Principle of Least Privilege

Principle of least privilege

- Privilege
  - Ability to access or modify a resource
- Principle of least privilege
  - A system module should only have the minimal privileges needed for intended purposes
- Privilege separation
  - Separate the system into independent modules
  - Each module follows the principle of least privilege
  - Limit interaction between modules

Unix access control

- File has access control list (ACL)
  - Grants permission to user ids
  - Owner, group, other
- Process has user id
  - Inherit from creating process
  - Process can change id
    - Restricted set of options
  - Special “root” id

	File 1	File 2	…
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
…			
User m	Read	write	write

Unix file access control list

- Each file has owner and group
- Permissions set by owner
  - Read, write, execute
  - Owner, group, other, setuid/setgid
  - Represented by vector of four octal values
- Only owner or root can change permissions
  - This privilege cannot be delegated or shared

Privileged Programs

- Privilege management is coarse-grained in today's OS
  - Root can do anything
- Many programs run as root
  - Even though they only need to perform a small number of priviledged operations
- What's the problem?
  - Privileged programs are juicy targets for attackers
  - By finding a bug in parts of the program that do not need privilege, attacker can gain root

What Can We Do?

- Drop privilege as soon as possible
- Ex: a network daemon only needs privilege to bind to low port # (<1024) at the beginning
  - Solution?
  - Drop privilege right after binding the port
- What benefit do we gain?
  - Even if attacker finds a bug in later part of the code, can't gain privilege any more
- How to drop privilege?
  - Setuid/setgid programming in UNIX

Effective user id (EUID) in UNIX

- Each process has three Ids
  - Real user ID (RUID)
    - same as the user ID of parent (unless changed)
    - used to determine which user started the process
  - Effective user ID (EUID)
    - from set user ID bit on the file being executed, or sys call
    - determines the permissions for process
      - file access and port binding
  - Saved user ID (SUID)
    - So previous EUID can be restored
- Real group ID, effective group ID, used similarly

Operations on UIDs

- Root
  - ID=0 for superuser root; can access any file
- Fork and Exec
  - Inherit three IDs, except exec of file with setuid bit
- Setuid system calls
  - seteuid(newid) can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID=0
- Details are actually more complicated
  - Several different calls: setuid, seteuid, setreuid

Setuid/setgid/sticky bits on executable Unix file

- Setuid/setgid/sticky bits
  - Setuid – set EUID of process to ID of file owner
  - Setgid – set EGID of process to GID of file
  - Sticky
    - Off: if user has write permission on directory, can rename or remove files, even if not owner
    - On: only file owner, directory owner, and root can rename or remove file in the directory

Setting UIDs

- **setresuid**() sets the real user ID, the effective user ID, and the saved set-user-ID of the calling process.
- **seteuid**() sets the effective user ID of the calling process.
- **setuid**() sets the effective user ID of the calling process. If the effective UID of the caller is root, the real UID and saved set-user-ID are also set.

Setting UIDs - What's Allowed?

Users choose any new UID to pass in to setuid(), but the OS checks them against certain rules and will raise an error, for example, if a normal user tries to call setuid(0).

setresuid(newruid, neweuid, newsuid)

```
(euid == 0)
  ||
(newruid in (ruid, euid, suid) &&
 neweuid in (ruid, euid, suid) &&
 newsuid in (ruid, euid, suid))
```

seteuid(neweuid)

```
(euid == 0)
||
(neweuid in (ruid, euid, suid))
```

setuid(newuid)

```
(euid == 0)
||
(newuid in (ruid, suid))
```

Cases

```
(euid == 0) ⇒ (ruid:=newuid,
  euid:=newuid, suid:=newuid)
(anything else) ⇒ (euid:=newuid)
```

Note: all policies are for Linux, differs on FreeBSD

Setting UIDs - What's Allowed?

Users choose any new UID to pass in to setuid(), but the OS checks them against certain rules and will raise an error, for example, if a normal user tries to call setuid(0).

setresuid(newruid, neweuid, newsuid)

```
(euid == 0)
  ||
(newruid in (ruid, euid, suid) &&
 neweuid in (ruid, euid, suid) &&
 newsuid in (ruid, euid, suid))
```

seteuid(neweuid)

```
(euid == 0)
||
(neweuid in (ruid, euid, suid))
```

setuid(newuid)

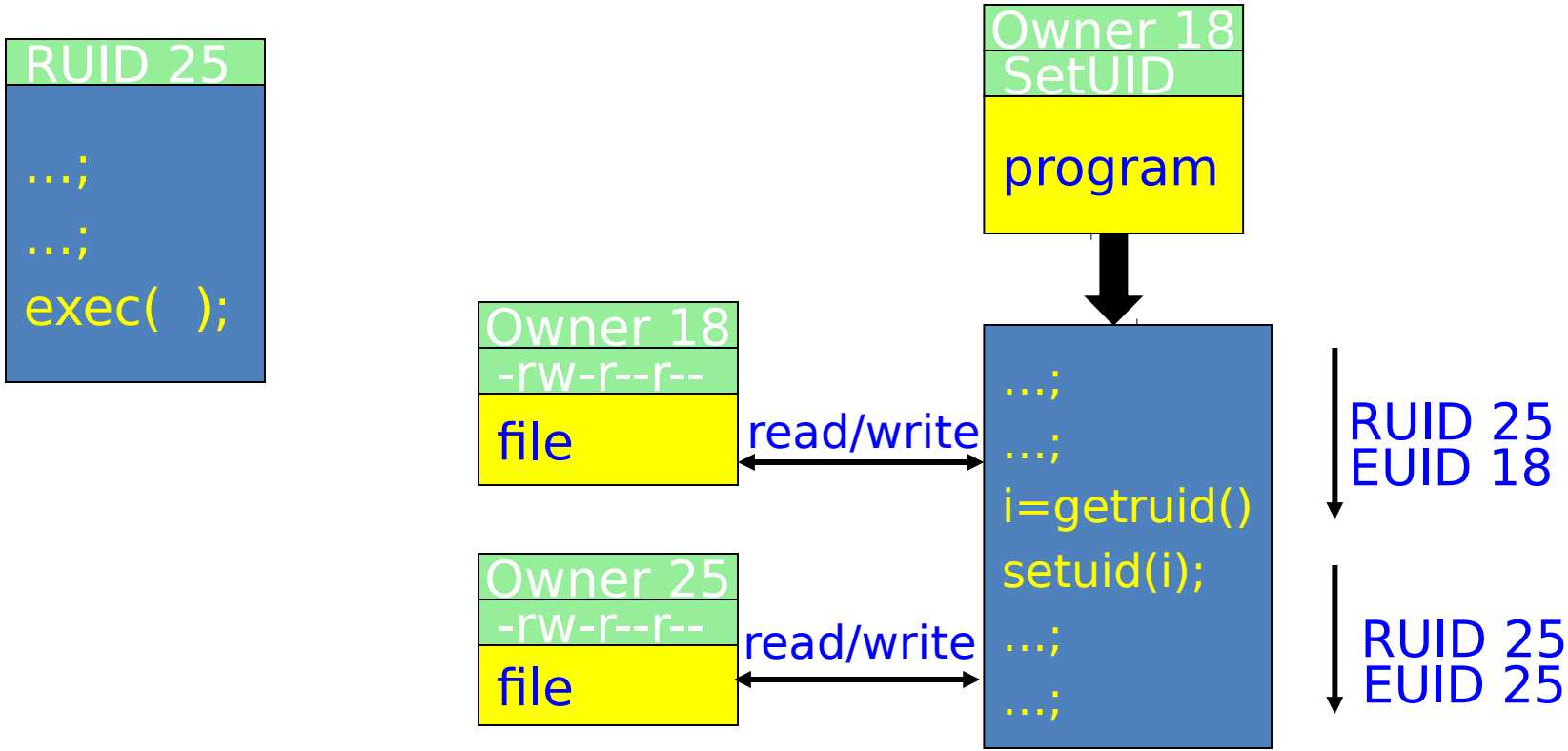
```
(euid == 0)
||
(newuid in (ruid, suid))
```

