

# Practical Secure Code Review

---

November 2019

# Introductions - Seth

Once known as the “King of Cola”, he ruled the mighty Carbonators during the pre-century, west-coast Soda Wars. His rule was cut short when HFCS was outlawed by California and the sugar prohibition era began. He led the first manned mission to Mars via a hot air balloon. Without hot air. Or the balloon. He speaks 14 languages, 12 of them based on the science fiction show “Star Trek”, 1 he made up with his imaginary friend Sharzazzle, and really - really poor Ol’ Timey English.

@sethlaw

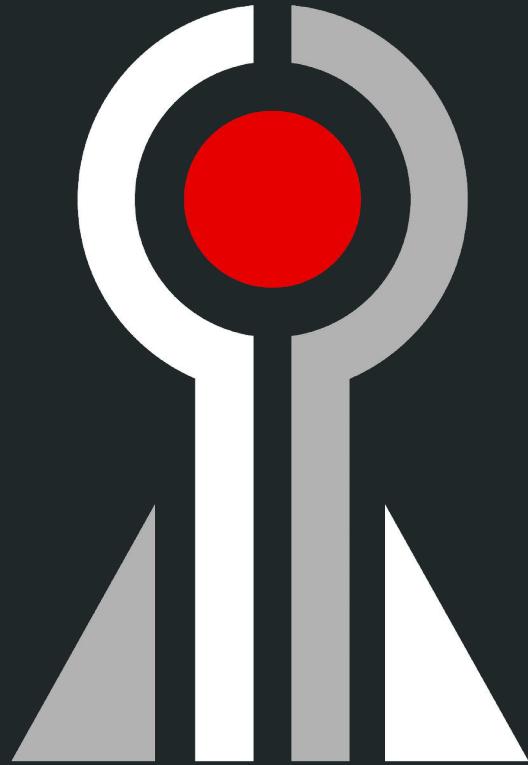


# Introductions - Justin

Originally a professional wood carver based in the Saharan desert, Justin started his career with computers at the ripe age of 24 months, when he was introduced to his first Enigma machine. After a extinguished career translating binary to septenary for extra-terrestrial satellite transmissions, he became a groupie for the three tenors. This naturally led to a vicious encounter with Bryan Adams, Cher, and Billy Joel that ended up with 2 of the 3 in jail in Singapore for selling chewing gum.

@Phant0mTrav3ler





Redpoint  
Security

# Overview

- Repeat after me:
  - Reviewing code is excellent!
  - Finding vulnerabilities is excellent!



# Philosophy - Part 1

- Increase the likelihood of success.  
Not “find every bug imaginable”.
- Successful is defined as:
  - Focusing on what matters by taking a risk based approach
  - Comprehensive
  - Timely
  - Easily consumed, well drafted reporting
  - Detailed notes



# Philosophy - Part 2

- Language Agnostic Principles
  - Learn how to be an expert in becoming an expert
- We're here to learn a repeatable systematic approach to finding bugs
  - But with understanding that you may later choose to tweak this approach to suit you

# What to Expect

- Our Experience and Stories
- Specific Information about Code Reviews
- Interactive Hands-On Exercises
- Practice, practice, practice

# Let's talk about approach



# Let's talk about approach

Several types of reviews:

- You decide how much time you spend on the review (ideal world)
- You have no time at all (real world)
- Consulting middleground where you have a set time and scoped (hopefully) well or at least semi “okay”

**NONE OF THESE TYPES MEAN “RUN A SCANNER AND ANALYZE THE RESULTS”**

# Let's talk about approach



# First, let's talk about what we're doing here

This course is intended to do the following, at a high level:

1. Show you where to look - Source to Sink
2. Show you what to look for, examples:
  - a. AuthZ/AuthN
  - b. Vulns specific to the type of app
  - c. General/OWASP AppSec vulns
3. Show where automation fits in the source code review process
4. Show some validation and remediation of the issues

# Also, let's talk about what we aren't doing

This course is not intended to:

1. Teach you the OWASP Top 10
2. Review web vulnerabilities
3. Cover techniques for finding vulnerabilities in running applications
4. Be a primer on building secure applications
5. Cover all the ways that vulnerabilities manifest in code.

# Tools/Lab Setup

- Virtual Machine Setup/Installation (Virtualbox || VMWare)
- Tool and environment familiarization



# OWASP

- How does the OWASP Top 10 relate?
- What resources does OWASP provide?
- How does this course differ from the OWASP Code Review Methodology?

# Secure Code Review Methodology

---

Overview

# Secure Code Review Methodology

- |   |   |
|---|---|
| 1. Application Overview & Risk Assessment | Checklists/Reviews  |
| 2. Information Gathering                  | <ul style="list-style-type: none"><li>• Authorization</li><li>• Authentication</li><li>• Auditing</li></ul> |
| 3. Checklist Creation                     | <ul style="list-style-type: none"><li>• Injection</li></ul>   |
| 4. Perform Reviews                        | <ul style="list-style-type: none"><li>• Cryptography</li><li>• Configuration</li></ul>                      |

# The Circle-K Framework

1. Open a file, take notes :-)
2. Get to know the application purpose
3. Map the application
4. Brainstorm risks to the application
- 5. Build list of review items**
6. Perform all reviews
7. Double back (3-6)



# General Principles

- Give yourself adequate time
- Work in small chunks
- Stay on task with current objective
- Don't make it personal
- Ask questions
- Framework/Code documentation is your friend
- Build the code
- Run the tests

# Focus on what's important



- So what happens when you run into 2 million lines of code?
- And only have two weeks to look at it?
- We will talk more about this...

# NOTE TAKING

# NOTE TAKING

- Typical Format
  - Commit #
  - Action Findings
  - Notes from Review
- Findings at the end
  - Description / Recommendation
  - Optional (or not) - Impact/Risk
  - Links to the code in question and/or screenshots
  - Reproduction steps if possible
- Review Notes
  - a. App Overview
  - b. Brainstorming/Risks
    - i. PAIR WITH OTHERS!!
  - c. Route Mapping
  - d. Checklist of items based on the threat model but also normal “things”

# Note Taking - Example

We assessed commit #74e64e1ccb617c83ba1db4cbbb24a33051e169f8

## Notes for you/your team

### Behavior

- What does it do? (business purpose)

Task Manager

- Who does it do this for? (internal / external customer base)

Internal Employees & External Customers

- What kind of information will it hold?

Tasks, Notes, Projects.. could be sensitive Date of Birth of users

## Brainstorming / Risks

- XSS - notes, projects and tasks
- Appears to use MD5 for passwords?
- TM employees using the product for managing their own products... ramifications
- noticed file uploads for profile pics - file access/handling
- What if sensitive pics are uploaded to the projects - CONFIRMED THAT PRC
- Image processing... RCE? Something else like traversal/LFI/RFI?

## Checklist of things to review based on Brainstorm

- Command Injection: system, call, popen, stdout, stderr, import os
- SQL Injection: raw, execute, select, where
- XSS: Autoescape, |safe, escapejs
  - Take a look at filenames and see if we render those unsafely anywhere
- File handling: File , django.core.files
- CSRF on the password change?
- IDOR on Projects/Notes/Tasks/Profile

# Review Types

---

# Review Types

- Full assessment of an entire application
- Assessment of new code developed since last review
- Pull request or small code changes review
- Scanner findings



# Full assessment of an entire application

1. Full Assessment 
  2. Only new code 
  3. Pull Request Review 
  4. Scanner Findings 
1. Everything is applicable
  2. Depends 🧐... seen this app before?
  3. Context specific bits (authz functions, logic processing, db queries, etc.)
  4. Meh

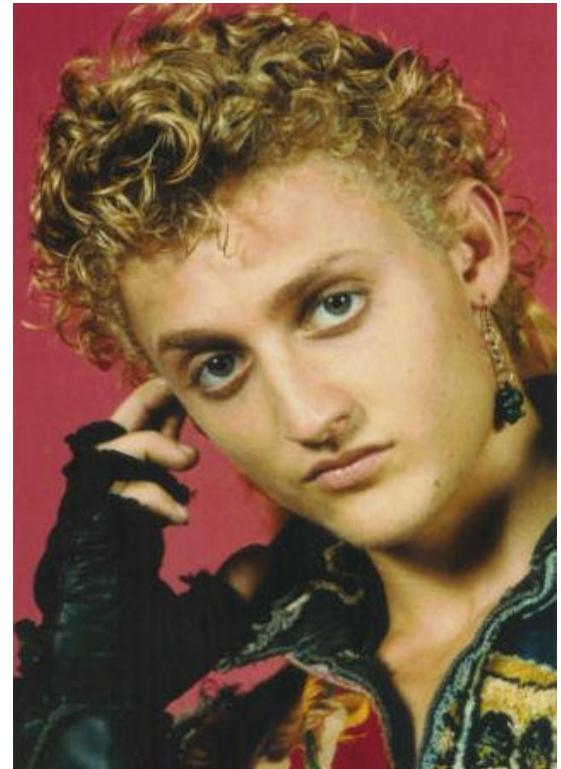
# Application Overview & Risk Assessment

---

# Application Overview & Risk Assessment

Build a portrait of the application

- Behavior Profile
- Technology Stack
- App Archeology
- Brainstorm Risks



# Behavior Profiling

- What does it do? (business purpose)
- Who does it do this for? (internal / external customer base)
- What kind of information will it hold?
- What are the different types of roles?
- What aspects concern your client/customer/staff the most?

# Technology Stack

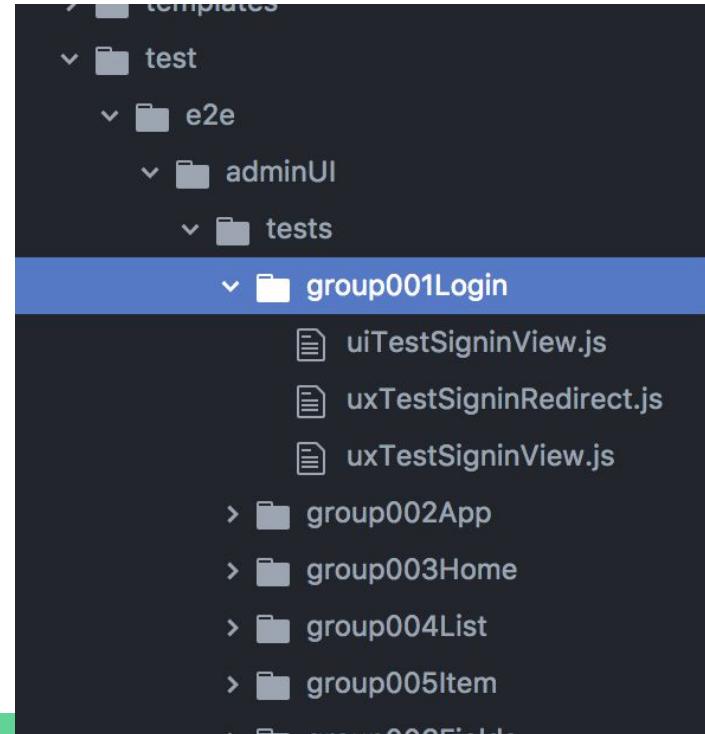
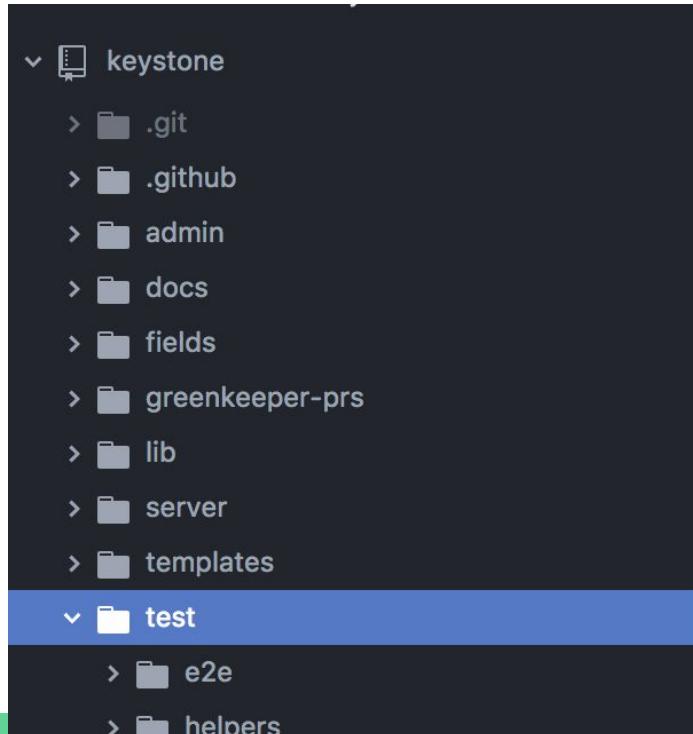
- Framework & Language - Rails/Ruby, Django/Python, mux/Golang
- 3rd party components, Examples:
  - Billing libraries (rubygem, npm, jar, etc.)
  - JavaScript widgets - (marketing tracking, sales chat widget)
  - Reliant upon other applications - such as receiving webhook events
- Datastore - Postgresql, MySQL, Memcache, Redis, Mongodb, etc.

