Practical Application Security for Developers

ADITI CHAUDHRY

TWO SIGMA

@aditichaudhry92
#WITSMW19

Disclaimer

This document is being distributed for informational and educational purposes only and is not an offer to sell or the solicitation of an offer to buy any securities or other instruments. The information contained herein is not intended to provide, and should not be relied upon for, investment advice. The views expressed herein are not necessarily the views of Two Sigma Investments, LP or any of its affiliates (collectively, "Two Sigma"). Such views reflect the assumptions of the author(s) of the document and are subject to change without notice. The document may employ data derived from third-party sources. No representation is made by Two Sigma as to the accuracy of such information and the use of such information in no way implies an endorsement of the source of such information or its validity.

The corporate and/or trademarks in some of the images, logos or other material used herein may be owned by entities other than Two Sigma. If so, such copyright and/or trademarks are most likely owned by the entity that created the material and are used purely for identification and comment as fair use under international copyright and/or trademark laws. Use of such image, copyright, or trademark does not imply any association with such organization (or endorsement of such organization) by Two Sigma, nor vice versa.

A. JAMES CLARK
SCHOOL OF ENGINEERING

About Me

@aditichaudhry92
#WITSMW19

Overview

- What is AppSec
- Why is AppSec Important
- Secure Code Principles
- OWASP Top 10
- Hands On!

Setup

Run “brew install node” for Linux

Use Windows Installerx
(**https://nodejs.org/en/download/**)

Slides and Code examples at
https://github.com/ac8wv/AppSec4Devs

What is AppSec?

PEN-TESTING

CODE REVIEW

DEVSECOPS

enhancing the security of our applications by finding & fixing vulnerabilities

Why is AppSec Important

Money

Reputation

Legal

@aditichaudhry92
#WITSMW19

Why is AppSec Important

- easiest way to make sure your application is secure is to build it securely in the first place