Cases

```
(euid == 0) ⇒ (ruid:=newuid,
  euid:=newuid, suid:=newuid)
(anything else) ⇒ (euid:=newuid)
```

Note: all policies are for Linux, differs on FreeBSD

Drop Privilege



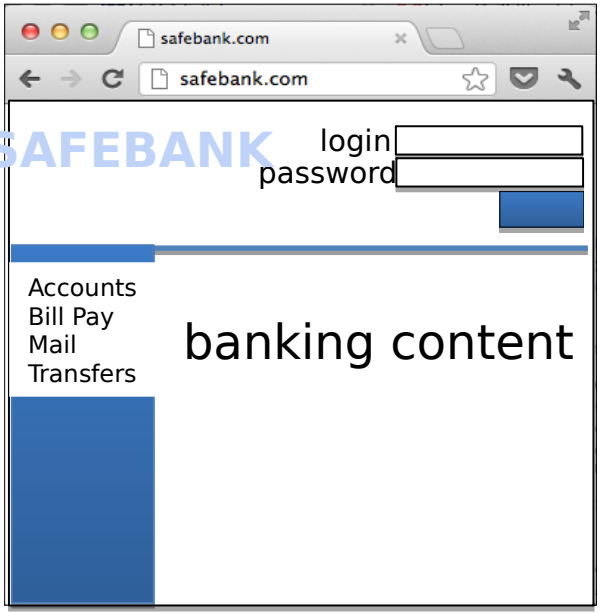
Web Security: Vulnerabilities & Attacks

Slide credit: Anthony Joseph and John Mitchell

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>
```

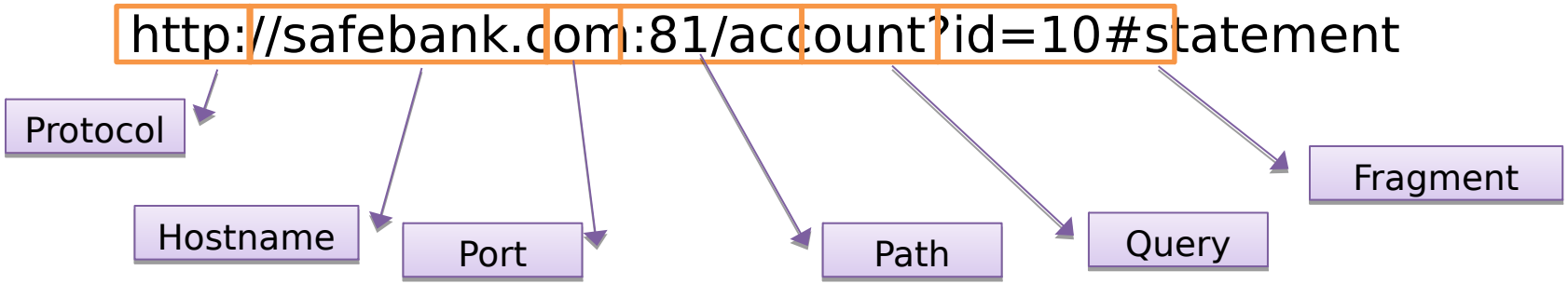
CLIENT

SERVER

URLs

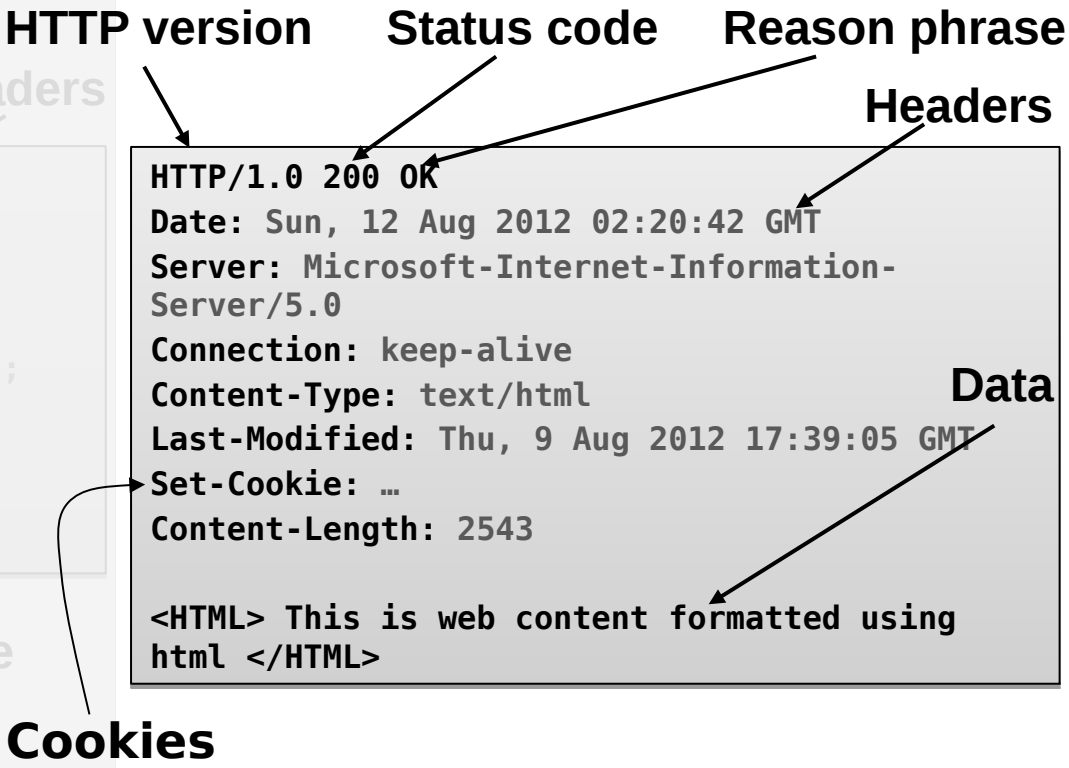
- Global identifiers of network-retrievable documents

- **Example:**



- Special characters are encoded as hex:
  - %0A = newline

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

