

Objectives

- Review: Usability
- Security
 - Design principles
 - SQL Injection
- Project

Review: Usability

- What does *usability* mean with respect to user interfaces?
 - What metrics should we consider when determining if our interface is usable?
- How are the users of web interfaces different from users of other interfaces?
 - How does that inform how we design user interfaces?
 - What principles should we follow?
- What is *accessibility*?
 - How can we evaluate how accessible our web site is?

Fun Facts

- Research from Microsoft Canada found that between 2000 and 2015, the average human attention span dropped from 12 seconds to 8 seconds.

Security

“A few lines of code can wreak more havoc than a bomb.”

--Tom Ridge

Former Secretary of the U.S.

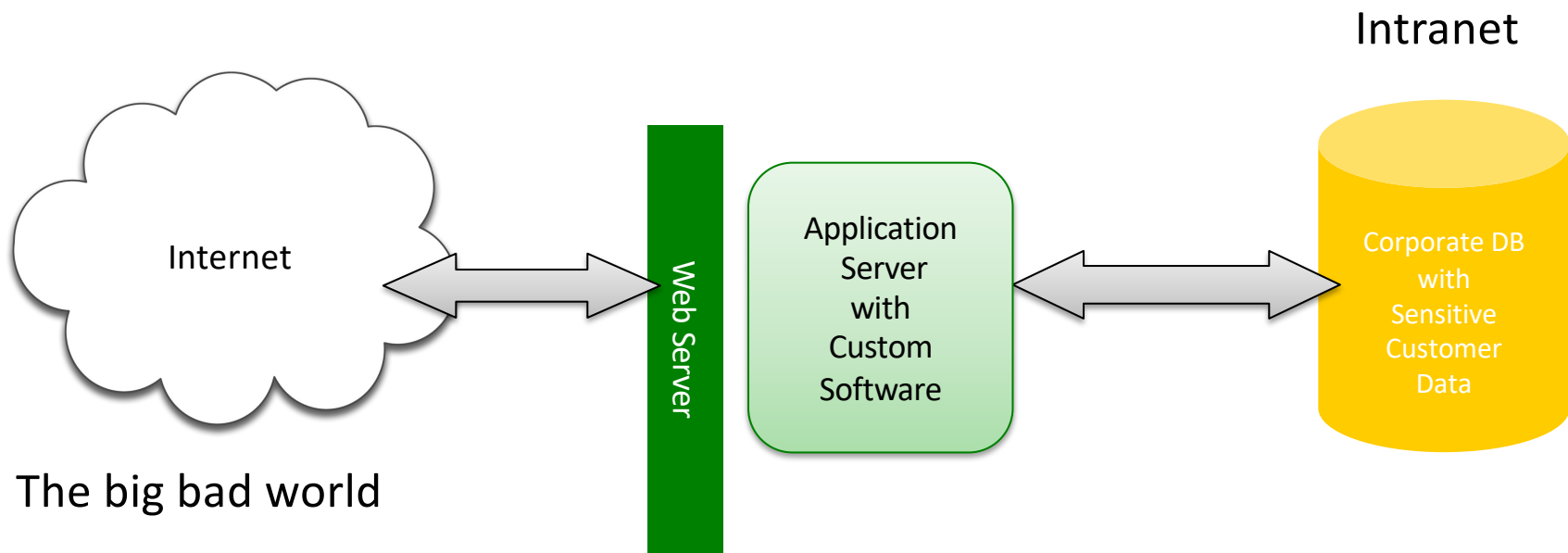
Department of Homeland Security

Discussion

- Security has been an underlying concern in many topics we discussed

Why is the Web a target?

