Client-State Manipulation

Slides adapted from "Foundations of Security: What Every Programmer Needs To Know" by Neil Daswani, Christoph Kern, and Anita Kesavan (ISBN 1590597842; http://www.foundationsofsecurity.com). Except as otherwise noted, the content of this presentation is licensed under the Creative Commons 3.0 License.

Overview

- *Web application* – collection of programs used by server to reply to client (browser) requests
  - □ Often accept user input: don't trust, validate!
- HTTP is *stateless*, servers don't keep state
  - □ To conduct transactions, web apps need state
  - □ State info may be sent to client who echoes it back in future requests
- Example Exploit: "Hidden" parameters in HTML are not really hidden, can be manipulated

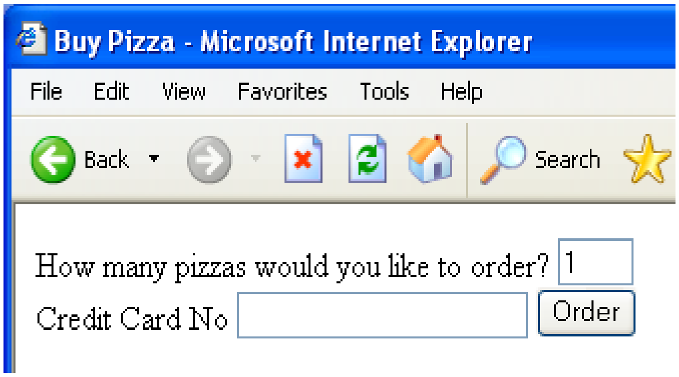
WARNING

- DO NOT TRY ANY OF THE ATTACKS DISCUSSED IN THIS COURSE ON REAL, PRODUCTION WEB SITES!!!

7.1. Pizza Delivery Web Site Example

- Web app for delivering pizza
  - Online order form: `order.html` – say user buys one pizza @ $5.50
  - Confirmation form: generated by `confirm_order` script, asks user to verify purchase, price is sent as hidden form field
  - Fulfillment: `submit_order` script handles user's order received as GET request from confirmation form (`pay` & `price` variables embedded as parameters in URL)

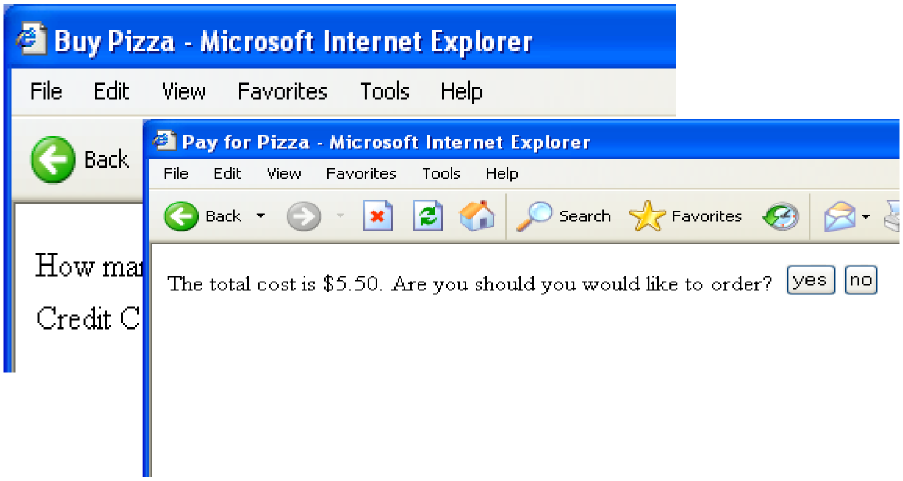
order.html → confirm_order


Buy Pizza - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Order

How many pizzas would you like to order? 1

Credit Card No

order.html → confirm_order


Buy Pizza - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
How ma
Credit C
Pay for Pizza - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
yes
no

The total cost is $5.50. Are you should you would like to order?

