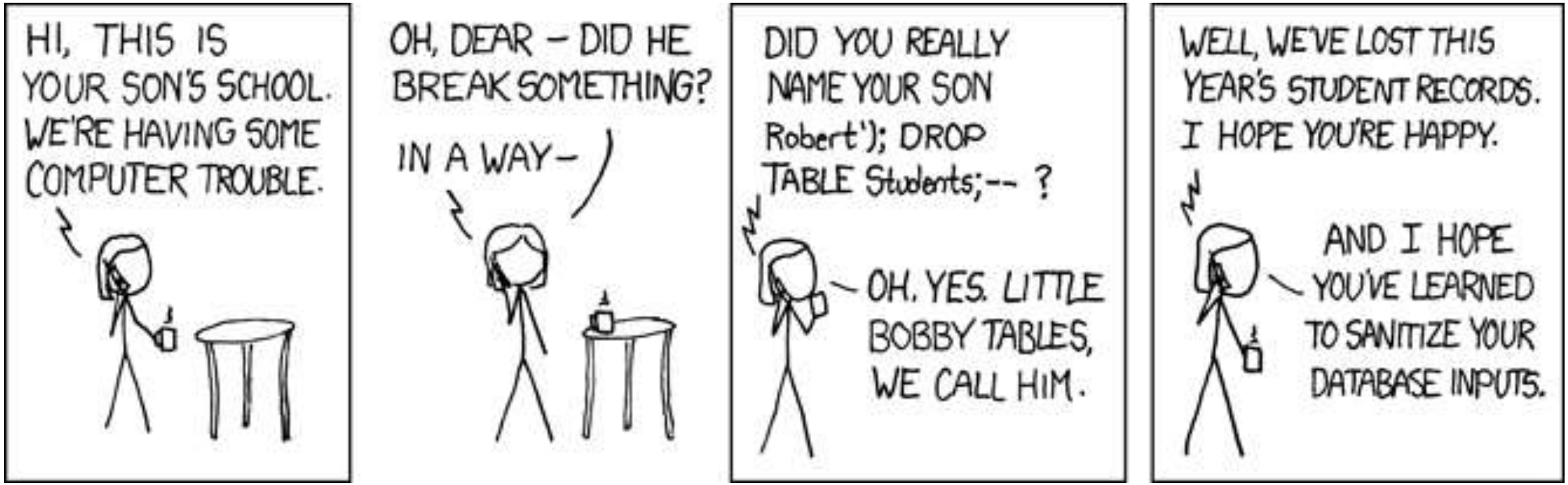
SQL Injection

COMP620


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.

XKCD

Second Exam

- The second exam in COMP620 will be on Friday, October 19
- It will cover all the material since the first exam

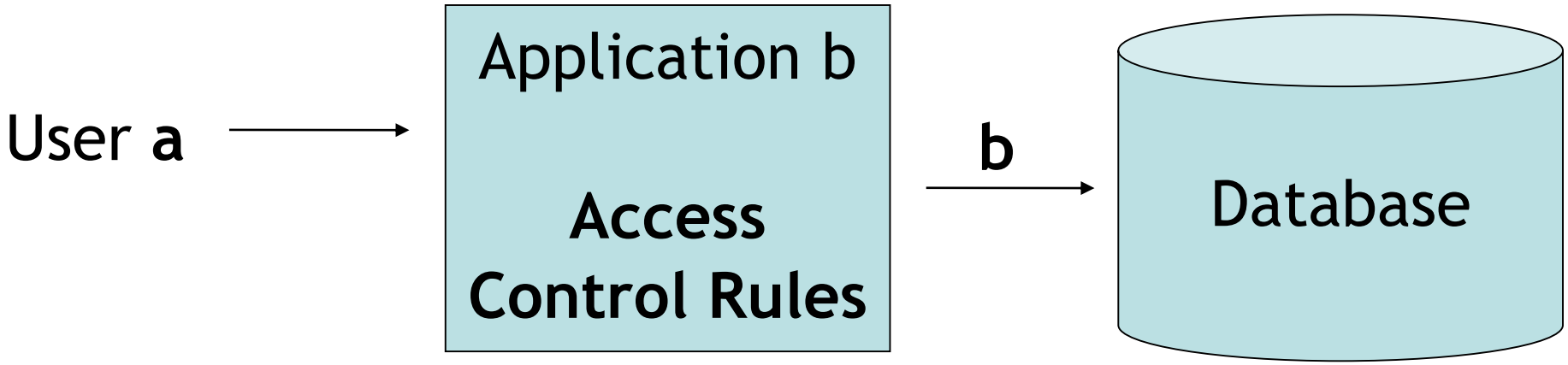
Serious Threat

- The United States Justice Department charged an American citizen Albert Gonzalez and two unnamed Russians with the theft of 130 million credit card numbers using an SQL injection attack. It was reportedly "*the biggest case of identity theft in American history*"

from the BBC

Targets for Attack

- Database applications often need to serve multiple users
- Programmers often give their applications elevated privileges



