CSC 405
Computer Security

Web Security

Alexandros Kapravelos
akaprav@ncsu.edu

(Derived from slides by Giovanni Vigna and Adam Doupe)

HTML Frames

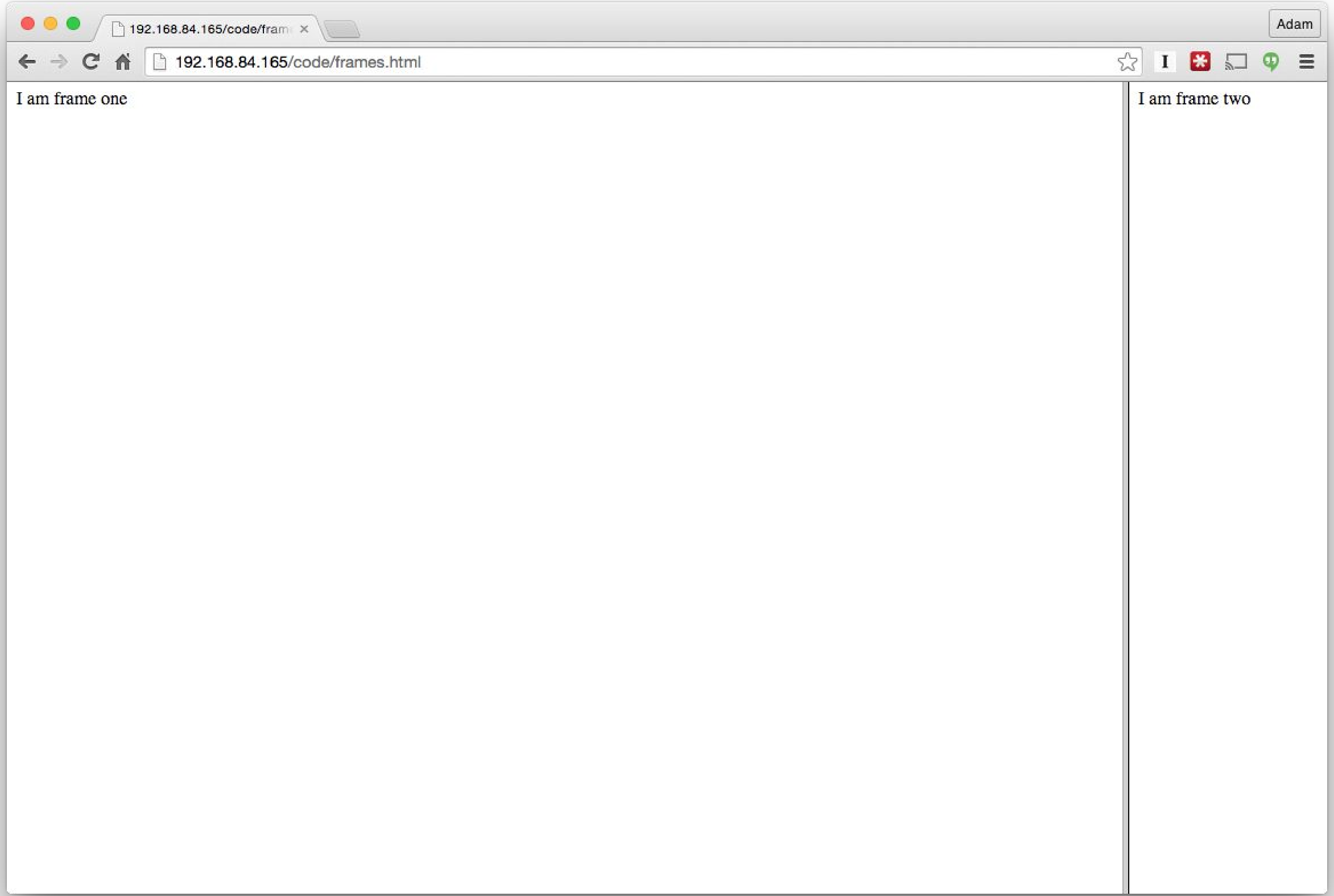
- Ability to tie multiple separate URLs together on one page
- Used in the early days to provide a banner or navigation element

frameset

```
<frameset cols="85%, 15%">
 <frame src="frame1.html" name="frame_1">
 <frame src="frame2.html" name="frame_2">
 <noframes>
   Text to be displayed in browsers that do not support frames
 </noframes>
</frameset>
```

