CSE 127 Computer Security

Stefan Savage, Fall 2019, Lecture 11

Web Security I

Web Security

Client

- Hey, server, you may remember me. You gave me this token when we talked some time back.
- Anyway, please run one of your scripts for me with the following arguments and let me see the result.
- kthxbye.

Server

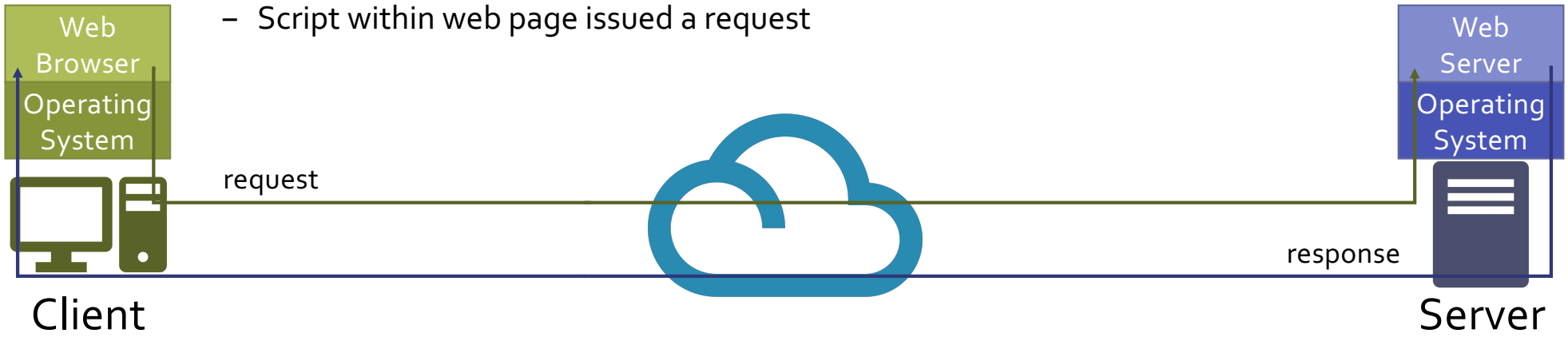
- Oh, yeah, that token does look familiar.
- Here is the result. Please execute this code on your machine.
- Also issue the following commands to these other servers and then execute the code that they give you.
- Or not. What do I care.

Web Architecture

- Web browser issues requests
- Web server responds
- Web browser renders response

Web Architecture

- Web browser issues requests. How? Why?
  - User typed in URL
  - User re-loaded a page
  - User clicked on a link
  - Web server responded with a redirect
  - Web page embedded another page
  - Script within web page issued a request



Web Architecture

- Web server responds. How?
  - Returns static file
  - Invokes a script and returns output
  - Invokes a plugin

Web Architecture

- Web browser renders response. How?
  - Renders HTML + CSS
  - Executes embedded JavaScript
  - Invokes a plugin

Web Architecture

- Web sites are **programs**
- Partially executed on the client side
  - HTML rendering, JavaScript, plug-ins (e.g. Java, Flash)
- Partially executed on the server side
  - CGI, PHP, Ruby, ASP, server-side JavaScript, SQL, etc.

Web Security Model

- Is the web browser trusted by the web server?
- Is the web server trusted by the web browser?
- Is the user trusted by the web server?
- Is the user trusted by the web browser?
- Is the web browser trusted by the user?
- Is the web server trusted by the user?

HTTP Basics

- HyperText Transfer Protocol
- Web client and server communicate using ASCII text messages
  - Client: "`GET https://cseweb.ucsd.edu/classes/sp18/cse127-a/`" …
  - Server: "`HTTP/1.1 200 OK`" …
- Stateless protocol
  - No innate notion of session (and hence no built-in login)
  - Each request/response pair is independent

HTTP Basics

- Client sends requests
  - Examples:
    - `GET`: retrieve a resource
    - `POST`: update a resource (submit a form, publish a post, etc.)
- Server responds
  - Status + optional body
  - Status examples:
    - `200: OK`
    - `303: See other` (redirect)
    - `404: Not found`

HTTP Basics

Secure | https://cseweb.ucsd.edu/classes/sp18/cse127-a/

CSE 127: Computer Security

- Home
- Syllabus
- Details
- Assignments

CSE 127

Schedule: Class (TTh 3:30-4:50 PCYNH 106), Discussion (M 5-5:50pm PCYNH 106)

Instructor: Stefan Savage
Office hours: M 4pm-5pm or by appt, CSE 3106

Teaching Assistants:

- Brown Farinholt
- Tejash Desai
- Haikuo Yin
- Krishna Kalubandi

Teaching Assistant office hours:

- Mondays 6:30PM-7:20PM (B240A)
- Wednesdays 1PM-1:50PM (B240A)
- Wednesdays 5PM-5:50PM (B215)
- Thursdays 1PM-1:50PM (B215)
- Fridays 3PM-3:50PM (B215)

Description

This course focuses on computer and network security, covering a wide range of topics on both the "defensive" and "offensive" side of this field. Among these will be code security and exploitation (buffer overflows, race conditions, SQL injection, etc), access control and authentication, covert channels, protocol attacks, firewalls, intrusion detection/prevension, viruses/worms and bots, spyware and phising, denial-of-service, privacy/anonymity, and computer forensics. The goal of the course is to provide an appreciation of how to think adversarially with respect to computer systems as well as an appreciation of how to reason about attacks and defenses.

