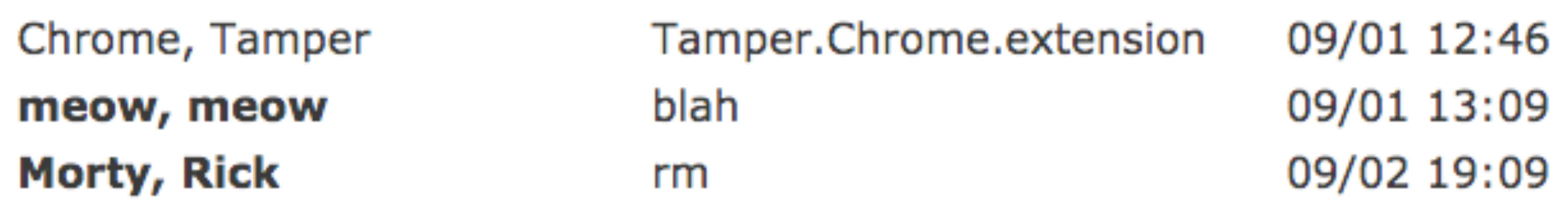
Attacking Authentication

Professor Larry Heimann
Web Application Security
Information Systems

Challenge from last class

- Just like fishing, it can be frustrating at times…
  - most needed multiple attempts, which is fine — casting
  - because of Tamper Data bugs, some just used a hidden field
  - most interesting attempts:



Chrome, Tamper	Tamper.Chrome.extension	09/01 12:46
meow, meow	blah	09/01 13:09
Morty, Rick	rm	09/02 19:09

- Importance of checking every step of the process
- Simple ways to defend against this attack

Password Popularity – Top 20

Rank	Password	Number of Users with Password (absolute)
1	123456	290731
2	12345	79078
3	123456789	76790
4	Password	61958
5	iloveyou	51622
6	princess	35231
7	rockyou	22588
8	1234567	21726
9	12345678	20553
10	abc123	17542
11	Nicole	17168
12	Daniel	16409
13	babygirl	16094
14	monkey	15294
15	Jessica	15162
16	Lovely	14950
17	michael	14898
18	Ashley	14329
19	654321	13984
20	Qwerty	13856

Data from an analysis of 320 million passwords recovered from rockyou.com in 2009

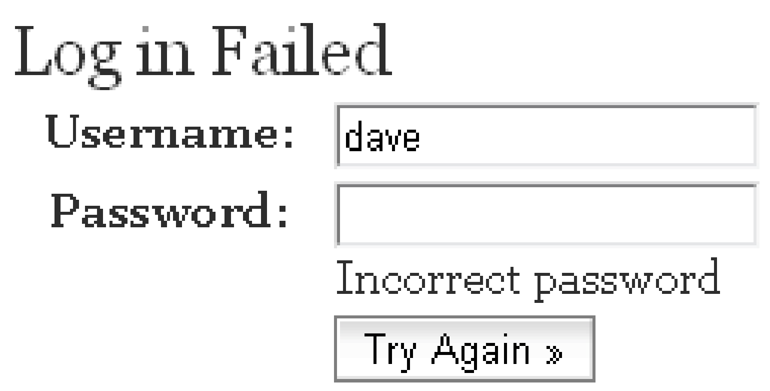
Authentication Technologies

- Various technologies are used, often in combination:
    - HTML forms-based
    - Multi-factor (passwords & tokens, etc)
    - Client SSL certificates & smartcards
    - HTTP basic / digest authentication
    - Windows-integrated authentication
    - Authentication services (e.g. MS Passport)
- The majority of Internet applications use simple forms-based authentication.
- **Most authentication flaws can arise with any technology**.

