SESSION MANAGEMENT
Sessions

- A sequence of requests and responses from one browser to one (or more) sites
  - Session can be long (Gmail — two weeks) or short
  - Without session management:
    users would have to constantly re-authenticate

- Session management:
  - Authorize user once
  - All subsequent requests are tied to user
Pre-history: HTTP auth

HTTP request: GET /index.html

HTTP response contains:

WWW-Authenticate: Basic realm="Password Required"

Browsers sends hashed password on all subsequent HTTP requests:

Authorization: Basic ZGFddfibzsdfgkjheczI1NXRleHQ=
HTTP Auth Problems

- Hardly used in commercial sites
  - User cannot log out other than by closing browser
    - What if user has multiple accounts?
    - What if multiple users on same computer?
  - Site cannot customize password dialog
  - Confusing dialog to users
  - Easily spoofed
Session Tokens

Browser

GET /index.html

set anonymous session token

GET /books.html
anonymous session token

POST /do-login
Username & password

elevate to a logged-in session token

POST /checkout
logged-in session token

Web Site

Check credentials

Validate token

55
Storing Session Tokens

- **Lots of options (but none are perfect)**
  - **Browser cookie:**
    
    ```
    Set-Cookie:  SessionToken=fduhye63sfdb
    ```
  - **Embed in all URL links:**
    
    ```
    https://site.com/checkout ? SessionToken=kh7y3b
    ```
  - **In a hidden form field:**
    
    ```
    <input type="hidden" name="sessionid" value="kh7y3b">
    ```
Storing Session Tokens: Problems

- **Browser cookie:**
  
browser sends cookie with every request, even when it should not

- **Embed in all URL links:**
  
token leaks via HTTP Referer header

- **In a hidden form field:**
  
can’t always use forms

**Best answer: A combination of all of the above**
Session Hijacking

- Attacker waits for user to login; then attacker obtains user’s session token and “hijacks” session

- Samy worm was an example
1. Predictable Tokens

- **Example: counter (Verizon Wireless)**
  \[ \Rightarrow \text{user logs in, gets counter value, can view sessions of other users} \]

- **Example: weak MAC (WSJ)**
  - token = \{userid, MAC_k(userid) \}
  - Weak MAC exposes k from few cookies

- **Apache Tomcat: generateSessionID()**

Session tokens must be unpredictable to attacker:

Use underlying framework (ASP, Tomcat, Jserv)
2. Cookie Theft

- **Example 1: login over SSL, but subsequent HTTP**
  - What happens at wireless cafe?
  - Other reasons why session token sent in the clear:
    - HTTPS/HTTP mixed content pages at site
    - Man-in-the-middle attacks on SSL

- **Example 2: Cross Site Scripting (XSS) exploits**

- **Amplified by poor logout procedures:**
  - Logout must invalidate token on server
Session Fixation Attacks

- Suppose attacker can set the user’s session token:
  - For URL tokens, trick user into clicking on URL
  - For cookie tokens, set using XSS exploits

- Attack: (say, using URL tokens)
  1. Attacker gets anonymous session token for site.com
  2. Sends URL to user with attacker’s session token
  3. User clicks on URL and logs into site.com
     - this elevates attacker’s token to logged-in token
  4. Attacker uses elevated token to hijack user’s session
Session Fixation: Lesson

- 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
  ⇒ Attacker’s token is not elevated
Session Management Takeaways

- Session management is a key enabler of web functionality

- Many ways to get it wrong
  - Predictable tokens, cookie theft, session fixation
Web Security Challenge

Guarantee security properties in the presence of adversary-supplied code and data
+ compatibility with current Web
+ usable for end-user
Two Representative Defenses

- **App Isolation**
  - Isolation between sensitive and non-sensitive sites

- **Native Client**
  - Software Fault Isolation for x86 native code modules in web applications
App Isolation: Get the Security of Multiple Browsers with Just One

[Chen et al., CCS 2011]
Multiple browsers for security

- Only use Chrome to log into bank
- Only use Firefox for the non-bank URLs
- Never type URLs from Firefox to Chrome

Is this really more secure?
Defeats cross-origin attacks

- Using multiple browsers defeats most *cross-origin* attacks
  - Origin: `<scheme>://<host>:`

- *cross-origin* attacks: attacker resides on different origin from victim

- Includes XSS and CSRF
Single Browser: CSRF Attack

Authenticate

Transfer $$

bank.com

evil.com
Multiple Browsers: CSRF Defense

**Rule #1:** Only use Chrome to log in to bank

**Rule #2:** Only use Firefox for the non-bank URLs
Problem solved?

- **Browsing with multiple browsers is difficult!**
  - Requires constant effort
  - $N$ sites = $N$ browsers?
  - Navigation is nearly impossible

- **Why not use just one browser?**
Problems with using one browser

1. Attack pages can request vulnerable URLs
2. Attack pages’ requests carry user’s credentials
3. Attack pages can exploit bugs to access user state

Claim:
These are not fundamental limitations of the browser!
Problems with using one browser

1. Attack pages can request vulnerable URLs
2. Attack pages’ requests carry user’s credentials
3. Attack pages can exploit bugs to access user state

Observation:
We can guard vulnerable URLs and isolate persistent/in-memory state
Problems with using one browser

1. Attack pages can request vulnerable URLs.
2. Attack pages’ requests carry user’s credentials
3. Attack pages can exploit bugs to access user state

Observation:
We can guard vulnerable URLs and isolate persistent/in-memory state
Problems with using one browser

1. Attack pages can request vulnerable URLs
2. Attack pages’ requests carry user’s credentials
3. Attack pages can exploit bugs to access user state

Observation:
We can guard vulnerable URLs and isolate persistent/in-memory state
Entry-point Restriction

- Goal: Don’t let attackers request vulnerable part of app
- List app’s “front door” and block other requests
Entry-point Restriction

- Apps provide a list of entry point URLs
- Other sites can only navigate to/embed these entry points
Guidelines for Entry Point

- **Entry point should**
  - cause no sensitive state change because attacker may request it with the user’s credentials
  - not return confidential information in a form that could be accessed on a cross-origin page

- **Entry point URL pattern should be tightly constrained to reduce possibility of attacker placing malicious code in URL**
Entry-point Restriction Implementation

- Apps opt-in using a manifest file
  - Similar to browser extensions
  - Developers specify entry point URLs

- Idea: Developers are more informed than users (better security and usability)
State Isolation

Don’t let an App **share a process** with others!
State Isolation

Browser Kernel

mail.com
Sandbox

bank.com
Sandbox

evil.com
Sandbox

Mail cookies
Bank cookies
Other cookies

Partition disk storage for each app!
State Isolation

Side effect:
third party cookies do not work for state isolation
App Isolation Takeaways

- Using multiple browsers is good for security!
  - Boils down to entry-point restriction and state isolation

- Can be achieved using one browser
  - Without too much additional compatibility cost
Acknowledgments


- Many slides for App Isolation provided by E. Chen
- Some slides for Native Client provided by D. Brumley