Web security

With material from Dave Levin, Mike Hicks, Lujo Bauer
Previously

• Attack and defense at host machines
  • Applications written in C and C++
  • Violations of memory safety

• Web security now
  • Attacking web services
  • Problems: Confusion of code/data; untrusted input
Web security topics

- Web basics (today)
- SQL injection, defenses (maybe today)
- Stateful web and session problems (next week)
- Dynamic web and XSS (next week)
Web Basics
The web, basically

(Much) user data is part of the browser

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

Resources which are identified by a URL
(Universal Resource Locator)

http://www.umiacs.umd.edu/~mmazurek/index.html

Protocol
ftp
https
tor

Hostname/server
Translated to an IP address by DNS
(e.g., 128.8.127.3)

Path to a resource
Here, the file index.html is static content
i.e., a fixed file returned by the server
Interacting with web servers

**Resources which are identified by a **URL** (Universal Resource Locator)**

Path to a resource:
http://facebook.com/delete.php?f=joe123&w=16

Arguments

Here, the file `delete.php` 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 data
Basic structure of web traffic

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

- Request types can be **GET** or **POST**
  - **GET**: all data is in the URL itself
  - **POST**: includes the data as separate fields
HTTP GET requests

https://krebsonsecurity.com

User-Agent is typically a browser but it can be wget, JDK, etc.
According to security firm **Shavlik**, the patches that address flaws which have already been publicly disclosed include a large **Internet Explorer** (IE) update that corrects 17 flaws and a fix for **Microsoft Edge**, Redmond’s flagship replacement browser for IE; both address [this bug](#) among others.

A **critical fix** for a Windows graphics component addresses flaws that previously showed up in two public disclosures, one of which Shavlik says is currently being exploited in the wild ([CVE-2015-2546](#)). The 100th patch that Microsoft has issued so far this year — a salve for **Windows**

---

**HTTP Headers**


GET /view/vuln/detail?vulnId=CVE-2015-1421 HTTP/1.1
Host: web.nvd.nist.gov
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
**Referer: https://krebsonsecurity.com/**
Connection: keep-alive

**Referrer URL:** site from which this request was issued.
HTTP POST requests

Posting on Piazza

Implicitly includes data as a part of the URL

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

- **User clicks**
  - **Responses** contain:
    - **Status** code
    - **Headers** describing what the server provides
    - **Data**
    - **Cookies** (much more on these later)
      - Represent *state* the server would like the browser to store
HTTP responses

HTTP version
Status code
Reason

HTTP/1.1 200 OK

Headers
- Cache-Control: private, no-store, must-revalidate
- Content-Length: 50567
- Content-Type: text/html; charset=utf-8
- Server: Microsoft-IIS/7.5
- Set-Cookie: CMSPREFERREDCULTURE=en-US; path=/; HttpOnly; Secure
- Set-Cookie: ASP.NET_SessionId=4t2oj4nthxmjs1waletxlqa; path=/; secure; HttpOnly
- Set-Cookie: CMSCURRENTTHEME=NVDLegacy; path=/; HttpOnly; Secure
- X-Frame-Options: SAMEORIGIN
- x-ua-compatible: IE=Edge
- X-AspNet-Version: 4.0.30319
- X-Powered-By: ASP.NET, ASP.NET

Data
<html> ..... </html>
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/327/
Server-side data

Long-lived state, stored in a separate database

Need to protect this state from illicit access and tampering
### SQL (Standard Query Language)

#### Table

**Users**

<table>
<thead>
<tr>
<th>Name</th>
<th>Gender</th>
<th>Age</th>
<th>Email</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connie</td>
<td>F</td>
<td>12</td>
<td><a href="mailto:connie@bc.com">connie@bc.com</a></td>
<td>sw0rdg1rl</td>
</tr>
<tr>
<td>Steven</td>
<td>M</td>
<td>14</td>
<td><a href="mailto:steven@bc.com">steven@bc.com</a></td>
<td>c00kieC4t</td>
</tr>
<tr>
<td>Greg</td>
<td>M</td>
<td>34</td>
<td><a href="mailto:mr.uni@bc.com">mr.uni@bc.com</a></td>
<td>i&lt;3ros3!</td>
</tr>
<tr>
<td>Vidalia</td>
<td>M</td>
<td>35</td>
<td><a href="mailto:vidalia@bc.com">vidalia@bc.com</a></td>
<td>sc&amp;On!0N</td>
</tr>
</tbody>
</table>