The obvious stuff

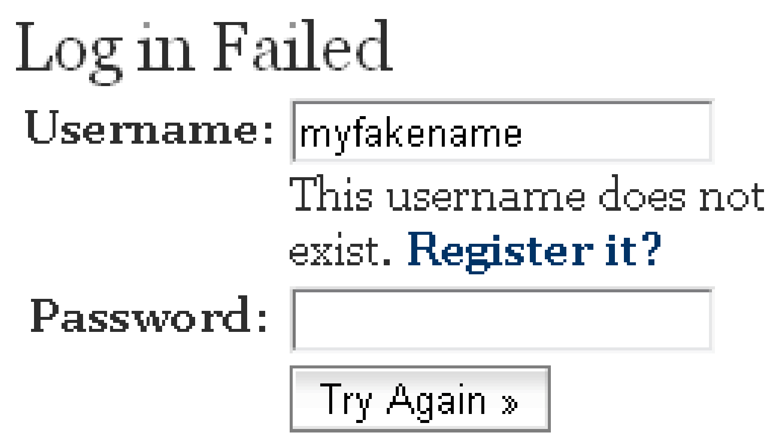
- Weak passwords

- Ability to enumerate usernames



Log in Failed

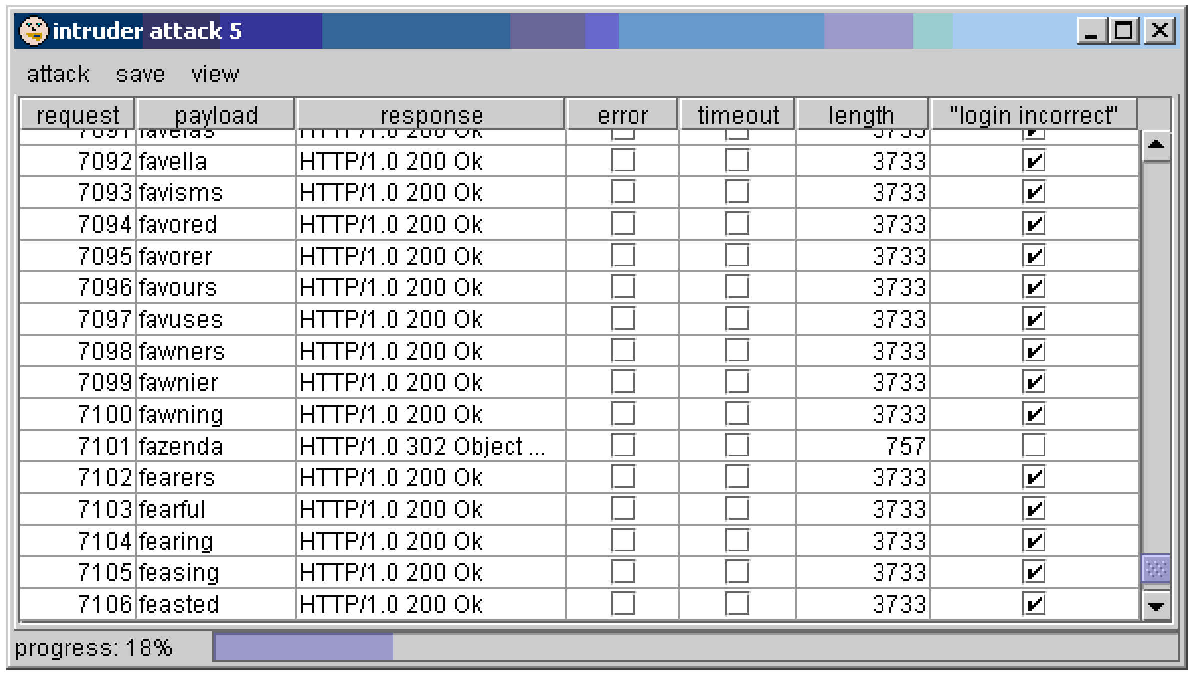
Username:	dave
Password:	
	Incorrect password
	Try Again »



Log in Failed

Username:	myfakename
	This username does not exist. **Register it?**
Password:	
	Try Again »

- Ability to brute force the login



request	payload	response	error	timeout	length	"login incorrect"
7092	favella	HTTP/1.0 200 Ok	☐	☐	3733	✓
7093	favisms	HTTP/1.0 200 Ok	☐	☐	3733	✓
7094	favored	HTTP/1.0 200 Ok	☐	☐	3733	✓
7095	favorer	HTTP/1.0 200 Ok	☐	☐	3733	✓
7096	favours	HTTP/1.0 200 Ok	☐	☐	3733	✓
7097	favuses	HTTP/1.0 200 Ok	☐	☐	3733	✓
7098	fawners	HTTP/1.0 200 Ok	☐	☐	3733	✓
7099	fawnier	HTTP/1.0 200 Ok	☐	☐	3733	✓
7100	fawning	HTTP/1.0 200 Ok	☐	☐	3733	✓
7101	fazenda	HTTP/1.0 302 Object ...	☐	☐	757	☐
7102	fearers	HTTP/1.0 200 Ok	☐	☐	3733	✓
7103	fearful	HTTP/1.0 200 Ok	☐	☐	3733	✓
7104	fearing	HTTP/1.0 200 Ok	☐	☐	3733	✓
7105	feasing	HTTP/1.0 200 Ok	☐	☐	3733	✓
7106	feasted	HTTP/1.0 200 Ok	☐	☐	3733	✓

progress: 18%

More subtle variations

- The application may require strong passwords but not validate them fully (e.g. case-insensitive check).
- Login failure messages may be the same on-screen, but contain subtle differences in the HTML source.
- Timing of different login failures could be different (timing attacks will be an issue later with injection attacks as well).
- Password guessing may be blocked in the browser but still possible using a scripted attack, due to reliance on client-side controls, logic flaws, etc.

