Web Security Part 1: Passwords & SQL Injection

Summer University 2014

Prof. Tom Austin
San José State University

A Little About Me…

- PhD from UC Santa Cruz
- Master's from San José State University
- Attended Summer University in 2007 at HEIG-VD
- Research interests:
  - Programming Languages
  - Security
  - Web Applications

A Little About Me…

- PhD from UC Santa Cruz
- Master's from San José State University
- Attended Summer University in 2007 at HEIG-VD
- Research interests:
  - Programming Languages
  - ***Security***
  - ***Web Applications***

Note for the Swiss Students

- I talk fast…
- … if I talk too fast, remind me to slow down.

Modern Secu…

Heroes and Villains

Computer security is generally taught from the defender's perspective.

In this session, we focus on the opposite side and think like an attacker.

Foolish Expectations

This session assumes some knowledge of:

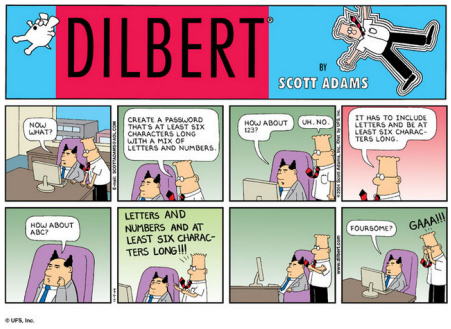
- HTML
- SQL
- Dynamically generated web pages

We will briefly review these technologies

Note for labs

- This course includes several labs
- Work in teams of 3-4 students
  - Pick partners from different universities
  - I will assign you a group name
  - Replace **<group>** with your group name
- Submit your labs by paper or email.
  - One submission by group
  - Include the name & school of each team member
  - If you email me, use thomas.austin@sjsu.edu
- **SJSU students**: if you took CS166 with me and know the answer, don't give it away

Passwords



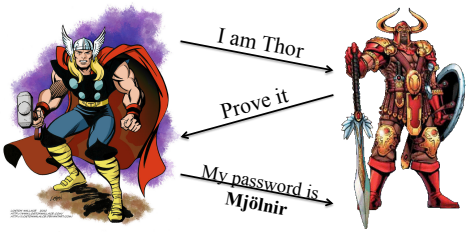
Authentication: Are You Who You Say You Are?

- How to authenticate human a machine?
- Can be based on…
  - Something you **know**
    - For example, a password
  - Something you **have**
    - For example, a smartcard
  - Something you **are**
    - For example, your fingerprint

Why Passwords?

- Why is "something you know" more popular than "something you have" and "something you are"?
- **Cost**: passwords are free
- **Convenience**: easier for admin to reset pwd than to issue a new thumb

Authenticating with Passwords



Authenticating with Passwords

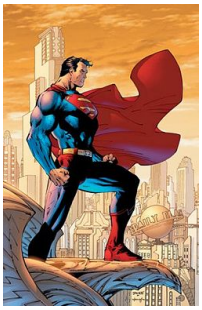
Authenticating with Passwords

Trouble with Passwords

- "Passwords are one of the biggest practical problems facing security engineers today."
- "Humans are incapable of securely storing high-quality cryptographic keys, and they have unacceptable speed and accuracy when performing cryptographic operations. (They are also large, expensive to maintain, difficult to manage, and they pollute the environment. It is astonishing that these devices continue to be manufactured and deployed.)"

Common advice given for passwords

- Do not reuse passwords for different sites
- Passwords should include:
  - mixed case
  - numbers
  - punctuation
- Everyone has heard this advice
- **No one follows it**



What makes a good password?

Good or Bad Password?

frank

Obviously bad – easy to guess

Good or Bad Password?

02191976

Bad – a birth date

Good or Bad Password?

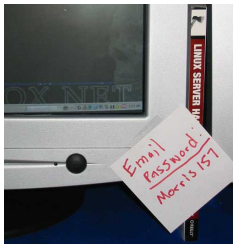
jfIej,43j-EmmL+y

Hard to guess…

…but also hard to remember.

So what does the user do?

He or she will put the password on a sticky note on their monitor.



Good or Bad Password?

09864376537263

????

If this is really random, then it is hard to remember.

If this is *not* random, then it is easy to guess.

Good or Bad Password?

4Sa7Yago

Hard to guess, but hard to remember…

...or is it?

Good or Bad Password?