To complete the projects in this course, you will need the ability to develop software programs using the C language, and some understanding of Assembly, PHP and SQL. We will not reach these in class and you will be expected to learn them on your own. If you do not know C, I recommend the

Elements | Console | Sources | Network | Performance | Memory | Application | Security | Audits | 308

View: | Group by frame | Preserve log | Disable cache | Offline | Online

Filter | Hide data URLs | All | XHR | JS | CSS | Img | Media | Font | Doc | WS | Manifest | Other

100 ms | 200 ms | 300 ms | 400 ms | 500 ms | 600 ms | 700 ms | 800 ms | 900 ms | 1000 ms | 1100 ms | 1200 ms

Name	Status	Type	Initiator	Size	Time	Waterfall 1.00 s
cse127-a/	200	document	Other	2.4 KB	13 ms	
emx_nav_left.css	200	stylesheet	(index)	(from memo...	0 ms	
tl_curve_white.gif	200	gif	(index)	(from memo...	0 ms	
tr_curve_white.gif	200	gif	(index)	(from memo...	0 ms	
favicon.ico	(canceled)		Other	0 B	28 ms	
cse.ucsd.edu	(blocked:mi...		Other	0 B	0 ms	

Web Sessions

- HTTP is a stateless protocol. No notion of session.
- But most web applications are session-based.
  - Session active until users logs out (or times out).
- How?
  - Cookies.

Web Cookies

- The web server provides a token in its response to the web browser.
  - `Set-Cookie: <cookie-name>=<cookie-value>`
- Attached to every subsequent request to that web server
- Cookie examples (good and bad)
  - UserID
  - SessionID
  - isAuthenticated
  - Preferences
  - Shopping cart contents
  - Shopping cart prices
  - DebugEnable

Web Cookies

- Session Cookies:
  - Expiration property not set
  - Exist only during current browser session
  - Deleted when browser is shut down*
  - *Unless you configured your browser to resume active sessions on re-start
- Persistent Cookies
  - Saved until server-defined expiration time

Web Cookies

- Additional Properties:
  - Domain: hosts to which cookie to be sent
  - Path: request path prefix for which cookie to be sent
  - Secure: to be sent only via HTTPS
  - More later…

Web Cookies

- What's the threat model?
- Who is trusted and who is a potential attacker?
- Does the web browser have to provide cookies?
- How hard is it for a user to modify their cookies?

HTML Basics

- Web pages are HTML + CSS + JavaScript + plugin objects
- HTML: HyperText Markup Language
  - Text with markup and hyperlinks
- CSS: Cascading Style Sheets
- JavaScript
  - Client-side program
- Plugin Objects
  - Examples: custom video decoders, Java, Flash, etc.

view-source:https://cseweb.ucsd.edu/classes/sp18/cse127-a/

```
<div id="pagecell1">
  <!--pagecell1-->
  <img alt="" src="tl_curve_white.gif" height="6" width="6" id="tl"> <img alt="" src="tr_curve_white.gif" height="6" width="6" id="tr">
  <div id="pageName">
    <h2>CSE 127: Computer Security </h2>

  </div>
  <div id="pageNav">
    <div id="sectionLinks">
      <p><a href="index.html">Home</a><a href="syllabus.html">Syllabus</a><a href="details.html">Details</a>  <a href="assignments.html">Assignments</a> </p>
    </div>
  </div>
  <div id="content">
    <div class="feature">

    <h3>CSE 127</h3>
    <p><strong>Schedule:</strong> Class (TTh 3:30-4:50  PCYNH 106),   Discussion (M 5-5:50pm PCYNH 106)<br>
<p><strong>Instructor:</strong> <a href="http://www.cs.ucsd.edu/~savage">Stefan Savage</a><br>
    <strong>Office hours:</strong> M 4pm-5pm or by appt, CSE 3106<br>
<p>
<span class="style1">Teaching Assistants:</span> <br>
<ul>
  <li><a href="mailto:bfarinho@cs.ucsd.edu">Brown Farinholt</a><br></li>
  <li><a href="mailto:tvdesai@ucsd.edu">Tejash Desai</a><br></li>
  <li><a href="mailto:h3yin@ucsd.edu">Haikuo Yin</a><br></li>
  <li><a href="mailto:vkaluban@ucsd.edu">Krishna Kalubandi</a><br></li>
</ul>

<span class="style1">Teaching Assistant office hours:</span> <br>
<ul>
  <li>Mondays 6:30PM-7:20PM (B240A)</li>
  <li>Wednesdays 1PM-1:50PM (B240A)</li>
  <li>Wednesdays 5PM-5:50PM (B215)</li>
  <li>Thursdays 1PM-1:50PM (B215)</li>
  <li>Fridays 3PM-3:50PM (B215)</li>
</ul>

    <h2>Description </h2>
    <p align="left" class="feature style3">This course focuses on
    computer and network security, covering a wide range of topics on
    both the "defensive" and "offensive" side of this field. Among
    these will be code security and exploitation (buffer overflows,
    race conditions, SQL injection, etc), access control and
    authentication, covert channels, protocol attacks, firewalls,
    intrusion detection/prevension, viruses/worms and bots, spyware
    and phising, denial-of-service, privacy/anonymity, and computer
    forensics. The goal of the course is to provide an appreciation of
    how to think adversarially with respect to computer systems as well as
    an appreciation of how to reason about attacks and defenses.
```

Web Server

- Serving static content
- Generating dynamic content
  - CGI
    - PHP, Perl, python
  - Web server modules
    - ASP, Rails, etc.
  - Database backend
    - SQL

URL Query String

- A way of passing name=value argument pairs to the server
- Part of the URL following the '?'
- Example

https://www.google.com/search?q=foobar&ie=utf-8&oe=utf-8&client=firefox-b-1

URL Query String

- Query string examples (good and bad)
  - Search terms
  - UserID
  - SessionID
  - isAuthenticated
  - Preferences
  - Shopping cart contents
  - Shopping cart prices
  - DebugEnable