The Frames

- frame1.html
  - I am frame 1
- frame2.html
  - I am frame two



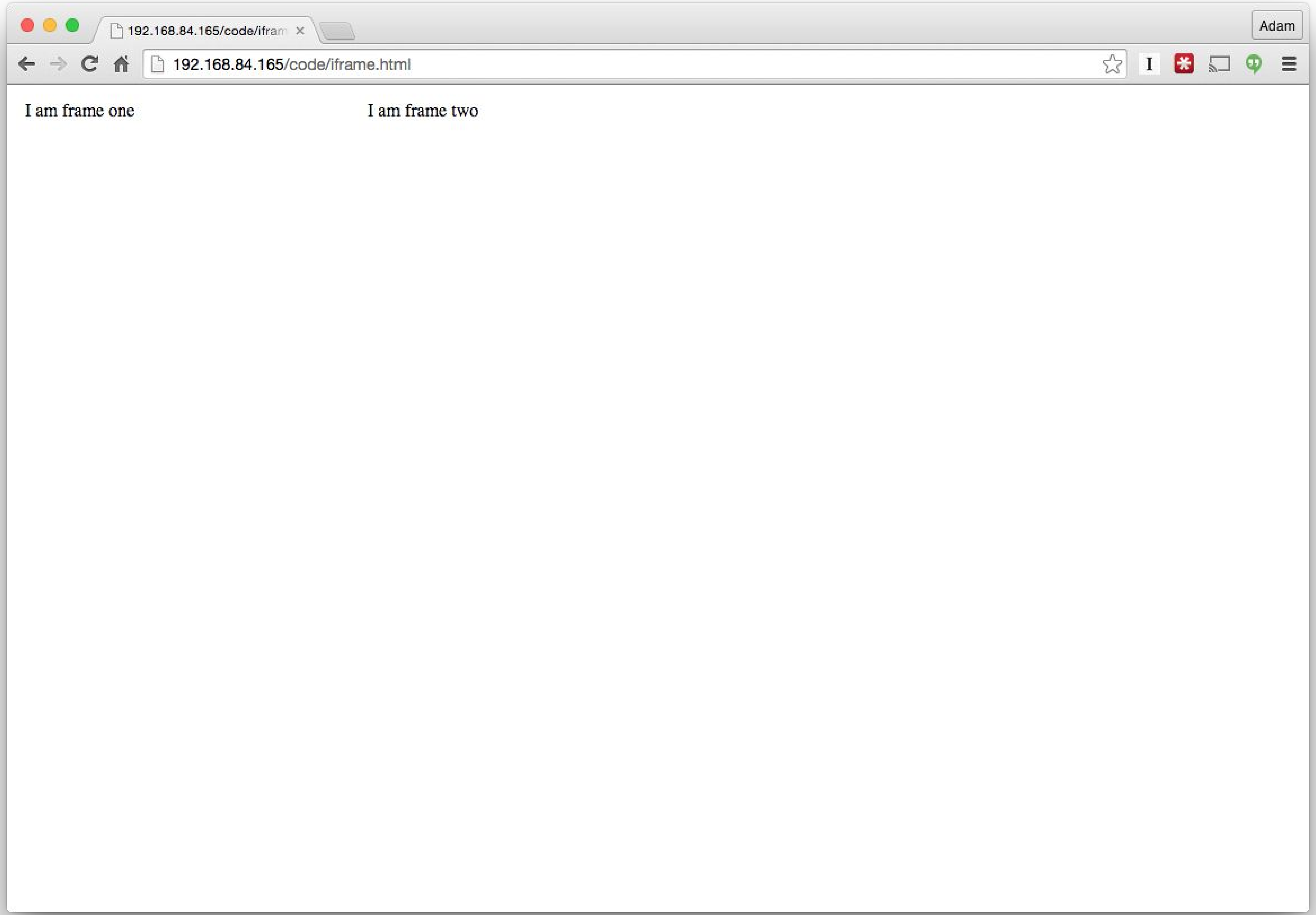
I am frame one

I am frame two

iframes

- Inline frames
- Similar to frames, but does not need a frameset

```
<iframe src="frame1.html" name="frame_1" frameBorder="0"></iframe>
<iframe src="frame2.html" name="frame_2" frameBorder="0"></iframe>
```