Principle of Least Privilege

- In almost every aspect of computer security, a basic rule is to **give a user or program only enough access necessary to accomplish the task**
- If a user only has read access to a database, then no matter what exploit they fall victim to, the attacker will not be able to change the database

Characterization of Attack

- Not a weakness of SQL
  - ...at least not in general
  - SQL Server may run with administrator privileges, and has commands for invoking shell commands
- Not a weakness of database, PHP/scripting languages, or Apache
- The problem occurs when using untrusted data in a situation where it will be interpreted

SQL Injection threat can be reduced by

A. Using non-SQL databases
B. Sanitizing user input
C. Converting user input to upper case
D. Only accessing the database as root
E. None of the above

Quick SQL Review (1)

- Querying tables:

```
select column1, column2 from table_name;
```

or

```
select * from table_name;
```

- Conditions:

```
select columns from table_name where condition;
```

Quick SQL Review (2)

- Inserting new rows:

```
insert into table_name values (value1,value2,
  ...);
```

  or

```
insert into table_name set column1=value1,
  column2=value2, ...;
```

- Updating rows:

```
update table_name set column1=value1 where
  condition;
```

Quick SQL Review (3)

- Deleting rows:

```
delete from table_name where condition;
```

- Set values in conditions:

```
select * from table_name
  where column in (select_statement);
```

or

```
select * from table_name
    where column in (value1, value2, ...);
```

Quick SQL Review (4)

- Joining tables:

```
select * from table1, table2 where
  table1.attribute1 = table2.attribute2;
```

- Built-in Functions

```
select count(*) from test;
```

Quick SQL Review (5)

- Pattern Matching

```
select * from test where col like '%c_t%';
```

- Other Keywords

```
select * from test where abc is null;
```

- Metadata Tables
  - Highly vendor-specific
  - Available tables, table structures are usually stored in some reserved table name(s)

Example Application

- This simple PHP program gets a name from a web form and creates a database query

```
$name = $HTTP_POST_VARS["name"];
$query = "select * from restaurants where
  name = '". $name ."'";
$result = mysql_query($query);
```

Login Example

- Assume we have an SQL table, **`login`**, containing columns username and password

```
$name = $HTTP_POST_VARS["user"];
$pwd  = $HTTP_POST_VARS["password"];
$query = "select count(*) from login where
  username = '$name' and password = '$pwd'";
result = mysql_query($query);
```

- Access is allowed if the result is not zero
- Example query:

```
select count(*) from login where
  username = 'KenW' and password = 'Secret'
```

Simple SQL Injection Example

- Logging in with:

```
select count(*) from login where
  username = '$username' and
  password = '$password';
```

- Setting the password to "**x' or 'a' = 'a**" gives:

```
select count(*) from login where username = 'alice'
  and password = 'x' or 'a' = 'a';
```

- In fact, username doesn't even have to match anyone in the database

Another Example

- Logging in with:

```
select count(*) from login where
  username = '$username' and
  password = '$password';
```

- Setting the password to

  “**X‘;insert into login set username=‘me’**

  password = ‘mySecret” gives:

```
select count(*) from login where username = 'alice‘
  and password = ‘x';
  insert into login set username=‘me’,
  password = ‘mySecret’;
```

Yet Another Example

- Logging in with:

```
select count(*) from login where
  username = '$username' and
  password = '$password';
```

- Setting the password to

  “X‘; update login set password = ‘mySecret’ where username = ‘admin” gives:

```
select count(*) from login where username = 'alice'
  and password = ‘x';
  update login set password =‘mySecret’
  where username = ‘admin’;
```

Write an attack

- Write user input for the password that will delete the login table preventing anyone from logging in

Possible Solution

- Set the password to

  “**X‘; drop table login**”

- This results in:

```
select count(*) from login where username = 'alice'
   and password = ‘x'; drop table login’;
```

Hashing Passwords

- Keeping a database of userids and passwords may cause a serious security breach
- It is better to store a one way hash (such as SHA-256) of the password instead of the password itself
- A program can compute the one way hash of an entered password and compare it to the stored hash value

