XSSfat XSS Guide

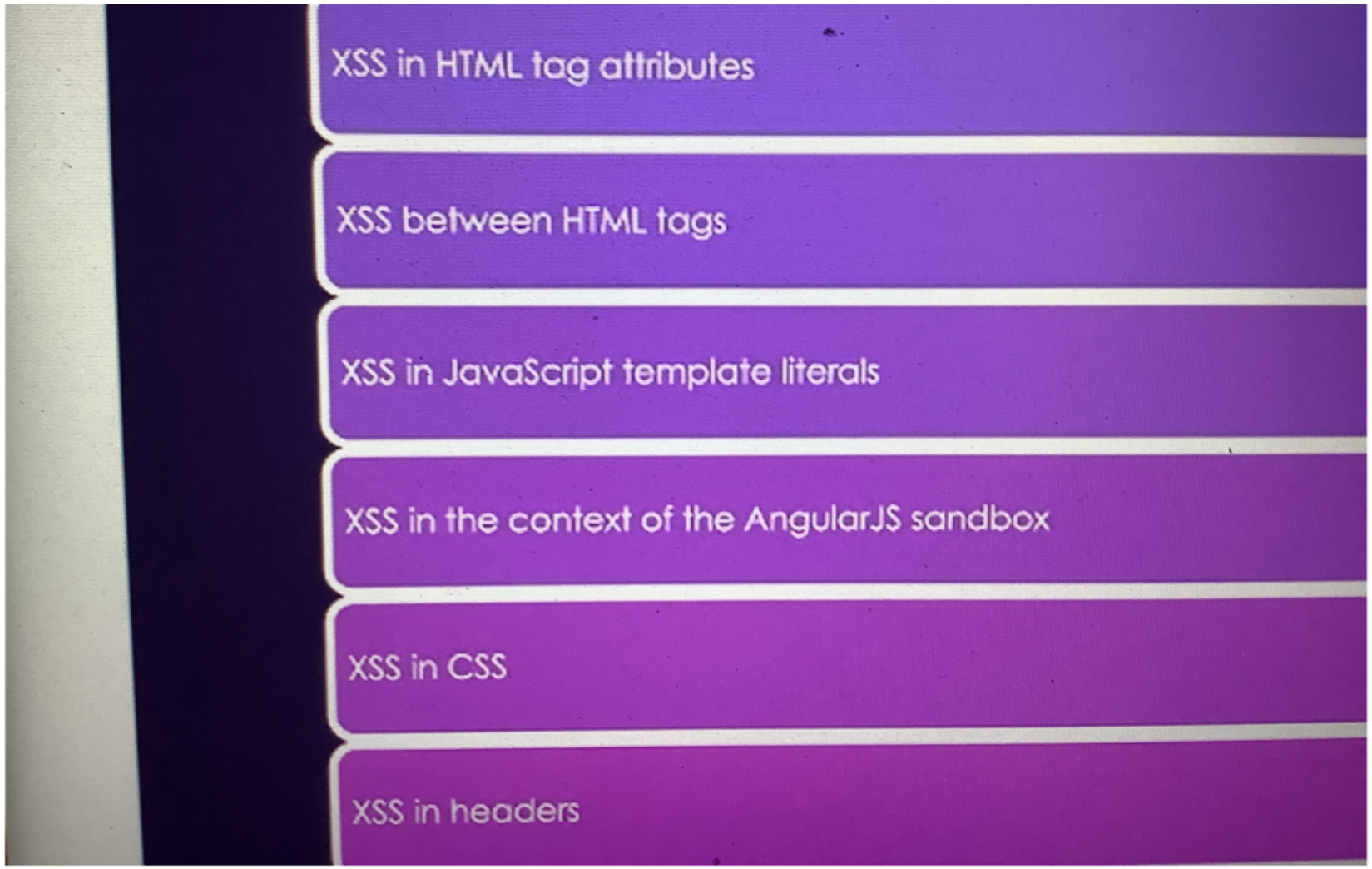
XSS is injecting a "script" or piece of code that will arbitrarily execute on client-side browsers.

