Botnets

Botnets

- Collection of compromised machines (bots) under (unified) control of an attacker (botmaster)
- Upon infection, new bot "*phones home*" to rendezvous w/ botnet *command-and-control* (**C&C**)
- Botmaster uses C&C to push out commands and updates

Method of control

- Lots of ways to architect C&C:
  - Star topology; hierarchical; peer-to-peer
  - Encrypted/stealthy communication

Method of compromise

- Method of compromise decoupled from method of control
  - Launch a worm / virus / drive-by infection / etc.

Botnets vs. Worms

- Constitute the *Great Modern Threat* of Internet security: *Generic Platform For Badness*
- Why botnets rather than worms?
  - Greater control
  - Less emergent
  - Quieter
  - Optimal flexibility
- Why the shift towards valuing these instead of seismic worm infection events?

  $$ Profit $$
- How can attackers leverage scale to monetize botnets?

Monetizing Botnets

- General malware monetization approaches
  - Keylogging: steal financial/email/social network accounts
  - Ransomware
  - *Transaction generators*
    - Malware watches user's surfing …
    - … waits for them to log into banking site (say) …
    - … and then injects additional banking transactions like "*send $50,000 to Nigeria*" …
    - … and alters web server replies to mask the change in the user's balance

Monetizing Botnets

- Monetization that leverages scale
  - DDoS (extortion)
  - Spam
  - *Click fraud*
  - Scam infrastructure
    - Hosting web pages (e.g., phishing)
    - Redirection to evade blacklisting/takedown (DNS)

- Which of these cause serious pain for infected user?
  - **None**. Users have little incentive to prevent ($\Rightarrow$ ***externality***)

Fighting Bots / Botnets

- How can we defend against bots / botnets?
- Approach #1: prevent the initial bot infection
  - Because the infection is decoupled from bot's participation in the botnet, this is equivalent to preventing malware infections in general .... HARD

Fighting Bots / Botnets, con’t

- Approach #2: seize the domain name used for C&C
  - This is what’s currently often used, often to good effect …
- … Botmaster counter-measure?
  - Each day (say), bots generate a large list of possible domain names using a Domain Generation Algorithm
    - Large = 50K, in some cases
  - Bots then try a random subset looking for a C&C server
    - Server **signs** its replies, so bot can’t be duped
    - Attacker just needs to hang on to a small portion of names to retain control over botnet
- Counter-counter measure?
  - Behavioral signature: look for hosts that make a lot of failed DNS lookups (research)

Addressing The Botnet Problem

- Angle #1: detection/cleanup
  - Detecting infection of individual bots hard as it's the *defend-against-general-malware* problem
  - Detecting bot doing C&C likely a losing battle as attackers improve their sneakiness & crypto
  - Cleanup today lacks oomph:
    - **Who's responsible?** … and do they **care**? (externalities)
    - Landscape could greatly change with different model of liability

Addressing The Problem, con't

- Angle #2: go after the C&C systems / botmasters
  - Difficult due to ease of Internet anonymity & complexities of international law
    - But: a number of recent successes in this regard
    - Including some via peer pressure rather than law enforcement (McColo)
  - One promising angle: policing domain name registrations

Addressing The Problem, con’t

- Angle #3: prevention
  - Secure code
  - structure OS/browser so code runs with Least Privilege
    - Does this solve the problem?
    - Depends on how granular the privileges are … and how the decision is made regarding just what privileges are “least”
      - E.g., iTunes App Store model (vetting), Android model (user confirmation)

Web Security: Vulnerabilities & Attacks

Introduction

Web & http

(browser)


```
HTTP REQUEST:
GET /account.html HTTP/1.1
Host: www.safebank.com
```

```
HTTP RESPONSE:
HTTP/1.0 200 OK
<HTML> . . . </HTML>
```