I am frame one

I am frame two

JavaScript Security

- Browsers are downloading and running foreign (JavaScript) code, sometimes concurrently
- The security of JavaScript code execution is guaranteed by a sandboxing mechanism (similar to what we saw in Java applets)
  - No access to local files
  - No access to (most) network resources
  - No incredibly small windows
  - No access to the browser's history
  - …
- The details of the sandbox depend on the browser

Same Origin Policy (SOP)

- Standard security policy for JavaScript across browsers
  - Incredibly important to web security
    - If you learn only one thing from this class, let it be the Same Origin Policy
- Every frame or tab in a browser's window is associated with a domain
  - A domain is determined by the tuple: <protocol, domain, port> from which the frame content was downloaded
- Code downloaded in a frame can only access the resources associated with that domain
- If a frame explicitly includes external code, this code will execute within the SOP
  - On example.com, the following JavaScript code has access to the <http,example.com, 80> SOP
  - `<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>`

SOP example

Original URL http://store.company.com/dir/page.html

- Which of the following belong to the SOP?

http://store.company.com/dir2/other.html	Success
http://store.company.com/dir/inner/another.html	Success
https://store.company.com/secure.html	Failure
http://store.company.com:81/dir/etc.html	Failure
http://news.company.com/dir/other.html	Failure

Storing Information

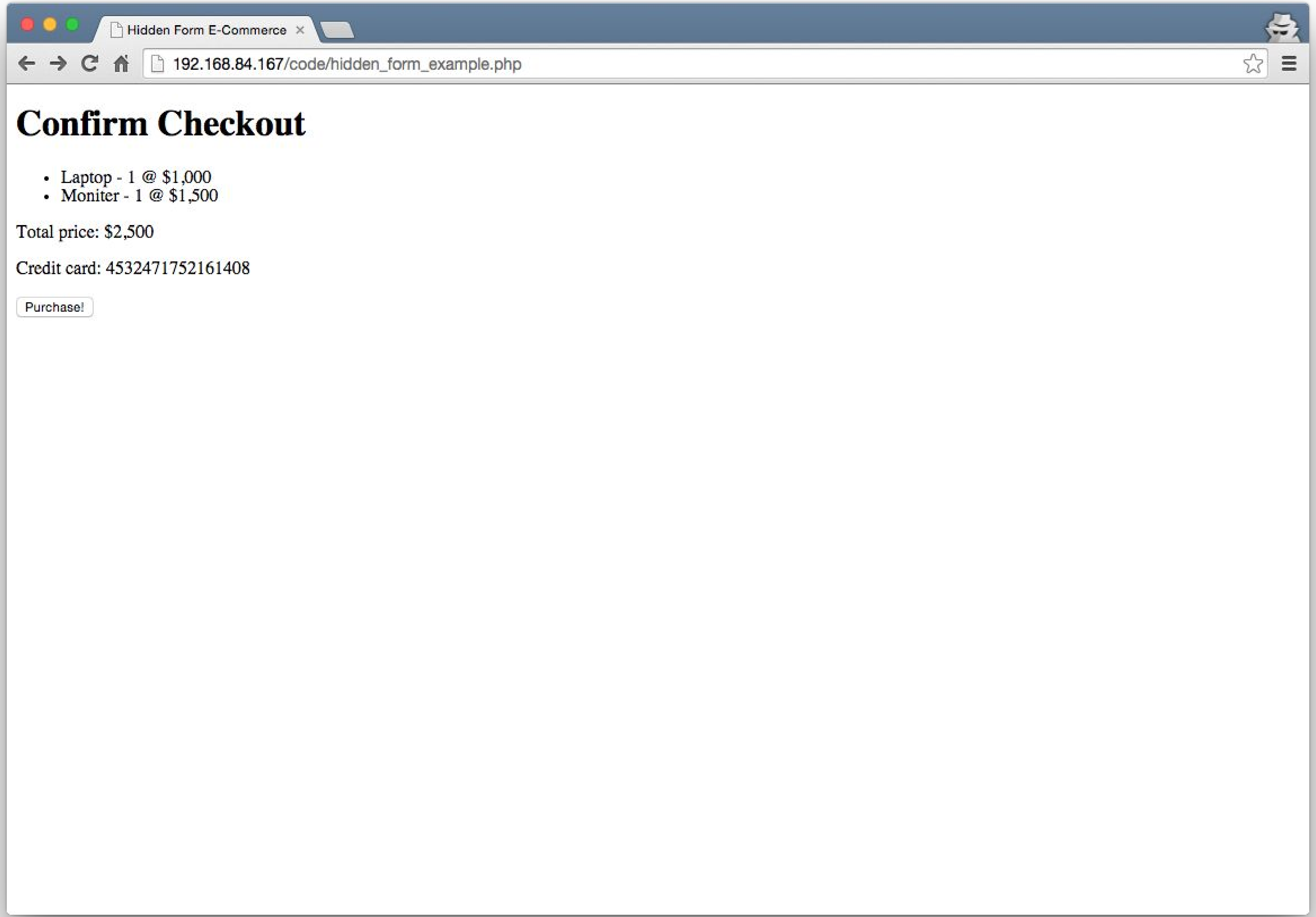
- As we've seen, information can be stored on the client's browser
  - Cookies
  - URLs
  - Forms
  - Plugin (Applets, Flash, Silverlight)
  - LocalStorage

