Securing Apache and PHP

Justin Mayhue and Christopher Pau

Motivation

- Via LAMP (Linux, Apache, MySQL, and PHP) web hosts have a popular, free OS/software solution
- According to Netcraft, over 47% of webservers use the Apache as of October 2007
- Popularity makes for an easy target, as oftentimes configurations are not well thought-out

Who should be concerned?

- **Webmasters** – installation of scripts (permissions errors: chmod 777? 755?)
- **Web Developers** – creating software for various configurations (what is enabled? Disabled?)
- **Web Hosts** – security of their server and possible abuse
- **Average Users** – where is their data saved? Who can access it? How is it secured?

Background: Apache

- Free, open-source, Unix-based webserver software available for Linux, Mac OSX, Windows, and other platforms
- Design features revolve around modules used to control, interpret, and deliver web content to clients
- Supports virtual hosting – multiple users can host independent sites on the same server platform

Background: PHP

- PHP (**PHP: Hypertext Processor) is** a scripting language aimed at creating dynamic content
- Like Apache, it is free, open-source software available for a variety of operating systems
- Works with Apache to filter user input and provide content for Apache to serve back to the users

Theory: CGI Binaries

- CGI (**Common Gateway *Interface)*** is a protocol for exchanging information between a server and an external application
- Under this protocol, the external application is run upon each request (i.e., a new process is created), and this application closes after delivering output to the server
- PHP was originally designed to run as a set of CGI ***binaries***
- *#!/usr/local/bin/php*

