

1  **HOW DO THEY DO IT?**

Impersonation

2 

"When the first just and friendly man appeared on the earth, from that day a fatal Waterloo was visible for all the men of pride and fraud and blood."

Charles Fletcher Dole (1845 - ?)

3 

Introduction

⊙ Impersonation

- The ability of one person to take on the identity of another and thereby subsume their role in society.
- Affects an estimated 750,000 people annually
- The #1 consumer complaint filed with the Federal Trade Commission (US)
- ⊙ "Cyber Impersonation" is even easier than in the physical world
 - Few understand the risks; few take the proper precautions

4 

Session Hijacking

⊙ A stolen identity & a broken date

- Alice meets Charles in a chat room
- Arrangements are made for a date
- Alice is stood up by Charles at a fancy Italian restaurant
- But what does Bob have to do with all this?
- Actually, quite a bit! Welcome to the Session Hijacking Love Triangle!
- Bob wanted to ask Alice out but had not yet (wimp!)

5 

Session Hijacking

⊙ A stolen identity & a broken date (cont)

- Bob is the company security administrator
- Let's run the whole process to see what really happened...
- ⊙ March 5, 7:00AM – Alice's Residence
 - Alice logs on to her personal email provider, <http://ewebmail.example.com>.
 - She emails Charles accepting his invitation to dinner & a movie
 - Ewebmail is similar to Gmail or Yahoo! Mail...

6 

Session Hijacking

⊙ March 5, 7:00AM – Alice's Residence (cont)

- Login mechanism uses typical username & password via HTML form
- Alice uses the "Compose" screen to write the email and sends the note
- Alice realizes she is late and hurries off to work...

7 

Session Hijacking

⊙ March 5, 8:30AM – Alice's Workplace

- In her work lobby, she tells Nichole, her "friend" about the date
- Nichole tells Bob; Bob upset; tries to sabotage the date
- Bob knows Alice uses eWebMail for personal email
- His first step is to create his own eWebMail account: bob@ewebmail.com

8 

Session Hijacking

⊙ March 5, 10:00AM – Bob's Office

- The features offered by eWebMail were similar to those offered by other email providers
- Bob studied how eWebMail was written...
 - Used Java servlets and JSP
 - Sent cookies to the browser
- Bob uses "Cookie Pal" (www.kburra.net) to manage cookies
- Discovers eWebMail sends a cookie named uid with a long, maybe hex value

9 

Session Hijacking

⊙ March 5, 10:00AM – Bob's Office (cont)

- Bob knew that some Web Apps. used cookies to manage session identifiers
- To see if this is the case, Bob creates 3 more accounts, bob1, bob2 & bob3
- Bob logs in to each to retrieve the cookie
- Bob compares the cookies to see if there is some way to “decode” it.

10 **Session Hijacking**

- ⊙ March 5, 10:00AM – Bob’s Office (cont)
 - 2 things become obvious when the cookies are laid out:
 1. The number of bytes are exactly the same
 2. There is only a 1 byte difference between the last 3 email addresses

```
C8C5C8EACFDDFC8C7CBC3C684CFD2CBC7DAC6CF84C9C5C7:1
C8C5C89BEACFDDFC8C7CBC3C684CFD2CBC7DAC6CF84C9C5C7:1
C8C5C898EACFDDFC8C7CBC3C684CFD2CBC7DAC6CF84C9C5C7:1
C8C5C899EACFDDFC8C7CBC3C684CFD2CBC7DAC6CF84C9C5C7:1
```

 - It seemed that the email addresses were somehow encoded into the cookie
 - Simply using the ASCII encoding of the email address seemed (and was) not done
 - 1.

11 **Session Hijacking**

- ⊙ March 5, 10:00AM – Bob’s Office (cont)
 - The cookie could be an encrypted email address, though
 - Since the strings were almost identical, Bob tried a simple XOR encryption technique to decode
 - A simple Perl script does the trick by XORing all the bytes of the cookie by all 256 values
 - 0xAA did the trick! The cookie strings were encoded by XORing every character in the email address with 0xAA

12 **Session Hijacking**

- ⊙ March 5, 10:00AM – Bob’s Office (cont)
 - Bob’s not sure what the “:1” is for..
 - Bob also noticed that the cookie expiration is set to 1 hour past the time the cookie was set.
 - Bob has all he needs to try to hack into Alice’s email account!
 - Bob creates a cookie string using the above info for Alice’s email address

13 **Session Hijacking**

- ⊙ 11:00AM – Bob’s Office
 - Bob logs in to his eWebMail account in Netscape
 - Closes Netscape
 - He then edits his Netscape cookies file, cookies.txt
 - Searches for the eWebMail cookie
 - Replaces the value of uid with his computed uid value
 - Reopens Netscape and requests <http://ewedmail.example.com>
 - Voila! Alice’s eWebMail account!