7.1. Pizza Web Site Code

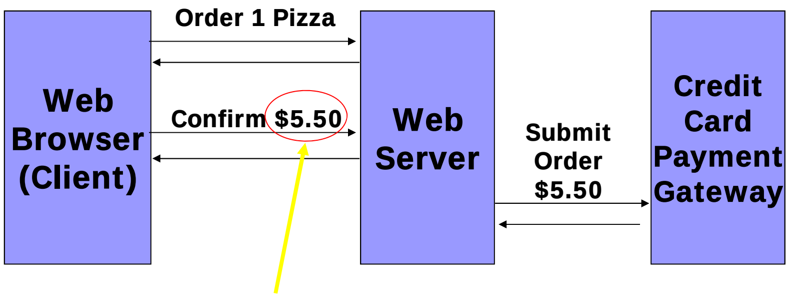
- Confirmation Form:

```
<HTML><head><title>Pay for Pizza</title></head>
<body><form action="submit_order" method="GET">
<p> The total cost is 5.50. Are you sure you
would like to order? </p>
<input type="hidden" name="price" value="5.50">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">
</form></body></HTML>
```

- Submit Order Script:

```
if (pay = yes) {
  success = authorize_credit_card_charge(price);
  if (success) {
    settle_transaction(price);
    dispatch_delivery_person();
  } else { // Could not authorize card
    tell_user_card_declined();
  }
} else { display_transaction_cancelled_page(); // no}
```

