Introduction to Security
Web Security

Ming Chow (mchow@cs.tufts.edu)

Twitter: @0xmchow

Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

Learning Objectives

- By the end of this week, you will be able to:
  - Perform and defend against the following attacks:
    - Cross-Site Scripting (XSS)
    - SQL injection
    - Cross-Site Request Forgery (CSRF)
    - Session hijacking
    - Cookie tampering
    - Directory traversal
    - Command injection
    - Remote and local file inclusion

Why Web Security?

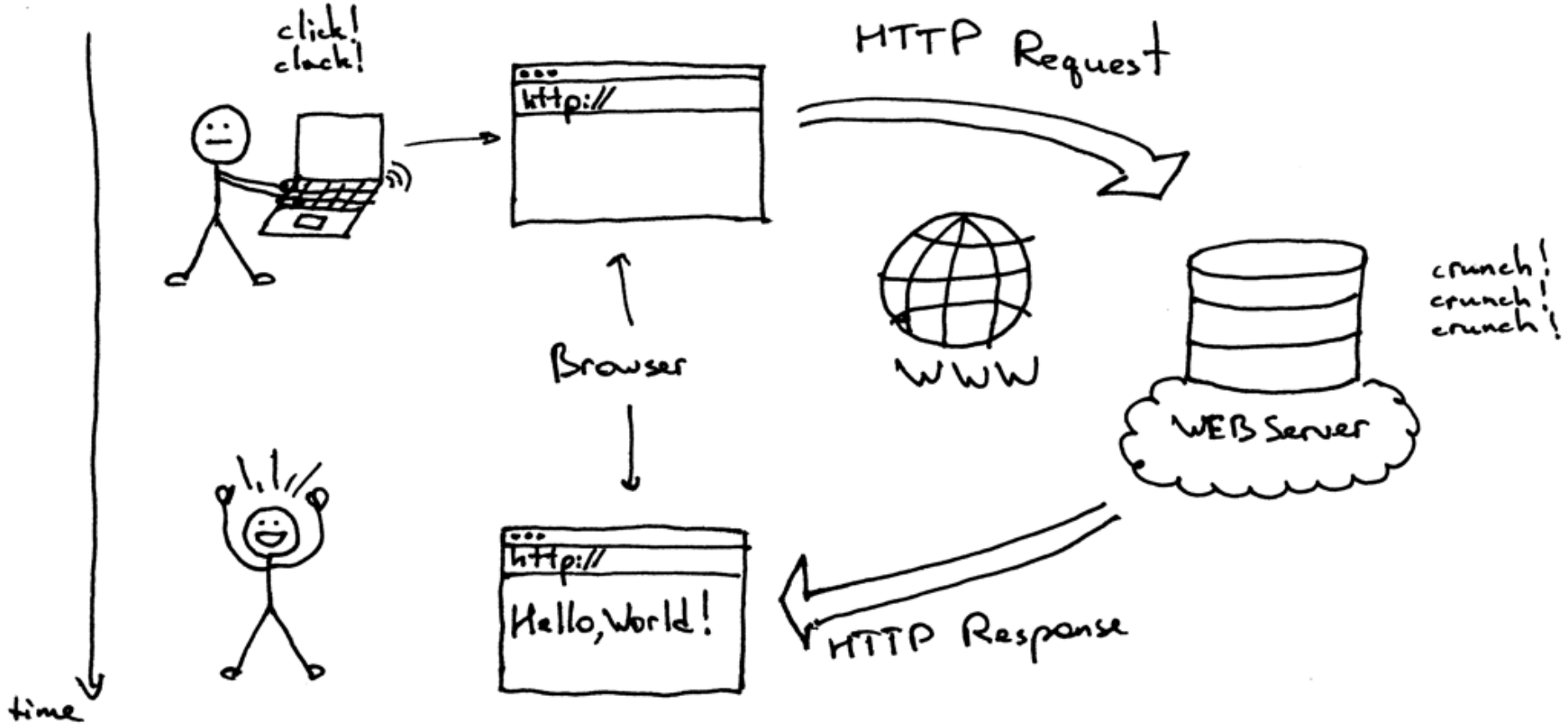
- So far, we have seen networking, attacking networking, and cryptography. Web security is a very logical next step.
- Wait, why aren't we covering exploitation, reverse engineering, and the classic buffer overflow next?
  - Buffer overflow has become much harder to do now thanks to protection mechanisms including Address Space Layout Randomization (ASLR), StackGuard, etc.
  - Let this sink in: **"69 percent of web applications are plagued by vulnerabilities that could lead to sensitive data exposure, and 55 percent by cross-site request forgery flaws; 25% of web apps still vulnerable to eight of the *OWASP Top Ten*" (circa 2017: https://www.helpnetsecurity.com/2017/02/14/web-application-vulnerabilities/)**
  - Alas, we are still battling the same issues as we have been for decades.

Preliminaries

What is the Web?

- NOT to be confused with the Internet
- The World Wide Web (WWW) a.k.a., the web
- A subset of the Internet
- A collection of web sites, pages, and content from around the world

How Does the Web Work?