# Application Archeology

- Look at the documentation. Examples:
  - Using SSO? SAML/OAuth?
  - What third-party elements does communicate with? (eg: external reporting, billing)
  - Custom authentication schemas - for instance - different auth for API versus web interface
- Git allows you to do a little spelunking
- Unit Tests
  - Learn about the data endpoints expect to receive
  - Learn about expected behavior
  - Often can find the way to form/craft a request

# Application Overview

<https://github.com/keystonejs/keystone>



# Application Overview & Risk Assessment

```
uxTestSigninRedirect.js
1 module.exports = {
2   before: function (browser) {
3     browser.adminUIApp = browser.page.adminUIApp();
4     browser.adminUISigninScreen = browser.page.adminUISignin();
5
6     browser.url(browser.adminUIApp.url + 'users');
7     browser.adminUIApp.waitForSigninScreen();
8     browser.assert.urlEquals(browser.adminUIApp.url + 'signin?from=/keystone/users');
9   },
10  after: function (browser) {
11    browser.
12      end();
13  },
14  'AdminUI should allow users to login and redirect to custom url': function (browser) {
15    browser.adminUISigninScreen.signin({wait: false});
16    browser.adminUIApp.waitForListScreen();
17    browser.assert.urlEquals(browser.adminUIApp.url + 'users');
18  },
19};
20
```

# Risk Assessment

- Use the initial data about the app to perform a “mini” threat model
- Think about application concerns based on the application purpose, technology stack, and archeology.
- If possible, get someone else’s opinion of relevant risks.



# Application Overview & Risk Assessment

## Exercise Napoleon

1. Start a secure code review of  
~/code/bhima
  - a. Tech Stack
  - b. Business Purpose
  - c. Application Archeology
  - d. Risk Assessment



# Napolean Exercise - Post-Mortem

At a minimum, these items were of note:

- Business Purpose
  - Basic Hospital Information Management Application
- Tech Stack
  - Node.js / Express
  - Angular
  - MySQL/Redis
- Documentation
  - <https://github.com/IMA-WorldHealth/bhima/wiki>
  - <https://docs.bhi.ma/en/>
- Risks
  - Patient Data
  - Employee Payroll Information



**Basic Hospital Information  
Management Application**

Bhima is a free, open source accounting and hospital information management system (HIMS) tailored for rural hospitals in Africa. We are an international team based in the Democratic Republic of the Congo.

# Napoleon Exercise - Post-Mortem

Other security important parts of the app:

- Express Configuration -
  - server/config/express.js
  - Uses default helmet.js
- User Authentication
  - server/controllers/auth.js
- Admin folder
  - Logic for admin logins -  
server/controllers/admin/users/index.js



# Information Gathering

---

# Information Gathering

1. Create Application Map
2. Identify Authorization Functions

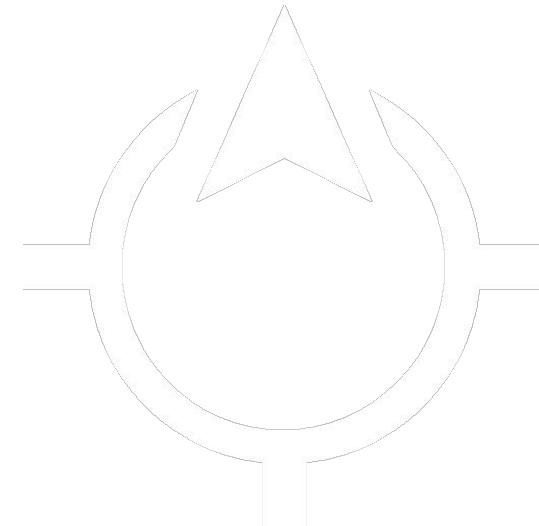


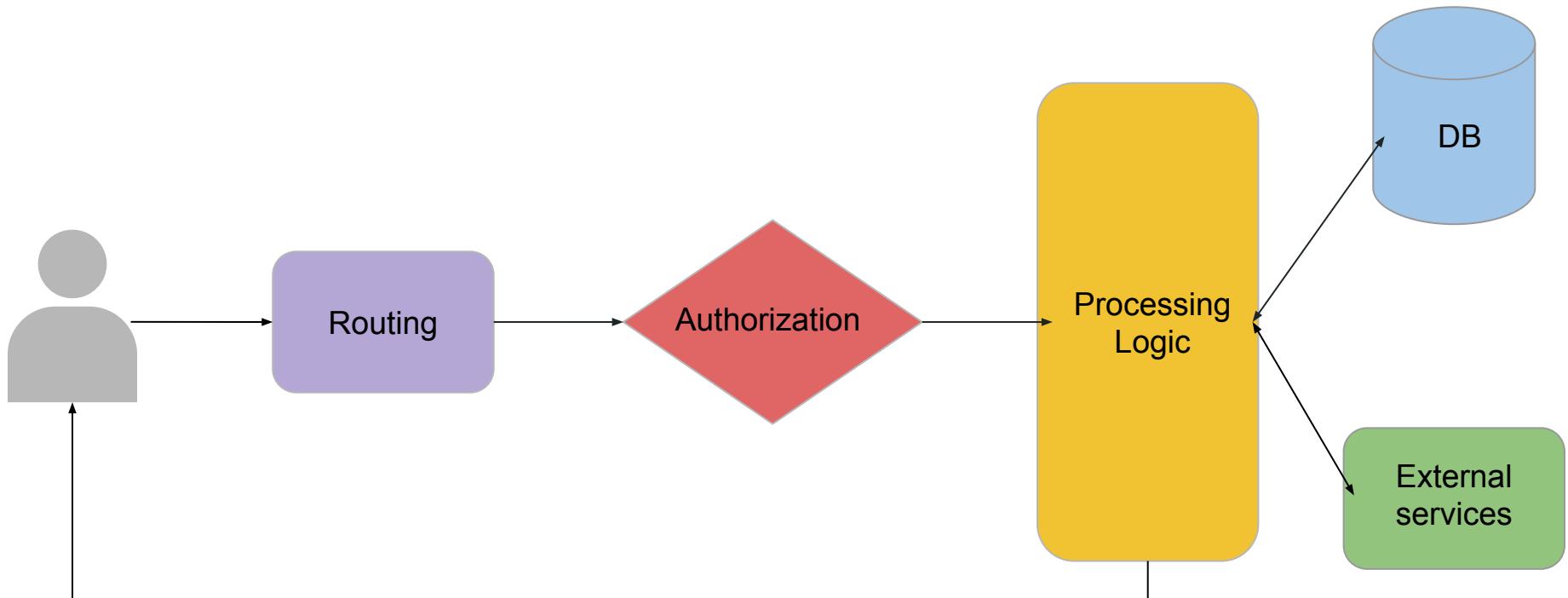
# Mapping



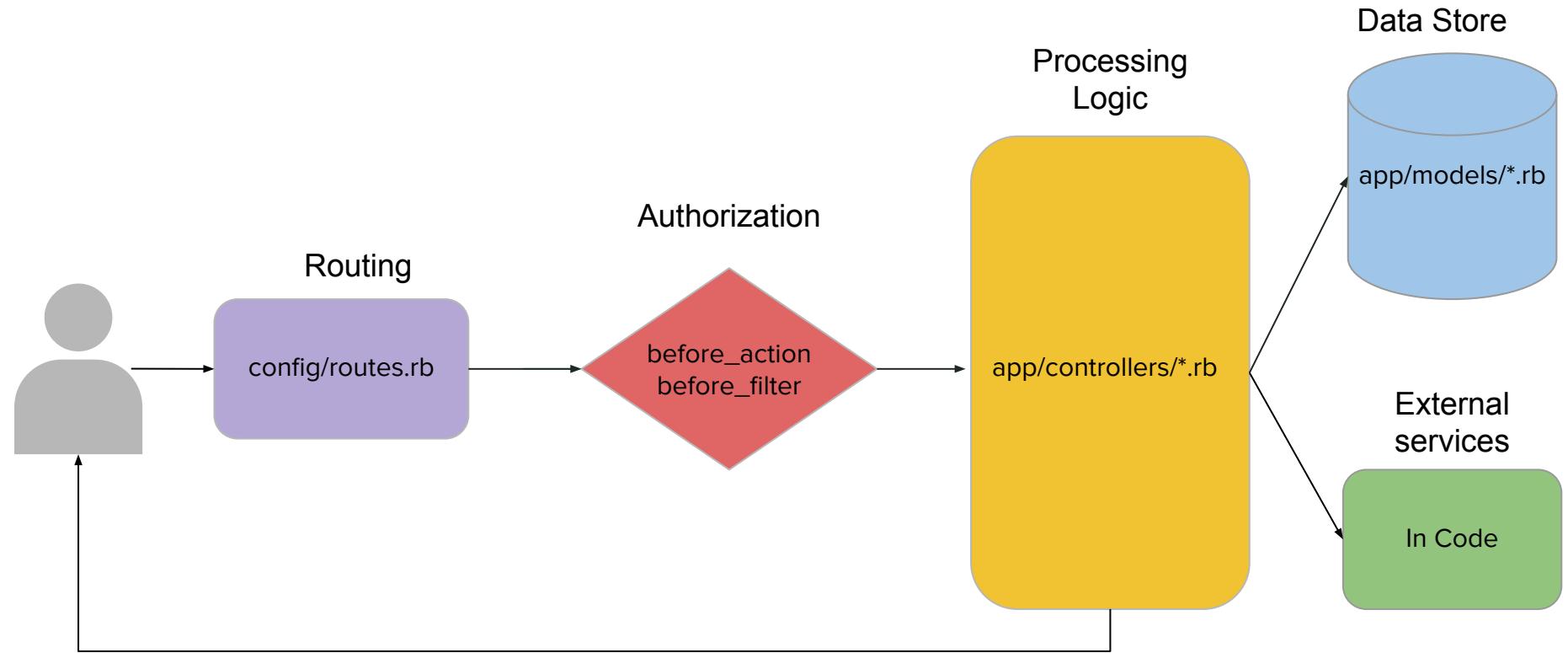
# Information Gathering - Application Map

- Identify endpoints, typical examples:
  - Rails = config/routes.rb - rake routes
  - Django= urls.py - manage show\_urls
  - Node.js = index.js
  - Java Spring = \*Controller.java
  - .Net Core = \*Controller.cs
- Endpoints typically have at least three qualities
  - Authorization Filter
  - Logic processing
  - Datastore access





Typical Application Flow



Rails Application Flow

# Rails - Map

- Run “rake routes” if you can!

```
cktricky@Kens-MacBook-Pro ~ code/railsboat master rake routes
Prefix Verb URI Pattern Controller#Action
  login GET /login(.:format) sessions#new
  signup GET /signup(.:format) users#new
  logout GET /logout(.:format) sessions#destroy
forgot_password GET /forgot_password(.:format) password_resets#forgot_password
                  POST /forgot_password(.:format) password_resets#send_forgot_password
password_resets GET /password_resets(.:format) password_resets#confirm_token
                  POST /password_resets(.:format) password_resets#reset_password
dashboard_doc GET /dashboard/doc(.:format) dashboard#doc
sessions GET /sessions(.:format) sessions#index
          POST /sessions(.:format) sessions#create
new_session GET /sessions/new(.:format) sessions#new
edit_session GET /sessions/:id/edit(.:format) sessions#edit
session GET /sessions/:id(.:format) sessions#show
         PATCH /sessions/:id(.:format) sessions#update
         PUT /sessions/:id(.:format) sessions#update
        DELETE /sessions/:id(.:format) sessions#destroy
user_account_settings GET /users/:user_id/account_settings(.:format) users#account_settings
user_retirement_index GET /users/:user_id/retirement(.:format) retirement#index
```

