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.

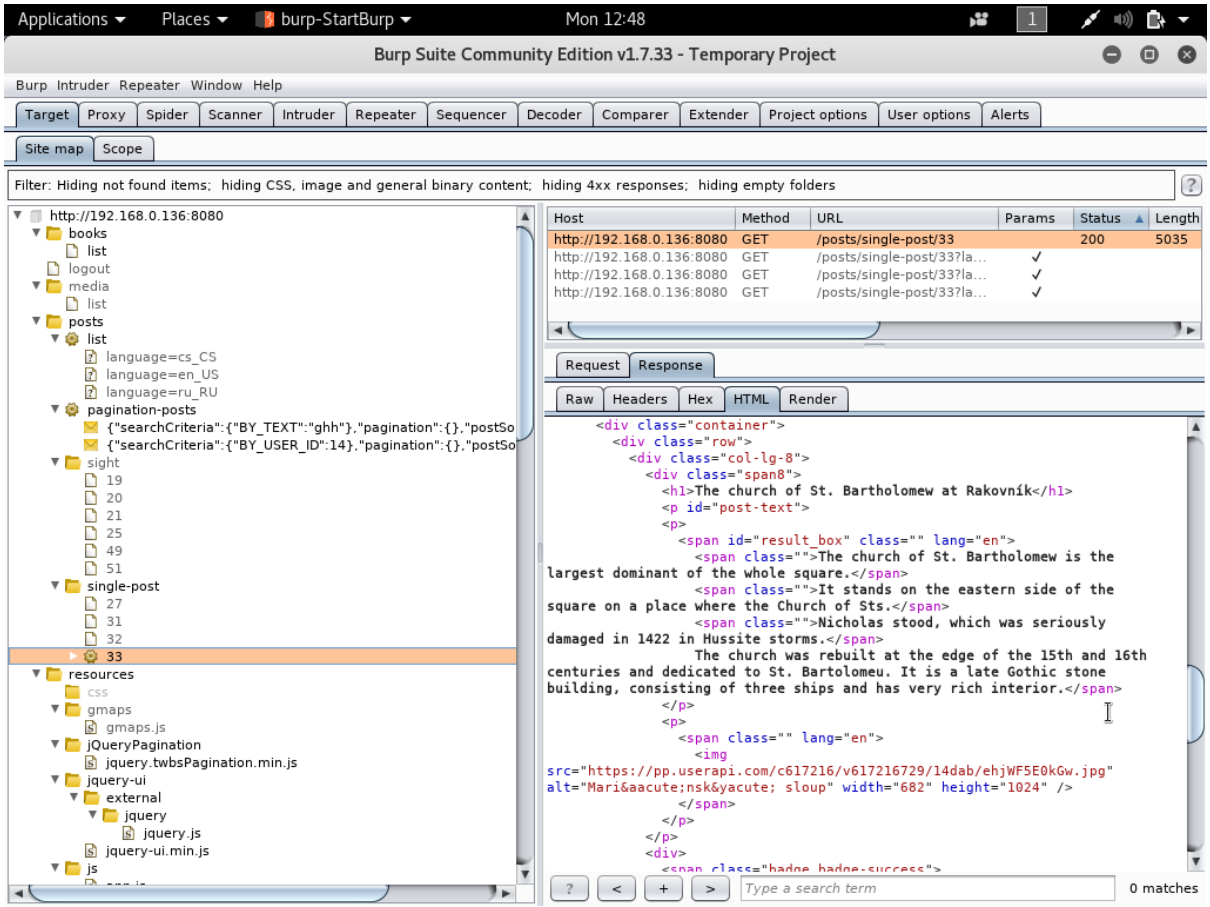


Figure 1: Basic structure of web