Unsafe programming

Vera Vsevolozhsky

Agenda

- **Failure to Preserve SQL Query Structure ('SQL Injection')**
- **Information Exposure Through an Error Message**
- **Cross-Site Request Forgery (CSRF)**
- **Improper Access Control (Authorization)**
- **Client-Side Enforcement of Server-Side Security**

Failure to Preserve SQL Query Structure ('SQL Injection')

Failure to Preserve SQL Query Structure ('SQL Injection') – Intro(1)

- **Many web applications take user input from a form**
- **Often this user input is used literally in the construction of a SQL query submitted to a database. For example(PERL code):**
  - $query = “SELECT prodinfo

    FROM prodtable

    WHERE prodname = ‘” . $_POST[‘prod_search’] . “’”;
- **What is SQL injection**?

  A SQL injection attack involves placing SQL statements in the user input

Failure to Preserve SQL Query Structure ('SQL Injection') - Demonstrative Examples(1)

Product Search-input: blah’ OR ‘a’ = ‘a

- **This input is put directly into the SQL statement within the Web application:**
  - $query = “SELECT prodinfo
    FROM prodtable
    WHERE prodname = ‘” . $_POST[‘prod_search’] . “’”;

- **Creates the following SQL:**
  - SELECT prodinfo
    FROM prodtable
    WHERE prodname = ‘blah‘ OR ‘a’ = ‘a’
  - Attacker has now successfully caused the entire **prodtable** to be returned.

Failure to Preserve SQL Query Structure ('SQL Injection') - Demonstrative Examples(2)

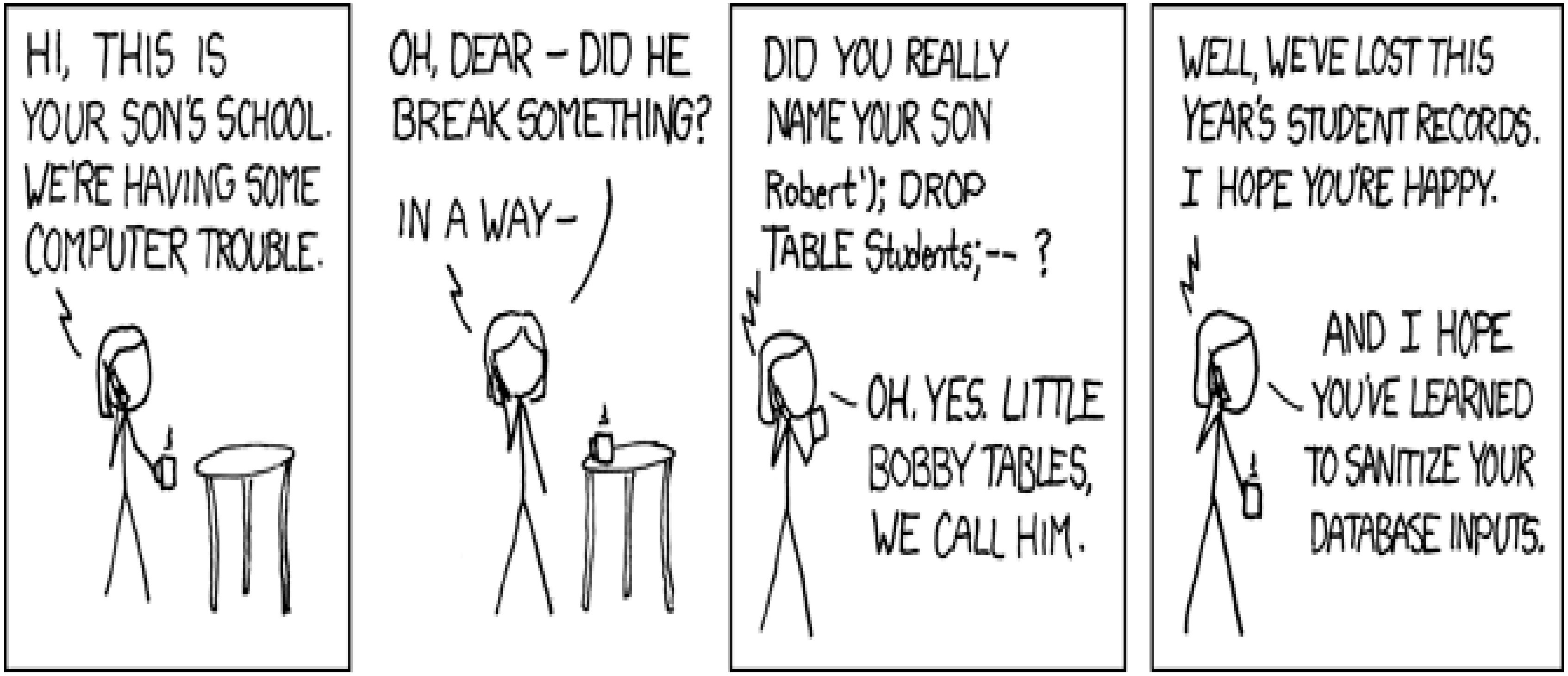
- What if the attacker had instead entered:
  - **blah‘; DROP TABLE prodinfo; --**
- Results in the following SQL:

  -
    ```
    SELECT prodinfo
    FROM   prodtable
    WHERE  prodname = ‘blah’; DROP TABLE prodinfo; --’
    ```
  - Note how comment (--) consumes the final quote (in Microsoft(R) SQL Server 2000)
- Causes the **prodinfo table** is deleted

Failure to Preserve SQL Query Structure ('SQL Injection') - other injection possibilities

- Using SQL injections, attackers can:
  - Add new data to the database
    - Could be embarrassing to find yourself selling politically incorrect items on an **eCommerce** site
    - Perform an INSERT in the injected SQL
  - Modify data currently in the database
    - Could be very costly to have an expensive product (**BMW**) suddenly be deeply ‘discounted’
    - Perform an UPDATE in the injected SQL

Failure to Preserve SQL Query Structure ('SQL Injection') - satire


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.

Failure to Preserve SQL Query Structure ('SQL Injection') - Demonstrative Examples(3)

A shell command execution injection:

'; exec master..xp_cmdshell 'dir' –

- **This input is put directly into the SQL statement:**

