

PRACTICAL SECURE CODE REVIEW

(SETH & KEN'S EXCELLENT ADVENTURE IN SECURE CODE REVIEW)

2024

RESOURCES YOU NEED

Python3 + AWS Bedrock Key (or Ollama & Llama 3)

Repositories:

- <https://github.com/IMA-WorldHealth/bhima> - Node App
- https://github.com/railsbridge/bridge_troll - RAILS App
- <https://github.com/absoluteappsec/handouts> - templates
- https://github.com/absoluteappsec/skea_django
- https://github.com/absoluteappsec/skea_node
- <https://github.com/sethlaw/vtm> - Vulnerable Task Manager

INTRODUCTIONS - SETH

Founder of **Redpoint Security**

Based in Salt Lake City, Utah

Consultant, Programmer,
Trainer, Speaker and of
course... a lot of code review.

@sethlaw



INTRODUCTIONS - KEN

CTO of **DryRun Security**

Lives in Northern Virginia

- GitHub
- CTO @ services co.
- LivingSocial
- Fishnet Security
- Pentagon

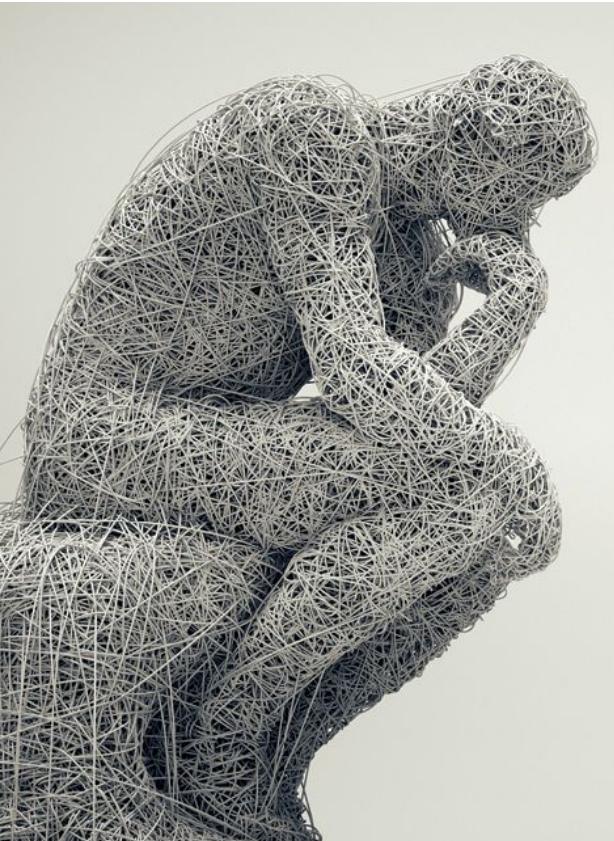
@cktricky





ABSOLUTE
APPSEC

PHILOSOPHY - PART 1

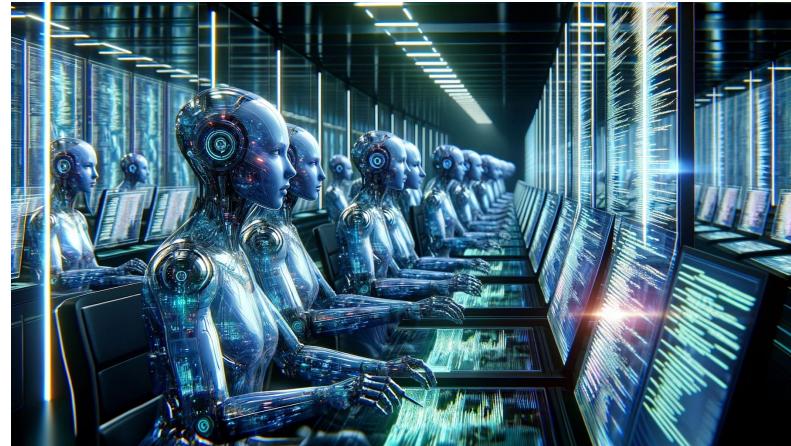


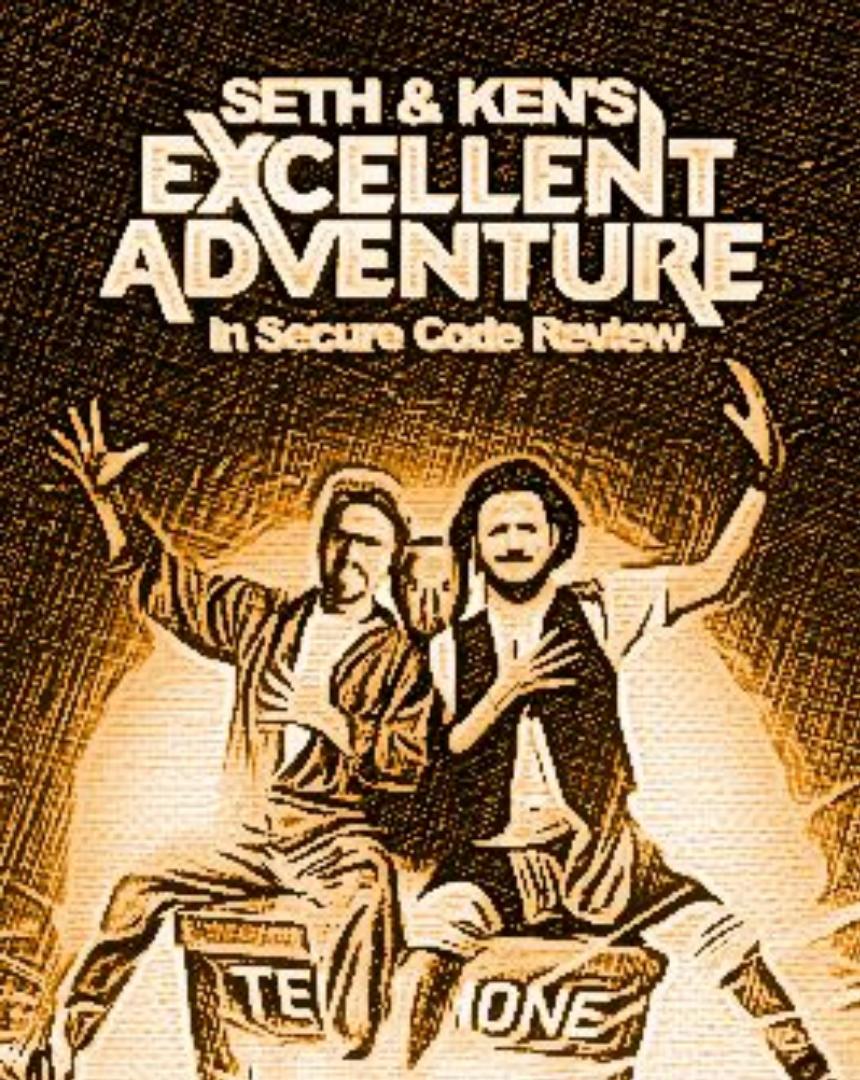
- Increase the likelihood of success.
- Successful defined as:
 - Risk-based (find what matters)
 - Comprehensive
 - Timely
 - Easily-consumed notes & report

PHILOSOPHY - PART 2

- Agnostic Principle
 - Become an expert in becoming an expert
 - Tech stack expertise/proficiency will come with repetition
- Repeatable, systematic approach to finding bugs
 - Caveat: You will tweak this approach over time to suit your specific needs

ENHANCE EVERYTHING WITH AI





SETH & KEN'S
EXCELLENT
ADVENTURE
in Secure Code Review

WHAT TO EXPECT

- Our Experience and Stories
- Framework Nuances
- Interactive Hands-On Exercises
- Practice, practice, practice

LET'S TALK ABOUT APPROACH

Review situations:

- You decide how much time you spend (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”

**NOTE: THERE IS NO SITUATION WHERE YOU SHOULD
“JUST SCAN IT”**

FIRST, LET'S TALK ABOUT WHAT WE'RE DOING HERE

At a high level, this course is intended to:

1. Explain risk-based code review principles
2. Show what to look for - Crocs & Socks, Vulnerabilities
(General & App Specific)
3. Show where to look - Attack Surface, Framework Nuances
4. Demonstrate how to enhance code reviews using Gen AI

CODE REVIEW & GEN AI CONSIDERATIONS

- Manual Code Review augmented with Gen AI
 - We are NOT scanning using AI
 - We ARE asking specific questions to speed up analysis
 - Gen AI is not perfect, understand this going in
 - **Context** will have two meanings:
 - i. **Context** we PROVIDE Gen AI
 - ii. **Context** we GAIN from using Gen AI

ALSO, LET'S TALK ABOUT WHAT WE AREN'T DOING

This course does ***not***:

1. Teach you the OWASP Top 10.
2. Review all known web vulnerabilities.
3. Have exercises for finding vulnerabilities in running applications (ie - dynamic analysis).
4. Instruct how to build secure applications
5. Cover all the ways that vulnerabilities manifest in code.

LET'S TALK ABOUT THE OWASP IN THE ROOM

- 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
2. Information Gathering
3. Checklist Creation
4. Perform Reviews

General Purpose Checklists

- Authorization
- Authentication
- Auditing
- Injection
- Cryptographic
- Configuration
- Reporting

THE CIRCLE-K FRAMEWORK

1. Open a file, take notes :-)
2. Identify 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

NOTE TAKING



NOTE TAKING

- Format
 - Commit # (if available)
 - Notes from the review
 - Route mapping
 - “Threat model” details (really, rambling thoughts about what could go wrong)
 - i. PAIR WITH OTHERS!!!
 - Checklist of items you’ve checked for based on your threat model but also your normal “things” you should look at

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 PROBABLY NOT A VULNERABILITY
- Image processing... RCE? Something else like traversal/LFI/RFI?