Attacking Web Servers

- Threat model:
  - Any request/connection is potentially malicious.
  - Must protect confidentiality, integrity, and availability of server data from unauthorized compromise.
    - User-specific data must be protected from other users
- Untrusted user input handled by the server
  - Request contents
    - URL
    - Query string
    - Form contents
  - Request headers
    - Cookies

Attacking Web Servers

- Many web applications have a database component
  - Users, products, etc.
- The server-side component of such applications interact with the database based on user input
  - Example: Validate supplied username and password

SQL Basics

- Structured Query Language
- Example
  - `SELECT * FROM books WHERE price > 100.00 ORDER BY title`
- Also, be aware:
  - AND, OR, NOT, logical expressions
  - Two dashes (--) indicates a comment (until line end)
  - ; is a statement terminator
-

SQL Injection

- Server-side web **applications generate** SQL queries based on **user input**.
- Example
  - `sql_query = “SELECT count(*) FROM users ” +`
    `“WHERE userName=‘” + request.getParameter(“user”) + “’” +`
    `“ AND userPass=‘” + request.getParaemeter(“pass”) + “’”;`
- What’s the problem?
  - Once again, mixing code and data
  - ***Command Injection***

SQL Injection

- Example
  - sql_query = “SELECT count(*) FROM users ” +
    “WHERE userName=‘” + request.getParameter(“user”) + “’” +
    “ AND userPass=‘” + request.getParaemeter(“pass”) + “’”;
- What if
  - userName is ”alice’ ;--”?
  - sql_query = “SELECT count(*) FROM users”
    “WHERE userName=‘alice’ ;--” + “’”
    “ AND userPass= ” + request.getParaemeter(“pass”) + “’”;

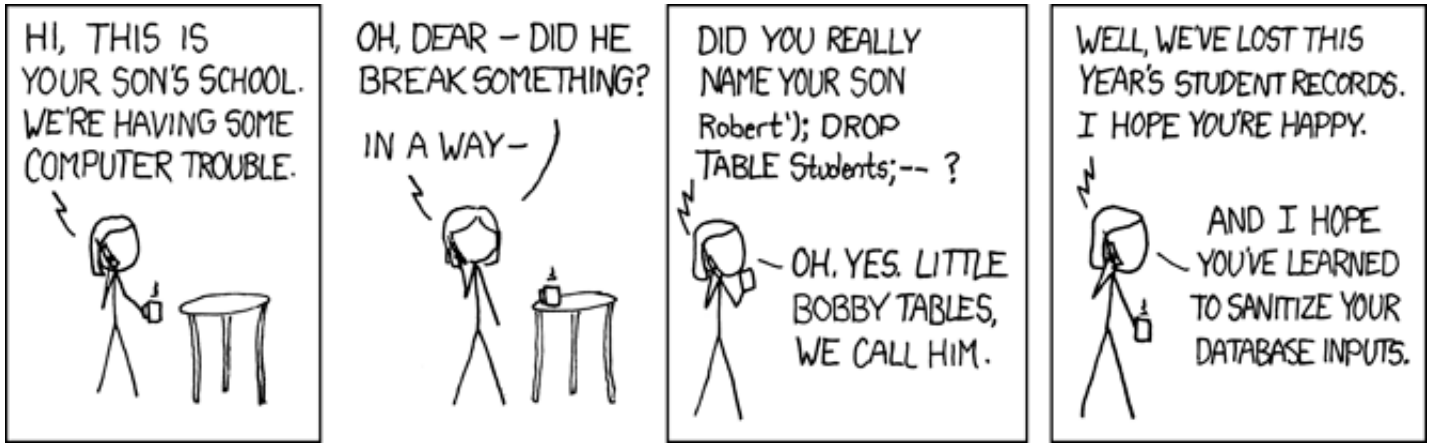
SQL Injection

- Example
  - sql_query = “SELECT count(*) FROM users ” +
    “WHERE userName=‘” + request.getParameter(“user”) + “’” +
    “ AND userPass=‘” + request.getParaemeter(“pass”) + “’”;
- What if
  - userName is ”alice”
  - userPass is ”foo’ OR 1=1 --”?
  - sql_query = “SELECT count(*) FROM users”
    “WHERE userName=‘alice’ AND userPass=‘foo’ OR 1=1 --’”;

SQL Injection

- Not just for bypassing validation
- Potential for executing any SQL command
  - Reading tables
  - Updating tables
  - Deleting tables: drop tables
  - Executing shell commands : exec xp_cmdshell('foo.exe')

Little Bobby Tables



https://xkcd.com/327/

Preventing SQL Injection

- Input sanitization
- Parameterized queries (prepared statements)
- Web Application Firewalls

Preventing SQL Injection

- Input sanitization: make sure only safe (sanitized) input is accepted
- What is unsafe?
  - Single quotes? Spaces? Dashes?
  - All could be part of legitimate input values
- Use proper escaping/encoding
  - How hard could it be? Just add '/' before ''
  - Most languages have libraries for escaping SQL strings
- Should you sanitize on the client or on the server?

