Changing Gears

Continuing with Software Security

- New attacks and countermeasures:
  - SQL injection
  - Background on web architectures

Getting insane with [Input sanitization] drop table slides

SQL injection

```php
$result = mysql_query("select * from Users
where(name='\'frank\' OR 1=1); --");
```

```php
$result = mysql_query("select * from Users
where(name='frank' OR 1=1); --
and password='whocares');");
```
Can chain together statements with semicolon:

```
STATEMENT 1 ; STATEMENT 2
```

SQL injection countermeasures

- **Blacklisting:** Delete the characters you don’t want
  - 
  - `;`
  - `--`
  - `;`

- **Downside:** “Peter O’Connor”
  - You want these characters sometimes!
  - How do you know if/when the characters are bad?
SQL injection countermeasures

1. Whitelisting

- Check that the user-provided input is in some set of values known to be safe
  - Integer within the right range

- Given an invalid input, better to reject than to fix
  - “Fixes” may introduce vulnerabilities
  - Principle of fail-safe defaults

- Downside:
  - Um.. Names come from a well-known dictionary?

2. Escape characters

- Escape characters that could alter control
  - `' => '\'
  - `; => '\';
  - `- => '\`
  - `\` => `\`

- Hard by hand, but there are many libs & methods
  - magic_quotes_gpc = On
  - mysql_real_escape_string()

- Downside: Sometimes you want these in your SQL!
The underlying issue

```php
$result = mysql_query("select * from Users
where(name='$user' and password='$pass');");
```

- This one string combines the **code** and the **data**
- Similar to buffer overflows:

  *When the boundary between code and data blurs, we open ourselves up to vulnerabilities*
SQL injection countermeasures

3. Prepared statements & bind variables

Key idea: *Decouple* the code and the data

```php
$result = mysql_query("select * from Users where(name='\$user' and password='\$pass');");
```

$db = new mysql("localhost", "user", "pass", "DB");

```php
$statement = $db->prepare("select * from Users where(name=? and password=?);");
```

**Bind variables**

**Decoupling lets us compile now, before binding the data**

```php
$statement->bind_param("ss", $user, $pass);
$statement->execute();
```

**Bind variables are typed**

---

**The underlying issue**

```php
$statement = $db->prepare("select * from Users where(name=? and password=?);");
```

Prepare causes the structure of the tree to be **fixed**

- select / from / where
  - *
  - Users
  - and
    - =
      - name
      - $user
    - =
      - password
      - $pass
Mitigating the impact

- Limit privileges
  - Can limit commands and/or tables a user can access
    - Allow SELECT queries on Orders_Table but not on Creditcards_Table
  - Follow the **principle of least privilege**
  - Incomplete fix, but helpful

- Encrypt sensitive data stored in the database
  - May not need to encrypt Orders_Table
  - But certainly encrypt Creditcards_Table.cc_numbers

Summary

- Blacklisting
- Whitelisting
- Prepared statements and bind variables
- Mitigation
Web security

A very basic web architecture

Browser

Web server

Database

(Client)

(Private) Data

Server

(Much) user data is part of the browser

DB is a separate entity, logically (and often physically)
Interacting with web servers

Get and put resources which are identified by a URL

http://www.cs.umd.edu/~pjk/home.html

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Hostname/server</th>
<th>Path to a resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftp</td>
<td>Hostname/server</td>
<td>Translated to an IP address by DNS (more on this later)</td>
</tr>
<tr>
<td>https</td>
<td>Hostname/server</td>
<td>Translated to an IP address by DNS (more on this later)</td>
</tr>
<tr>
<td>tor</td>
<td>Hostname/server</td>
<td>Translated to an IP address by DNS (more on this later)</td>
</tr>
</tbody>
</table>

Here, the file home.html is static content i.e., a fixed file returned by the server

http://facebook.com/delete.php?f=joe123&w=16

Path to a resource Arguments

Here, the file home.html is dynamic content i.e., the server generates the content on the fly

Basic structure of web traffic

- HyperText Transfer Protocol (HTTP)
  - An “application-layer” protocol for exchanging collections of data
Basic structure of web traffic

Client

Browser

Server

HTTP Request

User clicks

- Requests contain:
  - The URL of the resource the client wishes to obtain
  - Headers describing what the browser can do

- Requests be GET or POST
  - GET: all data is in the URL itself (supposed to have no side-effects)
  - POST: includes the data as separate fields (can have side-effects)

HTTP GET requests

http://www.reddit.com/r/security

User-Agent is typically a browser but it can be wget, JDK, etc.
Referrer URL: the site from which this request was issued.

