Computer Security
CS 419

Web Security

Slides adopted from Prof. Ninghui Li at Prudue and Prof. V. N. Venkatakrishnan at UIC

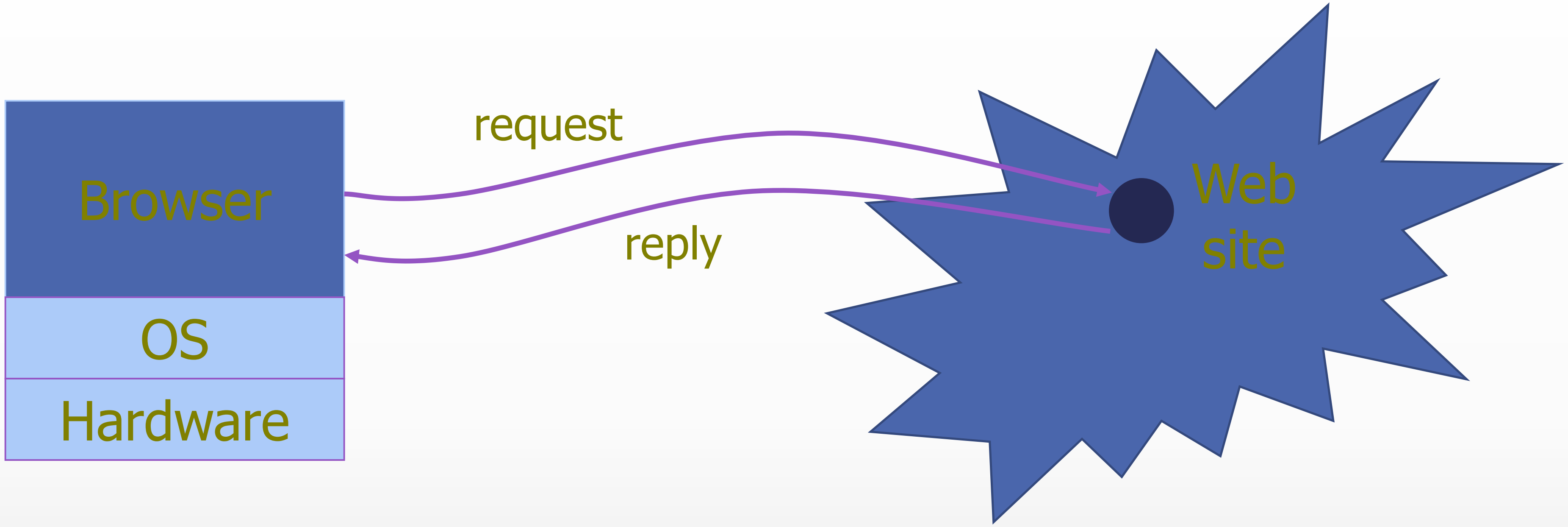
Web security

Background

- Many sensitive tasks are done through web
  - Online banking, online shopping
  - Database access
  - System administration

- Web applications and web users are targets of many attacks
  - Cross site scripting
  - SQL injection
  - Cross site request forgery
  - Information leakage
  - Session hijacking

Web Browser and Network



- Browser sends requests
- Web site sends response pages, which may include code
- Interaction susceptible to network attacks