application Diary for travelers

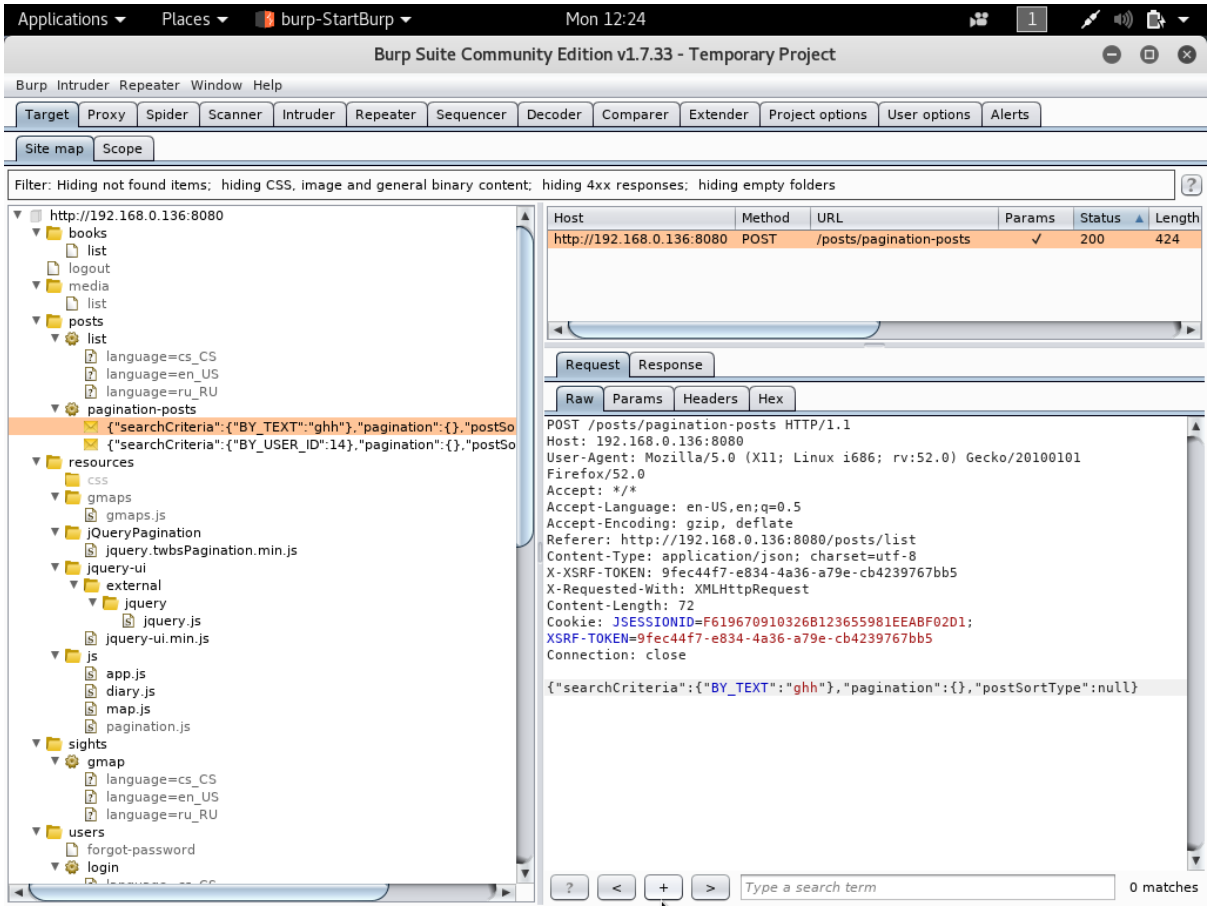


Figure 2: Discovering functional paths by Burp