Tampering with Client-Side Information

- Nothing prevents us from not tampering with client-side information
  - Tampering, by itself, is not a vulnerability
- The question is: how does the server-side code respond to our tampering?
  - If the server-side code allows our tampering **and** that tampering compromises the security of the application, then there is a vulnerability

Hidden Form Fields

- As we saw when studying web applications, an HTML input element with the type attribute of hidden will not be shown in the browser
- Many legitimate uses for this behavior
  - CAPTCHA
  - CSRF protection
- The problem is when the server-side code blindly trusts the data that is placed in the hidden form



Confirm Checkout

- Laptop - 1 @ $1,000
- Moniter - 1 @ $1,500

Total price: $2,500

Credit card: 4532471752161408

Purchase!

```html
<!DOCTYPE html>
<html>
 <head>
   <meta charset="UTF-8">
   <title>Hidden Form E-Commerce</title>
 </head>

 <body>
  <h1>Confirm Checkout</h1>
  <ul>
    <li>Laptop - 1 @ $1,000</li>
    <li>Moniter - 1 @ $1,500</li>
  </ul>
  <p>
    Total price: $2,500
  </p>
  <p>
    Credit card: 4532471752161408
  </p>

  <form action="purchase.php" method="POST">
    <input type="submit" value="Purchase!">
    <input type="hidden" name="oid" value="5929">
      <input type="hidden" name="price" value="2500">
      <input type="hidden" name="cur" value="usd">
  </form>
 </body>
</html>
```

How to approach

- What possible hidden values are there to test?
- What might they mean?
- What would be malicious versions of those values?
- How to test the hypothesis?

Let's hack it!