Aside: Canonicalization

- Frequently input is encoded into url:
  - http://website.com/products.asp?user=savage

- Can still encode spaces, escapes, etc
  - E.g., '-> `%27`, space -> `%20`, = -> `%3D`
  - http://website.com/products.asp?user=crud%27%20OR%201%3D1%20-
  - Lots of different ways…

Preventing SQL Injection

- Note that perfect quoting doesn't entirely solve problem, because problem is lack of typing:
  SELECT fields from TABLE where id= **52 OR 1=1**
- Parameterized queries (prepared statements)
  - Pre-compiled queries that separates commands from input
- Example (C#)

```
command.CommandText =
      "SELECT count(*) FROM users WHERE userName = @name AND userPass = @pass";
command.Parameters.AddWithValue("@name", username);
command.Parameters.AddWithValue("@pass", pass);
```

Attacking Web Servers Indirectly

- Clients will render HTML provided by the server and execute embedded JavaScript code within the browser sandbox.
- These client actions may involve sending additional requests to this and other servers, based on the contents of server response.
- If the server response can be made to contain malicious content, an adversary may be able to "trick" the unwitting client into sending unintended requests to the victim server.
  - Confused deputy, reflection attacks

Attacking Web Servers Indirectly

- How?
  - Responding server is malicious
  - Response corrupted in transit
  - Responding server is tricked into supplying malicious data

Web Client

- Renders received HTML + CSS …
  - And executes embedded JavaScript …
  - And invokes plug-ins for embedded objects
- May involve resources included by reference
  - Iframes, images, scripts, etc.

JavaScript

- Scripts embedded in web pages
  - Used to implement interactive web applications
  - Example
- Executed by the browser
  - Attackers get to execute code on user's machine, by intent!

```
<html>
      …
<p> The script on this page adds two
  numbers
<script>
    var num1, num2, sum
    num1 = prompt("Enter first number")
    num2 = prompt("Enter second number")
    sum = parseInt(num1) + parseInt(num2)
    alert("Sum = " + sum)
</script>
    …
</html>
```

Browser: Basic Execution Model

- Each browser window or frame:
  - Loads content
  - Renders
  - Responds to events
- Scripts can run
  - before HTML is loaded
  - before page is viewed
  - while it is being viewed
  - when leaving the page
- Events
  - User actions: OnClick, OnMouseover
  - Rendering: OnLoad
  - Timing: setTimeout(), clearTimeout()

Event-Driven Script Execution

Script defines a page-specific function

```
<script type="text/javascript">
     function whichButton(event) {
     if (event.button==1) {
           alert("You clicked the left mouse button!")
  }
     else {
           alert("You clicked the right mouse button!")
     }}
</script>
…
<body onmousedown="whichButton(event)">
…
</body>
```

Function gets executed when some event happens

Other events:
onLoad, onMouseMove, onKeyPress, onUnLoad

JavaScript History

- Scripting language for Navigator 2
  - Designed and implemented in 10 days (literally)
- Related to Java in name only
  - Name was part of a marketing deal
  - "Java is to JavaScript as car is to carpet"
- Became the Web scripting language
  - Google Maps, Hotmail, Facebook, etc

JavaScript in Web Pages

- Embedded in HTML page as `<script>` element
  - JavaScript written directly inside <script> element
    - `<script> alert("Hello World!") </script>`
  - Linked file as src attribute of the `<script>` element
    - `<script type="text/JavaScript" src="functions.js"></script>`
- Event handler attribute
    - `<a href="http://www.yahoo.com" onmouseover="alert('hi');">`
- Pseudo-URL referenced by a link
    - `<a href="JavaScript: alert('You clicked');">Click me</a>`

JavaScript Security Model

- Script runs in a "sandbox"
  - No direct file access, restricted network access
- ***Same-Origin Policy***
  - Can only read properties of documents and windows from the same server, protocol, and port
  - If the same server hosts unrelated sites, scripts from one site can access document properties on the other
- User can grant privileges to signed scripts
  - UniversalBrowserRead/Write, UniversalFileRead, UniversalSendMail

Library Import

- Same-origin policy **does not apply** to scripts loaded in enclosing frame from arbitrary site

```
<script type="text/javascript">
      src="http://www.example.com/scripts/somescript.js">
</script>
```

  - This script runs as if it were loaded from the site that provided the page!

- Server can also explicitly tell browser that other domains should be allowed via "`Access-Control-Allow-Origin`" header

Document Object Model (DOM)

- DOM provides representation of HTML tree hierarchy
- Scripts can read, modify, add elements within the DOM
  - i.e. change read and modify page contents
- Examples
  - Properties: document.alinkColor, document.URL, document.forms[ ], document.links[ ], document.anchors[ ], …
  - Methods: document.write(document.referrer)
    - These change the content of the page!
- Also Browser Object Model (BOM)
  - Window, Document, Frames[], History, Location, Navigator (type and version of browser)