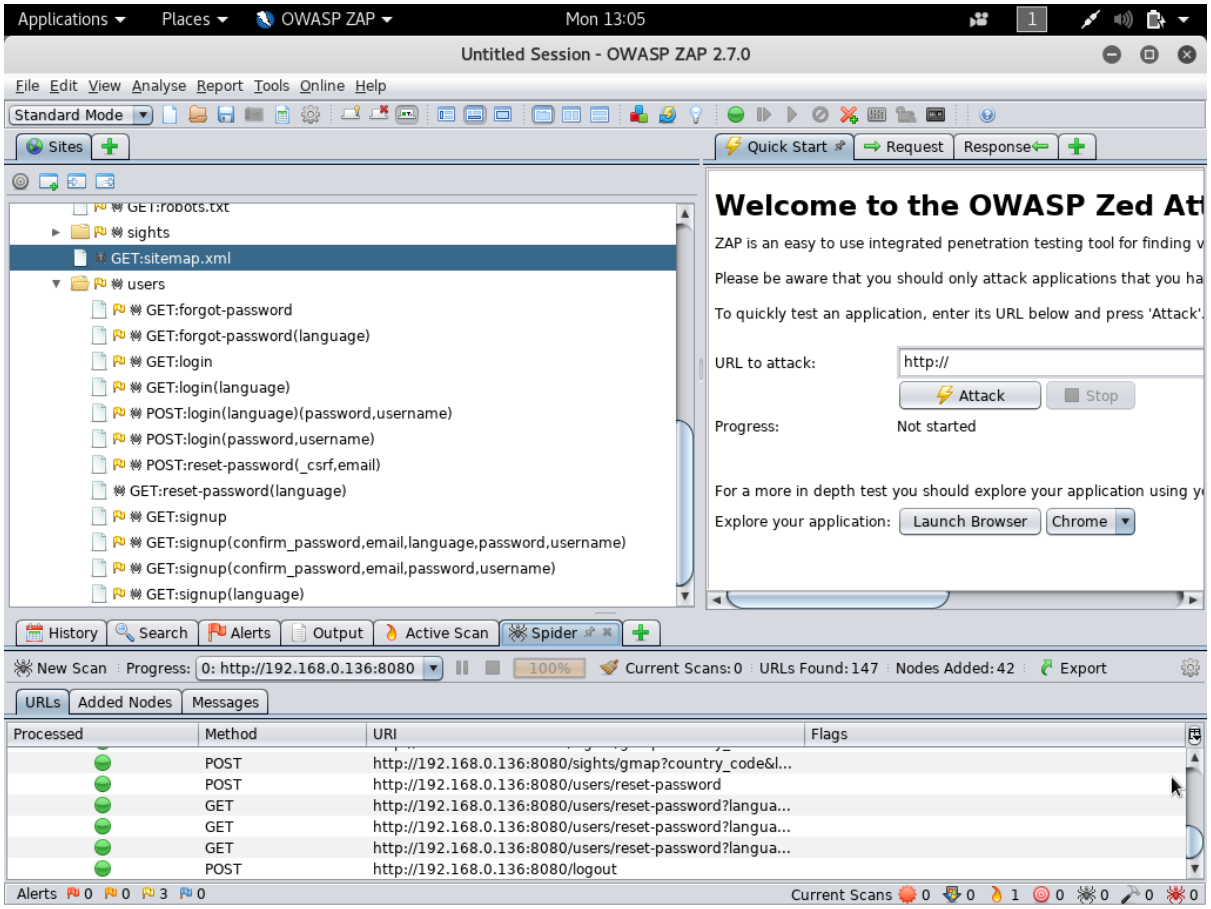


Figure 3: Discovering user's functionality by OWASP ZAP

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].