Secure Code Principles

Encrypt all the things

All user input is untrusted & potentially malicious

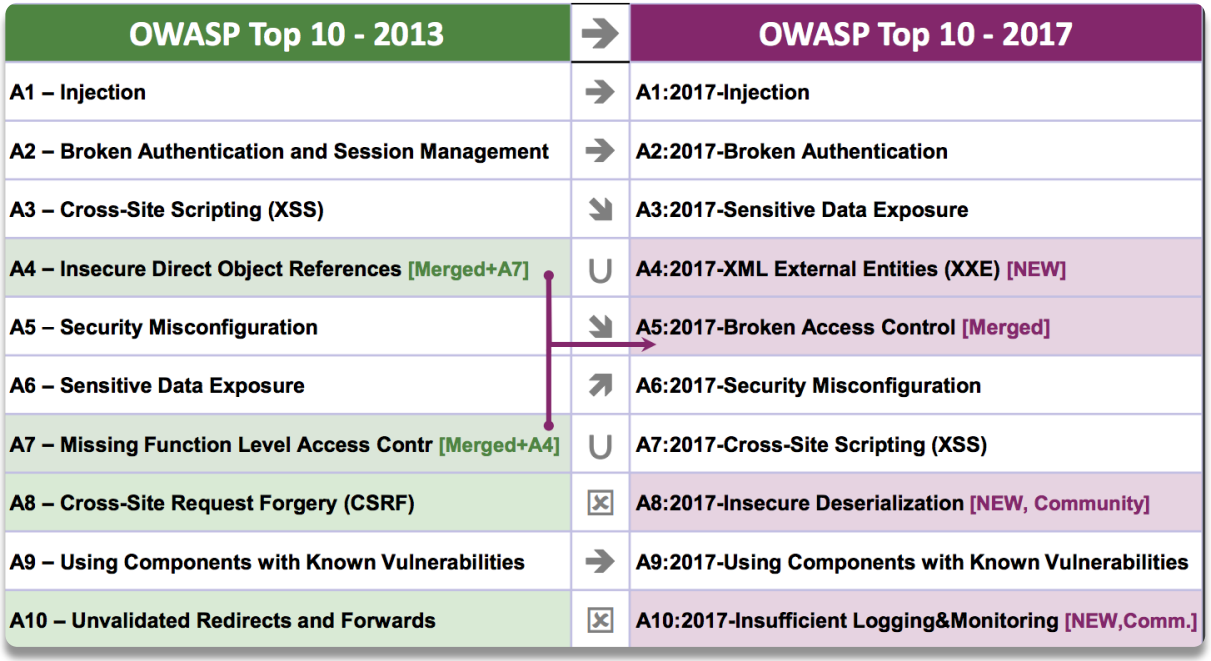
Separate data from instructions

Hash & salt passwords

Don't reveal more information than necessary

OWASP

- "The OWASP Top 10 is a powerful awareness document for web application security. It represents a broad consensus about the most critical security risks to web applications."



OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Use HTTPS

Encrypts all data in transit between client & server

Provides privacy, authentication, and integrity of data

Over TLS

How to configure TLS:

- Need a cert from a Certificate Authority
- Can use self-signed cert for development

Encrypt All the Things

@aditichaudhry92
#WITSMW19

Setup

- If you want to follow along, encryption examples can be found on my GitHub
  - **git clone https://github.com/ac8wv/AppSec4Devs.git**
  - **cd encryption**
  - **npm install**
  - **npm start**
- If you want to try out yourself:
  - **mkdir encryption** (or whatever folder name you want)
  - **npm install express**
  - Create a file for your server (I call mine express.js)

Generate Cert w/OPENSSL

```
openssl req -x509 -newkey rsa:2048 -keyout aditiprivatekey.pem -out cert.pem -days 365 -subj '/C=US/ST=NY/O=WITS' -nodes
```

Use cert while setting up HTTPS Express Server

Setting up HTTPS Express Server

- Creating server with our public and private key
- Run **npm start** or **node express.js**
- Go to **https://localhost:8000**
- Won't work over HTTP

```
1   var express = require('express');
2   var https = require('https');
3   var fs = require('fs');
4   var app = express();
5
6   var options = {
7       key: fs.readFileSync('aditiprivatekey.pem'),
8       cert: fs.readFileSync('cert.pem')
9   }
10
11  https.createServer(options, app).listen(8000, function(err){
12      if (err) throw err;
13      console.log('Server started on port 8000');
14  });
15
16  app.get('/', function (req, res) {
17      res.header('Content-type', 'text/html');
18      return res.end('<h1>Securely served over HTTPS!!</h1>');
19  });
```

Don’t Trust User Input

- Cross Site Scripting (XSS)
  - Attackers injecting malicious code, usually JavaScript, into your website
  - 3 types
    - Stored or persistent XSS
    - Reflected XSS
    - Self-XSS

Stored XSS

Attacker injects script into a website's database

e.g. attacker pastes payload into a comment on a forum, which is saved in the thread

Malicious JavaScript executed

every time victim visits the page

Reflected XSS

Victim clicks on link crafted to execute payload on page load

e.g. link in a phishing email or social media post

http://example.com/index.php?user=<script>alert(123)</script>

XSS Code Example

https://ironhackers.es/en/cheatsheet/cross-site-scripting-xss-cheat-sheet/

Preventing XSS

- HTML escape user provided data before adding to the DOM
- HTML escaping > sanitization
  - Attackers will find ways to get around sanitization libraries
- Recommend lodash escape function
  - _.escape()
- Use X-XSS-Protection and Content Security Policy headers
- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.md

