CS 361S

Web Authentication and Session Management

Vitaly Shmatikov

Reading Assignment

- Read Kaufman, Chapter 25
- Read “Dos and Don’ts of Client Authentication on the Web”

HTTP Digest Authentication

client

server

Request URL with
GET or POST method

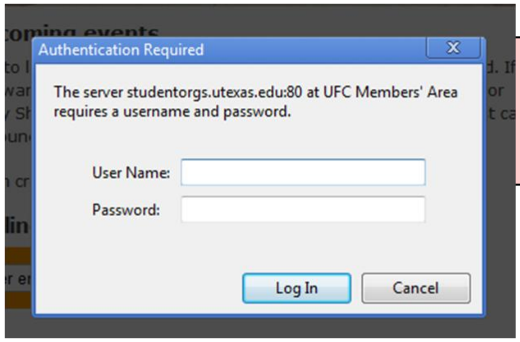
- HTTP 401 Unauthorised
- Authentication "realm"
  (description of system being accessed)
- Fresh, random nonce

H1=hash(username,
realm, password)
H2=hash(method, URL)

H3=hash(H1, server nonce, H2)

Recompute H3
and verify

WWW-Authenticate:
Basic realm="Password Required"



Authentication Required

The server studentorgs.utexas.edu:80 at UFC Members' Area requires a username and password.

User Name:

Password:

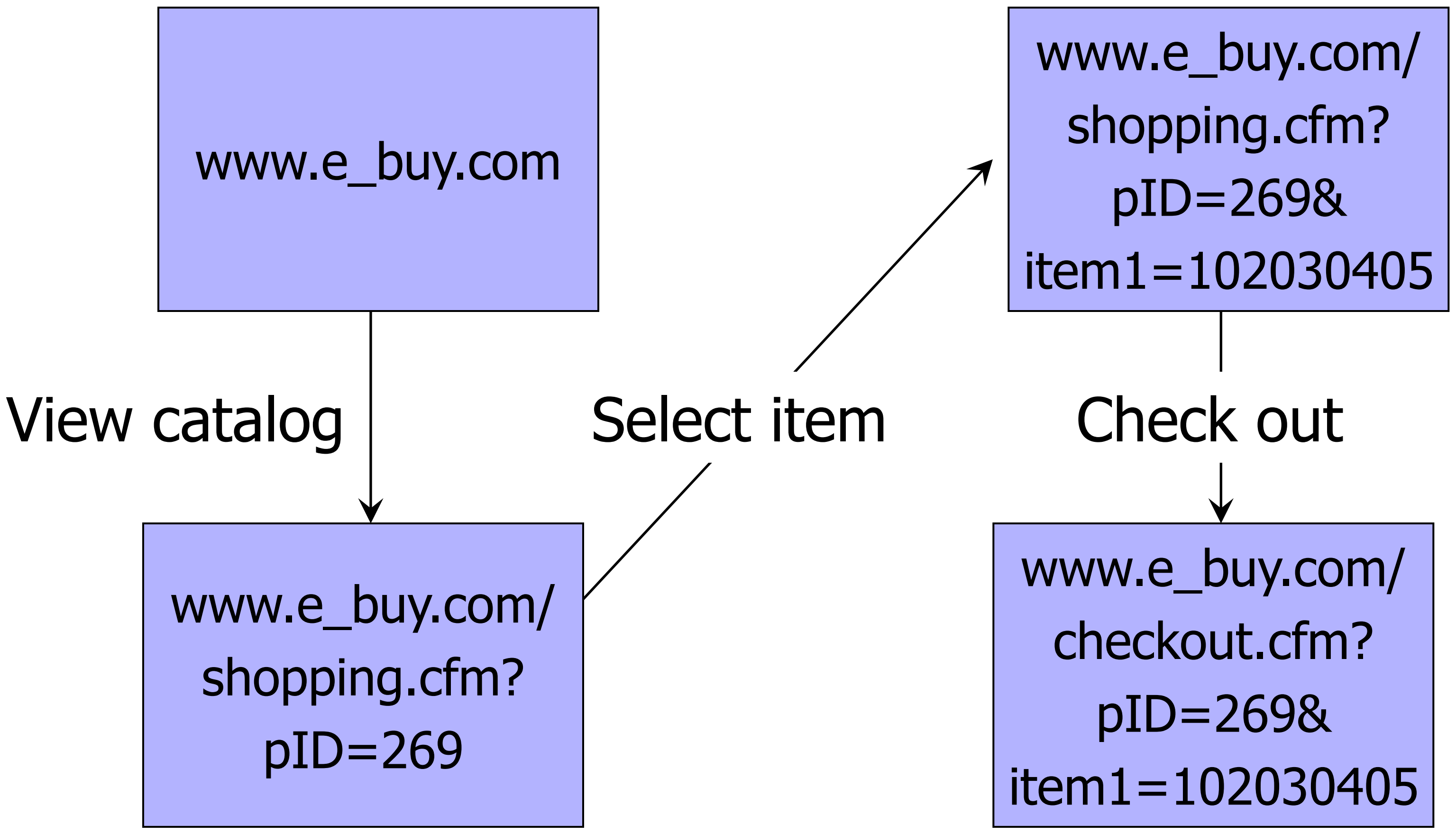
Problems with HTTP Authentication

- Can only log out by closing browser
  - What if user has multiple accounts? Multiple users of the same browser?
- Cannot customize password dialog
- Easily spoofed
- In old browsers, defeated by TRACE HTTP
  - TRACE causes Web server to reflect HTTP back to browser, TRACE via XHR reveals password to a script on the web page, can then be stolen
- Hardly used in commercial sites

Sessions

- A sequence of requests and responses from one browser to one or more sites
  - Can be long or short (Gmail – 2 weeks)
  - Without session management, users would have to constantly re-authenticate
- Session management
  - Authorize user once
  - All subsequent requests are tied to user

Primitive Browser Session


www.e_buy.com
View catalog
www.e_buy.com/ shopping.cfm? pID=269
Select item
www.e_buy.com/ shopping.cfm? pID=269& item1=102030405
Check out
www.e_buy.com/ checkout.cfm? pID=269& item1=102030405

Store session information in URL; easily read on network

Bad Idea: Encoding State in URL

- Unstable, frequently changing URLs
- Vulnerable to eavesdropping
- There is no guarantee that URL is private
  - Early versions of Opera used to send entire browsing history, including all visited URLs, to Google

Storing State in Hidden Forms

- 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">

  Black Leather purse with leather straps<

  <INPUT TYPE=HIDDEN NAME=name      VALUE="Black leather purse">
  <INPUT TYPE=HIDDEN NAME=price     VALUE="20.00">
  <INPUT TYPE=HIDDEN NAME=sh        VALUE="1">
  <INPUT TYPE=HIDDEN NAME=img       VALUE="p
  <INPUT TYPE=HIDDEN NAME=custom1   VALUE="B
       with leather straps">