Checklist of things to review based on Brainstorming

- 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

- 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



Objectives:

- Identify what is most at risk
- Identify the most likely areas where opportunities to exploit said risks might manifest themselves
- Map the attack surface of an application

APPLICATION OVERVIEW & RISK ASSESSMENT

Common ways to gather information:

- README.md
- Unit Tests
- Documentation:
 - API Docs (Swagger, PandaDoc, etc.)
 - Wiki
 - /docs folder
- Database Schema & development data “seed” files
- Package library files (eg: Gemfile, Package.json, pom.xml)

```
~/code/vtm/taskManager master cat models.py
# Vulnerable Task Manager

import datetime

from django.contrib.auth.models import User

from django.utils import timezone
from django.db import models


class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    image = models.CharField(max_length=3000, default="")
    reset_token = models.CharField(max_length=7, default="")
    reset_token_expiration = models.DateTimeField(default=timezone.now)
    dob = models.CharField(max_length=8, default="00/00/00")
```

APPLICATION OVERVIEW & RISK ASSESSMENT

Many ways/places to gather information and gain context about an application using only the source code.

- README
- Unit Tests
- Documentation:
 - API Docs (Swagger, PandaDoc, etc.)
 - Wiki
 - /docs folder
- Database Schema & development data “seed” files
- Package library files (eg: Gemfile, Package.json, pom.xml)



Great time to use AI

APPLICATION OVERVIEW & RISK ASSESSMENT

Build a portrait of the application

- Behavior Profile
- Technology Stack
- App Archeology



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 - **django**   
- 3rd party components:
 - Supporting libraries (rubygem, npm, jar, etc.)
 - JavaScript widgets - (marketing tracking, sales chat widget)
 - Service to Service
- Datastore - Postgresql, MySQL, Memcache, Redis, Mongodb

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

BRAINSTORMING RISKS!

- Use the information we've collected
- Work with one or more (preferably more) people because diversity of thought equates to comprehensive checklists
- Often new risks will pop up the more you perform your review... expect and embrace it



APPLICATION OVERVIEW & RISK ASSESSMENT

<https://github.com/IMA-WorldHealth/bhima>

```
});  
  
const WEAK_PASSWORD = 'hello';  
const MEDIUM_PASSWORD = 'L0b1Ec0simba';  
const STRONG_PASSWORD = 'N@pM@ch3N#L1my3B0ndy3@!';  
  
beforeEach(inject(_PasswordMeterService_, _SessionService_, _MockDataService_) => {  
    PasswordMeterService = _PasswordMeterService;  
    Session = _SessionService;  
  
    const user = _MockDataService_.user();  
    const project = _MockDataService_.project();  
    const enterprise = _MockDataService_.enterprise();  
    Session.create(user, enterprise, project);  
  
    // make sure password validation is on  
    Session.enterprise.settings.enable_password_validation = true;  
});
```

```
it('#counter() should return -1 for no password', () => {  
    const count = PasswordMeterService.counter();  
    expect(count).to.equal(-1);  
});  
  
it('#validate() should return false for no password', () => {  
    const validate = PasswordMeterService.validate();  
    expect(validate).to.equal(false);  
});  
  
it('#counter() should return 0 for a weak password', () => {  
    const count = PasswordMeterService.counter(WEAK_PASSWORD);  
    expect(count).to.equal(0);  
});  
  
it('#validate() should return false for a weak password', () => {  
    const validate = PasswordMeterService.validate(WEAK_PASSWORD);  
    expect(validate).to.equal(false);  
});  
  
it('#counter() should return 3 for a medium password', () => {  
    const count = PasswordMeterService.counter(MEDIUM_PASSWORD);  
    expect(count).to.equal(3);  
});  
  
it('#validate() should return true for a medium password', () => {  
    const validate = PasswordMeterService.validate(MEDIUM_PASSWORD);  
    expect(validate).to.equal(true);  
});
```

GEN AI INTRODUCTION

DOWNLOAD .ENV FILE CRED\$

1. Check your email for subject “Credentials for Bedrock”
2. Create .env file in [handouts/tree/master/GenAI/v2](#) and place credentials into this file
3. You should now have access to Bedrock



GEN AI FUNDAMENTALS - TERMINOLOGY

-  We need to be able to give source code in a format suitable for LLMs (**Documents**)
-  We need to provide the LLM with instructions (**Prompts**)
-  We need to be able to provide **Documents**, **Prompts**, and other information to the LLM (**LLM Chain**)
-  The “Chain” becomes “RAG” or **Retrieval Augmented Generation**

GEN AI FUNDAMENTALS - TECHNOLOGY

Popular LLMs for Code Analysis (*right now*):

- OpenAI / GPT
- Claude / Anthropic models
- Gemini/Gemma
- LLaMa

Playgrounds:

- <https://deepinfra.com/chat>
- <https://labs.perplexity.ai/>

GEN AI FUNDAMENTALS - TECHNOLOGY



Amazon Bedrock

GEN AI FUNDAMENTALS - TECHNOLOGY

Run locally w/ Ollama



Langchain library to automate



LangChain

GEN AI FUNDAMENTALS - CODE

https://github.com/absoluteappsec/handouts/blob/master/GenAI/v2/readme_ingestion.py



Basic example:

1. Retrieve Juice-Shop Readme
2. Put it into a “Document” format
3. Ask the LLM to summarize the contents of the README

GEN AI FUNDAMENTALS - CODE

Basic example:

README

Summarization -

Part 1

```
# UNCOMMENT FOR OLLAMA
#llm = Ollama(model="llama3", temperature=0.6)
llm = ChatBedrock(
    model_id='anthropic.claude-3-haiku-20240307-v1:0',
    model_kwargs={"temperature": 0.6},
)

question = """
QUESTION
=====
{question}

CONTEXT
=====
{context}
"""

prompt = ChatPromptTemplate.from_template(template=question)
```

GEN AI FUNDAMENTALS - CODE

Basic example:

README

Summarization

Part 2

```
# readme_ingestion.py > ...
19 README_URL = 'https://raw.githubusercontent.com/juice-shop/juice-shop/master/README.md'
20 response = requests.get(README_URL)
21 if response.status_code == 200:
22     readme_content = response.content
23     doc = Document(
24         page_content=readme_content,
25         metadata={"source": "README.md"}
26     )
27     chain = [
28         {"context": RunnablePassthrough(), "question": RunnablePassthrough()},
29         || prompt
30         | llm
31         | StrOutputParser()
32     ]
33     reply = chain.invoke({
34         "question": "You are being provided the README file from a software
35             project. Please provide a summary of the purpose of the application and
36             any other relevant details you can think to share.",
37         "context": doc
38     })
39     print(reply)
```

GEN AI FUNDAMENTALS - CODE

Basic example: README Summarization Part 3

- * The Juice Shop is a web application that allows users to practice ethical hacking and test their skills against a real-world target environment.
- * It was developed by Björn Kimminich, a penetration testing expert at the Open Web Application Security Project (OWASP).
- * The app has over 30 vulnerable functions, including SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF) attacks.
- * Users can use various tools to exploit these vulnerabilities and gain access to the web application's underlying data.
- * The Juice Shop is intended for testing and training purposes only, as it exposes sensitive information such as passwords and credit card numbers.
- * It has been certified by OWASP as a "Project" and is part of the organization's "Web Security Testing Guide".
- * The app has received numerous awards, including the 2018 "Hack the Box" challenge winner at the German Cybersecurity Challenge.

**BUT IF WE WANT TO LOAD MORE THAN A
README...**

GEN AI FUNDAMENTALS - CODE

https://github.com/absoluteappsec/handouts/blob/master/GenAI/v2/profile_app.py



Advanced example:

1. Retrieve Juice-Shop Source
2. Put useful parts of code into a “Document” format
3. Store in vector db
4. Ask the LLM to summarize important key details using vectorized codebase to retrieve that information

GEN AI FUNDAMENTALS - CODE (DEEPER INSPECTION)

```
repo_url = 'https://github.com/juice-shop/juice-shop.git'  
local_path = './repo'  
  
if os.path.isdir(local_path) and os.path.isdir(os.path.join(local_path, '.git')):  
    print("Directory already contains a git repository.")  
else:  
    try:  
        repo = git.Repo.clone_from(repo_url, local_path)  
        print(f"Repository cloned into: {local_path}")  
    except Exception as e:  
        print(f"An error occurred while cloning the repository: {e}")
```

GEN AI FUNDAMENTALS - CODE (DEEPER INSPECTION)

```
docs = []
for dirpath, dirnames, filenames in os.walk(local_path):
    for file in filenames:
        extension = Path(file).suffix
        if (
            extension == '.md' or
            extension == '.ts' or
            file == 'package.json' or
            file == 'compose.yml'):
            try:
                print(f"Processing {file}")
                loader = TextLoader(os.path.join(dirpath, file), encoding='utf-8')
                docs.extend(loader.load_and_split())
            except Exception as e:
                pass
```

GEN AI FUNDAMENTALS - CODE (DEEPER INSPECTION)

```
embeddings = BedrockEmbeddings(model_id='amazon.titan-embed-text-v1')

splitter = RecursiveCharacterTextSplitter(
    chunk_size=8000,
    chunk_overlap=20
)

texts = splitter.split_documents(docs)

system_prompt_template = """
You are a helpful code review assistant who is proficient
in both security as well as functional review. You will be
provided source code of a web application and tasked with
answering questions about it.

<context>
{context}
</context>
"""