14 **Session Hijacking**

- ⊙ 11:00AM – Bob’s Office (cont)
 - Bob sees Charles’ replay suggesting a different restaurant
 - Bob deletes it... Alice will never meet Charles!
- ⊙ 12:30PM – Alice’s Office
 - Alice’s meeting lasts longer than she thought
 - She checks her personal email before she goes to lunch
 - She thinks all is well for the Italian restaurant...

15 **Session Hijacking**

- ⊙ 9:30PM – Bertolini’s Italian Cuisine

- She is waiting for him...
- He's waiting for her at Las Brisas
- Alas, the 2 shall not meet!
- ⊙ What really is wrong here?!?
 - Despite all the changes that have happened to the Web over the years, HTTP remained exactly the same: stateless!
 - Poorly implemented state-maintaining schemes result in hacks like session hijacking

16  **HTTP & Session Tracking**

- ⊙ So, if HTTP is stateless how is session tracking achieved over HTTP?
- ⊙ Example: Alice's use of eWebMail
- ⊙ Look at the state diagram...

17 

18  **HTTP & Session Tracking**

- ⊙ State: Login (1)
 - Alice:
 - Starts at the login state
 - Credentials are verified
 - Transition out to "Check Email", then to "Inbox" state
 - App:
 - Creates a "UID" and returns it as a cookie
 - Sets expiration of the cookie
 - Now, cookie sent with each HTTP request
 - Sends an HTTP response with the Inbox view as the data

Note: the user and the app. are only logically connected; no ongoing connection

19 

20  **HTTP & Session Tracking**

- ⊙ State: Inbox (2)
 - Alice clicks "Compose" to write a new email
 - Browser sends request w/ cookie
 - Server gets request, decodes cookie w/ XOR algorithm and learns the request is from Alice
 - Q: how does the app. know Alice was in the Inbox?
 - A: Remember the ":1"? It's the state number!
 - In eWebMail, each state was assigned a number
 - The state # is passed between client & server

21  **HTTP & Session Tracking**

- ⊙ State: Inbox (2) (cont)
 - 1=Inbox, 2=Read Email, 3=Compose, etc.
 - The app. sees the :1, and knows how to transition, to state 3 in this case
 - App. internally sets the state to 3
 - Sends response containing the Compose form and a cookie post-fixed with :3

22 









23  **HTTP & Session Tracking**

- ⊙ State: Compose (3)
 - When finished Alice clicks "Send"
 - HTTP request is sent containing the cookie
 - App. transitions from state 3 to state 1
 - Cookie has a :1 again

24 

Stateless vs. Stateful Apps.

- ⊙ So, cookies were used to retain state information as the app executed
- ⊙ Is this a *truly stateful* app.?

- Here, truly stateful means that the app. keeps track of sessions and states independently
- ⊙ To be truly stateful, the session tracking should be performed server-side
- 25  **Stateless vs. Stateful Apps.**
 - ⊙ But here, the entire burden of the session tracking is handed over to the client
 - Cookies are an easy way of passing info back and forth
 - Client-side session tracking is easy for the app!
 - In this fashion, app programmers tend to forget the golden rule: thou shalt not trust data coming from the client
 - A cookie is data coming from the client!
 - Bob tampered with the cookie thus tampering with the app input
- 26 
- 27  **Implementing Session & State Tracking**
 - ⊙ Session hijacking is possible mainly because the session and state tracking is entirely on the client
 - ⊙ Unfortunately, even if session and state tracking done server-side, the ids can be spoofed with the same results
 - ⊙ Predictable session ids also lead to session hijacking
 - ⊙ So... what to do?
- 28  **Implementing Session & State Tracking**
 - ⊙ Following is an incomplete list of "rules"
 1. Session ids should be unique
 - A logical session must be established between client and server app
 - The session id is composed of a string or number of pieces of data
 - All session ids must be unique
 - Do not reuse even for a return user
- 29  **Implementing Session & State Tracking**
 - ⊙ Following is an incomplete list of "rules" (cont)
 2. Session ids should not be "guessable"
 - Serial incrementing or time stamp ids are cause for concern
 - Can be guessed by rapidly generating user sessions
 - Fix? Use a random number + current time stamp + secret number to generate a hash
- 30  **Implementing Session & State Tracking**
 - ⊙ Following is an incomplete list of "rules" (cont)
 3. Session ids should be independent
 - Do not derive ids from usernames, passwords or app states
 - Use a lookup table server-side to match session id with user credentials
- 31  **Implementing Session & State Tracking**
 - ⊙ Following is an incomplete list of "rules" (cont)
 4. Session ids should be mapped with client-side connections
 - Keep track of the clients IP address and time of session creation
 - This will help prevent sniffing and reusing of session ids by an attacker
 - Every time a request is received from the client, compare the current client info to the stored info
- 32  **Summary**
 - ⊙ Session hijacking is a bit more difficult to perform
 - ⊙ But, attacks are just as serious!
 - ⊙ Session hijacking attacks are purely an app development issue
 - ⊙ Oversights in the server apps development or implementation of session tracking are to blame
 - ⊙ No Op Sys, patch, firewall or server configuration can stop session hijacking