CS390S, Week 11: Web Application Security Issues

Pascal Meunier, Ph.D., M.Sc., CISSP
March 14, 2008
Developed thanks to support and contributions from Symantec Corporation, support from the NSF SFS Capacity Building Program (Award Number 0113725) and the Purdue e-Enterprise Center

http://www.cerias.purdue.edu
Web Application Security Issues

- GET vs POST
- Domain Security
- Cross-site request forgeries (attack type, ID 352)
  - JavaScript Hijacking
- Cross-site Scripting Vulnerabilities (ID 79)
  - Mostly JavaScript Code Injection
  - Disambiguation: JavaScript Injection
- Session mechanism vulnerabilities
  - Session fixation (ID 384)
  - Identification vs authentication
GET and POST Policies

- GET requests (\?variable=value) are not supposed to have any side effects
- POST requests (forms) are used to change things
- Be aware that browsers are supposed to have extra protections for POST requests, so web applications should conform to that usage for better security
  - Example: Redirection
    - A web server is allowed to redirect a GET request anywhere else
    - POST requests are protected by asking the user if this is OK
      - Prevents data from going elsewhere unexpectedly
        » Perhaps the server was compromised?
    - Several web browsers implemented this incorrectly
      - Ilia Alshanetsky
Domain Security

- Browsers need to maintain separations between domains (a.k.a “same origin policy”)
  - On the client, a document or script loaded from example.com shouldn't be able to read or change a document loaded from whatever.com or example.gov
    - Unless whatever.com wants to include a script from example.com
  - However, a script loaded from example.com can send requests or post data to servers at whatever.com or example.gov

- Origin: domain + protocol + port
Domain Security Failures

- DNS race conditions
  - A domain may refer to a different IP address between two requests
    - Browsers are supposed to use “DNS pinning” to prevent this
      - Doesn't always work, e.g., under error conditions

- Weak browser and plugin implementations

- Exploiting browser features
  - Links that change color if they have been visited
  - Caches
Cross-Site Scripting: Outline

- Survey of client-side scripting technologies
- Definition
- Attack Scenarios and Risks
- Security zones
- Examples
- Types of XSS
  - Without storage
  - With storage
  - Indirect (a.k.a. "DOM-based")
    - With or without storage
- Other JavaScript vectors
Client-side Scripting

- **JavaScript Family**
  - ECMAScript (ECMA-262 standard)
    - based on JavaScript 1.1
    - Third edition is now current
  - JavaScript (now at V. 1.5, compatible with ECMA 3rd Ed.)
  - JScript is Microsoft's implementation

- **ActiveX Family**
  - VBScript
    - Requires Internet Explorer on Windows
  - ActiveX controls

- **Java Family**

- **ActionScript (Flash)**
A cross-site scripting vulnerability allows the introduction of malicious content (scripts) on a web site, that is then served to users (clients)
- Malicious scripts get executed on clients that trust the web site
- Problem with potentially all client-side scripting languages

Use “XSS” to refer to these vulnerabilities, to avoid confusion with “CSS” (cascading style sheets)
XSS Concept

- Any way to fool a legitimate web site to send malicious code to a user’s browser
- Almost always involves user content (third party)
  - Error messages
  - User comments
  - Links

References
Attack Scenarios

- Cross-Site Attacks
- Same-Site Attacks
- Browser Exploits
XSS Risks vs. Attack Scenarios

- Theft of account credentials and services
- User tracking (stalking) and statistics
- Misinformation from a trusted site
- Denial of service
- Exploitation of web browser
  - Create phony user interface
  - Exploit a bug in the browser
  - Exploit a bug in a browser extension such as Flash or Java
- Etc.
### Attack Scenarios: 1. Cross-Site Attacks

- You think that you interact with site Z
- Site Z has been poisoned by attacker (Malory)
- The "poison" (e.g., JavaScript) is sent to you, along with legitimate content, and executes. It can exploit browser vulnerabilities, or contact site M and send away your cookies, usernames and passwords...
Stolen Account Credentials

- With XSS, it may be possible for your credentials to be stolen and used by attacker.
- Web sites requiring authentication need to use a technological solution to prevent continuously asking users for passwords.
  - Credentials have the form of a SessionID or nonce:
    - Url encoding (GET method)
      - http://www.site.com?ID=345390027644
    - Cookies are commonly used to store credentials
      - These are usually accessible to client-side scripts
Cookie Mechanism and Vulnerabilities