```

GEN AI FUNDAMENTALS - CODE (DEEPER INSPECTION)

```
# CORRECT/FORMAL WAY TO PERFORM PROMPTING
prompt = ChatPromptTemplate.from_messages(
    [
        ("system", system_prompt_template),
        ("human", """<question>{question}</question>""")
    ]
)

# UNCOMMENT FOR OLLAMA/LLAMA
#llm = Ollama(model="llama3.1", temperature=0.6)

llm = ChatBedrock(
    model_id='anthropic.claude-3-haiku-20240307-v1:0',
    model_kwargs={"temperature": 0.6},
)

db = FAISS.from_documents(texts, embeddings)

retriever = db.as_retriever(
    search_type="mmr", # Also test "similarity"
    search_kwargs={"k": 8},
)
```

GEN AI FUNDAMENTALS - CODE (DEEPER INSPECTION)

```
chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

GEN AI FUNDAMENTALS - CODE (DEEPER INSPECTION)

```
# This is an optional addition to stream the output in chunks
# for a chat-like experience
for chunk in chain.stream():
    """Tell me the following information about the code base I am providing you:
    - Purpose of the Juice Shop application
    - Web technologies used in the Juice Shop application
    - Templating language used in the Juice Shop application
    - Database used in the Juice Shop application
    - Authentication mechanisms used in the Juice Shop application
    - Authorization mechanisms used in the Juice Shop application

    List libraries by their name, purpose, and version that are
    used in the Juice Shop application for the following categories:
    - Security
    - Testing
    - Documentation
    - Build
    - Database
    - Authentication / Authorization
    - HTML Templating (ex: pug, handlebars)
    - CSS Frameworks (ex: bootstrap, tailwind)
    - widgets / UI components
    ....
```

GEN AI FUNDAMENTALS - SUMMARY

- We learned how to provide various sources of information to the LLM (swagger file, unit tests, README, etc.)
- We passed questions and contextually relevant information to the LLM
- Leveraged that information to decipher what the application is doing

GEN AI FUNDAMENTALS - CAVEATS/NOTES (PART 1)



- Llama2/CodeLLaMa is TERRIBLE at differentiating between context that *is* code and actual code. When needed, we use other LLMs
- Langchain and other abstraction libraries may not write the prompt tags correctly for you and so you may have to do this yourself.

GEN AI FUNDAMENTALS - CAVEATS/NOTES (PART 2)

```
second_pass_prompt_template = """  
    <s>[INST] <>SYS>>  
        You will be provided code formatted in git's "patch" or "diff" format. You will also be provided with the name of the file (file  
        FIELDS  
        =====  
        type - This is the type of vulnerability  
        code - This is the snippet of code that is vulnerable  
        description - The reason the code is vulnerable  
        line_start - is where the vulnerable function begins  
        line_end - is where the definition of the function ends  
        filename - is the name of the file containing the vulnerable code  
        confidence - This is a decimal representation from 0.0 - 10.0 and represents the confidence you have in the answer being provided  
        risk - This is a decimal representation from 0.0 - 10.0 and represents the risk of the vulnerability. (10.0 being the highest risk)  
    <>/SYS>>  
  
    Filename: {filename}  
  
    Does the following code contain a high risk vulnerability? Code:  
    {question} [/INST]  
  