How Does the Web Work? (continued)

- Previous image source: https://twitter.com/ThePracticalDev/status/709351333195882496
- Client-server technology
  - **Client** - A program running on your computer
    - Web browser - a client application that displays web pages (e.g., Chrome, Firefox, Microsoft Internet Explorer, Safari, Opera, lynx)
  - **Server** - A computer running web server software on a remote computer; delivers information to other clients
    - Examples: Nginx, Apache HTTP Server, Microsoft IIS

How Does the Web Work? Uniform Resource Locators (URLs)

- A universal naming scheme to specify the location of a document on a web site. That is, for finding and locating content.
- A subset of the Uniform Resource Identifier (URI)
- Created by Tim Berners-Lee in 1994
- Format: `protocol://machine_or_server/directory/file.type`
  - Protocols (*Application Layer on OSI Model*): `http, ftp, telnet, gopher, mailto, file`
  - Example: `http://www.eecs.tufts.edu/index.html`
    - `http` - Hypertext Transport Protocol
    - `www.eecs.tufts.edu` - machine www, domain eecs.tufts.edu
    - `index.html` - a file in the Hypertext Markup Language (.html)
- Query string with parameters: portion of URL where data, in key-value pairs separated by ampersand, is passed to a web server or web application (think variables). The first question mark is used as a separator, and is not part of the query string.
  - Example: `https://www.google.com/search`**`?q=grand+theft+auto&lr=lang_zh-TW`** (returns Google results on "Grand Theft Auto" in Chinese Traditional language)
  - `q` => Google's key in query string for "query"
  - `lr` => Google's key in query string for "language"
  - Notice example URL uses `https`. That is HTTP + Transport Layer Security (TLS)

How Does the Web Work? HyperText Transfer Protocol (HTTP)

- On Application Layer of the OSI Model (recall Networking)
- The idea: *request-response* protocol. Think question-and-answer
- Plaintext protocol (insecure)
- Stateless protocol
- RFC 2616: http://www.ietf.org/rfc/rfc2616.txt

HTTP Request

- Two parts: header and body
- (Client request) header: details about the request. Think of the details on an envelope.
  - List of header fields: http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html
  - Commands sent from a web browser (the client) to web server. Request methods:
    - **GET** - Download data from server. This is always the HTTP command used when you type in a URL into address bar on a modern web browser and then you press “Enter” on keybooard
    - **POST** - Sent to server from a form
    - **PUT** - Upload
    - **DELETE**
    - Additional HTTP commands: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- Body: data to be sent to server including query string key-value pairs

HTTP Response

- Two parts: header and body
- Server response header: Define characteristics of the data that is requested or the data that has been provided
  - List of header fields: http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html
  - Response status codes:
    - 200 - OK
    - 301 - Moved Permanently
    - 302 - Found (the request was redirected to another URL/URI)
    - 401 - Unauthorized
    - 403 - Forbidden
    - 404 - Not Found
    - 500 - Internal Server Error
    - Complete list: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
- Server response body: the content data (e.g., HTML, text, JSON, etc.)

A Little More on HTTP Response Status Codes

HTTP status ranges in a nutshell:

1xx: hold on

2xx: here you go

3xx: go away

4xx: you fucked up

5xx: I fucked up

-via @abt_programming

HTML and JavaScript

- **HTML** - HyperText Markup Language
  - To learn more, take my Web Programming class: https://tuftsdev.github.io/WebProgramming/notes/html.html
- **JavaScript** – Programming Language
  - To learn more, take my Web Programming class: https://tuftsdev.github.io/WebProgramming/notes/javascript.html
  - Now can be used for client-side and server-side programs
  - We will be focusing on client-side JavaScript to abuse web pages

JavaScript (Source: Reddit, https://i.redd.it/h7nt4keyd7oy.jpg)

HTTP Cookie

- A small amount of information sent by a server to a browser, and then sent back by the browser on future page requests to same site
- Data in form of key-value pairs
- RFC 2109: https://www.ietf.org/rfc/rfc2109.txt
- The maximum size of a cookie is 4 KB
- The total number of cookies that can be stored is 300 with a maximum of 20 cookies accepted from a particular server or domain
- All cookies set by server are sent to server during interaction
- Same-Origin Policy: a domain cannot access a cookie set by another domain!
- Can be manipulated on (i.e., stored as file on client)
- Used for authentication, user tracking, maintaining states (e.g., preferences, shopping cart)
- Can be persistent (i.e., last longer that browsing session)
- Via JavaScript:
  - Setting a cookie: `document.cookie = updatedCookie;` where `updatedCookie` is a string of form `key=value`
  - See all the cookies set by site: `allCookies = document.cookie;`
  - Getting the value of a cookie: find it in `document.cookie`
- Reference: https://developer.mozilla.org/en-US/docs/DOM/document.cookie
- Live example: https://tuftsdev.github.io/WebProgramming/examples/cookies_localstorage/cookies_example.html

Web Security

Tufts UNIVERSITY | SCHOOL OF ENGINEERING
Computer Science

OWASP Top 10

- OWASP: Open Web Application Security Project; non-profit, international organization
  - https://www.owasp.org/
- What is the OWASP Top 10 Project? To “educate developers, designers, architects, managers, and organizations about the consequences of the most important web application security weaknesses"
- UPDATED! The list for 2017: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
- While not perfect, the OWASP Top 10 has been instrumental raising awareness on web security

OWASP Top 10 Application Security Risks (2017)

A1:2017 - Injection
A2:2017 - Broken Authentication
A3:2017 - Sensitive Data Exposure
A4:2017 - XML External Entities (XXE)
A5:2017 - Broken Access Control
A6:2017 - Security Misconfiguration
A7:2017 - Cross Site Scripting (XSS)
A8:2017 - Insecure Deserialization
A9:2017 - Using Components with Known Vulnerabilities
A10:2017 - Insufficient Logging & Monitoring

CWE/SANS TOP 25 Most Dangerous Software Errors

- From SANS Institute
- Last list: circa 2011
- https://www.sans.org/top25-software-errors/
- Notice the similarities with the OWASP Top 10 list

Is There a Legal Way or Place to Practice Attacking Web Applications?