Web Application Architecture



"75% of cyber attacks and Internet security violations are generated through Internet Applications" - Gartner Group

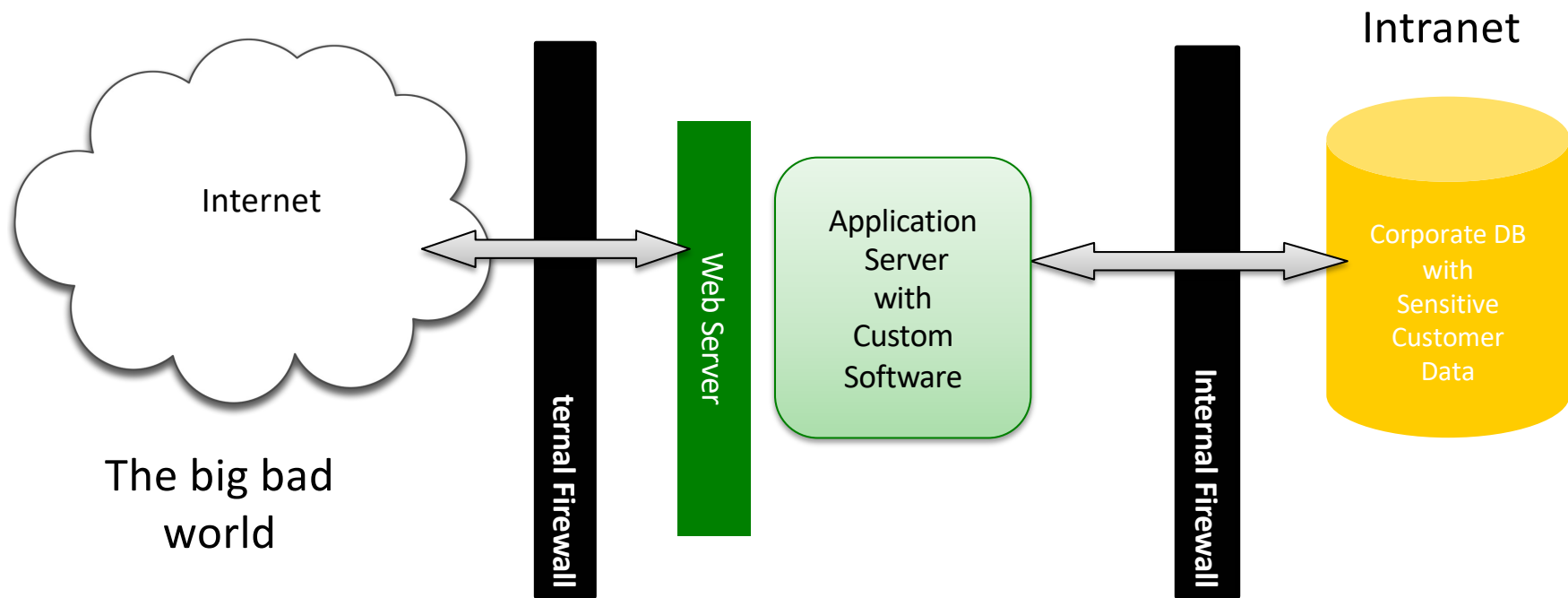
Why is the Web a Target?

- New environments: lots of vulnerabilities
 - Continually new technologies to exploit
- Many different targets
 - Web browsers, Web servers, databases, personal computers, network traffic, humans
- Efficient, lucrative
 - Can get a lot done quickly
 - Personal info, credit card info, databases
 - Businesses: could lose millions of dollars/hour

Why Do We Need Security?

- Protect personal identifying information
- Preserve privacy
- Limit legal liability
- Prevent theft (\$, intellectual property, ...)
- Prevent malicious damage

Web Application Architecture



"75% of cyber attacks and Internet security violations are generated through Internet Applications" - Gartner Group

Security Concepts

Authentication → Who are you?

Authorization → What are you allowed to do?