    ....
```

APPLICATION OVERVIEW & RISK ASSESSMENT - EXAMPLE

ⓘ README.md

≡ Preview: 'Preview: 'README.md'' ×

□ ...

skea_django

SKEA == Seth & Ken's Excellent Adventures (in Code Review) A small django application for use in demo-ing django routes, etc.

Requirements

Python3 + Django > 2.1

Quickstart

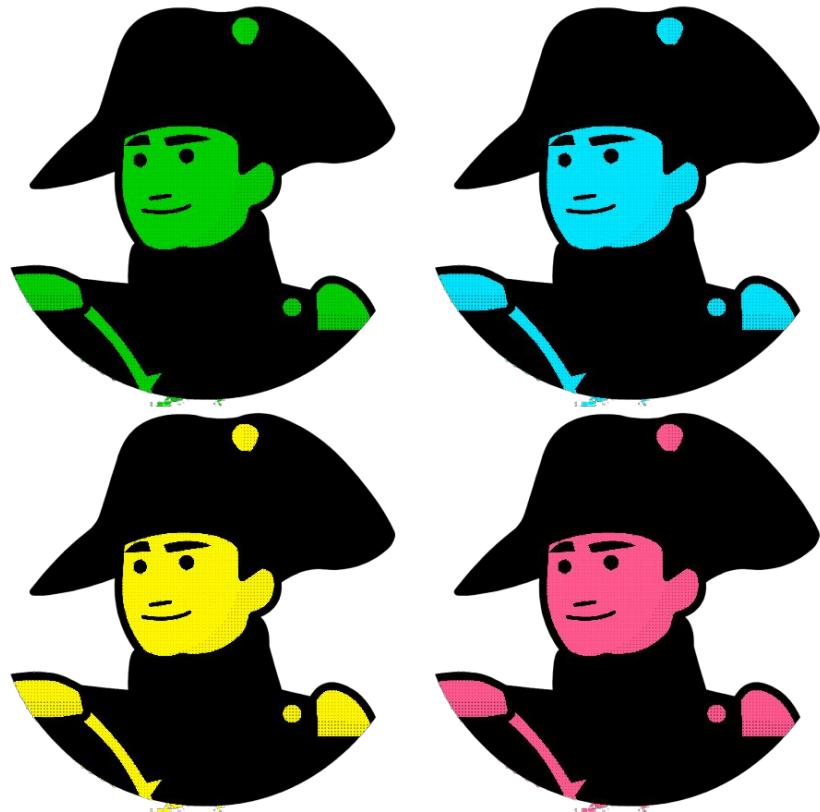
1. pip3 install -r requirements.txt
2. ./manage.py migrate
3. ./manage.py runserver
4. Navigate to <http://localhost:8000/>

APPLICATION OVERVIEW & RISK ASSESSMENT - EXERCISE

NAPOLEON

1. bhima

- a. Business Purpose
- b. Tech Stack
- c. Application Risk
- d. Anything else?



NAPOLEON EXERCISE - POST-MORTEM

At a minimum, these items were of note:

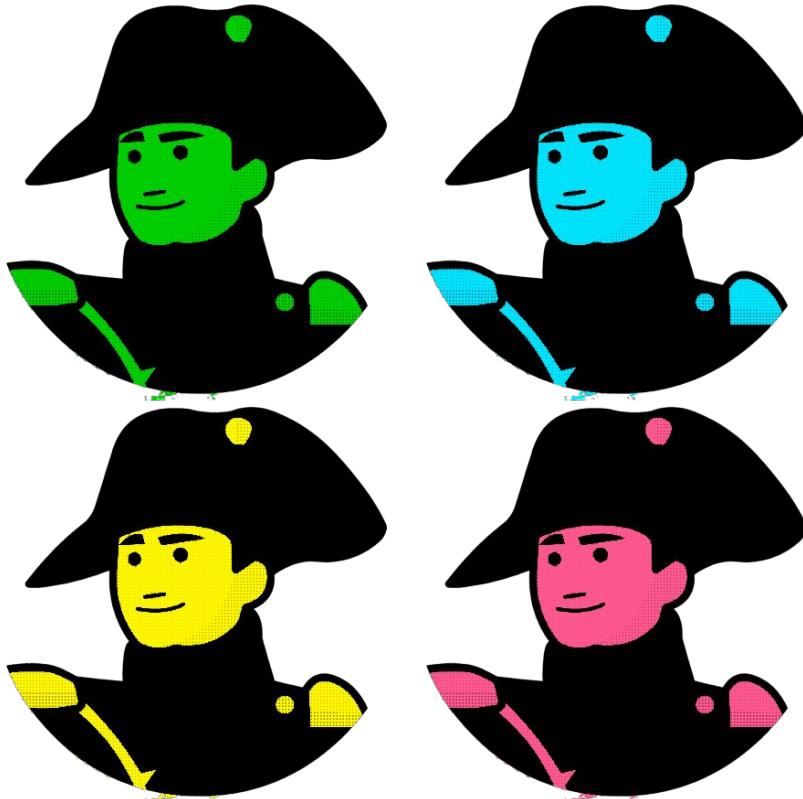
- **Business Purpose**
 - This is a community management platform for Tech Communities across the world. The inspiration being the personal experiences of Community Organizers.
- **Tech Stack**
 - Node.js / Express
 - Angular
- **Documentation**
 - <https://github.com/IMA-WorldHealth/bhima/wiki>
 - <https://docs.bhi.ma/en/>
- **Risks**
 - Patient Data
 - Employee Payroll Information



NAPOLEON EXERCISE - POST-MORTEM

Other important security portions
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



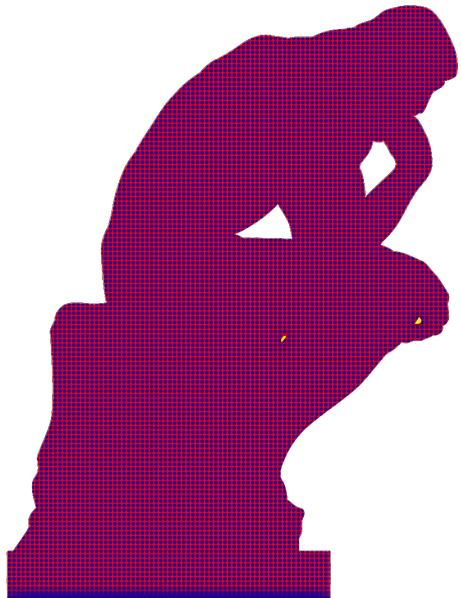
INFORMATION GATHERING

INFORMATION GATHERING

“THE MOST IMPORTANTEST THING IS TO UNDERSTAND
THE APP”

“- @sethlaw”

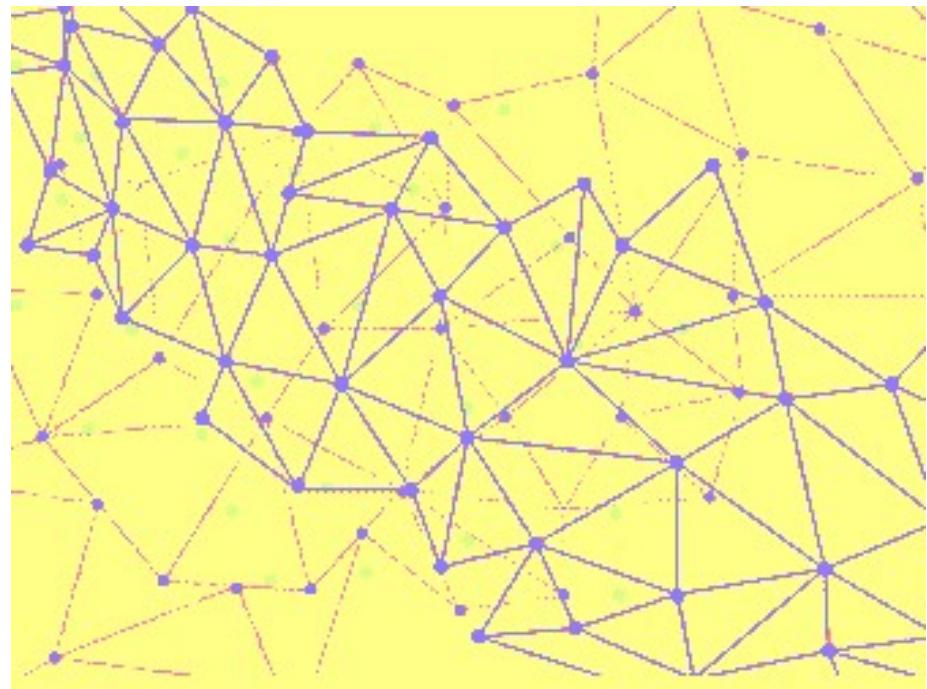
- @cktricky



INFORMATION GATHERING

Steps

1. Create Application Map and Attack Surface
2. Identify Authorization Functions

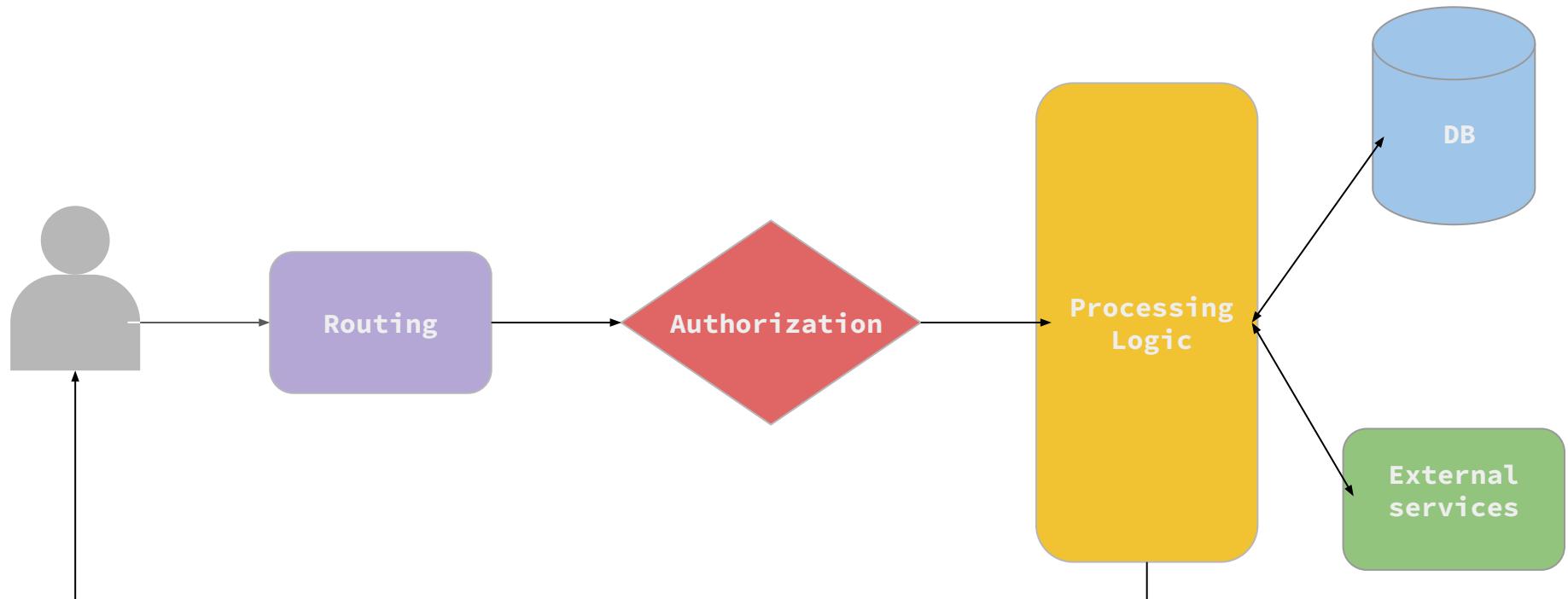


MAPPING

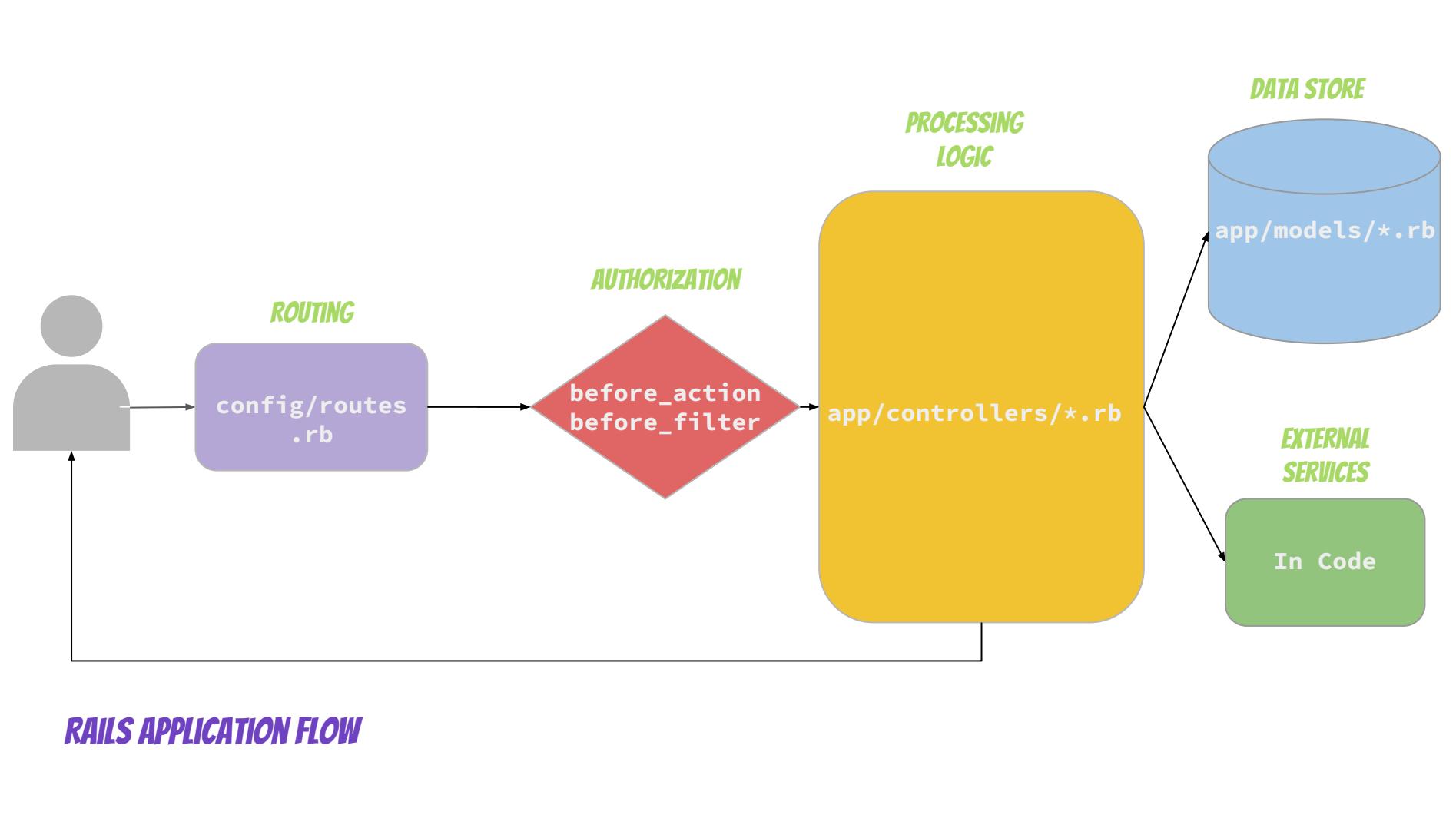


INFORMATION GATHERING - CREATE A MAP

- Identify endpoints. Some frameworks have tools:
 - Rails = config/routes.rb - `rake routes`
 - Django= urls.py - `manage show_urls`
- In other situations, we'll probably harness Gen AI
- Endpoints typically have at least three qualities
 - Authorization Filter
 - Logic processing
 - Datastore access



TYPICAL 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

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

Sometimes, you won't have a nifty tool... **GEN AI!**

ChatGPT 3.5 ▾

You

The following is a routes.rb file from a Ruby on Rails application. Please ingest this content and produce output similar to what running the command "rake routes" would produce:

```
# frozen_string_literal: true
Railsgoat::Application.routes.draw do

  get "login" => "sessions#new"
  get "signup" => "users#new"
  get "logout" => "sessions#destroy"

  get "forgot_password" => "password_resets#forget_password"
  post "forgot_password" => "password_resets#send_forget_password"
  get "password_resets" => "password_resets#confirm_token"
  post "password_resets" => "password_resets#reset_password"

  get "dashboard/doc" => "dashboard#doc"
```

ChatGPT 3.5 ▾

ChatGPT

The output similar to what running the command "rake routes" would produce for the provided routes.rb file is as follows:

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#forget_password
	POST	/forgot_password(.:format)	password_resets#send_forget_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(.:f__at)	sessions#create

RAILS - ROUTING WEIRDNESS ALERT

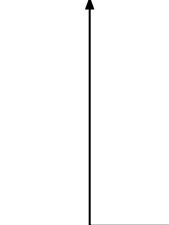
- What do **PUT**, **PATCH**, **DELETE**, and **POST** all have in common?
- That's right, they are all **POST** requests
- Rails uses the **_method** parameter to determine what kind of request it is

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



ROUTING

index.js
(could be
other places)

AUTHORIZATION

app.post()
app.get()

PROCESSING LOGIC

Last call in route
definition
app.post("/", index);

DATA STORE

Typically in
processing logic.

EXTERNAL SERVICES

In Code

NODE.JS/EXPRESS - MAP

Route definition, destructured

42

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

NODE.JS/EXPRESS - MAP

- Our formula used to be super basic, we'd search for:
 - `app.get`
 - `app.post`
 - `app.delete`
- Problem is that routes can be defined in a myriad of ways and using functions other than the above
- We can also ask Express directly, we show a version of this here:

https://github.com/absoluteappsec/handouts/blob/master/nodejs_generic_checks.md#print-all-routes

OR... PERFORM MAPPING W/ GEN AI

NODE.JS/EXPRESS - MAP WITH GEN AI PART 1

1. Clone repo
2. Load from file system into memory

```
repo_url = 'https://github.com/absoluteappsec/skea_node.git'
local_path = './repo'

if os.path.isdir(local_path) and os.path.isdir(os.path.join(local_path, '.git')):
    print("Directory already contains a git repository.")
else:
    try:
        repo = git.Repo.clone_from(repo_url, local_path)
        print(f"Repository cloned into: {local_path}")
    except Exception as e:
        print(f"An error occurred while cloning the repository: {e}")

loader = GenericLoader.from_filesystem(
    local_path,
    glob="**/*",
    suffixes=[".js"],
    parser=LanguageParser(language=Language.JS),
    show_progress=True
)

documents = loader.load()
```

NODE.JS/EXPRESS - MAP WITH GEN AI PART 2

```
embeddings = BedrockEmbeddings(model_id='amazon.titan-embed-text-v1')

splitter = RecursiveCharacterTextSplitter.from_language(language=Language.JS,
    chunk_size=8000,
    chunk_overlap=20
)
texts = splitter.split_documents(documents)

db = FAISS.from_documents(texts, embeddings)

system_prompt_template = """
You are a helpful code assistant who is given access to a
code base stored in vector format. You will be asked questions about that code.
Please provide helpful and accurate responses to the best of your ability.

</context>
{context}
</context>
"""

```

1. Split code loaded in memory by top level functions/classes
2. Create a FAISS db to store code
3. Create template that pulls in context

NODE.JS/EXPRESS - MAP WITH GEN AI PART 3

1. Create prompt
2. Create retriever which access code base
3. Choose your LLM

```
# CORRECT/FORMAL WAY TO PERFORM PROMPTING
prompt = ChatPromptTemplate.from_messages(
    [
        ("system", system_prompt_template),
        ("human", """<question>{question}</question>""")
    ]
)
retriever = db.as_retriever(
    search_type="mmr", # Also test "similarity"
    search_kwargs={"k": 8},
)

# UNCOMMENT FOR OLLAMA/LLAMA
#llm = Ollama(model="llama3.1", temperature=0.6)

llm = ChatBedrock(
    model_id='anthropic.claude-3-haiku-20240307-v1:0',
    model_kwargs={"temperature": 0.6},
)
```

NODE.JS/EXPRESS - MAP WITH GEN AI PART 3

1. Create chain
2. Give question with an example
3. Ask the question and stream the response

```
chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)

question = """
Please provide me with a summary of where the HTTP
routes are defined in this code base. Please print them out in a format
similar to how rake routes would display them with the following collumns:

VERB | URI | FILE | LINE NUMBERS

Example:

GET | / | app.js | 10-20
::::

# This is an optional addition to stream the output in chunks
# for a chat-like experience
for chunk in chain.stream(question):
    print(chunk, end="", flush=True)
```

NODE.JS/EXPRESS - MAP WITH GEN AI PART 5

Supported languages for splitting (with more being added):

```
9  ↘  map_extension_to_language = {  
10      "py": Language.PYTHON,  
11      "js": Language.JS,  
12      "java": Language.JAVA,  
13      "cpp": Language.CPP,  
14      "hpp": Language.CPP,  
15      "cs": Language.CSHARP,  
16      "go": Language.GO,|  
17      "rs": Language.RUST,  
18      "rst": Language.RST,  
19      "swift": Language.SWIFT,  
20      "php": Language.PHP,  
21      "rb": Language.RUBY,  
22      "kt": Language.KOTLIN,  
23      "scala": Language.SCALA,  
24      "ts": Language.TS,  
25      "proto": Language.PROTO,  
26      "html": Language.HTML,  
27      "md": Language.MARKDOWN,  
28      "tex": Language.LATEX,  
29      "html": Language.HTML,  
30      "sol": Language.SOL,  
31      "cbl": Language.COBOL,  
32      "cob": Language.COBOL  
33  }  
34
```

NODE.JS/EXPRESS - MAP WITH GEN AI PART 5

Repository cloned into: ./repo

100% | 4/4 [00:

Here is a summary of the HTTP routes defined in the code base:

VERB	URI	FILE	LINE NUMBERS
------	-----	------	--------------

---	---	---	---
-----	-----	-----	-----

GET	/	routes/index.js	4-6
-----	---	-----------------	-----

GET	/users	routes/users.js	4-6
-----	--------	-----------------	-----

GET	/users/w*f	routes/users.js	9-11
-----	------------	-----------------	------

GET	/users/wtf+!	routes/users.js	14-16
-----	--------------	-----------------	-------

GET	/debug	app.js	26-28
-----	--------	--------	-------

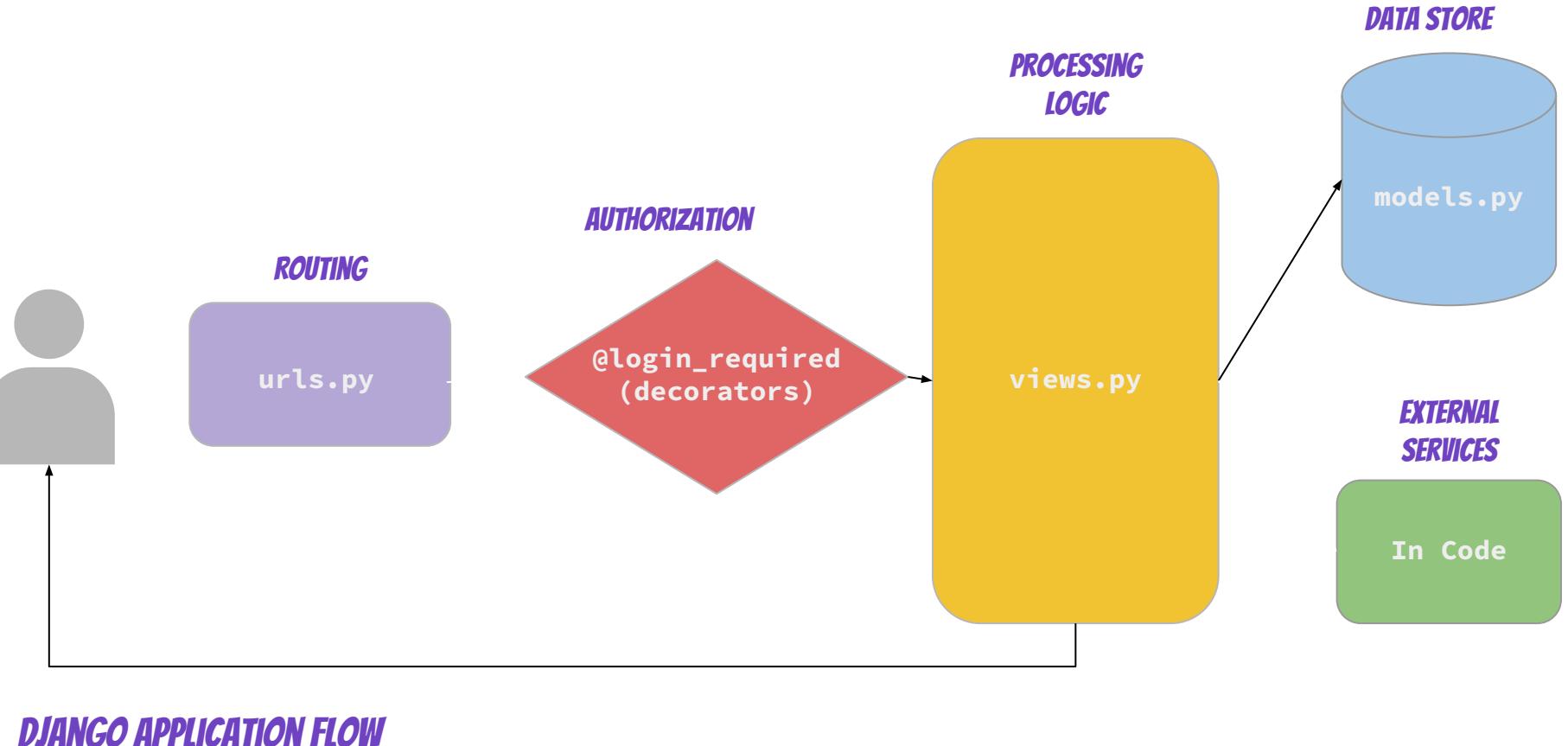
The routes are defined in the following files:

- `app.js` defines the `/debug` route.
- `routes/index.js` defines the root `/` route.
- `routes/users.js` defines the `/users`, `/users/w*f`, and `/users/wtf+!` routes.

NODE.JS/EXPRESS - MAP WITH GEN AI (CODE LOCATION)

You can access the code at

https://github.com/absoluteappsec/handouts/blob/master/GenAI/find_routes_express.py



DJANGO - MAP