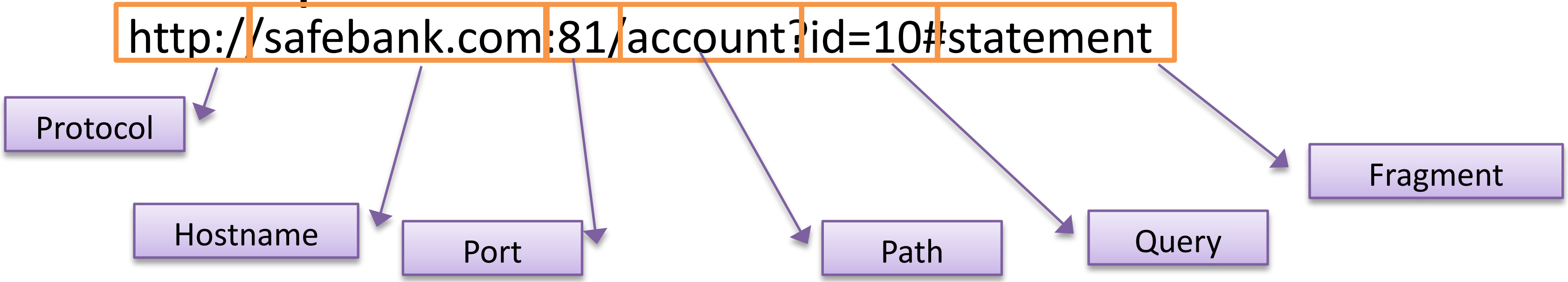
CLIENT

SERVER

URLs

- Global identifiers of network-retrievable documents

- **Example:**



- Special characters are encoded as hex:
  - %0A = newline
  - %20 or + = space, %2B = + (special exception)

HTTP Response

HTTP Request

Method File HTTP version

Headers

GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Chrome/21.0.1180.75 (Macintosh; Intel Mac OS X 10_7_4)
Host: www.safebank.com
Referer: http://www.google.com?q=dingbats

Blank line

Data – none for GET

GET : no side effect

POST : possible side effect

HTTP version Status code Reason phrase

Headers

HTTP/1.0 200 OK
Date: Sun, 12 Aug 2012 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 9 Aug 2012 17:39:05 GMT
Set-Cookie: …
Content-Length: 2543

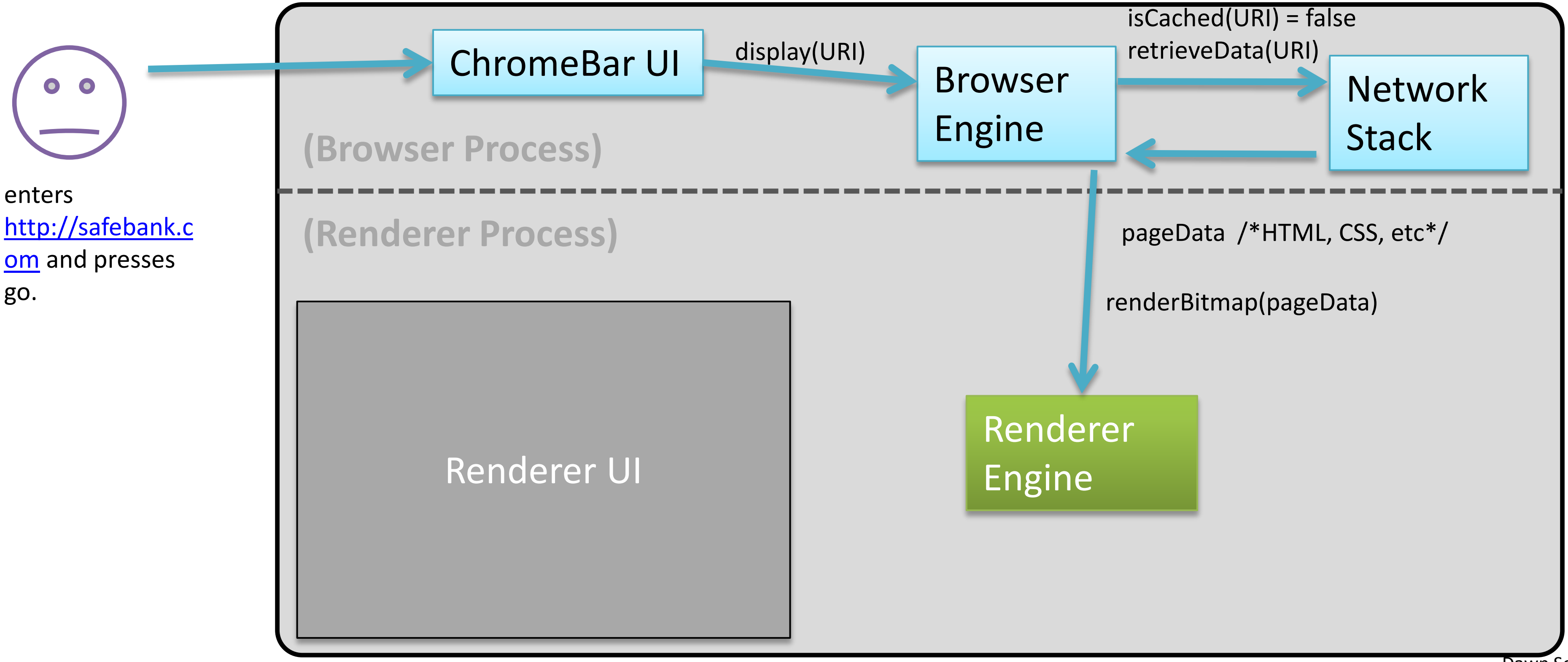
<HTML> This is web content formatted using html </HTML>

Data

Cookies

How browser renders a page

Suppose you are visiting http://safebank.com in a modern web browser.