HTTP POST requests
Posting on Piazza

Explicitly includes data as a part of the request's content
**Basic structure of web traffic**

- **Client**
  - Browser

- **Server**
  - Web server

**User clicks**

- Responses contain:
  - Status code
  - Headers describing what the server provides
  - Data
  - Cookies
    - State it would like the browser to store on the site’s behalf

**HTTP responses**

- **HTTP version**
- **Status code**
- **Reason phrase**

```
HTTP/1.1 200 OK
Date: Tue, 16 Feb 2014 08:20:34 GMT
Server: Apache
Set-Cookie: session-zdnet-production=6bbqca10ccbciaqu11sisac2p3; path=/; domain=zdnet.com
Set-Cookie: zdropdown=MT15ljuMT15lE1Mzp1czp1czp1zDjmNW5YTDkODU1N2QYzMSNGU3M212TRmnW
Set-Cookie: zdropdown=MT15ljuMT15lE1Mzp1czp1czp1zDjmNW5YTDkODU1N2QYzMSNGU3M212TRmnW
Set-Cookie: zdropdown=MT15ljuMT15lE1Mzp1czp1czp1zDjmNW5YTDkODU1N2QYzMSNGU3M212TRmnW
Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=zdnet.com
Set-Cookie: session-zdnet-production=59ob9b7fipnc48bg6de4dvgq11; path=/; domain=zdnet.com
Set-Cookie: user_agent=desktop
Set-Cookie: zdnet_ad_session=f
Set-Cookie: firsttp=0
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
X-UA-Compatible: IE=edge,chrome=1
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 18922
Keep-Alive: timeout=70, max=146
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

<html> ....... </html>
```
HTTP is **stateless**

- The lifetime of an HTTP **session** is typically:
  - Client connects to the server
  - Client issues a request
  - Server responds
  - Client issues a request for something in the response
  - …. repeat ….  
  - Client disconnects

- HTTP has no means of noting “oh this is the same client from that previous session”

  - *With this alone, you’d have to log in at every page load*
Maintaining state across HTTP sessions

- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

Online ordering

socks.com
Order
$5.50

socks.com
Pay
The total cost is $5.50.
Confirm order?
Yes
No
Separate page
Online ordering

What’s presented to the user

```html
<html>
<head> <title>Pay</title> </head>
<body>

<form action="submit_order" method="GET">
The total cost is $5.50. Confirm order?
<input type="hidden" name="price" value="5.50">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">

</form>
</body>
</html>
```

Online ordering

The corresponding backend processing

```java
if(pay == yes && price != NULL) {
    bill_creditcard(price);
    deliver_socks();
} else
    display_transaction_cancelled_page();
```
Online ordering

What’s presented to the user

```html
<html>
<head> <title>Pay</title> </head>
<body>

<form action="submit_order" method="GET">
The total cost is $5.50. Confirm order?
<input type="hidden" name="price" value="0.01">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">

</form>
</body>
</html>
```

Minimizing trust in the client

What’s presented to the user

```html
<html>
<head> <title>Pay</title> </head>
<body>

<form action="submit_order" method="GET">
The total cost is $5.50. Confirm order?
<input type="hidden" name="sid" value="781234">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">

</form>
</body>
</html>
```
Minimizing trust in the client

The corresponding backend processing

```
price = lookup(sid);
if(pay == yes && price != NULL) {
    bill_creditcard(price);
    deliver_socks();
} else
    display_transaction_cancelled_page();
```

We don’t want to pass hidden fields around all the time

Statefulness with Cookies

- Server stores state, indexes it with a cookie
- Send this cookie to the client
- Client stores the cookie and returns it with subsequent queries to that same server
Cookies are key-value pairs

Set-Cookie: key=value; options; ....

Cookies are key-value pairs

Cookies

Browser

Client

(Private) Data

Data

Http/1.1 200 OK
Date: Tue, 18 Feb 2014 08:20:34 GMT
Server: Apache
Set-Cookie: session-zdnet-production=6bhqcAl0cbciagui11sisac2p3; path=/; domain=zdnet.com
Set-Cookie: zdregion=MT5i1juMT15LuE1Mzpcp1zpcjZDjmNWYSYdkODU1NQ2YyM5NGU3M2y1zTRmN
Set-Cookie: zdezone=MT5i1juMT15LuE1Mzpcp1zpcjZDjmNWYSYdkODU1NQ2YyM5NGU3M2y1zTRmN
Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=.zdnet.com
Set-Cookie: session-zdnet-production=sf; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=zdnet.com
Set-Cookie: firsttg=0
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
X-UA-Compatible: IE=edge,chrome=1
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 18922
Keep-Alive: timeout=70, max=146
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

<html> ..... </html>

Semantics

- Store “us” under the key “edition” (think of it like one big hash table)
- This value is no good as of Wed Feb 18...
- This value should only be readable by any domain ending in .zdnet.com
- This should be available to any resource within a subdirectory of /
- Send the cookie to any future requests to <domain>/<path>
Requests with cookies

Subsequent visit

Why use cookies?

