CSE 484 / CSE M 584: Computer Security and Privacy

# Web Security: Loose Ends

Spring 2017

Franziska (Franzi) Roesner
franzi@cs.washington.edu

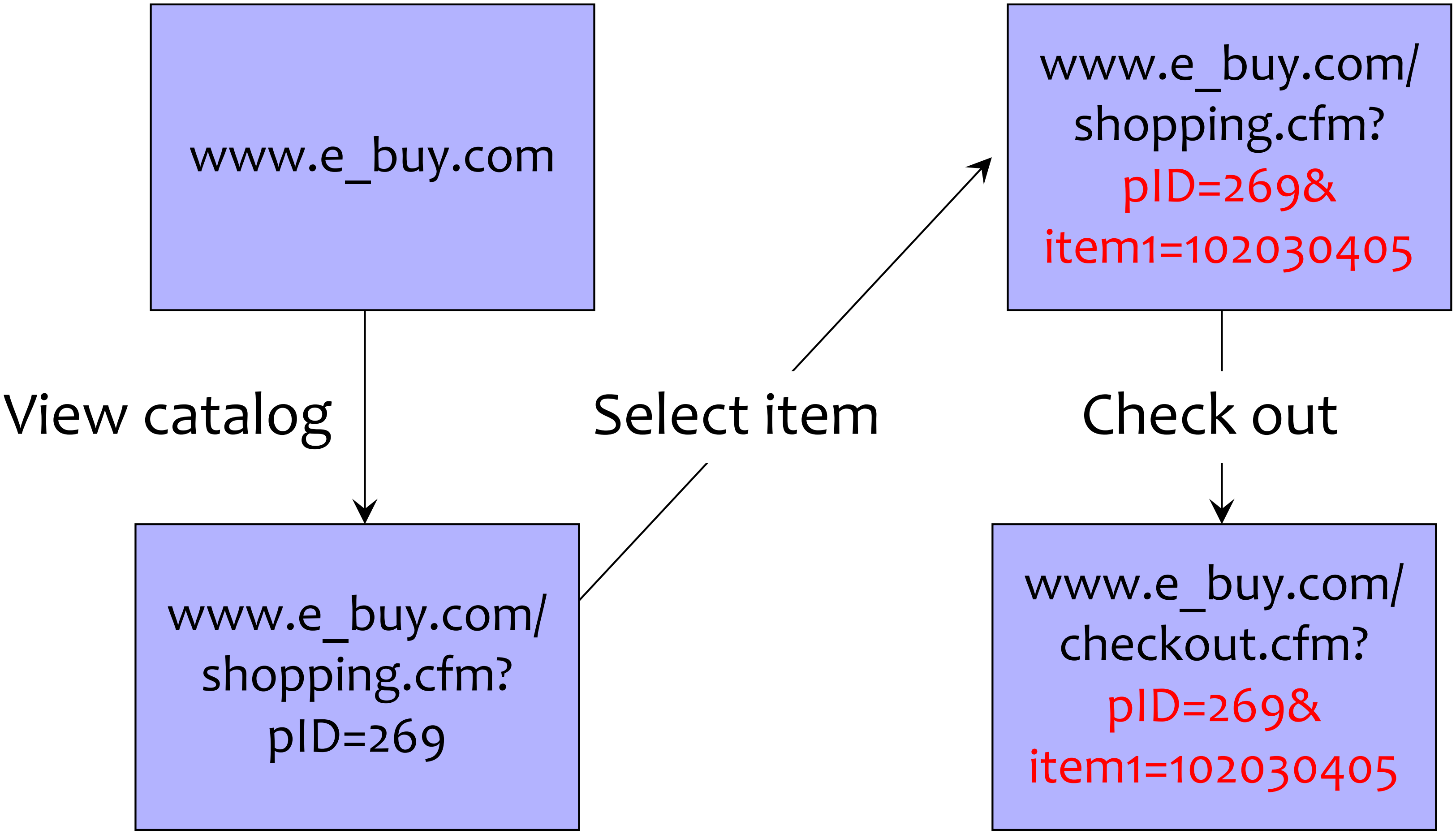
Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