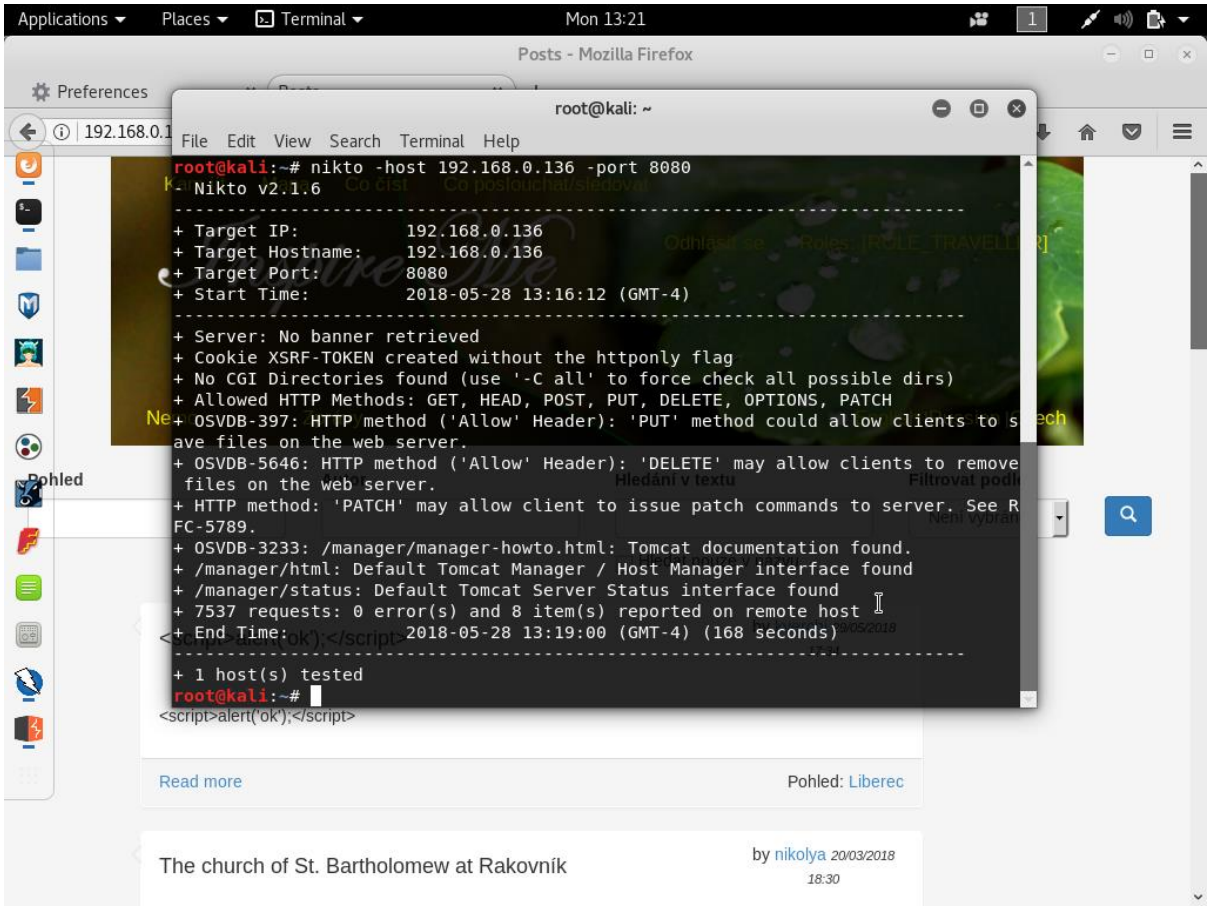


Figure 4: Using Nikto tool for scanning application's web server

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:

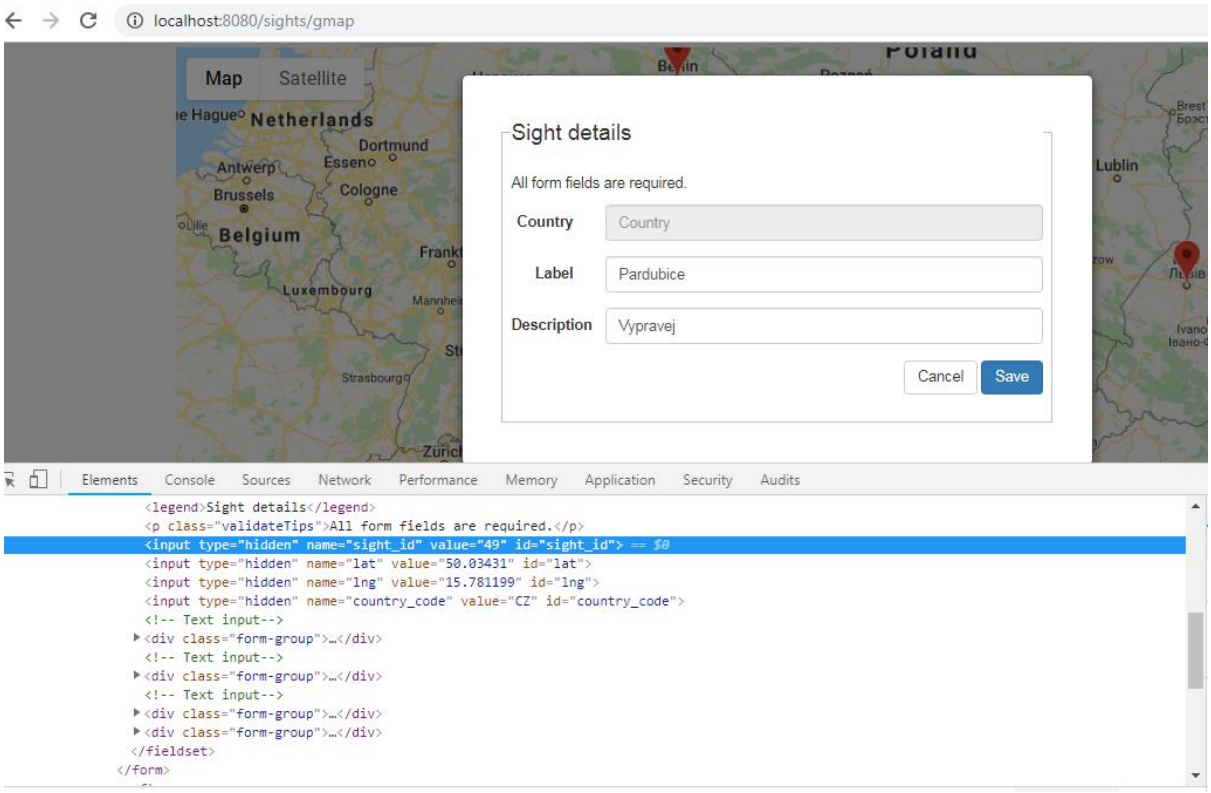
- Client-side validation: if checks are replicated on the server side
- SQL injection
- Cross-site scripting
- Login functionality: username enumeration, weak passwords, ability to use brute force login credentials
- Session: predictable tokens, insecure handling of tokens, session hijacking, capture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

- Weaknesses of application components: known vulnerabilities and configuration weaknesses

Bypassing Client-Side Controls

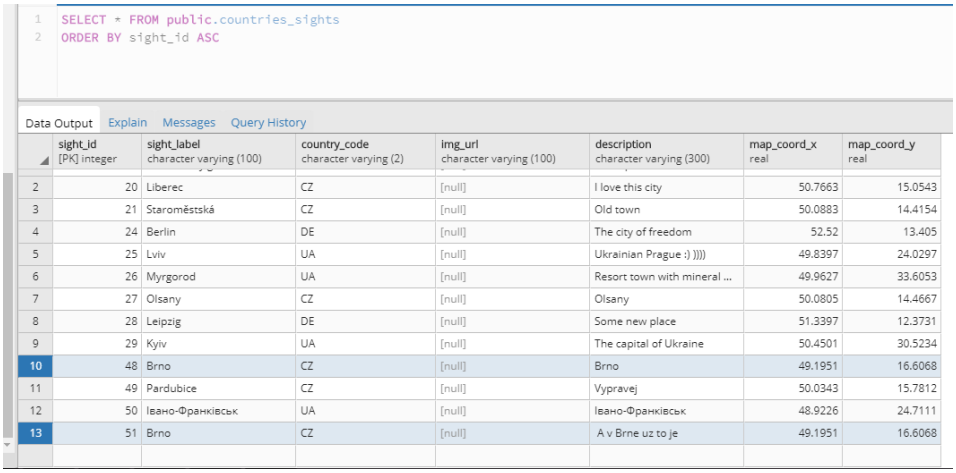
Hidden Form Fields

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.