Confidentiality → Only allow authorized users

Integrity → Information is correct and current

Availability → Information is available when needed

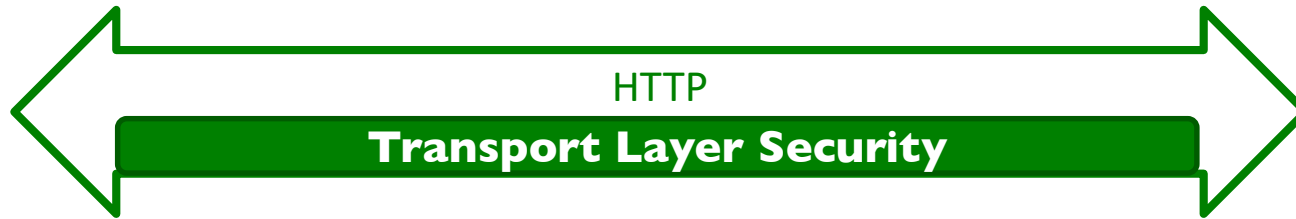
Programming Authentication

- Use the password input field
 - `<input type="password" name="password">`
 - Validate the username and password *on the server*
 - Never on the client!
- Store whether the user has been authenticated
 - For example, in the session object
 - **Never** pass this information back to the client!
- Don't forget to lock the *back doors*
 - Check authentication in *every* software component
 - If the user is not authenticated, go to the login screen
 - common vulnerability in web applications

Don't Reinvent the Wheel

- Building a secure, high-performance web server is a very challenging task
 - Apache: www.apache.org
- Use trusted components
 - Keep up-to-date with security patches

HTTPS: Hypertext Transfer Protocol Secure



- HTTP over Transport Layer Security (TLS)
 - Formerly over Secure Socket Layer (SSL)
- De-facto standard used for secure web-based transactions
 - Otherwise, bad guys can get access to your session
- Encrypt every HTTP message to and from the web server using standard PKI (public key infrastructure) technology
- Default URL <https://some.domain.com> with default port number 443

HTTPS: Hypertext Transfer Protocol Secure



- Creates a secure connection over insecure network
- TLS works well because only one side (the server) needs to be authenticated for the protocol to work.
- Protects the data, URL, query parameters in the HTTP request

Security Design Principles

- Least Privilege
- Defense in Depth
- Secure Weakest Link
- Fail-safe Stance
- Secure By Default
- Simplicity
- Usability

Security Design Principle:

Principle of Least Privilege

- Give just enough permissions to get the job done
 - Minimize damage if compromised
- Common world example: Valet Keys
 - Can start the car but can't access glove compartment, trunk
- A web server should only be given access to set of HTML files that web server is to serve
 - If web server is compromised, attacker can only read HTML files

Security Design Principle:

Defense in Depth

- Also called redundancy / diversity
- Common world example: Banks
 - What security does a bank provide so that bad guys won't steal money?
- Passwords:
 - Require users to choose strong passwords
 - Monitor web server logs for failed login attempts
 - If 3 incorrect password attempts, lock account for some period of time

Discussion of Passwords

- What makes a good password?
 - To the software?
 - To a user?
- How do passwords and their restrictions affect software?
 - How do they affect those who try to break them?
- How often should users change their passwords?
 - What are the tradeoffs in frequency?

Security vs. Usability

- Hard to design passwords that are **both secure** and **easy to remember**
- If users have to change their password more than every 6 months, security decreases
 - Come up with schemes to make passwords easier to remember

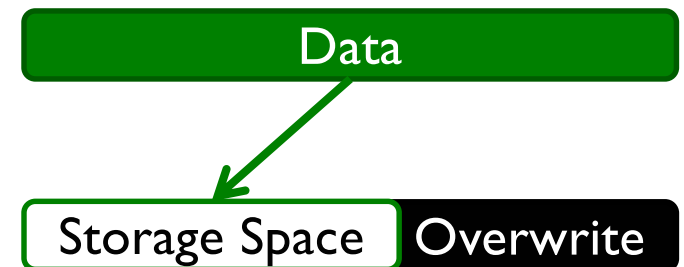
Security Design Principle:

Secure the Weakest Link

- Common Weak Links:
 - Weak Passwords - Crack
 - People - Social Engineering Attacks
 - “I am owed an inheritance; you can get a cut if you just give me your bank account number...”
 - Buffer Overflows
 - Slapper Worm: exploited OpenSSL, Apache to create peer-to-peer networks → DDOS

Buffer Overflows

- An error caused when a program tries to store too much data in a temporary storage area
 - Exploited by hackers to execute malicious code
- Not an issue in Java, Python



Security Design Principle:

Fail-Safe Stance

- Even if 1 or more components of system fail, there is some security
- Common world example: Elevators
 - If elevator power fails, grip the cable
- System failure should be expected (and planned for)
 - If firewall fails, let **no** traffic in
 - Deny access by default

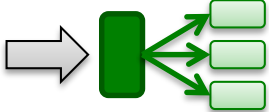
Security Design Principle:

Secure By Default

- “Hardening” a system: All unnecessary services off by default
 - Only enable the 20% of the product’s features that are used by 80% of the user population
 - Ex: don’t enable insecure networking apps by default
 - If choose to use those apps, should know what you’re doing
- More features enabled → more potential exploits → less security!

