
TESTING
TECHNIQUES

Contents

- XSS
- Clickjacking
- CSRF
- SQLi
- Session Fixation
- Improper Session Invalidation
- Cookie Settings
- User Enumeration
- Information Disclosure
  - Error Messages
  - Server Headers

Testing for XSS

- Place input in all the variables you can
  - Use something sufficiently unique (farm animals)
- See what is reflected back onto the page
- See what characters you can get through
- Figure out what you can do with the space you have and the characters you can get through

Testing for XSS

- If you can inject into the tag but can't escape it (< and > are unavailable) then try and use:
  - ‘ onmouseover=alert(0) [may need to use “]
  - ‘ onload=alert(0) [may need to use “]
- If you can escape the tag (< and > are available) try and use:
  - “”><script>alert(0)</script>
  - “”><img/src/onerror=alert(0)>

Injecting into HTML Tag Parameters

- Sometimes user input ends up inside tags like:
  - `<img src="http://example.com/image.jpeg" alt="[user input]">`
- In this case, if you want to script to execute you need to break out of the tag so you can start a new one
  - Payload: `"><script>alert(0)</script>`
  - Injected tag:
    - `<img src="http://example.com/image.jpeg" alt=""><script>alert(0)</script>">`
    - `<img src="http://example.com/image.jpeg" alt=""> <script>alert(0)</script>`
- What if you cant use the "<" or ">" symbols?
  - You wont be able to break out of the tag, but you may be able to break out of the parameter
  - Payload: `" onmouseover="alert(0)" blah="`
  - Injected tag:
    - `<img src="http://example.com/image.jpeg" alt="" onmouseover="alert(0)" blah="">`
    - `<img src="http://example.com/image.jpeg" alt="" onmouseover="alert(0)" blah="">`

Injecting into Existing JavaScript

- Sometimes user input ends up inside existing JavaScript:

```
<script>
    $(document).ready( function() {
      if(window.location.hash) {
          var dogs = ‘[user input]’;
          document.getElementById(”dogs").innerHTML = dogs;
      }
    });
</script>
```

- In this case, if you want your script to execute you need to insert valid JavaScript so the existing script executes without errors
  - Payload: `‘; alert(0); var a = ‘`
  - Injected line: `var dogs = ‘‘; alert(0); var a = ‘’;`
- How else could you insert alert(0);?

Testing for Clickjacking

- Does the server return the X-Frame-Options header?
- Does the HTML source contain frame-busting JavaScript code?
- Can you embed the page in an iFrame hosted on another domain? (it should not be embeddable)

Testing for CSRF

- Watch the traffic between the browser and server when submitting a form
  - Is there a unique token sent with the form? (should be present)
  - Is the token in the header or the body of the request? (should at least be in the body)
  - Does the request still go through when the token is removed? (request should be blocked)
  - Does the request still go through when the token is altered? (request should be blocked)
- When in doubt, write a proof of concept and see if it works!

Testing for SQLi

- Supply unexpected user input such as ‘ “) -- #
- Identify any error messages or changes in response/behavior
- Determine if your input is being executed as code
- Types of searching:
  - Regular – see if extra data is returned
  - Equivalency – see if statements are executed differently
  - Blind – see if you can cause a backend delay or out-of-band response

Testing for SQLi (text data)

- Does the DB send an error back when it receives a ‘ or “ or) or –
- If you get an error, read it
- Does sending ‘’ (two single ticks) alleviate the error?
- Test to see if the DB does the same thing when you input FOO as it does when you input:
  - ‘||’FOO (Oracle)
  - ‘+’FOO (MS-SQL)
  - ‘ ‘FOO (space between the single ticks) (MySQL)

Testing for SQLi (numerical data)

- Supply a simple mathematical expression
  - If testing for two supply 1+1 or 3-1
- User a more complicated expression such as:
  - 67-ASCII('A') 67 – 65 = 2
  - 51-ASCII(1) 51 – 49 = 2

Testing for Session Fixation

- Login
- Change the session token (keep the same format)
- Login again
- See if a new token is issued

Testing for Improper Session Invalidation

- Login
- Copy the session token
- Logout
- Put the old session token back
- See if you are logged in

Testing Cookie Settings

- Expiration - when the cookie expires
  - Don’t set! Otherwise it is written to disk
  - If expiration or Max-Age aren’t set it will be deleted when the browser closes
- HttpOnly - whether the cookie is accessible to JS
  - Set to True
- Secure - whether the cookie is sent encrypted
  - Set to True if the site uses SSL/TLS

Testing for User Enumeration

- Error messages
  - Try logging in with several usernames (some valid and some invalid) several times
    - Use invalid passwords
  - See if the error messages change
  - Do error messages indicate whether or not the username used is valid? (it shouldn’t)

Information Disclosure – Server Response Headers

- Server response headers are leaky!