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
```

Data Output | Explain | Messages | Query History

	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

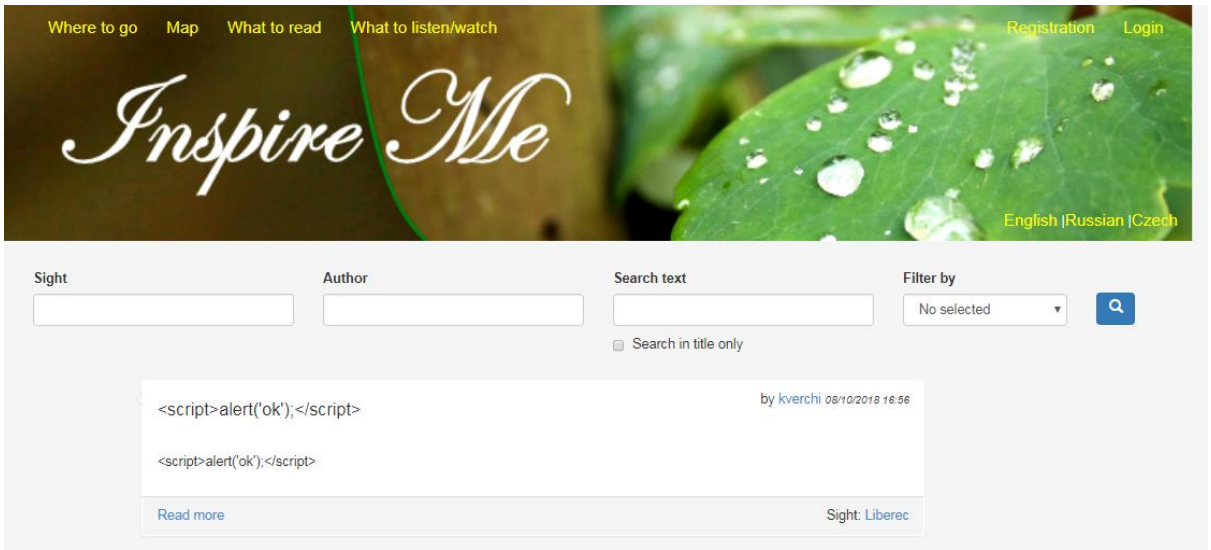
Picture: A sight with ID 51 was overrode by sight with ID 48

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

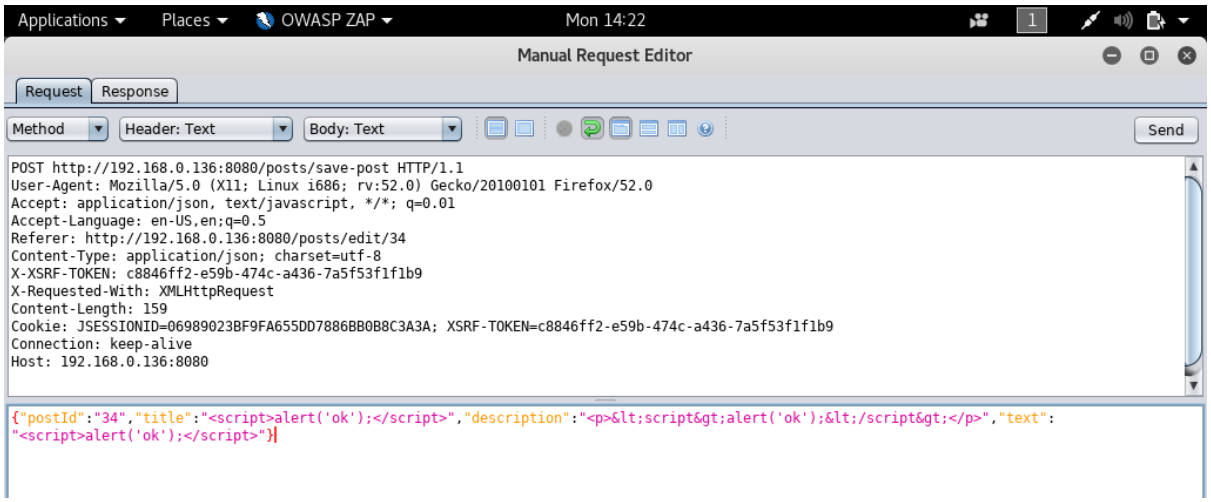
If submit the segment of JavaScript code as a text of a new post, it will be stored in database without any user's input sanitization.



Picture: Result of submitting a piece of JavaScript code

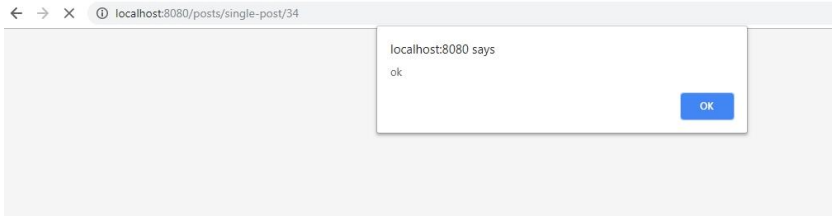
Probably there is some template engine escapes html and JS tags, so it's impossible to reproduce stored XSS attack. But there is a stored XSS attack in this web application [2].