- Used to store state on the client browser
- Access Control
  - Includes specification of which servers can access the cookie (a basic access control)
    - Including a path on the server
  - So cookie can be used to store secrets (sessionIDs or nonces)
- Side Note: Vulnerabilities in implementations
  - Cross-Domain Cookie Injection Vulnerability in IE 6.0.0, Firefox 0.9.2, Konqueror
XSS -- Point for Cookies

- XSS vulnerabilities bypass the access control mechanism for cookies
- The scripts from the poisoned server take the cookie (and any other data) and send it to the attacker
  - No vulnerabilities needed in the client browser
Privacy Risks

- Scripts can "spy" on what you do
  - Access history of sites visited
  - Track content you post to a web site

- Privacy ("I have nothing to hide")
  - Knowledge about you can be valuable and be used against you
    - Divorces, religion, politics, hobbies, opinions
    - etc...
Attack Scenarios: 2. Same Site Exploit

Hostile Code Executes and Issues Commands As If From the User
Misinformation, Modification and Self-Propagation

- Scripts can misinform
  - Modify the web page you are viewing
    - Scripts can rewrite the body of a page
    - Load different images at any time
    - Create web pages where there were none
  - Modify content that you post
    - e.g., "on submit" event
    - intercept your form submission, replace choices, redirect

- Scripts can take actions in your name and effectively hijack your browser
  - Change your preferences
    - Who is your "hero" (MySpace worm)
  - Self-propagate (add itself to your personalized content)
Same-Site: Phishing

- **User Interface Modifications**
  - Present fake authentication dialogs, capture information, then perhaps redirect user to real web site
  - Replace location toolbar to make user think they are visiting a certain web site

- **Phishing Scenario**
  - Victim logs into a web site
  - Attacker has spread "mines" using an XSS vulnerability
  - Victim stumbles upon an XSS mine
  - Victim gets a message saying that their session has expired, and they need to authenticate again
  - Victim's username and password are sent to attacker
JavaScript can exploit some browser vulnerabilities:

1. Poison
2. Surfing
3. Poison
4. Boom!
Browser Exploit: Other Page Modification

- Cross-frame vulnerabilities, a.k.a. "Frame Injection"
  - A web page can modify a frame presented in another window
    - CVE-2004-0717 to -0721
  - Demo:
- Impact: A malicious script running from one frame (e.g., from a previously visited site with XSS vulnerabilities) can modify subsequently visited sites in another frame
Browser Exploit: Denial of Service

- Nasty JavaScripts can make your web site inaccessible
  - Make browsers crash or become inoperable
  - Redirect browsers to other web sites

- See: "Nasty JavaScript Bombs"
  - http://home1.swipnet.se/~w-26654/javaf.htm
    - See also http://www.cerias.purdue.edu/weblogs/pmeunier/kudos-opinions-rants/post-51/
  - Several scripts implement DoS attacks on browser
    - Need to force-quit or kill browser!
Exploitation of browser vulnerabilities

- JavaScript, ActiveX, etc… allow the exploitation of browser vulnerabilities
  - Run locally on your machine
  - User security confirmation bypass vulnerability in Microsoft Internet Explorer 6.0 SP2:
    - http://securityfocus.com/bid/11200
    - Allows malicious users to trivially bypass the requirement for user confirmation to load JavaScript or ActiveX.
- Installation of malicious code
- Installation of user interfaces
  - Mozilla/FireFox XUL Interface spoofing vulnerability
    - CVE-2004-0764
    - Secunia Advisory SA12188
Defeated Security Zones Model

- **Internet Explorer**
  - Local, Trusted, Internet, Restricted

- **Scenario:**
  - Trusted sites are allowed to run scripts
  - One of the trusted sites has a XSS vulnerability
  - A malicious script is planted on it
  - The script is trusted and run, and so can steal usernames, passwords, session cookies, etc...
    - stolen values can be sent as part of a contacted url (GET: url?v=value)
Defeated Accountability

- Accountability normally restrains the maliciousness of scripts on web sites.
- This is broken by XSS vulnerabilities; there is no limit to the maliciousness of a script.
  - Authors are not accountable because they are unidentified
History of Malicious Scripts

- 2000: Microsoft forced to shut down Hotmail
  - Script intercepted Hotmail authentication cookies and took over users' accounts
    - Javascript forwarded cookies to another site
- 2000: Zkey.com JavaScript exploit
  - XSS vulnerability allowed hacker to capture usernames and passwords
    - Social engineering aspects (phishing); Javascript mimicked the Zkey.com login dialog box ("please re-login")
Other Malicious Scripts

- 2001: Japanese auction web site "Price Loto" disseminated a malicious script that "altered the configuration of users' PCs" (users even had trouble shutting down the computer). The web site closed temporarily.
  - Miyake K., IDG News Service