- IMPORTANT: NEVER DEPLOY THESE WEB APPLICATIONS TO THE PUBLIC INTERNET OR ON A PRODUCTION SYSTEM!
- Damn Vulnerable Web Application (DVWA) - http://www.dvwa.co.uk/
- Mutillidae - https://sourceforge.net/projects/mutillidae/
- Hacme Casino - https://www.mcafee.com/us/downloads/free-tools/hacme-casino.aspx (old; Ruby on Rails based)
- WebGoat - https://github.com/WebGoat/WebGoat/wiki; by OWASP
- A plethora deliberately vulnerable web applications to install and practice on

Metasploitable 2

- An intentionally vulnerable Linux virtual machine (VM)
- Under 2 GB
- Developed by Rapid7
- Download: https://sourceforge.net/projects/metasploitable/
- Uses VMware by default; can run on VirtualBox
- Contains Damn Vulnerable Web Application, Mutillidae, phpMyAdmin, etc.
- Great practice environment
- References:
  - https://community.rapid7.com/docs/DOC-1875
  - https://www.offensive-security.com/metasploit-unleashed/requirements/

Before We Begin: Using Web Proxies

- A web proxy will be an important tool for testing and breaking web applications
- Recall HTTP: request-response protocol; client makes request to server, server sends response to client
- What a web proxy does: intercepts requests and responses so you can modify HTTP request header fields and request body including query strings and data; records and logs HTTP(S) traffic
- Many web proxie software available:
  - Burp Suite
  - OWASP Zed Attack Proxy (ZAP)
  - Tamper Data for Firefox
  - mitmproxy

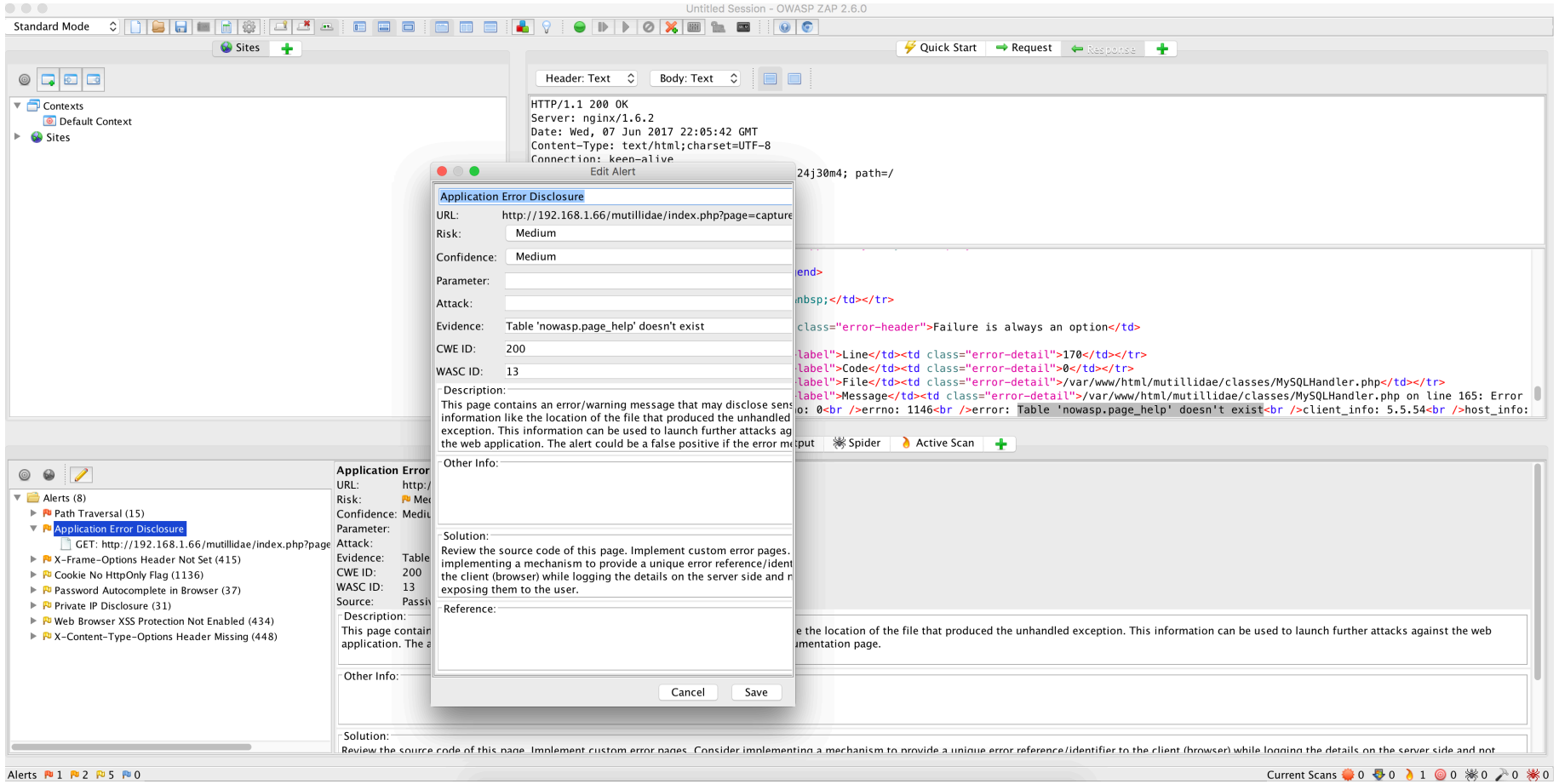
Tool: Burp Suite

- https://portswigger.net/burp/
- Available on Kali Linux
- Java-based
- Free version: intercept browser traffic using man-in-the-middle proxy
- Paid version: automated crawler and scanner for common vulnerabilities including “over 100 generic vulnerabilities, such as SQL injection and cross-site scripting (XSS), with great performance against all vulnerabilities in the OWASP top 10."