Dawn Song

Rendering and events

- Basic execution model
  - Each browser window or frame
    - Loads content
    - Renders
      - Processes HTML and scripts to display page
      - May involve images, subframes, etc.
    - Responds to events
- Events can be
  - User actions: OnClick, OnMouseover
  - Rendering: OnLoad, OnBeforeUnload
  - Timing: setTimeout(), clearTimeout()

Document Object Model **(DOM)**

HTML

```
<html>
  <body>
    <div>
      foo
      <a>foo2</a>
    </div>
    <form>
      <input type="text” />
      <input type=”radio” />
      <input type=”checkbox” />
    </form>
  </body>
</html>
```

Object-oriented interface used to read and write rendered pages

– web page in HTML is structured data
– DOM provides representation of this hierarchy

Examples

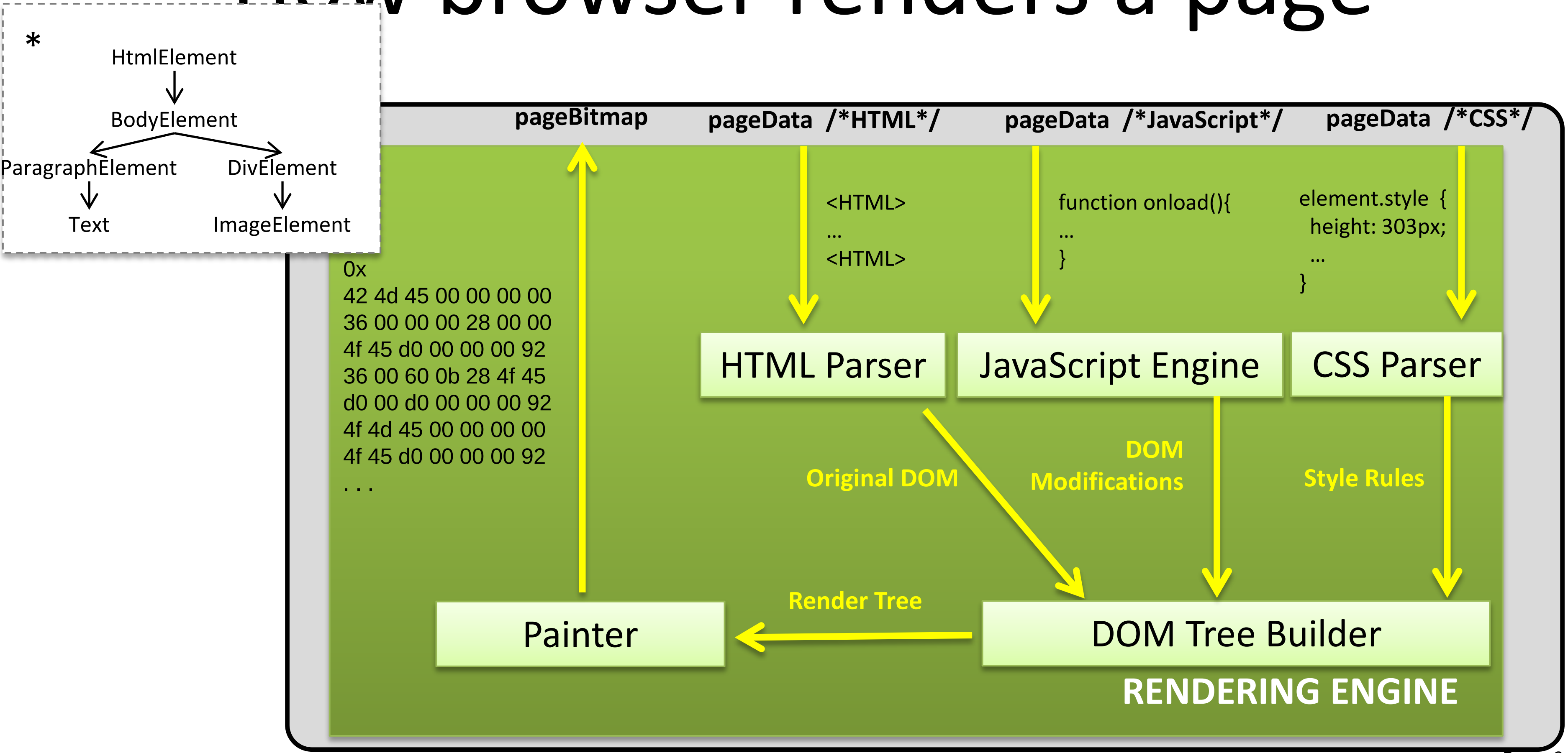
– Properties: document.alinkColor, document.URL, document.forms[ ], document.links[ ], document.anchors[ ]
– Methods: document.write(document.referrer)