```
SELECT *
FROM PRODUCT
WHERE ITEM_CATEGORY='$user_input‘
```

- **Creates the following SQL:**

```
SELECT *
FROM PRODUCT
WHERE ITEM_CATEGORY='';
exec     master..xp_cmdshell 'dir' --‘
```

- **Now, this query can be broken down into:**

1. SQL query:

```
SELECT *
FROM PRODUCT
WHERE   ITEM_CATEGORY='';
```

2. SQL query, which executes the dir command in the shell:

```
exec   master..xp_cmdshell 'dir'
```

3. An MS SQL comment: --'

Failure to Preserve SQL Query Structure ('SQL Injection') - Observed Examples

- SQL injection via user name (NoseRub 0.5.2)

  (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6602)

- SQL injection in security product, using a crafted group name (web-based administration interface for iisPROTECT 2.2-r4)

  (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0377)

Failure to Preserve SQL Query Structure ('SQL Injection') - Defenses

- Use provided functions for escaping strings
  - Many attacks can be thwarted by simply using the SQL string escaping mechanism
    - ‘ → \’ and “ → \”
  - mysql_real_escape_string() is the preferred function for this
- Use lowest privileges to run the necessary tasks

Failure to Preserve SQL Query Structure ('SQL Injection') - More Defenses

- Check syntax of input for validity
  - Many classes of input have fixed languages
    - Email addresses, dates, part numbers, etc.
    - Verify that the input is a valid string in the language
    - Sometime languages allow problematic characters (e.g., ‘*’ in email addresses); may decide to not allow these
    - If you can exclude quotes and semicolons that’s good
  - Not always possible: consider the name Bill O’Reilly
    - Want to allow the use of single quotes in names

Failure to Preserve SQL Query Structure ('SQL Injection') – more defence

- **SQL injection in stored procedure:**

```
CREATE PROCEDURE sp_getUser
        @username varchar(200) = NULL AS
        DECLARE @sql nvarchar(4000)
        SELECT @sql = ‘ SELECT [name], [address] ‘ + ‘ FROM [USERS] Where
        [username] = ”’ + @username  + ””
 EXEC (@sql)
```

- **What if the attacker had instead entered:**

    badguy’;DROP TABLE USERS;

    *SELECT * FROM Countries WHERE name LIKE ‘%*

- **Then the final SQL query executed at the server will be**

```
SELECT [name], [address]
FROM [USERS] Where [username] = ‘badguy’;
DROP TABLE USERS;
SELECT * FROM Countries WHERE name LIKE ‘%’
```

Failure to Preserve SQL Query Structure ('SQL Injection') – even more defences

- **JAVA example (insecure version):**

```
Statement s = connection.createStatement();
ResultSet rs = s.executeQuery
                    ("SELECT email
                      FROM member
                     WHERE name = " + formField); // *boom*
```

- **If possible, use bound variables**
– Some libraries allow you to bind inputs to variables inside a SQL statement
– JAVA example(secure version): (from http://www.unixwiz.net/techtips/sql-injection.html)

```
PreparedStatement ps =
connection.prepareStatement ( "SELECT email
                                FROM member
                               WHERE name = ?");
            ps.setString(1, formField);
            ResultSet rs = ps.executeQuery();
```

Information Exposure Through an Error Message

Information Exposure Through an Error Message-Intro

- The software generates an error message that includes sensitive information about its environment, users, or associated data.
- The sensitive information may be valuable information on its own (such as a password), or it may be useful for launching other, more deadly attacks.
- Error information provided by the server to launch another more focused attack (SQL injection, etc.)

Information Exposure Through an Error Message-Demonstrative Examples(1)

- In the following example, sensitive information might be printed depending on the exception that occurs:

```
try {
/…/
}
catch (Exception e) {
System.out.println(e);
}
```

Sensitive information exposure
(SQL query structure
or private
information, etc.)
to the end user.

Information Exposure Through an Error Message-Demonstrative Examples(2)

- The following code generates an error message that leaks the full pathname of the configuration file :

```
$ConfigDir = "/home/myprog/config";
$uname = GetUserInput("username");
ExitError("Bad hacker!")
if ($uname !~ /^\w+$/);
   $file = "$ConfigDir/$uname.txt";

if (! (-e $file))
{
          ExitError("Error: $file does not exist");
}
```

By submitting a username that does not produce a **$file** that **exists**, an attacker could get this pathname of configuration directory.

Information Exposure Through an Error Message-Demonstrative Examples(3)

- In the following example, If an SQLException is raised when querying the database, an error message is created and output to a sensitive log file:

```
public BankAccount getUserBankAccount(String username, String accountNumber) {
BankAccount userAccount = null;
String query = null;
try {
        if (isAuthorizedUser(username))
        {
                query = "SELECT * FROM accounts WHERE owner = "
                + username + " AND accountID = " + accountNumber;
                DatabaseManager dbManager = new DatabaseManager();
                Connection conn = dbManager.getConnection();
                Statement stmt = conn.createStatement();
                ResultSet queryResult = stmt.executeQuery(query);
                userAccount = (BankAccount)queryResult.getObject(accountNumber);
        }
} catch (SQLException ex) {
        String logMessage = "Unable to retrieve account information from database,\nquery: " + query;
        Logger.getLogger(BankManager.class.getName()).log(Level.SEVERE,
                                    logMessage, ex);  }
        return userAccount;
}
```

Sensitive information exposure (database or query logic, as the table name and column names) that can be easily used in the SQL injection attack.

Information Exposure Through an Error Message-Demonstrative Examples(4)

- The following example checks validity of the supplied username and password and notifies the user of a successful or failed login:

```
my $username=param('username');
my $password=param('password');
if (IsValidUsername($username) == 1)
{
  if (IsValidPassword($username, $password) == 1)
  {
     print "Login Successful";
  }
  else
  {
     print "Login Failed - incorrect password";
  }
}
else
{
     print "Login Failed - unknown username";
}
```

Obtaining half of the necessary authentication credentials.
Correct msg.-"Login Failed – incorrect username or password."

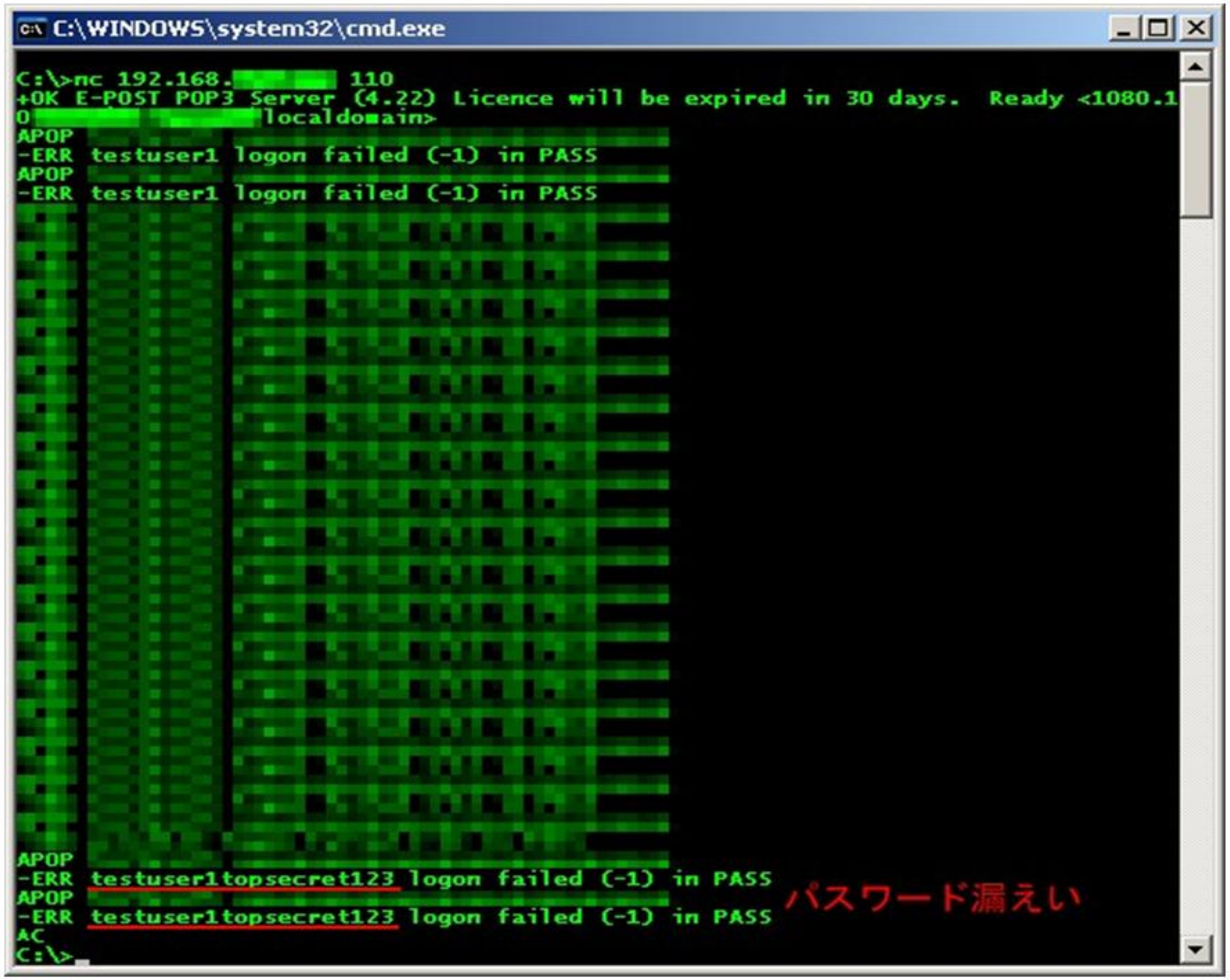
Information Exposure Through an Error Message- E-Post Mail Server vulnerability(1)

- POP3 (EPSTPOP3S.EXE) 4.22 in **E-Post Mail Server** 4.10 reveals a password in an error message after multiple APOP commands are sent. (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2049)

  - **B**y issuing several specially-crafted APOP commands to the POP3 server, the user's password will be displayed back to the attacker in the POP3 error message. The vulnerability is due to a coding error in the APOP processing code in EPSTPOP3S.EXE 4.22, and is **not** related in any way to the MD5 collision weakness in APOP authentication.

Information Exposure Through an Error

For example, in the screenshot below, the password of **testuser1** is revealed as **topsecret123**.