Web Security Issues

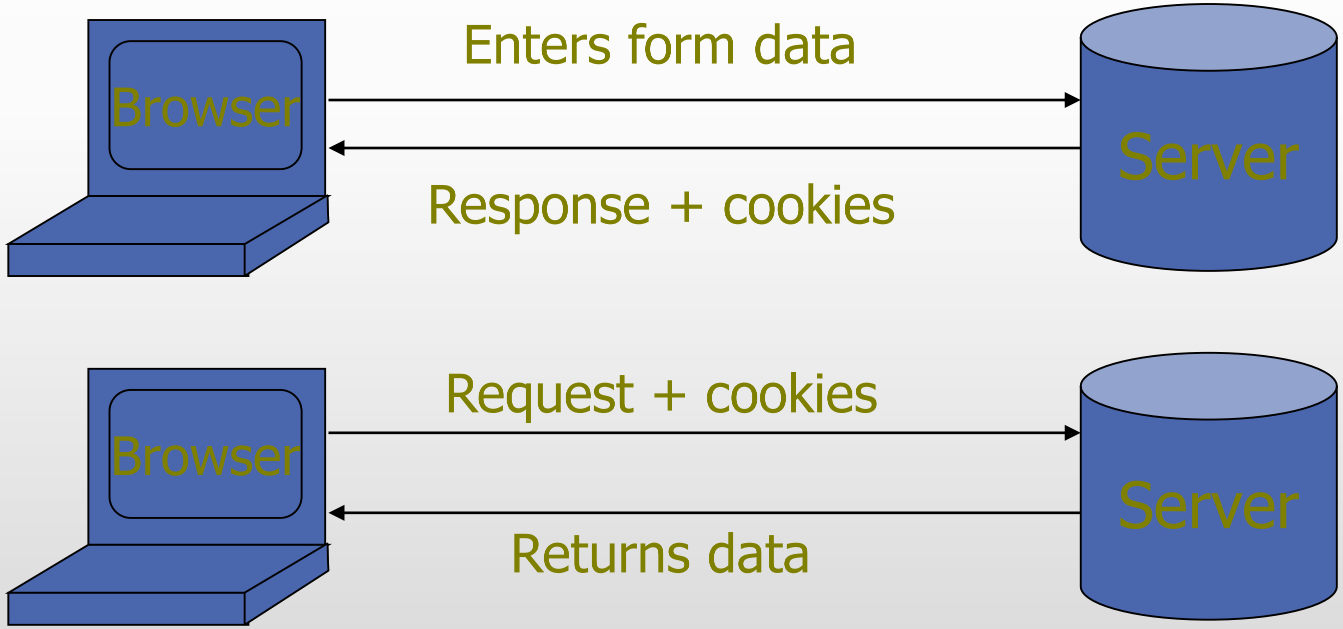
Secure communications between client & server	HTTPS (HTTP over Secure Socket Layer)
User authentication & session management	Cookies & other methods
Active contents from different websites	Protecting resources maintained by browsers
Web application security	
Web site authentication (e.g., anti-phishing)	
Privacy concerns	

HTTP: HyperText Transfer Protocol

- Browser sends HTTP requests to the server
  - Methods: GET, POST, …
  - GET: to retrieve a resource (html, image, script, css,…)
  - POST: to submit a form (login, register, …)
- Server replies with a HTTP response
- Stateless request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications

Use Cookies to Store State Info

- Cookies
  - A cookie is a name/value pair created by a website to store information on your computer



Browser

Enters form data

Response + cookies

Server

Browser

Request + cookies

Returns data

Server

Http is stateless protocol; cookies add state

Cookies Fields

- An example cookie from my browser
  - Name session-token
  - Content "s7yZiOvFm4YymG….”
  - Domain .amazon.com
  - Path /
  - Send For Any type of connection
  - Expires Monday, September 08, 2031 7:19:41 PM

