10. Web Authentication and Session Hijacking
ENEE 757 | CMSC 818V

Prof. Tudor Dumitraș
Assistant Professor, ECE
University of Maryland, College Park

http://ter.ps/757
https://www.facebook.com/SDSAtUMD

Today’s Lecture

• Where we’ve been
  – Security Basics
  – Passwords
  – Biometrics
  – Authorization logic

• Where we’re going today
  – Web authentication
  – Web session hijacking attacks

• Where we’re going next
  – Security analytics: Hands-on lab
  – Usability issues in authentication and access control
Analytics Lab and Homework 2

- Next week: hands-on analytics labs
  - Given by my graduate students Ziyun and Octavian (I will be at CCS)
  - Topics
    - Supervised and unsupervised learning
    - Graph analytics
    - TF-IDF
    - Locality sensitive hashing
    - Spark programming

- Homework 2: will cover these topics
  - You must program in Python
  - Not a hard language to learn; tutorials and reference materials, at http://www.umiacs.umd.edu/~tdumitra/courses/ENEE757/Fall15/homeworks.html

HTTP Digest Authentication

Client

- Request URL with GET or POST method

Server

- 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"
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
  – Cross-site tracing (XST): malicious scripts injected in the page may access cookies and HTTP headers normally inaccessible to Javascript
  • May reveal passwords and authenticators
• Hardly used in commercial sites

Sessions

• HTTP is a stateless protocol

• Need to provide the concept of a session: 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
  – Authenticate user once
  – All subsequent requests are tied to user
Priming 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
  – URLs are stored in Web server logs
Storing State in Hidden Forms

• Dansie Shopping Cart (2000)
  “A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order.”

```html
<FORM METHOD=POST ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">
  Black Leather purse with leather straps<br>
  Price: $20.00<br>
  <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="purse.jpg">
  <INPUT TYPE=HIDDEN NAME=custom1 VALUE="Black leather purse 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
  - Adgrafs: 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 2.6
  - McMurtrey/Whitaker & Associates: Cart32 3.0
  - Rich Media Technologies: JustAddCommerce 5.0
  - Web Express: Shoptron 1.2

pknutsen@nethut.no: CartMan 1.04
SmartCart: SmartCart Any version
Session Tokens (Identifiers)

Browser

GET /index.html

set anonymous session token

GET /opencontent.html

anonymous session token

POST /do-login

Username and password

set logged-in session token

POST /checkout

logged-in session token

Website

check credentials

validate token

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

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
  – Result: predictable SessionIDs

• ATT’s iPad site (2010):
  – 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
Cookies

Web Authentication with Cookies

- Cookie = mechanism for web servers to store state on the clients
  - Set-cookie: key=value

- 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**

Client:
- POST /login.cgi
- Set-Cookie: authenticator
- GET /restricted.html
- Cookie: authenticator
- Restricted content

Server:
- Verify that this client is authorized
- Check validity of authenticator – recompute hash(key, session)

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

---

**Dos and Don’ts of Web Authentication**

[Fu, Sit, Smith and Feamster, 2001]

- Why should authenticators be unforgeable?
- Why should authenticators be tamper proof?
- Why is SSL not enough for Web authentication?
- Why re-authenticate before changing passwords?
- How do you think Fu et al.’s attack against the Wall Street Journal worked?
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


It is possible to force 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: cookie injection attack
- 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
- Some sites forget to mark token as expired, enabling session hijacking after logout
  - Attacker can use old token to access account

Review of Lecture

- What did we learn?
  - Web authentication
  - Web session hijacking attacks

- Sources
  - Vitaly Shmatikov

- What’s next?
  - Security analytics: Hands-on lab
  - Usability issues in authentication and access control