DOM Tree

```
|-> Document
  |-> Element (<html>)
    |-> Element (<body>)
      |-> Element (<div>)
        |-> text node
        |-> Anchor
          |-> text node
      |-> Form
           |-> Text-box
           |-> Radio Button
           |-> Check Box
           |-> Button
```

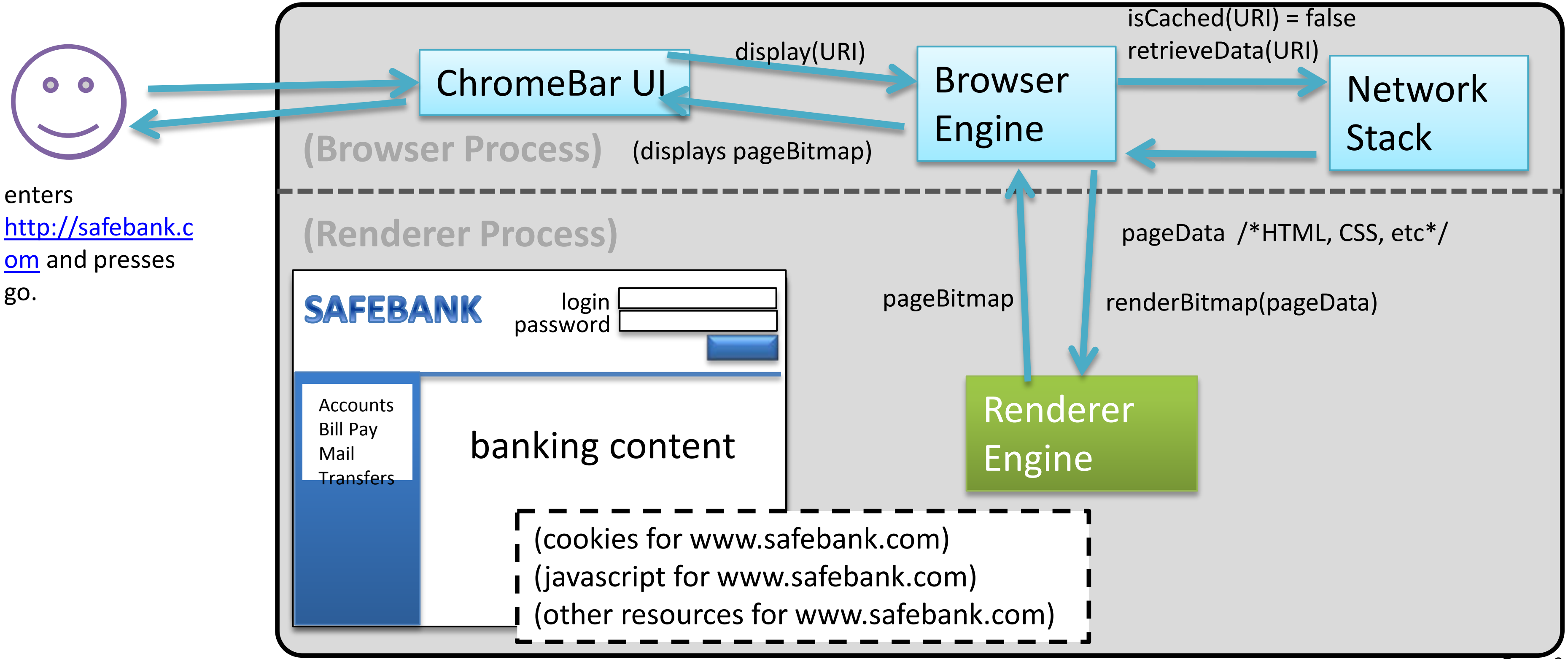
- Also: **Browser Object Model (BOM)**
  – window, document, frames[], history, location, navigator (type and version of browser)