4Sa7Yago

"Four score and seven years ago…"

This is an example of a *passphrase*

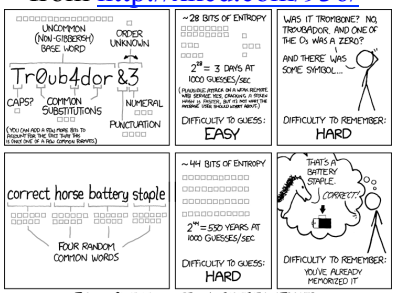
Password Experiment

- Three groups of users — each group advised to select passwords as follows
  - **Group A:** At least 6 chars, 1 non-letter
  - winner → **Group B:** Password based on passphrase
  - **Group C:** 8 random characters
- Results
  - **Group A:** About 30% of pwds easy to crack
  - **Group B:** About 10% cracked
    - Passwords easy to remember
  - **Group C:** About 10% cracked
    - Passwords hard to remember

Password Experiment

- User compliance hard to achieve
- In each case, 1/3rd did not comply
  - And about 1/3rd of those easy to crack!
- Assigned passwords sometimes best
- If passwords not assigned, best advice is…
  - Choose passwords based on passphrase
  - Use pwd cracking tool to test for weak pwds
- Require periodic password changes?

"Correct horse battery staple" from http://xkcd.com/936/



Password game

Remember this pass phrase:

spooky hook UFO pathology

Password game

What was the password on the previous slide?

spooky hook UFO pathology

Password game

Now remember this password:

4rx99t3ch!

Password game

What was the password on the previous slide?

4rx99t3ch!

But do you still remember the pass phrase?

spooky hook UFO pathology

The problem

There are ways of choosing strong passwords, but many actual passwords are easily guessed.

Breaking Passwords

Problem 1.1: Find a login

Time to think like a villain:

1. Go to `http://cs31.cs.sjsu.edu/<group>/login1.php`.
2. Try to log in. Find some common passwords:
   - http://www.zdnet.com/blog/security/25-most-used-passwords-revealed-is-yours-one-of-them/12427
3. If you succeed, write down:
   - username
   - password
   - steps you took to determine this password

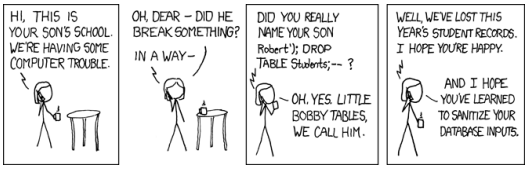
Some logins you may have discovered

Username	Password
aquaman	fish
guest	guest
admin	admin123
wolverine	harley
superman	superman
wonderwoman	letmein
spiderman	password

Searching for common passwords can be effective, but is time-consuming.

Other vulnerabilities allow information to be stolen quicker.

SQL Injection



"Exploits of a Mom", http://xkcd.com/327/

Crash Course in SQL

- The database used on the server is PostgreSQL
- For those not familiar with PostgreSQL, note that you should use single quotes and not double quotes.
- For more details, see http://www.postgresql.org/

SELECT statements

- Used to fetch information from the database
- **SELECT** clause specifies the fields
- **FROM** clause specifies the name of the table
- **WHERE** restricts what values are returned.
- Returns a table of results

SELECT example

```
SELECT id, name
FROM hero
WHERE status='alive'
```

hero table

id	name	status
1	'Batman'	'alive'
2	'Rorschach'	'dead'
3	'Spiderman'	'alive'
4	'Superman'	'unknown'

id	name
1	'Batman'
3	'Spiderman'

UNION keyword

- The **UNION** keyword combines the results of two different queries
- This may be particularly useful for an attacker trying to steal information
- ***Note**: both queries must return the same number of columns.*
- Example:
  SELECT name, status FROM hero **UNION**
  SELECT name, location FROM villain

INSERT statements

- Used to add new records to the database
- **INSERT INTO** clause specifies the table name and the field names
- **VALUES** clause specifies values for each field
- Example:

```
INSERT INTO hero (name, status)
  VALUES ('Iron Man', 'alive');
```

INSERT example

```
INSERT INTO hero
 (name, status)
 VALUES ('Iron Man',
 'alive');
```

hero table

id	name	status
1	'Batman'	'alive'
2	'Rorschach'	'dead'
3	'Spiderman'	'alive'
4	'Superman'	'unknown'
5	'Iron Man'	'alive'

Generated automatically (in this case)

UPDATE statements

- Updates an existing record in the database
- **UPDATE** clause specifies a table to update
- **SET** indicates which fields should be updated
- **WHERE** clause limits records that are updated
- Example: UPDATE hero
  SET status = 'alive'
  WHERE name = 'Superman'

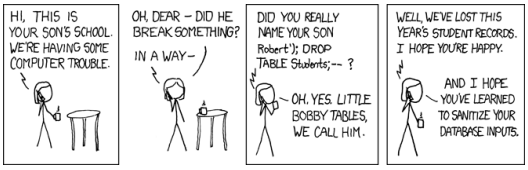
UPDATE statements

```
UPDATE hero
    SET status='alive'
    WHERE name='Superman'
```

id	name	status
1	'Batman'	'alive'
2	'Rorschach'	'dead'
3	'Spiderman'	'alive'
4	'Superman'	**'alive'**
5	'Iron Man'	'alive'

SQL injection

If an input string is not sanitized, it could become part of a SQL query string



How would this attack work?

