Web Security
Section 5
XSS

• Idea: Place User provided data in the page
• Pros: makes pages more interactive and personal
• Cons: improperly used data could be interpreted as code
• Solutions: Make sure that user data is sanitized and validated with a whitelist approach
XSSI

- Idea: Some script content can be loaded in the context of the current domain, against the usual mechanism of the same domain policy
- Pros: Allows reuse of scripted code under the context of the current domain, allows for javascript services
- Cons: Attackers can provide malicious scripts or invoke services in the current domain's context
- Solution: Make sure the code comes from a trusted site
XSRF

- Idea: Have web services and accounts whose data can be changed via the web
- Pros: A website can be customized and made more user friendly
- Cons: Someone can make these calls for you with undesired results
- Solutions: inspect header, require user-provided secret, add nonce token to forms and verify legitimate requests
HTTP State

• HTTP is a stateless protocol
• This means the state machine for the protocol is very simplistic (request and response)
• However developers want state in order to build staged user experiences
• Solution: provide state to user and have them echo it back in future requests
Attempt 1: Hidden Fields

Let's give the user hidden fields that will hold state variables for us to use on later requests.

```html
<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>
```
Attempt 1: Problems

Web Browser (Client) → Order 1 Pizza → Confirm $5.50 → Web Server

Web Server → Submit Order $5.50 → Credit Card Payment Gateway

Price Stored in Hidden Form Variable: submit_order?price=5.50

Attacker will modify
Attempt 1: Problems

Web Browser (Client) → Web Server
- Order 1 Pizza
- Confirm $5.50

Web Server → Credit Card Payment Gateway
- Submit Order $0.01

Credit Card Payment Gateway ← Web Server

Attacker modified Price!
Attempt 2: Cookies

- Let's store state on the client side, but in a special file that can only be accessed by code from the same domain

```javascript
<script type="text/javascript">
    document.cookie = "price=$5.00";
</script>
```
Attempt 2: Problems

- Cookies can still be sniffed from HTTP requests
- Cookies can still be stolen from injected scripts
- So not much improvement except that the parameters are not directly visible in Get requests
Attempt 3: Sessions

- Let's store state on the server side and only give the user an identifier for it.
- If we place this identifier in a cookie it will be harder to gather.
- If we make the session id a hash of the user's ip address and a nonce it will be harder to spoof.

```php
<?php
    session_start();
    $_SESSION['price'] = 5.0;
?>
```
Attempt 3: Problems

- All user state is stored server side, this can be a ton of data
- Search for user session data can be a bottleneck in the response time
- Sessions need to expire or else sessions can be reused by attackers
- Placing the session id in cookies does not eliminate XSRF attacks