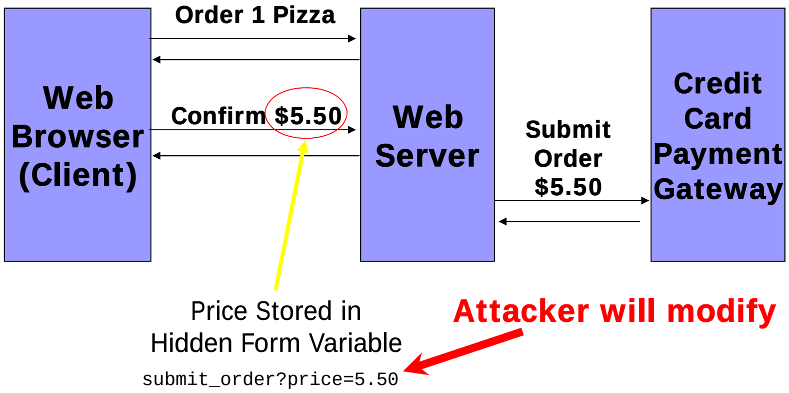
7.1. Buying Pizza Example



Price Stored in
Hidden Form Variable

`submit_order?price=5.50`

7.1. Buying Pizza Example



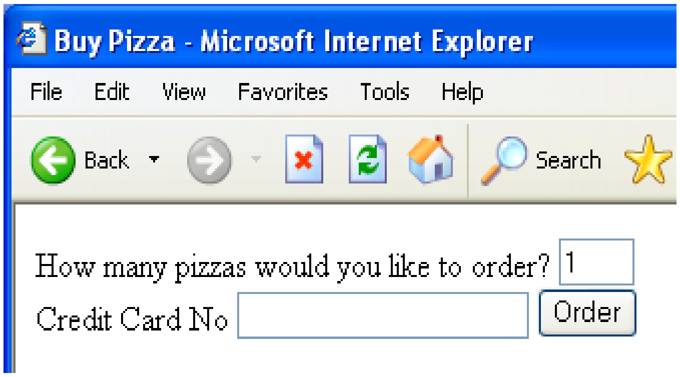
Price Stored in Hidden Form Variable

`submit_order?price=5.50`

Attacker will modify

7.1.1. Attack Scenario (1)

- Attacker navigates to order form…

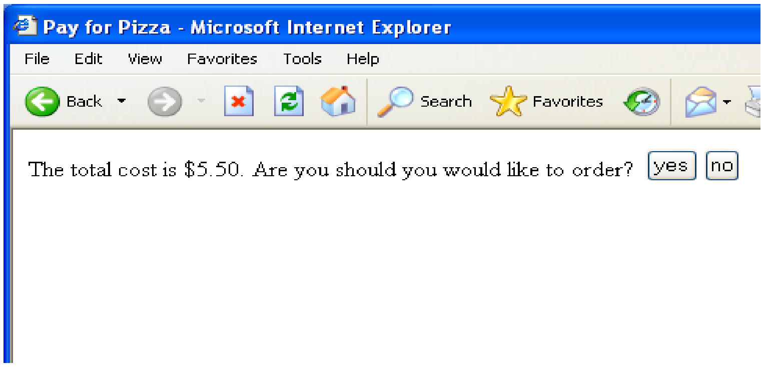


How many pizzas would you like to order? 1

Credit Card No

Order

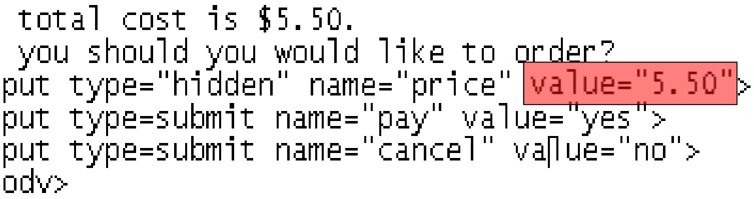
7.1.1. Attack Scenario (2)



The total cost is $5.50. Are you should you would like to order? yes no

7.1.1. Attack Scenario (3)

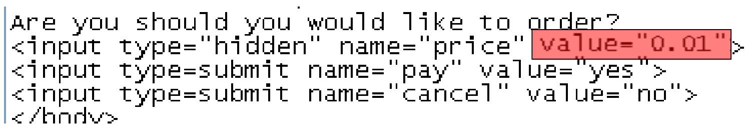
- And he can View | Source:



```
 total cost is $5.50.
 you should you would like to order?
put type="hidden" name="price" value="5.50">
put type=submit name="pay" value="yes">
put type=submit name="cancel" value="no">
odv>
```

7.1.1. Attack Scenario (4)

- Changes price in source, reloads page!

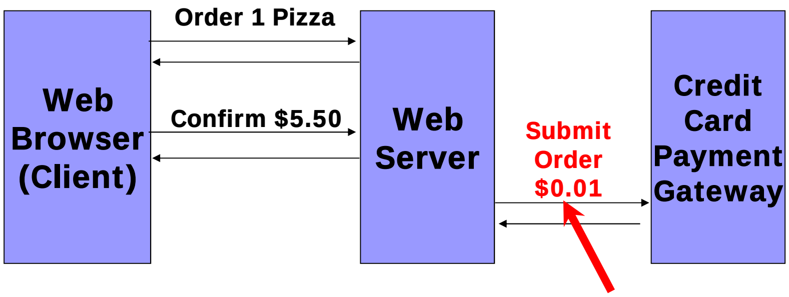


```
Are you should you would like to order?
<input type="hidden" name="price" value="0.01">
<input type=submit name="pay" value="yes">
<input type=submit name="cancel" value="no">
</body>
```

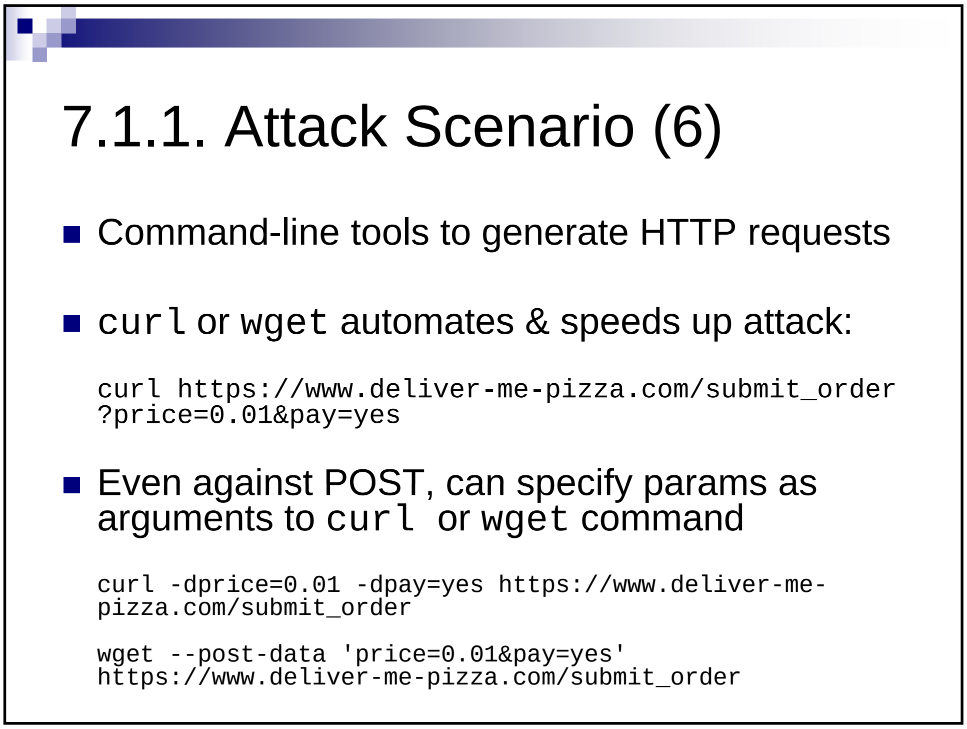
- Browser sends request:
  `GET /submit_order?price=0.01&pay=yes HTTP/1.1`

