZAP continue the WebGoat service:



- We can now login using the password we sniffed using ZAP. Examples like this show that in *any* communication we should use some form of *encryption*, so that if someone is listening in using ZAP or a similar tool, they won't be able to see sensitive information in cleartext.

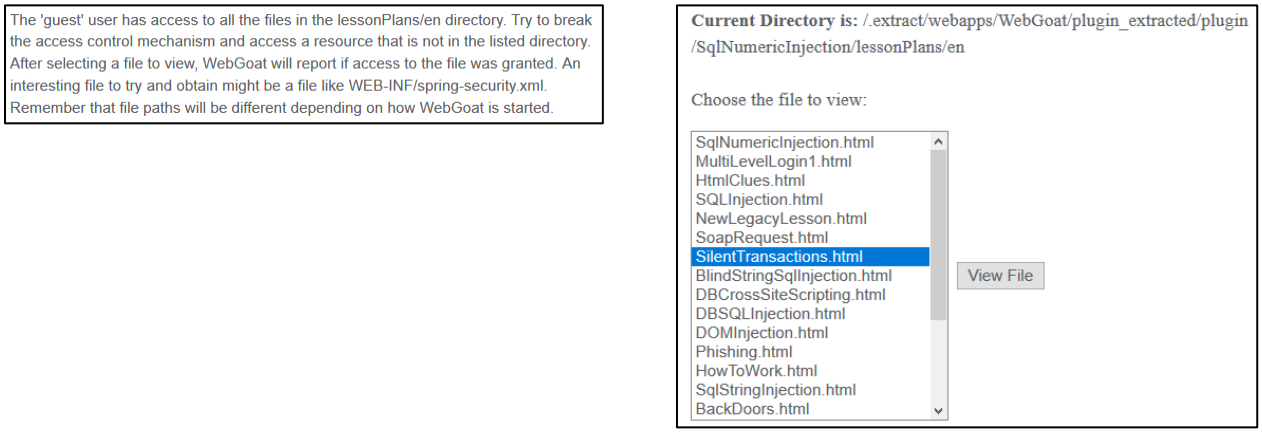


* You completed Stage 1!

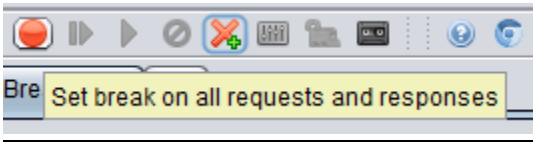
Part 3: Access Control Flaws

Bypass a Path Based Access Control Scheme

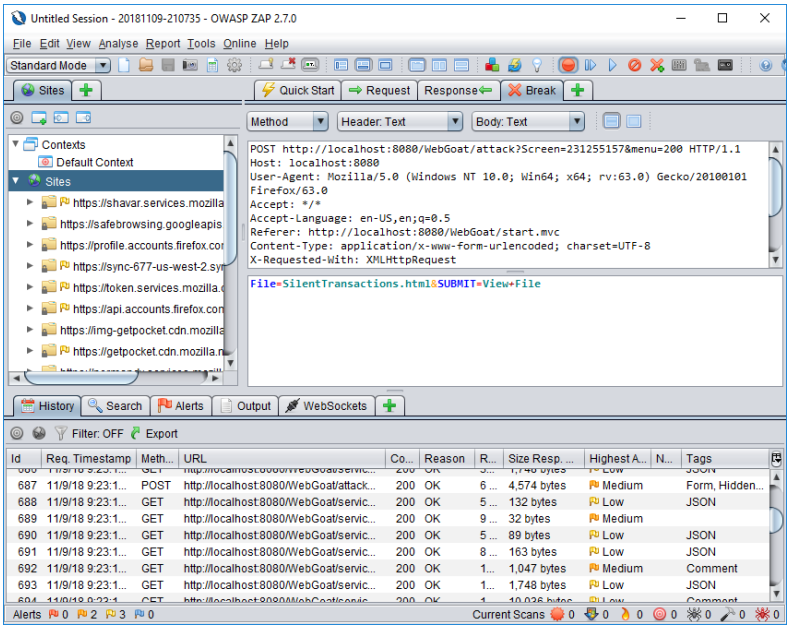
Now let's try something a little different. Here we have a file viewer, in which we're only allowed to view files within a single directory within the WebGoat app. Let's see if we can break out of these confines and access a file we shouldn't be able to.



