Web Security: Attacks & Defenses

Dawn Song
dawnsong@cs.berkeley.edu

Class Projects

- Nov 12, no class
- Nov 14, Milestone Report Due
  - Electronic submission before class
  - All electronic submission goes to summary gmail account
  - Hardcopy submission in class
- Nov 15, Milestone Report Feedback
  - 1-2:50pm
  - 10 min per group
  - Remember your time slot
- Poster session:
  - Dec 5, 4-6pm, Woz
  - Report due by 4pm, Dec 5
  - Electronic submission to summary gmail account
  - Hardcopy submission to office mailbox

Milestone Report

- Enhance the proposal document
- Clear problem definition, motivation, & scope
- Proposed approach
- Proposed metrics of success
- Time plan
Guest Lecture Planning

- Last lecture: historical view in web security
  - Input validation
    - Session management
- This lecture: some other attacks & defenses in web security
  - Input validation
- Oct 31, Guest Lecture (Raph, Google)
  - Trust metrics & sybil attacks in social networks
  - Pioneered work in this area
- Nov 5, Guest Lecture (Ophir, Director of Security R&D at VMWare)
  - Security issues & applications in virtualization
  - More of an open discussion format
- Nov 7, Guest Lecture (Kourosh, Team Lead of Google Traffic Quality Team)
  - AdFraud

Input Validation

- SQL injection attack
- XSS attack
- HTTP Response Splitting attack

SQL Injection
The setup

- User input is used in SQL query
- Example: login page (ASP)

```sql
set ok = execute("SELECT * FROM UserTable
WHERE username='" & form("user") & '
AND password='" & form("pwd") & '"");
If not ok.EOF
  login success
else fail;
```

- Is this exploitable?

Bad input

- Suppose `user = 'or 1 = 1 -- '` (URL encoded)
- Then scripts does:
  ```sql
  ok = execute( SELECT ...
  WHERE username=' 'or 1=1 -- ... )
  - The '--' causes rest of line to be ignored.
  - Now ok.EOF is always false.
  ```

- The bad news: easy login to many sites this way.

Even worse

- Suppose user = 
  ```cmdshell
  'net user badguy badpwd' / ADD --
  ```
- Then script does:
  ```sql
  ok = execute( SELECT ...
  WHERE username=' 'exec ... )
  ```

If SQL server context runs as "sa", attacker gets account on DB server.
Cross-Site Scripting (XSS) Attacks

The setup

• 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)

• 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
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
- 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://>
  - ... 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.
HTTP Response Splitting

The setup

• User input echoed in HTTP header.
• Example: Language redirect page (JSP)
  `<% response.sendRedirect("/by_lang.jsp?lang=\n  request.getParameter("lang")\n  %>`
• 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
  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
```

So what?

- What just happened:
  - Attacker submitted bad URL to victim.com
    - URL contained spoofed page in it
  - Got back spoofed page

- So what?
  - Cache servers along path now store spoof of victim.com
  - Will fool any user using same cache server

Defense

- Lack of types, hidden assumption
- Input validation
  - Taint tracking: figure out what variables need to be sanitized
    - Static taint analysis: Challenges?
    - 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
Other Defenses

- **Client side XSS defense**
  - Defense against reflected XSS attack
    - Check out-going requests with incoming responses for overlapping javascripts
  - Defense against XSS attack from stealing info
    - Check whether sensitive info is sent to another site
- **New browser tags**
  - How does Mashup OS address XSS attack?
  - What other tags you may want to add?

Session Management

- **Cookie forgery**
- **Cross-site Request Forgery (CSRF)**

Cookie Forgery
Cookies

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

  ![Diagram](image)

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

  - Browser
  - GET ...
  - Cookie: NAME = VALUE

  **Http is stateless protocol; cookies add state**

- **Browser 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.

- **Similar behavior with hidden fields:**
  
  `<INPUT TYPE="hidden" NAME=price VALUE="150">`
Prevalent  (as of 2/2000)

• 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:
      - 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:

• 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">

Defense

• Cookie authentication alone is insufficient

• Request also contains a hidden field using a shared secret btw client & server

• Other defenses?

Summary

• Web is complex & constantly evolving, web security is tricky

• Many other attacks

• http://www.owasp.org