- Hidden form variables are essentially in clear

7.1.1. Attack Scenario (5)



Attacker modified Price!



7.1.1. Attack Scenario (6)

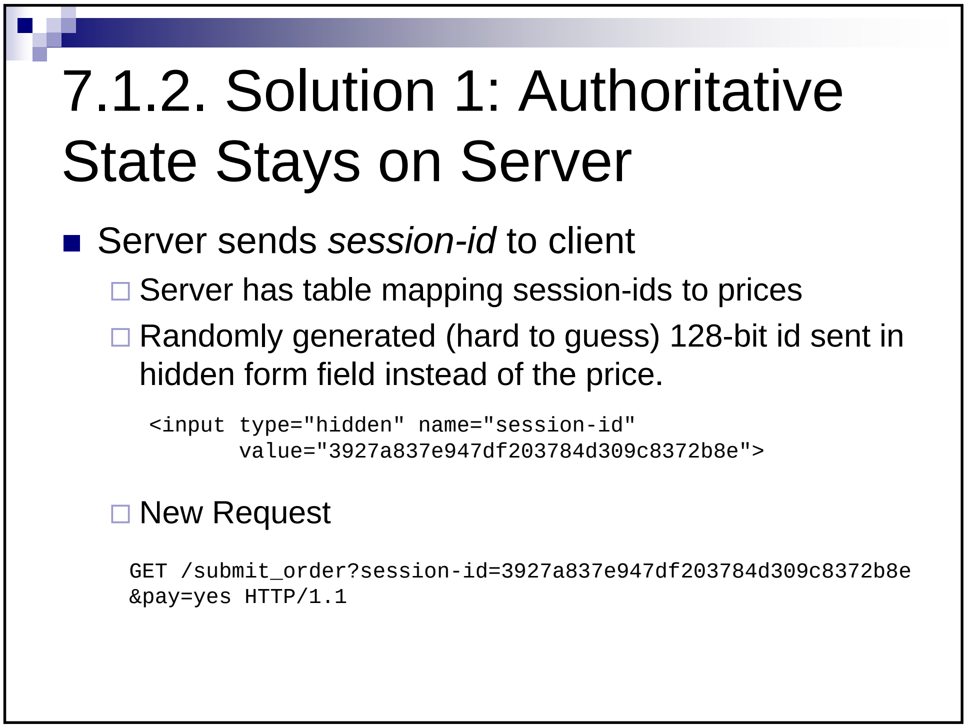
- Command-line tools to generate HTTP requests
- `curl` or `wget` automates & speeds up attack:

```
curl https://www.deliver-me-pizza.com/submit_order
?price=0.01&pay=yes
```

- Even against POST, can specify params as arguments to `curl` or `wget` command

```
curl -dprice=0.01 -dpay=yes https://www.deliver-me-
pizza.com/submit_order

wget --post-data 'price=0.01&pay=yes'
https://www.deliver-me-pizza.com/submit_order
```