- Select a file from the list above, then set a break point in ZAP and then click 'View File' in WebGoat.



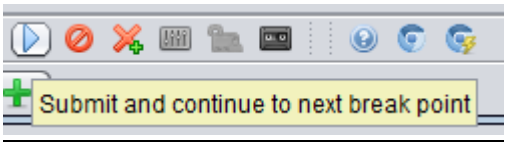
- ZAP lets us see that '`File=SilentTransactions.html`'. Let's tell WebGoat to retrieve a different file:



- We can tell the server to navigate *up* one directory by adding '`../`' at the beginning of our Filename. We can use this to navigate to the root directory and open the 'WEB-INF/sprint-security.xml' file:

```
File=../../../../../WEB-INF/spring-security.xml&SUBMIT=View+File
```

- Resume the WebGoat service within ZAP.



- We've successfully accessed a file outside of our designated directory. We can use an exploit like this to break into file paths containing privileged information.

* Congratulations! Access to file allowed. ==> /.extract/webapps/WebGoat/WEB-INF/spring-security.xml

Current Directory is: /.extract/webapps/WebGoat/plugin_extracted/plugin/SqlNumericInjection/lessonPlans/en

Viewing file: /.extract/webapps/WebGoat/WEB-INF/spring-security.xml

```
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-3.2.xsd">
```

Part 4: Injection Flaws

String SQL Injection

There are a number of different ways we can cause a web application to execute unwanted code. In general, the process is known as *code injection*. In particular, the #1 form of web attack in 2018 is *SQL query injection*. We are not limited to injecting unwanted code in SQL database queries; we can inject code into JSON, XML, and AJAX as well. Let's begin by demonstrating a simple string SQL injection, then we can perform more advanced manipulation of an SQL database.

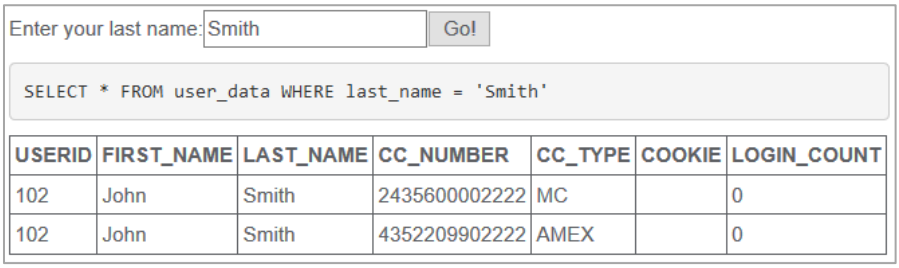
- We have a text input field which will retrieve information from an SQL database based on last name. Let's see if we can gain access to more than one user's information.

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

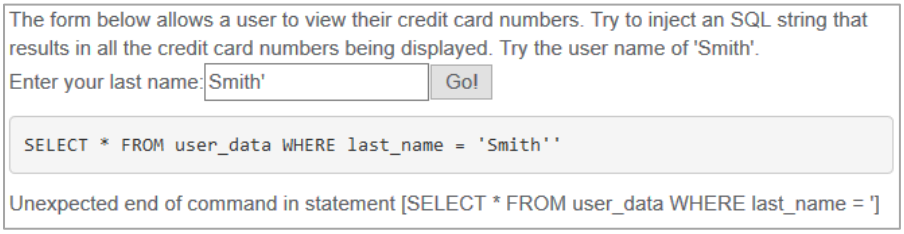
Enter your last name: Your Name Go!

```
SELECT * FROM user_data WHERE last_name = 'Your Name'
```

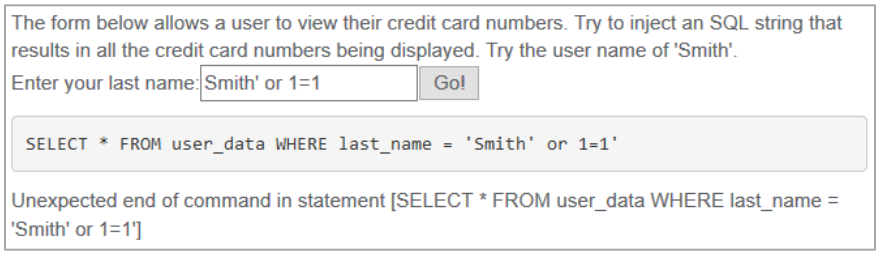
- First let's enter the name 'Smith' and see what sort of result we get:



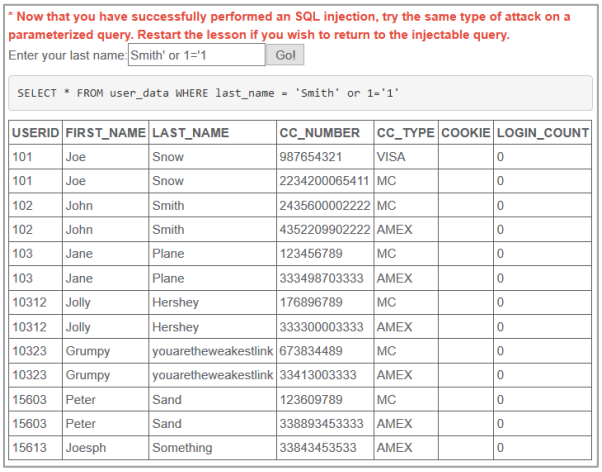
- Let's tinker a bit until we find something that works. First we'll see how the database responds to a single quote "'".



- Now we can begin creating our injection. If we use a single quote "'" after 'Smith', we can *end* the string parameter for the database query *early*, which allows us to inject code into the SQL query. We treat our string input as a *logical statement*, and OR that with a *tautology* (a statement that is always true).



- We can see from the previous error message that `'last_name = 'Smith' or 1=1''`. Let's fix that malformed string and query the database for `'Smith' or 1='1'`. This should return the entire database table:

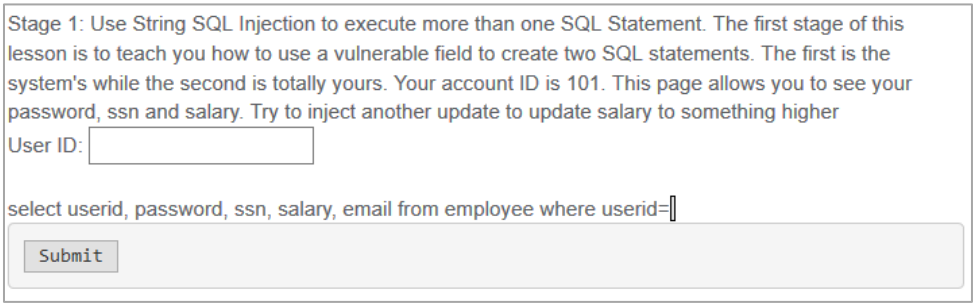


We're successful! We've cracked into this SQL database and retrieved lots of dangerous, privileged information. We see that we can trick a database transaction by treating a string as a logical statement (or), which we can then inject a tautology into (1='1'). This is a great example of a simple string SQL injection, and will guide our learning during the next exploit.

Database Backdoors

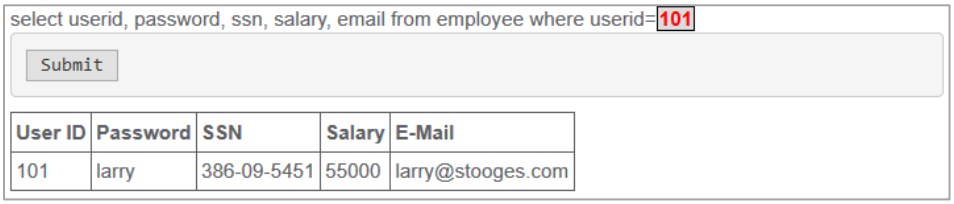
So far, we've stuck to relatively "nice" examples; simple cases of bypassing an authentication system in order to log in as a privileged user. In this next exercise, we will go a bit "above and beyond" or "off the rails", depending on how you look at it. We'll show that we can not only inject an *update* into an SQL query, but a *delete* as well. If we can gain access to an SQL database, we can manipulate every table element and indeed purge the database of data completely.

- Again, we have a simple text input field to login to a simple SQL database. Here our goal is to change our salary, using only an SQL query injection.