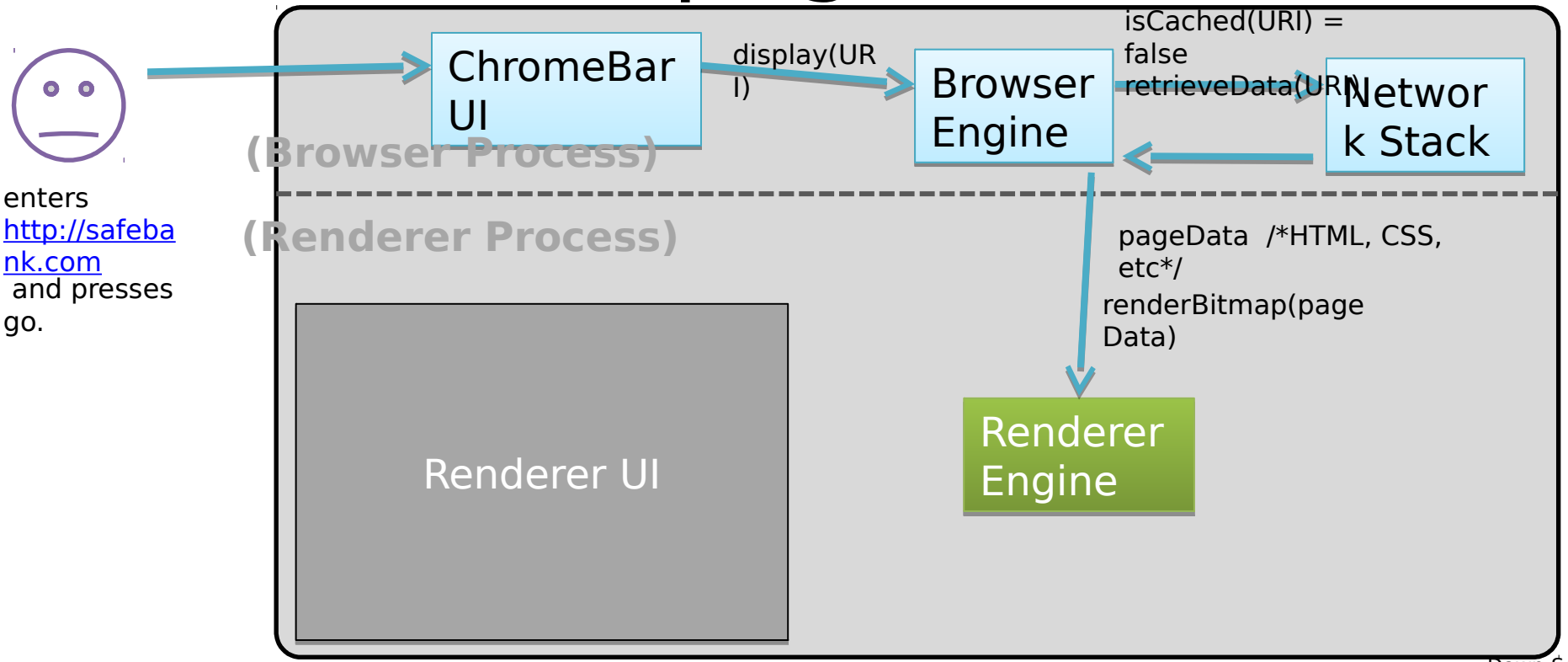
Data – none for GET

GET : no side effect

POST : possible

How browser renders a page

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



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

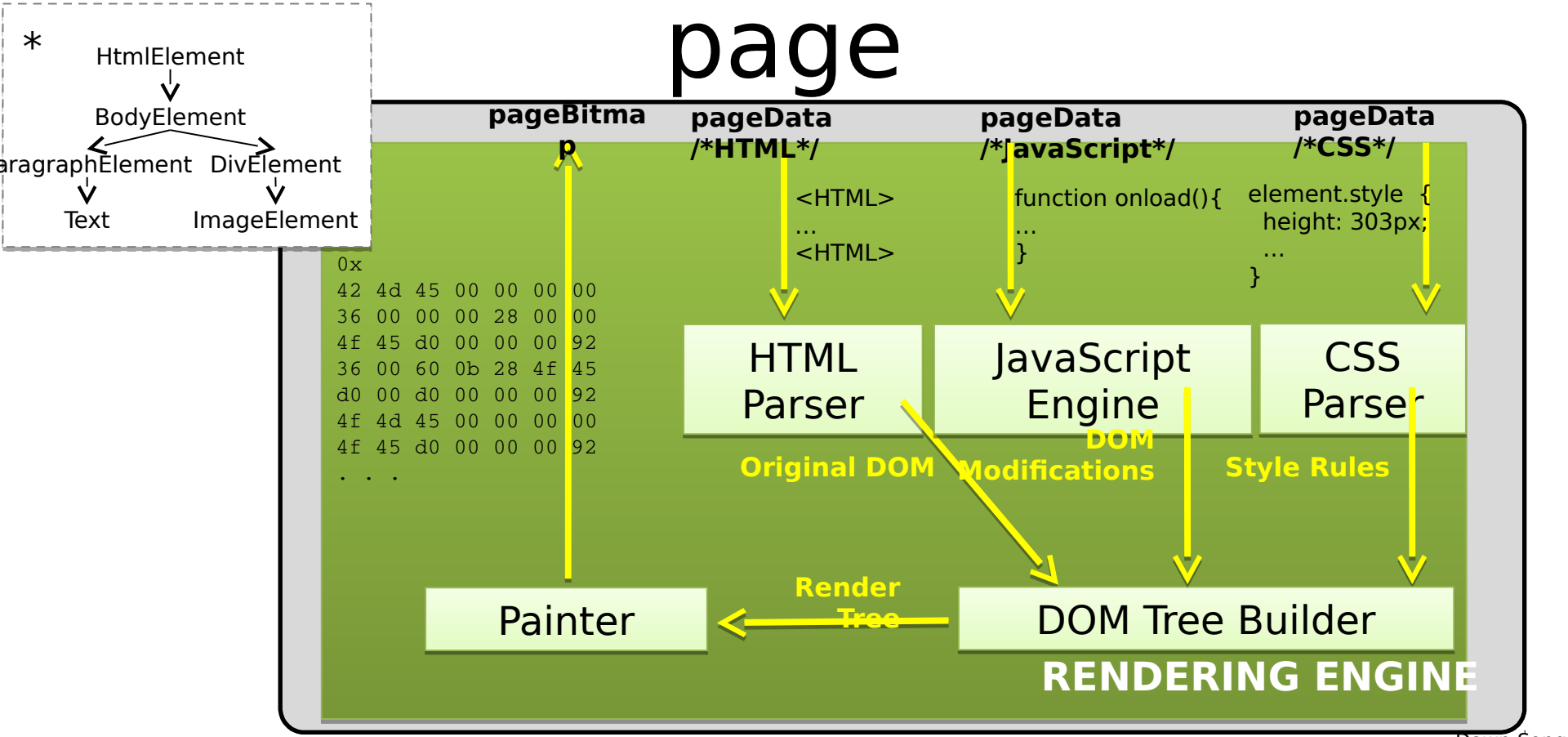
Examples

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

DOM Tree

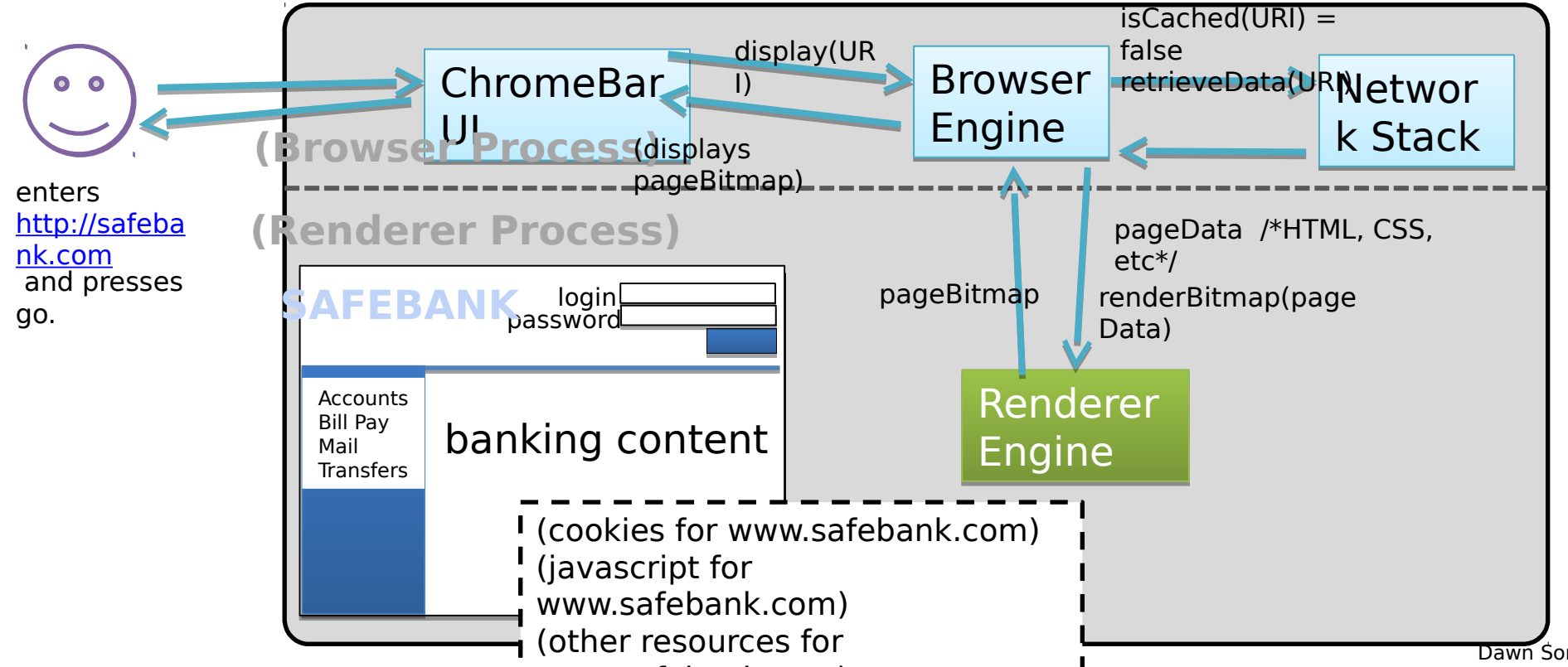
```
|-> Document
   |-> Element
(<html>)
     |-> Element
(<body>)
       |-> Element
(<div>)
         |-> text node
         |-> Anchor
           |-> text node
       |-> Form
            |-> Text-box
            |-> Radio