```
C:\>nc 192.168.[redacted] 110
+OK E-POST POP3 Server (4.22) Licence will be expired in 30 days.  Ready <1080.1
0[redacted]localdo■ain>
APOP [redacted]
-ERR testuser1 logon failed (-1) in PASS
APOP [redacted]
-ERR testuser1 logon failed (-1) in PASS
[redacted]
APOP [redacted]
-ERR testuser1topsecret123 logon failed (-1) in PASS
APOP [redacted]
-ERR testuser1topsecret123 logon failed (-1) in PASS
AC
C:\>
```

Information Exposure Through an Error Message-Defence

- Containing minimal details
- Tradeoff: to be cryptic or not cryptic
- Not revealing methods that were used to determined the error.
- Tracking errors in the log file
- Do not record highly sensitive info’
- Message standardization
- Product should be compiled in the release mode

Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) Intro

- **What is CSRF?**

  -**T**he web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.

- **Consequences**

  - **A**n attacker performs could effectively perform any operations as the victim (think banking)

  - **O**btaining complete control over the web application.

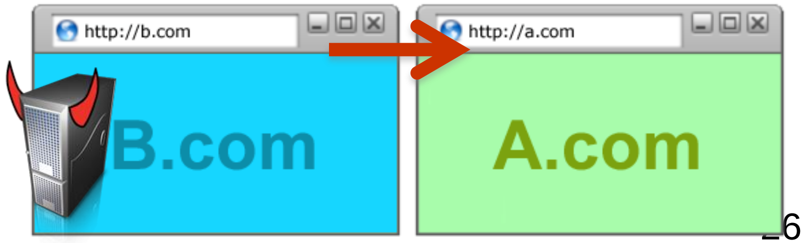
  - **D**eleting or stealing data (GMAIL – steal contacts)

Classic CSRF attack-example(1)

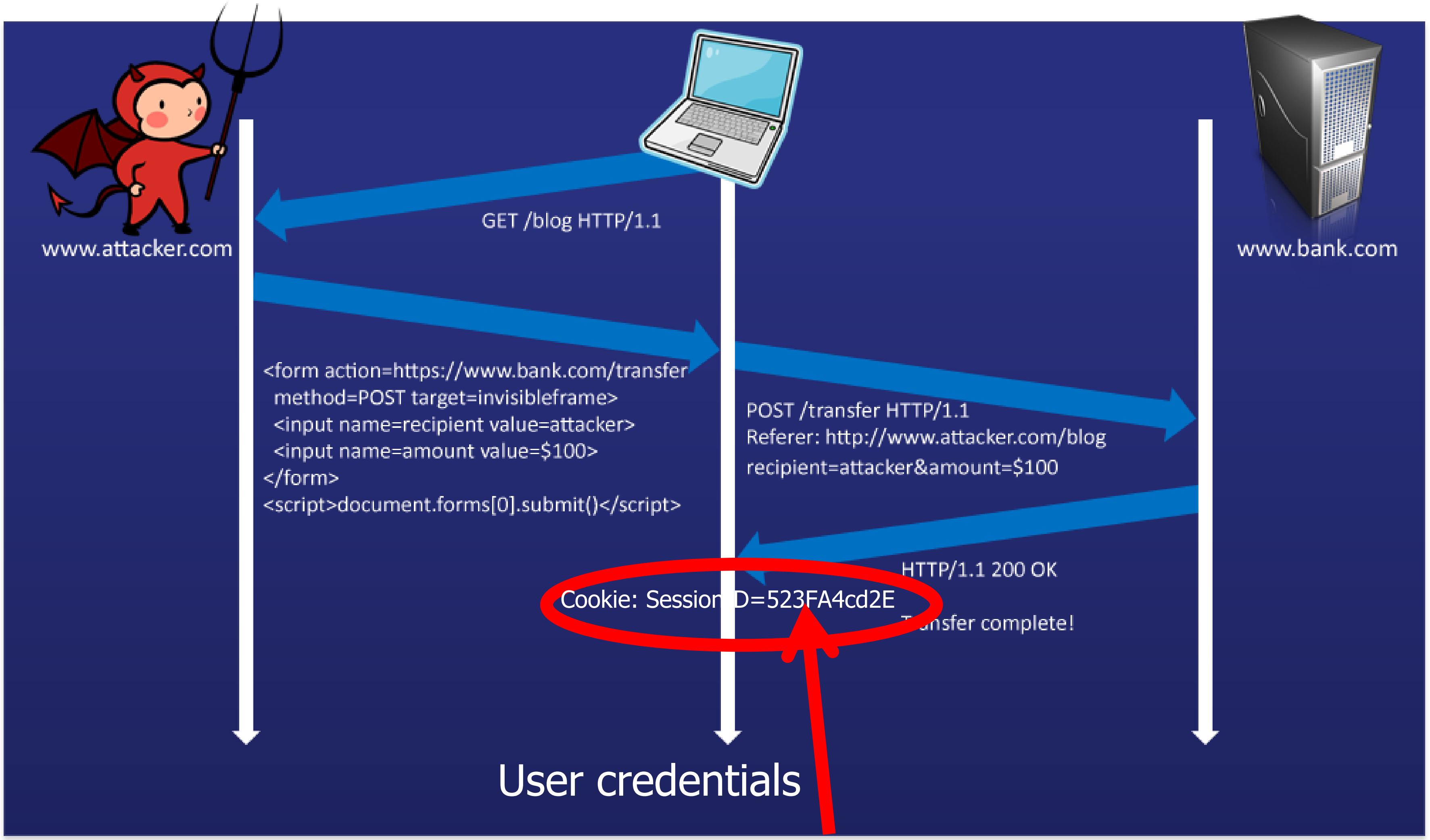
- User visits victim site site
  - Logs in (save credential in **browser session cookie**)

- User loads attacker's site
  - Or encounters attacker's iframe on another site
  - JavaScript - OnClick(), OnMouseOver()

- http//b.com sends HTTP requests to victim
  - Victim site assumes requests originate from itself



Classic CSRF attack-example (2)



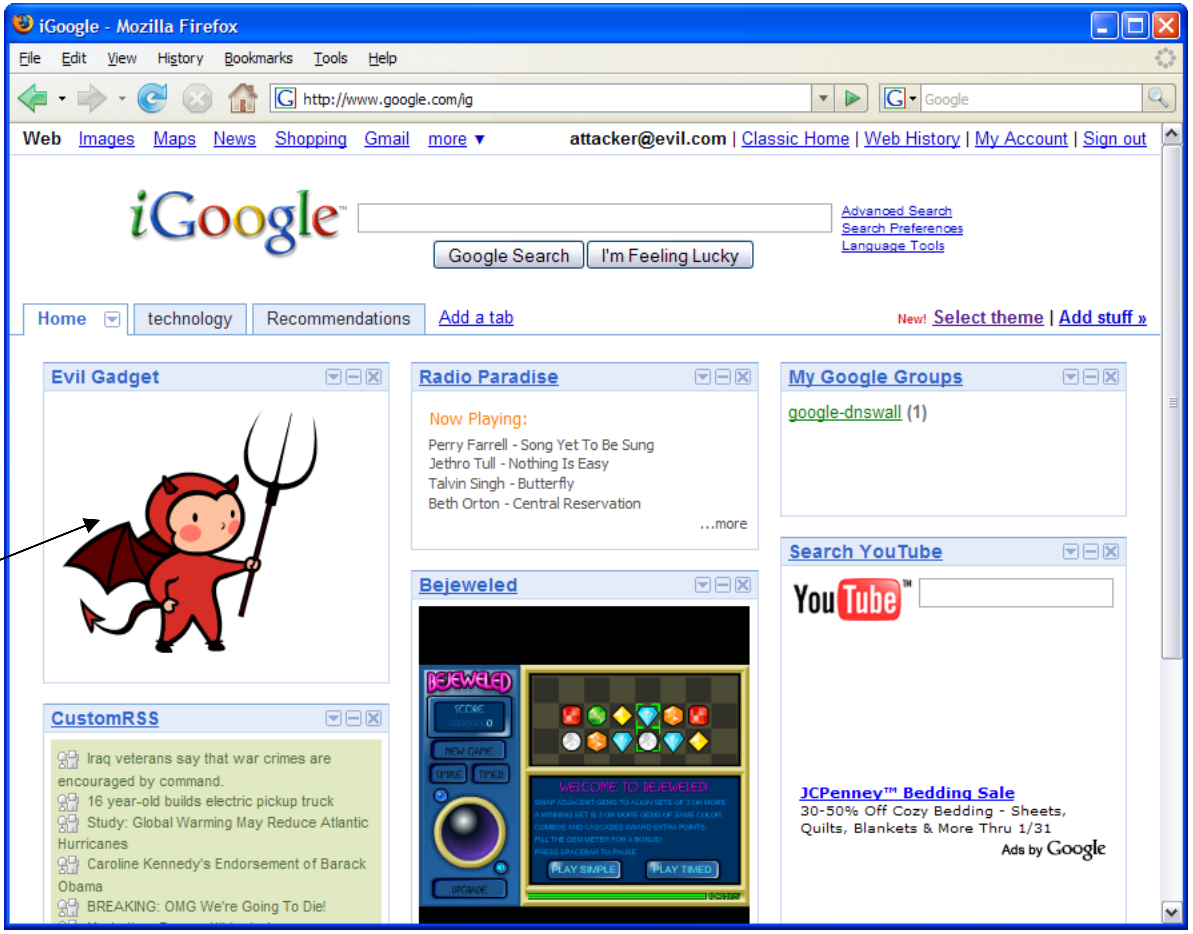
```
<form action=https://www.bank.com/transfer
  method=POST target=invisibleframe>
  <input name=recipient value=attacker>
  <input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E
```