# Rails - Map

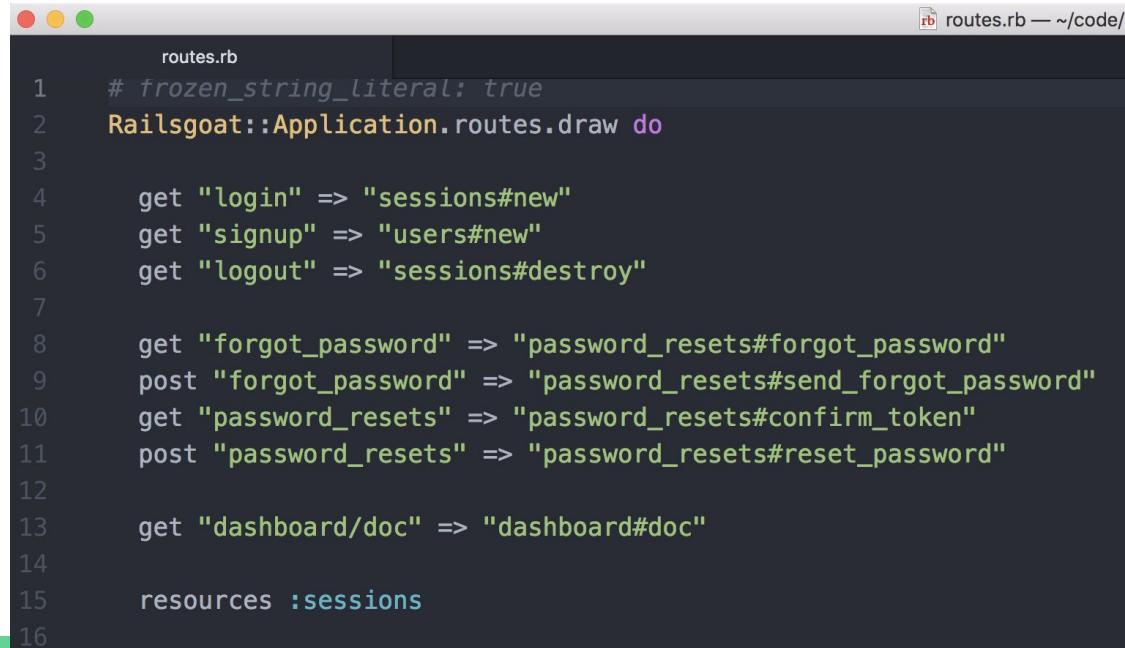
```
forgot_password GET /forgot_password(.:format)
```

```
password_resets#forgot_password
```

- Four parts
  - Path name (eg: forgot\_password\_path)
  - HTTP Verb (eg: GET)
  - HTTP Path (eg(s): forgot\_password, forgot\_password.json, forgot\_password.xml)
  - Controller and Action (eg: Controller = password\_resets, Action = forgot\_password)

# Rails - Map

- If you cannot run rake routes, you'll have to review the config/routes.rb file:



A screenshot of a terminal window titled "routes.rb" showing the contents of a Ruby file. The file contains code for defining routes in a Rails application. The code includes routes for user authentication (login, signup, logout), password resets (forgot\_password, password\_resets), and a dashboard. It also defines a resource named :sessions.

```
routes.rb
1 # frozen_string_literal: true
2 Railsgoat::Application.routes.draw do
3
4   get "login" => "sessions#new"
5   get "signup" => "users#new"
6   get "logout" => "sessions#destroy"
7
8   get "forgot_password" => "password_resets#forgot_password"
9   post "forgot_password" => "password_resets#send_forgot_password"
10  get "password_resets" => "password_resets#confirm_token"
11  post "password_resets" => "password_resets#reset_password"
12
13  get "dashboard/doc" => "dashboard#doc"
14
15  resources :sessions
16
```

# Rails - Routing Weirdness Alert

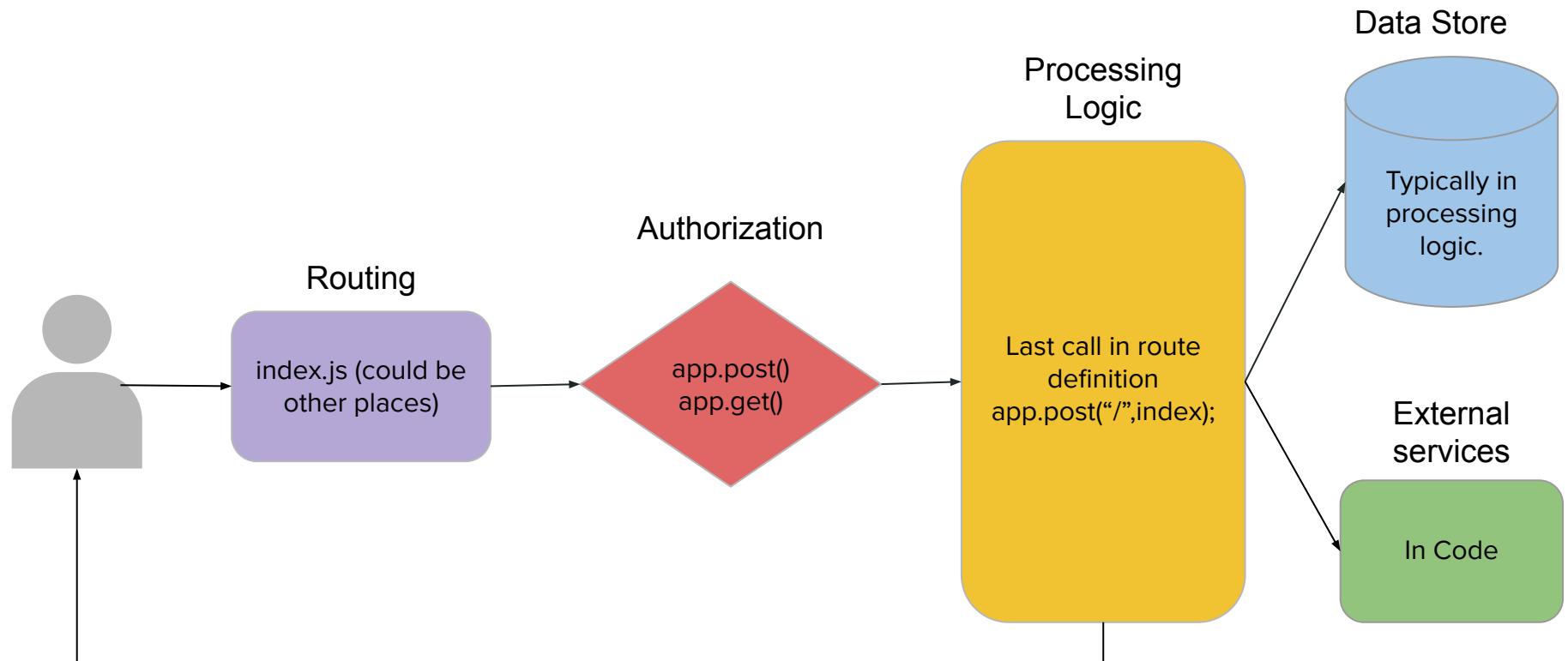
- What do PUT, PATCH, DELETE, and POST all have in common?
- That's right, they are all POST requests
- Rails uses the `_method` parameter to determine what kind of request it is

# Rails - Routing Weirdness Alert

---

```
POST /join HTTP/1.1
Host: github.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:66.0)
Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://github.com/join?source=header-home
Content-Type: application/x-www-form-urlencoded
Content-Length: 233
Connection: close
Cookie: logged_in=no
Upgrade-Insecure-Requests: 1

utf8=%E2%9C%93&authenticity_token=1YJBp5UZPnafW2qSy6GhEgg4ob05APDlMVUbmvLCCv
VqQS5JYA5cuVOcLWOS%2FDELy7z8J1AeH1TyxGCWojSxcw%3D%3D&user%5Blogin%5D=cktric
ky-example&user%5Bemail%5D=cktricky-example%40skeaa.com&user%5Bpassword%5D=d
olphin1&_method=patch
```



Node.js/Express Application Flow

# Node.js/Express - Map

- Formula has been basic, searching for:
  - app.get
  - app.post
  - app.delete
- Annotate which of these is actually using authorization functions and middleware
- Create a checklist from the map for tracing

# Node.js/Express - Map

Here is an example from Nodegoat, I downloaded it, and searched for app.get

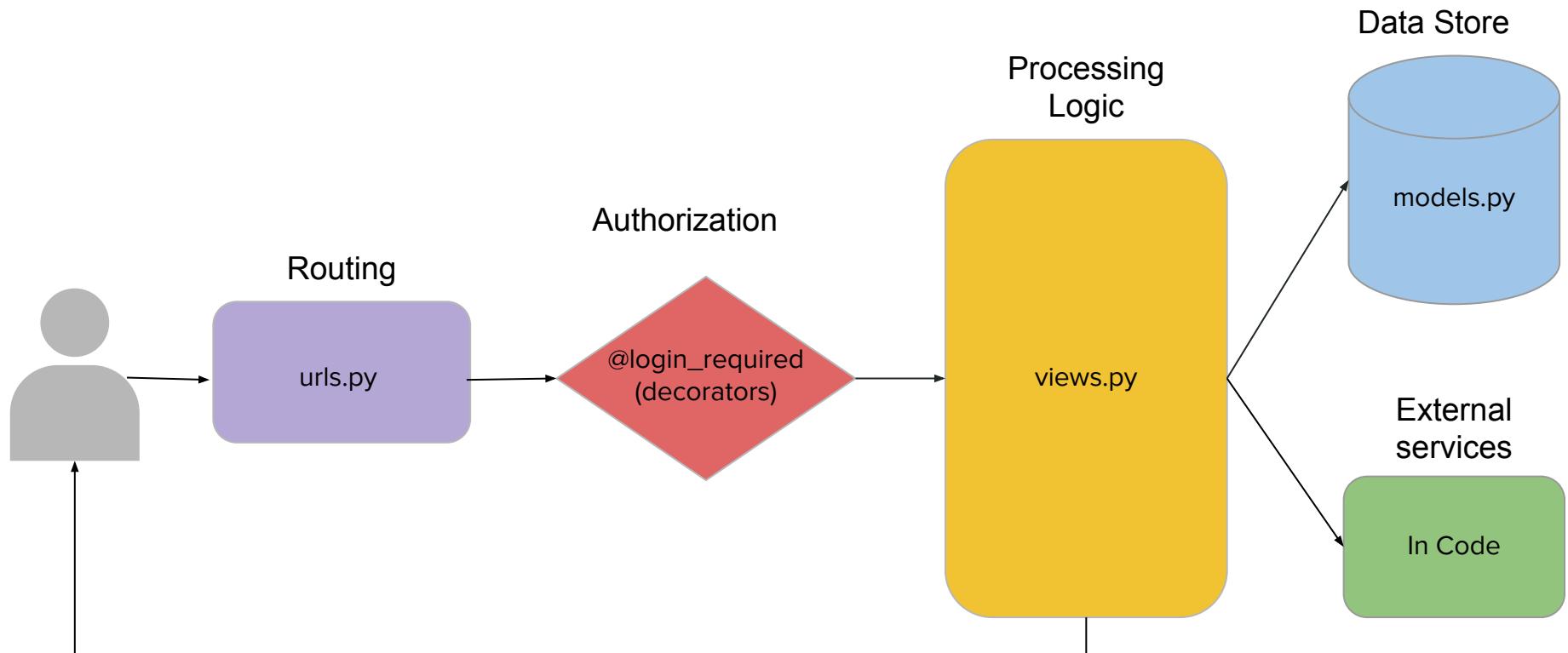
The screenshot shows a terminal window with a dark theme. On the left is a 'Project' sidebar listing files and folders for a project named 'NodeGoat'. On the right is a 'Project Find Results' pane with the title 'Project Find Results — ~/code/NodeGoat'. The search query 'app.get' is entered in the search bar. The results show 17 matches found in 3 files. The first file listed is 'app/routes/index.js' with 14 matches. The results are as follows:

```
28 app.get("/", sessionHandler.displayWelcomePage);
31 app.get("/login", sessionHandler.displayLoginPage);
35 app.get("/signup", sessionHandler.displaySignupPage);
39 app.get("/logout", sessionHandler.displayLogoutPage);
42 app.get("/dashboard", isLoggedIn, sessionHandler.displayWelcomePage);
45 app.get("/profile", isLoggedIn, profileHandler.displayProfile);
49 app.get("/contributions", isLoggedIn, contributionsHandler.displayContributions);
53 app.get("/benefits", isLoggedIn, benefitsHandler.displayBenefits);
56 app.get("/benefits", isLoggedIn, isAdmin, benefitsHandler.displayBenefits);
61 app.get("/allocations/:userId", isLoggedIn, allocationsHandler.displayAllocations);
64 app.get("/memos", isLoggedIn, memosHandler.displayMemos);
68 app.get("/learn", isLoggedIn, function(req, res, next) {
74 app.get("/tutorial" function(req, res, next) {
```

