XSS

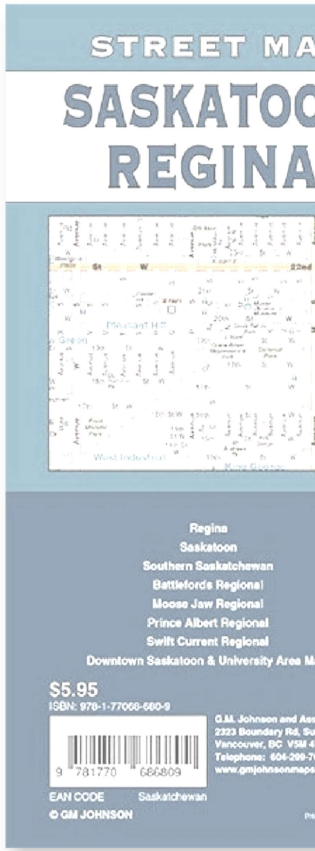
Develop Intelligence

Goals

- Name 2 flavors of XSS attack
- Describe how to mitigate

Roadmap

1. Basics
2. Server-Side XSS
3. DOM-Based XSS



Develop
Intelligence

Basics

What is Cross-Site Scripting

- Attacker sneaks a malicious script into a web app
- Script runs on other users when they run the app*
- Attacker can access
  - Cookies
  - Session tokens
  - Misc Data

What's the Worst that Can Hap

- The malicious script has access to everything on th app
- Arbitrarily manipulate the DOM

Possibly Worse

- Impersonate the user in interacting with Web APIs
  - Access sensitive data
  - Break stuff
- Access HTML5 APIs:
  - Geolocation
  - Webcam / mic

Famous Examples

- Samy broke MySpace in 2005
- Lots more

XSS Flavors

1. **Reflected** - Non-persistent
2. **Self-XSS** - User tricked into pasting malicious code
   console
3. **Server-side** - aka 'persistant' Sneak content into a
4. **DOM-Based** - Exploits SPA vulnerabilities

Reflected XSS

- Old-school
- Usually manipulates a query string
- Popular for phishing
- **Solution**: Don't trust query parameters

Reflected XSS Vulnerability

```
<body>
  <h1>Welcome Home, <span id='greeting'>Loading...</span></h1>
</body>
<script>
  const queryString = location.search;
  const urlParams = new URLSearchParams(queryString);
  const userName = urlParams.get('user');
  document.getElementById('userName').innerHtml= userName || 'Anonymous';
</script>
```

Not Just `<script>` Elements

- Injected script elements are often blocked by the br
- Malformed tags can work better:
  - `<IMG SRC="javascript:alert('pwned');">`
- Also events:
  - `<img src='http://example.com' onload='alert("pwned")'>`