```
cktricky@Kens-MBP ~code/vtm master ./manage.py show_urls
/      taskManager.views.index index login_required
/admin/ django.contrib.admin.sites.index admin:index
/admin/<app_label>/ django.contrib.admin.sites.app_index admin:app_list
/admin/auth/group/ django.contrib.admin.options.changelist_view admin:auth_group_changelist
/admin/auth/group/<path:object_id>/ django.views.generic.base.RedirectView
/admin/auth/group/<path:object_id>/change/ django.contrib.admin.options.change_view admin:auth_group_change
/admin/auth/group/<path:object_id>/delete/ django.contrib.admin.options.delete_view admin:auth_group_delete
/admin/auth/group/<path:object_id>/history/ django.contrib.admin.options.history_view admin:auth_group_history
/admin/auth/group/add/ django.contrib.admin.options.add_view admin:auth_group_add
/admin/auth/group/autocomplete/ django.contrib.admin.options.autocomplete_view admin:auth_group_autocomplete
/admin/auth/user/ django.contrib.admin.options.changelist_view admin:auth_user_changelist
/admin/auth/user/<id>/password/ django.contrib.auth.admin.user_change_password admin:auth_user_password_change
/admin/auth/user/<path:object_id>/ django.views.generic.base.RedirectView
/admin/auth/user/<path:object_id>/change/ django.contrib.admin.options.change_view admin:auth_user_change
/admin/auth/user/<path:object_id>/delete/ django.contrib.admin.options.delete_view admin:auth_user_delete
/admin/auth/user/<path:object_id>/history/ django.contrib.admin.options.history_view admin:auth_user_history
/admin/auth/user/add/ django.contrib.auth.admin.add_view admin:auth_user_add
/admin/auth/user/autocomplete/ django.contrib.admin.options.autocomplete_view admin:auth_user_autocomplete
/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:password_change
/admin/password_change/done/ django.contrib.admin.sites.password_change_done admin:password_change_done
```

DJANGO - MAP (*SEMGREP)

<https://semgrep.dev/s/minusworld:docs-missing-auth-annotation>

New Load Examples Tools Help

Python save ↗

SEMGREP RULE

Simple Advanced

code is

```
@$APP.route(...)  
def $FUNC(...):  
    ...
```

+ x

and is not inside

```
@$APP.route(...)  
@login_required  
def $FUNC(...):  
    ...
```

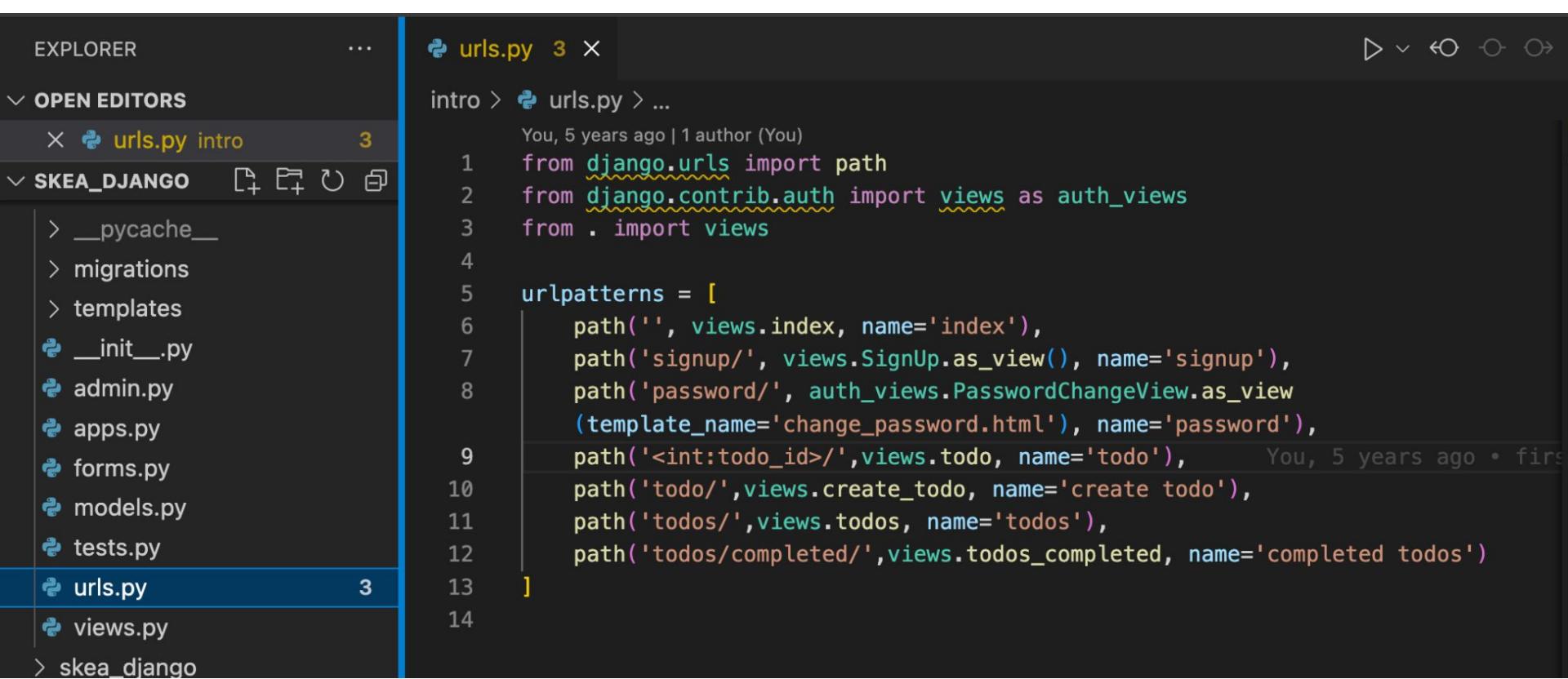
+ x

TEST CODE

```
1 import flask  
2 app = flask.Flask()  
3  
4 @app.route("/echo/<msg>", methods=["POST"])  
5 @login_required  
6 def echo(msg):  
7     return msg  
8  
9 @app.route("/reverse/<msg>")  
10 def echo_reverse(msg):  
11     return msg.reverse()
```

Run ↗

MAPPING EXAMPLE



The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: Shows the project structure with files like __pycache__, migrations, templates, __init__.py, admin.py, apps.py, forms.py, models.py, tests.py, urls.py (selected), and views.py.
- OPEN EDITORS** sidebar: Shows the urls.py file with 3 changes.
- urls.py Editor View:**
 - Path: intro > urls.py > ...
 - Author: You, 5 years ago | 1 author (You)
 - Content:

```
from django.urls import path
from django.contrib.auth import views as auth_views
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('signup/', views.SignUp.as_view(), name='signup'),
    path('password/', auth_views.PasswordChangeView.as_view(
        template_name='change_password.html'), name='password'),
    path('<int:todo_id>', views.todo, name='todo'),
    path('todo/', views.create_todo, name='create todo'),
    path('todos/', views.todos, name='todos'),
    path('todos/completed/', views.todos_completed, name='completed todos')
]
```
 - Small note at the bottom right: You, 5 years ago • first

MAPPING EXERCISE

RUFUS

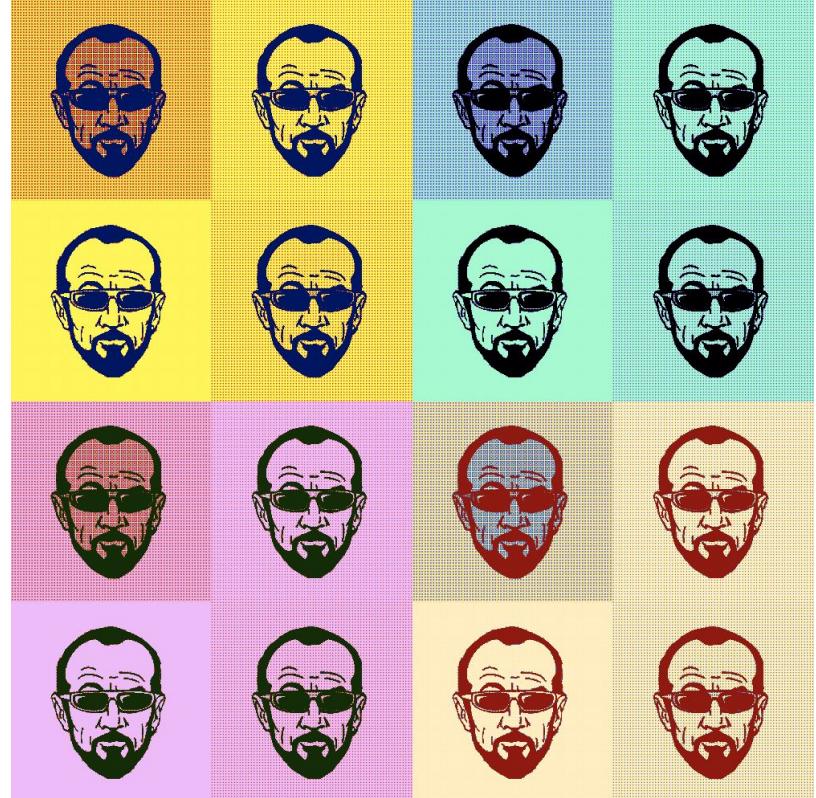
1. skea_node

- a. Perform Info Gathering
 - i. Tech Stack/Purpose/Risk

- b. Map Routes

2. bhima

- a. Map Routes
 - i. At least 10



RUFUS - POST-MORTEM

- Notes should include:
 - Commit #
 - End points shown here
 - Any else you found interesting, such as:
 - auth function doesn't do anything
 - debug statement in our app and its printing env variables

The HTTP routes in this Express.js application are defined in the `routes` directory, which contains two files:

1. `index.js`: This file defines a route for the root URL (`'/'`) that renders the `users/index` view using the `res.render()` method.
2. `users.js`: This file defines a route for URLs that start with `/w*f` (i.e., any string that starts with `/w` followed by `*f`), which also renders the `users/index` view using the `res.render()` method.

In both cases, the route handler is defined as a function that takes three arguments:

1. `req`: The HTTP request object, which contains information about the incoming HTTP request (e.g., URL, headers, etc.).
2. `res`: The HTTP response object, which is used to send a response back to the client.
3. `next`: A callback function that should be called once the route handler has finished executing in order to move on to the next middleware or route in the chain.

In this codebase, there are no routes defined using `app.get()`, `app.post()`, etc. methods, but instead the routes are defined using a single file named `index.js` under the `routes` directory.

AUTHORIZATION FUNCTIONS

INFORMATION GATHERING - AUTHORIZATION FUNCTIONS

- A later step is dedicated to creating authorization checks
- This is about getting to know the application better
- Get to know how users are identified/authorized to perform actions

INFORMATION GATHERING - AUTHORIZATION FUNCTIONS

- Patterns & Anti-patterns
 - How do we identify the user? eg: Session, Token, Basic Auth
 - What is the purpose? Authenticated users, Role check, or something else?
- Identify what could go wrong
 - User-supplied parameters (IDOR)
 - CSRF
 - Client-side cookies, JWTs etc. (just general wonkiness in persistence)

STORY TIME:

REDIRECTION & AUTHORIZATION IN .NET



301

REDIRECTION & AUTHORIZATION ISSUE IN .NET

NORMAL

Results Target Positions Payloads Options

Filter: Showing all items

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 for an "Intruder attack 17". The top navigation bar includes tabs for Results, Target, Positions, Payloads, and Options. The Results tab is selected. A filter bar below the tabs says "Filter: Showing all items". The main table displays 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.

AUTHORIZATION FUNCTIONS - INCORRECT 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

- Identify the unique values used to identify an actor:
 - Sessions
 - API Tokens
 - Basic Authentication
 - JWTs
 - Something... else?

AUTHORIZATION FUNCTIONS CHECKLIST

- Identify the purpose of the function:
 - Is it checking that we're authenticated?
 - AuthN
 - Is it looking for a specific role?
 - RBAC
 - Is it doing some sort of HMAC verification for events?
 - CSRF

AUTHORIZATION FUNCTIONS CHECKLIST

- Brainstorm / Identify what could go wrong:
 - Insecure timing comparisons
 - IDOR / BOLA / MFLAC
 - Lack of rate limiting
 - Not applied uniformly
 - Insecure CSRF implementation (CSRF = Authz)
 - **Get creative, you're trying to decide what the risk, if any, is here**

EXPLORER

...

views.py 6 ×

▷ ⏴ ⏵ ⏴ ⏵ ⏴ ⏵ ⏴ ⏵ ⏴ ⏵

OPEN EDITORS

views.py intro 6

SKEA_DJANGO

> __pycache__

> migrations

> templates

views.py

admin.py

apps.py

forms.py

models.py

tests.py

urls.py

views.py 6

> skea_django

> venv

.gitignore

db.sqlite3

intro > views.py > index

cktricky, 17 months ago | 2 authors (You and others)

```
1 from django.http import HttpResponseRedirect, HttpResponseRedirect, Http404
2 from django.contrib.auth.decorators import login_required
3 from django.shortcuts import render
4 from django.urls import reverse_lazy
5 from django.views import generic
6
7 from .forms import TodoUserCreationForm, TodoForm
8 from .models import Todo
9
10 import logging
11 logger = logging.getLogger('django')
12
13 # Create your views here.
14 def index(request):|    You, 5 years ago • first commit ...
15     return HttpResponseRedirect("Welcome to Seth & Ken's Excellent Adventures (in Code
16                               Review). Party on, dudes!")
17 @login_required
18 def todo(request, todo_id):
19     if request.method == 'GET':
```

AUTHORIZATION FUNCTIONS EXAMPLE

The image shows a code editor interface with a sidebar on the left displaying a file tree and a main pane on the right showing the content of a Python file.

File Tree:

- models.py
- tests.py
- urls.py
- views.py
- skea_django
 - > __pycache__
 - __init__.py
- settings.py** (highlighted in blue)
- urls.py
- wsgi.py
- > venv
- .gitignore
- db.sqlite3
- info.log

Code Content (settings.py):

```
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',
47     'django.middleware.common.CommonMiddleware',
48     'django.middleware.csrf.CsrfViewMiddleware',
49     'django.contrib.auth.middleware.AuthenticationMiddleware' (highlighted in pink)
50     'django.contrib.messages.middleware.MessageMiddleware',
51     'django.middleware.clickjacking.XFrameOptionsMiddleware',
52 ]
53
54 ROOT_URLCONF = 'skea_django.urls'
55
```

AUTHORIZATION FUNCTIONS - GEN AI PART 1

Code located at:

[https://github.com/absoluteapps
ec/handouts/blob/master/GenAI
/v2/list_authz_decorators.py](https://github.com/absoluteapps/ec/handouts/blob/master/GenAI/v2/list_authz_decorators.py)



```
# This is an optional addition to stream the output in chunks
# for a chat-like experience
for chunk in chain.stream("Which Django authorization decorators are used in
this application code base and where are they located?"):
    print(chunk, end="", flush=True)
```

AUTHORIZATION FUNCTIONS - GEN AI PART 2

The Django authorization decorators used in this application code base are:

1. `@login_required`:

- This decorator is used to ensure that a user is logged in before accessing certain views. It is used in multiple functions throughout the `views.py` file, such as `manage_tasks`, `manage_projects`, `manage_groups`, `upload`, `download`, `download_profile_pic`, `task_create`, `task_edit`, `task_delete`, `task_complete`, `project_create`, `project_edit`, `project_delete`, `index`, `profile_view`, `project_details`, `note_create`, `note_edit`, `note_delete`, `task_details`, `project_list`, `task_list`, `search`, and `profile`.

2. `@user_passes_test(can_create_project)`:

- This decorator is used to ensure that the user has the necessary permissions to create a new project. It is used in the `project_create` function.

3. `@user_passes_test(can_edit_project)`:

- This decorator is used to ensure that the user has the necessary permissions to edit a project. It is used in the `project_edit` function.

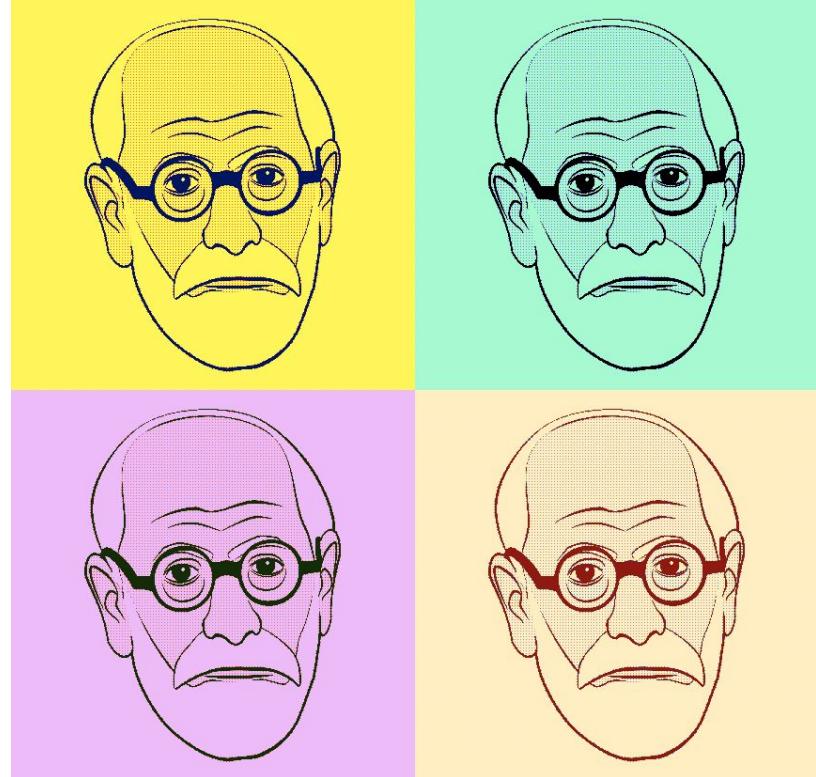
4. `@user_passes_test(can_delete_project)`:

- This decorator is used to ensure that the user has the necessary permissions to delete a project. It is used in the `project_delete` function.

AUTHZ FUNCTIONS EXERCISE

SIGMUND FREUD

1. **bridge_troll**
 - a. Perform Info Gathering
 - i. Tech/Purpose/Risk
 - b. Map Routes
 - c. Identify AuthZ Functions
2. How are users authorized in **bridge_troll**?
 - a. Could take some time



SIGMUND FREUD - POST-MORTEM

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?

