Web Security

CS 161: Computer Security

Ruta Jawale and Rafael Dutra

July 25, 2019

The web architecture is a mess when it comes to security

Slides credit: Raluca Ada Popa, David Wagner, Dan Boneh
Announcements

- Project 2 due next week Monday (7/29)
  - Project party tomorrow (3-5 pm @ Soda 606)
- Homework 2 due next week Thursday (8/1)
- Midterm 2 in 1.5 weeks (8/5)
  - Make sure to attend lectures and discussions
Web 101
What is the Web?

A platform for deploying applications and sharing information, portably and securely
HTTP
(Hypertext Transfer Protocol)
A common data communication protocol on the web
URLs

Global identifiers of network-retrievable resources

Example:

```
http://safebank.com:81/account?id=10#statement
```

- **Protocol**
  - http, https, ftp, ...
- **Port**
  - http: 80, https: 443, ...
- **Sent to web server**
  - path, query
- **Local to client browser**
  - fragment
HTTP

CLIENT BROWSER

SAFEBANK

Accounts
Bill
Pay
Mail
Transfers

Alice Smith

WEB SERVER

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

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

GET: no side effect
POST: possible side effect

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

Data – none for GET

Blank line

Method  Path  HTTP version  Headers
HTTP REQUEST:
GET /account.html HTTP/1.1
Host: www.safebank.com

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

HTTP version | Status code | Reason phrase |
-------------|-------------|---------------|
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>
HTML

A language to create structured documents
One can embed images, objects, or create interactive forms

index.html
<html>
  <body>
    <div>
      foo
      <a href="http://google.com">Go to Google!</a>
    </div>
    <form>
      <input type="text" />
      <input type="radio" />
      <input type="checkbox" />
    </form>
  </body>
</html>
CSS (Cascading Style Sheets)

Style sheet language used for describing the presentation of a document

```css
index.css

p.serif {
  font-family: "Times New Roman", Times, serif;
}
p.sansserif {
  font-family: Arial, Helvetica, sans-serif;
}
```
Javascript

Programming language used to manipulate web pages. It is a high-level, untyped and interpreted language with support for objects.

Supported by all web browsers

```
<script>
function myFunction() {
document.getElementById("demo").innerHTML = "Text changed.";
}
</script>

Very powerful!
HTTP

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

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

webpage
Page rendering

- HTML
- CSS
- Javascript

HTML Parser
CSS Parser
JS Engine

DOM

modifications to the DOM

Painter

bitmap
DOM (Document Object Model)

a cross-platform model for representing and interacting with objects in HTML

HTML
<html>
  <body>
    <div>
      foo
    </div>
    <form>
      <input type="text" />
      <input type="radio" />
      <input type="checkbox" />
    </form>
  </body>
</html>

DOM Tree

|-> Document
  |-> Element (<html>)
  |-> Element (<body>)
  |-> Element (<div>)
    |-> text node
    |-> Form
      |-> Text-box
      |-> Radio Button
      |-> Check Box
Web & HTTP 101

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

HTTP_RESPONSE:
HTTP/1.0 200 OK
<HTML> . . . </HTML>
The power of Javascript

Get familiarized with it so that you can think of all the attacks one can do with it
What can you do with Javascript?

Almost anything you want to the DOM!

A JS script embedded on a page can modify in almost arbitrary ways the DOM of the page. The same happens if an attacker manages to get you load a script into your page.

w3schools.com has nice interactive tutorials: https://www.w3schools.com/js
Example of what Javascript can do…

Can change HTML content:

```html
<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick="document.getElementById('demo').innerHTML = 'Hello JavaScript!'">
    Click Me!</button>
```

DEMO from w3schools.com
Other examples

- Can change images
- Can change style of elements
- Can hide elements
- Can unhide elements
- Can change cursor
Other example: can access cookies

Will learn later that cookies are useful for authentication.

JS can read cookie:

```javascript
var x = document.cookie;
```

Change cookie with JS:

```javascript
document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```
Frames
Frames

- Enable embedding a page within a page

```
<iframe src="URL"></iframe>
```
Frames