Exploiting common login defects

- Experiment to determine what password quality rules are enforced.
- Check whether credentials are being validated in full.
- Review every detail of failed login messages to find username enumeration bugs. Check the page source, HTTP headers, and response times.
- Experiment to identify any account lockout defenses.
- Identify every possible target for mounting a brute force attack.
- Perform password guessing attacks breadth-first not depth-first – that is, work through a list of common passwords trying each password with every username in turn. Start with the most obvious and common passwords.

Other authentication functions

- Most applications contain other functionality to support the primary login, which can often be used to attack the overall mechanism:
    - User registration
    - Password change
    - Account recovery
    - “Remember me”

Other authentication functions

- User registration functions very often contain username enumeration flaws, because the application indicates whether a chosen username is already registered.

- Password change functions may allow username enumeration and brute force password guessing even if these are blocked in the main login function.

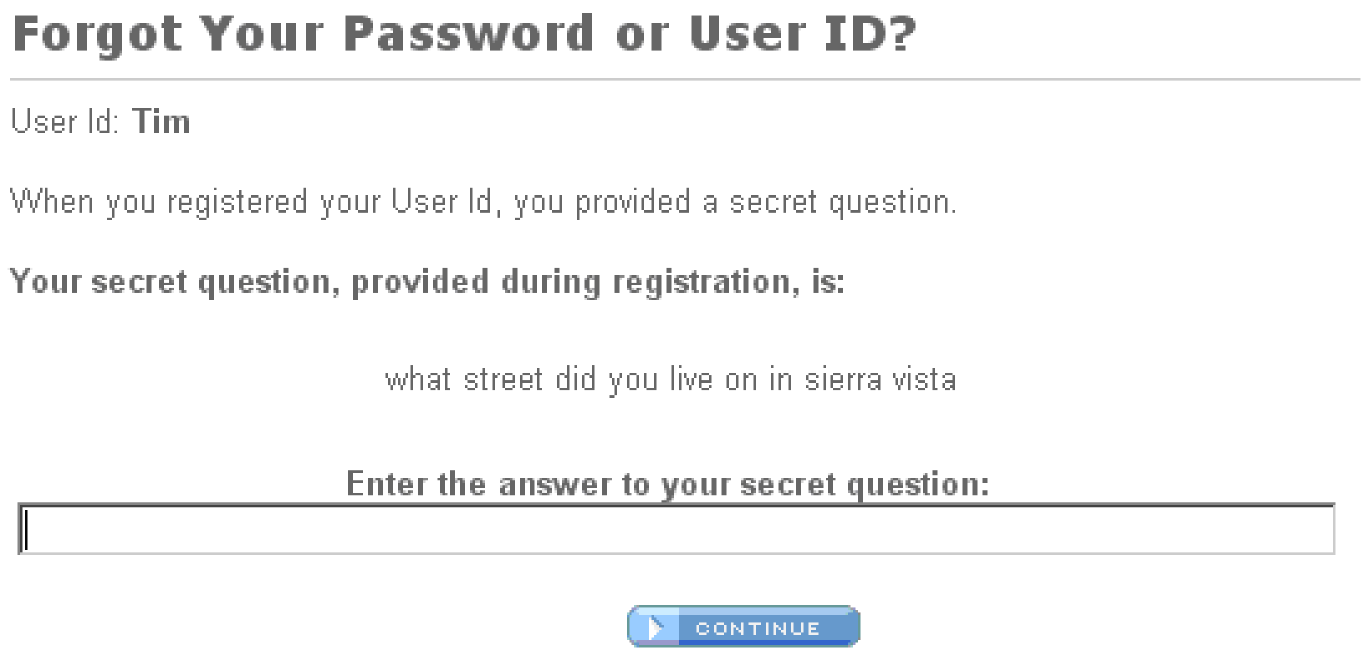
- “Remember me” functions often contain logic flaws or access control defects:

```
Set-Cookie: RememberUser=edgruberman

Set-Cookie: autologin=true
```

Other authentication functions

- Account recovery functions often involve a secondary challenge which is presents a considerably lower bar than the main login function (e.g. “Do I own a pet?”).