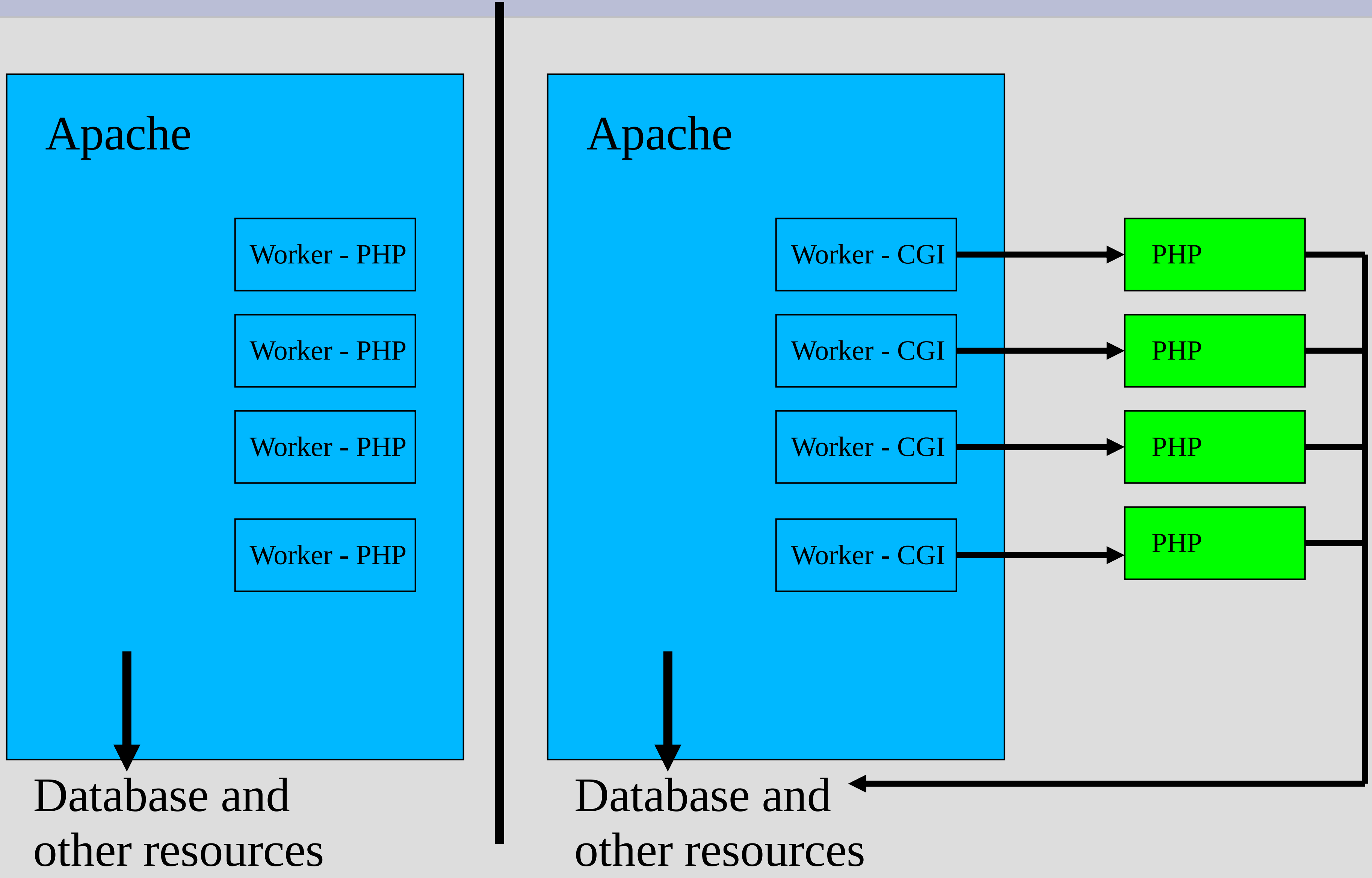
Theory: Apache Modules

- Rather than creating an instance of a process for every request, Apache provides functionality for persistent modules to run
- These modules can handle certain types of requests indefinitely without terminating
- PHP has been adapted to run as an Apache module, as well, and was used this way in the Web Security lab

Apache Module vs. CGI Binaries

- Which is optimal for running PHP?
- PHP as an Apache module yields speed
  - Only one instance of PHP, the module itself, needs to be running at one time
  - Resources such as dedicated connections to a MySQL database can be preserved across sessions
- Also, this allows PHP to be configured via .htaccess directives – an Apache-specific function