# Admin

- Final project deadline #1 tonight
  - Upload to Catalyst a PDF file that contains
    - (1) your group members' names and UWNetIDs and
    - (2) a brief description of the topic of your presentation.
  - Sample presentations posted
- My office hours next week: 9:15am Wed (**not Fri**)

# Web Session Management

# 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

www.e_buy.com/ checkout.cfm? pID=269& item1=102030405

Store session information in URL; easily read on network

# Bad Idea: Encoding State in URL

- Unstable, frequently changing URLs
- Vulnerable to eavesdropping and modification
- There is no guarantee that URL is private

# 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 not 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

    https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555752

  - Partial fix: use random authenticators

# Typical Solution:
# Web Authentication via Cookies

- Servers can use cookies to store state on client
  - When session starts, server computes an authenticator and gives it back to browser in the form of a cookie
    - Authenticators must be **unforgeable** and **tamper-proof**
      - Malicious client shouldn't be able to compute his own or modify an existing authenticator
    - Example: MAC(server's secret key, session id)
  - With each request, browser presents the cookie
  - Server recomputes and verifies the authenticator
    - Server does not need to remember the authenticator

# Storing State in Hidden Forms

- 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."

```
<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="p
  <INPUT TYPE=HIDDEN NAME=custom1   VALUE="E
    with leather straps">

  <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">

</FORM>
```

Change this to 2.00

Bargain shopping!

Fix: MAC client-side data, or, more likely, keep on server.

# Top Web Vulnerabilities: Summary

- XSS (CSS) – cross-site scripting
  - Malicious code injected into a trusted context (e.g., malicious data presented by an honest website interpreted as code by the user's browser)
- SQL injection
  - Malicious data sent to a website is interpreted as code in a query to the website's back-end database
- XSRF (CSRF) – cross-site request forgery
  - Bad website forces the user's browser to send a request to a good website
- Broken authentication and session management

# Cross-Origin Communication?

- Websites can embed scripts, images, etc. from other origins.
- **But:** AJAX requests (aka XMLHttpRequests) are not allowed across origins.

On example.com:

```
<script>
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = handleStateChange; // Elsewhere
xhr.open("GET", “https://bank.com/account_info”, true);
xhr.send();
</script>
```

# Cross-Origin Communication?

- Websites can embed scripts, images, etc. from other origins.
- **But:** AJAX requests (aka XMLHttpRequests) are not allowed across origins.

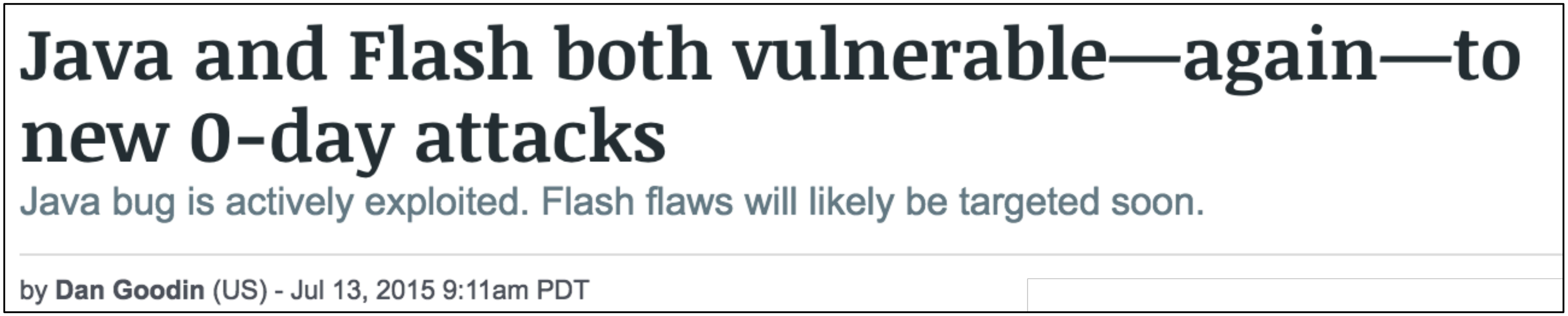
- Why not?
  - Browser automatically includes cookies with requests (i.e., user credentials are sent)
  - Caller can read returned data (clear SOP violation)