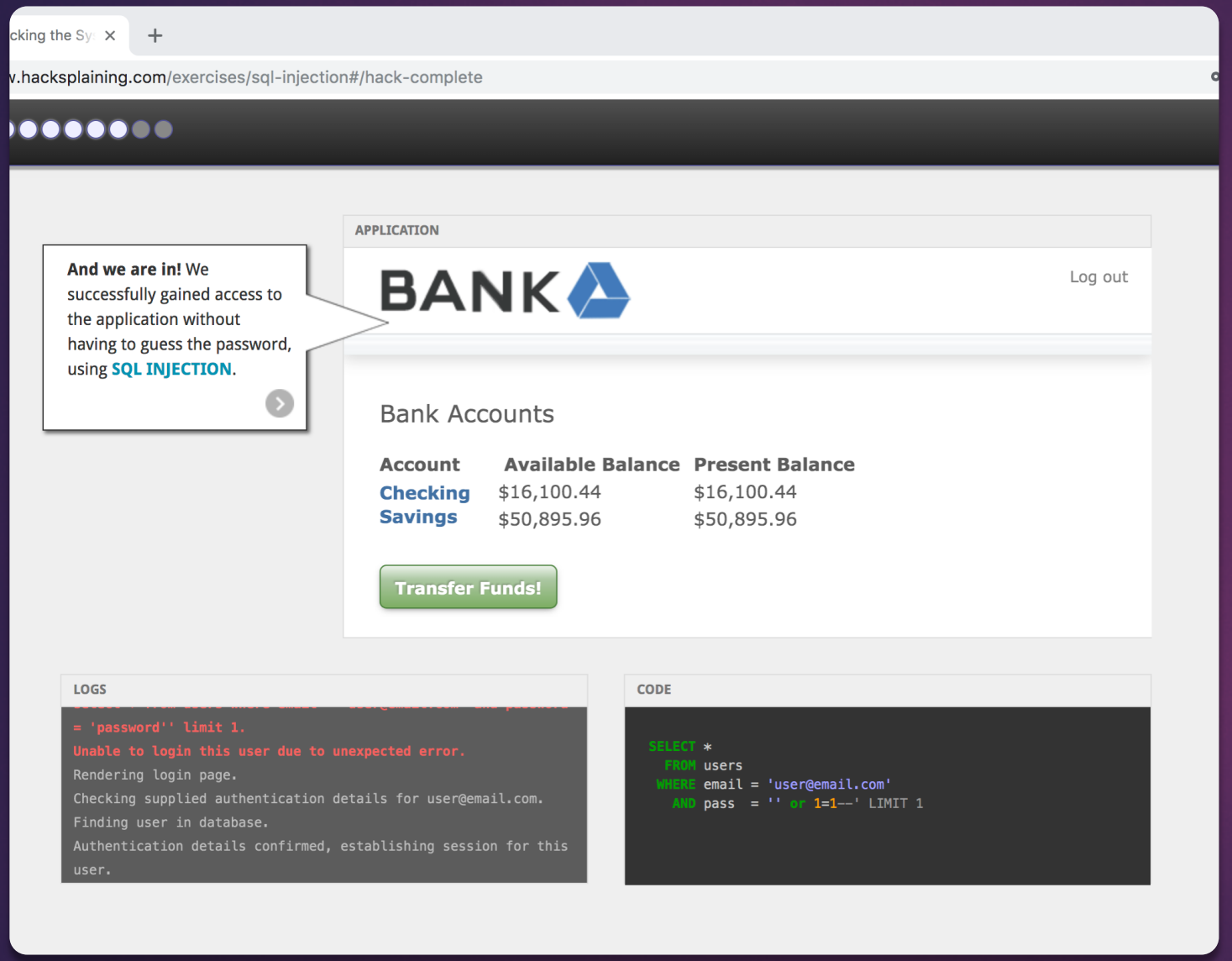
Separate Data from Instructions

SQL Injection

- Placement of malicious code in SQL statements via input data from the client to application
- can read sensitive data from db
- modify db data (Insert/Update/Delete),
- execute admin operations on the db

‘ or 1=1--

- Characters cause db to ignore the SQL statement, performing authentication without supplying real password



SQL Injection Example

HTTPS://WWW.HACKSPLAINING.COM/EXERCISES/SQL-INJECTION#

@aditichaudhry92
#WITSMW19

Preventing SQL Injection

- Use of Prepared Statements (with Parameterized Queries)
- Use of Stored Procedures
- Whitelist Input Validation
- Escaping All User Supplied Input
- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md

Hash & Salt Passwords

Please don't write your own authentication system!

