Web Security, con’t

CS 161: Computer Security
Prof. Vern Paxson
TAs: Jethro Beekman, Mobin Javed,
Antonio Lupher, Paul Pearce
& Matthias Vallentin

http://inst.eecs.berkeley.edu/~cs161/

February 26, 2013
Announcements

• **Project #1** is out: due in 3 weeks (**Mon Mar 18**)
  – Even if you wait for most of it, get the VM set up soon!

• **Homework #2** is out, due **Wed Mar 6**
  – Day before the Midterm; meant to help cement understanding of second half of Midterm material
    • Including opportunity for discussion/questions in Section that day

• **Midterm** is **Thu Mar 7, 3:40PM-5PM**
  – Scope is material up through Feb 28 lecture
  – Material = lecture, section, “Notes” as indicated in Readings on the web page
  – You can bring a single (double-sided) sheet of paper for a “cheatsheet” of notes
    • Needs to be readable normally by you
Game Plan

• Complete discussion of injection attacks on web servers …
• … especially SQL injection

• Begin discussion of browser interaction threats:
  – Cookie authentication
  – Cross-site request forgery (CSRF)
  – Drive-by attacks
Defending Against Command Injection

• In principle, can prevent injection attacks by properly sanitizing input sent to web servers
  – Remove or escape meta-characters
  – Easy to get wrong by overlooking a meta-character or escaping subtlety

• Better: avoid using a feature-rich API
  – KISS + defensive programming
  – E.g., use execve() to invoke a desired program, rather than system()
This is the core problem. `system()` provides too much functionality!
- treats arguments passed to it as full shell command

If instead we could just run `grep` directly, no opportunity for attacker to sneak in other shell commands!
/ print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
    char *path = "/usr/bin/grep";
    char *argv[10]; /* room for plenty of args */
    char *envp[1]; /* no room since no env. */
    int argc = 0;

    argv[argc++] = path; /* argv[0] = prog name */
    argv[argc++] = "-e"; /* force regex as pat. */
    argv[argc++] = regex;
    argv[argc++] = "phonebook.txt";
    argv[argc++] = 0;

    envp[0] = 0;

    if ( execve(path, argv, envp) < 0 )
        command_failed(....);
}
/ * print any employees whose name * 
* matches the given regex */
void find_employee(char *regex) 
{
    char *path = "/usr/bin/grep";
    char *argv[10]; /* room for plenty of args */
    char *envp[1]; /* no room since no env. */
    int argc = 0;

    argv[argc++] = path; /* argv[0] = prog name */
    argv[argc++] = "-e"; /* force regex as pat. */
    argv[argc++] = regex;
    argv[argc++] = "phonebook.txt";
    argv[argc++] = 0;

    envp[0] = 0;

    if ( execve(path, argv, envp) < 0 )
        command_failed(.....);
}
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
    char *path = "/usr/bin/grep";
    char *argv[10];/* room for plenty of args */
    char *envp[1]; /* no room since no env. */
    int argc = 0;
    argv[argc++] = path; /* argv[0] = prog name */
    argv[argc++] = "-e"; /* force regex as pat. */
    argv[argc++] = regex;
    argv[argc++] = "phonebook.txt";
    argv[argc++] = 0;
    envp[0] = 0;
    if ( execve(path, argv, envp) < 0 )
        command_failed(.....);
}
/ * print any employees whose name matches the given regex */

void find_employee(char *regex)
{
    char *path = "/usr/bin/grep";
    char *argv[10]; /* room for plenty of args */
    char *envp[1]; /* no room since no env. */
    int argc = 0;

    argv[argc++] = path; /* argv[0] = prog name */
    argv[argc++] = "-e"; /* force regex as pat. */
    argv[argc++] = regex;
    argv[argc++] = "phonebook.txt";
    argv[argc++] = 0;

    envp[0] = 0;

    if (execve(path, argv, envp) < 0) command_failed(...);
}

No matter what weird goop “regex” has in it, it’ll be treated as a single argument to grep; no shell involved
Command Injection in the Real World

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.

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. 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
From the looks of it, however, one our suspects an **SQL injection**, in which the Web site. Markovich also questioned not noticed the hack for six months, a

-May 8, 2009 1:53 PM PDT

**UC Berkeley computers hacked, 160,000 at risk**

by Michelle Meyers

This post was updated at 2:16 p.m. PDT with comment from an outside database security software vendor.

Hackers broke into the University of California at Berkeley's health services center computer and potentially stole the personal information of more than 160,000 students, alumni, and others, the university announced Friday.

At particular risk of identity theft are some 97,000 individuals whose Social Security numbers were accessed in the breach, but it's still unclear whether hackers were able to match up those SSNs with individual names, Shelton Waqqener, UCB's chief technology officer, said in a press conference Friday afternoon.
December 8, 2010, 4:18 PM

‘Operation Payback’ Attacks Fell Visa.com

By ROBERT MACKEY

A message posted on Twitter by a group of Internet activists announcing the start of an attack on Visa’s Web site, in retaliation for the company’s actions against WikiLeaks.