# Node.js/Express - Map

Take a close looksie

```
app.get("/dashboard", isLoggedIn, sessionHandler.displayWelcomePage);
```



Django Application Flow

# Django - Map

```
urls.py — ~/code/vtm
urls.py
1 # Vulnerable Task Manager
2
3 from django.conf.urls import include, url
4 from django.contrib import admin
5 from django.http import HttpResponseRedirect
6 from django.conf import settings
7 from django.views.defaults import page_not_found
8
9 from taskManager.views import index
10
11 urlpatterns = [
12     url(r'^$', index, name='index'),
13     url(r'^taskManager/', include(('taskManager.taskManager_urls','taskManager'), namespace="taskManager")),
14     url(r'^admin/', admin.site.urls ),
15 ]
16
```

# Django - Map

taskManager\_urls.py

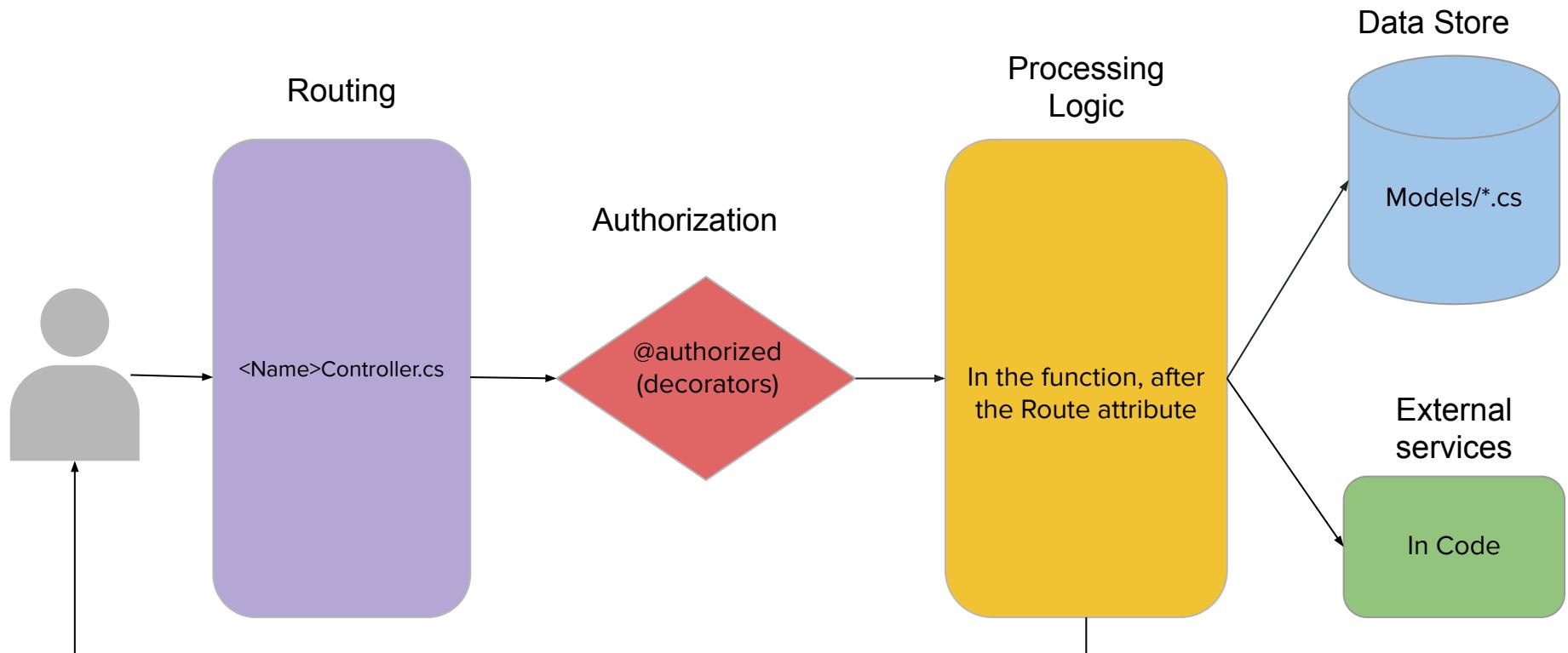
```
1 # Vulnerable Task Manager
2
3 from django.conf.urls import url
4
5 from taskManager import views
6
7 urlpatterns = [
8     url(r'^$', views.index, name='index'),
9
10    # User details
11    url(r'^view_all_users/$',
12        views.view_all_users, name='view_all_users'),
13
14    # File
15    url(r'^download/(?P<file_id>\d+)/$',
16        views.download, name='download'),
17    url(r'^(?P<project_id>\d+)/upload/$',
18        views.upload, name='upload'),
19    url(r'^downloadprofilepic/(?P<user_id>\d+)/$',
20        views.download_profile_pic, name='download_profile_pic'),
```

# Django - Map

```
# User details
url(r'^view_all_users/$',
    views.view_all_users, name='view_all_users'),
```

# Django - Map - ./manage.py show\_urls

```
/taskManager/<project_id>/task_edit/<task_id> taskManager.views.task_edit taskManager:task_edit
/taskManager/<project_id>/upload/ taskManager.views.upload taskManager:upload
/taskManager/change_password/ taskManager.views.change_password taskManager:change_password
/taskManager/dash.* taskManager.views.dashboard taskManager:dashboard
/taskManager/dashboard/ taskManager.views.dashboard taskManager:dashboard
/taskManager/download/<file_id>/ taskManager.views.download taskManager:download
/taskManager/downloadprofilepic/<user_id>/ taskManager.views.download_profile_pic taskManager:do
wnload_profile_pic
/taskManager/forgot_password/ taskManager.views.forgot_password taskManager:forgot_password
/taskManager/login/ taskManager.views.login taskManager:login
/taskManager/logout/ taskManager.views.logout_view taskManager:logout
/taskManager/manage_groups/ taskManager.views.manage_groups taskManager:manage_groups
/taskManager/manage_projects/ taskManager.views.manage_projects taskManager:manage_projects
/taskManager/ping/ taskManager.views.ping taskManager:ping
/taskManager/profile/ taskManager.views.profile taskManager:profile
/taskManager/profile/<user_id> taskManager.views.profile_by_id taskManager:profile_by_id
/taskManager/profile_view/<user_id> taskManager.views.profile_view taskManager:profile_view
/taskManager/project_create/ taskManager.views.project_create taskManager:project_create
/taskManager/project_list/ taskManager.views.project_list taskManager:project_list
/taskManager/register/ taskManager.views.register taskManager:register
/taskManager/reset_password/ taskManager.views.reset_password taskManager:reset_password
/taskManager/search/ taskManager.views.search taskManager:search
```



.Net Application Flow

# .NET Core - Map

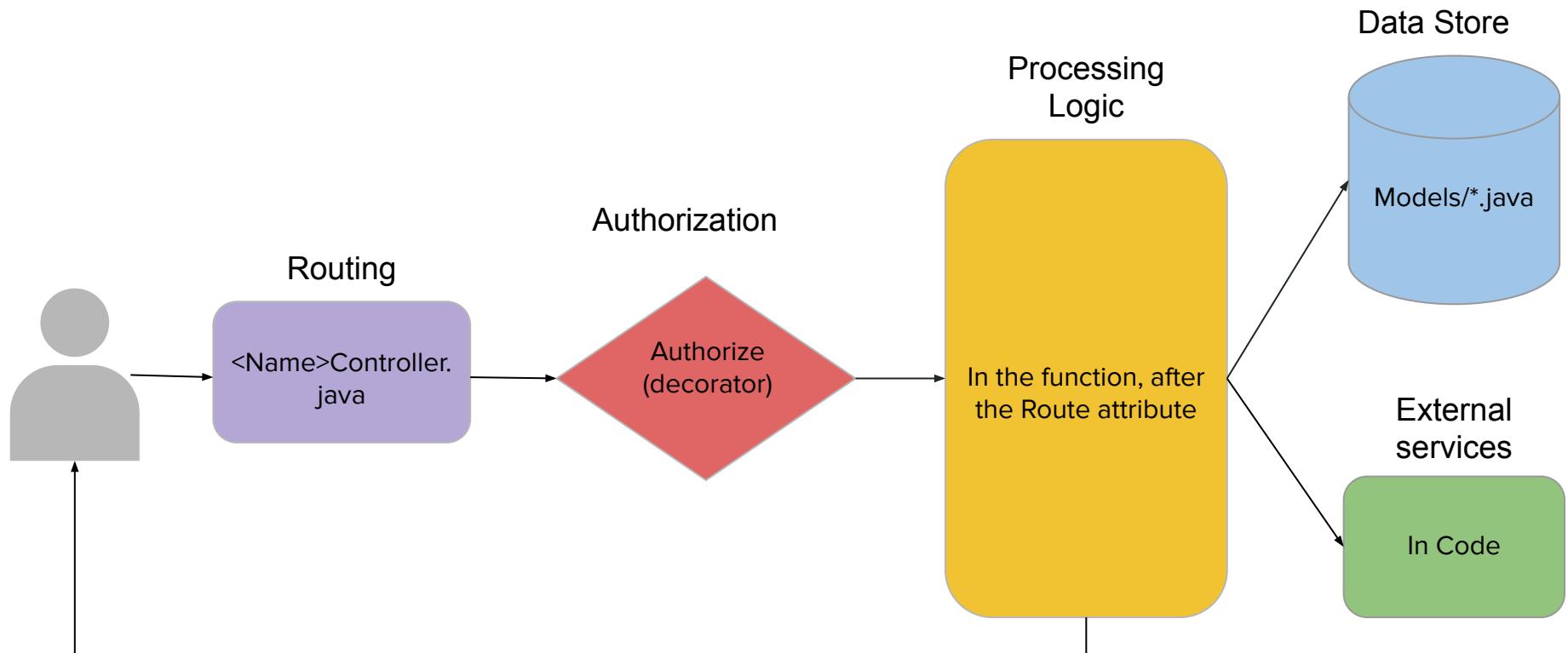
## AccountController.cs

```
1  using Microsoft.AspNetCore.Authentication;
2  using Microsoft.AspNetCore.Authentication.Cookies;
3  using Microsoft.AspNetCore.Authorization;
4  using Microsoft.AspNetCore.Mvc;
5  using Miniblog.Core.Models;
6  using System.Security.Claims;
7  using System.Threading.Tasks;
8  using Miniblog.Core.Services;
9
10 namespace Miniblog.Core.Controllers
11 {
12     [Authorize]
13     public class AccountController : Controller
14     {
```

# .NET Core - Map

```
[Route("/login")]
[AllowAnonymous]
[HttpGet]
public IActionResult Login(string returnUrl = null)
{
    ViewData["ReturnUrl"] = returnUrl;
    return View();
}

[Route("/login")]
[HttpPost, AllowAnonymous, ValidateAntiForgeryToken]
public async Task<ActionResult> LoginAsync(string returnUrl, LoginViewModel
```