- Passport module
  - passport-local for username + password

Never store unencrypted, plain text passwords in your database

- Hash and salt passwords
- Salt - unique random string
  - Use long salts
  - Never reuse a salt
  - Protects against rainbow table attacks

Setup

- GitHub
  - **cd hashing**
  - **npm install**
  - **npm start**
- Or
  - **mkdir hashing** (or whatever folder name you want)
  - **npm install bcrypt**
  - Create a file (I call mine hashing.js)

Hash & Salt using bcrypt

- npm start
- node hashing.js

```
var password = 'mysupersecurepassword'
var saltRounds = 10;

bcrypt.hash(password, saltRounds, function(err, hash) {
    // in real life we would store the has in a DB
    console.log('The password is ' + password + ' with ' + saltRounds + ' salt rounds');
    console.log('The password hash is ' + hash);

    // compare password to the hash stored in your database
    bcrypt.compare(password, hash, function(err, res) {
        // res should be "true"
        console.log(res);
    });

    // compare a bad password to what's stored in your database
    bcrypt.compare('badpassword', hash, function(err, res) {
        // res should be "false"
        console.log(res);
    });
```

Cookies

- Cookie = user information written to a text file
- Ex: Session cookies used for authentication
  - Like a conference badge, you check-in (authenticate) once and have access to the conference until it's over (cookie expires)
- Concern is if stolen, can be used to hijack user session

Flags to Secure Cookies

secure flag

- cookie can only be sent over HTTPS, not HTTP

httpOnly flag

- cookie cannot be accessed via JavaScript (important for XSS)
- Setting httpOnly flag will break any JavaScript that relies on access to cookies

Setup

- GitHub
  - **cd cookies**
  - **npm install**
  - **npm start**
- Or
  - **mkdir cookies** (or whatever folder name you want)
  - **npm install express**
  - **npm install cookie-parser**
  - **npm install express-session**
  - Create a file (I call mine cookies.js)