Last Updated | 6:54 p.m. A group of Internet activists took credit for crashing the Visa.com Web site on Wednesday afternoon, hours after they launched a similar attack on MasterCard. The cyber attacks, by activists who call themselves Anonymous, are aimed at punishing companies that have acted to stop the flow of donations to WikiLeaks in recent days.

The group explained that its distributed denial of service attacks — in which they essentially flood Web sites site with traffic to slow them down or knock them offline — were part of a broader effort called Operation Payback, which
Anonymous speaks: the inside story of the HBGary hack

By Peter Bright | Last updated a day ago

The hbgaryfederal.com CMS was susceptible to a kind of attack called SQL injection. In common with other CMSes, the hbgaryfederal.com CMS stores its data in an SQL database, retrieving data from that database with suitable queries. Some queries are fixed—an integral part of the CMS application itself. Others, however, need parameters. For example, a query to retrieve an article from the CMS will generally need a parameter corresponding to the article ID number. These parameters are, in turn, generally passed from the Web front-end to the CMS.

It has been an embarrassing week for security firm HBGary and its HBGary Federal offshoot. HBGary Federal CEO Aaron Barr thought he had unmasked the hacker hordes of Anonymous and was preparing to name and shame those responsible for co-ordinating the group's actions, including the denial-of-service attacks that hit MasterCard, Visa, and other perceived enemies of WikiLeaks late last year.

When Barr told one of those he believed to be an Anonymous ringleader about his forthcoming exposé, the Anonymous response was swift and humiliating. HBGary’s servers were broken into, its e-mails pillaged and published to the world, its data destroyed, and its website defaced. As an added bonus, a second site owned
Structure of Modern Web Services

Browser

URL / Form

command.php?arg1=x&arg2=y

Web server
Structure of Modern Web Services

Web server

Database server

Database query built from x and y

Browser

URL / Form

command.php?arg1=x&arg2=y
Structure of Modern Web Services

Browser

Web server

Custom data corresponding to x & y

Database server
Structure of Modern Web Services

Web page built using custom data
Databases

- Management of groups (tuples) of related values

<table>
<thead>
<tr>
<th>AcctNum</th>
<th>Username</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1199</td>
<td>zuckerberg</td>
<td>7746533.71</td>
</tr>
<tr>
<td>0501</td>
<td>bgates</td>
<td>4412.41</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Databases

• Management of groups (tuples) of related values
• Widely used by web services to track per-user information
• Database runs as separate process to which web server connects
  – Web server sends queries or commands customized by incoming HTTP request
  – Database server returns associated values
  – Database server can instead modify/update values

<table>
<thead>
<tr>
<th>AcctNum</th>
<th>Username</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1199</td>
<td>zuckerberg</td>
<td>7746533.71</td>
</tr>
<tr>
<td>0501</td>
<td>bgates</td>
<td>4412.41</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
SQL

• Widely used database query language
  – (Pronounced “ess-cue-ell” or “sequel”)
• Fetch a set of records (simplified):
  
  SELECT field FROM table WHERE condition

returns the value(s) of the given field in the specified table, for all records where condition is true.

• E.g:

  SELECT Balance FROM Customer
  WHERE Username='bgates'

will return the value 4412.41

<table>
<thead>
<tr>
<th>AcctNum</th>
<th>Username</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1199</td>
<td>zuckerberg</td>
<td>7746533.71</td>
</tr>
<tr>
<td>0501</td>
<td>bgates</td>
<td>4412.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Can add data to the table (or modify):
  INSERT INTO Customer
  VALUES (8477, 'oski', 10.00) -- oski has ten buckaroos
  
  An SQL comment
<table>
<thead>
<tr>
<th>AcctNum</th>
<th>Username</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1199</td>
<td>zuckerberg</td>
<td>7746533.71</td>
</tr>
<tr>
<td>0501</td>
<td>bgates</td>
<td>4412.41</td>
</tr>
<tr>
<td>8477</td>
<td>oski</td>
<td>10.00</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
SQL, con’t

• Can add data to the table (or modify):
  INSERT INTO Customer
  VALUES (8477, 'oski', 10.00) -- oski has ten buckaroos

• Or even delete entire tables:
  DROP Customer

• Semicolons separate commands:
  INSERT INTO Customer VALUES (4433, 'vladimir', 888.99); SELECT AcctNum FROM Customer
  WHERE Username='vladimir'
  returns 4433.
SQL Injection Scenario

• Suppose web server front end stores URL parameter “recipient” in variable $recipient and then builds up a string with the following SQL query:

```sql
$sql = "SELECT AcctNum FROM Customer WHERE Balance < 100 AND Username='$recipient' ";
```

• Query accesses recipient’s account if their balance is < 100.