Security Design Principle:

Simplicity

- Complex software is more likely to have security holes
 - More code that wasn't tested thoroughly
 - More opportunities for buffer overflows, etc. 
- Use **choke points** to keep security checks localized
 - Centralized piece of code through which control must pass
- Generally: good rule of thumb to keep software simple

Security Design Principle:

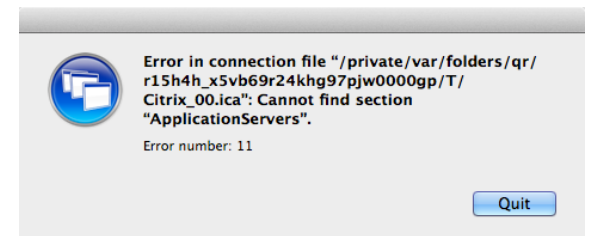
Usability

- Users typically do not read documentation
 - Enable security by default
- Users can be lazy
 - Assume: They ignore security dialogs
- **Secure-by-default features in software forces users and vendors to be secure**

CAREFUL EXCEPTION HANDLING

Careful Exception Handling

- Error messages and observable behavior can tip off an attacker to vulnerabilities
 - Don't provide more information to a user than is needed
- Exploit vulnerabilities using **Fault Injection**
 - Provide an application with input that it does not expect and observe its behavior
 - Bad guys use info to figure out application's vulnerabilities
 - Server may have fatal exception



Error Pages

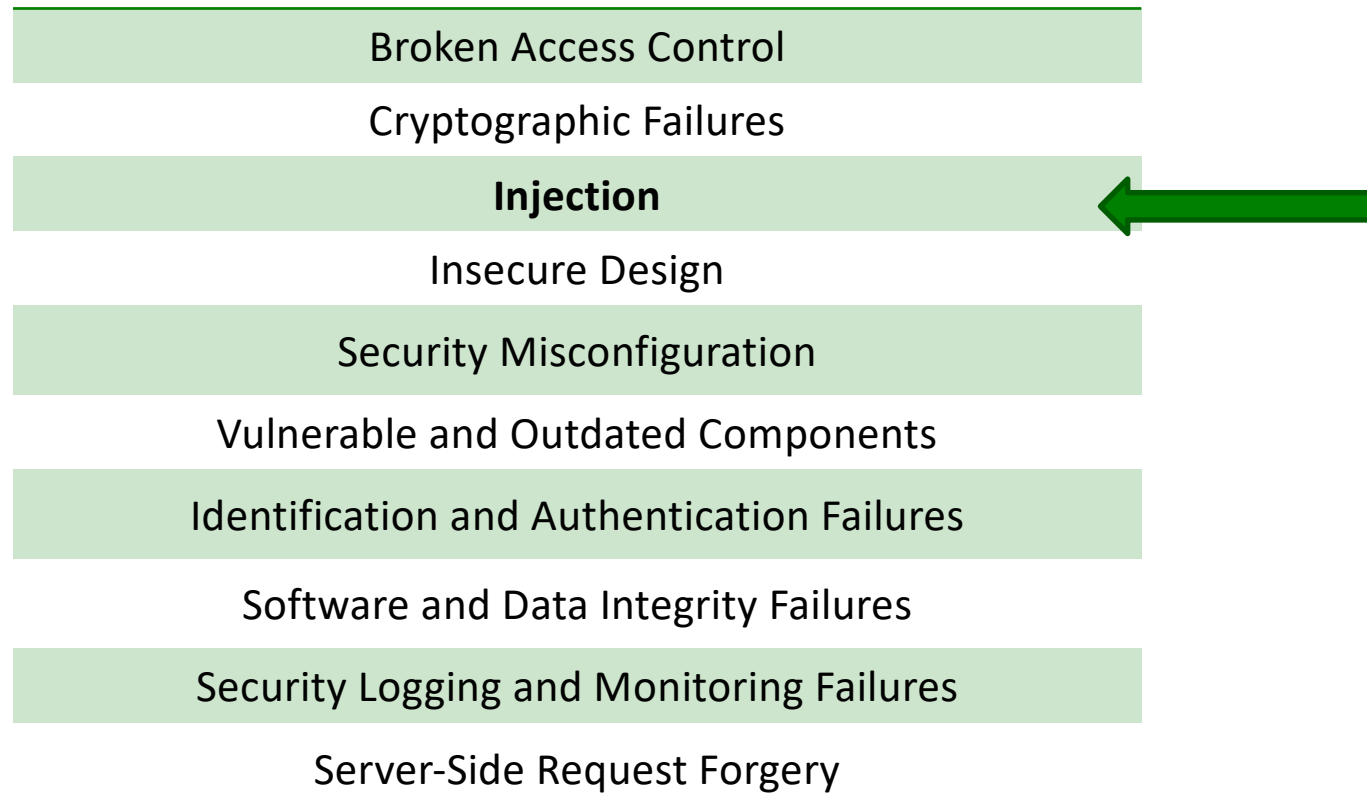
- What you don't want
 - Show the Server's default exception page
 - Show the stack trace
 - Looks like your application is broken to user
- Want error page to look like the rest of your site
 - While you're debugging, that's useful, but when you're ready for deployment, switch to a more generic page