Hacking

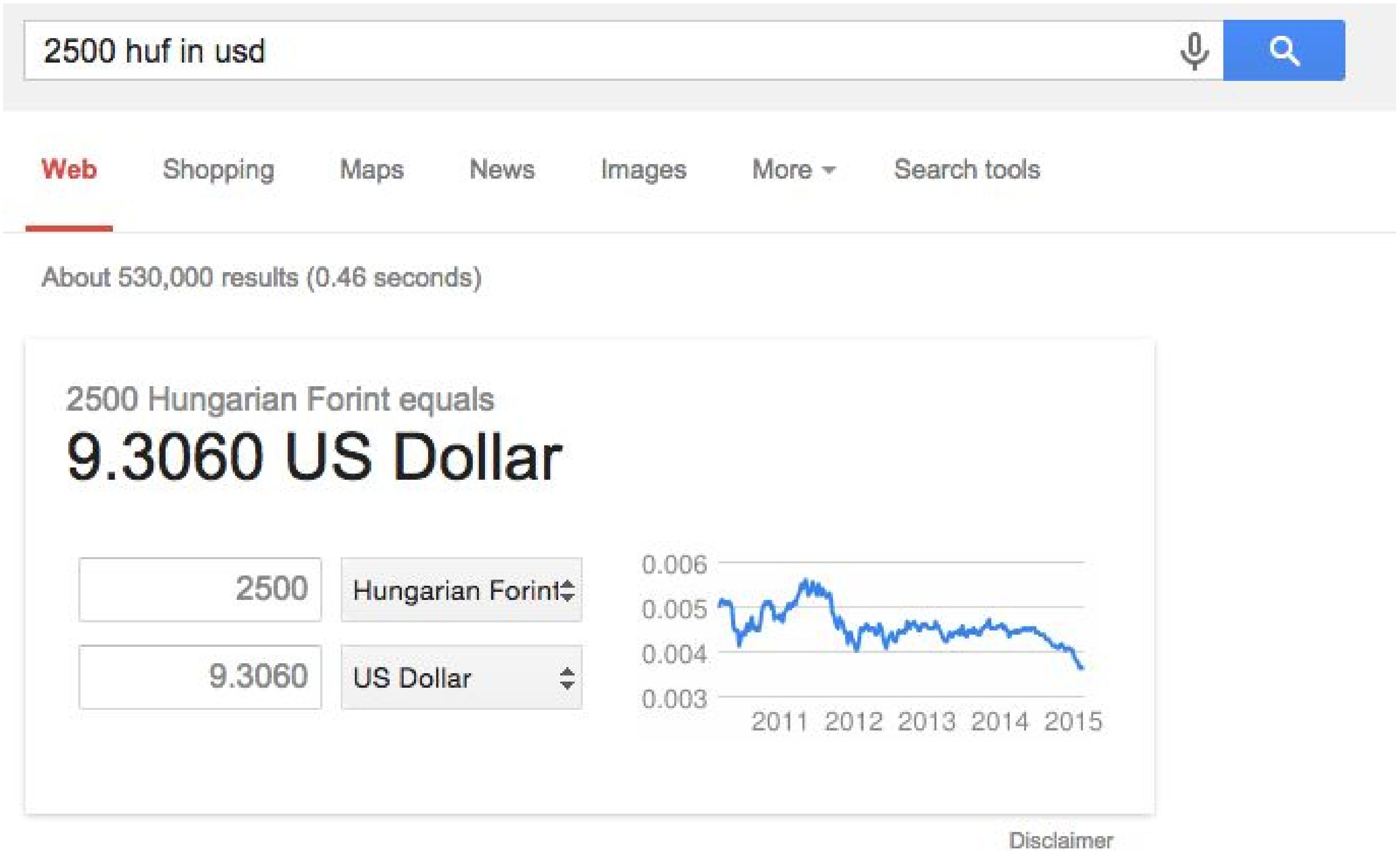
- All that's needed is a browser and a command-line tool (I use curl)
- Using curl, we can create a request to the purchase page
  - `curl -F oid=5929 -F price=2500 -F cur=usd http://192.168.84.167/code/purchase.php`
  - `I don't know who you are, go away`
- What's the problem?
  - Not sending cookies, so must be a session issue

Hacking

- First, we need to find our cookie from our browser
- Then, we can use curl to include that cookie using the -b options
  - `curl -b PHPSESSID=n7591kbbse8rug4dfn019skv05 -F oid=5929 -F price=2500 -F cur=usd http://192.168.84.167/code/purchase.php`
  - `Purchase successful, your final order total is 2,500 usd charged to your CC XXXXXXXXXXXX1408`
- Hurray, we were able to make a successful order

Hacking

- What happens when we manipulate the values?
- What could oid stand for?
  - `curl -b PHPSESSID=n7591kbbse8rug4dfn019skv05 -F oid=1 -F price=2500 -F cur=usd http://192.168.84.167/code/purchase.php`
  - `FAIL, not your order!`
- What does price stand for?
  - `curl -b PHPSESSID=n7591kbbse8rug4dfn019skv05 -F oid=5929 -F price=1 -F cur=usd http://192.168.84.167/code/purchase.php`
  - `FAIL, not the correct price!`
- What does cur stand for?
  - `curl -b PHPSESSID=n7591kbbse8rug4dfn019skv05 -F oid=5929 -F price=2500 -F cur=huf http://192.168.84.167/code/purchase.php`
  - `Purchase successful, your final order total is 2,500 huf charged to your CC XXXXXXXXXXXX1408`