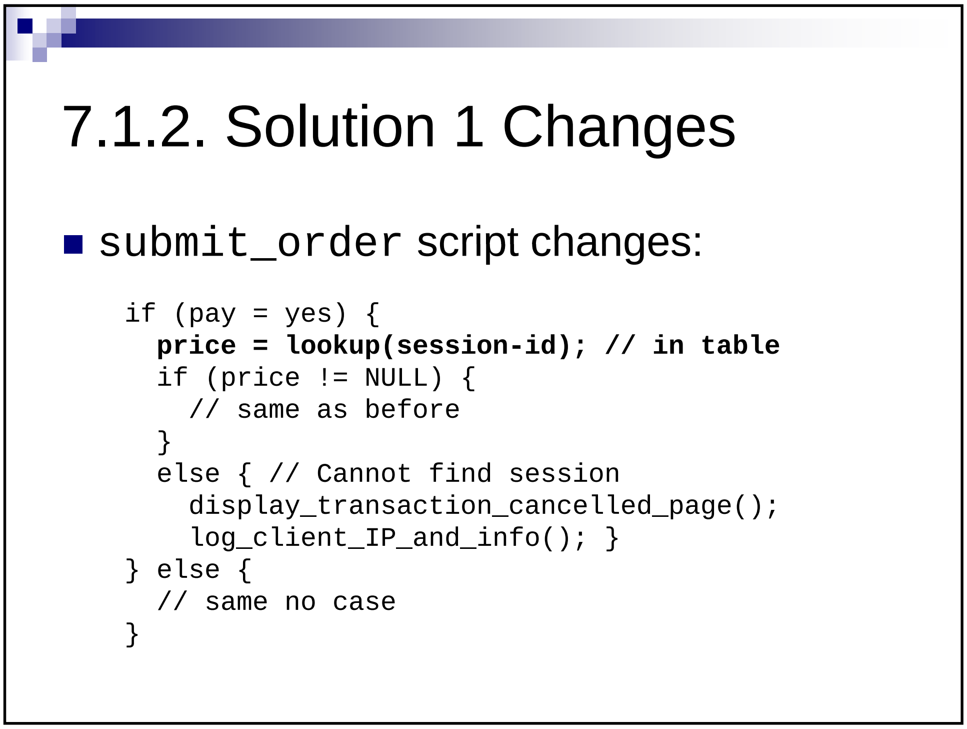
7.1.2. Solution 1: Authoritative State Stays on Server

- Server sends *session-id* to client
  - □ Server has table mapping session-ids to prices
  - □ Randomly generated (hard to guess) 128-bit id sent in hidden form field instead of the price.

```
<input type="hidden" name="session-id"
       value="3927a837e947df203784d309c8372b8e">
```

  - □ New Request

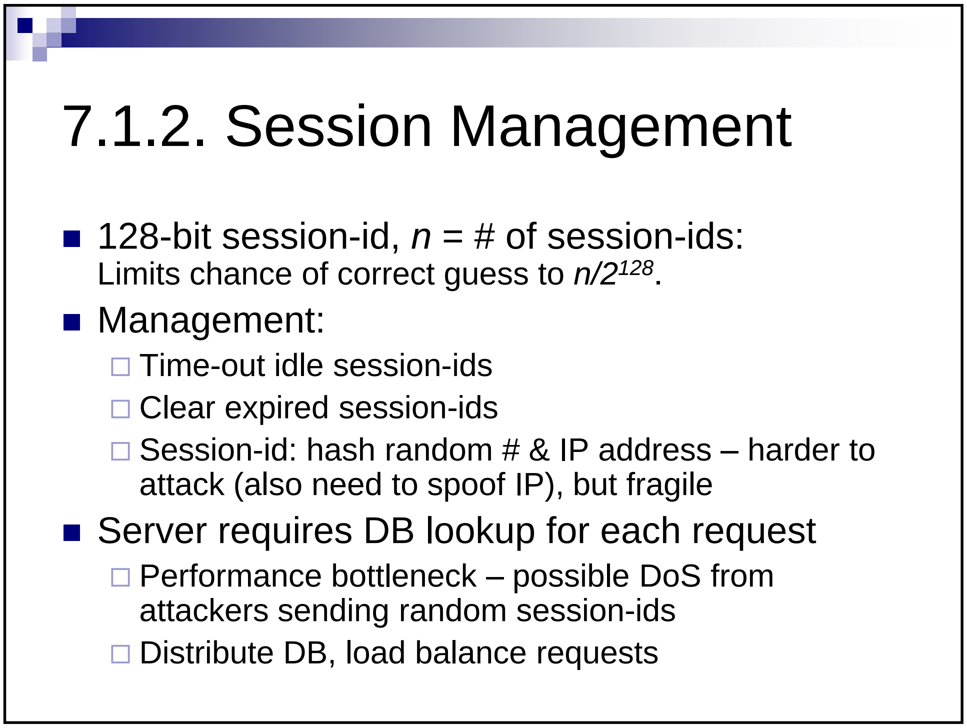
```
GET /submit_order?session-id=3927a837e947df203784d309c8372b8e
&pay=yes HTTP/1.1
```