Tool: OWASP Zed Attack Proxy (ZAP)

- Free and open source
- Java-based
- https://github.com/zaproxy
- Similar to Burp Suite
- Includes vulnerability scanner and spider

Tool: OWASP ZAP (continued)



Tool: Tamper Data

- Add-on for Firefox web browser
- https://addons.mozilla.org/en-US/firefox/addon/tamper-data/
- You can intercept HTTP requests, you can modify parameters, but not as feature rich as Burp Suite or OWASP ZAP

The Vulnerabilities

The Principle of Least Privilege –Or Lackthereof

- The issue: connecting to a database or system as root or as administrator -- which has the power to do anything
- Example of bad code
- Get the root password and you get the keys to the kingdom to do anything that you want
- Prevention and defense: create a separate user for web applications *with access only such information, operations, and resources that are necessary to its legitimate purpose* (which is the definition of least privilege)

```php
<?php
 $conn =mysql_connect( "127.0.0.1",
                       "root",
                       "pass");
?>
```

Hard-Coded Credentials

- The issue: username and password or key are hard-coded in source code
- Well, the credentials are there for the taking
  - God forbid if you push source code with the credentials to GitHub
- Prevention and defense: don't hard-code credentials into source code; store credentials in system environment variables

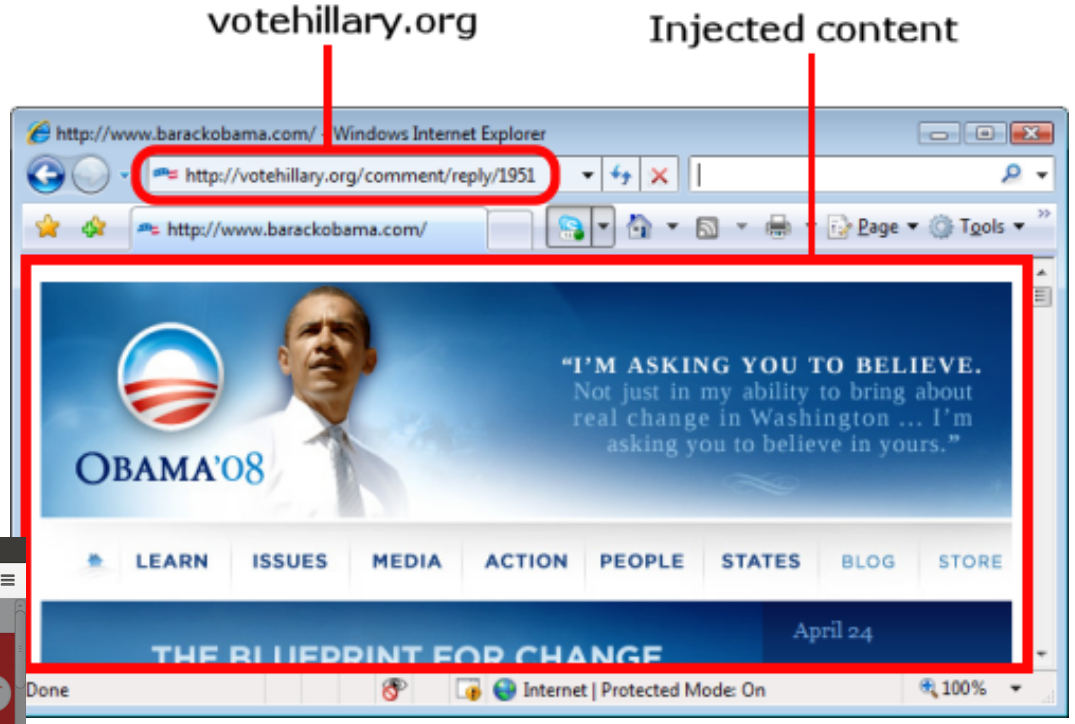
```
<?php
 $conn =mysql_connect( "127.0.0.1",
                                    "root",
                                    "pass");
?>
```