- 2002: VBScript changes favorites and home page
  - JS.IEStart, a.k.a. FunChina, VBS.Passon (CA), VBS.PassOn (NAV) VBS/IEstart.gen.
  - Alters registry key: HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main\Start Page
VBscripts that change Registry Keys

- 10/2003: QHosts-1 Exploits an Internet Explorer vulnerability
- Creates a new registry key, and modifies 6 others
- Distributed by getting people to visit an infected website
- Performs man-in-the-middle attack on DNS
- Many more examples of scripts changing registry keys
MySpace Worm

- October 2005
- Self-propagating XSS exploit
- Payload: Make Samy your hero
- Script code in a Cascading Style Sheet (CSS)
  - Note: a CSS allows you to specify things such as the font and its size for html tags (H1, TD, etc...) only once for an entire document
Types of XSS Vulnerabilities

- Without storage (reflection)
- Storage
- Indirect
  - Without storage
  - With storage
**XSS Vulnerability: Reflection**

- A vulnerable web site is one that "reflects" or echoes data back to a user
  - No storage needed on the vulnerable web site itself
    ```php
    <?php
      echo $input
    ?>
    
- The attacker creates an html link with some script in it as input to vulnerable web site. This may be in an email, or Malory’s own web site.
  - `<A HREF='http://vulnerable.com?input=<malicious code'>Click here for free stuff!</A>`

- What happens when Alice clicks on the link?
Results

- Alice clicks on link
- Alice is taken to the correct site
- Malory’s code is echoed by the vulnerable site and executed by Alice’s browser in the context of the vulnerable site
  - sends Alice’s cookies, visited urls, etc. to Malory’s computer
- Variations: error or status messages that quote the malicious code
- Example: VBulletin forum
  - CVE-2004-0091
  - http://www.securityfocus.com/archive/1/353673
XSS Vulnerability: Stored

- Malory enters comments or text that contains an embedded script, in a forum, newsgroup, feedback section of a web site, etc...
- The malicious code is stored by the vulnerable site, and presented to visitors. Each instance can be thought of as a "mine".
- Alice reads the comments. Malory’s code is executed on Alice’s computer...
- Example: CVE-2003-1031
  - XSS vulnerability in register.php for vBulletin 3.0 Beta 2 allows remote attackers to inject arbitrary HTML or web script via optional fields such as (1) "Interests-Hobbies", (2) "Biography", or (3) "Occupation."
XSS Vulnerability: Indirect

- The visited web site gives to the browser some JavaScript (or Flash) that is not malicious
  - However, that script is vulnerable to Javascript injection
- The script gets data from elsewhere
  - Reflected: the data comes from the URL
  - Storage: the data comes from another server. That data has been poisoned by Malory
- The script changes the web page as part of its normal duties (e.g., AJAX web application), but includes Malory's script which is also executed!
Javascript Injection Methods

- Trivial: `<script>` tag
- Javascript urls
- Cascading Style Sheets
- Wrapped in some other client-side technology
  - Flash actionscript (really an example of Indirect XSS)
- What else?
JavaScript urls

- JavaScript urls have the format "javascript:code"
  - An example JavaScript url is
    - `javascript:alert("Hello World")`
  - Type it in your browser's address bar, watch the alert window popup
  - Works also in `<A>` HTML links
    - "javascript:alert(document.cookie)"
    - JavaScript urls could be injected into the history list and then executed in the local machine zone or some other zone
      - CVE-2003-1026
      - CVE-2003-0816 (several injection methods)
    - JavaScript url in a frame (Opera <= 6.01; CVE-2002-0783) was executed in the context of other sites
Variation on Indirect Injection

- ActionScript (Flash) can load a JavaScript script from a url
  - Flash objects can be specified with the <embed> tag
    - ActionScript allows the getURL("url") function call
    - The url can be a JavaScript url!
- Forums that allow Flash content are vulnerable
  - People viewing the Flash content get a trojan JavaScript
- See http://www.cgisecurity.com/lib/flash-xss.htm
Web Sessions

- Sessions only identify a client
  - Session_ID = 76543
- Session IDs are given without authentication
- There is no proof of identity (authentication)
- Malory can use the same session ID as Alice
  - But how does Malory know what is Alice's session ID?
Session Fixation

- Malory lays out a trap in the form of a URL or HTTP redirect
  - The trap specifies a session ID
- Alice clicks on it
  - The web server accepts the session ID as valid
    - Should it?
- Alice authenticates (gives user name and password)
- Session status is now "authenticated" and linked to Alice's user name
- Malory can now send commands!
Analysis: Why Does the Attack Work?