Frame and iFrame

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

```
<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

Remote Scripting

http://developer.apple.com/internet/webcontent/iframe.html

- Goal: exchange data between client-side app in a browser and server-side app (w/o reloading page)
- Methods
  - Java applet or ActiveX control or Flash
    - Can make HTTP requests and interact with client-side JavaScript code, but requires LiveConnect (not available on all browsers)
  - XML-RPC
    - Open, standards-based technology that requires XML-RPC libraries on your server and in client-side code
  - Simple HTTP via a hidden IFRAME
    - IFRAME with a script on your web server (or database of static HTML files) is by far the easiest remote scripting option

HTML Injection

- Many interactive web applications echo user input
  - Examples: search queries, tweets, forum posts, etc.

https://www.google.com/search?q=ajasdhwed

Google

ajasdhwed

All Maps Videos Images Shopping More Settings Tools

Did you mean:

uas dharwad ajay sadhwani aja west

No results containing all your search terms were found.

Your search - **ajasdhwed** - did not match any documents.

Suggestions:

- Make sure all words are spelled correctly.
- Try different keywords.
- Try more general keywords.

HTML Injection

- What if user input contains HTML markup tags?
  - If the server application does not properly sanitize and encode it, the markup will appear verbatim in the response and will be rendered by the web browser
- Similar to SQL injection
  - Command injection
- What if user input contains Javascript?
  - Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS)

- Reflected XSS (non-persistent)
  - Malicious content is injected via URL encoding (query parameters, form submission) and reflected back by the server in the response
- Stored XSS (persistent)
  - Malicious content saved by the server and included in subsequent responses

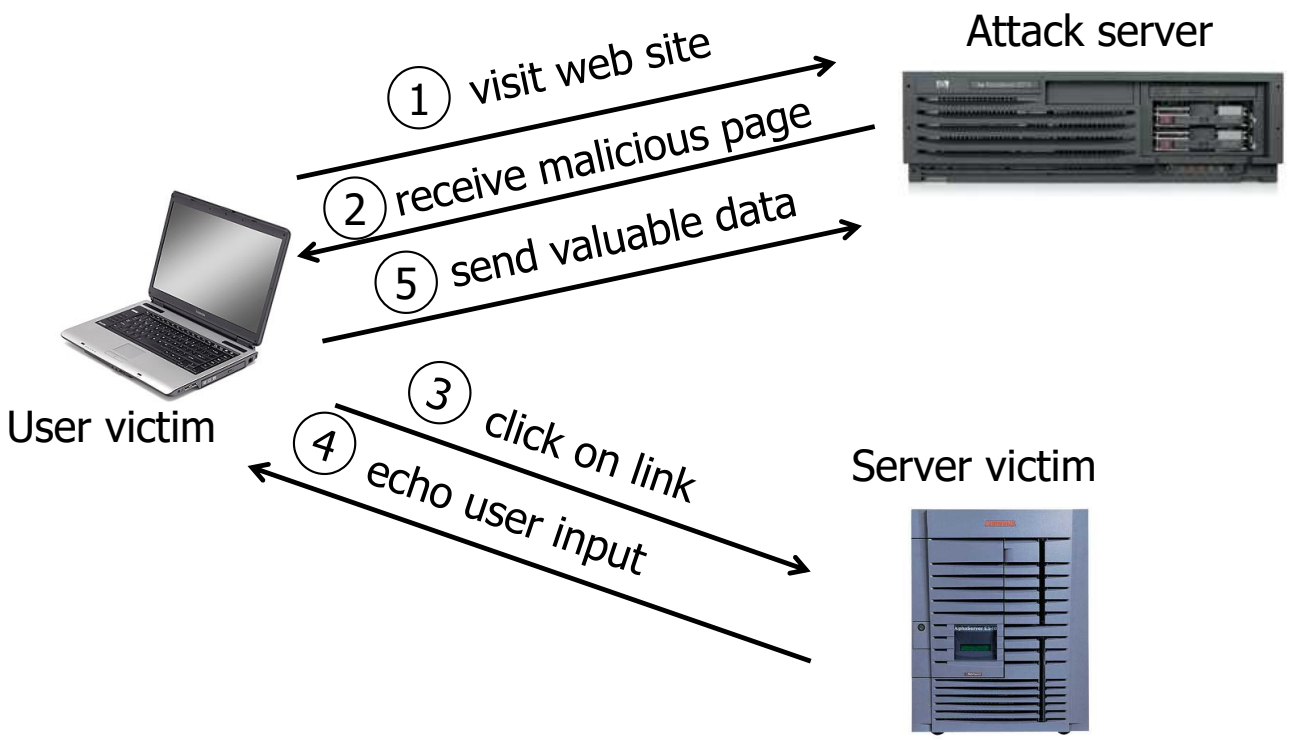
Cross-Site Scripting (XSS)

- So what?
- Anything a legitimate client-side script served by that server could do.
  - Access to the DOM
    - Show bogus information, request sensitive data
    - Control form fields (read or modify entered data)
  - Access to the site's cookies
    - Impersonate user to the site
  - Trigger HTTP requests from the client
    - Use browser to attack other sites

Cross-Site Scripting (XSS)

- Injection vectors
  - Embedded in URL
    - `http://someserver.com/login?URI=“>><script>AttackScript</script>`
    - Use phishing email to drive users to this URL
  - Hidden in user-created content
    - Social sites, blogs, forums, wikis
- XSS is a form of “***reflection attack***”
  - User is tricked into visiting a badly written website
  - A bug in website code causes it to return and the user’s browser to execute an arbitrary attack script