How browser renders a page



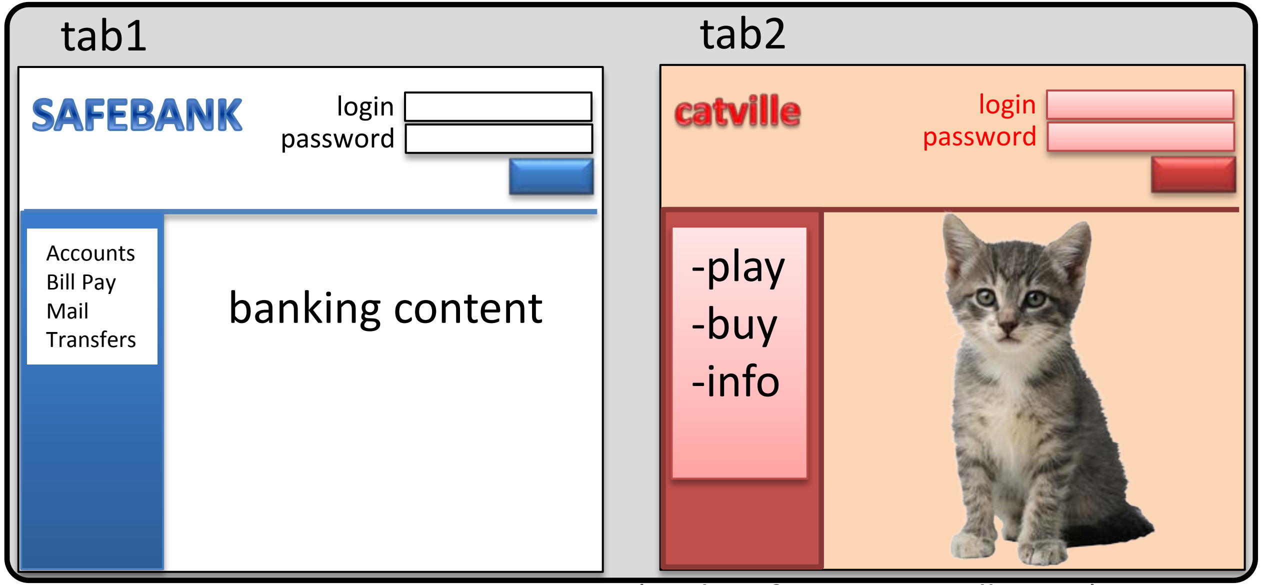
How browser renders a page

Suppose you are visiting http://safebank.com in a modern web browser.



Web Security Goals & Threat Model

Web Browser Security Goals



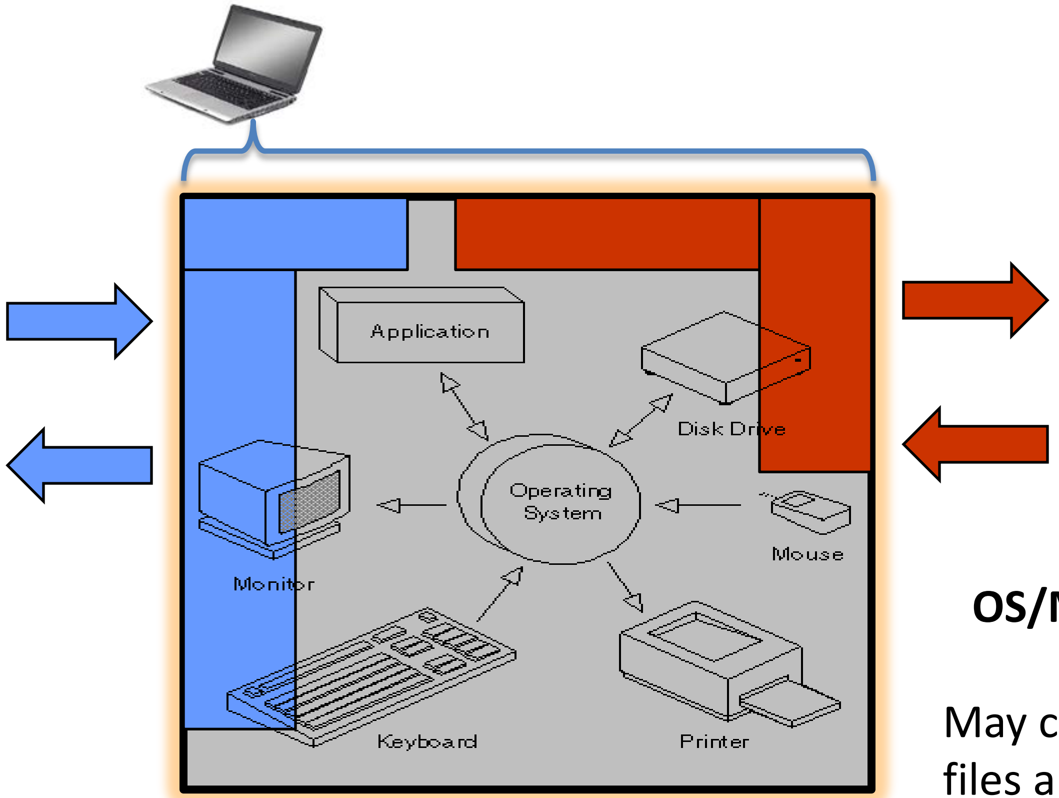
(cookies for www.safebank.com)
(javascript for www.safebank.com)
(other resources for www.safebank.com)