```
HTTP/1.1 200 OK

Transfer complete!
```

CSRF attack-example (3)


iGoogle - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.google.com/ig
Google
Web Images Maps News Shopping Gmail more ▼
attacker@evil.com | Classic Home | Web History | My Account | Sign out
iGoogle
Google Search I'm Feeling Lucky
Advanced Search
Search Preferences
Language Tools
Home technology Recommendations Add a tab
New! Select theme | Add stuff »
Evil Gadget
Radio Paradise
My Google Groups
Search YouTube
Bejeweled
CustomRSS
YouTube

Now Playing:
Perry Farrell - Song Yet To Be Sung
Jethro Tull - Nothing Is Easy
Talvin Singh - Butterfly
Beth Orton - Central Reservation
...more

google-dnswall (1)

Iraq veterans say that war crimes are encouraged by command.
16 year-old builds electric pickup truck
Study: Global Warming May Reduce Atlantic Hurricanes
Caroline Kennedy's Endorsement of Barack Obama
BREAKING: OMG We're Going To Die!

JCPenney™ Bedding Sale
30-50% Off Cozy Bedding - Sheets, Quilts, Blankets & More Thru 1/31
Ads by Google

Inline Gadgets

CSRF – Defenses/Prevention(1)

- When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.
- Logging out of sites
- Avoiding their "remember me" features
- Limiting the lifetime of session cookies
- Not displaying external images
- Not clicking links in spam or untrusted e-mails

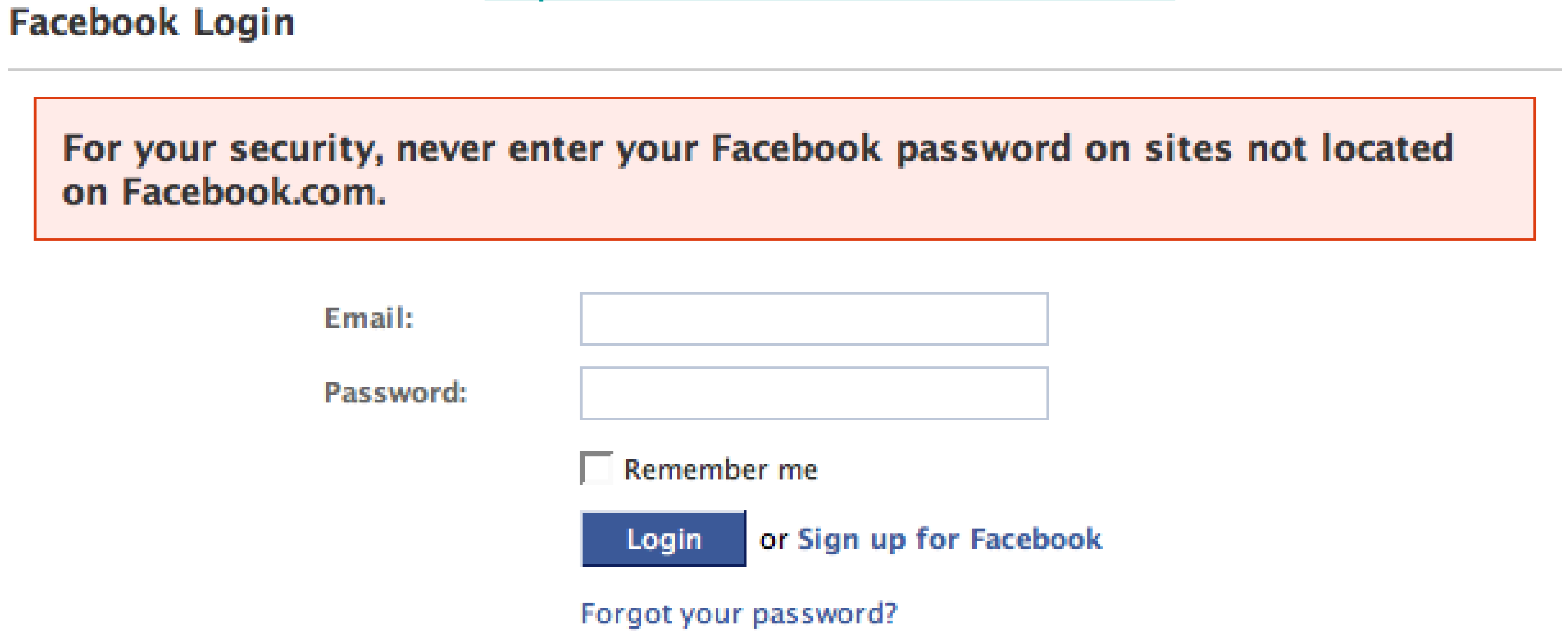
CSRF – Defenses(2)

•HTTP Referer Validation

-Check the HTTP Referer header to see if the request originated from an expected page:

-HTTP Referer header : http://www.facebook.com - **YES**

-HTTP Referer header : http://www.attacker-facebook.com - **NO**


Facebook Login

For your security, never enter your Facebook password on sites not located on Facebook.com.

Email:

Password:

Remember me

CSRF – Defenses(3)

- **Secret Validation Token in HTML**

  -**T**okens are the inserted within the HTML forms and links associated with sensitive server-side operations.