An improperly sanitized query

```
<?php                          name = "Joe"
pg_query($con,
  "INSERT INTO STUDENTS (name)"
  . " VALUES('" . $_POST['name'] .
  "')";
?>
```

```
INSERT INTO STUDENTS (name)
   VALUES('Joe')
```

An improperly sanitized query

```
<?php                          name = "Robert');
                               DROP TABLE STUDENTS;--"
pg_query($con,
  "INSERT INTO STUDENTS (name)"
  . " VALUES('" . $_POST['name'] .
  "')";
?>
```

Quote ends data: now in SQL

```
INSERT INTO STUDENTS (name)
   VALUES('Robert');
DROP TABLE STUDENTS;--')
```

Remainder commented out.

Correctly sanitized query

```
<?php                          name = "Robert');
                               DROP TABLE STUDENTS;--"
pg_query_params($con,
  "INSERT INTO STUDENTS (name)"
  . " VALUES($1)",
  ARRAY($_POST['name']);
?>
```

Quote escaped.

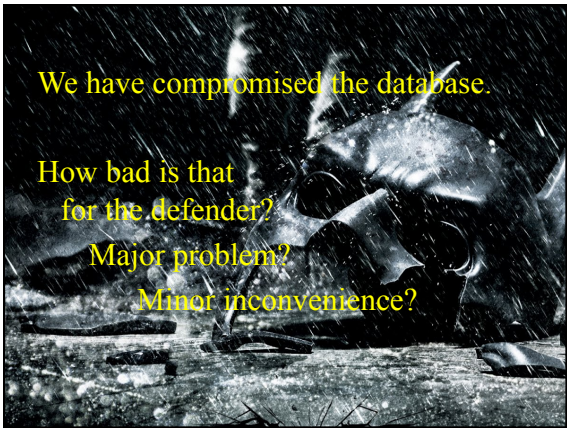
```
INSERT INTO STUDENTS (name)
   VALUES('Robert');
DROP TABLE STUDENTS;--')
```

Problem 1.2: Get all login credentials

Go to `http://cs31.cs.sjsu.edu/<group>/thanks.php`.
Login credentials are in the `user1` table, which contains `id`, `username`, & `password`.

Using SQL injection:

a. Determine login credentials from the `user1` table. Submit the complete list.
b. Add a new account to the `user1` table. Write down the username and password.



Now that you have access, you can exploit it to make your job easier.

Seek out passwords, documentation, architecture diagrams.

Problem 1.3: Corrupting information

Exploit your access to corrupt information on the database, describing your steps.

a. Add Commissioner Gordon to the list of villains.
b. Change the status of the Joker to "reformed".
c. Delete the record for Talia al Ghul.

The problem was that the passwords were stored in *plaintext*.

A better approach: store the *hash values* of the passwords rather than the actual passwords.

Cryptographic hash functions

- Hash functions compress input to a small, fixed-length output.
  - hash("Some long input") = 87d9417d8cd12635
- A good hash function should be:
  - **one-way**: given a value y it is infeasible to find an x such that h(x) = y
  - **collision-resistant**: infeasible to find ***any*** x and y, with $x \neq y$ such that h(x) = h(y)
  - **efficient?**

Some Popular Cryptographic Hashes

- MD5: 128 bit output
  - MD5 ("secret") = 5ebe2294ecd0e0f08eab7690d2a6ee69
  - MD5 has been broken, but is still widely used (though it should not be)
- SHA-1: 160 bit output
  - SHA-1("secret") = e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4

Reverse-lookup tables

Although a cryptographic hash is not reversible, it is possible to construct tables of common hashes.

MD5 hash value	Password
5ebe2294ecd0e0f08eab7690d2a6ee69	"secret"
084e0343a0486ff05530df6c705c8bb4	"guest"
5f4dcc3b5aa765d61d8327deb882cf99	"password"
482c811da5d5b4bc6d497ffa98491e38	"password123"
0d107d09f5bbe40cade3de5c71e9e9b7	"letmein"

Problem 1.4: Attacking hashed logins

An improved login page is at `login2.php`, which uses table `user2`.

Determine as many usernames and passwords as you can. These resources may be helpful:

- http://md5.gromweb.com
- http://www.stringfunction.com/md5-decrypter.html

Salted hashes

- With lookup tables, a single hash allows you to check all passwords for matches.
- Using *salt values* forces the attacker to check each password individually.