2500 huf in usd

About 530,000 results (0.46 seconds)

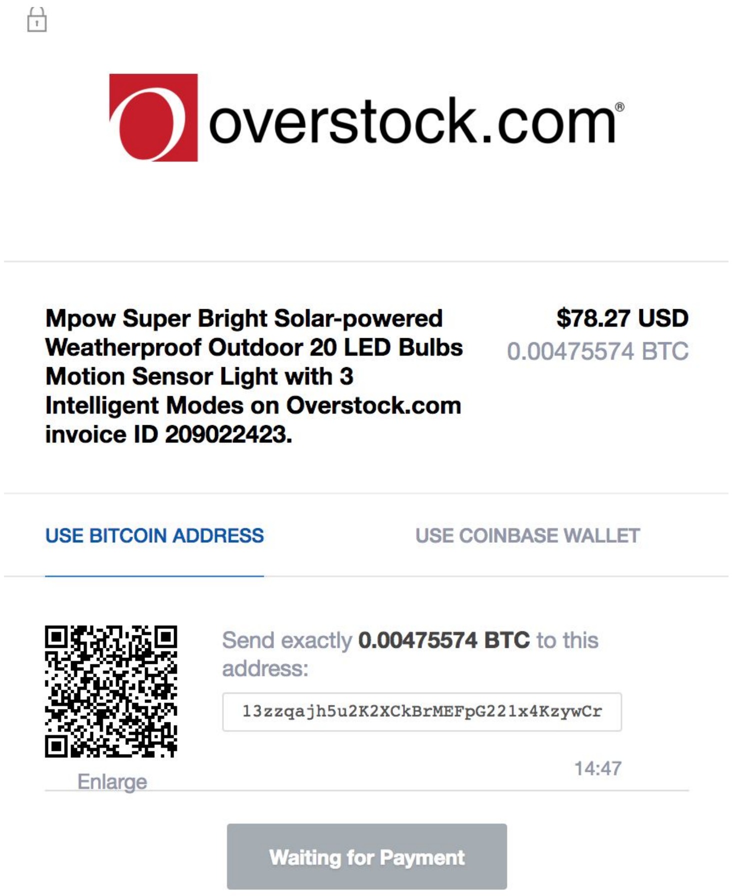
2500 Hungarian Forint equals

9.3060 US Dollar

2500	Hungarian Forint
9.3060	US Dollar

Your Security Zen (interrupt)

"Website Glitch Let Me Overstock My Coinbase"



overstock.com

Mpow Super Bright Solar-powered Weatherproof Outdoor 20 LED Bulbs Motion Sensor Light with 3 Intelligent Modes on Overstock.com invoice ID 209022423.

$78.27 USD
0.00475574 BTC

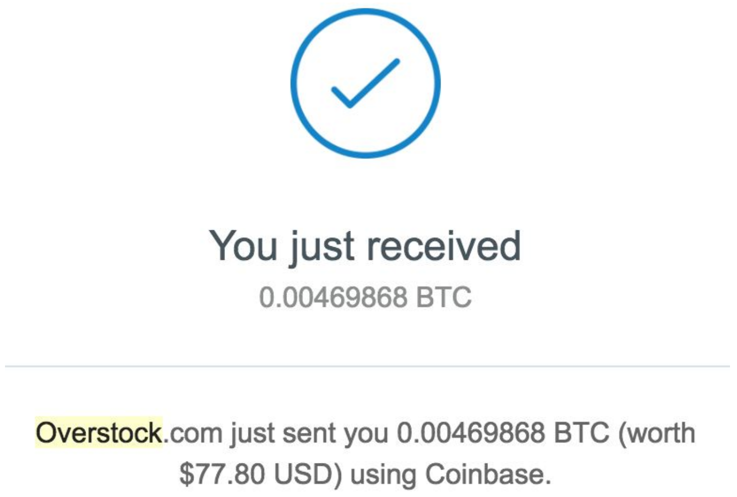
USE BITCOIN ADDRESS USE COINBASE WALLET

Send exactly **0.00475574 BTC** to this address:

13zzqajh5u2K2XCkBrMEFpG221x4KzywCr

Enlarge 14:47

Waiting for Payment



You just received
0.00469868 BTC

Overstock.com just sent you 0.00469868 BTC (worth $77.80 USD) using Coinbase.