# Allowing Cross-Origin Communication

- Domain relaxation
  - If two frames each set document.domain to the same value, then they can communicate
    - E.g. www.facebook.com, facebook.com, and chat.facebook.com
    - Must be a suffix of the actual domain
- Access-Control-Allow-Origin: <list of domains>
  - Specifies one or more domains that may access DOM
  - Typical usage: Access-Control-Allow-Origin: *
- HTML5 postMessage
  - Lets frames send messages to each other in controlled fashion
  - Unfortunately, many bugs in how frames check sender's origin

# What about Browser Plugins?

- **Examples:** Flash, Silverlight, Java, PDF reader
- **Goal:** enable functionality that requires transcending the browser sandbox
- Increases browser's attack surface



## Java and Flash both vulnerable—again—to new 0-day attacks

Java bug is actively exploited. Flash flaws will likely be targeted soon.

by **Dan Goodin** (US) - Jul 13, 2015 9:11am PDT

- Good news: plugin sandboxing improving, and need for plugins decreasing (due to HTML5 and extensions)

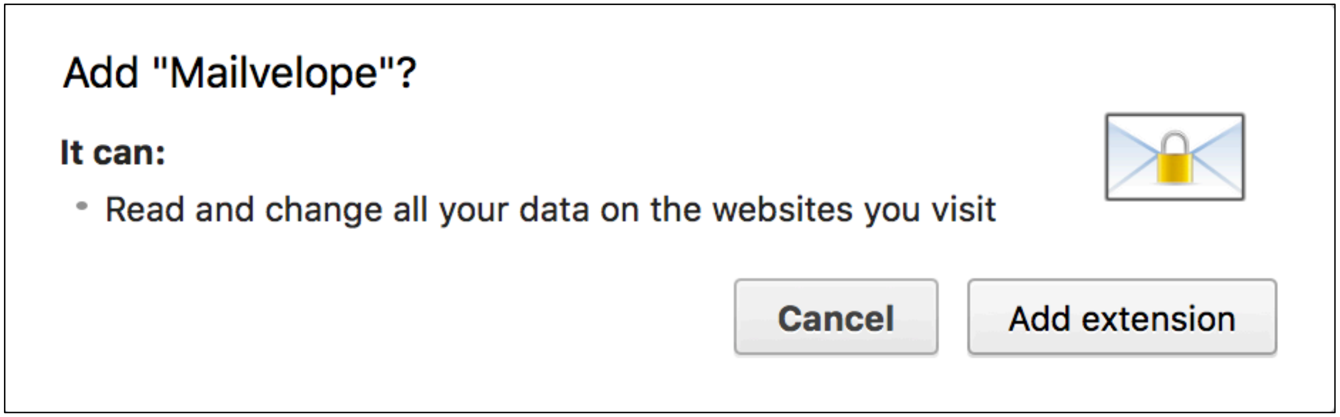
# What about Browser Extensions?

- Most things you use today are probably extensions
- **Examples:** AdBlock, Ghostery, Mailvelope
- **Goal:** Extend the functionality of the browser

- (Chrome:) Carefully designed security model to **protect from malicious websites**
  - Privilege separation: extensions consist of multiple components with well-defined communication
  - Least privilege: extensions request permissions

# What about Browser Extensions?

- But be wary of malicious extensions: **not subject to the same-origin policy** – can inject code into any webpage!



Add "Mailvelope"?

**It can:**

- Read and change all your data on the websites you visit