REVERSE PROXY EXERCISE

What is a Reverse Proxy?

Reverse proxy is server side component that intercepts traffic and relays them back to the back-end servers. This prevents direct exposure of backend server. The reverse proxy can act as a load balancer, security filtering, authentication and enforce security policy. The reverse proxy that works in the Application layer (HTTP) are HTTP reverse proxy.

Problems in current proxy?

Observation:

The current proxy implementation gets the URL of the backend server and makes a curl call to the server. Does simple authentication with a static API-key. Adds x-frame-options header with ‘SAMEORIGIN. The PHP script doesn’t contain the end tag. But the PHP documentation says this is valid and in fact it is better, since it avoids any additional spaces

Issues:

- No checks on the target backend server. Essentially acts an open proxy
- Since the backend server is a parameter it restricts the type of traffic it can relay
- No security filtering, load balancing etc.
- Credential (API key) in the URL
- The proxy returns internal servers errors from back end without filtering

New Reverse Proxy

Tools:

Java Spring Boot framework, Netflix Zuul proxy, ESAPI

Testing Demo App:

bWAPP (buggy web app for testing)

Features:

1) Reverse proxy relay and routing for all traffic
2) Map to allowed back-end servers
3) Simple static key based authentication using authorization header mimicking oauth type token
4) Security header: XSS, XFrame, CSP, Cache-control etc.
5) Security filtering: Javascript and SQL Injection
6) Logging of responses and requests

Working:

The proxy server relays all http traffic based on the mapping present in application properties. In this example, we use this proxy to relay traffic to bWAPP application. This application is inherently buggy.

Security Headers:

The security header configurations are done in class WebSecurityHeader.java. Few points to know:

- Disabled CSRF headers due to the nature of the demo app (bWAPP). But this must be enabled in reality

- Disabling HSTS again for convenience

Filters:

The requests and response goes through series of filter to inspect traffic and log packets. The current implementations have 4 filters:

- Request log Filter – logs requests
- Response log Filter – logs response headers
- Token Authentication Filter – authenticates the token in authorization header
- Security Filter + Injection Filter – Does Script and SQL injection filtering and encoding

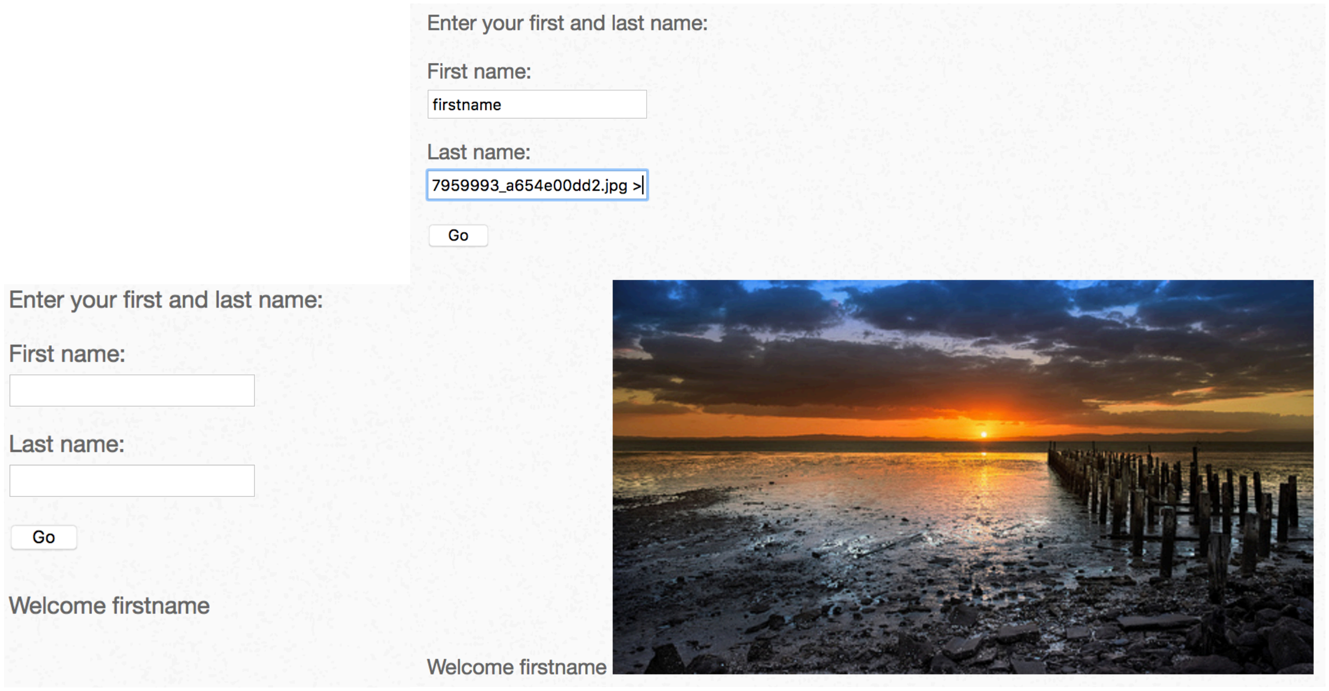
Screen shots of the working:

CSP in action:

Prevents loading img from another source

Payload:

<img src=http://farm4.staticflickr.com/3826/12897959993_a654e00dd2.jpg >


Enter your first and last name:

First name:

firstname

Last name:

7959993_a654e00dd2.jpg >

Enter your first and last name:

First name:

Last name:

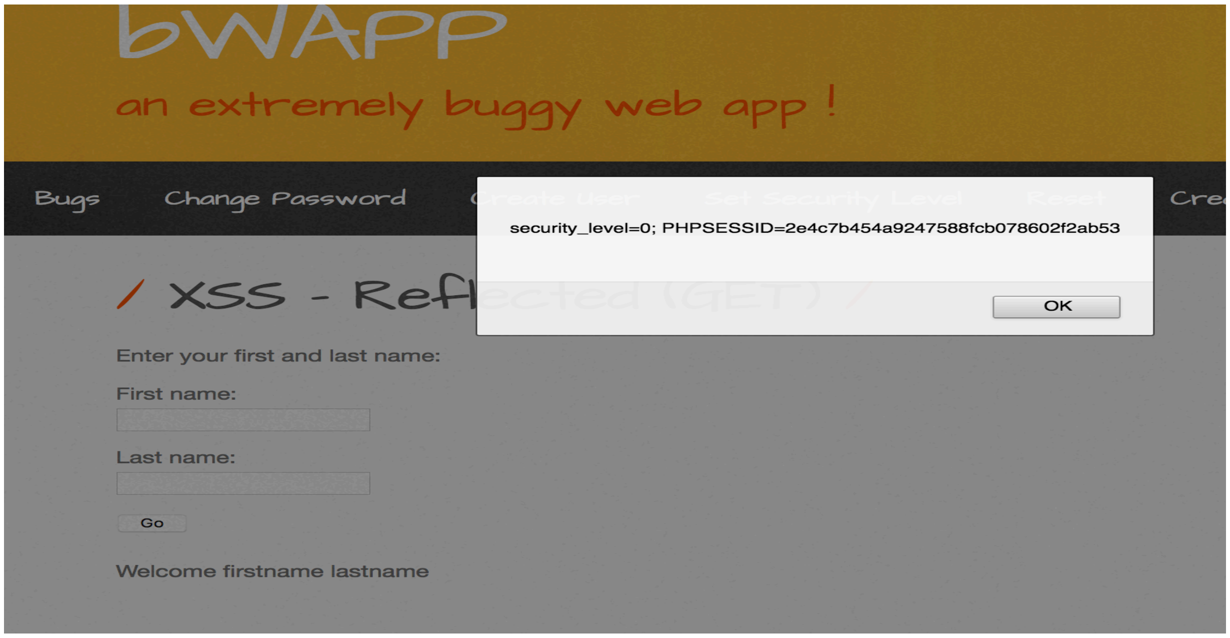
Welcome firstname

Welcome firstname

XSS Filtering:

Prevents execution of XSS in get params:

Payload: lastname <script>alert(document.cookie);</script>



security_level=0; PHPSESSID=2e4c7b454a9247588fcb078602f2ab53

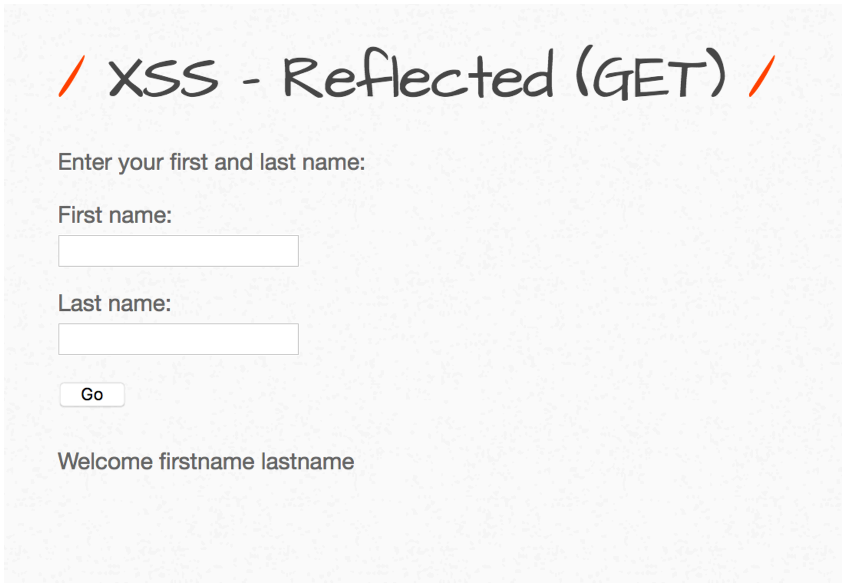
/ XSS - Reflected (GET) /

Enter your first and last name:

First name:

Last name:

Welcome firstname lastname



/ XSS - Reflected (GET) /

Enter your first and last name:

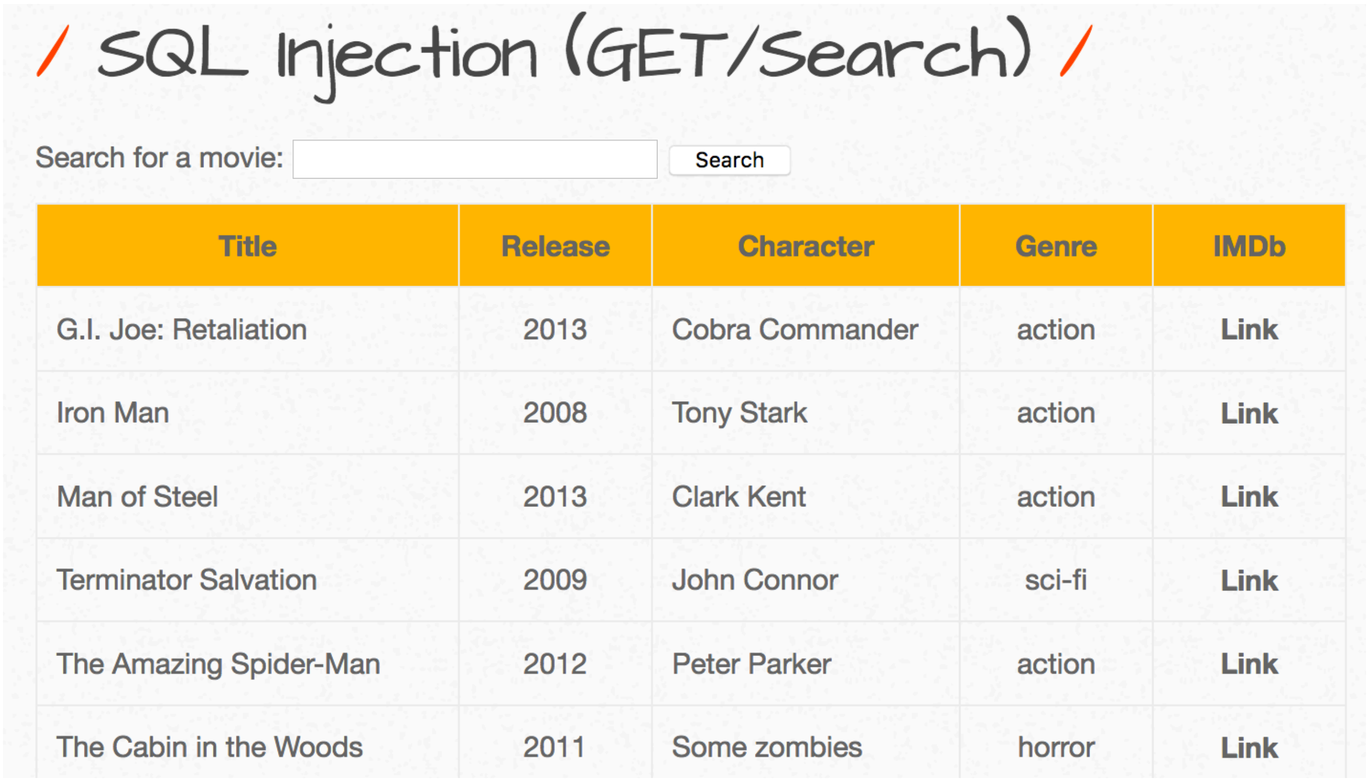
First name:

Last name:

Welcome firstname lastname

SQL Injection Filtering:

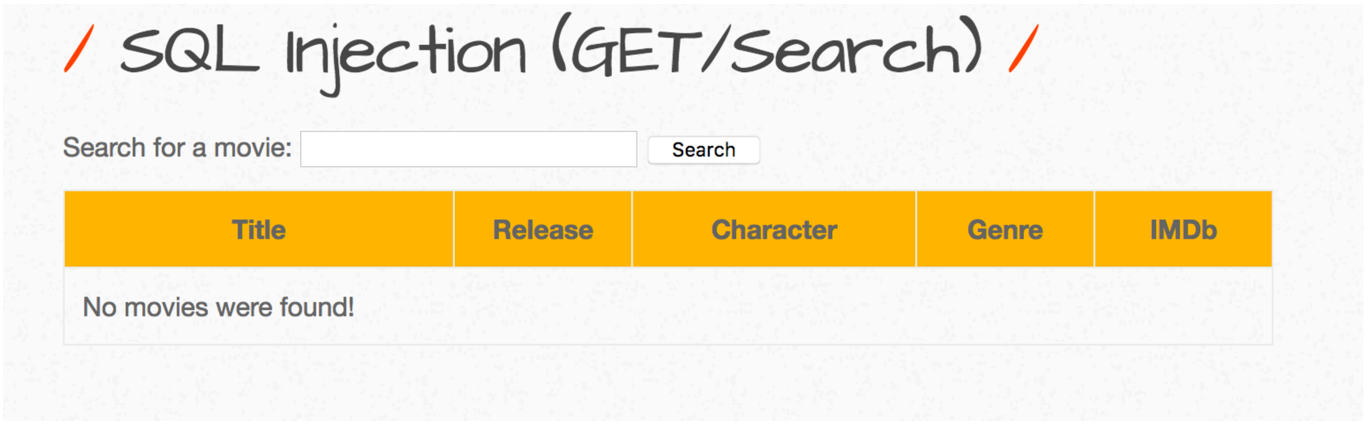
Payload: ' OR '1' = '1



/ SQL Injection (GET/Search) /

Search for a movie: Search

Title	Release	Character	Genre	IMDb
G.I. Joe: Retaliation	2013	Cobra Commander	action	Link
Iron Man	2008	Tony Stark	action	Link
Man of Steel	2013	Clark Kent	action	Link
Terminator Salvation	2009	John Connor	sci-fi	Link
The Amazing Spider-Man	2012	Peter Parker	action	Link
The Cabin in the Woods	2011	Some zombies	horror	Link



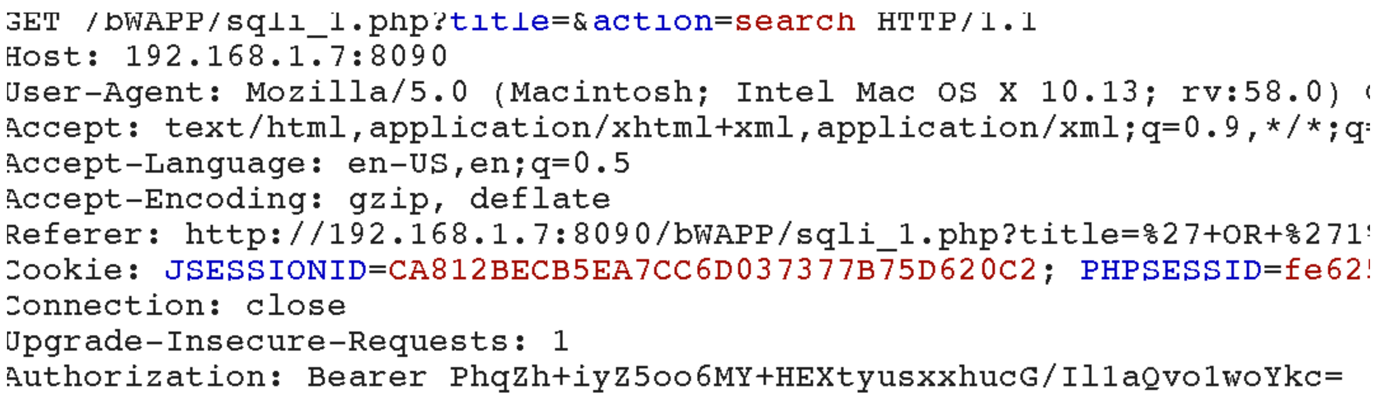
/ SQL Injection (GET/Search) /

Search for a movie: Search

Title	Release	Character	Genre	IMDb
No movies were found!				

Authentication:

Authentication with static token in authorization header:



```
GET /bWAPP/sqli_1.php?title=&action=search HTTP/1.1
Host: 192.168.1.7:8090
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:58.0)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.7:8090/bWAPP/sqli_1.php?title=%27+OR+%271
Cookie: JSESSIONID=CA812BECB5EA7CC6D037377B75D620C2; PHPSESSID=fe62
Connection: close
Upgrade-Insecure-Requests: 1
Authorization: Bearer PhqZh+iyZ5oo6MY+HEXtyusxxhucG/Il1aQvo1woYkc=
```

Improvements:

List of improvements that can be done:

1) Authentication is too simple with static keys and no authorization
2) Filtering satisfies for very basic conditions and no encoding in response
3) Not filtering sensitive headers while logging
4) Very minimal testing