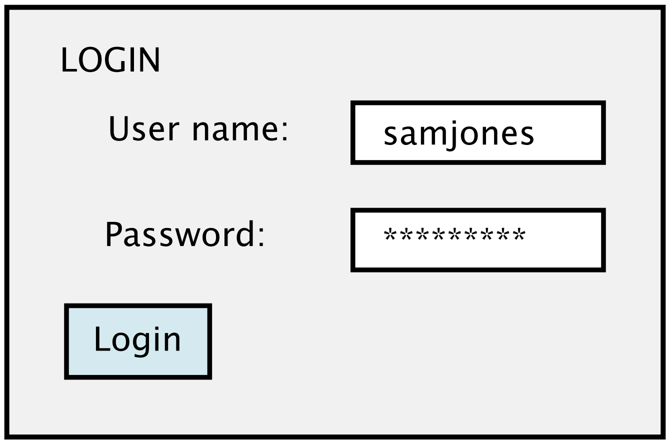
Working With Passwords

- What NOT To Do
- Storing Password Hashes
- Forgotten Passwords

Working With Passwords

- Many Web applications need to authenticate users before providing access
- The most common way to authenticate is user names and secret passwords
- JEE Web applications should follow good practices for working with passwords



Instead of a user name, some Web applications use an email address.

Note how the password is obscured to prevent bystanders from seeing the secret.

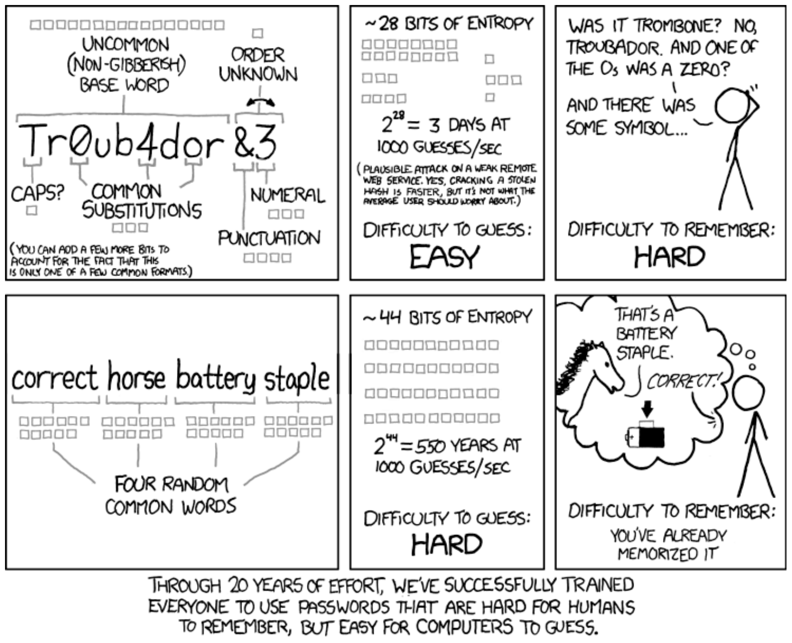
To authenticate, most applications compare the entered info against credentials stored in a database or a registry such as LDAP or Active Directory.

Password Strength

- It's relatively easy for attackers to guess common passwords, so applications may enforce password-strength guidelines
- Common rules include:
    - Minimum password length
    - Required usage of non-alphabetic characters
- Note that too stringent rules may result in hard-to-remember passwords (that users may write down on a sticky note)

Password Strength, cont'd

- http://xkcd.com/936:


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

The xkcd Web comic has a interesting take on password-strength guidelines.

What NOT to Do

- Never store passwords unencrypted in a database or registry
- In fact, never store passwords at all! (Use a hash)
- Never run login pages over non-HTTPS connections (except for unimportant applications)

userID	username	password
3456	samjones	abc123
4384	suesmith	pa33w0rd

Storing unencrypted (or lightly encrypted) passwords in a database is a very bad idea:

The passwords are exposed to database administrators and are also stored in any backups.

If an attacker gains access to the database, perhaps via an SQL injection, all of the passwords are easily accessible.

Using Hashed Passwords

- A **hashed password** is a **digest** calculated from a user's password
- Java provides digesting algorithms that generate hashes, and there are third-party libraries such as BCrypt available
- Instead of storing passwords in a database or registry, store the hash instead
- When a user attempts to login, calculate a hash from input and compare to the stored hash
- Side effect: Applications cannot send forgotten passwords to users!

The reason that the application can provide a forgotten password is that hashing is irreversible (at least it's supposed to be).

There are various hashing algorithms available, but note that fast performance is actually undesirable. On average, people log in infrequently (so slow performance is OK) and slow algorithms make brute-force attacks less feasible.

Perhaps the best algorithm is BCrypt, which is intentionally slow for this very purpose. There's a Java implementation available at:

```
http://www.mindrot.org/projects/jBCrypt/
```

Note that BCrypt actually stores salt in the resulting hash itself, so applications don't need to store the salt separately (salt is coverered on the next page).

Using Salt in the Hash

- Using hashed passwords is a good start, but hashed passwords are still vulnerable to a **rainbow table** attack
- The solution is to add randomness to the hashed passwords referred to as **salt**
- Salt is a fixed-length random value that the application generates for each hashed password

userID	username	passwordhash	salt
3456	samjones	46e4a906271a7cbb09fbc602da7533544d17b3c9	58b89701-be36-4560-bcf1-a279e4841bb1
4384	suesmith	b93569b63ecb550f3f4f28909f913e03c5b12e82	ac7e66cc-55c3-4264-9f70-c93bec010c56

A "rainbow table" is a list of precomputed hashes of common passwords. The application typically stores the salt as cleartext in the database in the same row as the hashed password.