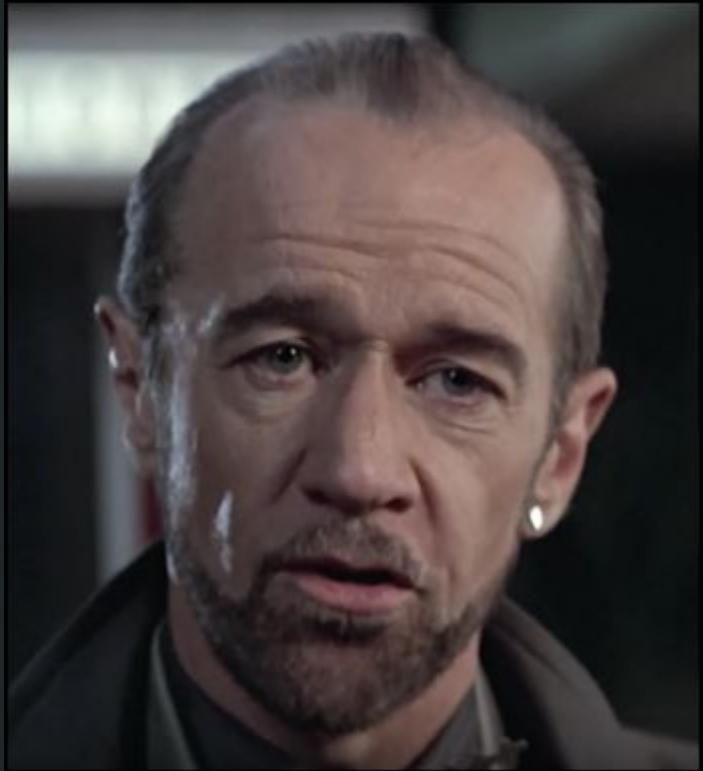


Java Application Flow

# Mapping

## Exercise Rufus

1. Start secure code review of  
~/code/skea\_node
  - a. Perform Info Gathering
    - i. (Biz Purpose/Tech Stack/App Arch/Risk Assessment)
  - b. Map Routes
2. Continue secure code review of ~/code/bhima
  - a. Identify routes



# Rufus Exercise - Post-Mortem

- Notes for skea\_node
  - A commit #
  - The endpoints as shown
  - Interesting items you may have found:
    - The fact the auth function doesn't do anything
    - The fact we have a Debug statement in our app and its printing env variables

```
▼ └ routes/index.js (1 match)
  5 router.get('/', function(req, res, next) {
▼ └ routes/users.js (3 matches)
  5 router.get('/', function(req, res, next) {
  10 router.get('/wtf', function(req, res, next) {
  15 router.get('/wtf!', function(req, res, next) {
```

```
▼ └ app.js (2 matches)
  28 app.get('/debug', function(req, res,next) {
```

# Information Gathering - Authorization Functions

- A later step is dedicated to creating authorization checks
- This is about getting to know the application better
- Identify pattern used for access control on the application endpoints

# Information Gathering - Authorization Functions

- Patterns & Anti-patterns
  - How is user identified? eg: Session, Token, Basic Auth
  - What is the purpose? Authenticated users, Role check, CSRF?
- What could go wrong? (add to the risks checklist)
  - User-supplied parameters (IDOR)
  - CSRF
  - Client-side cookies, JWTs etc. (just general wonkiness in persistence)

Story:

Redirection &  
Authorization Issue  
in .NET

# Redirection & Authorization Issue in .NET

## Normal

Results	Target	Positions	Payloads	Options				
Request	Payload		Status	Error	Timeout	Length	Comment	
0			404	<input type="checkbox"/>	<input type="checkbox"/>	1332		
1	example.aspx		302	<input type="checkbox"/>	<input type="checkbox"/>	249		

# Redirection & Authorization Issue in .NET

Definitely **NOT** Normal

The screenshot shows a tool interface titled "Intruder attack 17". The top navigation bar includes tabs for "Results" (which is selected), "Target", "Positions", "Payloads", and "Options". Below the navigation is a filter bar stating "Filter: Showing all items". The main area is a table with the following columns: Request, Payload, Status, Error, Timeout, Length, and Comment. There are two rows of data:

Request	Payload	Status	Error	Timeout	Length	Comment
0		404	<input type="checkbox"/>	<input type="checkbox"/>	1332	
1	example.aspx	301	<input type="checkbox"/>	<input type="checkbox"/>	301220	

# Redirection & Authorization Issue in .NET

## Redirect(String, Boolean)

Redirects a client to a new URL. Specifies the new URL and whether execution of the current page should terminate.

C#

 Copy

```
public void Redirect (string url, bool endResponse);
```

### Parameters

**url** String

The location of the target.

**endResponse** Boolean

Indicates whether execution of the current page should terminate.

# Authz Functions - Incorrectly Configured Example

```
application_controller.rb
1  # frozen_string_literal: true
2  class ApplicationController < ActionController::Base
3    before_action :authenticated, :has_info, :create_analytic, :mailer_option
4    helper_method :current_user, :is_admin?, :sanitize_font
5
6    # Our security guy keep talking about sea-surfing, cool story bro.
7    # Prevent CSRF attacks by raising an exception.
8    # For APIs, you may want to use :null_session instead.
9    protect_from_forgery #with: :exception
10
```

# Authorization Functions Checklist

## ❑ How are users identified

- Session
- Token
- Basic Authentication
- Something else?

# Authz functions checklist

## Identify its purpose

- Is it just checking that we're authenticated? - AuthN
- Is it looking for a specific role? - RBAC
- Is it doing some sort of HMAC verification for events?

# Authz functions checklist

## ❑ What could go wrong?

- Insecure timing comparisons
- Framework nuances
- IDOR
- Lack of rate limiting (see this often with basic auth or tokens)
- Think about if it was *\*not\** applied, what would happen
- **Get creative, you're trying to decide what the risk, if any, is here**

# Authz Functions

## Exercise Sigmund Freud

1. Start secure code review of  
~/code/skea\_rails
  - a. Perform Info Gathering
    - i. (Tech Stack/Biz Purpose/Risk Assessment)
  - b. Map Routes
  - c. Map AuthZ Functions
2. How are users authorized in  
~/code/skea\_rails?
  - a. Could take some time



# Sigmund Freud Exercise - Post-Mortem

- Yeah, you probably have questions re: “**authenticate\_user!**”
  - a. Which is why its part of the exercise :=D. THIS HAPPENS A LOT!

```
home_controller.rb
1 class HomeController < ApplicationController
2   before_action :authenticate_user!
3 
```

- How to decipher where this action is defined will be defined by:
  - a. Access to an interactive runtime env?
  - b. Experience with the framework/libs?
  - c. No previous experience with it, only access to source?

# Sigmund Freud Exercise - Post-Mortem

- Did you catch the CSRF? - “Framework Nuance”

```
post      "/articles", to: "articles#create"
get       "/articles/new", to: "articles#new", as: :new_article
get       "/articles/:id/:action", to: "articles#edit", as: :edit_article
get       "/articles/:id", to: "articles#show", as: :article
post      "/articles/:id/vote/:type", to: "articles#vote", as: :vote
```

# Sigmund Freud Exercise - Post-Mortem

- Anyone know what the request attack string would look like?



http://localhost:3000/articles/1/vote?type=like

```
get      "/articles/:id/:action", to: "articles#edit", as: :edit_article
```

# Sigmund Freud Exercise - Post-Mortem

- So what's the point with this whole thing?
  - You need to do your homework on framework nuances
  - Good example of why I often double-back and check one last time... (story time)



# Checklists & Reviews

---

# Authorization

---

# Authorization Review

- Analyze source for role enforcement, appropriate user boundaries, privileges required for access, and business-logic flaws
- Roles and associated enforcement routines must be identified during information gathering
- Pay attention to any endpoints that include sensitive data or functionality
  - Vertical authorization weaknesses - escalated privileges
    - Authenticated and unauthenticated access
  - Horizontal authorization weaknesses - access another user's data

# Authorization Review Vulnerabilities

- Broken Access Control - OWASP Top 10 A5:2017
  - Privilege Escalation
  - Missing Function Level Access Control
  - Insecure Direct Object Reference
- Sensitive Data Exposure - OWASP Top 10 A3:2017
- Mass Assignment
- Business Logic Flaws

# Authorization Review Checklist

- What are the different authorization functions?
  - Token/User/Role validation?
  - Is this handled by custom framework functionality?
  - Decorators vs. static source code
- Can non-privileged users view, add, or alter accounts?
- Is there functionality to add accounts with higher access levels than their own access?
- How is separation of duties handled?
- Are disabled accounts prevented from accessing content?
- Can password-protected pages be directly accessed without authentication?

# Mass-Assignment

- Goes by a few names
  - Insecure Binder Vulnerability
  - Insecure Object Mapping
  - Mass-Assignment
- Typically happens when a database entry is created or modified and we do so by throwing in \*all\* user-supplied parameters
- For example...

## Mass-Assigment - Node

```
var user = new User(req.body);
user.save();
```

# Mass-Assigment - Rails

## Rails 2/3

```
def create
  user = User.new(params[:user])
  if user.save
    session[:user_id] = user.id
    redirect_to home_dashboard_index_path
  else
    @user = user
    flash[:error] = user.errors.full_messages.to_sentence
    redirect_to :signup
  end
end
```

# Rails 4+

```
def create
  user = User.new(user_params)
  if user.save
    session[:user_id] = user.id
    redirect_to home_dashboard_index_path
```

```
def user_params
  params.require(:user).permit!
```

# Authorization Review Checklist

- Is it possible to bypass authorization restrictions by accessing content directly (e.g. can a non-privileged user access the administration pages of an application)?
- Can a user escalate privileges through cookie modification, altering form input values and HTTP headers, or by fuzzing URL-based parameters?
- Are there horizontal escalation/Insecure Direct Object References in the source code?
- Are authentication and authorization flows the first logic executed for each request?
- Are authorization checks granular (page and directory level) or per-application?

# Authorization Review Checklist

- Are access to sensitive pages and data denied by default?
- Are users forced to re-assert their credentials for requests that have critical side-effect (account changes, password reset, etc)?
- Do authorization checks have clearly defined roles?
- Can authorization be circumvented by parameter or cookie/token manipulation?
- Are CSRF protections in place and appropriate?

# Authorization

## Exercise Genghis Khan

1. Start secure code review of  
~/code/skea\_django
  - a. Info Gathering
    - i. (Tech Stack/Biz Purpose/App Arch/Risk Assessment)
  - b. Map Routes
  - c. AuthZ Functions
2. Build an authorization checklist
3. Test checklist items



# Genghis Khan Exercise - Post-Mortem

- First of all look into views.py

The screenshot shows a code editor with the following details:

- File Tabs:** settings.py (closed), views.py (open).
- File Path:** ... / RPS / Training / SKEA / skea\_django / intro / views.py / create\_todo
- Code Content (views.py):**

```
12
13 # Create your views here.
14 def index(request):
15     return HttpResponse("Welcome to Seth & Ken's Excellent Adventures")
16
17 @login_required
18 def todo(request, todo_id):
19     if request.method == 'GET':
20         try:
```
- Line 17:** The line containing `@login_required` is highlighted with a red rectangular box.

# Genghis Khan Exercise - Post-Mortem

- What about objects? Does it protect against IDOR? How?

# Authentication

---

# Authentication Review

- Authentication establishes user identity
- Examine the user identification process of the application.
- Available application resources include both unidentified and identified users
- Use enumeration of the application endpoints to trace the authentication flow and functions.
- Sensitive application and business functionality should redirect as appropriate to the authentication flow to properly identify a user
- Include an application functionality that identifies a user in this review

# Authentication Review

How does an application confirm identity?

# Authentication Review Vulnerabilities

- Broken Authentication - OWASP Top 10 A2:2017
- User Enumeration
- Session Management Issues
- Authentication Bypass
- Brute-Force Attacks

# User Enumeration - Login Page

The image displays two overlapping login forms, each showing an error message above the input fields. The top form has a light gray background and shows the message "Invalid Username. Please try again". The bottom form has a white background and shows the message "Login failed. Please try again". Both forms contain identical fields: "Username" (text input), "Password" (text input), and a red "Submit" button. Below the "Submit" button is a link "Forgot your password?".

Invalid Username. Please try again

**LOGIN TO TASK MANAGER**

Username

Password

Submit

[Forgot your password?](#)

Login failed. Please try again

**LOGIN TO TASK MANAGER**

Username

Password

Submit

[Forgot your password?](#)

# Broken Authentication - Node

```
20 exports.update = function(req, res) {
21   |
22   if (req.body.new_password == req.body.new_password_confirmation){
23     username = req.body.username
24     current_password = req.body.current_password
25     isMatch = true
26
27     db.User.find({where: {username: username}}).success(function (user){
28       hash = user ? user.password : ''
29       isMatch = db.User.validPassword(current_password, hash, function () { console.lo
30     });
31     if (isMatch) {
32       req.user.username = username
33       req.user.password = req.body.new_password
34       req.user.save()
35     } else {
36       console.log('Bad Password')
37     }
38   }
39
40   res.redirect('/account')
41 };
```

# Authentication Review Checklist

- What are the different authentication flows?
  - User Login
  - User Registration
  - Forgot Password
- How are users identified? What information do they have to provide?
  - Username, email, password, 2fa token, etc.
- How is authentication handled on each application endpoint?
  - Sensitive endpoints should require authentication
- Are there any hard-coded accounts?
- Does application allow for easily-guessed, default, or common passwords?