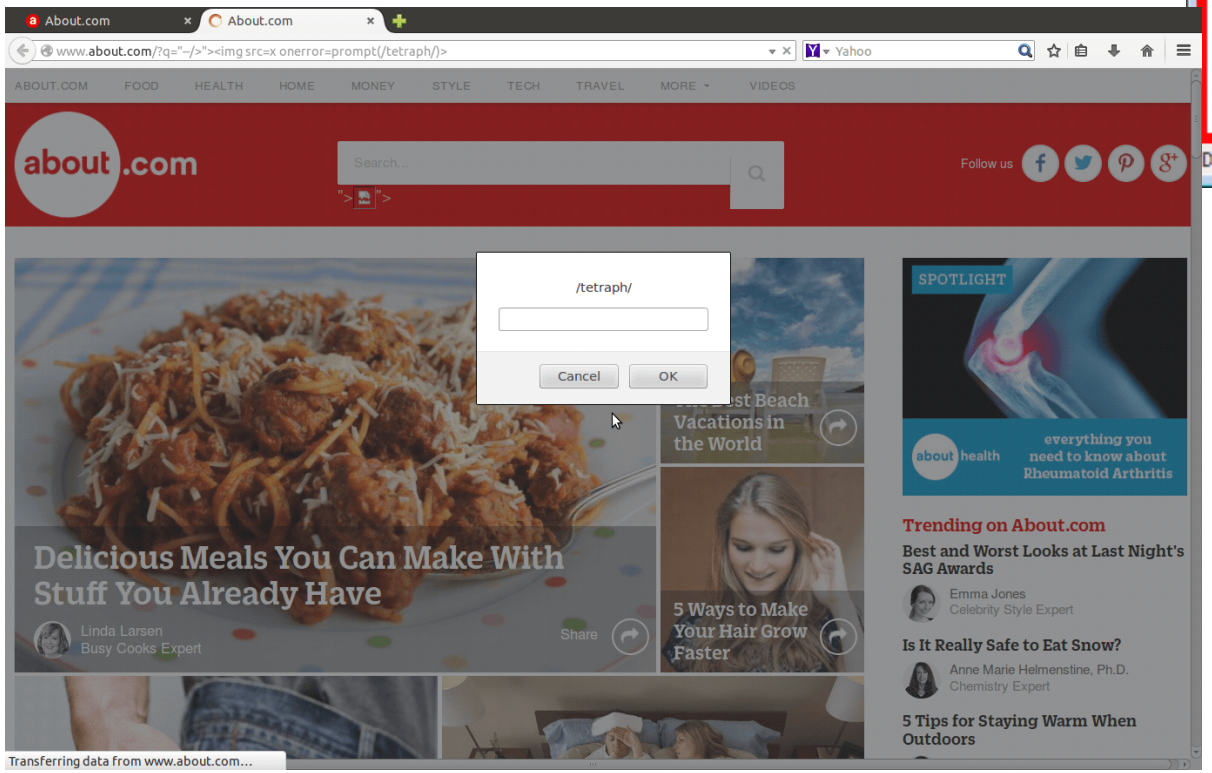
Cross-Site Scripting (XSS)

- The idea: instead of entering legitimate data in fields, enter script code (read: JavaScript) to be executed on someone's web browser
- Potential consequences:
  - Present all users with fraudulent web content
  - Steal cookie information
  - Malicious code injection
  - Annoying messages
- Not the same as phishing
- Conducting the attack: where users input data that is echoed to other users. Example: message board
- How do you embed a script into an HTML page? **`<script>`**
- XSS Payload Samples:
  - `<script>alert('XSS');</script>`
  - `<script>window.document.getElementById("SOME_ID").innerHTML='<img src="SOME_IMAGE_URL" />';</script>`
    - `Example: <script>window.document.getElementById("searchedWords").innerHTML='<img src="http://imagemacros.files.wordpress.com/2009/06/its_a_trap.jpg" />';</script>`
- Prevention and defense:
  - Remove the ability for data to be interpreted as code. Pay attention to the angle brackets. Change:
    - Change < to `<`
    - Change > to `>`
  - Draconian: filter out all special characters from user inputs

XSS Examples

- https://www.reddit.com/r/xss/





Tufts UNIVERSITY | SCHOOL OF ENGINEERING Computer Science

SQL Injection

- This is really bad! Gain access to data or even to a database that you should not have access to
- The idea: twist SQL queries via input data => access or modify data you should not have access to
- Where to attack: web applications with a database; attack form fields or URL parameters
- The culprit: the single quote
- How to determine SQL injection: errors displayed on page
- Blind SQL injection: asks the database true or false questions and determines the answer based on the applications response
- Prevention and defense:
  - Filter out special characters especially single and double quotes
  - Use prepared statements
  - Limit data and privileges that a database has access to => least privilege
- Cheat sheet and tutorial: https://www.veracode.com/security/sql-injection

SQL Injection Example

- Assume a database table named `users` exist with fields `id`, `username`, `password`. Assume there is a record for `username=batman`, `password=????`, and `id=????` (???? => who cares, doesn't matter)
- A legitimate SQL query to check if username and password exist in database table, returns user's ID: `SELECT id FROM users WHERE username='batman' AND password='foo';` => will most likely return nothing unless password for `batman` really is `foo` (unlikely)
- BUT what if instead of using foo as password, use: **`WHATEVER' OR '1'='1`**
- Now we have: `SELECT id FROM users WHERE username='batman' AND password='`**`WHATEVER' OR '1'='1`**`';` => syntactically correct, a legal SQL statement, and will always return something

Tool: sqlmap

- “open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers”
- http://sqlmap.org/
- Source code: https://github.com/sqlmapproject/sqlmap

Tool: sqlmap (continued)