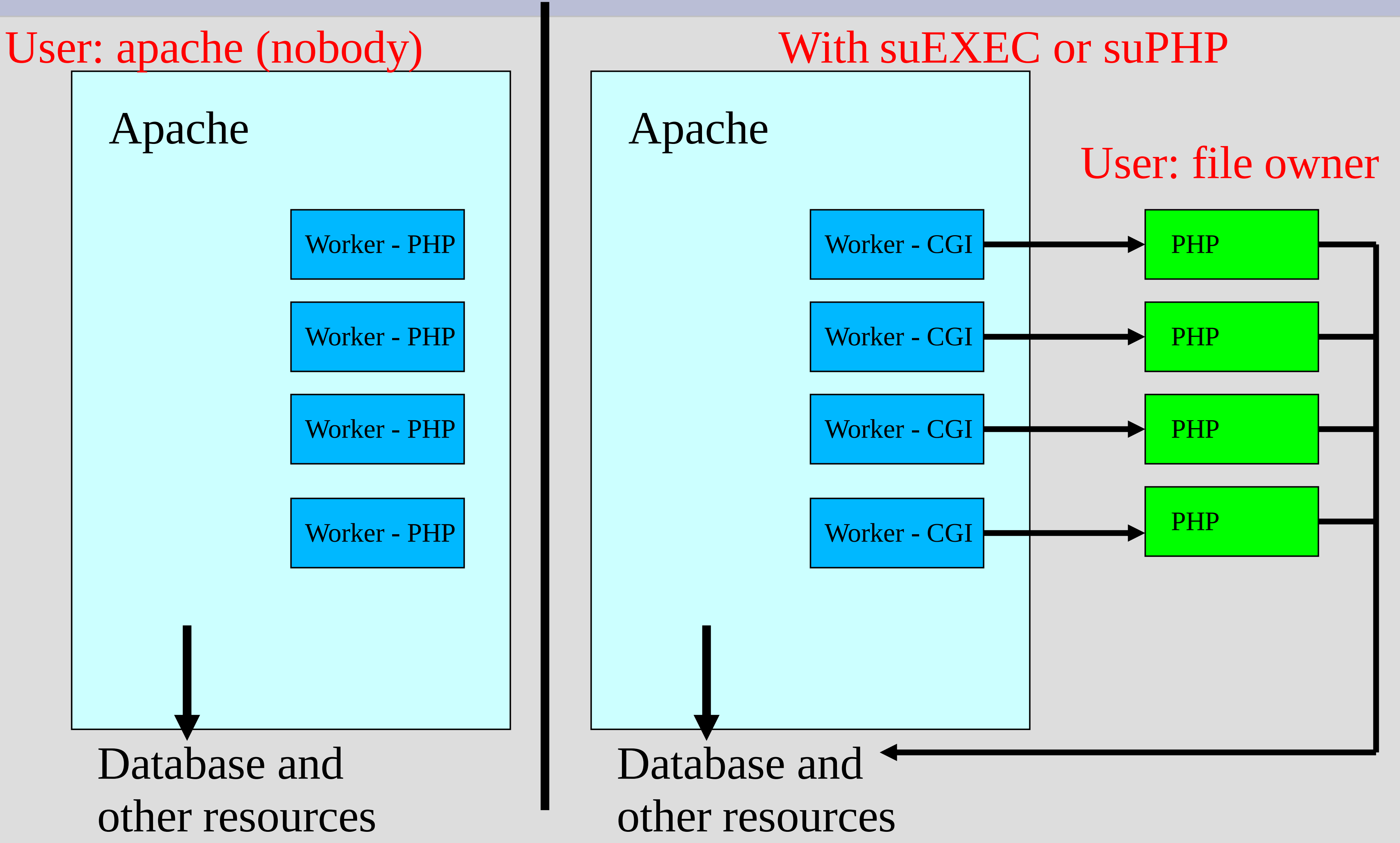
Apache Module vs. CGI Binaries



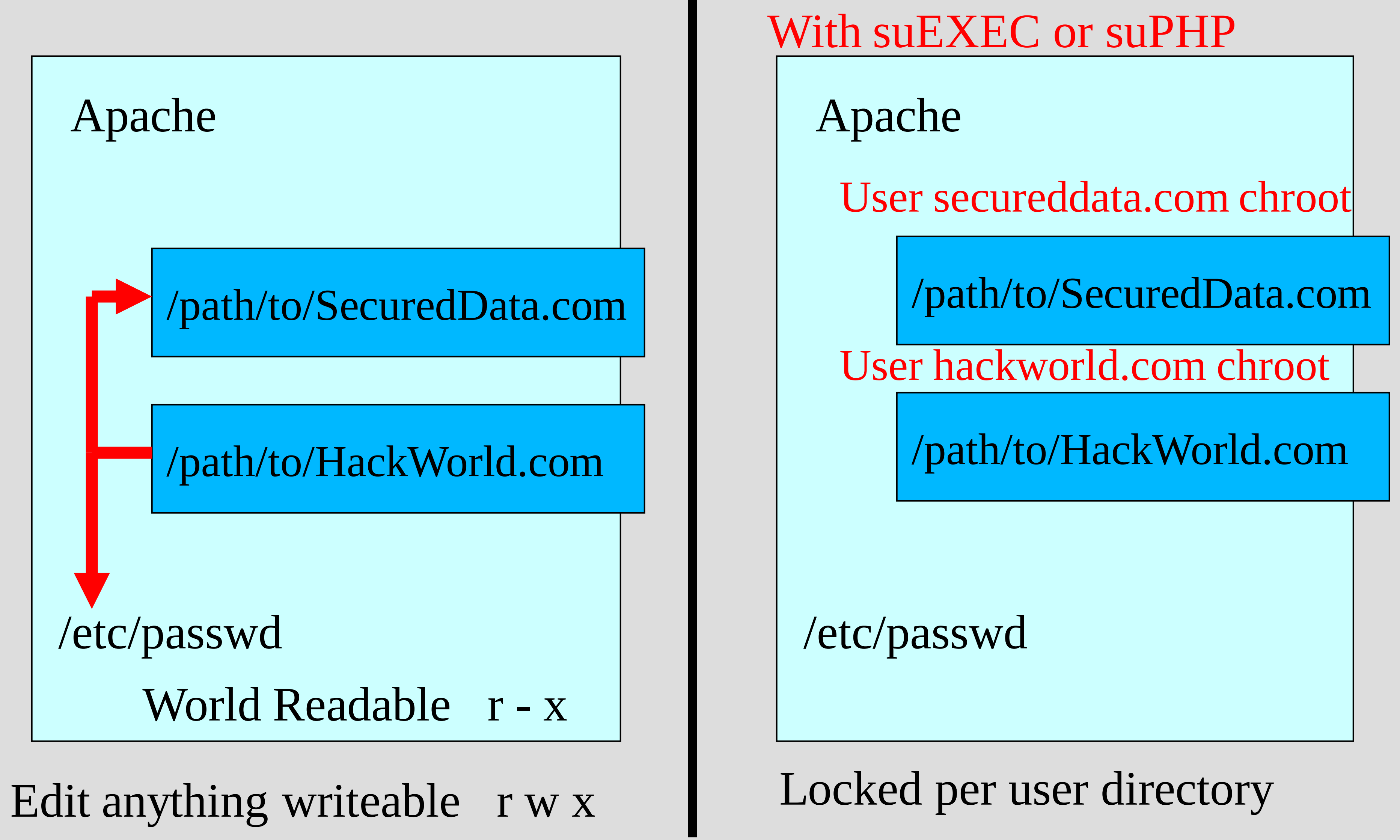
Apache Module vs. CGI Binaries

- However, Apache runs as a separate user, inheriting permissions of “nobody” -- thus all resources PHP uses must be accessible to “nobody” as well
- PHP as a CGI binary favors security
  - CGI binaries can interface with an Apache module (such as suEXEC or suPHP) to run as a separate user
  - This allows separate permissions for multiple users and multiple resource needs in a shared hosting environment