Button
            |-> Check Box
            |-> Button
```

How browser renders a page



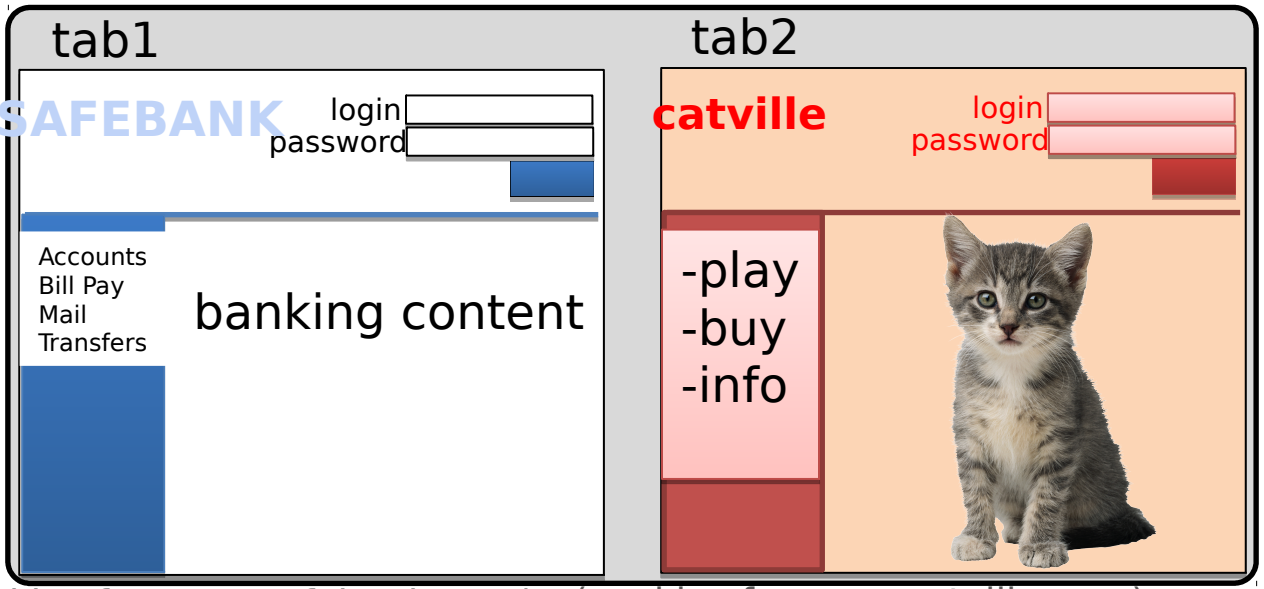
How browser renders a page

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



Web Security Goals & Threat Model

Web Browser Security Goals

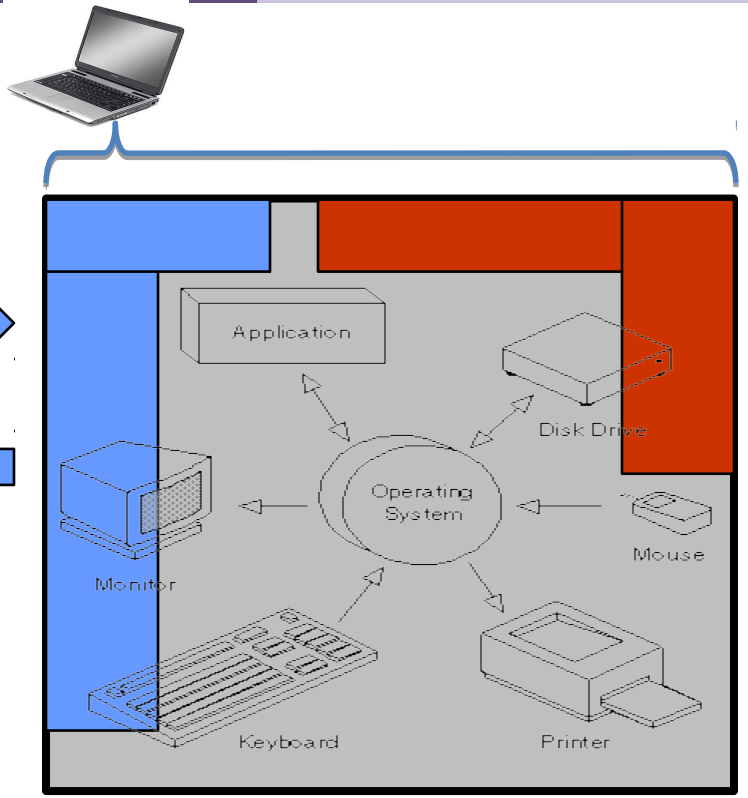


Security Goals

- 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

ookies for www.safebank.com) (cookies for www.catville.com)
vascript for www.safebank.com) (javascript for www.catville.com)
ther resources for www.safebank.com) (other resources for www.catville.com)

User

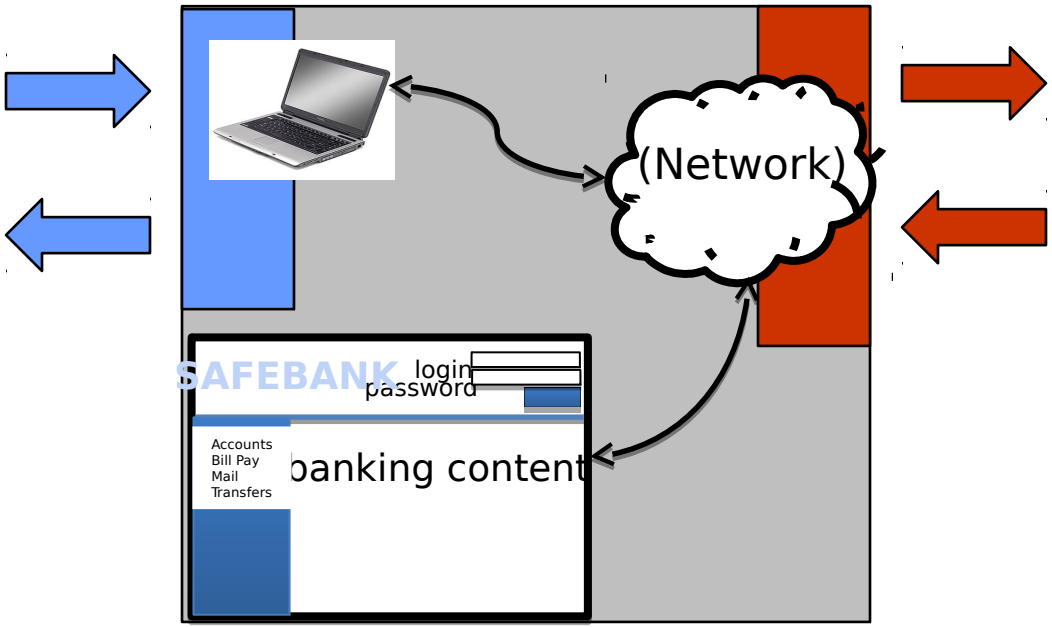