Inferring Database Layout

- The attacker may be able to determine the names of your tables and fields
- Guess at column names with input such as:

```
' and email is null--
```

```
' and email_addr is null--
```

- The presence or absence of error messages will tell the attacker if they are successful

Inferring Database Layout (2)

- Guess at table name

```
' and users.email_addr is null--
```

```
' and login.email_addr is null--
```

  - Can be done with an automated dictionary attack
  - Might discover more than one table in the query

- Guess at other table names

```
' and 1=(select count(*) from test)--
```

Input Verification

- Input should be checked for improper values
- It is much more effective to use pattern matching to recognize good input (white list) than to try to detect bad input (black list)
- May be tricky if we want to allow arbitrary text
- This can be much harder than it looks if you consider escaped characters, %27 is a quote
- International sites need to consider multiple character sets

No Special Characters Needed

- In the input value is numeric, it may not be necessary to use quotes or other special characters

```
SELECT fields FROM table
    WHERE id = 23 OR 1=1
```

- You may be able to scan the input string for undesirable word

Parameterized Statements

- Database systems allow you to create queries that take data values from variables.

```
$sqlObj = new mysqli( ... );
$delConfQ = "delete from config where
             groupID = ? and player = ?";
$sqldel = $sqlObj->prepare($delConfQ);
$sqldel->bind_param("ss", $groupID, $player);
$sqldel->execute();
```

- All error checking has been removed from the example

What is the most secure login code?

A. Concatenate userid and password to make SQL query
B. Concatenate userid and hash of password to make SQL query
C. Parameterized SQL with userid and password
D. Parameterized SQL with userid and hash of password

Secure SQL

- With parameterized SQL statements untrusted user input is used only as data
- Untrusted user input is never interpreted
- Parameterized SQL statements are no more difficult than the very insecure creation of an SQL statement from user input
- Parameterized SQL statements can be much faster if you need to repeat the same statement with different data

System Oriented Prevention

- MySQL doesn’t allow multiple queries per request
- Limit database privileges of application
- Run in non-admin user space to prevent system calls (*e.g.* MS SQL Server)
- Have length limits on input
- Hide error messages

Blind SQL Injection

- Some production systems display detailed error messages that give the attacker lots of information
- You can still use SQL injection if you only get a “True” or “False” response
- This can be a slow process, but there are tools to automate the attack

Benefits of an Automated Tool

- We can ask the server as many yes/no questions as we want
- Finding the first letter of a username with a binary search takes 7 requests
- Finding the full username if it’s 8 characters takes 56 requests
- To find the username **is** 8 characters takes 6 requests
- 62 requests just to find the username
- This adds up