Cookies

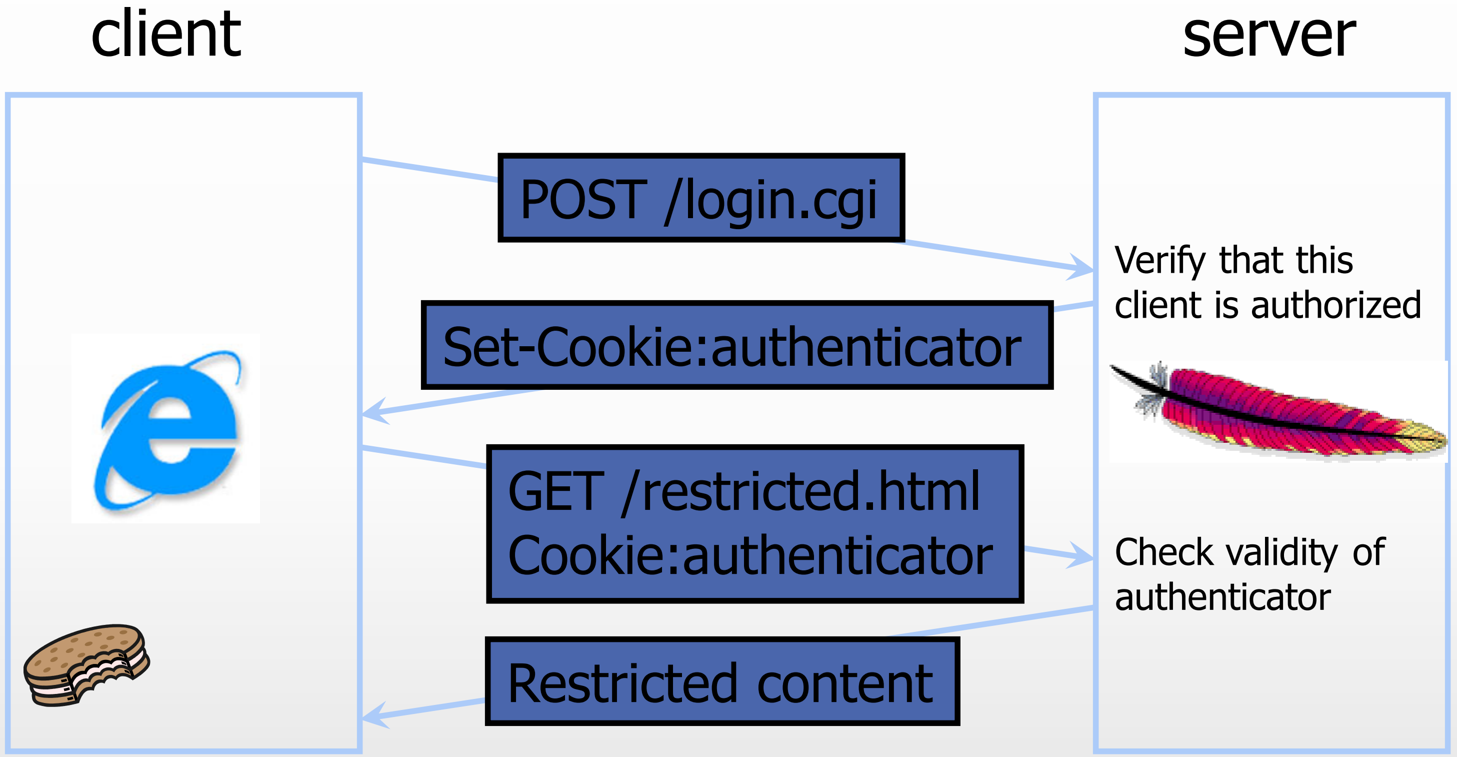
- Stored by the browser
- Used by the web applications
  - used for authenticating, tracking, and maintaining specific information about users
    - e.g., site preferences, contents of shopping carts
  - data may be sensitive
  - may be used to gather information about specific users
- Cookie ownership
  - Once a cookie is saved on your computer, only the website that created the cookie can read it

Web Authentication via Cookies

- HTTP is stateless
    - How does the server recognize a user who has signed in?

- Servers can use cookies to store state on client
    - After client successfully authenticates, server computes an authenticator and gives it to browser in a cookie
        - Client cannot forge authenticator on his own (session id)
    - With each request, browser presents the cookie
    - Server verifies the authenticator

A Typical Session with Cookies



client

server

POST /login.cgi

Verify that this client is authorized

Set-Cookie:authenticator

GET /restricted.html
Cookie:authenticator

Check validity of authenticator

Restricted content

Authenticators must be unforgeable and tamper-proof

(malicious clients shouldn’t be able to modify an existing authenticator)

How to design it?

Browser Cookie Management

- Cookie Same-origin ownership
  - Once a cookie is saved, ***only*** the Web site that created the cookie can read it.
- Variations
  - Temporary cookies
    - Stored until you quit your browser
  - Persistent cookies
    - Remain until deleted or expire
  - Third-party cookies
    - Originates on or sent to a web site other than the one that provided the current page

Third-party cookies

- Get a page from xxx.com
    - Contains <img src=http://doubleclick.com/advt.gif>
    - Image fetched from DoubleClick.com
        - DoubleClick knows IP address and page you were looking at
- DoubleClick sends back a suitable advertisement
    - Stores a cookie that identifies "you" at DoubleClick
- Next time you get page with a doubleclick.com image
    - Your DoubleClick cookie is sent back to DoubleClick
    - DoubleClick could maintain the set of sites you viewed
    - Send back targeted advertising (and a new cookie)
- Cooperating sites
    - Can pass information to DoubleClick in URL, …

Cookie issues

- Cookies maintain record of your browsing habits
  - Cookie stores information as set of name/value pairs
  - May include *any* information a web site knows about you
  - Sites track your activity from multiple visits to site
- Sites can share this information (e.g., DoubleClick)
- Browser attacks could invade your “privacy”

Cross Site Scripting

Client Side Scripting

Web pages (HTML) can embed dynamic contents (code) that can be executed on the browser

JavaScript

embedded in web pages and executed inside browser

Java applets

small pieces of Java bytecodes that execute in browsers

HTML and Scripting

```
<html>
    …
  <P>
<script>
    var num1, num2, sum
    num1 = prompt("Enter first number")
    num2 = prompt("Enter second number")
    sum = parseInt(num1) + parseInt(num2)
    alert("Sum = " + sum)
</script>
    …
</html>
```

