CS 203 / NetSys 240

Web Security

Web Browser and the Internet

Browser
OS
Hardware

Website (server) somewhere on the Internet

request
reply
HTTP: HyperText Transfer Protocol

- Used to request and return data
  - Methods: GET, POST, HEAD, ...
- Stateless request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications
- Evolution
  - HTTP 1.0: simple
  - HTTP 1.1: more complex
  - Etc, etc

HTTP Request

<table>
<thead>
<tr>
<th>Method</th>
<th>File</th>
<th>HTTP version</th>
<th>Headers</th>
</tr>
</thead>
</table>
| GET      | /default.asp | HTTP/1.0     | Accept: image/gif, image/x-bitmap, image/jpeg, */*
|          |              |              | Accept-Language: en
|          |              |              | User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
|          |              |              | Connection: Keep-Alive
|          |              |              | If-Modified-Since: Sunday, 17-Apr-96 04:32:58 GMT

Data – none for GET
HTTP Response (PUT)

HTTP version | Status code | Reason phrase |
-------------|-------------|---------------|
HTTP/1.0 200 OK | Date: Sun, 21 Apr 1996 02:20:42 GMT | Server: Microsoft-Internet-Information-Server/5.0 |
| Connection: keep-alive | | Content-Type: text/html |
| Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT | | Content-Length: 2543 |

<Data>
Some data... blah, blah, blah</Data>

HTTP Digest Authentication

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
**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

Store session information in URL; easily read on network

---

**FatBrain.com circa 1999**

- 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 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
  - Fix: use random authenticators

SEE: [http://cookies.lcs.mit.edu/seq_sessionid.html](http://cookies.lcs.mit.edu/seq_sessionid.html)
Examples of Weak Authenticators

◆ Verizon Wireless: counter
  • User logs in, gets counter, can view sessions of other users

◆ Apache Tomcat: generateSessionID()
  • MD5(PRNG) ... but weak PRNG
    – PRNG = pseudo-random number generator
  • Result: predictable SessionID’s

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
  • Also, consider HTTP proxies...
Cookies

Storing Info Across Sessions

- A **cookie** is a file created by a website to store information in your browser.

HTTP is a stateless protocol; cookies add state.
What Are Cookies Used For?

◆ Authentication
  • Use the fact that the user authenticated correctly in the past to make future authentication quicker
◆ Personalization
  • Recognize the user from a previous visit
◆ Tracking
  • Follow the user from site to site; learn his/her browsing behavior, preferences, and so on

Cookie Management

◆ Cookie ownership
  • Once a cookie is saved on your client device, only the website that created the cookie can read it
    – If cookie is “secure”, browser will only send it over HTTPS
    – ... but anyone can write a secure cookie!
◆ Variations
  • Temporary cookies: stored until you quit your browser
  • Persistent cookies: remain until deleted or expire
  • Third-party cookies: originate on, or sent to, another website
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
  • 2o7.net tracking cookie
◆ Browser attacks could invade your “privacy”

November 8, 2001:
Users of Microsoft’s browser and e-mail programs could be vulnerable to having their browser cookies stolen or modified due to a new security bug in Internet Explorer (IE), the company warned today.

The website “adinterax.com” has requested to save a file on your computer called a “cookie.” This file may be used to track usage information...
The Weather Channel

The website "twci.coremetrics.com" has requested to save a file on your computer called a “cookie.” This file may be used to track usage information...

MySpace

The website "insightexpressai.com" has requested to save a file on your computer called a “cookie”...
Let’s Take a Closer Look…