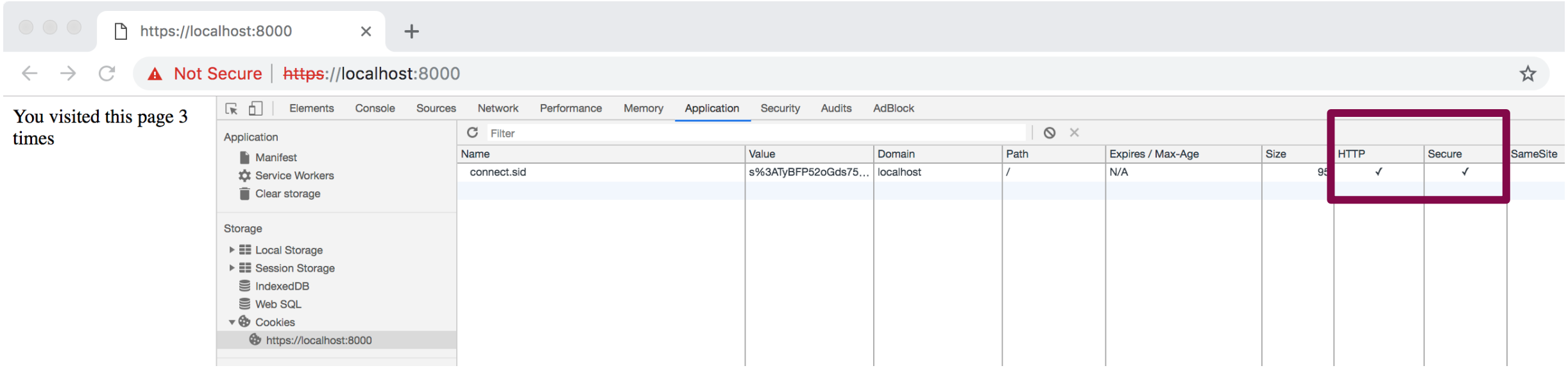
Cookie Code Example

- **npm start** or **node cookies.js**
- go to **https://localhost:8000**
- See cookies secured at: Inspect page -> Application -> Cookies

@aditichaudhry92
#WITSMW19

```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');
var https = require('https');
var fs = require('fs');

var app = express();

var options = {
    key: fs.readFileSync('../aditiprivatekey.pem'),
    cert: fs.readFileSync('../cert.pem')
}

https.createServer(options, app).listen(8000, function(err){
    if (err) throw err;
    console.log('Server started on port 8000');
});

app.use(cookieParser());
app.use(session({
    secret: "secret!",
    cookie: {
        secure: true,
        httpOnly: true
    }
}));

app.get('/', function(req, res){
   if(req.session.page_views){
      req.session.page_views++;
      res.send("You visited this page " + req.session.page_views + " times");
   } else {
      req.session.page_views = 1;
      res.send("Welcome to this page for the first time!");
   }
});
```

Cookies Secured!



Information Disclosure

- Don't disclose more info than necessary about your system/software
  - Easier to hack a system if you know it's architecture or what software it's running
- Disable HTTP headers that reveal software names and versions.
- Don't run in verbose mode
- Don't display stack traces/verbose errors
  - Keep error messages super generic
- Remove comments from prod
- Can use Helmet module w/Express server for secure headers

Secure Headers

- Using Helmet w/Node automatically sets security headers for you
  - **Strict-Transport-Security** enforces secure (HTTP over SSL/TLS) connections to the server
  - **X-Frame-Options** provides clickjacking protection
  - **X-XSS-Protection** enables the XSS filter built into most recent web browsers
  - **X-Content-Type-Options** prevents browsers from MIME-sniffing a response away from the declared content-type
  - **Content-Security-Policy** prevents a wide range of attacks, including Cross-site scripting and other cross-site injections

Setup

- GitHub
  - **cd helmet**
  - **npm install**
  - **npm start**
- Or
  - **mkdir helmet** (or whatever folder name you want)
  - **npm install express**
  - **npm install helmet**
  - Create a file (I call mine helmet.js)

Secure Headers Example

- Add to express server code:
  - **var helmet = require('helmet');**
  - **app.use(helmet())**
- Start server and navigate to browser
  - **https://localhost:8000**
- See headers set at: Inspect page -> Network -> Headers

```
1   var express = require(
2   var https = require('ht
3   var fs = require('fs');
```

▼ **Response Headers** view source

Connection: keep-alive
Content-Length: 37
Content-type: text/html; charset=utf-8
Date: Sun, 28 Apr 2019 22:32:41 GMT
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Content-Type-Options: nosniff
X-DNS-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block

Summary

company's applications ARE it's reputation

writing code securely from the beginning makes our lives easier & cost of building app cheaper

earlier we find & fix security issues, the safer our apps & companies will be

@aditichaudhry92
#WITSMW19

Additional Resources

- https://blog.risingstack.com/node-js-security-checklist
- https://paragonie.com/blog/2015/08/gentle-introduction-application-security
- https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
- https://code.likeagirl.io/links-for-getting-started-in-application-security-cc529d969cc6
- https://medium.com/@nodepractices/were-under-attack-23-node-js-security-best-practices-e33c146cb87d

Questions?

@aditichaudhry92
#WITSMW19

Contact Information

@aditichaudhry92

linkedin.com/in/aditi-chaudhry-47303a80

medium.com/@aditi.chaudhry92