• Web server will send value of $sql variable to database server to get account #s from database
SQL Injection Scenario

• Suppose web server front end stores URL parameter “recipient” in variable $recipient and then builds up a string with the following SQL query:

```sql
$sql = "SELECT AcctNum FROM Customer WHERE Balance < 100 AND Username='$recipient' ";
```

• So for “?recipient=Bob” the SQL query is:

```sql
"SELECT AcctNum FROM Customer WHERE Balance < 100 AND Username='Bob' ";
```
Parse Tree for SQL Example

SELECT AcctNum FROM Customer
WHERE Balance < 100 AND Username='Bob'
SQL Injection Scenario

• Suppose web server front end stores URL parameter “recipient” in variable $recipient and then builds up a string with the following SQL query:

```
$sql = "SELECT AcctNum FROM Customer
WHERE Balance < 100 AND Username='$recipient' ";
```

• How can $recipient cause trouble here?
  – How can we see anyone’s account?
    • Even if their balance is >= 100
SQL Injection Scenario, con’t

WHERE Balance < 100 AND
    Username='$recipient'

• Conceptual idea (doesn’t quite work): Set recipient to “foo' OR 1=1” …

    WHERE Balance < 100 AND
        Username='foo' OR 1=1'

• Precedence makes this:

    WHERE (Balance < 100 AND
        Username='foo') OR 1=1

• Always true!
SELECT AcctNum FROM Customer
WHERE (Balance < 100 AND Username='foo') OR 1=1
SQL Injection Scenario, con’t

- Why "'foo' OR 1=1" doesn’t quite work:

  WHERE Balance < 100 AND Username='"foo' OR 1=1"

  Syntax error: quotes aren’t balanced
  SQL server will reject command as ill-formed
SQL Injection Scenario, con’t

• Why “foo' OR 1=1” doesn’t quite work:
  WHERE Balance < 100 AND
  Username='foo' OR 1=1'

• Sneaky fix: use “foo' OR 1=1 --”
  Begins SQL comment …
SQL Injection Scenario, con’t

- Why “foo' OR 1=1” doesn’t quite work:
  WHERE Balance < 100 AND 
  Username='foo' OR 1=1'

- Sneaky fix: use “foo' OR 1=1 --”
- SQL server sees:
  WHERE Balance < 100 AND 
  Username='foo' OR 1=1--'

When parsing SQL query, SQL server ignores all of this since it’s a comment …

So now it finds the quotes balanced; no syntax error; **successful injection!**
SQL Injection Scenario, con’t

WHERE Balance < 100 AND Username='$_recipient$_

• How about $recipient = foo'; DROP TABLE Customer -- ?

• Now there are two separate SQL commands, thanks to ‘;’ command-separator.

• Can change database however you wish
Defenses

Language support for constructing queries
Specify query structure independent of user input:
Defenses

Language support for constructing queries
Specify query structure independent of user input:

```java
ResultSet getProfile(Connection conn, String arg_user) {
    String query = "SELECT AcctNum FROM Customer WHERE Balance < 100 AND Username = ?";
    PreparedStatement p = conn.prepareStatement(query);
    p.setString(1, arg_user);
    return p.executeQuery();
}
```

“Prepared Statement”
Defenses

Language support for constructing queries
Specify query structure independent of user input:

```java
ResultSet getProfile(Connection conn, String arg_user) {
    String query = "SELECT AcctNum FROM Customer 
                    WHERE Balance < 100 AND Username = ?";
    PreparedStatement p = conn.prepareStatement(query);
    p.setString(1, arg_user);
    return p.executeQuery();
}
```

Untrusted user input
Defenses

Language support for constructing queries
Specify query structure independent of user input:

```java
ResultSet getProfile(Connection conn, String arg_user)
{
    String query = "SELECT AcctNum FROM Customer WHERE Balance < 100 AND Username = ?";
    PreparedStatement p = conn.prepareStatement(query);
    p.setString(1, arg_user);
    return p.executeQuery();
}
```

Input is confined to a single SQL atom
SELECT / FROM / WHERE

AcctNum  Customer  AND

<  =

Balance  100  Username  ?

