Security in node.js

6.170 10/26/2015
Admin

- Concept design assignment due Monday 11 PM
- Next up: 6.170 Vacation Week!
- Then the Final Group Project
Introduction

Goal:

To prevent sophisticated attacks that exploit weaknesses in tools and protocols
Security Policy

A set of goals for what behavior you intend to support
Example Security Policy: Notes App

- Authors of a note have read, write, delete to their own notes
- Only authors can share a note with users with read or edit privileges
- Other than the author, only shared users can read or edit notes
Threat Model

A set of assumptions about the capabilities of your attackers
Example Threat Model: Notes App

- Unauthenticated user can construct requests (curl, Postman) to eavesdrop/tamper with other users’ notes

- User with basic credentials can construct requests or make requests via forms

- It’s unlikely much sensitive information will be stored on a note-taking application, so the likelihood of attracting interest from highly sophisticated attackers isn’t high
Exercise

Come up with a **security policy** and **threat model** for an **online voting app** for government elections.
Demo!

http://shoutkey.com/tangelo
Demo - SQL Injection

Sanitize your inputs before processing/saving/rendering!

See validator.js - https://github.com/chriso/validator.js
Demo - XSS

Mitigation:

- EJS and Handlebars HTML-escapes values using the syntax "<%=" for EJS and "{{" for Handlebars

  - Only compatible with IE9+ and Chrome
Demo: Cross Site Request Forgery (CSRF)

Mitigation:

● NPM Module: csurf ([Instructions](#))
  ○ Step 1: Install

```bash
npm install csurf
```
Demo: Cross Site Request Forgery (CSRF)

Mitigation:

- NPM Module: csurf
  - Step 2: Add csurf middleware to your Express app

```javascript
var csrf = require('csurf');
app.use(csrf());
```
Demo: Cross Site Request Forgery (CSRF)

Mitigation:

- NPM Module: csurf
  - Step 3: Pass the csrf-token to your view from inside your route or controller

```javascript
res.render('someform', { csrf: req.csrfToken() });
```
Demo: Cross Site Request Forgery (CSRF)

Mitigation:

- NPM Module: csurf
  - Step 4: Create a hidden input inside your view (inside your form)

```javascript
input(type="hidden", name="_csrf", value="#{csrf}")
```
Demo: Clickjacking

Mitigation:

- NPM module - `helmet.frameguard()`
Demo: Clickjacking

Usage:

```javascript
// Only let me be framed by people of the same origin:
app.use(helmet.frameguard('sameorigin'));

// Don't allow anyone to put me in a frame.
app.use(helmet.frameguard('deny'));

// Allow from a specific host:
app.use(helmet.frameguard('allow-from', 'http://example.com'));
```
Other Security Concerns

Brute-force attack against login/other systems

Mitigation:

- Use the npm package `ratelimiter`
- Account lockout
Other Security Concerns

Evil Regexes

Most Regular Expression implementations may reach extreme situations that cause them to work very slowly
Other Security Concerns

Evil Regexes

e.g. \(([a-zA-Z]+)\)*, (a\+)\+ or (a|a\?)\+

**Exercise:** What sort of inputs would cause heavy computation if these regexes are used?
Other Security Concerns

Evil Regexes

Mitigation: Use a node.js tool called safe-regex

$ node safe.js '^(x+x+)+y'$
false

$ node safe.js '^(beep|boop)*$'
true
Security Tips

- Don’t expose stack traces or debugging information

- You can audit your shrinkwrap or package.json

  ```bash
  npm install nsp -g
  
  nsp audit-shrinkwrap  # audit shrinkwrap
  
  nsp audit-package  # audit package.json
  ```
Questions?