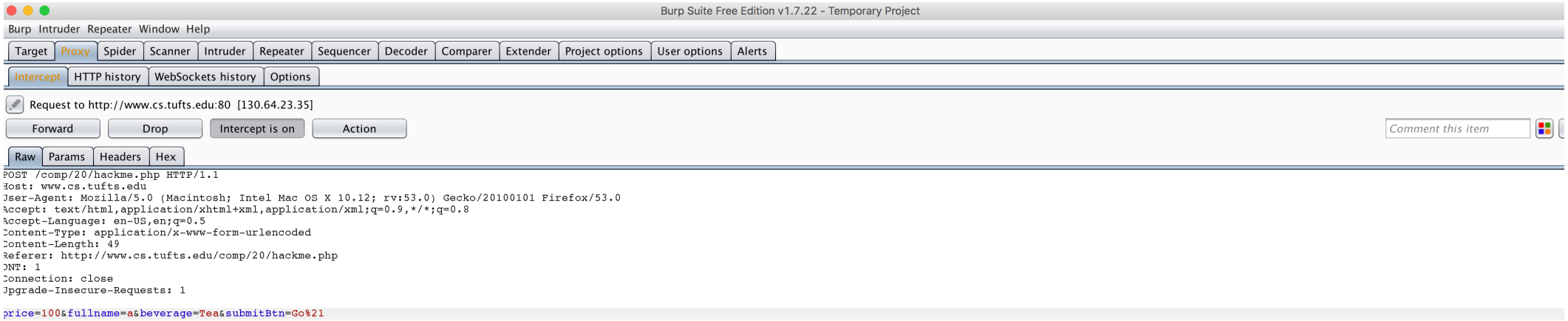
```
$ sqlmap -u https://192.168.1.66/mutillidae/index.php?page=
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.1.6#stable}
|_ -| . [,]     | .'| . |
|___|_  ['']_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal
 It is the end user's responsibility to obey all applicable local, state and federal laws. Develope
s assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 18:25:31

[18:25:31] [WARNING] provided value for parameter 'page' is empty. Please, always use only valid pa
ameter values so sqlmap could be able to run properly
[18:25:31] [INFO] testing connection to the target URL
[18:25:32] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[18:25:32] [INFO] testing if the target URL is stable
[18:25:33] [INFO] target URL is stable
[18:25:33] [INFO] testing if GET parameter 'page' is dynamic
[18:25:33] [INFO] confirming that GET parameter 'page' is dynamic
[18:25:33] [INFO] GET parameter 'page' is dynamic
[18:25:33] [WARNING] heuristic (basic) test shows that GET parameter 'page' might not be injectable
[18:25:34] [INFO] heuristic (XSS) test shows that GET parameter 'page' might be vulnerable to cross
site scripting attacks
```

Bypassing Restrictions on Input Choices

- Playground and example: http://www.cs.tufts.edu/comp/20/hackme.php
- Question: can you bypass the limited choices on form that you are given? Example: enter more than 15 characters for name and/or have “lemonade” as beverage?
- Can be applied on practically all input forms
- Conducting the attack: use a proxy program, intercept the HTTP request, modify values, have proxy send HTTP request to server
- Prevention and defense: server-side input validation

Using Burp Suite on the Hackme Playground

- With Burp turned on, intercept HTTP request after pressing “Go!” button. Under the “Proxy” tab, under ”Intercept”, modify the values for the fields `price`, `fullname`, `beverage`. Then press the “Forward” button.



Bypassing Restrictions on Input Choices: Hidden Form Values

- All hidden fields are sent to server on form submission (POST or GET)
- Very easy to identify: `<input type="hidden" name="...`
- Bad mistake: using hidden field to store sensitive information so it can be passed from one page to another. Examples: account number, password, product price
- Back-in-the-days: buy a plasma screen TV for $0.99
  http://www.edgeblog.net/2006/how-to-buy-a-plasma-for-99/
  - More details on SecurityFocus: http://www.securityfocus.com/bid/1237/discuss
  - Many software packages and vendors have plugged this hole
- Conducting the attack: use a proxy program, intercept the HTTP request, modify values, have proxy send HTTP request to server
- Prevention and defense: avoid using hidden fields for important information

Bypassing Restrictions on Input Choices: Cookie Tampering

- Conducting the attack: use a proxy program, intercept the HTTP request, modify cookie values, have proxy send HTTP request to server
- Prevention and protection: avoid storing important information such as password, administrator check (boolean value) in cookies

Cross-Site Request Forgery (XSRF or CSRF)

- Think *forgery*
- The idea: takes advantage of the fact that you are logged on to a website (e.g., online banking website); perform actions on behalf of you but without your consent –assuming that you are logged on to the website.
- Unlike Cross-Site Scripting (XSS), CSRF does NOT require JavaScript
- Prevention and defense:
  - Verify requests are coming from same origin
  - Make transaction require user interaction
  - “append unpredictable challenge tokens to each request and associate them with the user’s session” (Veracode)
  - More: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

Real XSRF Example via Zellen and Felten

- YouTube
- Circa ~2008
- Paper: https://www.cs.utexas.edu/~shmat/courses/cs378_spring09/zeller.pdf
- The idea: “to add a video to a user’s “Favorites,” an attacker simply needed to embed the following `<img>` tag on any site”
- This HTML image tag:

```
<img src="http://youtube.com/watch_ajax?action_add_favorite_playlist=1&video_id=[VIDEOID]&playlist_id=&add_to_favorite=1&show=I&button=AddvideoasFavorite"/>
```

- “An attacker could have used this vulnerability to impact the popularity of videos.”

Directory Traversal

- Also known as path traversal
- The idea: accessing files outside of the website root directory (e.g., on system) including password and configuration files
- “By manipulating variables that reference files with “dot-dot-slash (`../`)” sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files.” https://www.owasp.org/index.php/Path_Traversal
- Example: going to `http://somedomain/../../../../etc/passwd` should not render `/etc/passwd` file!
- This is not too common these days but bug still occurs because of web server misconfiguration or plain laziness
- Further reference: https://www.acunetix.com/blog/articles/directory-traversal/
- Defense:
  - Input validation, filter out special characters including "/", "." and "%"
  - Proper configuration of web server

Directory Traversal: WTF

- https://www.theregister.co.uk/2017/03/26/miele_joins_internet ofst_hall_of_shame/

The Register®
Biting the hand that feeds IT

DATA CENTER SOFTWARE **SECURITY** TRANSFORMATION DEVOPS BUSINESS PERSONAL TECH

Security 193

Dishwasher has directory traversal bug

Thanks a Miele-on for making everything dangerous, Internet of Things firmware slackers

Don't say you weren't warned: Miele went full Internet-of-Things with a network-connected dishwasher, gave it a web server, and now finds itself on the wrong end of a security bug report – *and* it's accused of ignoring the warning.