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.



Picture: Updating the HTTP request with OWASP ZAP interception tool

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



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.

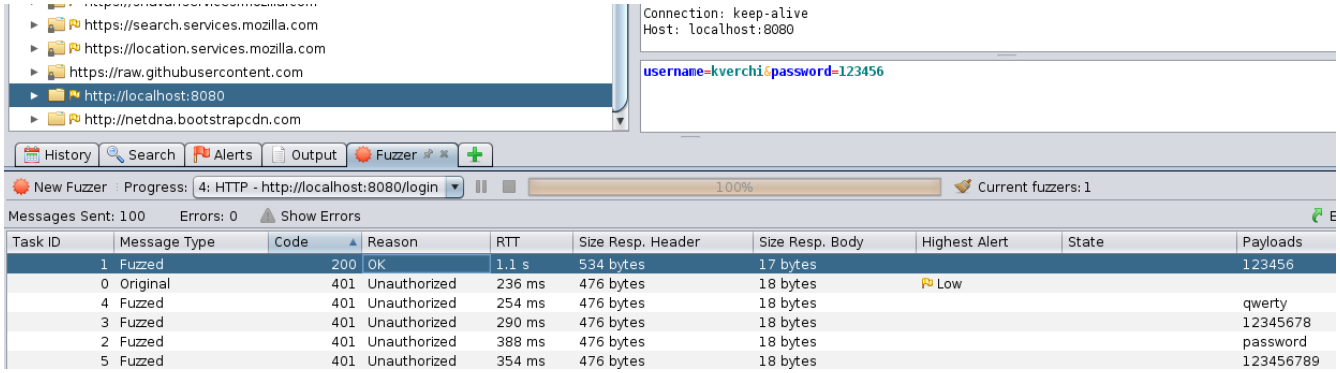


Figure 5: An account with weak password

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.

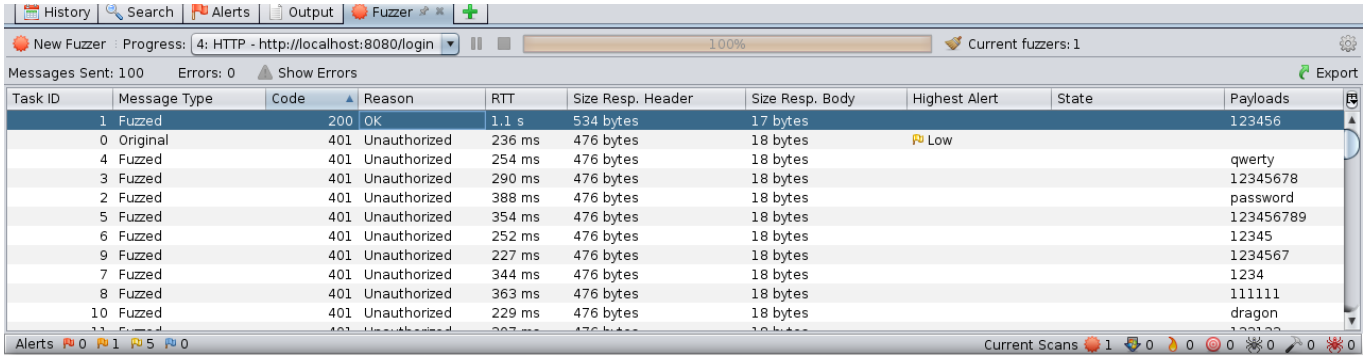


Figure 6: Unlimited attempts to guess a password

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].

Conclusion

Risk rating

Recommendations

Links
[1] –
[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