# Authentication Review Checklist

- How are usernames determined, what naming conventions?
  - Does registration allow for easy enumeration?
- Does the application allow for account enumeration through server responses or information disclosure?
  - login/registration/forgot password flows
- Are user credentials protected in the data store using modern password hashing algorithms?
- Are security policies configurable via environment variables and not hard-coded?
- What standard security frameworks are used?
  - Is there code specific to the application, especially when dealing with password storage?

# Authentication Review Checklist

- How are user management events such as authentication failures, password resets, password changes, account lockout and disabled accounts handled?
- Are suspicious event handling such as multiple failed login attempts, session replay and attempted access to restricted resources handled properly?
- Does the application implement strong password policies?
- Are authentication credentials being passed using technologies that could be cached (HTTP GET, lack of proper cache-control settings)?
- If applicable, are encryption mechanisms in place during authentication for secure communications (TLS, etc)?

# Authentication Review Checklist (Registration)

- Are limits on application access, such as geographical boundaries, enforced properly?
- Is there a manual approval process or is access granted automatically?
  - Can the automated process be abused or bypassed using scripting or brute-forcing?
- In cases where a validation email and link are required, how are the tokens generated?
- Can users elevate their initial access via mass assignment or business-logic bypasses?
- Are files and objects owned by a user removed or archived?

# Authentication Review Checklist (Sessions)

- Are encryption and hashing used properly?
- Do encryption protocols use strong algorithms and industry-standard key lengths?
- Are authentication tokens set with time limits?
- Are cookies security parameters set properly (e.g. Secure, HTTPOnly, path)?
- Are session IDs sent over a secure channel?
- Are session IDs invalidated before a new login is made?
- Are CSRF tokens set for all authentication requests?

# User Enumeration - Node

```
 31 User.findOne({
 32     username: req.body.username
 33 }, function(err, user) {
 34     if (err){
 35         res.send(err);
 36     }
 37     else{
 38         if (!user) {
 39             res.send("User not found");
 40         }
 41     else {
 42         if (user.password == req.body.password) {
 43             var token = jwt.sign(user, config.secret, {expi
 44
 45                 res.redirect('/homepage?token='+token);
 46
 47             } else {
 48                 res.send("Incorrect Password");
 49             }
 50         }
 51     }
 52 }
```



# Authentication

## Exercise Socrates

1. Continue review of  
~/code/skea\_django
  - a. Build/Review AuthN Checklist
  
1. Continue review of  
~/code/bhima
  - b. Build/Review AuthN Checklist



# So-crates Exercise - Post-Mortem

- So what exactly are we looking for?
  - a. ./manage.py show\_urls (django extensions have to be enabled).

```
/admin/jsi18n/    django.contrib.admin.sites.i18n_javascript      admin:jsi18n
/admin/login/     django.contrib.admin.sites.login            admin:login
/admin/logout/    django.contrib.admin.sites.logout          admin:logout
/admin/password_change/ django.contrib.admin.sites.password_change      admin:pa
ssword_change
/admin/password_change/done/   django.contrib.admin.sites.password_change_donea
dmin:password_change_done
/admin/r/<int:content_type_id>/<path:object_id>/           django.contrib.contentty
pes.views.shortcut      admin:view_on_site
/intro/  intro.views.index        index
/intro/<int:todo_id>/   intro.views.todo        todo
/intro/login/   django.contrib.auth.views.LoginView      login
/intro/logout/   django.contrib.auth.views.LogoutView     logout
/intro/password/  django.contrib.auth.views.PasswordChangeView password
/intro/password_change/ django.contrib.auth.views.PasswordChangeView password
```

# So-crates Exercise - Post-Mortem

- Where do they go?
  - a. *intro/urls.py* has no references to login????

urls.py

```
1 from django.urls import path
2 from django.contrib.auth import views as auth_views
3 from . import views
4
5 urlpatterns = [
6     path('', views.index, name='index'),
7     path('signup/', views.SignUp.as_view(), name='signup'),
8     path('password/', auth_views.PasswordChangeView.as_view(template_name='change_password.html'), name='password'),
9     path('<int:todo_id>/',views.todo, name='todo'),
10    path('todo/',views.create_todo, name='create todo'),
11    path('todos/',views.todos, name='todos'),
12    path('todos/completed/',views.todos_completed, name='completed todos')
13 ]
```

# So-crates Exercise - Post-Mortem

- Either does `skeate_django/urls.py` - but there is a reference to `django.contrib.auth.urls`

urls.py — intro	urls.py — skeate_django

```
1 from django.contrib import admin
2 from django.urls import include, path
3 from django.views.generic.base import TemplateView
4
5 urlpatterns = [
6     path('', TemplateView.as_view(template_name='home.html'), name='home'),
7     path('admin/', admin.site.urls),
8     path('intro/', include('intro.urls')),
9     path('intro/', include('django.contrib.auth.urls')),
10 ]
11
```

# So-crates Exercise - Post-Mortem

- Configuration documentation for django.contrib.auth leads us to *skeia\_django/settings.py*

```
103  
104 AUTH_USER_MODEL = 'intro.TodoUser'  
105  
106 LOGIN_REDIRECT_URL = 'home'  
107 LOGOUT_REDIRECT_URL = 'home'  
108 LOGIN_URL = '/intro/login/'  
109
```

# So-crates Exercise - Post-Mortem

- Searching for django.contrib.auth leads us to *intro/models.py*

login.html	models.py	views.py
------------	-----------	----------

```
1  from django.db import models
2  from django.contrib.auth.models import AbstractUser
3
4  # Create your models here.
5
6  class TodoUser(AbstractUser):
7
8      def __str__(self):
9          return self.email
10
11 class Todo(models.Model):
12     todo_text = models.TextField()
```

# So-crates Exercise - Post-Mortem

- If we go back to the django.contrib.auth documentation, it uses the templates in the registration directory.

login.html	forms.py	views.py
1   <!-- templates/registration/login.html --> 2   {% extends 'base.html' %} 3 4   {% block title %}Login{% endblock %} 5 6   {% block content %} 7   <div class="row"> 8     <div class="col-3"></div> 9     <div class="col-6"> 10       <h2>Login</h2> 11       <form method="post"> 12            {% csrf_token %} 13            {{ form.as_p }} 14            <button type="submit" class="btn btn-secondary">Login</button> 15		

# Auditing

---

# Auditing Review

- Validate that appropriate logging and exception handling are handled within application source
- One path in the trace of sensitive data from source to sink
- Logging functions and error messages are considered a data sink
- Logging should happen in any endpoint that performs a state-changing operation or has security implications
- This data is used for immediate analysis and future forensics needs.
- Check that sensitive data is appropriately handled (no credit card numbers, etc) and the correct details are logged
- Administrators must trust that logs may not be manipulated by unauthorized parties

# Auditing Review Vulnerabilities

- Sensitive Data Exposure - OWASP Top 10 A3:2017
- Insufficient Logging & Monitoring - OWASP Top 10 A10:2017
- Debug Messages
- Error Handling
- Information Leakage

# Auditing Review Checklist

- If an exception occurs, does the application fail securely?
- Do error messages reveal sensitive application or unnecessary execution details?
- Are Component, framework, and system errors displayed to end user?
- Does exception handling that occurs during security sensitive processes release resources safely and roll back any transactions?
- Are relevant user details and system actions logged?
- Is sensitive user input flagged, identified, protected, and not written to the logs?
  - Credit Card #s, Social Security Numbers, Passwords, PII, keys

# Auditing Review Checklist

- Are unexpected errors and inputs logged?
  - Multiple login attempts, invalid logins, unauthorized access attempts
- Are log details specific enough to reconstruct events for audit purposes?
- Are logging configuration settings configurable through settings or environment variables and not hard-coded into the source?
- Is User-controlled data validated and/or sanitized before logging to prevent log injection?

# Logging - Java

```
2  
3     import java.util.Date;  
4  
5     import org.apache.log4j.Level;  
6     import org.apache.log4j.Logger;  
7
```

```
20     public void createContactData(String[] sendBioData, Date dateCreate)  
21     {  
22         try{  
23             bioDAO.createContactData(sendBioData, dateCreate);  
24         }catch (Exception e) {  
25             logger.log(Level.ERROR, e.getMessage(), e);  
26         }  
27     }
```

# Auditing

## Exercise Abraham Lincoln

1. Continue review of  
~/code/skea\_django
  - a. Build/Review Auditing Checklist
  
2. Continue review of  
~/code/bhima
  - b. Build/Review Auditing Checklist



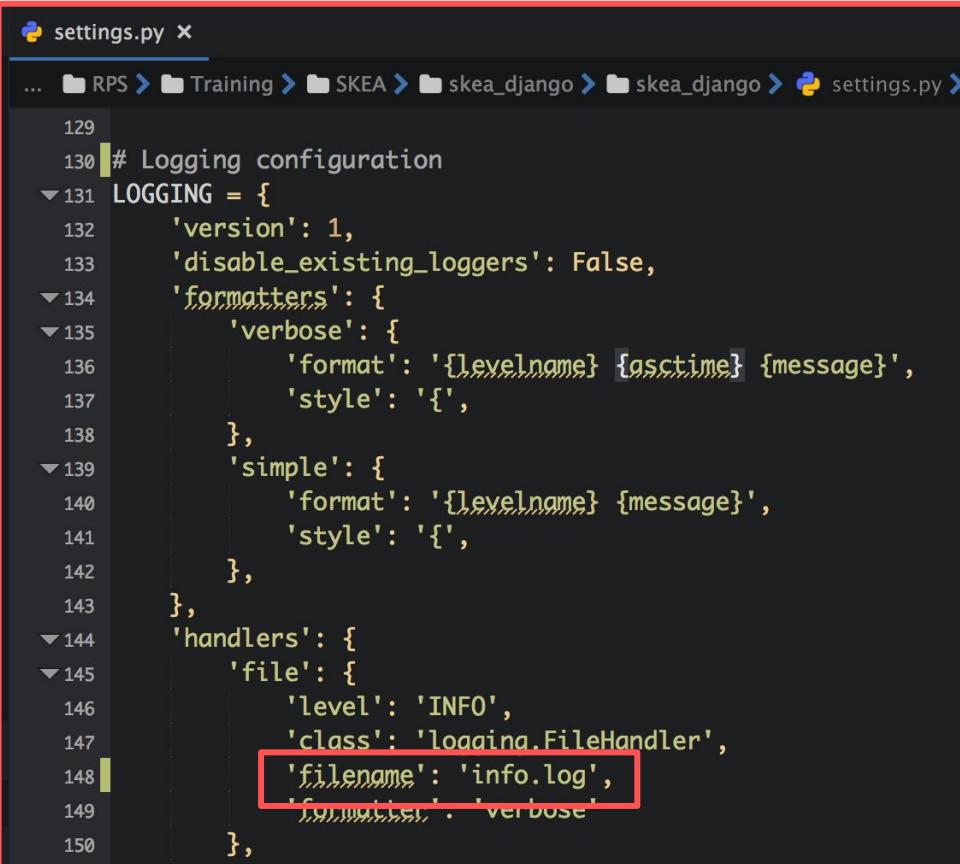
# Abraham Lincoln Exercise - Post-Mortem

- So where should we look? Sensitive functions.

```
47 @login_required
48 def create_todo(request):
49     if request.method == 'POST':
50         form = TodoForm(request.POST)
51         if form.is_valid():
52             t = Todo(todo_text=form.cleaned_data['todo_text'],
53                      todo_date = form.cleaned_data['todo_date'],
54                      completed = form.cleaned_data['completed'])
55             t.owner = request.user
56             t.save()
57             logger.info("Created todo %s by %s" % (todo_id,request.user.username))
58             return HttpResponseRedirect('/intro/todos/')
59
60     else:
61         form = TodoForm()
```

# Abraham Lincoln Exercise - Post-Mortem

- It's django, so everything is setup in settings.py.
- The intro application goes right along with this.



```
settings.py x
...
... RPS > Training > SKEA > skea_django > skea_django > settings.py >
129
130 # Logging configuration
131 LOGGING = {
132     'version': 1,
133     'disable_existing_loggers': False,
134     'formatters': {
135         'verbose': {
136             'format': '[levelname] {asctime} {message}',
137             'style': '{',
138         },
139         'simple': {
140             'format': '[levelname] {message}',
141             'style': '{',
142         },
143     },
144     'handlers': {
145         'file': {
146             'level': 'INFO',
147             'class': 'logging.FileHandler',
148             'filename': 'info.log',
149             'formatter': 'verbose'
150         },
151     }
}
```

# Injection

---

# Injection

- Causes:
  - Input Validation
  - Output Encoding
- Types
  - SQL Injection
  - HTML Injection (XSS)
  - LDAP, XML, Command ...