Apache Module vs. CGI Binaries



Apache Module vs. CGI Binaries



PHP Safe Mode

- As an alternative to running PHP as a CGI binary for increased permissions security, PHP provides the Safe Mode configuration
- Provides user permissions security
- Places limitations on the file structure PHP can access (PHP `fopen`)
- Can disable the use of certain potentially vulnerable functions

PHP Safe Mode

- However, such restrictions are considered “architecturally incorrect” -- PHP itself is blocking and manipulating its own normal operation
- Can easily be bypassed if functions are allowed to execute on command line

```
exec(“cat stuff >> /etc/passwd”)
```

- Will not be included in future versions PHP 6 and beyond
- Still, frequently used by many hosts

Alternative Solutions: ModSecurity

- ModSecurity is an Apache module designed as a web application firewall
- Operates at application layer
  - Protocol-level firewalls used in routers and gateway-level machines usually filter traffic based on source and destination
  - ModSecurity is generally used to filter traffic based upon the contents – including both the headers and payload data
- Rules-based, applied before sending traffic to handling application or module

Alternative Solutions: ModSecurity

Default rules block

- Protocol violations
- Protocol anomalies
- Request Limits
- Http Policy
- Bad Robots
- Generic Attacks
- Trojans
- Outbound Connections

Alternative Solution: Suhosin and PHP-Hardening Patch

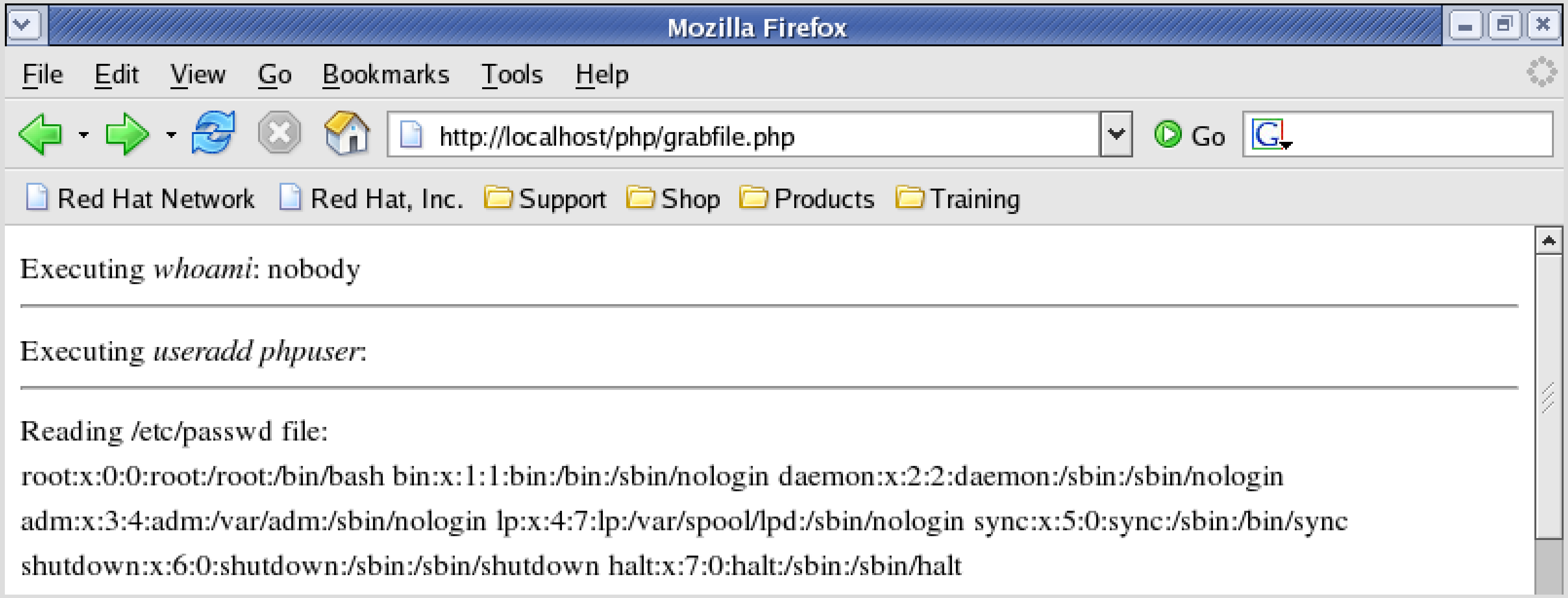
- Provides a third-party alternative to manual configuration solutions previously discussed
- Protects against known vulnerabilities such as buffer overflows, as well as PHP core vulnerabilities that could potentially be exploited
- Similar in concept to libsafe and other patch-based protection schemes