```
<form action="/transfer.do" method="post">
<input type="hidden" name="CSRFToken"
value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWEwYzU1YWQwMTVhM2JmNGYxYjJiMGI4MjJjZDE1ZD
ZjMTVi MGYwMGEwOA=="> … </form>
```

  -**T**okens are randomly generated such that it cannot be guessed by an attacker *(java.security.SecureRandom class)* .

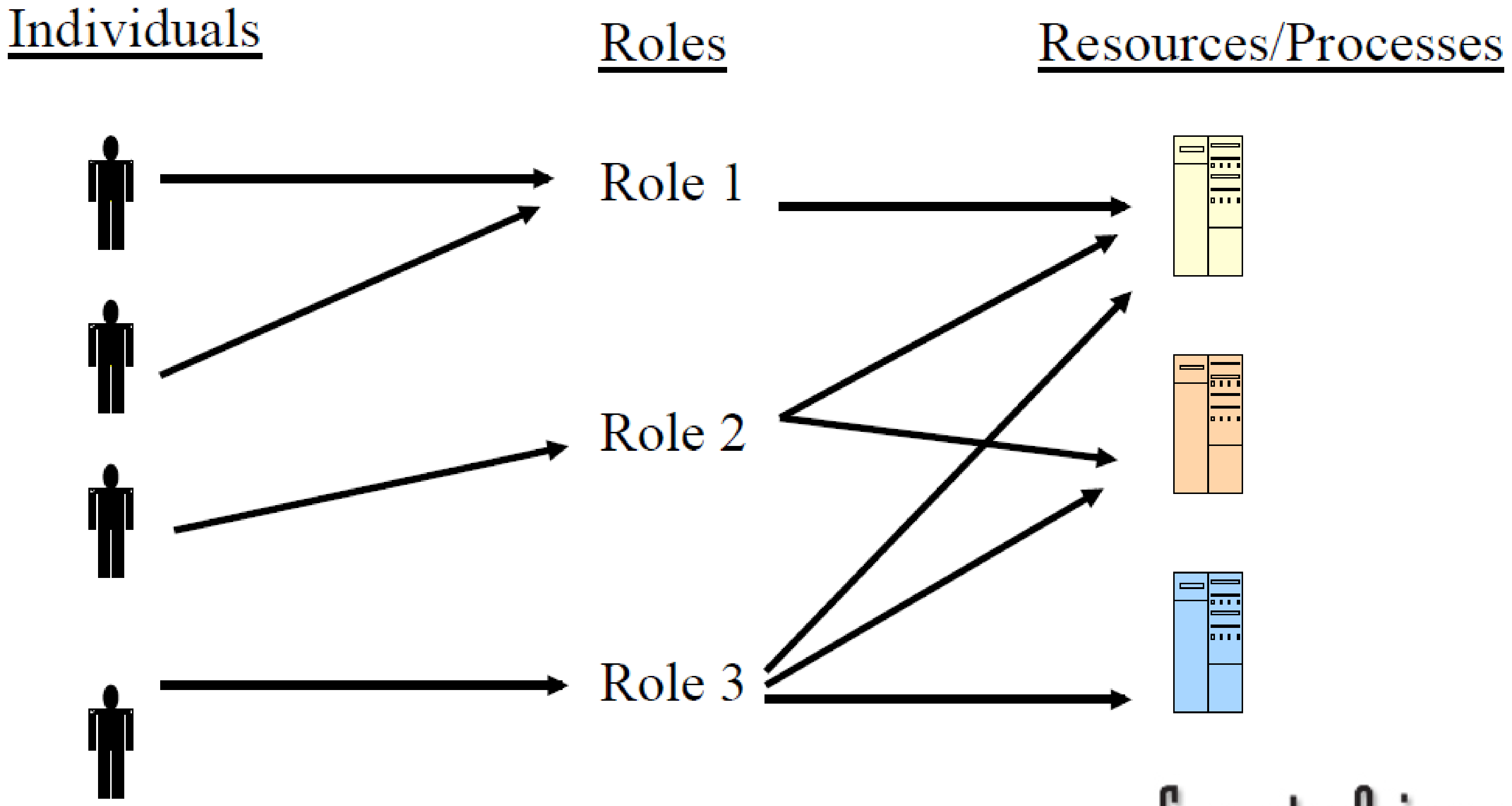
  -**D**evelopers need only generate this token once for the current session.

  - **A**fter initial generation of this token, the value is stored in the session and is utilized for each subsequent request until the session expires.

  - **V**erify the existence and correctness of this token in the server side

Improper Access Control

Role-Based Access Control



User’s change frequently, roles don’t

From: csrc.nist.gov/rbac/alvarez.ppt

Improper Access Control - Intro

- **What is Improper Authorization?**

  - In general, applications should allow access only to those who are permitted. If you don't ensure that your software's users are only doing what they're allowed to, then attackers will try to exploit your improper authorization and...

- **Consequences**

  - information exposures
  - denial of service
  - arbitrary code execution
  - read sensitive data from a store that is not properly restricted
  - read files or directories
  - modify application data; modify files or directories

- **Authorization weaknesses causes**

  - Lack of understanding about the underlying technologies (assuming cookies couldn't be modified by attacker)

Java Example:

```
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
Cookie c = cookies[i];
if (c.getName().equals("role"))
  {
     userRole = c.getValue();
  }
}
```

Improper Access Control – Demonstrative Examples(2)

- This program intends to authenticate the user before deciding whether a private message should be displayed. Assume that **LookupMessageObject**() ensures that the $id argument is numeric, constructs a filename based on that id, and reads the message details from that file. Also assume that the program stores all private messages for all users in the same directory.