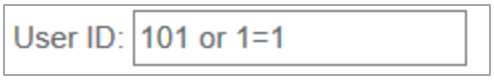


- First let's log in as Larry Stooge:

- This gives us some idea of what the database looks like.



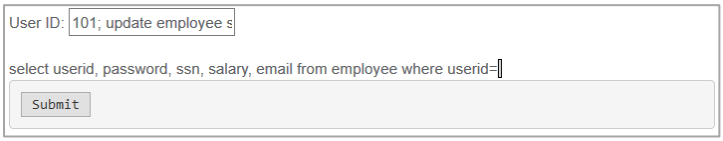
- Now let's see if we can perform a *numeric SQL injection* to see *all* of the database, as we learned in our previous exploit:



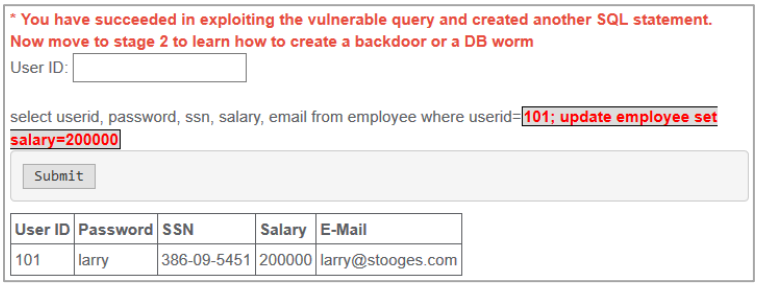
- Once again, we've cracked open the database:

User ID	Password	SSN	Salary	E-Mail
101	larry	386-09-5451	55000	larry@stooges.com
102	moe	936-18-4524	140000	moe@stooges.com
103	curly	961-08-0047	50000	curly@stooges.com
104	eric	445-66-5565	13000	eric@modelsrus.com
105	tom	792-14-6364	80000	tom@wb.com
106	jerry	858-55-4452	70000	jerry@wb.com
107	david	439-20-9405	100000	david@modelsrus.com
108	bruce	707-95-9482	110000	bruce@modelsrus.com
109	sean	136-55-1046	130000	sean@modelsrus.com
110	joanne	789-54-2413	90000	joanne@modelsrus.com
111	john	129-69-4572	200000	john@guns.com
112	socks	111-111-1111	450000	neville@modelsrus.com

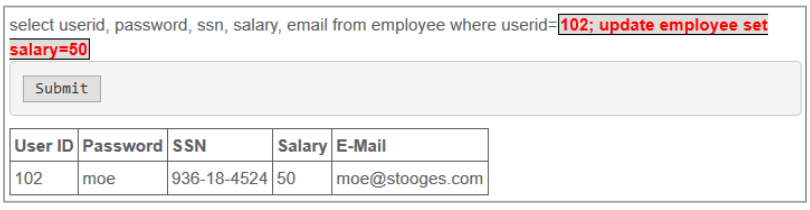
- Larry has always been the funniest Stooge, so let's give him a big raise. Enter '`101; update employee set salary=200000`'. Here we use a semicolon "`;`" to end the string parameter of the SQL query early, so that we may inject our SQL *update* into it.



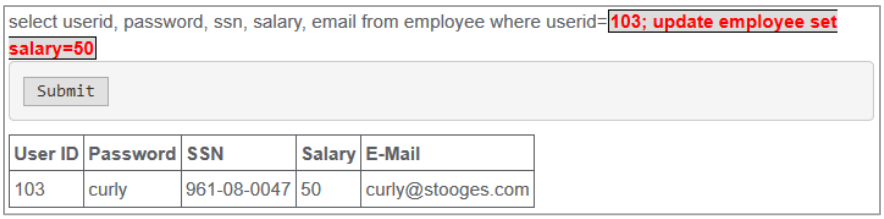
- We've successfully given Larry a much-deserved raise:



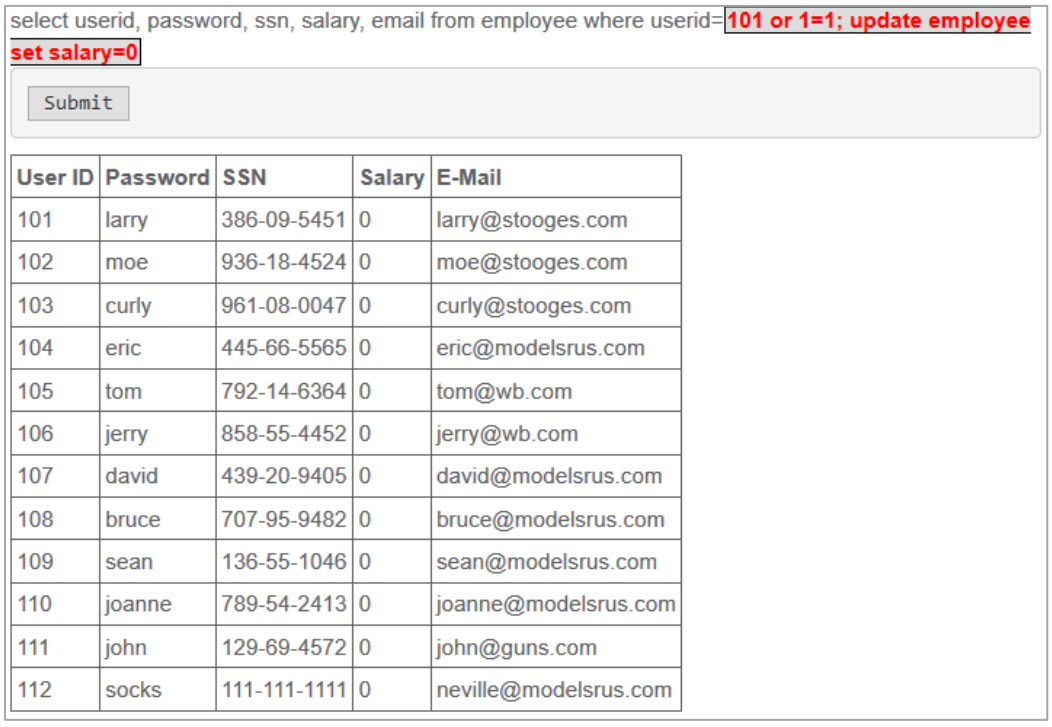
- Now let's give Moe a large pay cut. Enter '`102; update employee set salary=50`'.



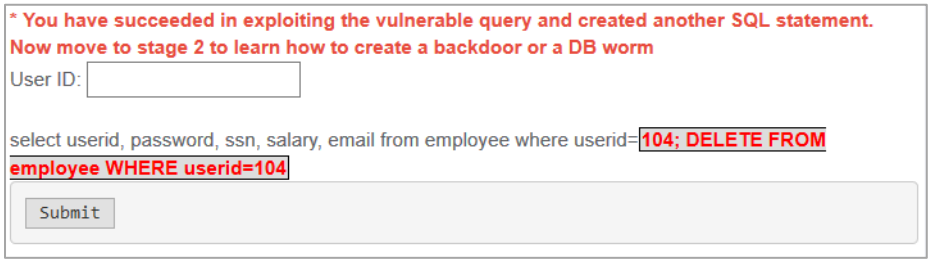
- We can do the same for Curly, who clearly hasn't been pulling his weight lately. Enter '`103; update employee set salary=50`'.



- We can give *everyone* the *maximum* pay cut using '`101 or 1=1; update employee set salary=0`'.



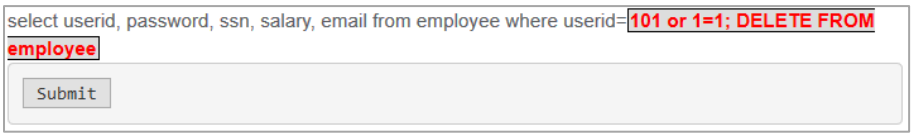
- If we want to 'fire' Eric from Models 'R Us, we can delete him from the database. Enter '`104; DELETE FROM employee WHERE userid=104`'.



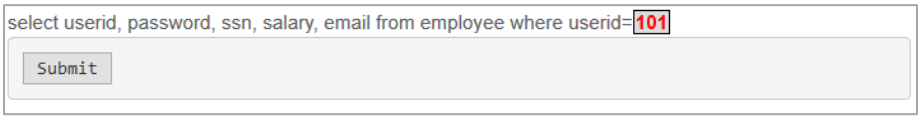
- Eric is gone forever.