Scripts are Powerful

- Client-side scripting is powerful and flexible, and can access the following resources
    - Local files on the client-side host
        - read / write local files
    - Webpage resources maintained by the browser
        - Cookies
        - Domain Object Model (DOM) objects
            - steal private information
            - control what users see
            - impersonate the user

Browser as an Operating System

- Web users visit multiple websites simultaneously
- A browser serves web pages (which may contain programs) from different web domains
    - i.e., a browser runs programs provided by mutually untrusted entities
    - Running code one does not know/trust is dangerous
    - A browser also maintains resources created/updated by web domains
- Browser must confine (sandbox) these scripts so that they cannot access arbitrary local resources
- Browser must have a security policy to manage/protect browser-maintained resources and to provide separation among mutually untrusted scripts

Same Origin Policy

- The basic security model enforced in the browser
- SoP isolates the scripts and resources downloaded from different origins
  - E.g., evil.org scripts cannot access bank.com resources
- Use origin as the security principal
- Origin = domain name + protocol + port
  - all three must be equal for origin to be considered the same

Same Original Policy: What it Controls

- Same-origin policy applies to the following accesses:
  - manipulating browser windows
  - URLs requested via the XmlHttpRequest
    - XmlHttpRequest is an API that can be used by web browser scripting languages to transfer XML and other text data to and from a web server using HTTP, by establishing an independent and asynchronous communication channel.
      - used by AJAX
  - manipulating frames (including inline frames)
  - manipulating documents (included using the object tag)
  - manipulating cookies

Problems with S-O Policy

- Poorly enforced on some browsers
  - Particularly older browsers
- Limitations if site hosts unrelated pages
  - Example: Web server often hosts sites for unrelated parties
    - http://www.example.com/account/
    - http://www.example.com/otheraccount/
  - Same-origin policy allows script on one page to access properties of document from another
- Can be bypassed in Cross-Site-Scripting attacks
- Usability: Sometimes prevents desirable cross-origin resource sharing

Cross Site Scripting (XSS)

- Recall the basics
  - scripts embedded in web pages run in browsers
  - scripts can access cookies
    - get private information
  - and manipulate DOM objects
    - controls what users see
  - scripts controlled by the same-origin policy
- Why would XSS occur
  - Web applications often take user inputs and use them as part of webpage (these inputs can have scripts)

How XSS Works on Online Blog

- Everyone can post comments, which will be displayed to everyone who view the post
- Attacker posts a malicious comment that includes scripts (which reads local authentication credentials and send of to the attacker)
- Anyone who view the post can have local authentication cookies stolen
- Web apps will check that posts do not include scripts, but the check sometimes fail.
- Bug in the web application. Attack happens in browser.

Effect of the Attack

- 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.

- Can infect other users: MySpace.com worm.

MySpace.com (Samy worm)

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no

    `<script>, <body>, onclick, <a href=javascript://>`
  - However, attacker find out that a way to include Javascript within CSS tags:

  `<div style="background:url('javascript:alert(1)')">`

  And can hide "`javascript`" as "`java\nscript`"
- With careful javascript hacking:
  - Samy's worm: infects anyone who visits an infected MySpace page … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.
- More info: http://namb.la/popular/tech.html

Avoiding XSS bugs (PHP)

- Main problem:
  - Input checking is difficult --- many ways to inject scripts into HTML.
- Preprocess input from user before echoing it
- PHP: **htmlspecialchars**(string)

  & → &amp; " → &quot; ' → &#039; < → &lt; > → &gt;

  - **htmlspecialchars**("<a href='test'>Test</a>", ENT_QUOTES);

    Outputs:

    &lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt;

Cross site request forgery

Cross site request forgery (abbrev. CSRF or XSRF)

- Also known as **one click attack** or **session riding**
- Effect: Transmits unauthorized commands from a user who has logged in to a website to the website.
- Recall that a browser attaches cookies set by domain X to a request sent to domain X; the request may be from another domain
  - Site Y redirects you to facebook; if you already logged in, the cookie is attached by the browser

CSRF Explained

- Example:
    - User logs in to bank.com. Forgets to sign off.
    - Session cookie remains in browser state
    - Then user visits another site containing:

```
<form  name=F  action=http://bank.com/BillPay.php>
<input  name=recipient   value=badguy> …
<script> document.F.submit(); </script>
```

-
    - Browser sends user auth cookie with request
        - Transaction will be fulfilled

- Problem:
    - The browser is a confused deputy; it is serving both the websites and the user and gets confused who initiated a request

GMail Incidence: Jan 2007

- Allows the attacker to steal a user's contact
- Google docs has a script that run a callback function, passing it your contact list as an object. The script presumably checks a cookie to ensure you are logged into a Google account before handing over the list.
- Unfortunately, it doesn't check what page is making the request. So, if you are logged in on window 1, window 2 (an evil site) can make the function call and get the contact list as an object. Since you are logged in somewhere, your cookie is valid and the request goes through.