System

OS/Malware Attacker

May control malicious files and applications on host

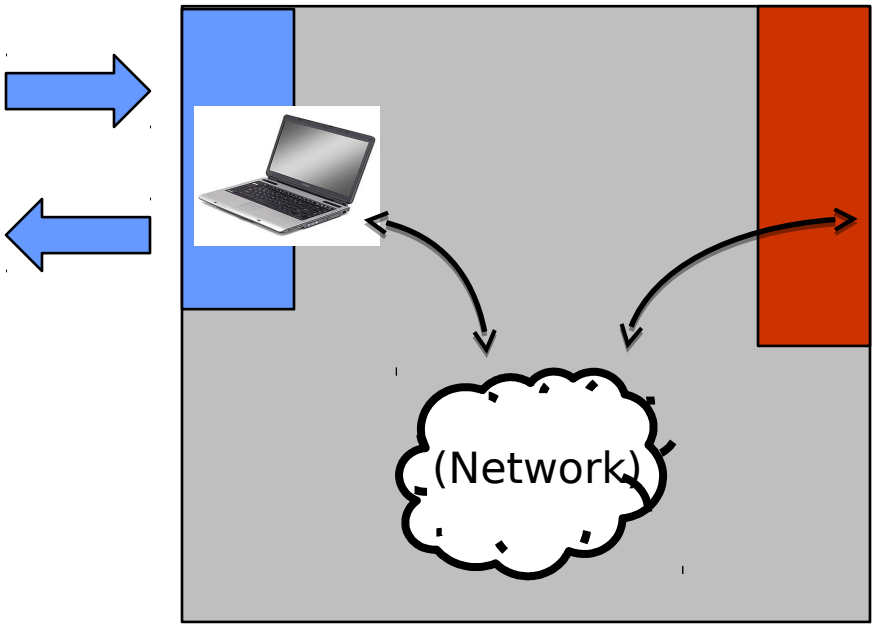
User



Network Attacker

Intercepts and controls network communication

User



Web Attacker

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

Web Threat Models

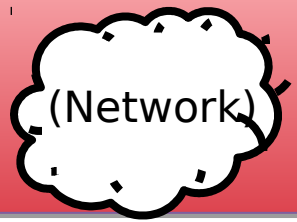
Web attacker

- 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



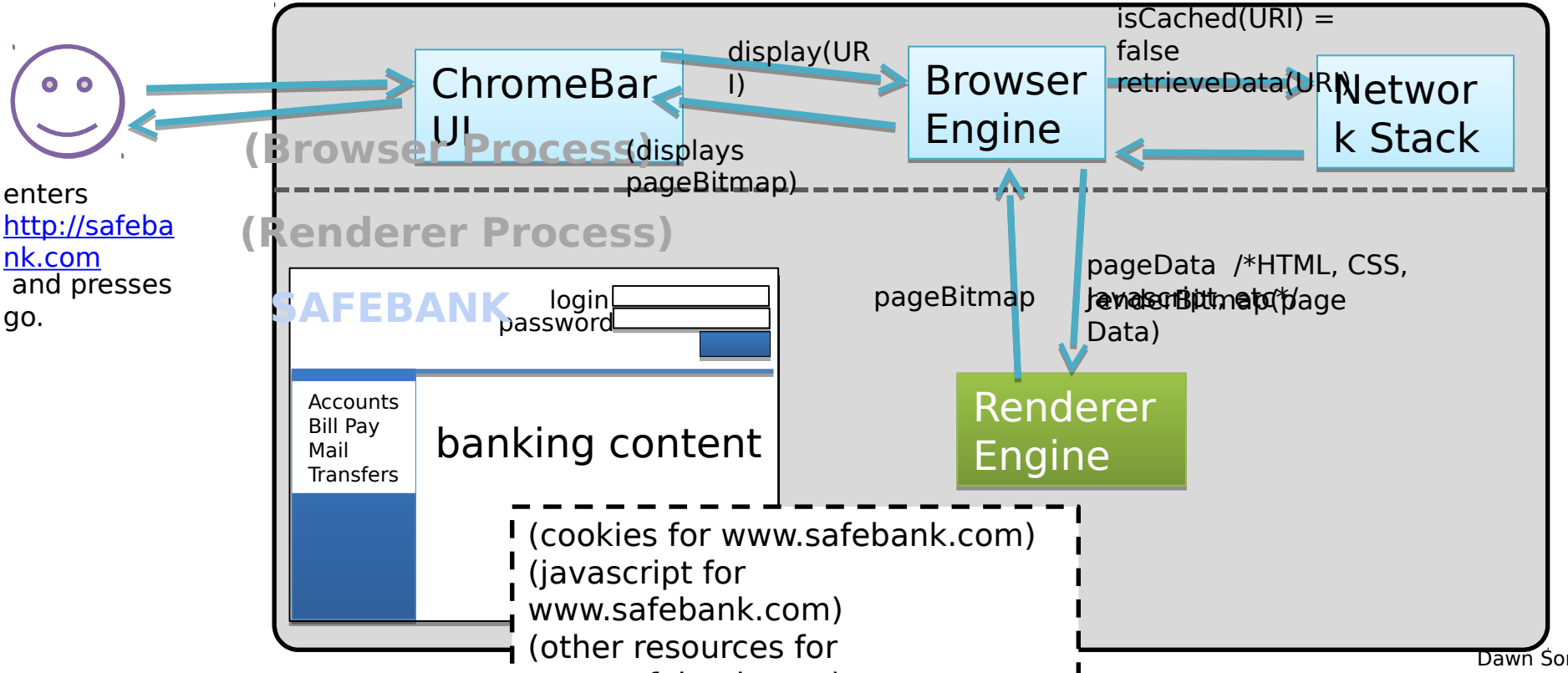
OS/Malware attacker

- Attackers may compromise host and install malware on host

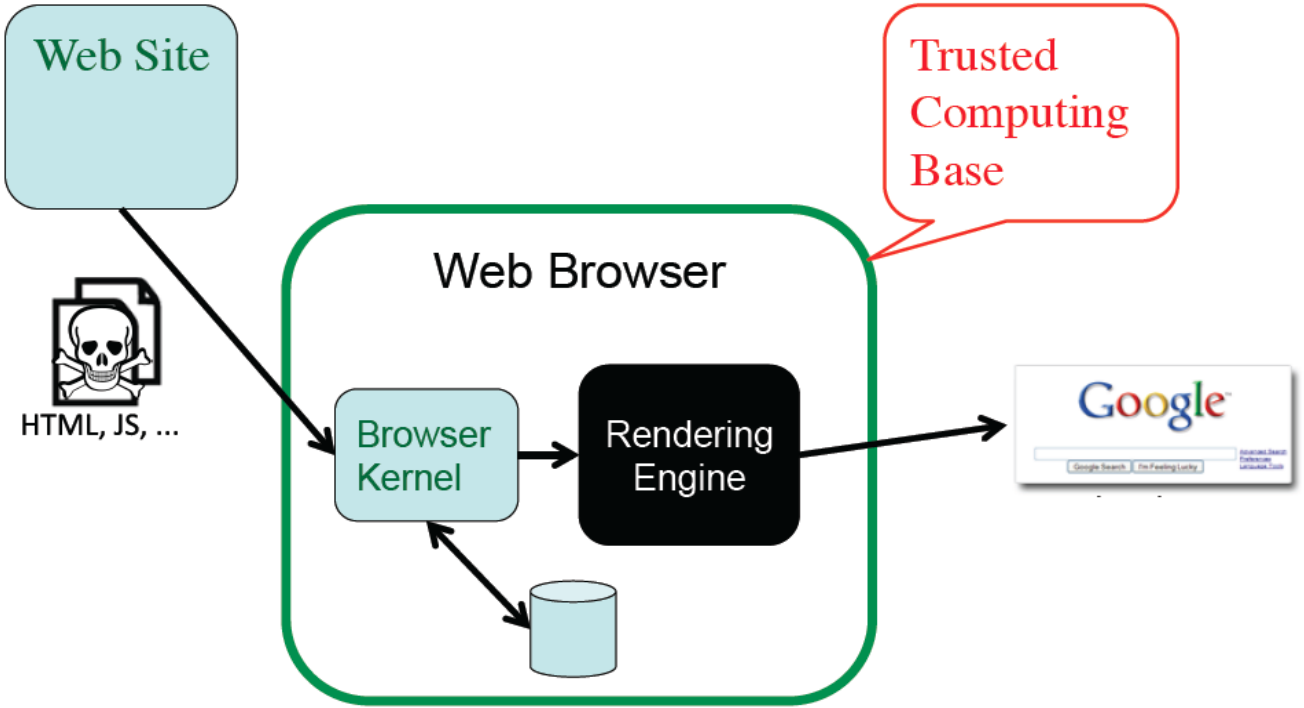
Isolation

How Browser Renders a Page

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