- Personalization
  - Let an anonymous user customize your site
  - Store font choice, etc., in the cookie
Why use cookies?

• Tracking users
  • Advertisers want to know your behavior
  • Ideally build a profile *across different websites*
    - Read about iPad on CNN, then see ads on Amazon?!
  • How can an advertiser (A) know what you did on another site (S)?

S shows you an ad from A; A scrapes the referrer URL

Option 1: A maintains a DB, indexed by your IP address

Option 2: A maintains a DB indexed by a *cookie*

**Problem: IP addr change**

- “Third-party cookie”
- Commonly used by large ad networks (doubleclick)

Ad provided by an ad network
Our first time accessing adzerk.net

I visit reddit.com

We are only sharing this cookie with *.adzerk.net; but we are telling them about where we just came from

Later, I go to reddit.com/r/security
Cookies and web authentication

• An extremely common use of cookies is to track users who have already authenticated

• If the user already visited http://website.com/login.html?user=alice&pass=secret with the correct password, then the server associates a “session cookie” with the logged-in user’s info

• Subsequent requests (GET and POST) include the cookie in the request headers and/or as one of the fields: http://website.com/doStuff.html?sid=81asf98as8eak

• The idea is for the server to be able to say “I am talking to the same browser that authenticated Alice earlier.”

Attacks?

Cross-Site Request Forgery (CSRF)
URLs with side-effects

GET requests should have no side-effects, but often do

What happens if the user is logged in with an active session cookie and visits this link?

How could you possibly get a user to visit this link?

Browser automatically visits the URL to obtain what it believes will be an image.
Cross-Site Request Forgery

• **Target:** User who has some sort of account on a vulnerable server where requests from the user’s browser to the server have a *predictable structure*

• **Attack goal:** make requests to the server via the user’s browser that look to the server like the user intended to make them

• **Attacker tools:** ability to get the user to visit a web page under the attacker’s control

• **Key tricks:**
  • Requests to the web server have predictable structure
  • Use of something like `<img src=…>` to force the victim to send it

---

CSRF protections

• **Client-side:**

  Disallow one site to link to another??

  The loss of functionality would be too high

• **Server-side:**

  Referrer URL: Only allow certain actions if the referrer URL is from this site, as well

  Make the request unpredictable; put the cookie into the request, as well

  `http://website.com/doStuff.html?sid=8lasf98as8eak`
How can you steal a session cookie?

- Compromise the user’s machine / browser
- Sniff the network
- DNS cache poisoning
  - Trick the user into thinking you are Facebook
  - The user will send you the cookie

Network-based attacks (more later)

Stealing users’ cookies

For now, we’ll assume this attack model:
- The user is visiting the site they expect
- All interactions are strictly through the browser
Dynamic web pages

- Rather than static HTML, web pages can be expressed as a program, e.g., written in Javascript:

```html
<html><body>
  Hello, <b>
  <script>
    var a = 1;
    var b = 2;
    document.write("world: ", a+b, "</b>" );
  </script>
</body></html>
```

Javascript (no relation to Java)

- Powerful web page programming language
- Scripts are embedded in web pages returned by the web server
- Scripts are executed by the browser. They can:
  - Alter page contents (DOM objects)
  - Track events (mouse clicks, motion, keystrokes)
  - Issue web requests & read replies
  - Maintain persistent connections (AJAX)
  - Read and set cookies
What could go wrong?

• Browsers need to **confine Javascript's power**

• A script on **attacker.com** should not be able to:
  • Alter the layout of a **bank.com** web page
  • Read keystrokes typed by the user while on a **bank.com** web page
  • Read cookies belonging to **bank.com**

Same Origin Policy

• Browsers provide isolation for javascript scripts via the **Same Origin Policy (SOP)**

• Browser associates **web page elements**…
  • Layout, cookies, events

• …with a given **origin**
  • The hostname (**bank.com**) that provided the elements in the first place

• **SOP = only scripts received from a web page's origin have access to the page's elements**
Cookies

Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=.zdnet.com

Semantics

- Store “us” under the key “edition”
- This value is no good as of Wed Feb 18…
- This value should only be readable by any domain ending in .zdnet.com
- This should be available to any resource within a subdirectory of /
- Send the cookie to any future requests to <domain>/<path>