Real World CSRF Vulnerabilities

- Gmail
- NY Times
- ING Direct (4th largest saving bank in US)
- YouTube
- Various DSL Routers
- Purdue WebMail
- PEFCU
- Purdue CS Portal
- …

Prevention

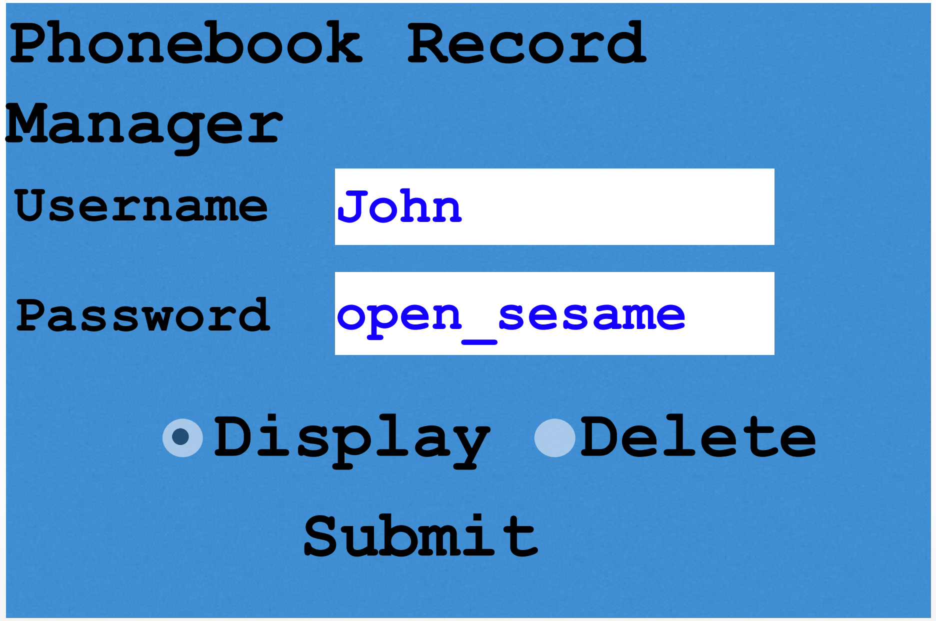
Server side:

- use cookie + hidden fields to authenticate a web form
  - hidden fields values need to be unpredictable and user-specific; thus someone forging the request need to guess the hidden field values
- requires the body of the POST request to contain cookies
  - Since browser does not add the cookies automatically, malicious script needs to add the cookies, but they do not have access because of Same Origin Policy

User side:

- logging off one site before using others
- selective sending of authentication tokens with requests (may cause some disruption in using websites)