7.1.2. Solution 1 Changes

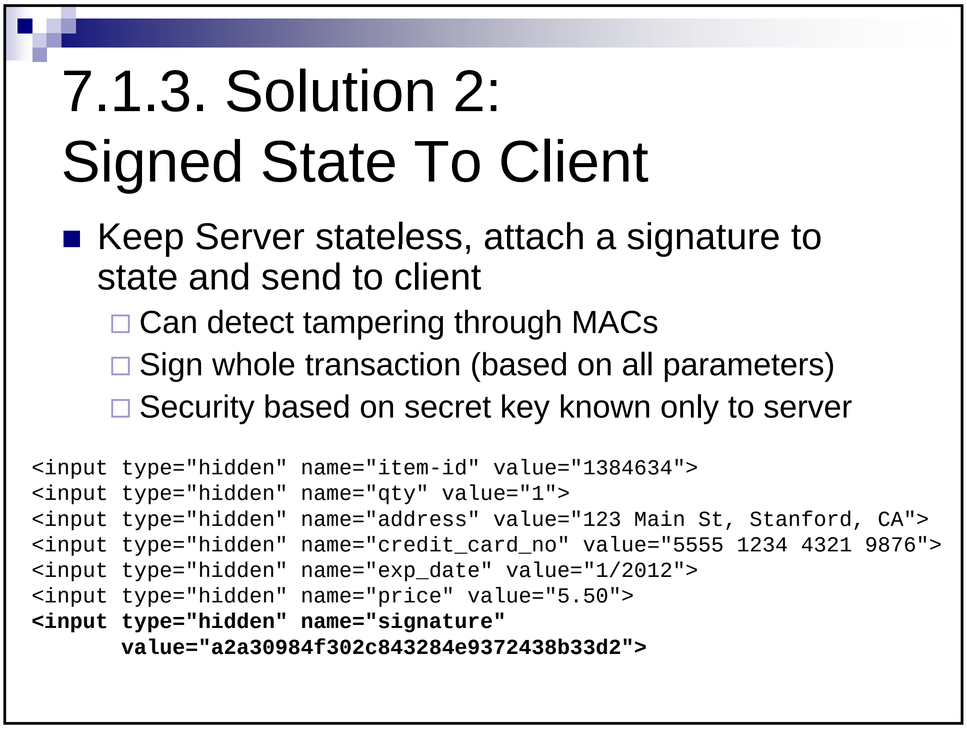
- `submit_order` script changes:

```
if (pay = yes) {
  price = lookup(session-id); // in table
  if (price != NULL) {
    // same as before
  }
  else { // Cannot find session
    display_transaction_cancelled_page();
    log_client_IP_and_info(); }
} else {
  // same no case
}
```