- When user loads the page, web server return the content and visitor’s browser executes script
  - Many sites try to filter out scripts from user content, but this is difficult

Cross site scripting: Basic Pattern



Example: echoing User Input

- Classic mistake in a server-side application

http://naive.com/search.php?term="Beyonce"

search.php responds with

<html> <title>Search results</title>

<body>You have searched for <?php echo $_GET[term] ?>… </body>

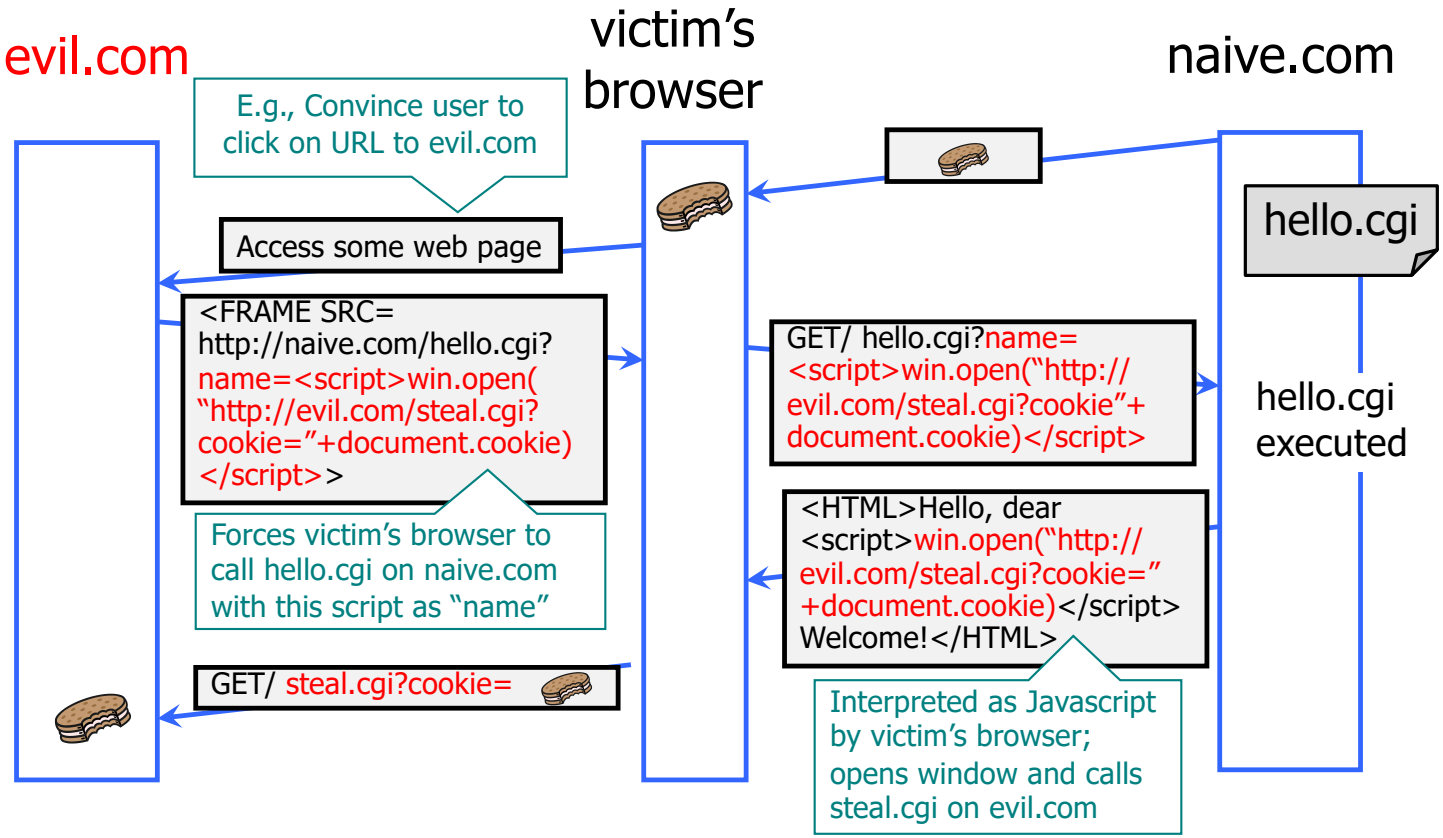
Or

GET/ hello.cgi?name=Bob

hello.cgi responds with

<html>Welcome, Bob</html>

XSS: Cross-Site Scripting



So What?

- Why would user click on such a link?
  - Phishing email in webmail client (e.g., Gmail)
  - Link in banner ad
  - bit.ly/xxxx on twitter
  - … many many ways to fool user into clicking
- So what if evil.com gets cookie for naive.com?
  - Cookie can include session authenticator for naive.com
    - Or other data intended only for naive.com
  - Violates the "intent" of the same-origin policy

Other Sources of Malicious Scripts

- Scripts embedded in webpages
  - Same-origin policy doesn't prohibit embedding of third-party scripts
    - Ad servers, mashups, etc.
- "Bookmarklets"
  - Bookmarked JavaScript URL
    - javascript:alert("Welcome to paradise!")
  - Runs in the context of current loaded page

Samy Worm

- MySpace: largest social networking site in the world 2004-2010
- Users can post HTML on their MySpace pages
- MySpace was sanitizing user input to prevent inclusion of JavaScript
- Samy Kamkar found a way to bypass existing checks and inject JavaScript onto his MySpace page (2005)
  - https://samy.pl/myspace/tech.html

Samy Worm

- Samy Kamkar found a way to bypass existing checks and inject JavaScript onto his MySpace page (2005)
  - https://samy.pl/myspace/
  - *"A Chipotle burrito bol and a few clicks later, anyone who viewed my profile who wasn't already on my friends list would inadvertently add me as a friend. Without their permission. I had conquered myspace. Veni, vidi, vici."*
  - *"If I can become their friend…then why can't their friends become my friend… I can propagate the program to their profile, can't I. If someone views my profile and gets this program added to their profile, that means anyone who views THEIR profile also adds me as a friend and hero, and then anyone who hits THOSE people's profiles add me as a friend and hero…"*

Samy Worm

- **10/04, 12:34 pm**: You have **73** friends.
  I decided to release my little popularity program. I'm going to be famous…among my friends.

- **1 hour later, 1:30 am**: You have **73** friends and **1** friend request.

- **7 hours later, 8:35 am**: You have **74** friends and **221** friend requests.
  Woah. I did not expect this much. I'm surprised it even worked.. 200 people have been infected in 8 hours. That means I'll have 600 new friends added every day. Woah.

- **1 hour later, 9:30 am**: You have **74** friends and **480** friend requests.