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.”

```html
<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="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

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 2.6
- 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
- pknutsen@nethut.no: CartMan 1.04
- SmartCart: SmartCart Any version
- Web Express: Shoptron 1.2

Other Risks of Hidden Forms

- Estonian bank’s web server
- HTML source reveals a hidden variable that points to a file name
- Change file name to password file
- Web server displays contents of password file
  - Bank was not using shadow password files!
- Standard cracking program took 15 minutes to crack root password
Storing State in Browser Cookies

- Set-cookie: price=299.99
- User edits the cookie... cookie: price=29.99
- What’s the solution?
- Add a MAC 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 via Cookies

- Need authentication system that works over HTTP and does not require servers to store session data
  - Why is it a bad idea to store session state on server?
- Servers can use cookies to store state on client
  - After client successfully authenticates, server computes an authenticator and gives it to browser in a cookie
    - Client cannot forge authenticator on his own
    - Example: hash(server’s secret key, session id, other stuff)
  - 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 (e.g., recompute hash(key, sessId))

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

WSJ.com around 1999

• 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.
• Implementation: user, 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

[Fu et al.]
Attack

<table>
<thead>
<tr>
<th>username</th>
<th>crypt(username, key, &quot;00&quot;)</th>
<th>authenticator cookie</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpsonH1</td>
<td>008H8LRFzUXvk</td>
<td>SimpsonH1008H8LRFzUXvk</td>
</tr>
<tr>
<td>SimpsonH2</td>
<td>008H8LRFzUXvk</td>
<td>SimpsonH2008H8LRFzUXvk</td>
</tr>
</tbody>
</table>

It gets worse... The WSJ “scheme” can be exploited to extract the server’s secret key.

Read the paper!

Dos and Don’ts of Client Authentication on the Web
Kevin Fu, Emil Sit, Kendra Smith, Nick Feamster

Better Cookie Authenticator

- Main lesson: don’t bake your own!
  - Homebrewed authentication schemes are often flawed
- There are standard cookie-based schemes

- Capability: Describes what user is authorized to do on the site that issued the cookie
- Expiration: Cannot be forged by malicious user; does not leak server secret
- Hash(server secret, capability, expiration)
Kerberos

Many-to-Many Authentication

How do users prove their identities when requesting services from machines on the network?

Naïve solution: every server knows every user’s password

- Insecure: break into one server ⇒ compromise all users
- Inefficient: to change password, user must contact every server
Requirements

◆ Security
  • Must be secure against attacks by passive eavesdroppers and actively malicious attackers (including rogue users)

◆ Reliability
  • Must be always available

◆ Transparency
  • Users should not notice authentication taking place
  • Entering password is OK, if done rarely enough

◆ Scalability
  • Must handle large numbers of users and servers

Threats

◆ User impersonation
  • Malicious user with access to a workstation pretends to be another user from the same workstation
    – Can’t trust workstations to verify users’ identities

◆ Network address impersonation
  • Malicious user changes network address of his workstation to impersonate another workstation
    – Can’t trust network addresses

◆ Eavesdropping, tampering and replay
  • Malicious user eavesdrops on, tampers with, or replays, other users’ conversations to gain unauthorized access
Solution: Trusted Third Party

- Trusted authentication service
  - Knows all passwords, can grant access to any server
  - Convenient, but also single point of failure
  - Requires high level of physical security

User proves its identity; requests ticket for some service

User receives ticket

Ticket is used to access desired network service

Knows all users’ and servers’ passwords

What Should a Ticket Look Like?

- Ticket cannot include server plaintext password
  - Otherwise, next time user will access server directly without proving its identity to authentication service

- Solution: encrypt some information with a key known to the server, but not to the user!
  - Server can decrypt ticket and verify information
  - User does not learn server key
What Should a Ticket Include?

- User name
- Server name
- Address of user’s workstation
  - Otherwise, a user on another workstation can steal the ticket and use it to gain access to the server
- Ticket lifetime
- A few other things (e.g., session key)

How to authenticate initially?

- Insecure: passwords are sent in plaintext
  - Eavesdropper can steal password and impersonate user
- Inconvenient: need to send the password each time to obtain the ticket for any network service