(cookies for www.catville.com)
(javascript for www.catville.com)
(other resources for www.catville.com)

Security Goals

- tab 2 cannot compromise the user's computer or data
- tab2 cannot steal information from tab1 (without user permission)
- tab 2 cannot compromise the session in tab 1

User

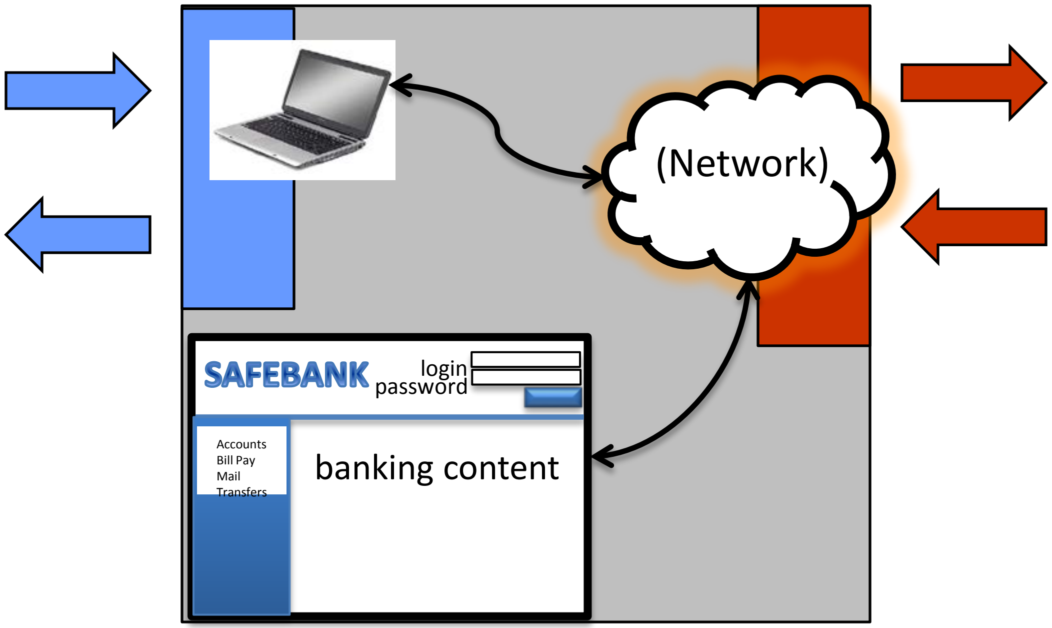


System

OS/Malware Attacker

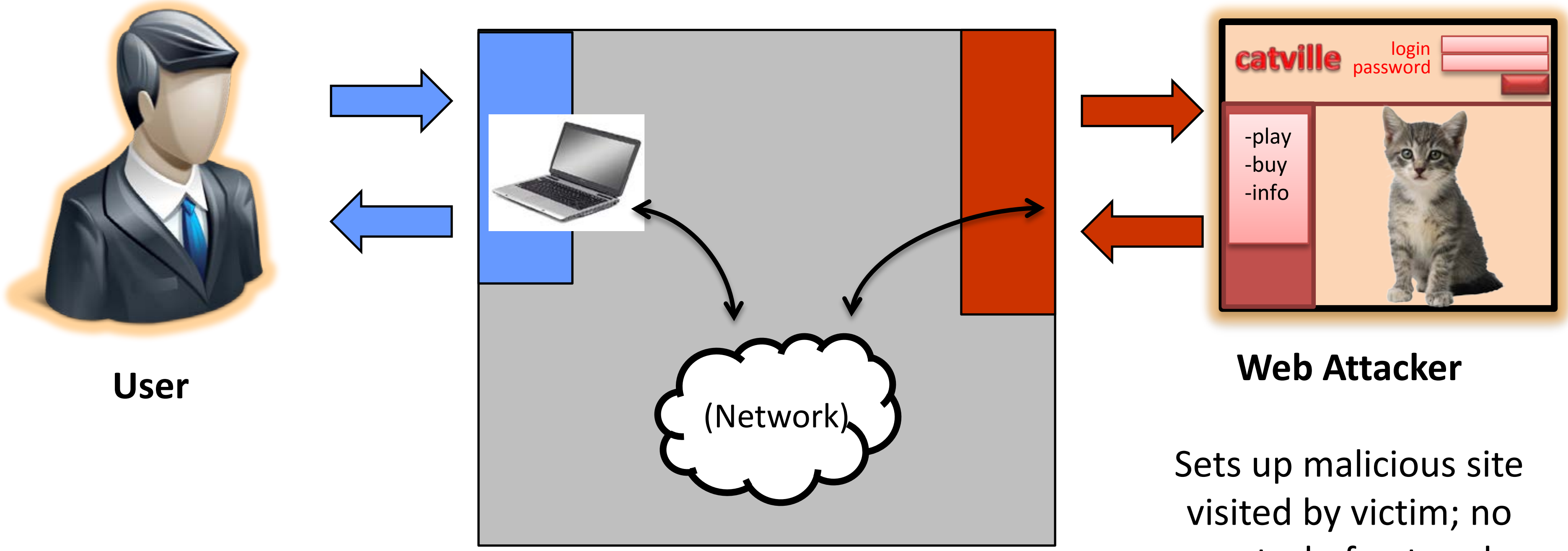
May control malicious files and applications on host