7.1.2. Session Management

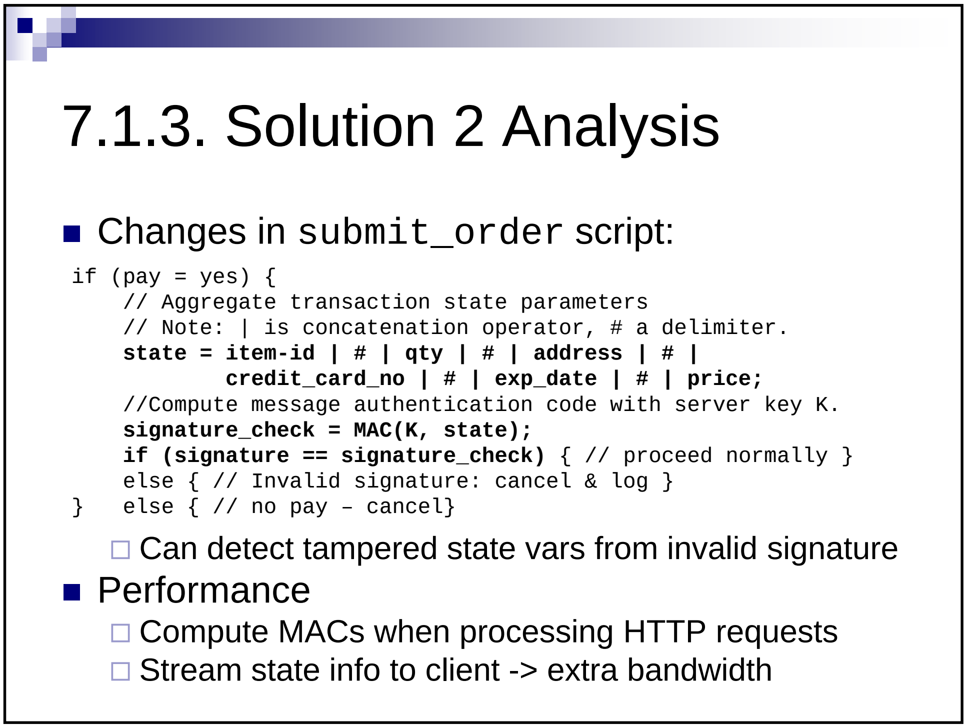
- 128-bit session-id, *n* = # of session-ids:
  Limits chance of correct guess to $n/2^{128}$.
- Management:
  - □ Time-out idle session-ids
  - □ Clear expired session-ids
  - □ Session-id: hash random # & IP address – harder to attack (also need to spoof IP), but fragile
- Server requires DB lookup for each request
  - □ Performance bottleneck – possible DoS from attackers sending random session-ids
  - □ Distribute DB, load balance requests



7.1.3. Solution 2: Signed State To Client

- Keep Server stateless, attach a signature to state and send to client
  - □ Can detect tampering through MACs
  - □ Sign whole transaction (based on all parameters)
  - □ Security based on secret key known only to server

```
<input type="hidden" name="item-id" value="1384634">
<input type="hidden" name="qty" value="1">
<input type="hidden" name="address" value="123 Main St, Stanford, CA">
<input type="hidden" name="credit_card_no" value="5555 1234 4321 9876">
<input type="hidden" name="exp_date" value="1/2012">
<input type="hidden" name="price" value="5.50">
<input type="hidden" name="signature"
       value="a2a30984f302c843284e9372438b33d2">
```



7.1.3. Solution 2 Analysis

- Changes in `submit_order` script:

```
if (pay = yes) {
    // Aggregate transaction state parameters
    // Note: | is concatenation operator, # a delimiter.
    state = item-id | # | qty | # | address | # |
            credit_card_no | # | exp_date | # | price;
    //Compute message authentication code with server key K.
    signature_check = MAC(K, state);
    if (signature == signature_check) { // proceed normally }
    else { // Invalid signature: cancel & log }
}   else { // no pay – cancel}
```

  - □ Can detect tampered state vars from invalid signature
- Performance
  - □ Compute MACs when processing HTTP requests
  - □ Stream state info to client -> extra bandwidth

7.2. Information Leakage

- GET: form params (e.g. session-id) leak in URL
  - □ Could anchor these links in lieu of hidden form fields
  - □ Alice sends Meg URL in e-mail, Meg follows it & continues transaction w/o Alice's consent
- `Referers` can leak through outlinks:
  - □ This `<a href="http://www.grocery-store-site.com/">` link
  - □ Sends request:

```
GET / HTTP/1.1 Referer:
https://www.deliver-me-pizza.com/submit_order?
session-id=3927a837e947df203784d309c8372b8e
```

  - □ Session-id leaked to `grocery-store-site`'s logs!

7.2. Benefits of POST

- `Referers` can still leak w/o user interaction
  - □ Instead of link, image: `<a href=http://www.grocery-store-site.com/banner.gif>`
  - □ GET request for `banner.gif` still leaks session-id
- POST Request:

```
POST /submit_order HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 45

session-id%3D3927a837e947df203784d309c8372b8e
```

  - □ Session-id not visible in URL
  - □ Pasting into e-mail wouldn't leak it
  - □ Slightly inconvenient for user, but more secure

7.3. Cookies

- *Cookie* - piece of state maintained by client
  - □ Server gives cookie to client
  - □ Client returns cookie to server in HTTP requests
  - □ Ex: session-id in cookie in lieu of hidden form field

```
HTTP/1.1 200 OK
Set-Cookie: session-id=3927a837e947df203784d309c8372b8e; secure
```