The utterly predictable vulnerability advisory on the Full Disclosure mailing list details CVE-2017-7240 – aka "Miele Professional PG 8528 - Web Server Directory Traversal." This is the builtin web server that's used to remotely control the glassware-cleaning machine from a browser.

"The corresponding embedded Web server 'PST10 WebServer' typically listens to port 80 and is prone to a directory traversal attack, therefore an unauthenticated attacker may be able to exploit this issue to access sensitive information to aide in subsequent attacks," reads the notice, dated Friday.

Proving it for yourself is simple: Using a basic HTTP GET, fetch...

```
/../../../../../../../../../../../../etc/shadow
```

Command Execution or Command Injection

- The idea: run system commands on web server (e.g., `ls, cat, ping, more`)
- Example URL before alteration: `http://sensitive/cgi-bin/userData.pl?doc=user1.txt`
- Example URL AFTER alteration (BAD!): `http://sensitive/cgi-bin/userData.pl?doc=/bin/ls`
- Source: https://www.owasp.org/index.php/Testing_for_Command_Injection_(OTG-INPVAL-013)

Command Execution or Command Injection (continued)

- Prevention and defense:
  - Input validation, filter out special characters
  - The dirty functions that will introduce risk of command execution or command injection (source: https://www.owasp.org/index.php/Testing_for_Command_Injection_(OTG-INPVAL-013)):
    - Java
      - `Runtime.exec()`
    - C/C++
      - `system`
      - `exec`
      - `ShellExecute`
    - Python
      - `exec`
      - `eval`
      - `os.system`
      - `os.popen`
      - `subprocess.popen`
      - `subprocess.call`
    - PHP
      - `system`
      - `shell_exec`
      - `exec`
      - `proc_open`
      - `eval`

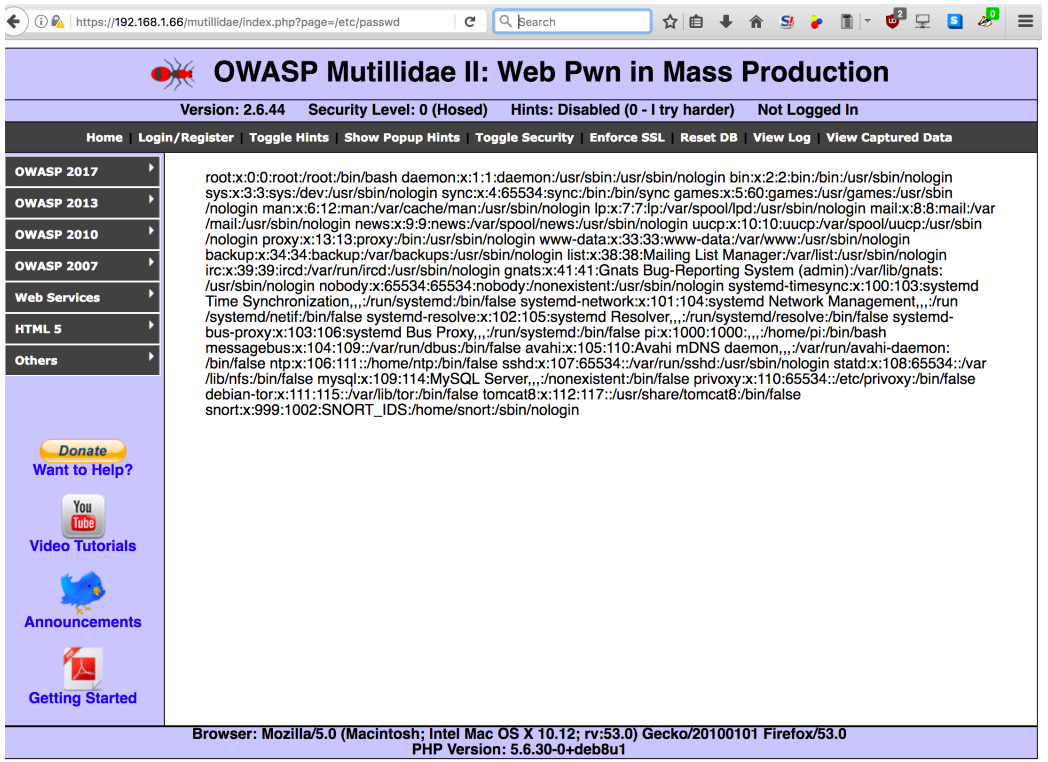
Remote and Local File Inclusion

- The idea: allow the user to submit input into file location or upload field and the input is taken for granted
- Local file inclusion: similar, but not quite the same as directory traversal or command injection; select a file on local system to use or display
- Remote file inclusion: use a remote file (e.g., URL of a website) as input
- Example URL, in Mutillidae:
  https://domain/mutillidae/index.php?page=**home.php**
- Prevention and defense:
  - Input validation, filter out special characters including "/", "." and "%"
  - In PHP, set `allow_url_fopen` and `allow_url_include` to “Off” (in `php.ini` configuration file)