Reflected = you send someone a link in order to execute the arbitrary code

Stored = regular clients of the web page will get the arbitrary code executed in their browser upon browsing to the malicious content. includes blind

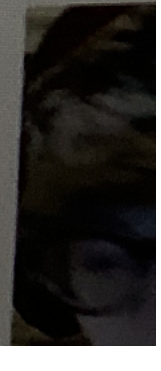
XSS can happen in many other languages than just javascript.

Common Places where XSS can happen:



Breaking out of tag attributes (example)

- `<script>var name= prompt("Please enter your name", "Harry Potter");</script>`
- `...`
- `<input onload="this.value=name;" style="margin-top: 3%">`
- We can
  - Break out of the attribute if our input is not sanitized
  - Insert our own arbitrary JS code
  - Execute XSS
  - `"><script>confirm()</script>`



(if there is no input sanitization)

Always use confirm or prompt instead of Alert.

Alert is one of the most filtered for javascript functions.

Types of XSS - Reflected

- User input gets reflected
  - Into an attribute of an html tag
  - Into the HTML page
  - Into the javascript context
  - ...
- User input **does not** gets stored into database
- User input is not properly sanitized
- User input can contain javascript code

- Zap fuzzer → reflected feature

* Only on Get parameters
* if user input contains Javascript → there is risk!

Stored XSS

Types of XSS - Stored

- User input gets stored into database
- Database value gets reflected
  - Into HTML page
  - Into HTML tag attribute
  - Into the javascript context
  - ...

Types of XSS - Stored

- User input is not sanitized properly
  - At input
  - **And** at write to the database
  - **And** at read from database
- User input can contain malicious javascript code

Stored DOM

Types of XSS - Stored

- var search = document.getElementById('search').value;
- var results = document.getElementById('results');
- results.innerHTML = 'You searched for: ' + search;
  - Results.innerHTML <<< DOM Sink

* use burp scanner for this! almost impossible to locate manually

General ~~55~~ Testing

How to test for XSS - General

- Attack vectors
  - Depend on the context
    - JS: '' ...
      - Single quote, double quote, backtick... to break out of JS function
    - HTML: <img src=x>
      - First test for HTML injection, then expand to XSS
    - HTML attribute: '>">`>...
      - Single quote, double quote, backtick... to break out of html tag attribute
  - All are simple, if they break the page or insert image, look deeper
  - Replace the <img src> tag with your own tag like <script>

* as soon as you see "broken html" look for filters as in your input reflected in the source

* check All input vectors!