  - □ Secure dictates using SSL
  - □ Browser Replies:

```
GET /submit_order?pay=yes HTTP/1.1
Cookie: session-id=3927a837e947df203784d309c8372b8e
```

7.3. Problems with Cookies

- Cookies are associated with browser
  - □ Sent back w/ each request, no hidden field to tack on
- If user doesn’t log out, attacker can use same browser to impersonate user
- Session-ids should have limited lifetime

7.4. JavaScript (1)

- Popular client-side scripting language
- Ex: Compute prices of an order:

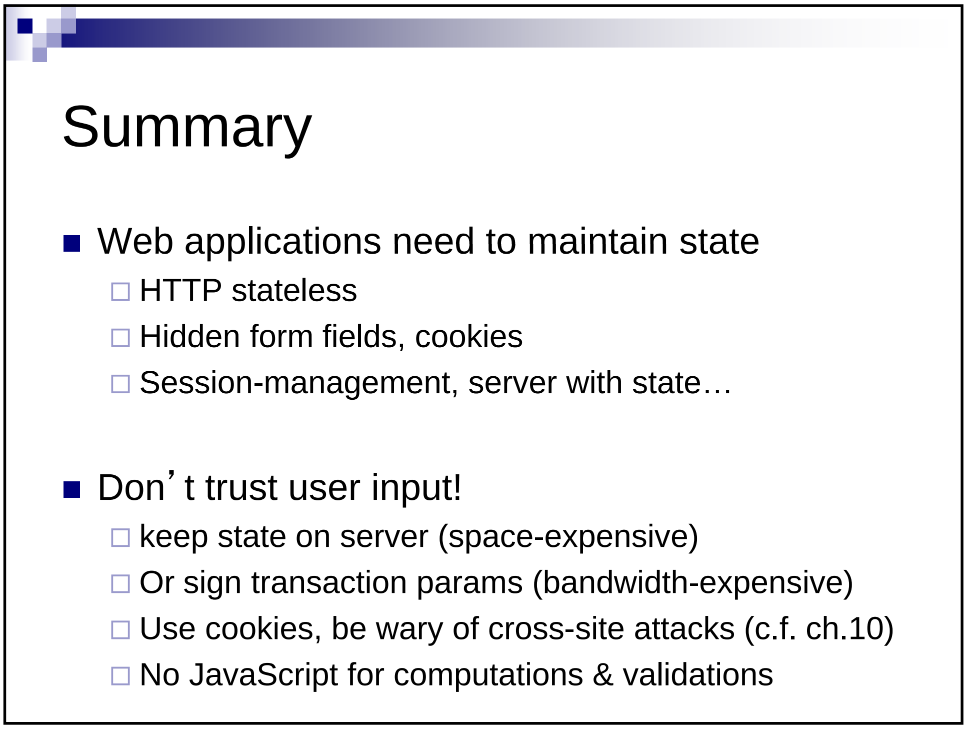
```
<html><head><title>Order Pizza</title></head><body>
 <form action="submit_order" method="GET" name="f">
   How many pizzas would you like to order?
   <input type="text" name="qty" value="1" onKeyUp="computePrice();">
   <input type="hidden" name="price" value="5.50"><br>
   <input type="submit" name="Order" value="Pay">
   <input type="submit" name="Cancel" value="Cancel">
   <script>
     function computePrice() {
       f.price.value = 5.50 * f.qty.value; // compute new value
       f.Order.value = "Pay " + f.price.value // update price
     }
   </script>
</body></html>
```

7.4. JavaScript (2)

- Evil user can just delete JavaScript code, substitute desired parameters & submit!
  - □ Could also just submit request & bypass JavaScript

```
GET /submit_order?qty=1000&price=0&Order=Pay
```

- **Warning**: data validation or computations done by JavaScript cannot be trusted by server
  - □ Attacker may alter script in HTML code to modify computations
  - □ Must be redone on server to verify



Summary

- Web applications need to maintain state
  - □ HTTP stateless
  - □ Hidden form fields, cookies
  - □ Session-management, server with state…

- Don’t trust user input!
  - □ keep state on server (space-expensive)
  - □ Or sign transaction params (bandwidth-expensive)
  - □ Use cookies, be wary of cross-site attacks (c.f. ch.10)
  - □ No JavaScript for computations & validations