SQL Injection



Phonebook Record Manager

Username John

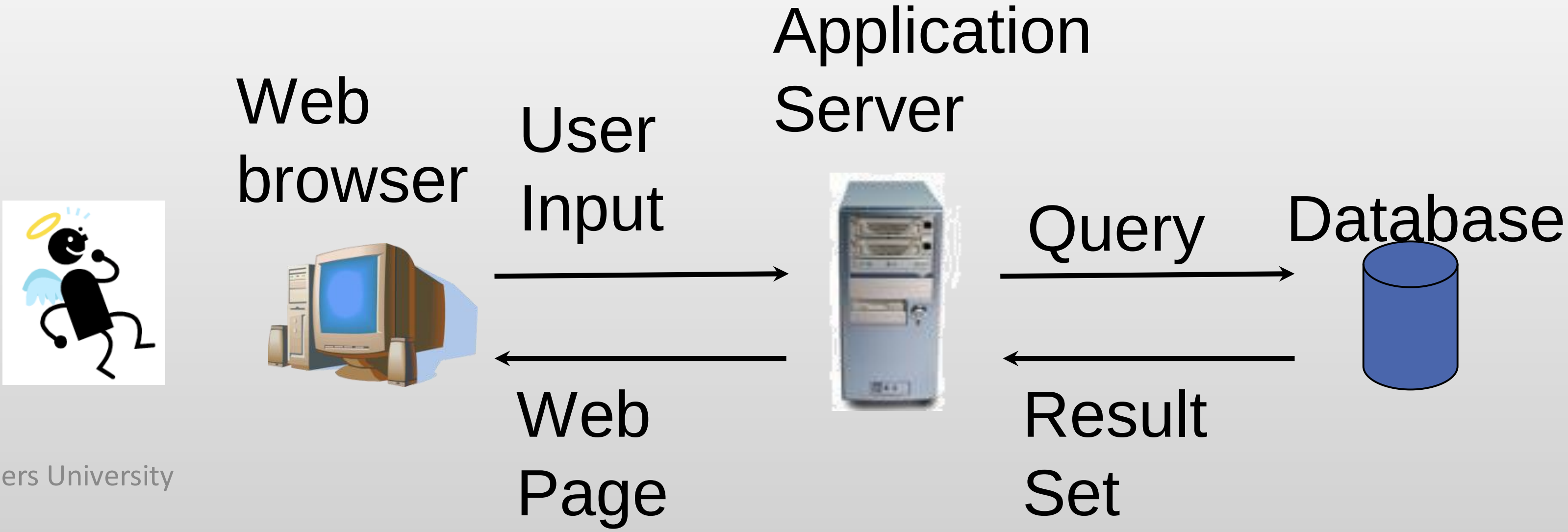
Password open_sesame

● Display ○ Delete

Submit

SELECT * FROM phonebook WHERE username = 'John' AND password = 'open_sesame'