# Injection Vulnerabilities

- Injection - OWASP Top 10 A1:2017
- XML External Entities (XXE) - OWASP Top 10 A4:2017
- Cross-Site Scripting (XSS) - OWASP Top 10 A7:2017
- Redirects
- SSRF

# Input Validation

- Analyze code that handles user input for type, format, and content validation before being used or stored by the application.
- Compile list of data sources to work through.
- Start with the routes identified in the information gathering phase, but also include:
  - Configuration files
  - Environment variables
  - External services
  - Database calls to external and internal databases.
  - ...

# Input Validation - Checklist

- Is all the input validated without exception?
- Do the validation routines check for known good characters and cast to the proper data type (integer, date, etc.)?
- Is the user data validated on the client or server or both (security should not rely solely on client-side validations that may be bypassed)?
- If both client-side and server-side data validation is taking place, are these validations consistent and synchronized?
- Do string input validation use regular expressions?
- Do these regular expressions use blacklists or whitelists?
- What bypasses exist within the regular expressions?

# Input Validation - Checklist

- Does the application validate numeric input by type and reject unexpected input?
- How does the application evaluate and process input length?
- Is a strong separation enforced between data and commands (filtering out injection attacks)?
- Is there separation between data and client-side scripts?
- Is provided data checked for special characters before being passed to SQL, LDAP, XML, OS and third party services?
- For web applications, are often forgotten HTTP request components, including HTTP headers (e.g. referrer) validated?

# SQL Injection - Django

```
@csrf_exempt
def forgot_password(request):

    if request.method == 'POST':
        t_email = request.POST.get('email')

    try:
        result = User.objects.raw("SELECT * FROM auth_user where email = '%s'" % t_email)

        if len(list(result)) > 0:
            result_user = result[0]
            # Generate secure random 6 digit number
            res = ""
            nums = [x for x in os.urandom(6)]
            for i in range(6):
                res += str(nums[i])
            print(res)
    except:
        print("Error")
```

# SQL Injection - Node.js

```
10
11 exports.search = function(req,res) {
12     q = "";
13     users = [];
14     if (req.query.q) {
15         q = req.query.q;
16         db.User.findAll({attributes: ['id', 'username'], where: {username: { like: '%' +req.query.q+'%' } }}).success(function(users){
17             //console.log('Users:', users)
18             res.render("search.ejs", { q: q, username: req.user.username, users: users
19                 });
20         });
21     } else {
22         db.User.findAll().then(function(users){
23             //console.log('Users:', users)
24             res.render("search.ejs", { q: q, username: req.user.username, users: users
25                 });
26         });
27     }
28 }
```

# Output Encoding

- Analyze code that sends user data to client for context, type, and format before sending to uncontrolled data sinks.
- Start with a list of data sinks where data is being stored, sent, processed.
  - Source code libraries
  - 3rd-party services
  - Storage components (database in any of its possible forms)
  - File system interactions
  - Log files
- XSS, SSRF

# Output Encoding - Checklist

- Do databases interactions use parameterized queries?
- Do input validation functions properly encode or sanitize data for the output context?
- How do framework-provided database ORM functions used?
- Does the source code use potentially-dangerous ORM functions? (.raw, etc)
- What output encoding libraries are used?
- Are output encoding libraries up-to-date and patched?
- Is proper output encoding used for the context of each output location?
- Are output encoding routines dependent on regular expressions? Are there any weaknesses or blind-spots in these expressions?

# XSS - Node.js (ejs templates)

```
▼ 106    <tbody>
107
▼ 108    <% for(var i=0; i < listings.length; i++) { %>
109      <tr>
110        <td><%- listings[i].created %></td>
111        <td><%- listings[i].name %></td>
112        <td><%- listings[i].description %></td>
▼ 113        <td><%- listings[i].deadline %>
114
115        </td>
116        <td><a class="icon-ok" href="javascript:alert('Apply for Position')"></a><a c
117      </tr>
118    <% } %>
119  </tbody>
120</table>
121</div>
122</div>
```

# XSS - Django

```
<!-- user login dropdown start-->
<li class="dropdown">
    <a data-toggle="dropdown" class="dropdown-toggle" href="#">
        <!--User Identity -->
        <span class="username"><i class="fa fa-user fa-fw"></i> {{ user.username|safe }}</span>
        <b class="caret"></b>
    </a>
    <ul class="dropdown-menu extended logout">
        {% if user.id %}
```

# Injection

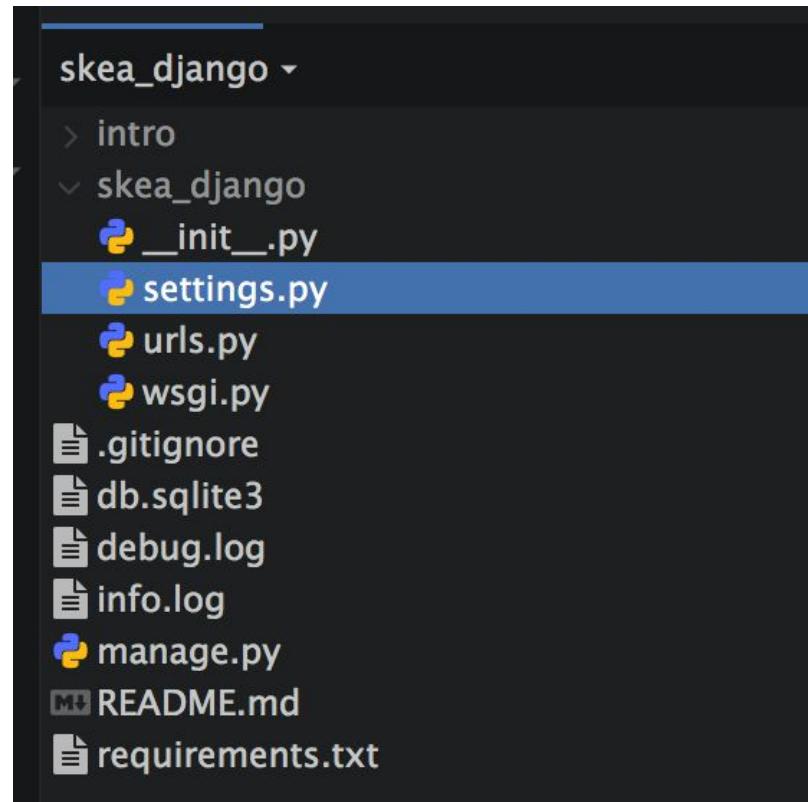
## Exercise Billy the Kid

1. `~/code/skea_django`
  - a. Build Injection Checklist
  - b. Trace all sources To sinks
  
1. `~/code/bhima`
  - b. Build Injection Checklist
  - c. Trace at least 3 different sources to sinks

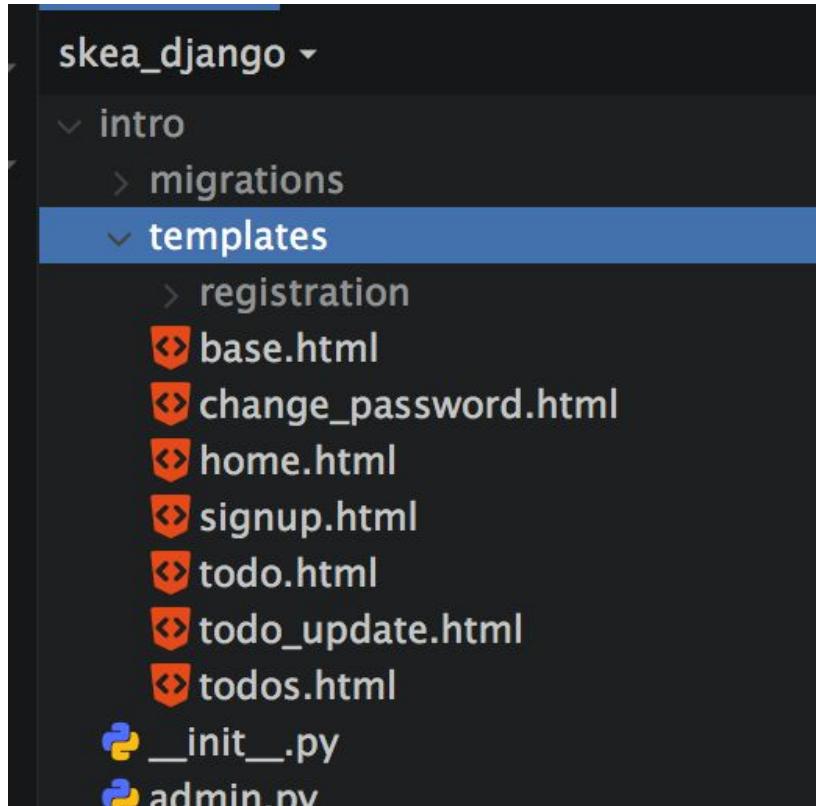


# Exercise Billy the Kid - Post-Mortem

- Some sources have already been identified. Are there others?
- Configuration files, additional services, even databases.
- Settings.py is somewhat trusted, but other people could manipulate data in the database before it gets to us.



# Exercise Billy the Kid - Post-Mortem



A screenshot of a file explorer window showing a Django project structure. The project is named 'ske\_a\_django'. It contains an 'intro' directory, a 'migrations' directory, and a 'templates' directory. The 'templates' directory is highlighted with a blue background. Inside 'templates', there is a 'registration' directory and several HTML files: 'base.html', 'change\_password.html', 'home.html', 'signup.html', 'todo.html', 'todo\_update.html', and 'todos.html'. At the bottom of the list are two Python files: '\_\_init\_\_.py' and 'admin.py'.

- ske\_a\_django ▾
  - intro
  - migrations
  - templates
    - registration
    - base.html
    - change\_password.html
    - home.html
    - signup.html
    - todo.html
    - todo\_update.html
    - todos.html
  - \_\_init\_\_.py
  - admin.py

- Now for sinks, we know that the database file is one, but what else?
- Anywhere data is /sent/ somewhere, file system, user, etc.

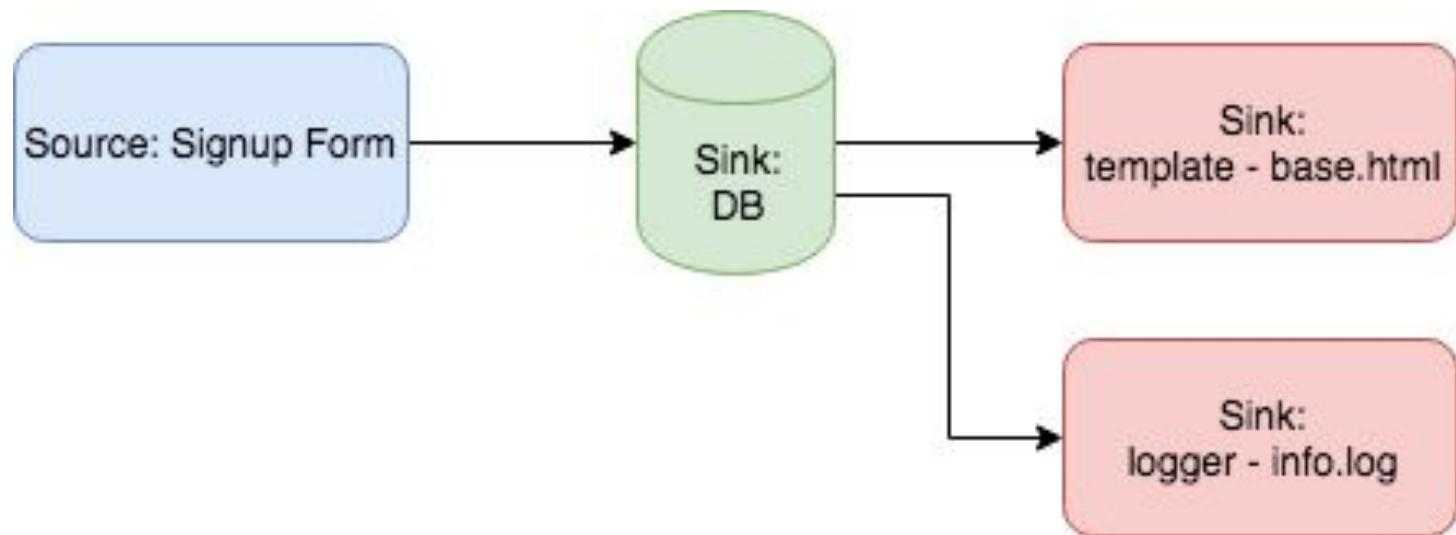
# Exercise Billy the Kid - Post-Mortem

- Search for user.username shows multiple references.