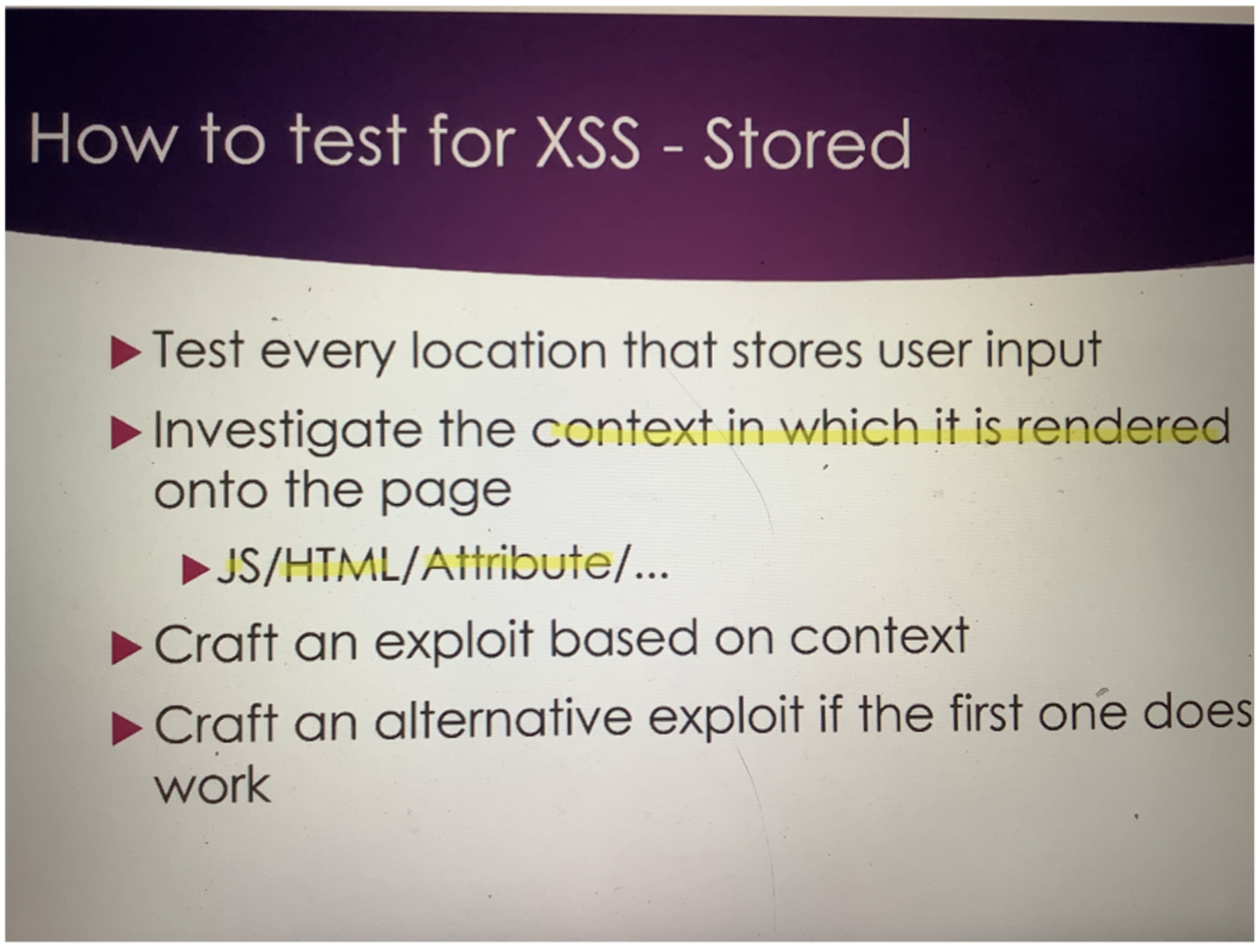
Testing for Reflected

How to test for XSS - Reflected

- Test every entry point
- Submit a random value (ex. Sdfs8f453168)
- Determine the reflection context
  - JS/HTML/Attribute/...
- Test a payload based on the location the value is reflected in
- Test alternative payloads

look at filters, think outside the box & bypass

Testing for Stored



Stored DOM

How to test for XSS - Stored

- Testing HTML sinks
  - Place random value into source
    - I.e. random value into location.search
      - https://www.google.com/submit.htm?email=dsfdfssdfsdfsdfsdfsdfsdfds
      - Location.search would return ?email=dsfdfssdfsdfsdfsdfsdfsdfds
- Inspect the HTML ***using DEVELOPER TOOLS only***
  - View source doesn't take DOM into account

* ONLY Dev tools, source code does not show DOM

* Again burp pro is best for testing LFIs

How to test for XSS - Stored

- Look for your random value in the DOM
- If you string appears in the DOM you need to
  - Identify context
  - Craft exploit based on context
    - I.e. if string enters in double qoute, we might break out by using double quote
- If your data gets URL-encoded before being processed
  - an XSS attack is unlikely to work.

Getting Around Filters

blacklist → (good for hackers)

Getting around filters

- Many different filters out there
- Blacklist based
  - Try fuzzing all possible HTML tags and javascript handlers
  - Try URL encoding XSS attack vector
  - Try double or triple encoding it
  - Try putting blacklisted word into other blacklisted word as filtering might only happen one time leaving our original blacklisted word intact

WAF's

Whitelist + Pattern are hard to get around!

How to bypass a WAF

- Whitelist based
  - Very hard to get around
  - Only allow certain words
  - Block the rest
- Pattern based
  - Tries to look for things that look like a XSS attack
  - Depends on the configuration

Raising Impact