VULNERABILITIES

OWASP Top 10 Most Critical Vulnerabilities

Open Web Application Security Project

<https://owasp.org/www-project-top-ten/>



SQL INJECTION ATTACK

SQL Injection

- Possible vulnerability when a program accepts *unvalidated input* from a user and uses that input to construct a dynamic SQL query to an SQL database
 - Client may construct crafted input that, when embedded in a string, is interpreted as an SQL query
 - Performs database operations not intended by application writers

SQL Injection

- **Root Cause:** Failure to properly scrub, reject, or escape domain-specific SQL characters from an input vector
- **Solution:**
 - Define accepted character-sets for input vectors, and enforce these accept lists rigorously.
 - Force input to conform to specific patterns when other special characters are needed
 - Example: dd-mm-yyyy
 - **Use SQL Prepared Statements**

SQL Injection

- Typical query to email forgotten password:

```
SELECT fieldlist  
FROM table  
WHERE field = '$EMAIL';
```

- Is the input sanitized? Try `sprenkles@wlu.edu'`

```
SELECT fieldlist  
FROM table  
WHERE field = 'sprenkles@wlu'';
```

Extra quote



➤ If not, the query will throw exception

SQL Injection

- How to exploit:

- User enters **anything' OR 'x'='x**

```
SELECT fieldlist  
FROM table  
WHERE field = 'anything' OR 'x'='x';
```

Always true



- Query expected to only return one entry

- But, with this input, it will return all entries in user table
- Probably only displays the first response

- Can start to guess columns in table and table's name

SQL Injection

- Suppress the last quote:

```
SELECT email, passwd, login_id, full_name  
FROM members  
WHERE email = 'x' AND members.email IS NULL; --';
```

- Consequence: Bad guy doesn't need to worry about matching quotes

SQL Comment

- Is database read-only?

```
SELECT email, passwd, login_id, full_name  
FROM members  
WHERE email = 'x'; DROP TABLE members; --';
```

Validating Input

- **Block list:** a list of input types that are expressly forbidden from being used as application input
- **Accept list:** a list of input types that are expressly allowed as application input
- Generally expressed as **regular expressions**
- Input validation ***must be server side***
 - Not (only) in JavaScript