- You could tell Coinbase to send 0.00475574 in bitcoin cash instead of bitcoin
- $78 purchase by sending approximately USD $12 worth of bitcoin cash
- the system refunded the purchase in bitcoin, not bitcoin cash!

source: https://krebsonsecurity.com/2018/01/website-glitch-let-me-overstock-my-coinbase/

HTTP Cookies

- As we have seen, cookies are used to store state on the browser
  - Server requests that the client store a bit of state on the browser
  - Cookie can be any arbitrary data
- Just as we saw in the previous example, we can manipulate cookies via curl or with browser extension

URL Parameters

- The query parameters of a URL could also be used as information
  - Perhaps the price is calculated from a query parameter
  - Why would a developer do this?
- Manipulating the query parameter could change the price
  - If the application accepts the new price

Referer Header

- The referer HTTP header is defined in the HTTP 1.0 RFC as
  - The Referer request-header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained. This allows a server to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistyped links to be traced for maintenance. The Referer field must not be sent if the Request-URI was obtained from a source that does not have its own URI, such as input from the user keyboard.
- The spelling was a typo and was not caught until people were already using referer
- Sent automatically by the browser when a link is clicked
- Can it be trusted?
  - Developers assume that because it is an HTTP header, it is trustworthy
  - What do you think?

Referer to Control Access

- The referer header is untrusted and can be manipulated
- Therefore, using a referer header to ensure that the user is visiting your application in the intended order is a mistake
- Using -H option of curl to set arbitrary HTTP headers on request

HTML Forms Input Restrictions

- Developer can specify HTML5 restrictions/validation on form input
  - required attribute
  - type=email
  - pattern attribute
- Custom validation using JavaScript
  - All can be bypassed

Definitions

- Authentication
  - Who is the user?
  - Breaking means impersonating another user
- Authorization
  - What is the user allowed to do?
    - Admin, regular, guest, …
  - Attacking means performing actions that you're not allowed to do
- Often intertwined
  - If you're able to break the authentication to log in as a different user, then you've also broken authorization

Attacking Authentication

- Eavesdropping credentials/authenticators
- Brute-forcing/guessing credentials/authenticators
- Bypassing authentication
  - SQL Injection (later)
  - Session fixation

Eavesdropping Credentials and Authenticators

- If the HTTP connection is not protected by SSL it is possible to eavesdrop the credentials:
  - Username and password sent as part of an HTTP basic authentication exchange
  - Username and password submitted through a form
  - The authenticator included as cookie, URL parameter, or hidden field in a form
- The "secure" flag on cookies is a good way to prevent accidental leaking of sensitive authentication information

Brute-forcing Credentials and Authenticators

- If authenticators have a limited value domain they can be brute-forced (e.g., 4-digit PIN)
  - Note: lockout policies might not be enforced in mobile web interfaces to accounts
- If authenticators are chosen in a non-random way they can be easily guessed
  - Sequential session IDs
  - User-specified passwords
  - Example: http://www.foo.bar/secret.php?id=BGH10110915103939 observed at 15:10 of November 9, 2010
- Long-lived authenticators make these attacks more likely to succeed

Bypassing Authentication

- Form-based authentication may be bypassed using carefully crafted arguments
- Authentication, in certain case can be bypassed using forceful browsing
- Weak password recovery procedures can be leveraged to reset a victim’s password to a known value
- Session fixation forces the user’s session ID to a known value
  - For example, by luring the user into clicking on a link such as: <a href=http://foo.com/vulnerable.php?SESSIONID=1234>foo</a>
- The ID can be a fixed value or could be obtained by the attacker through a previous interaction with the vulnerable system

HackPack Meetings

- 6:00-7:15 PM at 2220 EB3 on Wednesdays
- 4:10-6:15 PM at 2220 EB3 on Fridays

- https://ncsu-hackpack.slack.com/messages/general
- https://getinvolved.ncsu.edu/organization/HackPack/

- Get some practical experience in discovering and exploiting security problems by playing CTFs!