  <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">

</FORM>
```

Change this to 2.00

Bargain shopping!

Shopping Cart Form Tampering

http://xforce.iss.net/xforce/xfdb/4621

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

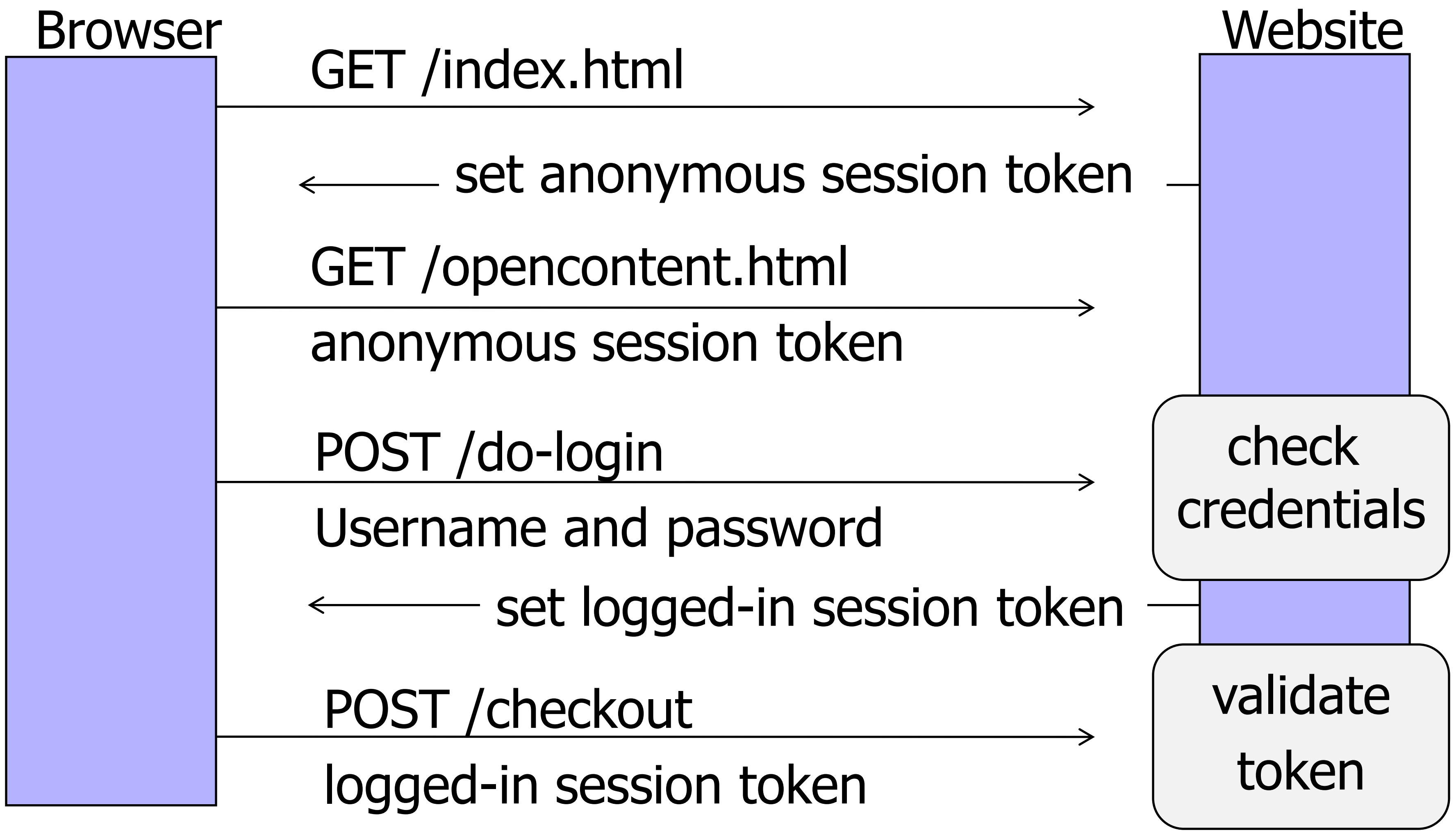
Other Risks of Hidden Forms

[From "The Art of Intrusion"]

Estonian bank's Web server…

- HTML source reveals a hidden variable that points to a file name
- Change file name to password file
- Server displays contents of password file
  - Bank was not using shadow password files!
- Standard cracking program took 15 minutes to crack root password

Session Tokens (Identifiers)



Generating Session Tokens (1)

- Option #1: minimal client state
- Token = random, unpredictable string
  - No data embedded in token
  - Server stores all data associated with the session: user id, login status, login time, etc.
- Potential server overhead
  - With multiple sessions, lots of database lookups to retrieve session state

Generating Session Tokens (2)

- Option #2: more client-side state
- Token = [user ID, expiration time, access rights, user info …]
- How to prevent client from tampering with his session token?
  - HMAC(server key, token)
- Server must still maintain some user state
  - For example, logout status (check on every request) to prevent usage of unexpired tokens after logout

FatBrain.com circa 1999

[Fu et al.]

- User logs into website with his password, authenticator token is generated, user is given a special URL containing the token

  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…
- Tokens are global sequence numbers
  - Easy to guess sequence number for another user

    https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555752
  - Fix: use random session tokens

Examples of Weak Tokens

- Verizon Wireless: counter
  - Log in, get counter, can view sessions of other users
- Apache Tomcat: generateSessionID()
  - MD5(PRNG) … but weak PRNG
    - PRNG = pseudo-random number generator
  - Result: predictable SessionID's
- ATT's iPad site: SIM card ID in the request used to populate a Web form with the user's email address
  - IDs are serial and guessable
  - Brute-force script harvested 114,000 email addresses

41 months in federal prison

Binding Token to Client’s Machine

Embed machine-specific data in the token…

- Client’s IP address
  - Harder to use token at another machine if stolen
  - If honest client changes IP address during session, will be logged out for no reason
- Client’s browser / user agent
  - A weak defense against theft, but doesn’t hurt
- HTTPS (TLS) session key
  - Same problem as IP address (and even worse)

Storing Session Tokens

- Embed in URL links
  - https://site.com/checkout ? SessionToken=kh7y3b
- Browser cookie
  - Set-Cookie: SessionToken=fduhye63sfdb
- Store in a hidden form field
  - <input type="hidden" name="sessionid" value="kh7y3b">
- Window.name DOM property

Issues

- Embedded in URL link
  - Token leaks out via HTTP Referer header
- Browser cookie
  - Browser sends it with every request, even if request not initiated by the user (cross-site request forgery)
- Hidden form field
  - Short sessions only
- DOM property
  - Not private, does not work if user connects from another window, short sessions only

HTTP Referer Header

GET /users/shmat HTTP/1.1

200 323

Referer:

http://www.google.com/search?q=shmatikov 361S solutions&hl=en ...

Referer leaks URL content (including session tokens) to any destination linked from the site

Typical Redirection Code