```
if (! AuthenticateUser($q->param('username'),
                       $q->param('password')))
 {
        ExitError("invalid username or password");
 }
my $id = $q->param('id');
DisplayPrivateMessage($id);
```

- While the program does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other users.

Improper Access Control – Observed Examples(2)

- Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords. **(Mevin Productions Basic PHP Events Lister 2.0)**

(http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3168)

- Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users. **(CuteFlow 2.10.3 and 2.11.0_c).**

(http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2960)

- Database server does not use appropriate privileges for certain sensitive operations **(PostgreSQL 8.4)**

(http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3230)

- Terminal server does not check authorization for guest **access(Sun Solaris 10)**

(http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2282)

Improper Access Control – Potential Mitigations(1)

- **Phase: Architecture and Design**

  -**D**ivide your application into anonymous, normal, privileged, and administrative areas.

  -**R**educe the attack surface by carefully mapping roles with data and functionality.

  -**P**erform access control checks related to your business logic.

  For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor.

  -**U**se a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid (for example, JAAS Authorization Framework & OWASP ESAPI Access Control feature).

Improper Access Control – Potential Mitigations(2)

- **Phase: System Configuration; Installation**

  -**U**se the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs that represents who/what has permissions to a given object (for example, in **UNIX** there is r/w/ex permission, users are divided into three classes for file access: owner, group owner, and all other users, in **Windows NT**, there are four basic types of permissions for files: “No access”, “Read / Change access”, and “Full control“, etc.)

Client-Side Enforcement of Server-Side Security

Client-Side Enforcement of Server-Side Security – example

- **Client** – attacker modifies the client-side behaviour (validation) & performs SQL Injection:

```
SELECT    prodinfo
FROM      prodtable
WHERE     prodname = ‘blah‘ OR ‘a’ = ‘a’
```

- **Server** – relies on protection mechanisms placed on the client side
- **Result** – Revealing **prodtable** info’

Client-Side Enforcement of Server-Side Security – Intro

- **What is Client-Side Enforcement of Server-Side Security?**

  - **S**erver relies on protection mechanisms placed on the client side, an attacker can modify the client-side behaviour to bypass the protection mechanisms resulting in potentially unexpected interactions between the client and server.

- **Common Consequences**

  - **C**lient-side validation checks can be easily bypassed, allowing malformed or unexpected input to pass into the application, potentially as trusted data. This may lead to unexpected states, behaviours and possibly a resulting crash.

  - **C**lient-side checks for authentication can be easily bypassed, allowing clients to escalate their access levels and perform unintended actions.

- **Enabling Factors for Exploitation**

  -**C**lient/**S**erver model

Client-Side Enforcement of Server-Side Security - Demonstrative Examples – Optional

Client:

```
$server = "server.example.com";
$username = AskForUserName();
$password = AskForPassword();
$address = AskForAddress();
$sock = OpenSocket($server, 1234);
writeSocket($sock, "AUTH
    $username $password\n");
$resp = readSocket($sock);
if ($resp eq "success") {
# username/pass is valid, go ahead
    and update the info!
writeSocket($sock, "CHANGE-
    ADDRESS $username
    $address\n";
}
else {
print "ERROR: Invalid
    Authentication!\n";
}
```

Server:

```
$sock = acceptSocket(1234);
($cmd, $args) = ParseClientRequest($sock);
if ($cmd eq "AUTH") {
($username, $pass) = split(/\s+/, $args, 2);
$result = AuthenticateUser($username, $pass);
writeSocket($sock, "$result\n");
# does not close the socket on failure; assumes the
# user will try again
}
elsif ($cmd eq "CHANGE-ADDRESS") {
if (validateAddress($args)) {
$res = UpdateDatabaseRecord($username, "address",
    $args);
writeSocket($sock, "SUCCESS\n");
}
else {
writeSocket($sock, "FAILURE -- address is malformed\n");
}
}
```

Client-Side Enforcement of Server-Side Security - Observed Examples

- ASP program allows upload of .asp files by bypassing client-side checks (OzzyWork Gallery 2.0)

  (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6994)

- Steganography products embed password information in the carrier file, which can be extracted from a modified client

  (SecureKit Steganography 1.7.1 and 1.8)

  (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0163)

- Client allows server to modify client's configuration and overwrite arbitrary files (The Perforce client)

  (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0100)

Client-Side Enforcement of Server-Side Security - Potential Mitigations

- **Phase: Architecture and Design**

  - Ensure that security checks that are performed on the client side are duplicated on the server side. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely

- **Phase: Testing**

  - Fuzz testing: providing invalid, unexpected, or random data to the inputs and monitoring

  - **Robustness testing:** exceptional inputs

  - Use tools and techniques that require manual (human) analysis

AND… To sum up

- **Discussed Topics:**

  - Failure to Preserve SQL Query Structure ('SQL Injection')

  - Information Exposure Through an Error Message

  - Cross-Site Request Forgery (CSRF)

  - Improper Access Control (Authorization)

  - Client-Side Enforcement of Server-Side Security

- **What we saw?**

  - Demonstrative, observed examples and different kinds of defenses

BE careful!!!