Variation: Self-XSS

Trick user into pasting something into the console

Develop
Intelligence

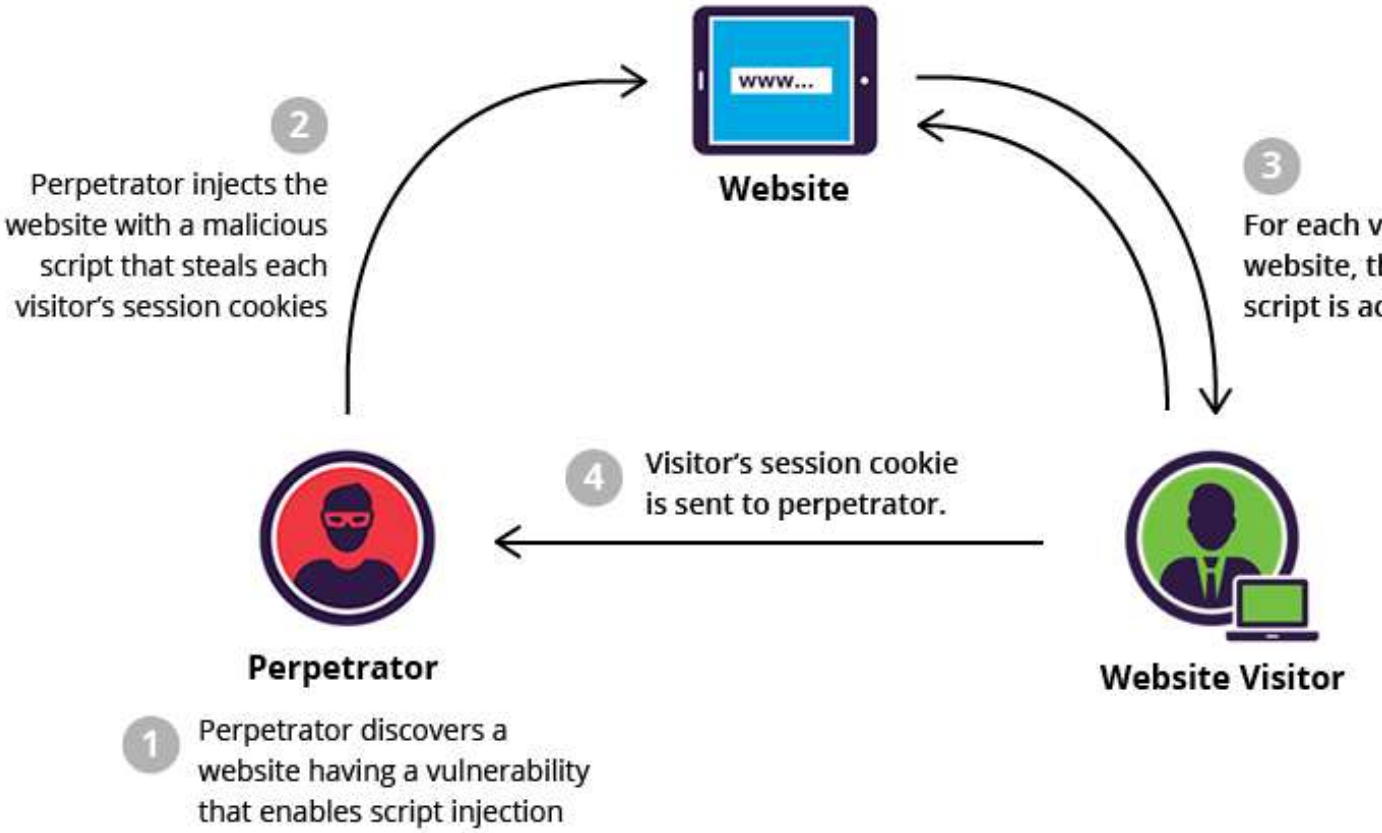
Server-Side XSS

Develop
Intelligence

Details

- Happens with apps containing user-generated cont
  - Forums
  - Content Management Systems
- Most dangerous because the script runs on everyor browser
- SPAs are affected too!

Server-side Example



XSS Mitigation

- Encode output data
- Razor engine does this automatically
- Angular does too
- *But this is still possible to screw up*

Develop
Intelligence

DOM-Based XSS

Develop Intelligence

Basics

- Doesn't involve trip to server
- Exploits vulnerabilities in
  - SPA Frameworks
  - Your app
- Filter potentially scary user input

Angular's XSS Approach

- Trusts no input
- Automatically sanitizes and escapes untrusted valu
- Opt-out with DOMSanitizer

Example: LoginComponent

```
export class LoginInfoComponent  {
  @Input()
  contents='<img src="assets/duck.jpg" onload="alert(\'SN8KED\')">';
}
```

Sanitized: Template Expression

```
<h2>Template Expression:</h2>
<div>{{contents}}</div>
```

Escaped: InnerHTML

- DOMSanitizer runs automatically

```
1 <h2>InnerHTML binding:</h2>
2 <div [innerHTML]='contents'></div>
```

Demo: Bad Sanitation

Use Case For Opting Out

- Dynamic Font Imports

Best Practices

- Keep current with Angular releases
- Don't modify your copy of Angular
- Avoid Angular APIs marked in the documentation as *Risk.*
- Avoid raw DOM manipulation

Develop
Intelligence

Review

- Name 2 flavors of XSS attack
- Describe how to mitigate