```
If (condition 1)
   redirect (http://site.com/B)
If (condition2)
   redirect (http://site.com/C/?sessionid=Au45fhds)
```

- User not logged in? Redirect to login page.
- User not admin? Redirect to access denied page.
- User admin? Show the admin menu.

XSUH: Cross-Site URL Hijacking

http://soroush.secproject.com/downloadable/XSUH_FF_1.pdf

- Firefox: modify window.onerror object to trap errors
- Learn destination, URL parameters of redirected page

Session token!

```
<script>
var destinationPage = 'http:// … your target here …';
window.onerror=fnErrorTrap;
function fnErrorTrap(sMsg, sUrl, sLine){
    alert('Source address was: ' + destinationPage +
    \n\nDestination URL is: ' + sUrl);
    return false;
}
document.write('<script src="'+destinationPage+'"><\/script>')
</script>
```

This will generate an error (why?)
Source of that error: final page after all redirections

Defenses Against XSUH

- Do not put session IDs, credentials, tokens, any important data into URLs
- Use POST and JavaScript to send confidential information to another destination
- Use AJAX to send/receive application messages
- Frame busting to prevent your page from being framed by other sites

Cookies

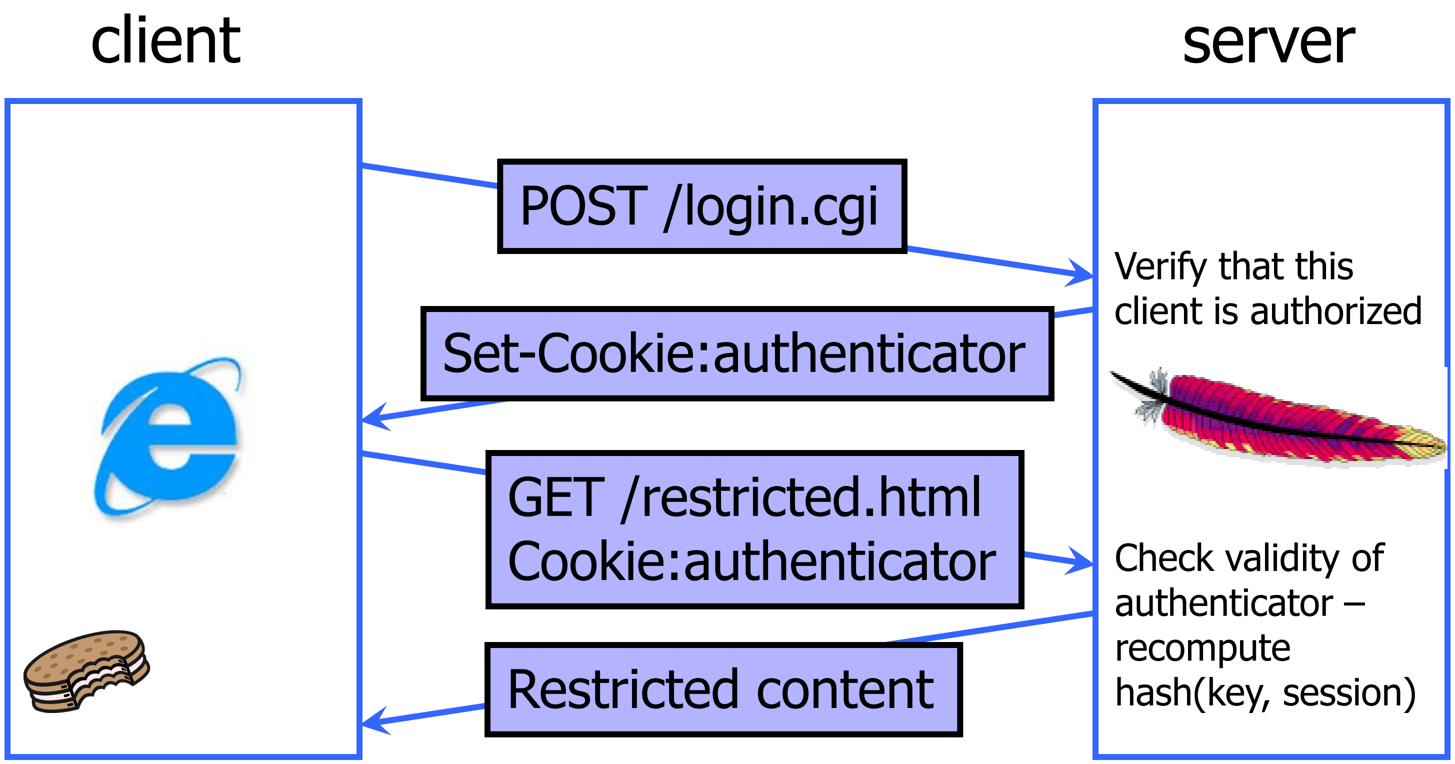
Storing State in Browser Cookies

- Set-cookie: price=299.99
- User edits the cookie… cookie: price=29.99
- What’s the solution?
- Add an HMAC to every cookie, computed with the server’s secret key
    - Price=299.99; HMAC(ServerKey, 299.99)
- But what if the website changes the price?

Web Authentication with Cookies

- Authentication system that works over HTTP and does not require servers to store session data
  - … except for logout status
- After client successfully authenticates, server computes an authenticator token and gives it to the browser as a cookie
  - Client should not be able forge authenticator on his own
    - Example: HMAC(server's secret key, session information)
- 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



Authenticators must be unforgeable and tamper-proof

(malicious client shouldn't be able to compute his own or modify an existing authenticator)

WSJ.com circa 1999

[Fu et al.]

- Idea: use hash(user,key) as authenticator
  - Key is secret and known only to the server… without the key, clients can't forge authenticators
- Implementation: crypt(user,key)
  - crypt() is 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

username	crypt(username,key,"00")	authenticator cookie
VitalySh1	008H8LRfzUXvk	VitalySh1008H8LRfzUXvk
VitalySh2	008H8LRfzUXvk	VitalySh2008H8LRfzUXvk
Create an account with a 7-letter user name…		
VitalySA	0073UYEre5rBQ	Try logging in: access refused
VitalySB	00bkHcfOXBKno	Access refused
VitalySC	00ofSJV6An1QE	Login successful! 1st key symbol is C
Now a 6-letter user name…		
VitalyCA	001mBnBErXRuc	Access refused
VitalyCB	00T3JLLfuspdo	Access refused… and so on

- Only need 128 x 8 queries instead of intended 128^8
- 17 minutes with a simple Perl script vs. 2 billion years

.NET 2.0

- System.Web.Configuration.MachineKey
  - Secret Web server key intended for cookie protection
  - Stored on all Web servers in the site
- Creating an encrypted cookie with integrity