Salt	Password	Hash
"AE"	"secret"	25c2f2345300e540d4f2b6a86002874e
"19"	"secret"	675c17712c444cd7512ceadb29fde6cf
"E0"	"secret"	d0633e11f62c38ad06d13545908ee223
"0B"	"secret"	5fb6bf90896adb43a2eb625d8e75f9f9

Choosing a good salt

- Use a long salt value
- Anything can be used as a salt, but it is best to choose something unpredicatable
- Use a cryptographically secure pseudo-random number generator (CSPRNG)

Password Cracking: Do the Math

- Assumptions:
  - Pwds are 8 chars, 128 choices per character
    - Then $128^8 = 2^{56}$ possible passwords
  - There is a **password file** with 2^{10} pwds
  - Attacker has **dictionary** of 2^{20} common pwds
- **Probability** of 1/4 that a pwd is in dictionary
- Work is measured by number of hashes

Password Cracking: Case I

- Attack **one** password (e.g. the admin account) without dictionary
  - Must try $2^{56}/2 = 2^{55}$ on average
  - Like exhaustive key search
- Does **salt** help in this case?

Password Cracking: Case II

- Attack **one** password *with* dictionary
- With **salt**
  - Expected work: $1/4\ (2^{19}) + 3/4\ (2^{55}) = 2^{54.6}$
  - In practice, try all passwords in dictionary…
  - …then work is at most 2^{20} and probability of success is 1/4
- What if **no salt** is used?
  - One-time work to compute dictionary: 2^{20}
  - Expected work still same order as above
  - But with precomputed dictionary hashes, the "in practice" attack is free…

Password Cracking: Case III

- **Any** of 1024 passwords in file, *without* dictionary
  - Assume all 2^{10} passwords are distinct
  - Need 2^{55} **comparisons** before expect to find password
- If **no salt** is used
  - Each computed hash yields 2^{10} comparisons
  - So expected work (hashes) is $2^{55}/2^{10} = 2^{45}$
- If **salt** is used
  - Expected work is 2^{55}
  - Each comparison requires a hash computation

Password Cracking: Case IV

- Any of 1024 passwords in file, **with** dictionary
  - Prob. one or more password in dict.: $1 - (3/4)^{1024} = 1$
  - So, we ignore case where no password is in dictionary
- If **salt** is used, expected work less than 2^{22}
  - See book, or slide notes for details
  - Approximate work: size of dict. / probability
- What if **no salt** is used?
  - If dictionary hashes not precomputed, work is about $2^{19}/2^{10} = 2^9$

Problem 1.5: Attacking passwords hashed with salt values

The login at `login3.php` uses **salt values** with the hash function. The table is `user3`.

A list of common passwords is available at http://cs31.cs.sjsu.edu/passwords.txt.

Write a program to crack as many passwords as you can.

List the usernames & passwords you can determine and submit your code.

Pepper Value

- Salt values slow down an attacker, but an attacker can still get many passwords.
- A *pepper value* is a secret value added to the hash input.
- Adding a pepper value requires additional work from the attacker (until it is broken).

Alternatives to using a pepper

- Using a pepper is similar to a *keyed hash,* also known as a *hashed mac* (HMAC).
- According to RFC 2104, the proper form for an HMAC is HMAC(M,K) = h(K ⊕ opad, h(K ⊕ ipad, M)), where
  - h is the hash function
  - K is the key
  - M is the message
  - ipad = 0x36 repeated B times, where B is the block length
  - opad = 0x5C repeated B times
- Alternately, encrypt the hash output with a secret key.

Problem 1.6: Attacking passwords hashed with salt & pepper

The page `login4.php` (using table user4) uses salts and a pepper value between '0' and '9':

```
hash(salt + pepper + password)
```

Update your program to crack as many passwords as you can. (You might not get them all).

a. What is the pepper value?
b. How much longer did it take your code to run?
c. How much slower would your code have run if the pepper were between 0 and 999,999?

Cryptographic hashes & efficiency

Generally, cryptographic hashes should be as efficient as possible.

Why might an efficient hashing algorithm be a bad choice for storing passwords?

Modern Password Hashing

- Newer algorithms use *key stretching* to increase the work required per hash.
  - The initial key is fed into an algorithm that outputs an **enhanced key**.
- Examples:
  - Bcrypt
  - PBKDF2 (Password-Based Key Derivation Function 2)

One Key Stretching Algorithm

```
String hashStretchKey(
      String password, String salt,
      String pepper, int workFactor)
{
  String hash = "";
  for (int i=0; i<workFactor; i++) {
    hash = hashFun(hash + salt
                + pepper + password);
  }
  return key;
}
```

Problem 1.7: Bringing it all together

There can be many ways to proceed with an attack. You must devise the best approach.

The page secret-identities.php contains information about the secret identities of several superheroes.

Using what you have learned, determine the identities of:

- Darkwing Duck
- Stupendous Man

Preview: Next session

- JavaScript-based attacks
- Cross-site scripting (XSS)
- Cross-site request forgery (CSRF)
- Browser defenses