Forgot Your Password or User ID?

User Id: **Tim**

When you registered your User Id, you provided a secret question.

Your secret question, provided during registration, is:

what street did you live on in sierra vista

Enter the answer to your secret question:

- Users assume that only they will see their challenge.
- An attacker can harvest a large number of challenges and choose the easy ones.
- Username enumeration and brute force password guessing may be possible even if these are blocked in the main login function

Other authentication functions

- Instead of a secondary challenge, account recovery often uses a password “hint”.
- An attacker can harvest large numbers of hints and then start guessing.
- Following successful completion of the account recovery challenge, the application often lets you:
    - Jump straight into an authentication session.
    - Recover the existing password.
    - Set a new password directly.
    - Receive a recovery URL to an arbitrary email address you specify.

Class Demonstration

DOGBERT'S PASSWORD RECOVERY SERVICE FOR MORONS
I DON'T REMEMBER MY PASSWORD.
IS IT "123"?
THAT'S JUST SPOOKY.
www.dilbert.com
scottadams@aol.com
© 2007 Scott Adams, Inc./Dist. by UFS, Inc.

Securing authentication

Use strong credentials

- Rules for minimum length, appearance of different character types, upper and lower case, avoidance of dictionary words, etc.
- Ensure any system-generated values are unpredictable. Handle credentials secretively.
- Use SSL for all authentication functions (both loading and submission of forms).
- Only transmit credentials using POST requests, and never pass them back to the client.
- Store credentials using salted one-way hashes.
- "Remember me" functions should only remember usernames.
- Implement a password change function that is also secure.

Securing authentication

Validate credentials properly

- Validate in full, case-sensitively.
- Defend aggressively against unexpected events during login processing (catch all exceptions and immediately invalidate the session).
- Implement proper access control over user impersonation functions.

Securing authentication

Prevent information leakage

- Remember every piece of functionality where credentials are validated.
- Use a single code component to handle all failed login attempts, and return a generic message.
- 2 ways self-registration functions can be designed to prevent username enumeration:
  - The application can generate its own usernames in an unpredictable way, avoiding the need to disclose that a selected username already exists.
  - The application can use email addresses as initial usernames. The first stage of registration involves entering an email address, and the application sends an email containing a one-time registration URL or an indication that the address is already registered.

Securing authentication

Prevent brute force attacks

- Suspend accounts after a small number of failed logins (e.g. three). Optionally, reinstate accounts after a short period (e.g. 30 minutes).
- To prevent information leakage, do not identify that any specific account has been suspended – after a failed login, simply state that accounts are suspended after a small number of failures.
- Do not disclose the metrics of the suspension policy.
- If an account is suspended, reject login attempts without checking the credentials, and records an additional failed login.
- Per-account measures will not prevent a stealthy breadth-first attack (for example, targeting every username with a small number of weak passwords).
- To defend against these attacks, controls like CAPTCHAs can be used

Securing authentication

Defend the password change function

- Allow access to authenticated users only.
- Do not allow users to specify a username (either on-screen or in a hidden request parameter).
- Require the existing password to be supplied.
- Defend against password guessing and information leakage.
- Notify the user via email that their password has been changed.

Securing authentication

Defend the account recovery function

- Do not use password “hints”
- To enable account recovery, send a one-time URL to the email address which the user provided during registration. Visiting the URL should allow the user simply to specify a new password.
- A secondary challenge may also be used before the one-time URL is sent:
- It should use the same question (or set of questions) for all users, rather than user-specified questions.
- Responses should contain reasonable entropy (e.g. name of first school is preferable to favorite color).
- Defend against username enumeration and brute force attacks.

Next Class:

Lab 1 on Authentication, Simple Attacks

You will need the following installed on a laptop before next class:

1. Git (1.8.x or higher)
2. Rails (3.2.13)
3. Gems -- rake (10.1.0), faker (1.2.0), thin (1.5.1), will_paginate (3.0.4), and sqlite3 (1.3.8)
4. Burp Suite (*free version from* http://portswigger.net/burp/download.html *is fine*)
5. Firefox or Chrome with appropriate extensions, tools for carrying out simple attacks