User ID	Password	SSN	Salary	E-Mail
101	larry	386-09-5451	55000	larry@stooges.com
102	moe	936-18-4524	140000	moe@stooges.com
103	curly	961-08-0047	50000	curly@stooges.com
105	tom	792-14-6364	80000	tom@wb.com
106	jerry	858-55-4452	70000	jerry@wb.com
107	david	439-20-9405	100000	david@modelsrus.com
108	bruce	707-95-9482	110000	bruce@modelsrus.com
109	sean	136-55-1046	130000	sean@modelsrus.com
110	joanne	789-54-2413	90000	joanne@modelsrus.com
111	john	129-69-4572	200000	john@guns.com
112	socks	111-111-1111	450000	neville@modelsrus.com

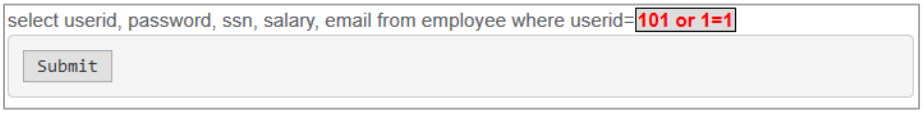
- If we want to have a 'fire sale', we can delete *everyone* from the database by entering '`101 or 1=1; DELETE FROM employee`'.



- Trying to retrieve Larry's information returns nothing.



- We also receive nothing when using our numeric injection to view the entire database. There's nothing in the database.



While we may enjoy a chuckle of malevolent glee in demonstrating this example, there is also the grim reality of how powerful and dangerous *unsanitized* SQL queries can be. The lesson here is to *sanitize your inputs and queries* and to *make backups*. We were able to restart the WebGoat server to recover the database here, but in the real world we might not be so lucky.

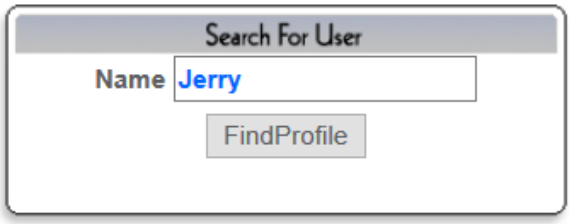
Part 5: Cross Site Scripting (XSS)

So far, we've seen that we can take over your file system, we can take over your database, but what about the *browser* itself? We've explored the back end of web security, now let's explore the front end/client side. Most browsers these days are equipped to run JavaScript, a scripting language which enables many dynamic features for web applications. Here we will use a *Cross Site Scripting (XSS)* attack to store some JavaScript to create a browser alert on the client side whenever a particular database item is viewed in FireFox.

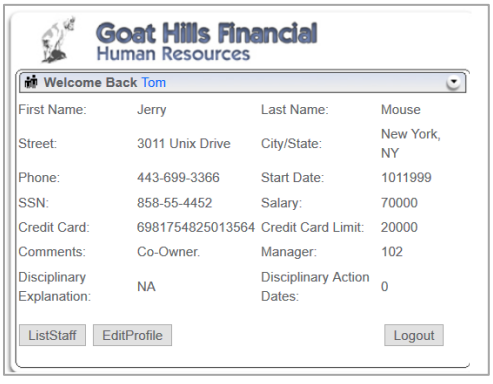
Stored XSS

- First let's login as Tom Cat, with password 'tom'.

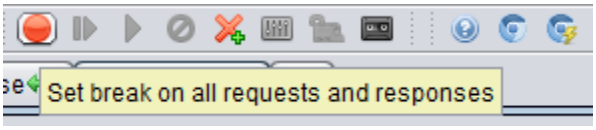
- Now let's find Jerry's profile:



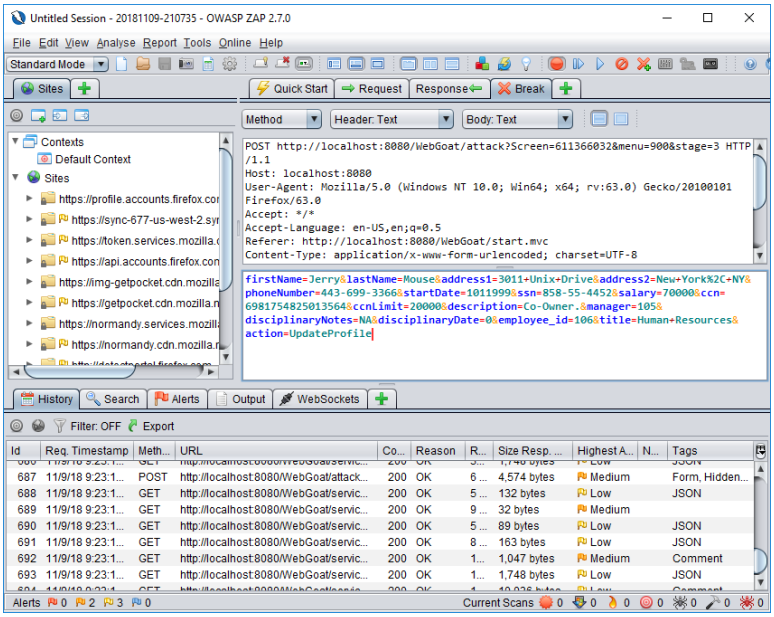
- Jerry's profile information, as viewed by Tom:



- Now let's set a break point in ZAP so we can perform our XSS attack:



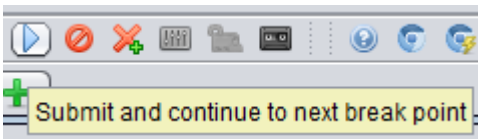
- In WebGoat click on 'Edit Profile' to view the contents of the HTTP request in ZAP:



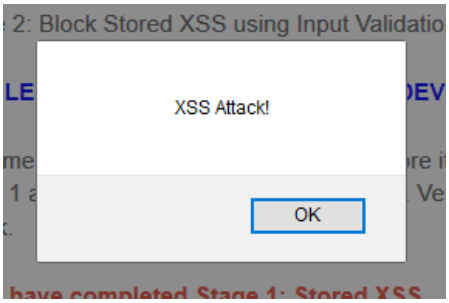
- Add the script '`<script>alert("XSS Attack!")</script>`' after Jerry's address in the 'address1' field to cause a browser alert whenever Jerry's profile information is viewed:

```
firstName=Jerry&lastName=Mouse&address1=3011+Unix+Drive<script>alert("XSS Attack!"
)</script>&address2=New+York%2C+NY&phoneNumber=443-699-3366&startDate=1011999&ssn=
858-55-4452&salary=70000&ccn=6981754825013564&ccnLimit=20000&description=Co-Owner.
&manager=105&disciplinaryNotes=NA&disciplinaryDate=0&employee_id=106&title=Human+R
esources&action=UpdateProfile
```

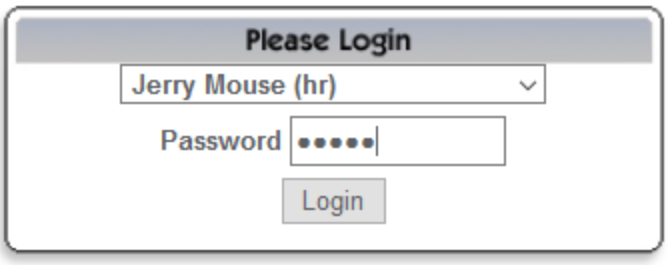
- Allow ZAP to continue the WebGoat service:



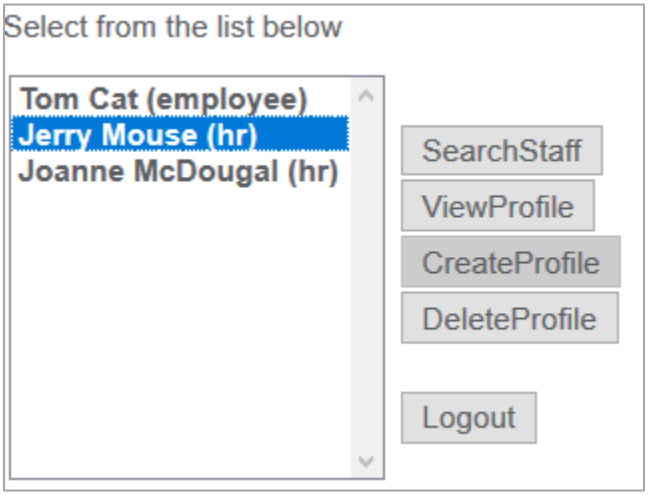
- In FireFox we can see that our script is running as intended:



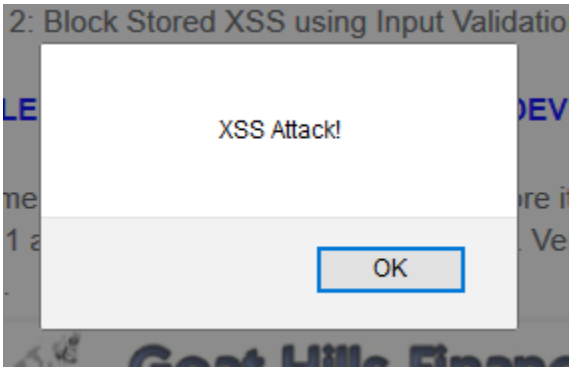
- Now let's login as Jerry to verify our Stored XSS Attack.



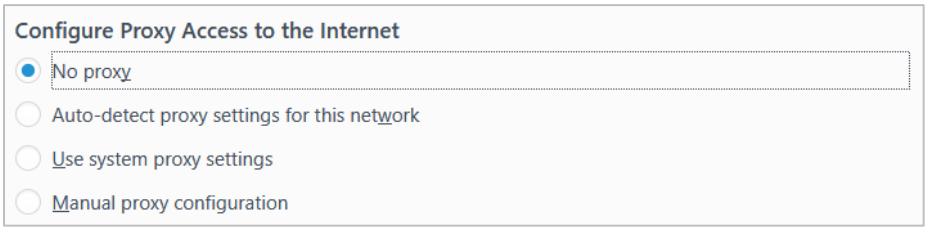
- Select Jerry Mouse and click 'View Profile':



- Jerry is under attack! We've successfully used a Stored XSS attack to store a JavaScript alert within the address of Jerry's profile.



- Now that we've completed our work, let's clean up our work environment. In the Windows command line enter 'Ctrl + C' to stop the WebGoat server. In Firefox, click on the Menu button, click 'Options', then scroll down to Network Settings and select 'Settings'. Select 'No proxy' to disable the proxy server we used to listen in on WebGoat with ZAP. Close ZAP.



Conclusion

We can't deny learning about web security is both fun and fascinating. What's not fun is the billions of dollars lost worldwide every year due to cybercrime. We've demonstrated some simple examples of web attacks here; we gained access to a filesystem, browser, and an entire database using various means of tricking WebGoat into misbehaving. As web developers we will need to be intimately responsible for every single input field and database transaction in our application. Our applications are only as secure as we can write defenses for attacks with which we are familiar. We must continually seek out and understand newer and more clever forms of web attacks if we want to defend against them. In the cyber world, knowledge is our first line of defense; we're thankful for tools like WebGoat that allow us to explore web security in a safe learning environment.

Bibliography

Gyurko, S. (2018). *Under the cover of Docker* [website]. Retrieved from https://medium.com/@TribalWorldwide/under-the-cover-of-docker-391e2b24ba37.

Docker Docs (2018). *Get started with Docker for Windows* [website]. Retrieved from https://docs.docker.com/docker-for-windows/.

Docker Hub (2016). *Webgoat/webgoat-7.1* [website]. Retrieved from https://hub.docker.com/r/webgoat/webgoat-7.1/.

OWASP (2016). *Webgoat 7.1* [computer software].

w3schools (2018). *SQL DELETE statement* [website]. Retrieved from https://www.w3schools.com/sql/sql_delete.asp.

Instructor Feedback

What was too difficult, too easy?

By far the most difficulty I had with this assignment was setting up Docker. WebGoat was a joy.

What would have made the learning experience better?

Perhaps a more in-depth exploration of ZAP and other similar tools. I would love to put some of my knowledge to practice and sanitize against injections and other attacks firsthand.

What did you learn?

I learned a great deal more about WebGoat and different kinds of web-based attacks. I have a more thorough understanding of the different kinds of injections we can perform.

How did you learn it?

In general, WebGoat itself is a great resource, though I leaned more on outside web-based resources in learning some SQL query syntax.