In the Lab

- Part I: Analyze PHP as an Apache module and Safe Mode as a security option
- Part II: Analyze PHP as a CGI binary and suPHP as a flexibility extension
- Part III: Explore mod_security as an application-layer firewall solution to security exploits

Part I: Apache Module

- PHP is already installed in lab as an Apache module from Web Security lab
- grabfile.php and newfile.php: test PHP's permissions for reading, writing, and modifying files



Executing *whoami*: nobody

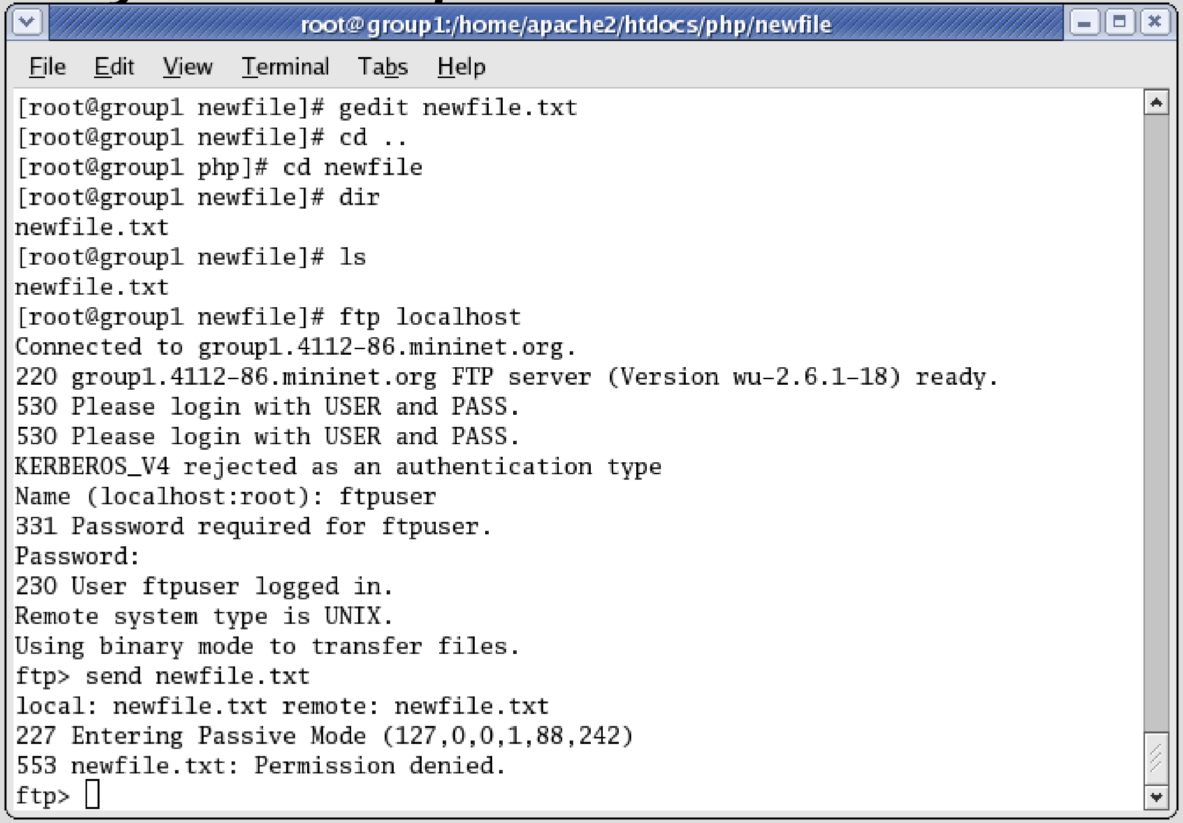
Executing *useradd phpuser*:

Reading /etc/passwd file:

```
root:x:0:0:root:/root:/bin/bash bin:x:1:1:bin:/bin:/sbin/nologin daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown halt:x:7:0:halt:/sbin:/sbin/halt
```