File	Line	Content
<skea_django>/intro/admin.py	12	list_display = ['email', 'username',]
<skea_django>/intro/forms.py	12	fields = ('username', 'first_name', 'last_name', 'email')
<skea_django>/intro/forms.py	18	fields = ('username', 'first_name', 'last_name', 'email')
<skea_django>/intro/views.py	28	logger.info("GET todo %s by %s" % (todo_id,request.user.username))
<skea_django>/intro/views.py	43	logger.info("Updated todo %s by %s" % (todo_id,request.user.username))
<skea_django>/intro/views.py	57	logger.info("Created todo %s by %s" % (todo_id,request.user.username))
<skea_django>/intro/views.py	68	logger.info("GET todos by %s" % (request.user.username))
<skea_django>/intro/views.py	74	logger.info("GET completed todos by %s" % (request.user.username))
<skea_django>/intro/migrations/0001_initial.py	27	('username', models.CharField(error_messages={'unique': 'A us'})
<skea_django>/intro/migrations/0001_initial.py	27	('username', models.CharField(error_messages={'unique': 'A us'})
<skea_django>/intro/migrations/0001_initial.py	27	('username', models.CharField(error_messages={'unique': 'A us'})
<skea_django>/intro/migrations/0001_initial.py	27	('username', models.CharField(error_messages={'unique': 'A us'})
<skea_django>/intro/templates/base.html	30	<span class="navbar-text">{{user.username}}

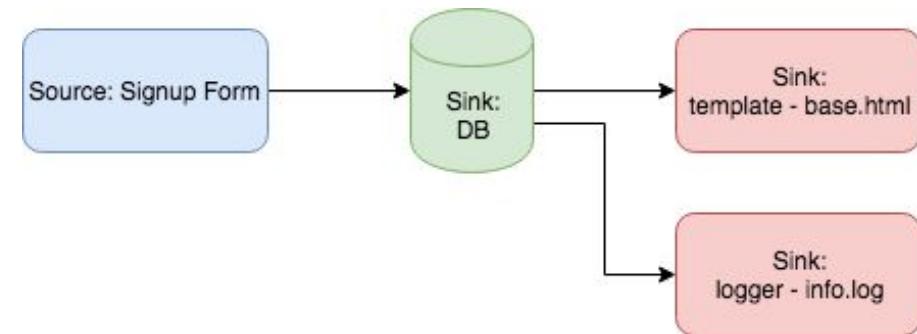
# Exercise Billy the Kid - Post-Mortem

- Diagram for user.username ends up looking like this.



# Exercise Billy the Kid - Post-Mortem

- This means we will be looking for the following injection flaws:
  - a. SQL Injection (database storage)
  - b. XSS (Stored/Reflected)
  - c. Log Forging/Injection



# Cryptographic Analysis

---

# Cryptographic Analysis

- Analyze code for encryption flaws, outdated protocols, custom-developed algorithms, weak encryption, and misuse
- Automated tools will uncover some of this, including
  - Use of older hashing algorithms (MD5, SHA-1, etc)
- Code and routes that handles sensitive information specifically should be reviewed
  - API Tokens
  - Credit Card Numbers
  - Social Security Numbers
  - Customer Data
- IDE Search for the following terms:
  - md5, sha1, base64, encrypt, decrypt, secure

# Cryptographic Analysis Vulnerabilities

- Lack of Encryption
- Improper Encryption
- Insecure Token Generation/Randomness

# Cryptographic Analysis - Checklist

- What are the standard encryption libraries are used for?
  - Hashing functions - password hashing, cryptographic signing, etc
  - Encryption functions - data storage, communications
- Do the strength of implemented ciphers meet industry standards?
  - Less than 256-bit encryption
  - MD5/SHA1 for password hashing
  - Any RC4 stream ciphers
  - Certificates with less than 1024-bit length keys
  - All SSL protocol versions
- Are cryptographic private keys, passwords, and secrets properly protected?

# Cryptographic Analysis

## Exercise Beeth-oven

1. `~/code/skea_django`
  - a. Build Crypto Checklist
  - b. How are passwords stored/tokens generated
2. `~/code/bhima`
  - a. Build Crypto Checklist
  - b. What hashing/crypto is being used?



# Exercise Beethoven - Post-Mortem

- Sessions - Only indication we have of activity is settings.py

```
32
33 INSTALLED_APPS = [
34     'intro.apps.IntroConfig',
35     'django.contrib.admin',
36     'django.contrib.auth',
37     'django.contrib.contenttypes',
38     'django.contrib.sessions', [Red box]
39     'django.contrib.messages',
40     'django.contrib.staticfiles',
41     'django_extensions',
42 ]
43
44 MIDDLEWARE = [
45     'django.middleware.security.SecurityMiddleware',
46     'django.contrib.sessions.middleware.SessionMiddleware', [Red box]
47     'django.middleware.common.CommonMiddleware',
48     'django.middleware.csrf.CsrfViewMiddleware',
49     'django.middleware.clickjacking.XFrameOptionsMiddleware'
```

# Exercise Beethoven - Post-Mortem

- Sessions - Django documentation helps us out a little.

## Configuring the session engine

By default, Django stores sessions in your database (using the model

**`django.contrib.sessions.models.Session`**). Though this is convenient, in some setups it's faster to store session data elsewhere, so Django can be configured to store session data on your filesystem or in your cache.

# Exercise Beethoven - Post-Mortem

- Sessions - but also gives us a warning...



## Warning

If the `SECRET_KEY` is not kept secret and you are using the `PickleSerializer`, this can lead to arbitrary remote code execution.

An attacker in possession of the `SECRET_KEY` can not only generate falsified session data, which your site will trust, but also remotely execute arbitrary code, as the data is serialized using pickle.

If you use cookie-based sessions, pay extra care that your secret key is always kept completely secret, for any system which might be remotely accessible.

# Exercise Beethoven - Post-Mortem

- Alright, so we should check the randomness of the SECRET\_KEY

21

```
22 # SECURITY WARNING: keep the secret key used in production secret!
23 SECRET_KEY = '*@ty00=62-recu@lsczr-00(%y97c(l11gnv9pu7o^)h%#e17d'
```

24

- Looks random and secure, right? What are the concerns you have with this?  
Bueller?

# Exercise Beethoven - Post-Mortem

- Possible issues (outside of encryption routines):
  - a. Hardcoded values stored in source
  - b. Same key used for dev/staging/production

# Exercise Beethoven - Post-Mortem

- Password Storage? Nothing in settings.py?

## How Django stores passwords

Django provides a flexible password storage system and uses PBKDF2 by default.

The **password** attribute of a **User** object is a string in this format:

```
<algorithm>$<iterations>$<salt>$<hash>
```

# Exercise Beethoven - Post-Mortem

- PASSWORD\_HASHERS in settings.py

```
[  
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',  
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',  
    'django.contrib.auth.hashers.Argon2PasswordHasher',  
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',  
    'django.contrib.auth.hashers.BCryptPasswordHasher',  
    'django.contrib.auth.hashers.SHA1PasswordHasher',  
    'django.contrib.auth.hashers.MD5PasswordHasher',  
    'django.contrib.auth.hashers.UnsaltedSHA1PasswordHasher',  
    'django.contrib.auth.hashers.UnsaltedMD5PasswordHasher',  
    'django.contrib.auth.hashers.CryptPasswordHasher',  
]
```

# Exercise Beethoven - Post-Mortem

- Did you run the app?

```
sqlite> select * from intro_todouser;
1|pbkdf2_sha256$120000$GJ1WIimqmSAI$6+WnzERREqiR44/FFLy8JjaEx160ysYFJW60MpdGizo=
|2018-10-05 21:14:37.054197|1|admin||admin@test.com|1|1|2018-10-05 20:20:38.818
426
2|pbkdf2_sha256$120000$a2sGvDg0aIXT$Qa1LbL4hWoLywacEqdEatLsH2Vcv0NlKopjq14oTC/E=
|2018-10-08 02:05:50.669012|0|test|First|Last|test@test.com|0|1|2018-10-05 20:21
· 00 520448
```

# Configuration Review

---

# Configuration Review

- Analyze any configurations included for security flaws
  - Includes language, framework, and server configurations
- Highly specific to the targeted language/framework
- Consult the server/framework documentation for guides on security flags and settings.
- Examples:
  - Administrative Functionality enabled through configuration files
  - CSRF settings
  - Cookie parameters

# Configuration Review Vulnerabilities

- Security Misconfiguration - OWASP Top 10 A6:2017
  - Insecure defaults
  - Incomplete configurations
  - Open cloud storage
- Using Components with Known Vulnerabilities - OWASP Top 10 A9:2017
  - Any dependencies, libraries, services

# Configuration Review - Checklist

- Are any endpoints enabled through configurations properly protected with authentication and authorization?
- Are security protections implemented in framework properly configured?
- Does the target language and framework version have any known security issues?
- Are configuration-controlled security headers implemented according to recommended best practices?

# Configuration Analysis

## Exercise Joan of Arc

1. `~/code/skea_node` and `bhima`
  - a. Build Config Checklist
  - b. Run/review npm audit and `nodejsscan`
2. `~/code/skea_rails` and `railsgoat`
  - a. Build Config Checklist
  - b. Run/review `brakeman`



# Security Misconfiguration - Java Spring

```
# Database Configuration
spring.datasource.url=jdbc:h2:mem:AZ;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
hibernate.hbm2ddl.import_files_sql_extractor=org.hibernate.tool.hbm2ddl.MultipleLinesSqlCommandExtractor
hibernate.hbm2ddl.auto=create

# H2 Options
security.basic.enabled=true
security.basic.authorize-mode=none
spring.h2.console.enabled=true
spring.h2.console.settings.web-allow-others=true
spring.h2.console.path=/console
```

# Additional Resources

---

# Additional Resources

- [OWASP Code Review Guide](#) - Includes some language-specific best practices for Java, .Net, C, C++.
- [Simplicable Secure Code Review Checklist](#)
- [Infosec Institute - Secure Code Review: A Practical Approach](#)
- [OWASP Input Validation Cheatsheet](#)
- [OWASP XSS Prevention Cheatsheet](#)
- [Recommended headers for internal and external Web User Interfaces](#)
- [Wikipedia - Principle of Layered Security](#)
- [Wikipedia - Principle of Least Privilege](#)
- [OWASP ASVS \(Application Security Verification Standard\)](#)

# Reporting and Retesting

---

# Reporting and Retesting

- This phase of the methodology is critical.
- Document findings in a manner that can be understood by a developer.
- Review the source after remediation using the same techniques (rinse & repeat).
- **THIS IS THE WHOLE POINT.**

# Report Writing (or ticket creation)

- Detailed findings include:
  - Description
  - Applicable File, Function, Variables, Line Numbers
  - Risk Severity
  - Likelihood of Exploitation
  - Ease of Exploitation
  - Remediation Approach
  - References

# Walkthrough: Vulnerable Task Manager

---

# Open Source Projects

---

**BE EXCELLENT TO EACH OTHER AND  
PARTY ON DUDES**

A photograph from the movie "Bill & Ted's Excellent Adventure". Bill and Ted, played by Alex Winter and Keanu Reeves, are standing on a stage under bright blue lights. They are both wearing white t-shirts and have their right arms raised in a triumphant fist pump. Their mouths are open as if they are shouting or singing. The background is dark, suggesting a concert or stage setting.

made on imgur

Seth - [seth@redpointsecurity.com](mailto:seth@redpointsecurity.com)

Justin - [jl@redpointsecurity.com](mailto:jl@redpointsecurity.com)