Using salt with a decent length makes rainbow tables impractical, since the salt increases the size of the rainbow table. Salt has the additional benefit that if two users happen to have the same password, the hash will be different – salt values should be random numbers.

When generating salt, it's important to use a good random number generator – the java.util.Random class is not good enough. Instead, Java applications can use java.security.SecureRandom, which does a better job.

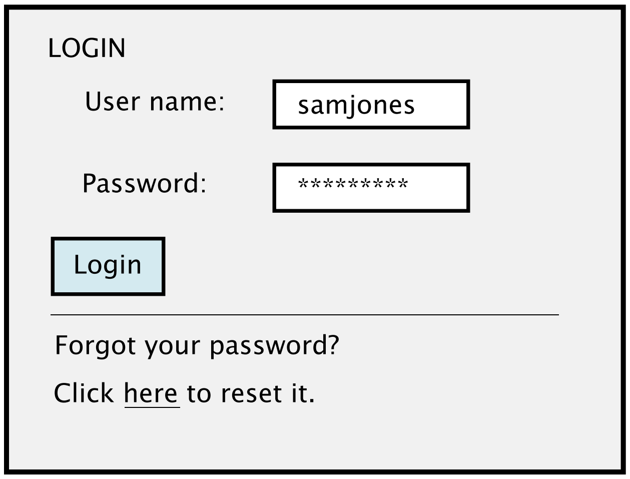
Note that even if you salt the passwords, applications are still vulnerable to dictionary attacks, but you can mitigate those by enforcing good password strength and by adding delays to login attempts and limiting the number of failed logins.

Also note that it's OK that we store the salt in the database alongside the salted password hash. The salt need not be secret because its only purpose is to make sure that if two users have the same password, the hash of their passwords will be different. Once the password has been hashed with the salt, there's no way that the salt can be " removed" from the hash, even if it is known by the password cracker.

Here's a couple of good articles on salting and hashing:

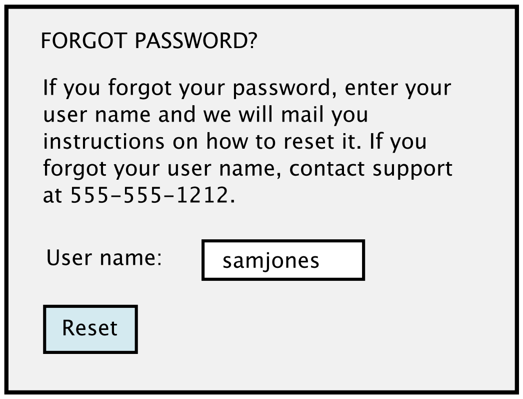
Forgotten Password Strategies

- If the application uses hashed passwords, it cannot send passwords to users who forget them (the application doesn't have them!)
- Instead, applications must prompt users who forget their passwords to reset their password (create a new one)



Forgotten Password Strategies, cont'd

- A popular approach is to require users at registration time to provide an email address
- If they forget their password, the application can send an email to the registered email, including an HTTPS link to a page that lets the user create a new password
- Another strategy is to have the application generate a temporary password and email it, but this is less secure


FORGOT PASSWORD?

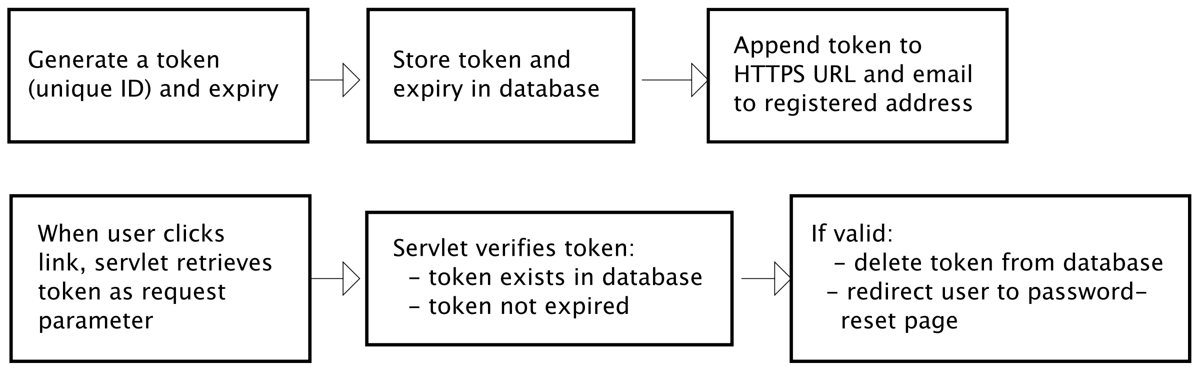
If you forgot your password, enter your user name and we will mail you instructions on how to reset it. If you forgot your user name, contact support at 555-555-1212.

User name: samjones

For best security, any temporary password or link to create a new password should have a relatively short timeout associated with it.

To reset the password, the user supplies a user name and the application emails the HTTPS link to the email address "on file" for that user name.

Forgotten Password Steps


Generate a token (unique ID) and expiry
Store token and expiry in database
Append token to HTTPS URL and email to registered address
When user clicks link, servlet retrieves token as request parameter
Servlet verifies token:
– token exists in database
– token not expired
If valid:
– delete token from database
– redirect user to password-reset page

A good discussion of password resetting is at:

`http://stackoverflow.com/questions/3164978/php-help-with-password-reset-and-token-expir`

OWASP Java Project and Passwords

- OWASP provides best practices, guidelines and sample code that architects and developers can use when working with passwords

See https://www.owasp.org/index.php/Hashing_Java for details.

Chapter Summary

In this chapter, you learned about:

- Good practices for storing passwords
- Strategies for handling forgotten passwords