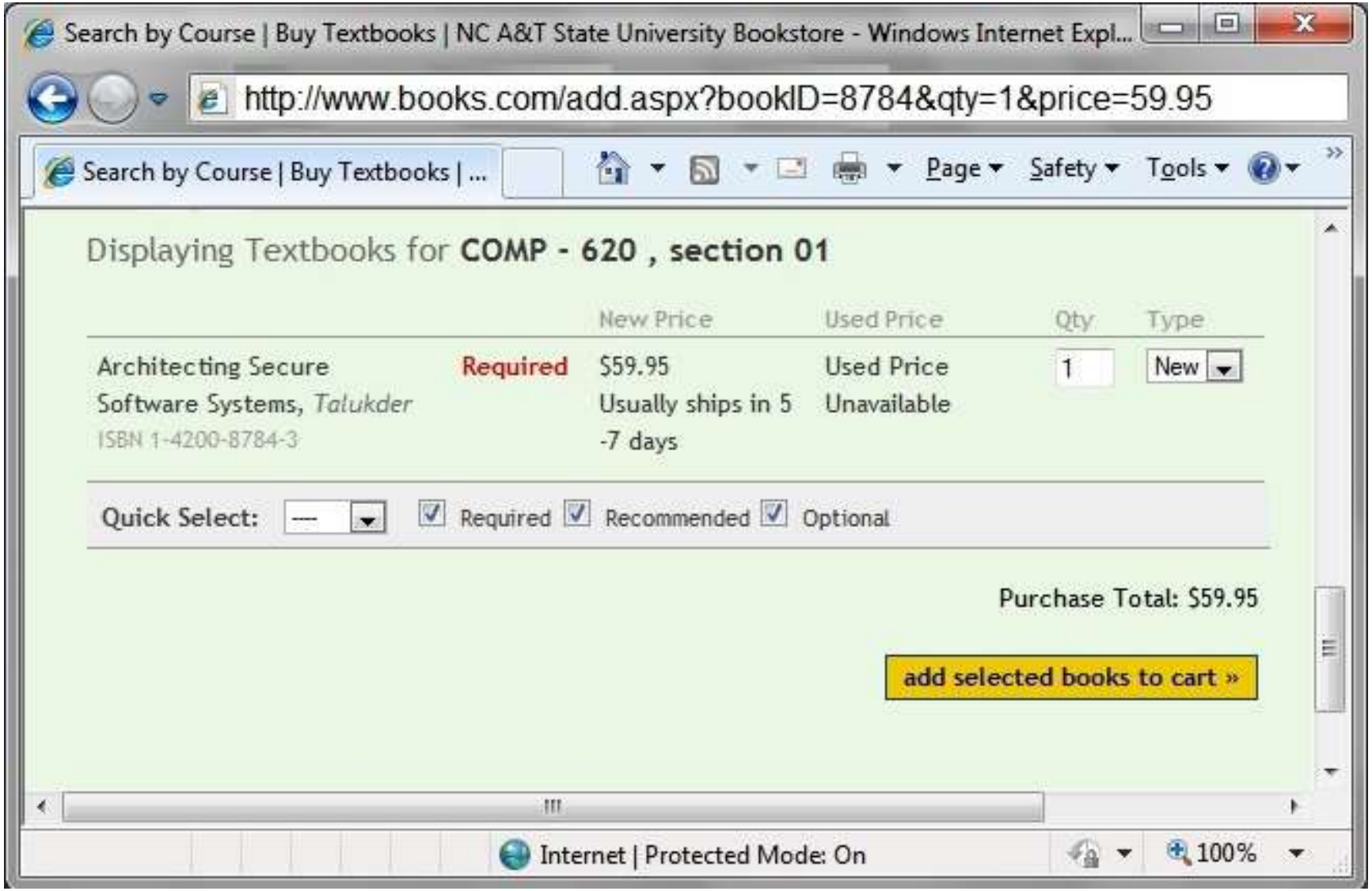
Automated Hacking Benefit

- Assuming it takes 10 seconds to make each request
- Assuming no mistakes are made, an 8 character username takes over ten minutes
- What if we want the schema or the entire database?

Parameter Tampering

- Some web applications transfer critical data between the browser and the server
- Changing that data can cause an exploit
- A proxy can be placed between the browser and the server to view and edit the HTTP requests
- The OWASP Zed Attack Proxy (ZAP) is one of the world's most popular free security tools

Example Application



Modifying Parameters

- The URL sent to the server contains parameters specifying what is to be done

```
http://www.books.com/add.aspx?
bookID=8784&qty=1&price=59.95
```

- This information might be passed to the application to add the book to the cart
- A user could enter this URL with a different price and pay much less for the book

Avoiding the Problem

- Applications should not trust input from the user
- The application should get the price of the book from its database

Releasing Too Much Information

- Consider the bookstore site that might show your past orders as shown where you can click on an order number to get more detail

Order #	Date	Title
2456	2/14/2010	Karma Sutra
2457	2/14/2010	Ulysses
3801	3/2/2010	Computer Security

Changing the URL

- When you click on an order number, it might send the following to the server

`http://www.books.com/order.aspx?ordernum=2457`

- You can probably guess that there are other orders with similar numbers. You might be able to see another person's order if you sent the above URL changing the number.

Guessing Names

- Sometimes you can guess what the file name of a web page will be

  williams.comp.ncat.edu/comp620/HW3sol.pdf

- Even if there is no link to the file, you can enter the expected URL
- This can be avoided by using long random names

Hidden Parameters

- There can be more parameters sent in a web form than are visible on the page. There may be parameter values in the HTML text that are also sent.
- Since the HTML can be viewed by the users, this information is not secret or secure.

```
<input type="hidden" id="admin" />
```

- Users can change the URL to change this value

HTML Comments

- A good programmer includes comments in any code they write, including an HTML document
- Sometimes programmers will comment out a section of the HTML document that is not currently being used
- Comments might give information on how to test the system.

```
<! use id tester with password
    “badchoice” to test />
```

Viewing Directories

- If a default.html or index.html file exists, the system will send these when no filename is specified in a URL
- If you enter a URL without a file name, the web server might show you all of the files in the directory
- This is usually a configuration option in the web server. It is best turned off.

Second Exam

- The second exam in COMP620 will be on Friday, October 19
- It will cover all the material since the first exam