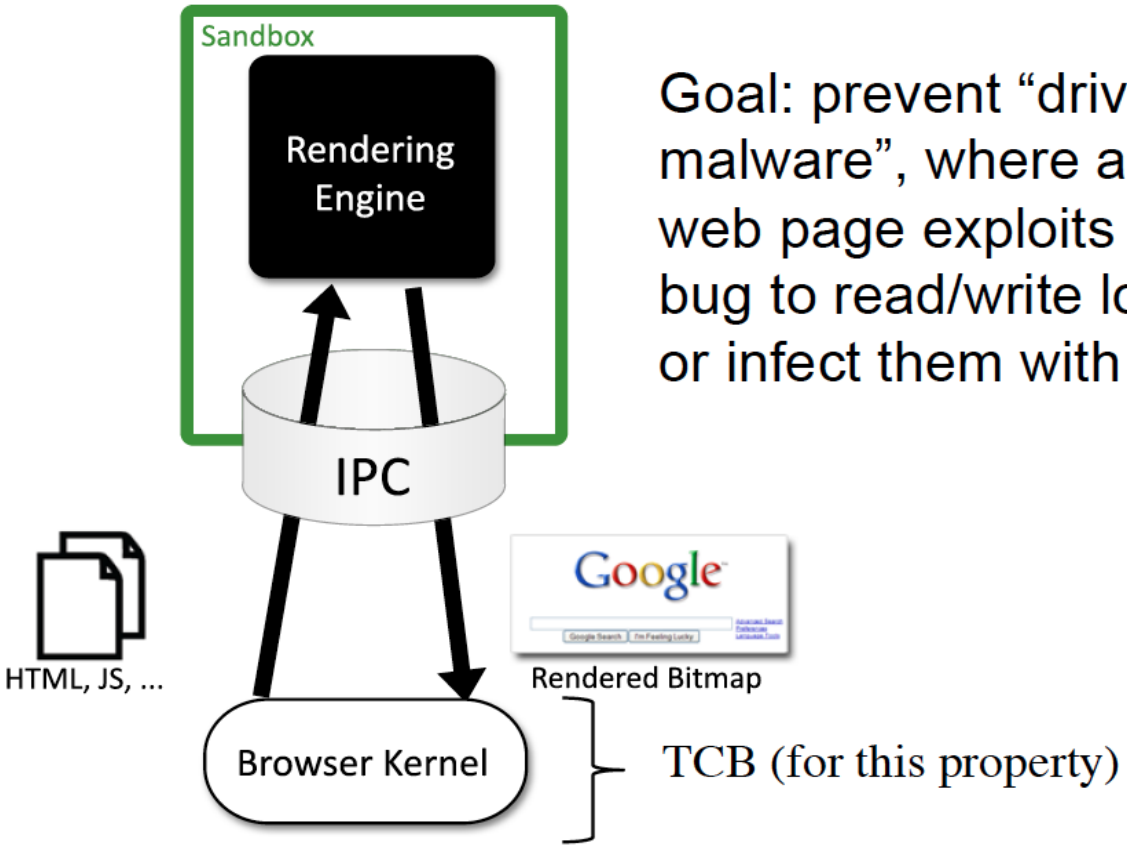


Web browser



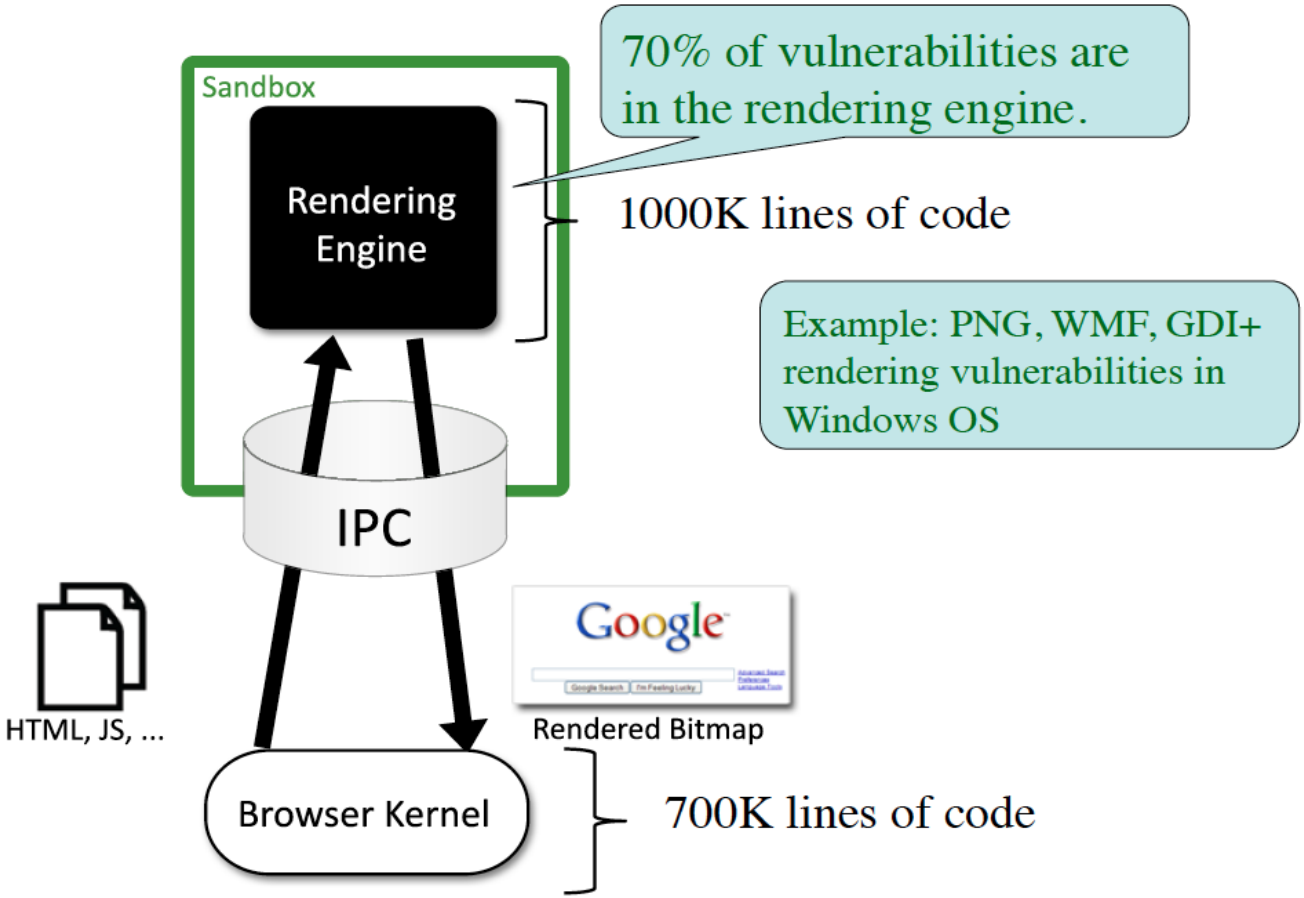
"Drive-by malware": malicious web page exploits a browser bug to read/write local files or infect them with a virus

The Chrome browser



Goal: prevent “drive-by malware”, where a malicious web page exploits a browser bug to read/write local files or infect them with a virus

The Chrome browser



Same Origin Policy

Operating system	Web Browser
	
Primitives:	
– Processes – System calls – File system	– Frames/iFrames – Content (including JavaScript, …) – Document object model, cookies, localStorage
Principals:	