- **Modularity**
  - Brings together content from multiple sources
  - Client-side aggregation

- **Delegation**
  - Frame can draw only on its own rectangle

*Slide from Dan Boneh*
Frames

• Outer page can specify only sizing and placement of the frame in the outer page
  • demo
• Frame isolation: Our page cannot change contents of inner page, inner page cannot change contents of outer page
Web Security
A historical perspective

• The web is an example of “bolt-on security”
• Originally, the web was invented to allow physicists to share their research papers
  – Only textual web pages + links to other pages; no security model to speak of
The web became complex and adversarial quickly

- Then we added embedded images
  - Crucial decision: a page can embed images loaded from another web server
- Then, Javascript, dynamic HTML, AJAX, CSS, frames, audio, video, ...
- Today, a web site is a distributed application
- Attackers have various motivations

Web security is a challenge!
Desirable security goals

• **Integrity**: malicious web sites should not be able to tamper with integrity of my computer or my information on other web sites

• **Confidentiality**: malicious web sites should not be able to learn confidential information from my computer or other web sites

• **Privacy**: malicious web sites should not be able to spy on me or my activities online

• **Availability**: attacker cannot make site unavailable
Security on the web

• Risk #1: we don’t want a malicious site to be able to trash my files/programs on my computer
  – Browsing to awesomevids.com (or evil.com) should not infect my computer with malware, read or write files on my computer, etc.

• Defense: Javascript is sandboxed; try to avoid security bugs in browser code; privilege separation; automatic updates; etc.
Security on the web

• Risk #2: we want data stored on a web server to be protected from unauthorized access
• Defense: server-side security
  – Think Project 2
Security on the web

- Risk #3: we don’t want a malicious site to be able to spy on or tamper with my information or interactions with other websites
  - Browsing to evil.com should not let evil.com spy on my emails in Gmail or buy stuff with my Amazon account
- Defense: the same-origin policy
  - A security policy grafted on after-the-fact, and enforced by web browsers
Security on the web

• Risk #4: we don’t want malicious websites to subvert or act in opposition to user’s intent
  – Clickjacking attack
• Defense: frame busting can help prevent some clickjacking attacks
Break Time: Spencer McCall

- Missouri -> San Diego, CA
- Enjoys game theory, also crypto
- English, French, Italian

- Accidentally DoS government server while web scrapping
Same-origin Policy
Same-origin policy

One origin should not be able to access the resources of another origin

Javascript on one page cannot read or modify pages from different origins
Same-origin policy

• Each site in the browser is isolated from all others
Same-origin policy

- Multiple pages from the same site are not isolated
Same-origin policy

• The origin of a site is derived from its URL
Same-origin policy

- The origin of a site is derived from its URL
  - Images adopt origin of site that loads them
Same-origin policy

- The origin of a site is derived from its URL
  - Images adopt origin of site that loads them
  - Javascript runs with the origin of the site that loaded it
Origins of Frames

• iframes do not adopt origin of site that loads them
  – iframe origin is the inner site (being displayed), and not the outer site (loading website)
Origin

- Granularity of protection for same origin policy
- Origin = protocol + hostname + port

http://coolsite.com:81/tools/info.html

- Origin is determined by **string matching**! If these match, it is same origin, else it is not.
  - However, port matching depends on browser implementation
<table>
<thead>
<tr>
<th>Originating document</th>
<th>Accessed document</th>
</tr>
</thead>
</table>

**Exercises**

Except **e** !!!
Chromodo
Private Internet Browser

Fast and versatile Internet Browser based on Chromium, with highest levels of speed, security and privacy!

Issue 704: Comodo: Comodo "Chromodo" Browser disables same origin policy, Effectively turning off web security.

13 people starred this issue and may be notified of changes

Project Member
Reported by tav...@google.com, Jan 21, 2016

When you install Comodo Internet Security, by default a new browser called Chromodo is installed and set as the default browser. Additionally, all shortcuts are replaced with Chromodo links and all settings, cookies, etc are imported from Chrome. They also hijack DNS settings, among other shady practices.


