# Security Test and Intro to Flask

Fall 2018 – Introduction to Full-Stack Development

Instructor: Matthew Fritter

matt@frit.me

# Stepping Back: Lab 5

- ► In Lab 5, we installed several security packages/modules and looked at some ways of minimizing attacks on your server:
  - ► Fail2Ban and Mod-Security
  - Changing the SSH port to minimize the number of drive-by bot attacks
  - ► Locking down the root account login

## Fail2Ban and Mod-Security

- The Fail2Ban and Mod-Security packages serve to automate certain security tasks, specifically blocking potential bad actors
  - ► Fail2Ban uses a regex system to review log files, and bans users who have an inordinate number of login/access attempts over a specified period of time
  - ► Fail2Ban is extremely configurable can be used to watch almost any port or service provided it produces a log file (SSH, HTTP, HTTPS, TCP, UDP, etc)
- Mod-Security is a more advanced module that detects suspicious behaviour in requests – strange headers, unidentified user agents, injection attempts
  - Mod-Security is very powerful, but prone to false positives. It requires a lot of testing and configuration

# Getting Rid of Root

- The root login is a big, known target attackers know it exists, and being able to access root provides full access to your server
- ► For this reason, it's common to disable the root login, or at least disable remote root login via SSH, and instead use a standard user account that has super user (sudo) privileges
- Recall an attacker trying to bruteforce SSH needs to know a username and a password. Why provide them with half of that information by allowing root?

# Changing your SSH Port

- ▶ We also changed our SSH port from the default port 22 to a less commonly used port like 762
- ► This won't stop a determined attacker port scanners can fairly easily identify ports that are open to network traffic, allowing identification of potential SSH ports other than 22
  - Can be mitigated somewhat by whitelisting SSH IPs, or using RSA keys for login
- ▶ Less of a serious security measure, and more about not filling up the log files with junk from the thousands of login attempts made by bots on a daily basis

#### Announcements

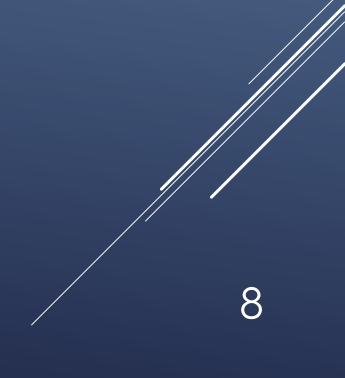
- ▶ Due to my illness last week and relatively small lab attendance last week, I'll be extending the due date for the Flask lab until Friday, Nov 2<sup>nd</sup>.
  - ► We'll cover the Flask lab today in-class to make sure everyone is on the same page

# Questions

- ► Are there any questions that people have run into regarding the Flask lab (lab 6)?
- ► Are there any questions about security topics before we begin the quiz?



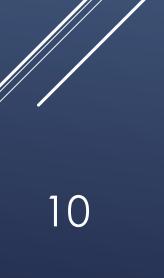
# Security Quiz



# Security Quiz

- ► The following IP has the website we'll be using for the quiz:
  - ► <a href="http://174.138.114.14">http://174.138.114.14</a>
- ► On this site there are 9 flags, each with a name and a short description
- ▶ Find all the tags and mark them as appropriate on your quiz sheet. Make sure all teammate's names are on the sheet please
- You may use the notes while taking the quiz. There will also be a surprise, closed-book bonus question at the end of the quiz period

# Introduction to Flask



#### ★ What is Flask?



- Flask is an extremely lightweight Python-based web development framework
- ► Licensed under the BSD license
- ► Allows use of Python for backend programming, and can be served through a variety of means by both Apache and NGINX
- ► Designed for applications where a lightweight footprint is important, and well suited to environments that come with Python installs by default (like CentOS 7)



# How Does it Compare to Laravel?

- ► Flask and Laravel have similarities in how they operate: you'll find routes, templates (like blades), request handling, sessions, etc
- ► However, consider Laravel like a massive set of Lego you have all the parts to build what you want, but you'll need to go searching for some of them, and you'll inevitably step on some
  - ▶ In comparison, Flask is far more freeform. Only the basic components are provided, the rest is up to you. This can be a good thing you won't be weighed down by components you'll never use
- ▶ In terms of backend language, Python is the clear winner in terms of readability and sensibleness, although it has it's own formatting quirks (whitespace is *important!*)



# ★ Installing Flask

- Flask can be easily installed using Pip, a package manager for Python
  - You'll need to install Pip see the Lab 6 instructions
- ► Once that's installed, you'll also need to install the Passenger Module. Passenger is an application that can serve as a standalone web server, but can also be used for integrating other services into Apache (like Python)
- You'll want to double check that Passenger installed correctly check your /etc/httpd/modules folder for a mod\_passenger.so file
  - Sometimes Passenger will decide to not install properly



# ★ Configuring Apache

- ► Once you've installed Flask and Passenger, you'll need to modify your Apache configuration files
- ➤ Specifically, create a new vhost (virtual host) for port 80 (HTTP), which will call the correct Passenger startup file and point to the correct folder for our Python application
- ▶ If Apache is not configured properly, Apache will not be able to interface with our Python backend double check the names of your PassengerStartupFile, and make sure you declare your PassengerAppType as wsgi
  - ► WSGI is Web Server Gateway Interface a standard for passing data from web server applications to Python



# ★ Building the Application

- Once configuration is complete, the majority of the hard work is done
- ► A minimal Flask install requires only two (2!) files in order to function properly: a passenger WSGI file that simply imports your project as an application; and an app.py file that actually contains your application logic
  - You can split application logic between files, but for a minimal system you can get away with doing everything in one – routing, logic, etc
  - Recall from Laravel though that this isn't a great idea for larger projects – your application file will very quickly become an organizational hurdle



## The Application File

- The first thing we do in our application file is import Flask itself Flask is, quite simply, a Python package, and is imported like any other Python package
- We create our application object and initialize it as an object of the Flask class
- ▶ Once we've done that, we can use our application object to define routes, and then define functions to call when those routes are called just like Laravel's routing, albeit less complication
- ▶ At the end, we call the run() function of the application object
- ► That's it once that's complete, we can visit our server IP and we should see our web page

## Templates

- ► For testing purposes, we can use Python to generate HTML at runtime and just directly return that HTML to the client
- ► However, this would produce a lot of overhead and messy code for actual webpages. Like Laravel, Flask includes a templating system to allow for a separation of web markup (HTML) from backend code (Python)
- ► Templates are stored in a /templates folder within your application folder. Unlike Laravel, these are actual HTML files
- ▶ The render\_template() function can be used to produce a string based on a given template file. This string can then be returned by the backend to the user

#### Tomorrow's Lab

- ► Tomorrow we'll be doing some more hands-on work with Flask
- ▶ Now that we've covered the basics of a Flask application, we'll be extending our application, allowing for dynamic web pages without having to generate all the HTML manually
  - ► Similar to Laravel data insertion into blade templates

# That's All Folks!