  Oh wait, it's exponential, isn't it. Oops.

- **1 hour later, 10:30 am**: You have **518** friends and **561** friend requests.
  Oh no. I'm getting messages from people pissed off that I'm their friend when they didn't add me. I'm also getting emails saying "Hey, how did you get onto my myspace…"

- **3 hours later, 1:30 pm**: You have **2,503** friends and **6,373** friend requests.
  I'm canceling my account. This has gotten out of control. People are messaging me saying they've reported me for "hacking" them due to my name being in their "heroes" list. Man, I rock. Back to my worries. … Apparently people are getting pissed because they delete me from their friends list, view someone else's page or even their own and get re-infected immediately with me. I rule. I hope no one sues me.

https://samy.pl/myspace/

Samy Worm

- **5 hours later, 6:20 pm**: I timidly go to my profile to view the friend requests. **2,503** friends. **917,084** friend requests.
  I refresh three seconds later. **918,268**. I refresh three seconds later. **919,664** (screenshot below). A few minutes later, I refresh. **1,005,831**.
- It's official. I'm popular.

https://samy.pl/myspace/

Samy Worm

- Kamkar was raided by the United States Secret Service
- Kamkar plead guilty to a felony charge of computer hacking in Los Angeles Superior Court.
  - $20,000 fine
  - 3 years of probation
  - 720 hours of community service
  - allowed to keep a single unnetworked computer, but explicitly prohibited from any internet access during his sentence.

https://samy.pl/myspace/

Preventing Cross-Site Scripting

- Key problem: rendering raw HTML from user input
- Preventing injection of scripts into HTML is hard!
  - Blocking "<" and ">" is not enough
  - Event handlers, stylesheets, encoded inputs (%3C), etc.
  - phpBB allowed simple HTML tags like <b>
  - <b c=">" onmouseover="script" x="<b ">Hello<b>
- Any user input must be preprocessed before it is used inside HTML
  - In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    - ' becomes ' " becomes " & becomes &

Partial fix: httpOnly Cookies (IE)

- Cookie sent over HTTP(S), but cannot be accessed by script via document.cookie
- Prevents cookie theft via XSS
- Does not stop most other XSS attacks!

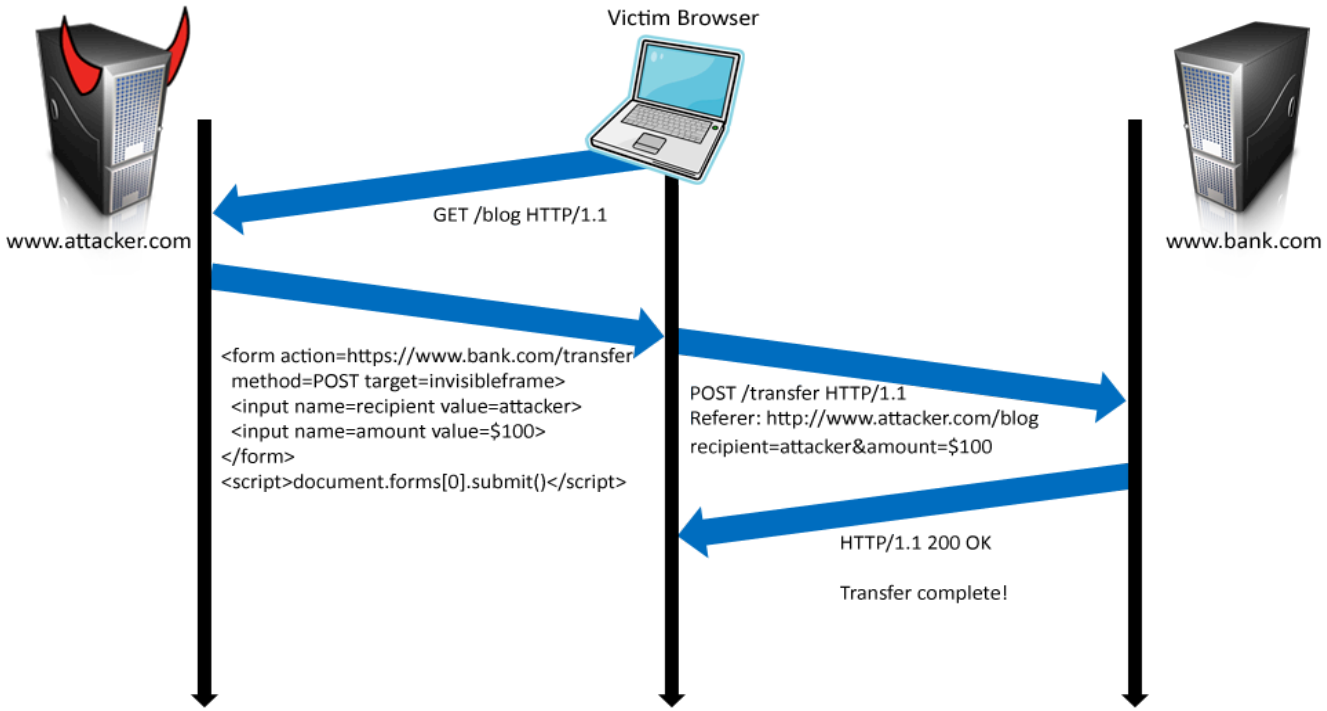
Cross-Site Request Forgery (CSRF)

- When a user's browser issues an HTTP GET request, it attaches all cookies from the target site.
  - If a user clicked on a link
    - What matter is where the link points to (target site), not where the link is located
      - Link could be located on the target web site, in email, on another site, etc.
  - If another page embedded the target page in an iframe
  - If a client-side script issued the request
    - What matter is where the target of the request, not the originator
      - Script could be on any site.