John's phonebook entries are displayed



Web browser

User Input

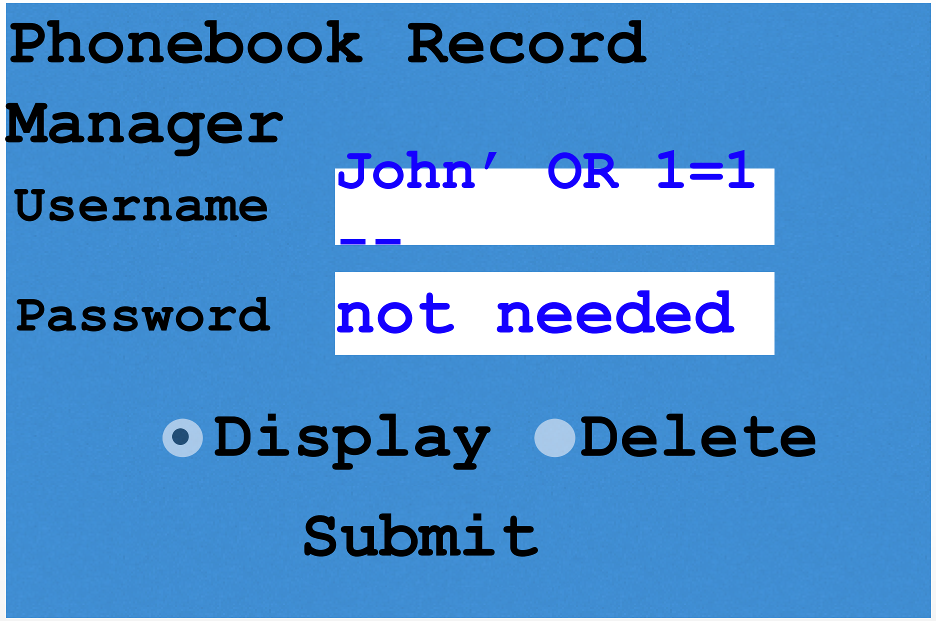
Application Server

Query

Database

Web Page

Result Set



Phonebook Record Manager

Username John' OR 1=1 --

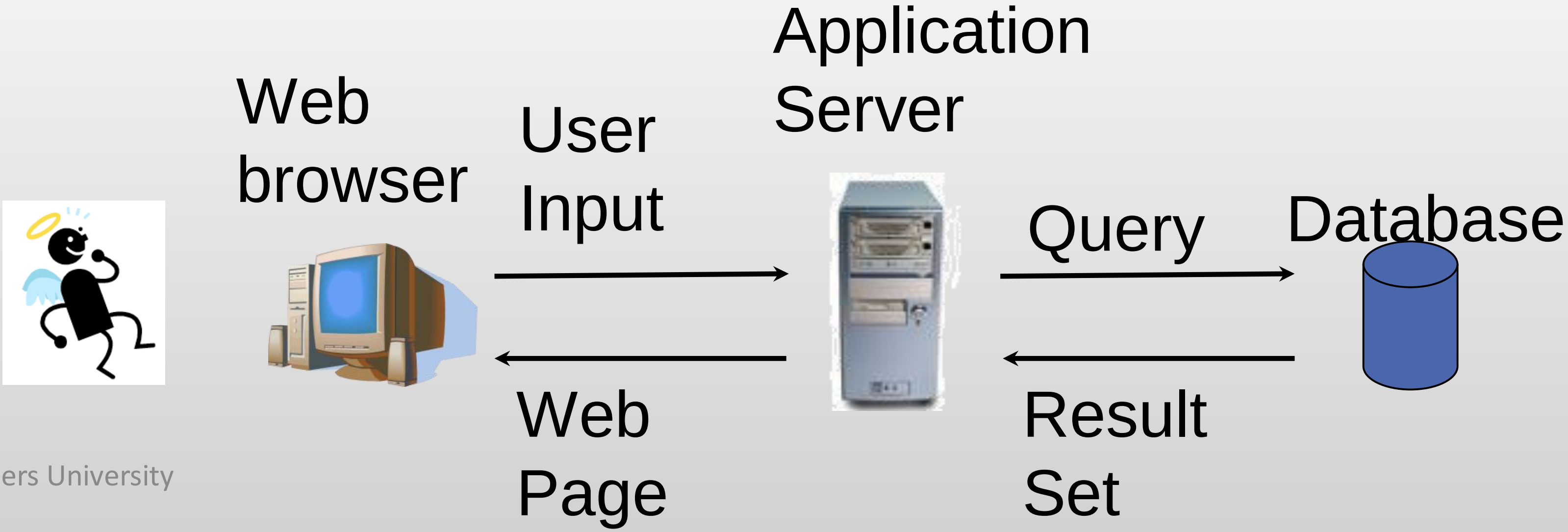
Password not needed

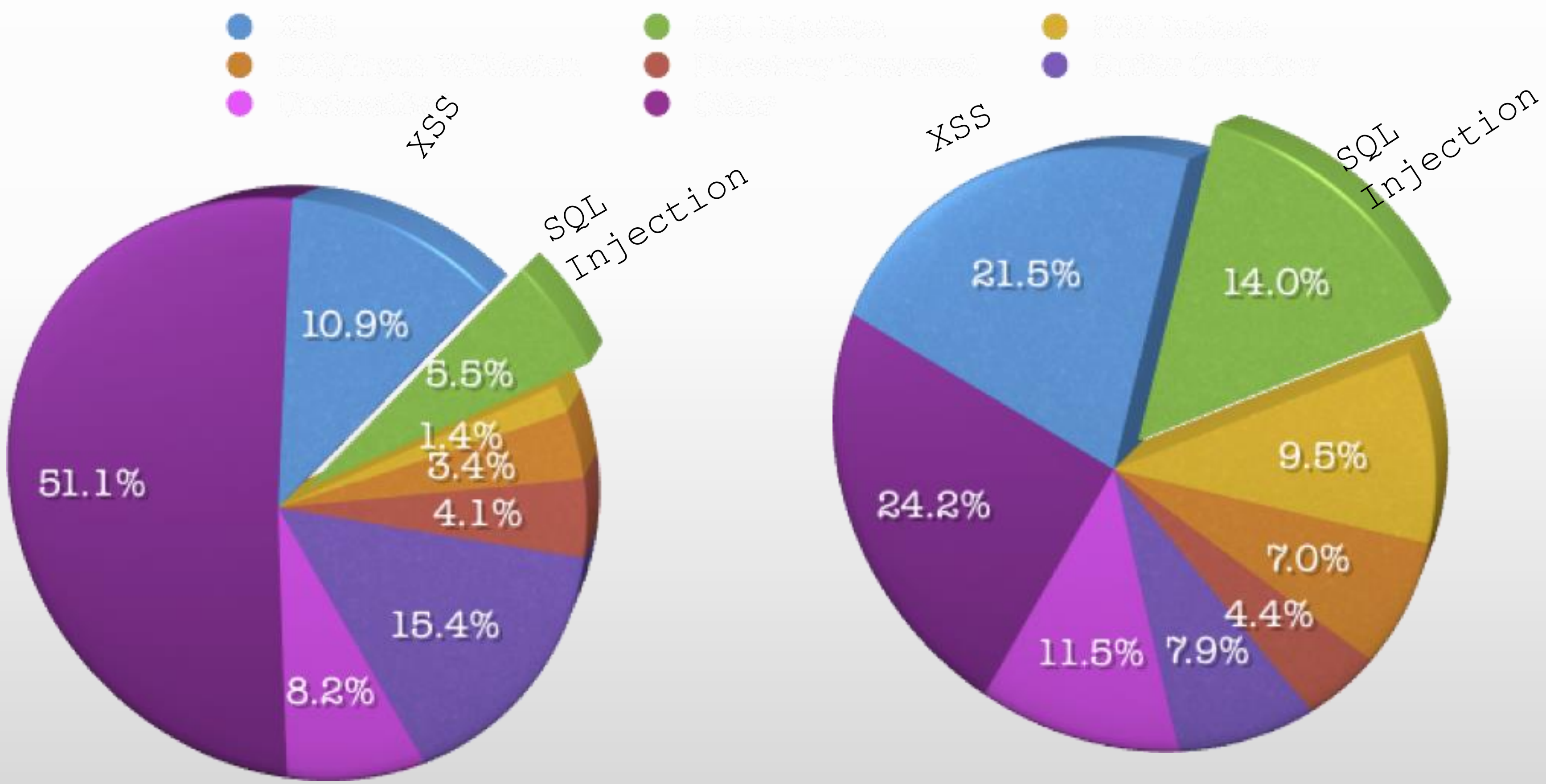
● Display ○ Delete

Submit

SELECT * FROM phonebook WHERE username = ‘John’ OR 1=1 --AND password = ‘not needed’

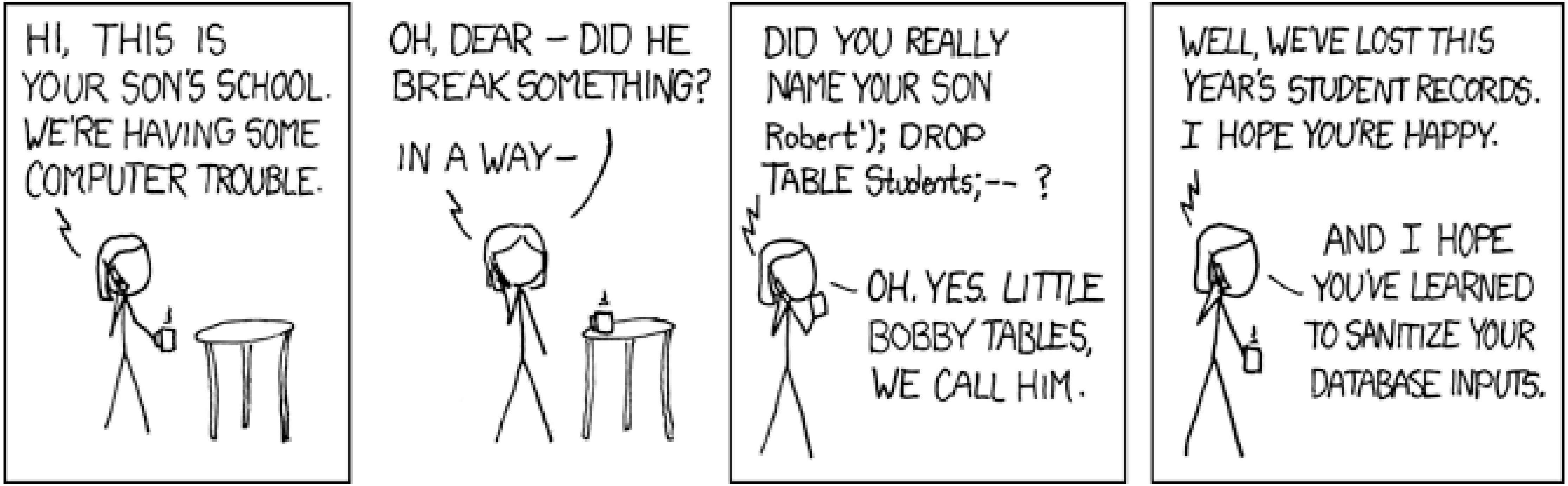
All phonebook entries are displayed




XSS
SQL Injection
PHP Includes
Directory Traversal
Other
XSS
SQL Injection
10.9%
5.5%
1.4%
3.4%
4.1%
15.4%
8.2%
51.1%
XSS
SQL Injection
21.5%
14.0%
9.5%
7.0%
4.4%
7.9%
11.5%
24.2%

SQL Injection

- Most systems separate code from data
- SQL queries can be constructed by arbitrary sequences of programming constructs that involve string operations
  - Concatenation, substring ….
- Such construct also involve (untrusted) user inputs
  - Inputs should be mere “data”, but in case of SQL results in “code”
- Result: Queries intended by the programmer can be “changed” by untrusted user input


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Acknowledgments: xkcd.com

Next Class

- Network Security