Note: prepared statement only allows ?’s for leaves, not internal nodes. So structure of tree is fixed.
Defenses

Language support for constructing queries
Specify query structure independent of user input:

```java
ResultSet getProfile(Connection conn, String arg_user)
{
    String query = "SELECT AcctNum FROM Customer WHERE Balance < 100 AND Username = ?";
    PreparedStatement p = conn.prepareStatement(query);
    p.setString(1, arg_user);
    return p.executeQuery();
}
```

Binds the value of arg_user to '?' atom
Defenses

Language support for constructing queries
Specify query structure independent of user input:

```java
ResultSet getProfile(Connection conn, String arg_user) {
    String query = "SELECT AcctNum FROM Customer WHERE Balance < 100 AND Username = ?";
    PreparedStatement p = conn.prepareStatement(query);
    p.setString(1, arg_user);
    return p.executeQuery();
}
```

No matter what input user provides, Prepared Statement ensures it will be treated as a single SQL datum
Parse Tree Template Constructed by Prepared Statement

```
SELECT / FROM / WHERE
  AcctNum
  AND
    <
      Balance
      100
    =
      Username
      'foo' OR 1=1 --
```
Parse Tree Template Constructed by Prepared Statement

This will never be true (assuming no bizarre Usernames!), so no database records will be returned.
5 Minute Break
5 Minute Break

Questions Before We Proceed?
Basic Structure of Web Traffic
Basic Structure of Web Traffic

Includes:
- Resource from URL
- Headers describing browser capabilities
- Associated data for POST
Basic Structure of Web Traffic

HTTP Request

Specified as a **GET** or **POST**
Includes “resource” from URL
Headers describe browser capabilities
(Associated data for POST)
Basic Structure of Web Traffic

Includes status code
Headers describing the answer
Data for returned item
Basic Structure of Web Traffic

HTTP Request

- Specified as a **GET** or **POST**
- Includes “resource” from URL
- Headers describe browser capabilities
  (Associated data for POST)

E.g., user clicks on URL:
http://login.html?user=alice&pass=bigsecret
HTTP Request

GET /login.html?user=alice&pass=bigsecret HTTP/1.1
GET /login.html?user=alice&pass=bigsecret HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: mybank.com
Referer: http://www.mybank.com/hello-customer.html
The **Referer** header indicates which web page we clicked on to generate this request.
GET /login.html?user=alice&pass=bigsecret HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: mybank.com
Referer: http://www.mybank.com/hello-customer.html

HTTP Request
Basic Structure of Web Traffic

Includes status code
Headers describing the answer
Data for returned item
HTTP Response

HTTP version  Status code  Reason phrase
HTTP/1.0  200  OK

Headers
Date: Sat, 23 Feb 2013 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Fri, 22 Feb 2013 17:39:05 GMT
Content-Length: 2543

Data
<HTML> Welcome to BearBucks, Alice ... blahblahblah </HTML>
HTTP Cookies

Servers can include “cookies” in their replies: state that clients store and return on any subsequent queries to the same server/domain

Cookie is just a name/value pair. (Value is a string).
HTTP Response

HTTP/1.0 200 OK
Date: Sat, 23 Feb 2013 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Fri, 22 Feb 2013 17:39:05 GMT
Set-Cookie: session=44ebc991
Content-Length: 2543

<HTML> Welcome to BearBucks, Alice ... blahblahblah </HTML>

Cookie

Here the server instructs the browser to remember the cookie “session” so it & its value will be included in subsequent requests
Cookies & Follow-On Requests

HTTP Request

Includes “resource” from URL Headers describing browser capabilities, including cookies
GET /moneyxfer.cgi?account=alice&amt=50&to=bob HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: mybank.com
Cookie: session=44ebc991
Referer: http://mybank.com/login.html?user=alice&pass...
Cookies & Web Authentication

- One very widespread use of cookies is for web sites to track users who have authenticated.
  - E.g., once browser fetched `http://login.html?user=alice&pass=bigsecret` with a correct password, server associates value of “session” cookie with logged-in user’s info.
  - Now server subsequently can tell: “I’m talking to same browser that authenticated as Alice earlier”
    ⇒ An attacker who can get a copy of Alice’s cookie can access the server impersonating Alice!
    - “Cookie theft”
Static Web Content

Visiting this boring web page will just display a bit of content.
Automatic Web Accesses

Visiting this page will cause our browser to automatically fetch the given URL.
Automatic Web Accesses

So if we visit a page under an attacker’s control, they can have us visit other URLs...
Web Accesses w/ Side Effects

• Recall our earlier banking URL:

  http://mybank.com/moneyxfer.cgi?account=alice&amt=50&to=bob

• So what happens if we visit evilsite.com, which includes:

  <img src="http://mybank.com/moneyxfer.cgi?Account=alice&amp;amt=500000&amp;to=DrEvil">

  – Our browser issues the request …
  – … and dutifully includes authentication cookie! :-(

• Cross-Site Request Forgery (CSRF) attack
CSRF Defenses

• Defenses?
  – Inspect Referer headers (require it to be from mybank.com)
    Referer: http://evilsite.com/testpage.html
  – Or: require authentication (not just session cookie!) per request - what a pain :-(
  – Or: use distinct URLs (including randomized components) for forms users should use for requests; per Section discussion tomorrow

• Note: only the server can implement these!