#### SQL Commands

- **SELECT** Age FROM Users WHERE Name='Greg';
  
- **UPDATE** Users SET email='mr.uni@bc.com'
  WHERE Age=34; -- this is a comment

- **INSERT** INTO Users Values('Pearl', 'F', ...);

- **DROP** TABLE Users;
Server-side code

Website

“Login code” (PHP)

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

Suppose you successfully log in as $user if this returns any results

How could you exploit this?
SQL injection

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

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

Login successful!

Problem: Data and code mixed up together
SQL injection: Worse

```php
$result = mysql_query("select * from Users
    where(name='frank' OR 1=1);
    DROP TABLE Users; --
    and password='whocares');
");

$result = mysql_query("select * from Users
    where(name='frank' OR 1=1);
    DROP TABLE Users; --
    and password='whocares');
");
```

Can chain together statements with semicolon:
STATEMENT 1 ; STATEMENT 2
SQL injection: Even worse

'); EXEC cmdshell 'net user badguy backdoor / ADD'; --

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

$result = mysql_query("select * from Users
where(name='');
EXEC cmdshell 'net user badguy backdoor / ADD'; --
and password='whocares');");
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/327/
SQL injection attacks are common

% of vulnerabilities that are SQL injection

SQL injection
countermeasures
The underlying issue

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

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

When the boundary between code and data blurs, we open ourselves up to vulnerabilities
The underlying issue

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

Should be **data**, not **code**
Prevention: Input validation

- We require input of a certain form, but we cannot guarantee it has that form, so we must **validate it**
  - Just like we do to avoid buffer overflows

- Making input trustworthy
  - **Check** it has the expected form, reject it if not
  - **Sanitize** by modifying it or using it such that the result is correctly formed
Sanitization: Blacklisting

- **Delete** the characters you don’t want

- **Downside**: “Lupita Nyong’o”
  - You want these characters sometimes!
  - How do you know if/when the characters are bad?

- **Downside**: How to know you’ve ID’d all bad chars?
Sanitization: Escaping

- **Replace** problematic characters with safe ones
  - Change ‘ to \\
  - Change ; to ;
  - Change – to –
  - Change \ to \\

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

- **Downside**: Sometimes you want these in your SQL!
  - And escaping still may not be enough
Checking: Whitelisting

• Check that the user input is known to be safe
  • E.g., integer within the right range

• Rationale: Given invalid input, safer to reject than fix
  • “Fixes” may result in wrong output, or vulnerabilities
  • Principle of fail-safe defaults

• Downside: Hard for rich input!
  • How to whitelist usernames? First names?
Sanitization via escaping, whitelisting, blacklisting is HARD.

Can we do better?
Sanitization: Prepared statements

- Treat user data according to its type
- Decouple the code and the data

```
$db = new mysql("localhost", "user", "pass", "DB");
$statement = $db->prepare("select * from Users
  where(name='\$user' and password='\$pass');");
$statement->bind_param("ss", \$user, \$pass);
$statement->execute();
$result = mysql_query("select * from Users
  where(name='\$user' and password='\$pass');");
```

Bind variables are typed

Decoupling lets us compile now, before binding the data
Using prepared statements

```php
$statement = $db->prepare("select * from Users
                       where(name=? and password=?);");
$ Stmt->bind_param("ss", $user, $pass);
```

Binding is only applied to the leaves, so the structure of the tree is *fixed*
Additional mitigation

• For defense in depth, also try to mitigate any attack
  • But should always do input validation in any case!

• Limit privileges; reduces power of exploitation
  • Limit commands and/or tables a user can access
  • e.g., allow SELECT on Orders but not Creditcards

• Encrypt sensitive data; less useful if stolen
  • May not need to encrypt Orders table
  • But certainly encrypt creditcards.cc_numbers