  -
    ```
    HttpCookie  cookie = new HttpCookie(name, val);
    HttpCookie  encodedCookie =
                    HttpSecureCookie.Encode (cookie);
    ```
- Decrypting and validating an encrypted cookie
  - `HttpSecureCookie.Decode (cookie);`

Cookie Theft: SideJacking

- SideJacking = network eavesdropper steals cookies sent over a wireless connection
- Case 1: a website uses HTTPS for login, the rest of the session is unencrypted
  - Cookies must not be marked as “secure” (why?)
- Case 2: accidental HTTPS→HTTP downgrade
  - Laptop sees Wi-Fi hotspot, tries HTTPS to Web mail
  - This fails because first sees hotspot’s welcome page
  - Now try HTTP… with unencrypted cookie attached!
  - Eavesdropper gets the cookie – user’s mail is pwned

Cookie Theft: Surf Jacking

[illegible] http://resources.enablesecurity.com/resources/Surf%20Jacking.pdf

It is possible to <u>force</u> an HTTPS→HTTP downgrade

- Victim logs into https://bank.com
  - Cookie sent back encrypted and stored by browser
- Victim visits http://foo.com in another window
- Network attacker sends "301 Moved Permanently" in response to the cleartext request to foo.com
  - Response contains header "Location http://bank.com"
- Browser thinks foo.com is redirected to bank.com, starts a new HTTP connection, sends cookie in the clear – network eavesdropper gets the cookie!

Session Fixation Attacks

- Attacker obtains an anonymous session token (AST) for site.com
- Sets user's session token to attacker's AST
  - URL tokens: trick user into clicking on URL with the attacker's token
  - Cookie tokens: need an XSS exploit (more later)
- User logs into site.com
- Attacker's token becomes logged-in token!
- Can use this token to hijack user's session

Preventing Session Fixation

- When elevating user from anonymous to logged-in, always issue a new session token
- Once user logs in, token changes to value unknown to attacker

Logout Issues

- Functionality: allow login as a different user
- Security: prevent others from abusing account
- What happens during logout?
  1. Delete session token from client
  2. Mark session token as expired on server
- Many sites forget to mark token as expired, enabling session hijacking after logout
  - Attacker can use old token to access account