Chromodo is described as "highest levels of speed, security and privacy", but actually disables all web security. Let me repeat that, they ***disable the same origin policy***... ?!?
Cross-origin communication

- Allowed through a narrow API: `postMessage`
- Receiving origin decides if to accept the message based on origin (whose correctness is enforced by browser)

```javascript
postMessage("run this script", script)
```

Check origin, and request!
Clickjacking
Clickjacking attacks

- Exploitation where a user’s mouse click is used in a way that was not intended by the user
Talk to your partner

- How can a user’s click be used in a way different than intended?
Simple example

```html
    Go to Google
</a>
```

What does it do?

• Opens a window to the attacker site

Why include `href` to Google?

• Browser status bar shows URL when hovering over as a means of protection
What happens in this case?

Funny cats website

JavaScript

Bank of America

Secure Sign-In

secret  secret  Sign In

BankAmericard

Banking  Credit Cards  Loans  Investments

$10
Frames: same-origin policy

- Frame inherits origin of its URL
- Same-origin policy: if frame and outer page have different origins, they cannot access each other
  - In particular, malicious JS on outer page cannot access resources of inner page
How to bypass same-origin policy for frames?

Clickjacking
Clickjacking using frames

- Evil site frames good site
- Evil site covers good site by putting dialogue boxes or other elements on top of parts of framed site to create a different effect
- Inner site now looks different to user
How can we defend against clickjacking?
Defenses

• User confirmation
  - Good site pops dialogue box with information on the action it is about to make and asks for user confirmation
  - Degrades user experience

• UI randomization
  - Good site embeds dialogues at random locations so it is hard to overlay
  - Difficult & unreliable (e.g. multi-click attacks)
Defense 3: Framebusting

Web site includes code on a page that prevents other pages from framing it

Demo
What is framebusting?

Framebusting code is often made up of

• a conditional statement and
• a counter action

Common method:

```java
if (top != self) {
    top.location = self.location;
}
```
A Survey

Framebusting is very common at the Alexa Top 500 sites

Top 10 60%
Top 100 37%
Top 500 14%

credit: Gustav Rydstedt
Many framebusting methods

if (top != self)
if (top.location != self.location)
if (top.location != location)
if (parent.frames.length > 0)
if (window != top)
if (window.top !== window.self)
if (window.self != window.top)
if (parent && parent != window)
if (parent && parent.frames && parent.frames.length>0)
if((self.parent && !(self.parent===self)) && (self.parent.frames.length!=0))
Many framebusting methods

```
    top.location = self.location
    top.location.href = document.location.href
    top.location.href = self.location.href
    top.location.replace(self.location)
    top.location.href = window.location.href
    top.location.replace(document.location)
    top.location.href = window.location.href
    top.location.href = "URL"
    document.write("")
    top.location = location
    top.location.replace(document.location)
    top.location.replace('URL')
    top.location.href = document.location
```
Most current framebusting can be defeated
Goal: bank.com wants only bank.com’s sites to frame it

Bank runs this code to protect itself:

```javascript
if (top.location != location) {
    if (document.referrer &&
        document.referrer.indexOf("bank.com") == -1)
    {
        top.location.replace(document.location.href);
    }
}
```

Problem: http://badguy.com?q=bank.com
Abusing the XSS filter

IE8 reflective XSS filters:

- Browser requested URL contains javascript:
  - http://www.victim.com?var=<script> alert('xss'); </script>
- Server responds
- Browser checks
  - If <script> alert('xss'); </script> appears in rendered page word for word, the IE8 filter will replace it with
    <sc#pt> alert('xss'); </sc#pt>

How can attacker abuse this?
Abusing the XSS filter

• Attacker figures out the framebusting code of victim site (easy: visit victim site in attacker’s browser and view the source code)
  ■ `<script> if(top.location != self.location) //framebust </script>`

• Framing page (attacker’s outer page) does:
  ■ `<iframe src="http://www.victim.com?var=<script>
    if(top.location != self.location) //framebust </script>">`

• IE8 XSS filter modifies victim site’s script to:
  ■ `<script> if(top.location != self.location) `

XSS filter disables legitimate framebusting code!!
Coming up:
attacks on web servers!