  - Separate authentication for email, printing, etc.
Two-Step Authentication

- Prove identity **once** to obtain special **TGS ticket**
- Use TGS to get tickets for any network service

Still Not Good Enough

- **Ticket hijacking**
  - Malicious user may steal the service ticket of another user on the same workstation and use it
    - IP address verification does not help
  - Servers must verify that the user who is presenting the ticket is the same user to whom the ticket was issued
- **No server authentication**
  - Attacker may mis-configure the network so that it receives messages addressed to a legitimate server
    - Capture private information from users and/or deny service
  - Servers must prove their identity to users
Symmetric Keys in Kerberos

- \( K_c \) is long-term key of client C
  - Derived from user’s password
  - Known to client and key distribution center (KDC)
- \( K_{TGS} \) is long-term key of TGS
  - Known to KDC and ticket granting service (TGS)
- \( K_v \) is long-term key of network service V
  - Known to V and TGS; separate key for each service
- \( K_{c,TGS} \) is short-term key between C and TGS
  - Created by KDC, known to C and TGS
- \( K_{c,V} \) is short-term key between C and V
  - Created by TGS, known to C and V

“Single Logon” Authentication

- Client only needs to obtain TGS ticket once (say, every morning)
  - Ticket is encrypted; client cannot forge it or tamper with it
Obtaining a Service Ticket

- Client uses TGS ticket to obtain a service ticket and a short-term key for each network service
  - One encrypted, unforgeable ticket per service (printer, email, etc.)

System command, e.g. "lpr --print"

Client

Knows $K_{TGS}$ and ticket$_{TGS}$

Ticket Granting Service (TGS) usually lives inside KDC

Knows key for each service

Obtaining Service

- For each service request, client uses the short-term key for that service and the ticket he received from TGS
Kerberos in Large Networks

- One KDC isn’t enough for large networks (why?)
- Network is divided into realms
  - KDCs in different realms have different key databases
- To access a service in another realm, users must...
  - Get ticket for home-realm TGS from home-realm KDC
  - Get ticket for remote-realm TGS from home-realm TGS
    - As if remote-realm TGS were just another network service
  - Get ticket for remote service from that realm’s TGS
  - Use remote-realm ticket to access service
  - N(N-1)/2 key exchanges for full N-realm interoperation

Summary of Kerberos

1. User logs on to workstation and requests service on host.
2. Authentication Server (AS) verifies user’s access rights.
3. Workstation prompts user for password and uses password to decrypt session key tag.
4. AS decrypts session key tag to get an authenticator that contains user’s name, network address, and time to TGS.
5. User receives ticket and authenticator.
6. Workstation sends ticket and authenticator to server.
7. Ticket-granting Server (TGS) verifies ticket and authenticator and issues a service ticket.
8. TGS decrypts ticket and authenticator, verifies request, then issues ticket for requested service.
Important Ideas in Kerberos

◆ Short-term session keys
  • Long-term secrets used only to derive short-term keys
  • Separate session key for each user-server pair
    – ... but multiple user-server sessions re-use the same key
◆ Proofs of identity are based on authenticators
  • Client encrypts his identity, address and current time using a short-term session key
    – Also prevents replays (if clocks are globally synchronized)
  • Server learns this key separately (via encrypted ticket that client can’t decrypt) and verifies user’s identity
◆ Symmetric cryptography only

Problematic Issues

◆ Dictionary attacks on client master keys
◆ Replay of authenticators
  • 5-minute lifetimes long enough for replay
  • Timestamps assume global, secure synchronized clocks
  • Challenge-response would have been better
◆ Same user-server key used for all sessions
◆ “Homebrewed” PCBC mode of encryption
◆ Superfluous double encryption of tickets
◆ No ticket delegation
  • Printer can’t fetch email from server on your behalf
Kerberos Version 5

◆ Better user-server authentication
  • Separate subkey for each user-server session instead of re-using the session key contained in the ticket
  • Authentication via subkeys, not timestamp increments
◆ Authentication forwarding
  • Servers can access other servers on user’s behalf
◆ Realm hierarchies for inter-realm authentication
◆ Richer ticket functionality
◆ Explicit integrity checking + standard CBC mode
◆ Multiple encryption schemes, not just DES

Practical Uses of Kerberos

◆ Email, FTP, network file systems and many other applications have been kerberized
  • Use of Kerberos is transparent for the end user
  • Transparency is important for usability!
◆ Local authentication
  • login and su in OpenBSD
◆ Authentication for network protocols
  • rlogin, rsh, telnet
◆ Secure windowing systems
  • xdm, kx