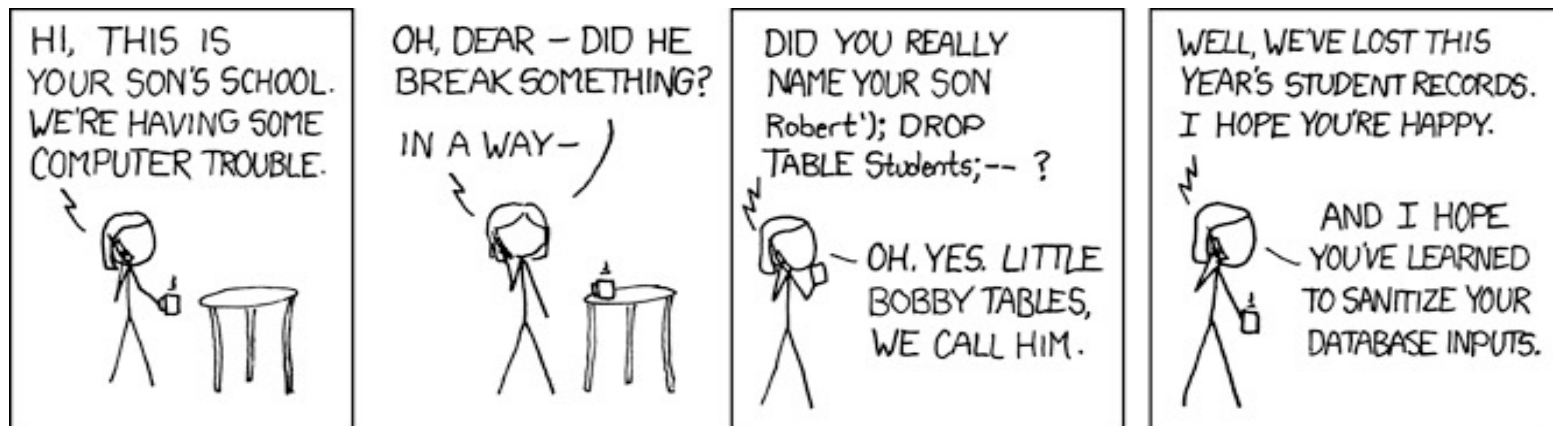
Preventing SQL Injection in Java

- Use PreparedStatement

How do PreparedStatement prevent SQL Injection?

Recap of Solutions to SQL Injection Attack

- Validate input
- Use PreparedStatements



<https://bobby-tables.com/>

DROP TABLE means to delete the whole table!

SECURITY EVALUATION

Security Evaluation Techniques

- Vulnerability scanners
 - Check to see if have vulnerability (but don't exploit)
- Penetration testing “pentest”
 - Simulated, authorized attack to see what can be penetrated
 - Exploit vulnerability – how far can they get?
- Bypass testing
 - Bypassing client-side input validation

Security Evaluation Techniques

- White box testing: static source code analysis
- Fuzz testing → fuzzing
 - Try random inputs – how does the application respond?
- Password cracking tools

Symantec reports that most security vulnerabilities are due to **software faults**

Security Features Do Not Imply Security

- Using one or more security algorithms/protocols will not solve *all* your problems!
 - Using encryption doesn't protect against weak passwords
 - Using HTTPS in a Web server doesn't protect against DoS attacks, access to various files, etc.

Security Features Do Not Imply Security

- Security features may be able to protect against *specific* threats
- If the software has bugs, is unreliable, or does not cover all possible corner cases:

The system may not be secure despite its security features

“Good Enough” Security

- Customers *expect* privacy and security
- BUT, time spent designing for security should be proportional to the number and types of threats that your software faces
- Design for security by incorporating “hooks” and other low-effort functionality from the beginning
 - Add more security as needed without having to resort to work-arounds

PROJECT

May 15, 2024

Sprenkle - CSCI335

46

Non-Technical-Skills Project Objectives

- Learn to work in a team
- Goal: well-balanced team
 - Each team member is doing $1/n^{\text{th}}$ of the work, where n is your number of team members
- Tasks of team members
 - design, implementation, testing, debugging, documentation, reviewing code, managing the group's efforts

Just-in-Time

- Learn what you need to learn when you need to learn it

Looking Ahead

- Starter tasks – due tomorrow at 11:59 p.m.
- Exam: Friday