Local File Inclusion Example

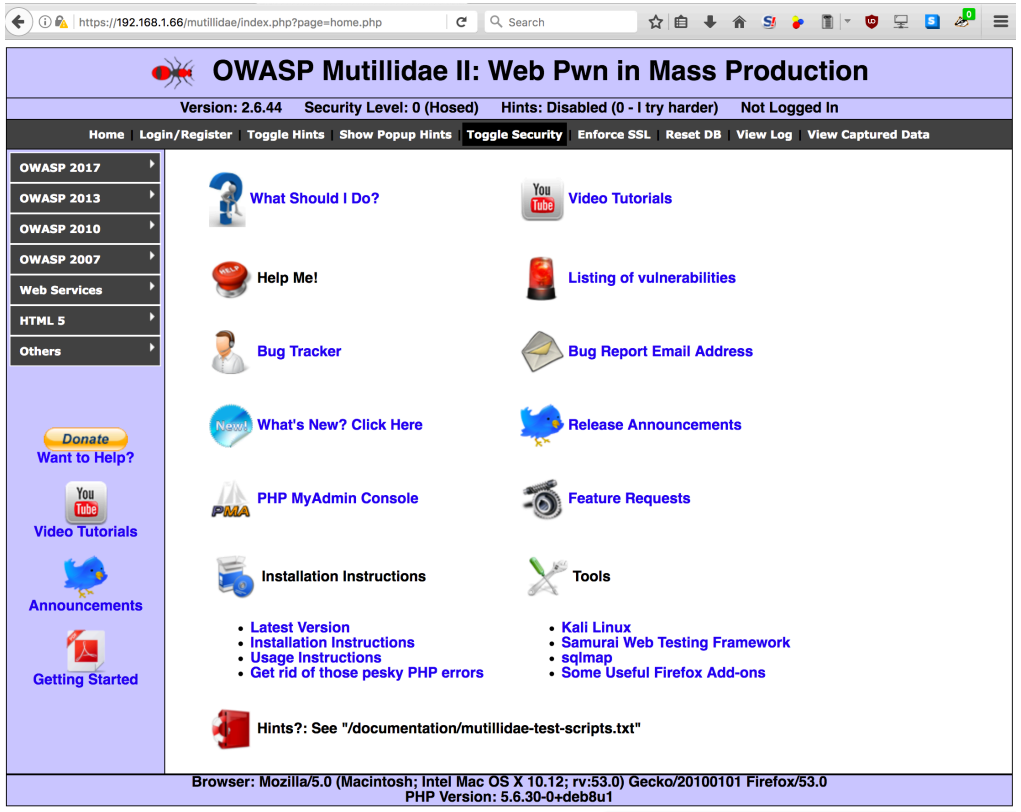
- https://domain/mutillidae/index.php?page=**home.php**

- https://domain/mutillidae/index.php?page=**/etc/passwd**

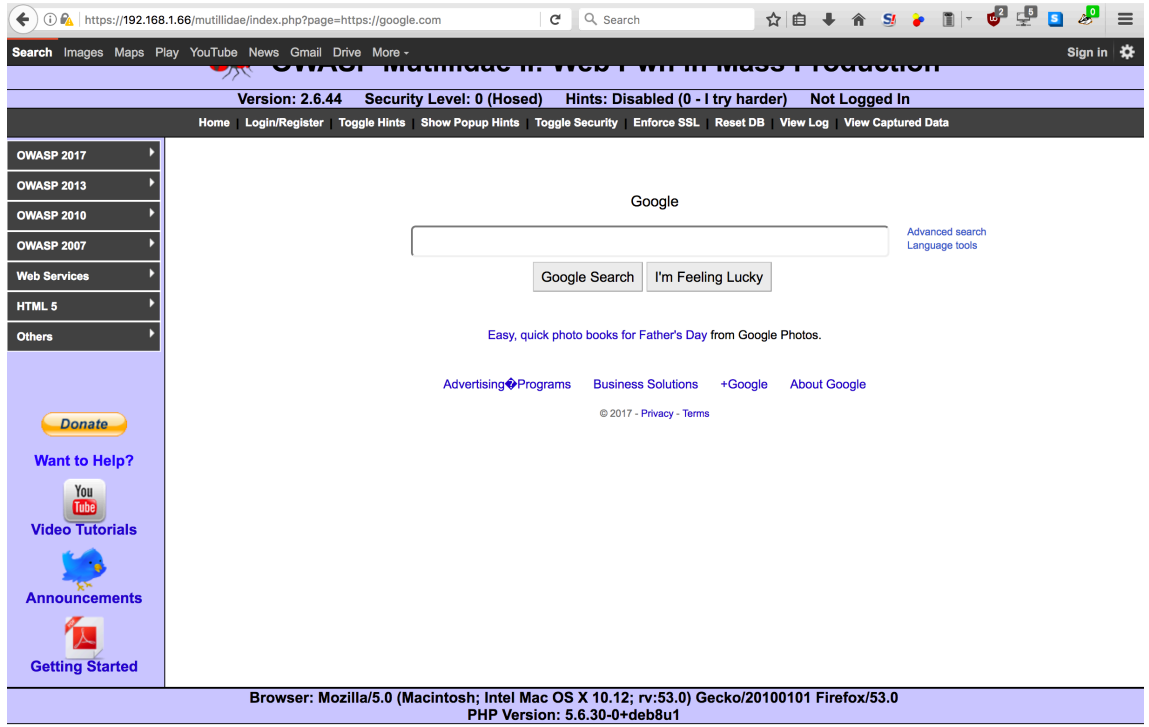


Remote File Inclusion Example

- https://domain/mutillidae/index.php?page=**home.php**



- https://domain/mutillidae/index.php?page=**https://google.com**



The Moral of the Story

- Never trust user input
- Never trust user input
- Never trust user input
- Never trust user input
- Never trust user input
- Never trust user input
- Never trust user input
- Never trust user input
- Never trust user input

Additional References

- http://blog.blackducksoftware.com/open-web-application-security-project-updated-top-10