User



Network Attacker

Intercepts and controls network communication



Sets up malicious site visited by victim; no control of network

Web Threat Models

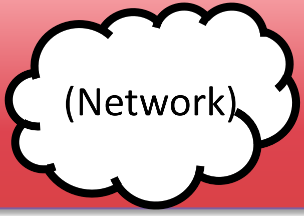
Web attacker

- Control malicious site, which we may call "attacker.com"
- Can obtain SSL/TLS certificate for attacker.com
- User visits attacker.com

  Or: runs attacker's Facebook app, site with attack ad, …

Network attacker

- Passive: Wireless eavesdropper
- Active: Evil router, DNS poisoning



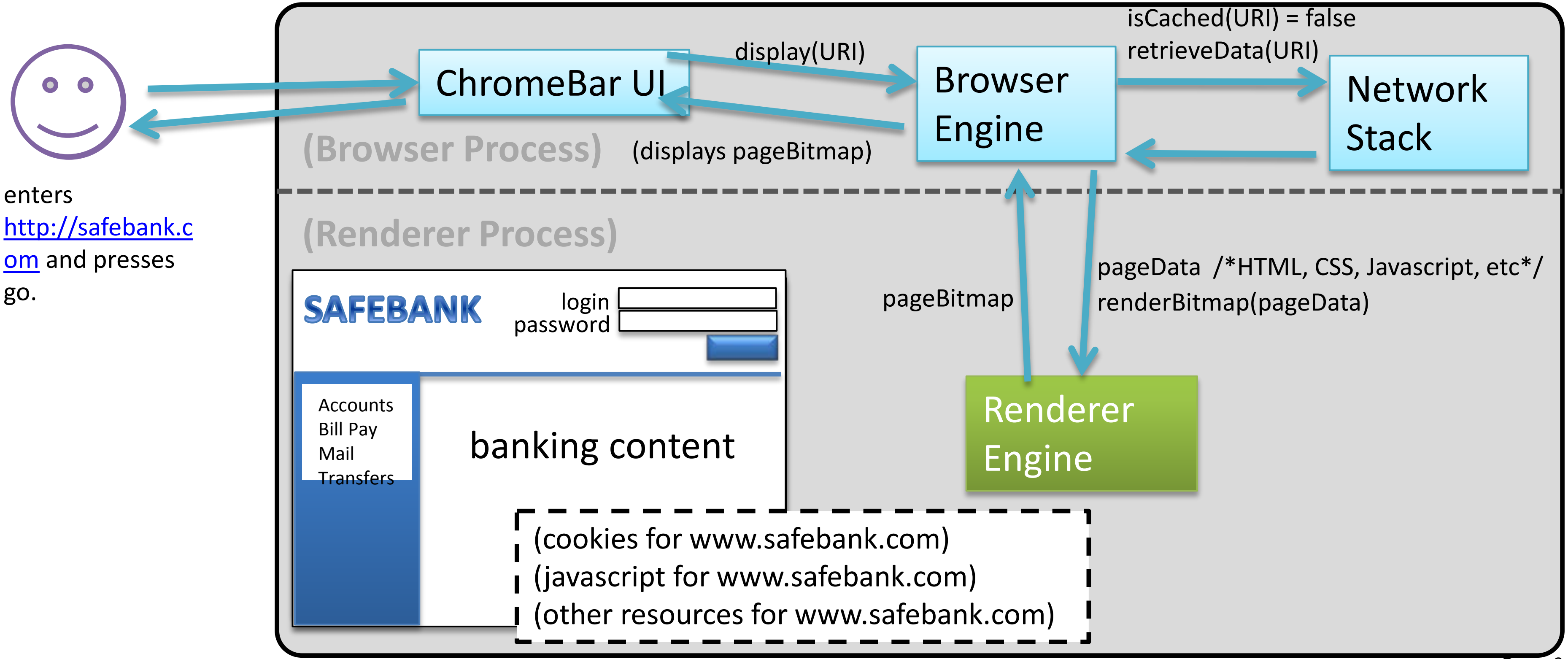
OS/Malware attacker

- Attackers may compromise host and install malware on host

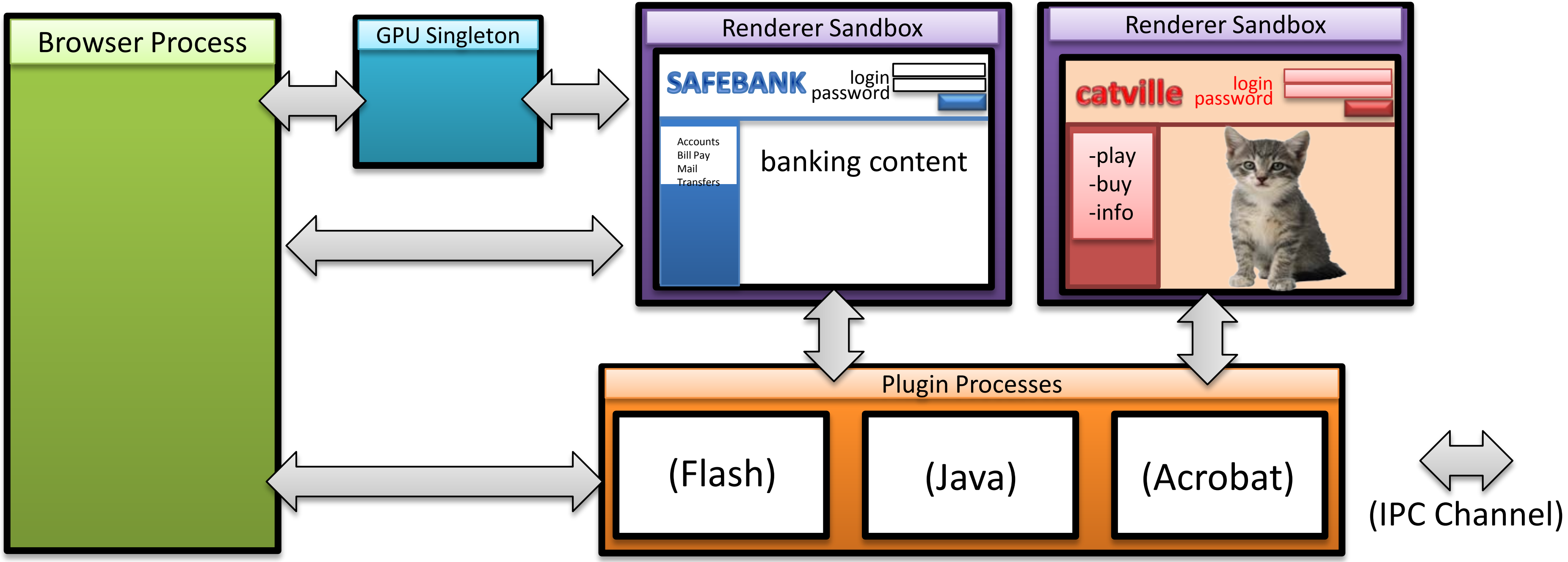
Isolation

How browser renders a page

Suppose you are visiting http://safebank.com in a modern web browser.



Chrome Security Architecture



Isolation: Separate web applications from each other, and separate browser components from each other

Principal of Least Privilege: Give components *only* the permissions they need to operate

Dawn Song

Render Sandbox

- Goal
  - Run remote web applications safely
  - Limited access to OS, network, and browser data
- Approach
  - Isolate sites in different security contexts
  - Browser manages resources, like an OS, so that each renderer has limited privilege

Frame and iFrame

- Window may contain frames from different sources
  - **Frame**: rigid division as part of frameset
  - **iFrame**: floating inline frame
- iFrame example

```
<iframe src="hello.html" width=450 height=100>
If you can see this, your browser doesn't understand IFRAME.
</iframe>
```

- Why use frames?
  - Delegate screen area to content from another source
  - Browser provides isolation based on frames
  - Parent may work even if frame is broken