- Only the target site sees the cookies, but…
  - It has no way of knowing if the request was authorized by the user

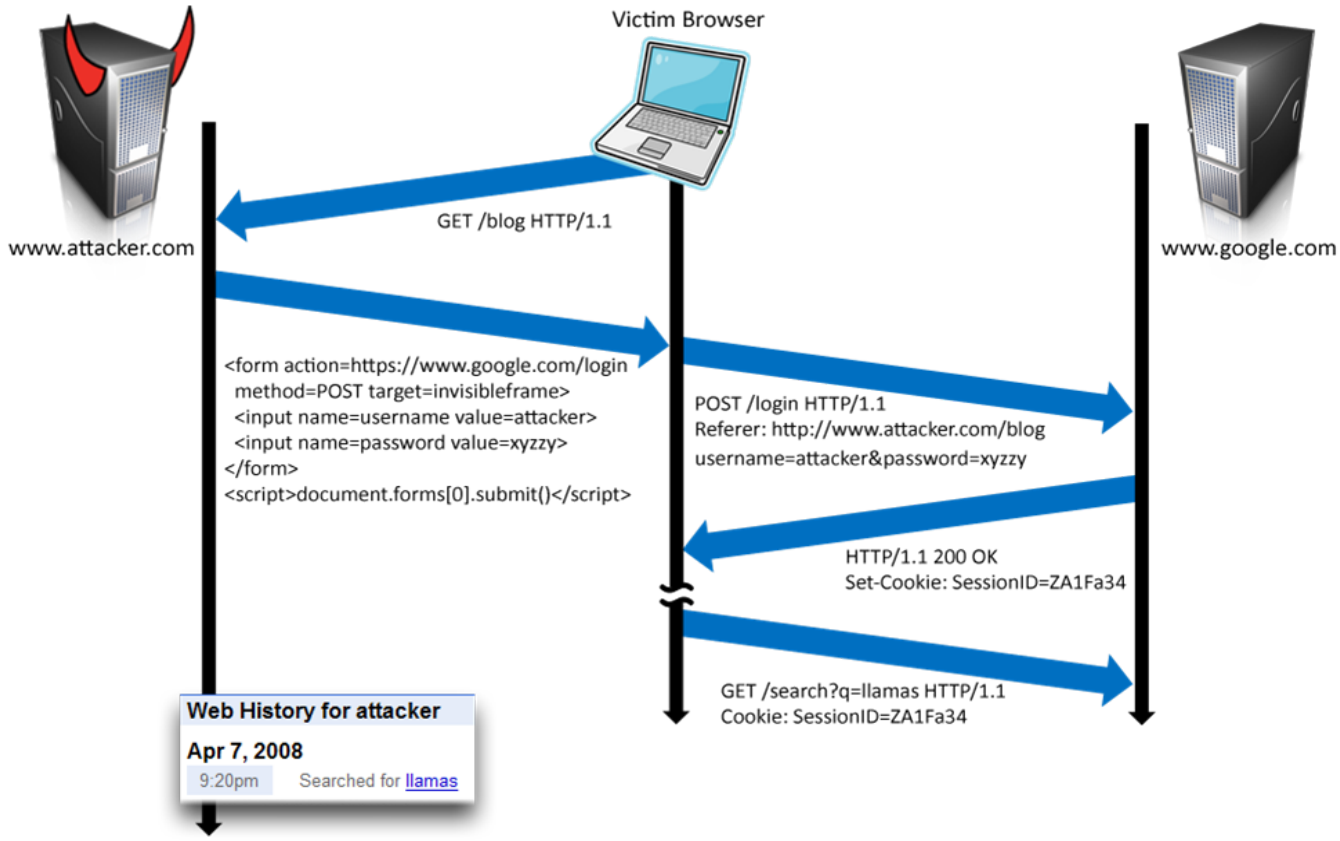
But, Cookies?

- User is signed into bank.com
  - An open session in another tab, or just has not signed off
  - Cookie remain in browser state
- User then visits a malicious website containing
  - `<form name=BillPayForm action=http://bank.com/BillPay.php>`
    `<input name=recipient value=badguy> …`
    `<script> document.BillPayForm.submit(); </script>`
- Browser sends cookie, payment request fulfilled!
- Cookie authentication is not sufficient when side effects can happen

XSRF example



Login XSRF (special case)



Preventing CSRF

- Don't alter state based on GET requests
- Secret Tokens
- Same-origin cookies (Chrome)

Unpredictable Validation Token

- Server injects secret, unpredictable tokens into each response, to be included in subsequent state-altering requests.
- Example
  - `<input type=hidden value=23a3af01b>`
- Only same-origin content can see the token

Unpredictable Validation Token

- Hash of user ID?
  - Can be forged by attacker
- Server has to be able to validate the token
  - Need to bind session ID to the token
  - OWASP CSRFGuard - Manage state table at the server
  - HMAC of session ID – no extra state!

CSRF vs. XSS

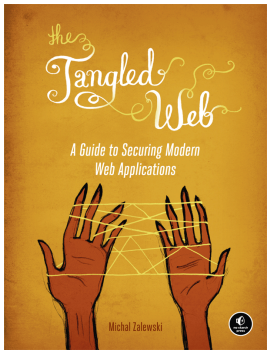
- Cross-site scripting
  - Server-side vulnerability
  - Attacker injects a script into the trusted website
  - User's browser executes attacker's script
- Cross-site request forgery
  - Server-side vulnerability
  - Attacker tricks user's browser into issuing requests
  - Website executes attacker's requests

Clickjacking

- Web page components can overlap.
- Web page components can be transparent.
- Attack:
  - User visits malicious web site.
  - Malicious web site displays a game (or anything else that would compel the user to click on displayed buttons or links).
  - Malicious web site overlays a transparent iframe of a victim site on top of its own content.
  - Victim site is positioned so that a specific UI element is right over a button on the malicious web site.
  - User thinks they are interacting with the game, while they are really acting on the victim site

Review

- SQL injection
  - Bad input checking leads to command injection on the server
- XSS (CSS) – cross-site scripting
  - Echoing untrusted input leads to command injection on the client
- XSRF (CSRF) – cross-site request forgery
  - Forged request leveraging ongoing session
- Clickjacking
  - Transparent UI elements hide real target of input events.

Additional References

- *Open Web Application Security Project (OWASP)*
  - https://www.owasp.org/index.php/Category:Attack
- *Mozilla Developers Network*
  - https://developer.mozilla.org/en-US/
- *Tangled Web, A Guide to Securing Modern Web Applications*
  - by Michal Zalewski
  - https://nostarch.com/tangledweb