Part I: Apache Module

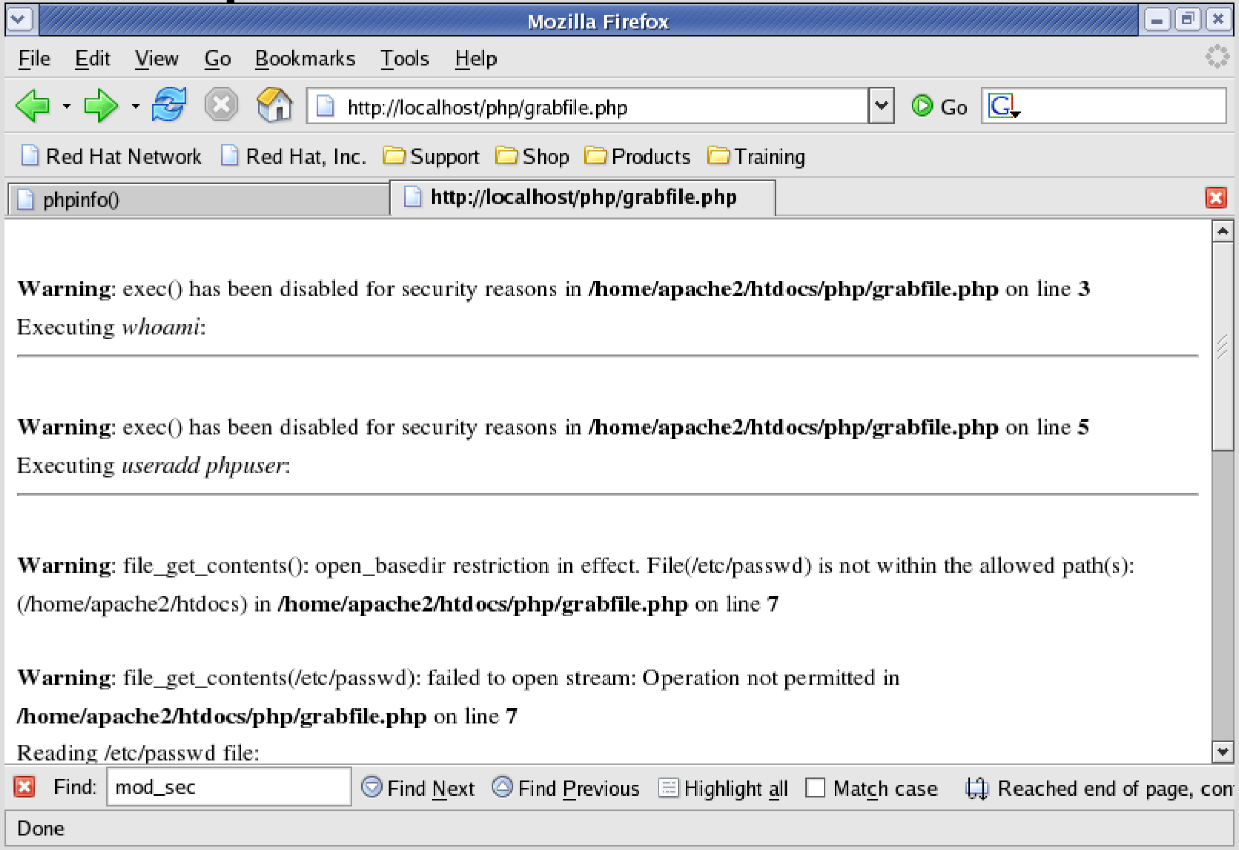
- Permission issue example: separate user, *ftpuser, attempting to modify an example file written by PHP/Apache*


root@group1:/home/apache2/htdocs/php/newfile
File Edit View Terminal Tabs Help

```
[root@group1 newfile]# gedit newfile.txt
[root@group1 newfile]# cd ..
[root@group1 php]# cd newfile
[root@group1 newfile]# dir
newfile.txt
[root@group1 newfile]# ls
newfile.txt
[root@group1 newfile]# ftp localhost
Connected to group1.4112-86.mininet.org.
220 group1.4112-86.mininet.org FTP server (Version wu-2.6.1-18) ready.
530 Please login with USER and PASS.
530 Please login with USER and PASS.
KERBEROS_V4 rejected as an authentication type
Name (localhost:root): ftpuser
331 Password required for ftpuser.
Password:
230 User ftpuser logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> send newfile.txt
local: newfile.txt remote: newfile.txt
227 Entering Passive Mode (127,0,0,1,88,242)
553 newfile.txt: Permission denied.
ftp> 
```

Part I: Safe Mode

- Safe Mode is then enabled via php.ini, and the tests repeated, as well as including remote scripts in PHP files


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/php/grabfile.php
Go
Red Hat Network
Red Hat, Inc.
Support
Shop
Products
Training
phpinfo()
http://localhost/php/grabfile.php

Warning: exec() has been disabled for security reasons in **/home/apache2/htdocs/php/grabfile.php** on line **3**
Executing *whoami*:

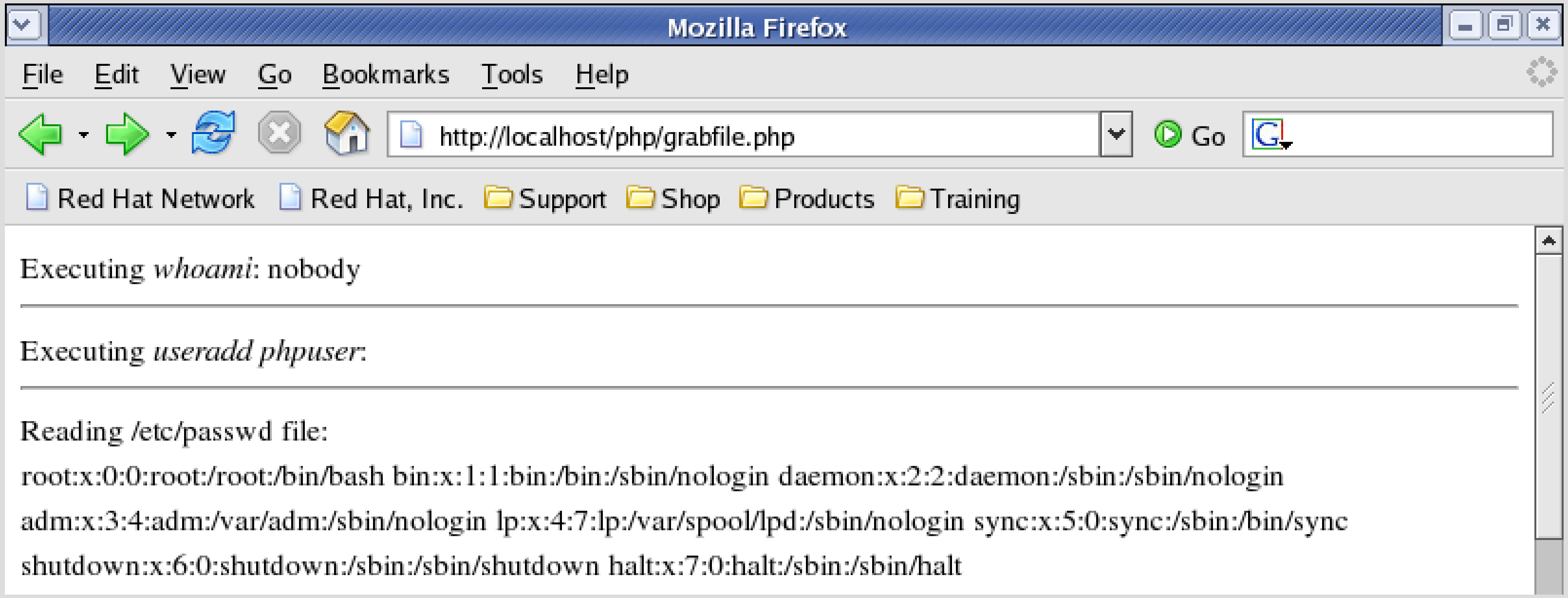
Warning: exec() has been disabled for security reasons in **/home/apache2/htdocs/php/grabfile.php** on line **5**
Executing *useradd phpuser*:

Warning: file_get_contents(): open_basedir restriction in effect. File(/etc/passwd) is not within the allowed path(s): (/home/apache2/htdocs) in **/home/apache2/htdocs/php/grabfile.php** on line **7**

Warning: file_get_contents(/etc/passwd): failed to open stream: Operation not permitted in **/home/apache2/htdocs/php/grabfile.php** on line **7**
Reading /etc/passwd file:

Part II: PHP as a CGI Binary

- Apache is then installed as a CGI binary, and http.conf is modified accordingly
- The tests are then repeated – still runs as "nobody" user



Executing *whoami*: nobody

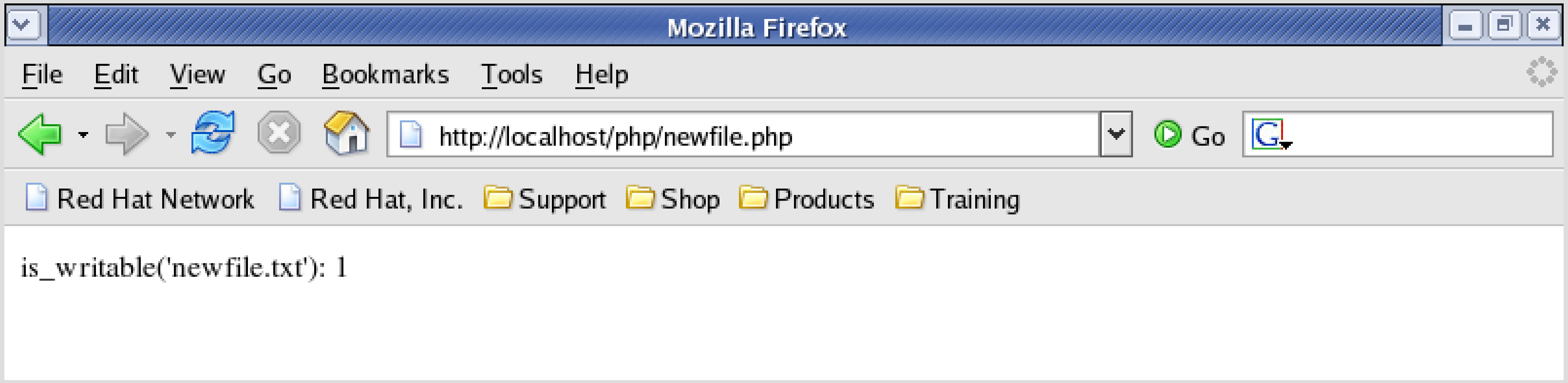
Executing *useradd phpuser*:

Reading /etc/passwd file:

```
root:x:0:0:root:/root:/bin/bash bin:x:1:1:bin:/bin:/sbin/nologin daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown halt:x:7:0:halt:/sbin:/sbin/halt
```

Part II: suPHP

- suPHP is then installed, and tests run again
- In this case, scripts are set to run as their owner, thus *ftpuser can modify file created by its own scripts*

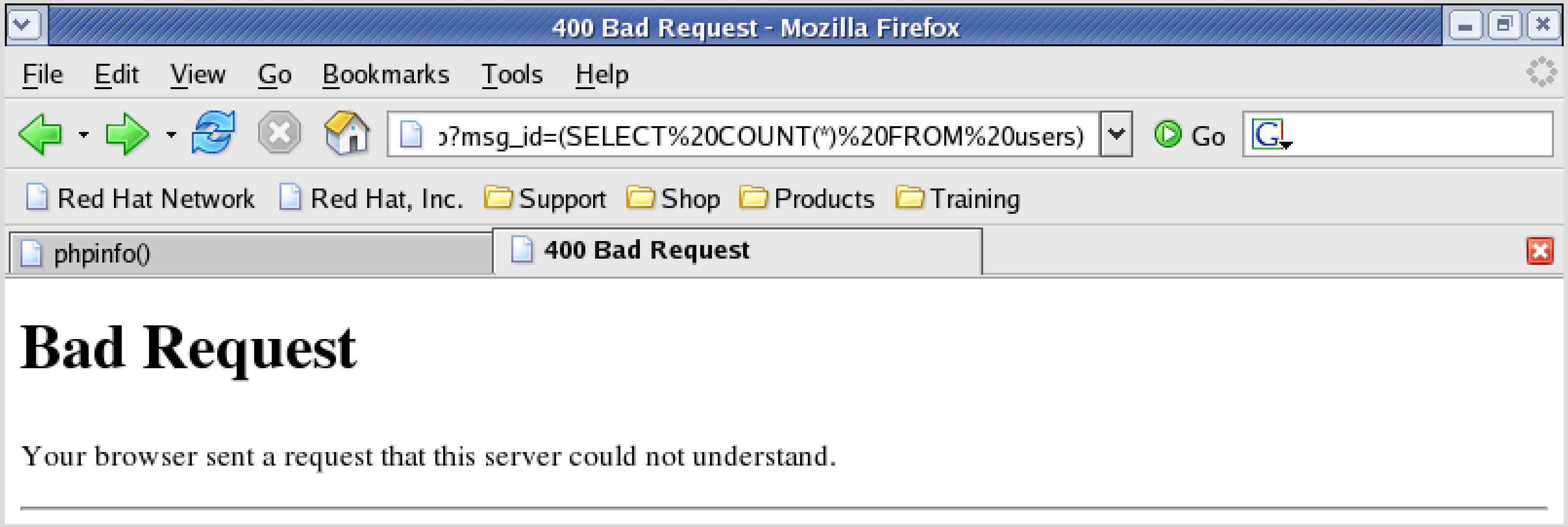


```
is_writable('newfile.txt'): 1
```

Part III: ModSecurity

- mod_security is then installed, and a few example rules are given
- A number of previous exploits are then attempted, and the results examined

```
SecRule REQUEST_FILENAME|ARGS “SELECT COUNT”
“deny,log,auditlog,status:400,msg:’SELECT COUNT query denied’
```



?msg_id=(SELECT%20COUNT(*)%20FROM%20users)

Bad Request

Your browser sent a request that this server could not understand.

Conclusions

- PHP run as an Apache module and as a CGI binary both have benefits and drawbacks
- Specific implementation depends upon the needs of the host, and of the webmasters using the host

Conclusions

- Companies and individuals should be familiar with the global platform and configuration of their chosen hosts
- Especially in shared hosting environments, users may not have the option of changing their configuration at will

Conclusions

- Hosts should be aware of the pros and cons of the hosting configuration(s) they offer, and be prepared to deal with updates
- For instance, PHP as an Apache module running in safe mode is very prevalent, but will soon be phased out – hosts must deal with this change

Questions?