- Is it because the client chose the session-ID?
  - No: an attacker can first get a valid session-ID from the real server, and keep it alive as long as necessary (until an attack succeeds)

- The server has no proof that Alice received the original session ID directly, and not through Malory
  - This shared knowledge is the pitfall
  - Resembles a partial Man-In-The-Middle attack
Fix for Session Fixation Attacks

- Remove the possibility of decoupling the identification and authentication

![Diagram showing the decoupling of credentials and session ID](http://www.cerias.purdue.edu)
Authentication Nonce

- Assign a proof of authentication nonce to Alice's browser only and directly in response to a successful authentication request
  - Request valid nonce for every request afterwards
Authentication Nonce

- Can replace the session_ID or
- Can be used in addition to the session_ID
- Should be strong
  - Long
  - Random characters or numbers
  - Goal: make it astronomically unlikely for an attacker to guess, or to stumble upon a valid nonce
- Should expire
  - When session expires
  - When user logs out
Conclusion

- It is easier to keep session IDs separate from authentication nonces to avoid mistakes.
- session IDs are useless for authentication purposes:
  - Unless replaced by an authentication nonce.
  - Session mechanisms may issue weak session IDs.
- If your web application doesn't provide any unauthenticated access, it doesn't need a session ID; it only needs an authentication nonce.
Cross-site Request Forgeries (a.k.a. XSRF)

- **Scenario:** You login on your favorite web site/application
  - Your browser keeps the cookie that proves you authenticated

- **Later, you do either of these two things:**
  - Visit another site that references your favorite site with, for example, an img tag
    - `<img src=http://www.victim.com?operation=nasty_stuff>`
  - Visit a page in your favorite site but that contains a javascript written by another user
    - Javascript sends nasty commands in your name
    - This exploits a Javascript injection vulnerability (a.k.a. cross-site scripting vulnerability)
Fix for Cross-site Request Forgeries (XSRF)

- CSRFs can be foiled by session-specific or user-specific nonces (large unguessable secrets)
- A CSRF mine can't contain valid nonces for all cases
- URL that processes form input checks that the correct nonce is present for the user or session
- XSS vulnerabilities bypass this mechanism
  - JavaScript can get a valid nonce to submit along with the malicious commands
  - No fix for CSRF if your site has a XSS!
Question

- Why can't you use the HTTP-referrer instead of this nonce mechanism, to check if the form was submitted from your web site?
  - Note: the referrer field is sent by the client to the server, with the address of the last visited site
How JavaScript Could Defeat XSRF Protections

- If a script was allowed to send a request and get a response, that response would include the XSRF nonce
  - The script could then send another message, using the XSRF nonce and defeating the protection!
    - Why a XSS vulnerability means you also have a XSRF

- XmlHttpRequest
  - Allows a script to make a request and read the response
    - Make very dynamic web site
    - Crucial part of AJAX technology
  - Restricted to the same site from which the script came
    - AJAX and mash-up developers hate this restriction
      - Always trying to find ways around it (iframes, etc...)
    - Note that whenever this protection is defeated, one should be careful about the risk of XSRF attacks
JavaScript Hijacking (Chess et al. 2007)

- Extension of the XSRF Attack
- Instead of causing something to happen, you can read data with XmlHttpRequest and JSON (JavaScript Object Notation)
- Imagine a victim server that returns an array of confidential data, in JSON format

Steps:
- Your script overrides the “Object” constructor
- Your script makes a request to a victim server
  - request is enclosed inside “<script>” tags and JSON is executed
- During the execution of the JSON, your constructor is called, and your script gets access to the data from the script from the other domain
Cross Domain Flash Attacks

- Flash Web APIs need to allow cross-domain access
  - Scenario: You are a big merchant allowing smaller ones to use your services. Scripts from affiliate web sites must be able to contact your server
- "cross-domain.xml" is a server file that lists allowed domains
  - sometimes set to "*" (all)
- And the web API is at the same domain as your customer store
- Vulnerability: ANY web site with a XSS vulnerability could contain exploits against your customers!
Cross-Domain Flash Attacks

- Possible Fixes:
  - Use separate domains for your API and your store
  - Or be more restrictive
  - Or use something else than Flash

- See Chris Shiflett's discussion:
  - http://shiflett.org/archive/263

- Will this problem become generalized with the W3C draft on Access Control for Cross-site Requests?
  - Same idea but for XMLHttpRequests
    - Noted by Justin Schuh
    - Can you trust web servers to be configured properly?
      - Fat chance
Questions or Comments?