- `[-]  x-powered-by: ASP.NET`
- `[-]  server: Microsoft-IIS/7.0`

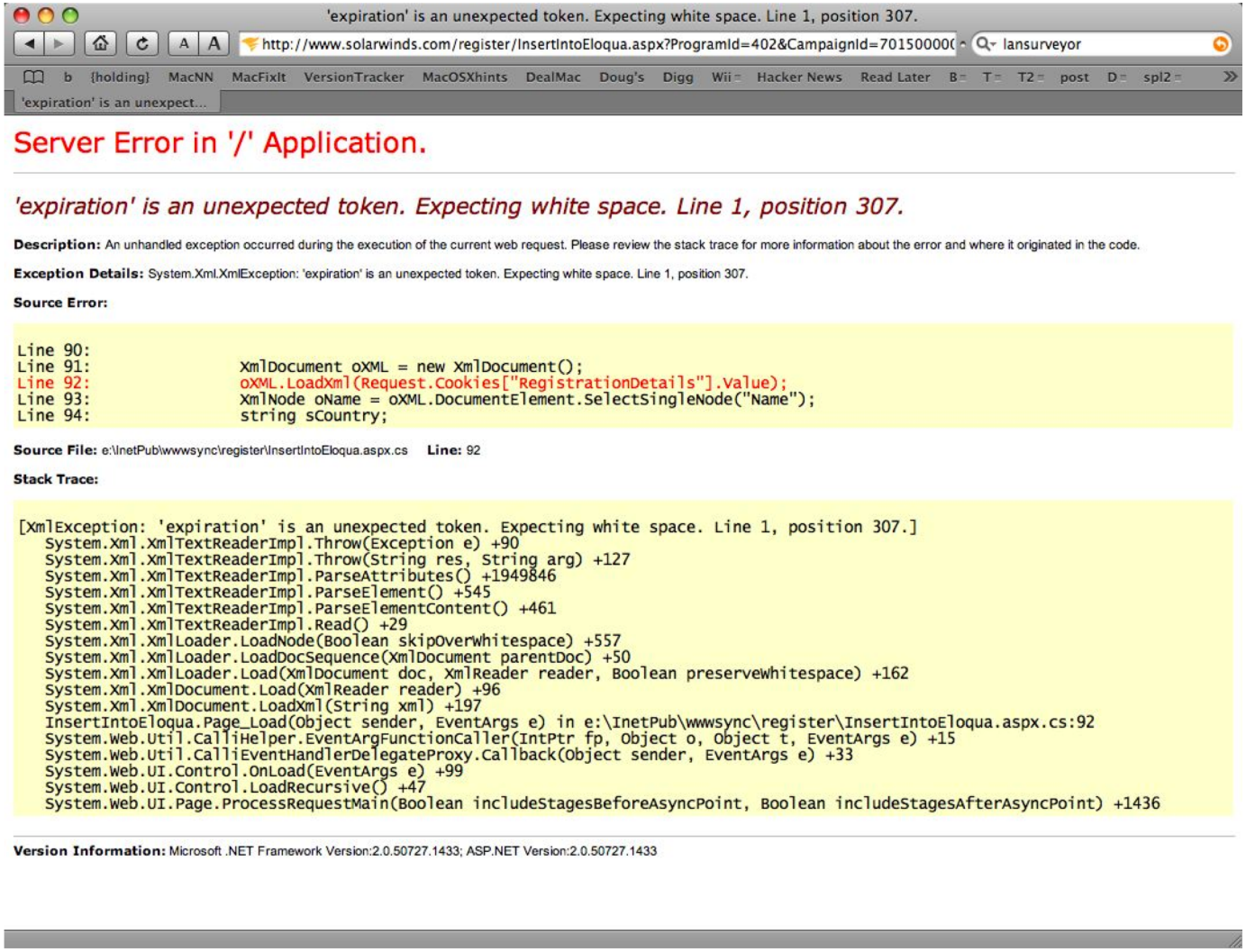
- `[-]  x-aspnet-version: 4.0.30319`
- `[-]  x-powered-by: ASP.NET`
- `[-]  server: Microsoft-IIS/7.5`
- `[-]  x-aspnetmvc-version: 3.0`

- `[-]  x-powered-by: PHP/5.3.27`
- `[-]  server: nginx/1.2.9`

- `[-]  x-powered-by: PHP/5.3.6`
- `[-]  server: Apache/2.2.3 (CentOS)`

- `[-]  x-powered-by: PHP/5.3.3`
- `[-]  server: Apache/2.2.15 (Red Hat) DAV/2 PHP/5.3.3 mod_ssl/2.2.15 OpenSSL/1.0.0-fips mod_perl/2.0.4 Perl/v5.10.1`

Information Disclosure – Error Messages



Information Disclosure – Error Messages