Users – Discretionary access control	“Origins” – Mandatory access control
Vulnerabilities:	
– Buffer overflow – Root exploit	– Cross-site scripting – Cross-site request forgery – Cache history attacks

Origin of Browser Primitives

Cookies

Setting Cookies:

Default origin is **domain** and **path** of setting URL

Javascript

Imported **in** a page or frame:

Has the **same origin** as **that** page or frame

Embedded **in** a page or frame:

Has the **same origin** as **that** page or frame

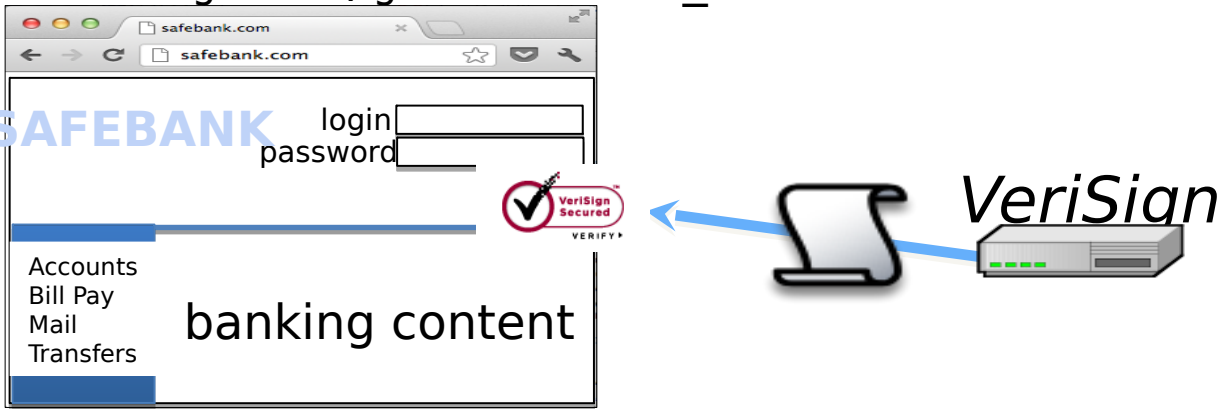
DOM

Each frame of a page:

Origin is
protocol://**host**:**port**

Library import

```
<script
   src=https://seal.verisign.com/getseal?host_name=safebank.com>
</script>
```



- Script has privileges of the importing page, NOT source server.

Same-origin policy
(for Javascript and DOM)

Two documents have the same origin if:

- Same **protocol** (https, http, ftp, etc)
- Same **domain** (safebank.com, etc)
- Same **port** (80, 23, 8080, etc)

Results of same-origin checks against “http://cards.safebank.com/c1/info.html”

Same origin:

“http://cards.safebank.com/c2/edit.html”
“http://cards.safebank.com/”

Different origin:

“http://www.cards.safebank.com”
“http://catville.com”
“https://cards.safebank.com”
“http://cards.safebank:8080”