```
class ApplicationController < ActionController::Base
  protect_from_forgery

  include Pundit
  rescue_from Pundit::NotAuthorizedError, with: :user_not_authorized
  after_action :verify_authorized, unless: :devise_controller?

  before_action :configure_permitted_parameters, if: :devise_controller?

  before_action do
    | Rack::MiniProfiler.authorize_request if current_user.try(:admin?)|
  end
```

AUTHZ FUNCTIONS - POST-MORTEM

So what's the point of this whole thing?

- Do your homework on framework nuances
- Good example of why we double-back and check one last time... (story time)



AUTHORIZATION FUNCTIONS - POST-MORTEM

“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
```

AUTHORIZATION FUNCTIONS - 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
```

PULL REQUEST REVIEWS

PULL REQUEST REVIEWS - OVERVIEW

Not an end-to-end code review

Snippets are often accompanied
by fuzzy context and
understanding context is critical

Common components:

- Tests (and there Needs to be tests)
- Conversation (context)
- Code

Allow iframe inside details liquid tag #11413

Closed akashdotsrivast... wants to merge 2 commits into `forem:main` from `akashdotsrivastava:akashdotsrivastava/fixing-parsing-of-yt-tags-in-details-tag-#11060`

Conversation 16 Commits 2 Checks 3 Files changed 5 +47 -1

akashdotsrivastava commented on Nov 15, 2020 · edited

What type of PR is this? (check all applicable)

Refactor
 Feature
 Bug Fix
 Optimization
 Documentation Update

Description

Added iframe and its missing attributes to the list of allowed tags and attributes for `sanitize` Rails helper, which allows `iframe` elements in details tag to be parsed correctly when sanitized.

Additional Changes

The issue was with `sanitize` used in

```
forem/app/liquid_tags/details_tag.rb
Line 13 in 918f79e
13   parsed_content = sanitize(content.xpath("//html/body").inner_html)
```

It by default, does not allow `iframe` as a tag while sanitizing, hence the youtube tag render, an iframe equivalent, though present in the xpath available, was not passing `sanitize`.

Unfortunately, there is no way to `add / insert` a new tag to the list of default allowed tags for `sanitize`. It either takes the default list (with no tags and attributes option), or only takes a list of tags and attributes to be allowed while sanitization, through array options tags and attributes.
<https://api.rubyonrails.org/classes/ActionView/Helpers/SanitizeHelper.html#method-i-sanitize>

To allow `iframe` along with all the existing tags and attributes by default, I picked the default lists of tags and attributes from <https://github.com/rails/rails-html-sanitizer/blob/5c64c650920107072456bb2c13fb7bb373d6/lib/html/sanitizer.rb#L108-L111>. I added the required tag for youtube video `iframe`, and a couple of missing attributes for the same in the attributes list.

Related Tickets & Documents

Closes #11060

QA Instructions, Screenshots, Recordings

1. Create a new post with this content (a details tag with some text content and a youtube video tag)

```
{% details Details and Video %}
This is a youtube video
[scratches out YouTube URL]
```

PULL REQUEST REVIEWS - OVERVIEW

General steps:

- Get to know the app
- Get to know the purpose of the pull request
- Read through other's comments
- Use git blame if necessary
- **Ask questions!!!**
- Seek data flow diagrams or something similar
- Common flaws associated with the portion of code you are reviewing (eg: controller == authz issues)

PULL REQUEST REVIEWS - OVERVIEW

- Factors that indicate risk:
 - Authorship - Never pushed to the repo, isn't on the team that owns the code, etc.
 - Code that has similar patterns to previously discovered vulnerabilities
 - Modification of brittle, complex, and/or infrequently updated code
 - General purpose of the feature being introduced
 - Security-sensitive files or function changes
 - Changes introduced into an already risky service

CHECKLISTS & REVIEWS

AUTHORIZATION

AUTHORIZATION REVIEW

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

AUTHORIZATION REVIEW - GRAPHQL EXAMPLE

```
class TeamMemberInput < GraphQL::Schema::InputObject
  argument :id, ID, required: false
  argument :member_id, ID, required: false
  argument :member_type, String, required: false
  argument :team_id, ID, required: false
  argument :effective_as_of, Types::DateType, required: false
  argument :effective_until, Types::DateType, required: false
end
```

```
def resolve(team_member_input)
  current_user = assert_user_present(context[:current_user])
  team_member = TypedId.find(TeamMember, team_member_input[:id])

  raise GraphQLUnauthorizedError, current_user unless
    TeamPolicy.new(current_user, team_member).update?

  update_record_with_graphql_inputs(team_member, team_member_input)

  { team_member: team_member }
end
```

AUTHORIZATION REVIEW VULNERABILITIES

- Broken Access Control - OWASP Top 10 A01:2021
 - Privilege Escalation
 - Missing Function Level Access Control
 - **Insecure Direct Object Reference**
- Sensitive Data Exposure - OWASP Top 10 A3:2017
- Mass Assignment
- Business Logic Flaws

```
func (u *Route) GetUserFromID(ctx *gin.Context, logger *zap.Logger, pg bun.IDB) {
    defer logging.LogFunctionDuration(logger)()

    paramID := ctx.Param("id")
    userID, err := uuid.Parse(paramID)
    if err != nil {
        logger.Error(fmt.Sprintf("Could not convert parameter (%s) to uuid", paramID), zap.Error(err))
        ctx.AbortWithStatusJSON(http.StatusBadRequest, gin.H{"error": "Could not parse parameter as uuid"})
        return
    }

    if userID.String() == uuid.Nil.String() {
        ctx.AbortWithStatusJSON(http.StatusNotFound, gin.H{"error": "Could not find requested user"})
        return
    }

    user := new(dao.User)
    user.ID = userID

    if selectErr := user.Select(pg); selectErr != nil || user.ID.String() == uuid.Nil.String() {
        logger.Error("Could not find user in database!", zap.Error(selectErr))
        ctx.AbortWithStatusJSON(http.StatusNotFound, gin.H{"error": "Could not find requested user"})
        return
    }

    ctx.JSON(http.StatusOK, gin.H{"data": user})
}
```

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

- https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html
- 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**

MASS-ASSIGNMENT - NODE

```
/*
  Parameters expected in the request:
  #####
  req.body.email = String
  req.body.password = String
  req.body.name = String
  req.body.age = Number

  Parameters NOT expected in the request:
  #####
  req.body.admin = Boolean
*/
var user = new User(req.body);
user.save();
```

MASS-ASSIGNMENT- RAILS 2 VS 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
```

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

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

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 CHECKLIST EXAMPLE

The screenshot shows a code editor with a sidebar on the left displaying a file tree for a Django project named 'SKEA_DJANGO'. The 'views.py' file is open in the main editor area, containing Python code for handling todo updates.

```
16
17     @login_required
18     def todo(request, todo_id):
19         if request.method == 'GET':
20             try:
21                 todo = Todo.objects.get(pk=todo_id, owner=request.user)
22                 form = TodoForm(initial={'todo_text': todo.todo_text,
23                                         'todo_date': todo.todo_date,
24                                         'completed': todo.completed})
25             except Todo.DoesNotExist:
26                 raise Http404("Todo does not exist")
27
28             logger.info("GET todo %s by %s" % (todo_id, request.user.username))
29             return render(request, "todo_update.html", {'todo': todo, 'form': form})
30
31     elif request.method == 'POST':
32         form = TodoForm(request.POST)
33         if form.is_valid():
34             try:
35                 todo = Todo.objects.get(pk=todo_id)
36             except Todo.DoesNotExist:
37                 raise Http404("Todo does not exist")
```

AUTHORIZATION EXERCISE

GENGHIS KHAN

1. **bridge_troll**
 - a. Build an authorization checklist
 - b. Test checklist items



GENGHIS KHAN - POST-MORTEM

Authorization Checklist Items

- Insecure Direct Object Reference
 - Profiles, Unlisted Events
- Privilege Escalation, Mass Assignment
 - User object updates
- CSRF Implementation Checks
- Authorization Patterns for access control
- Sensitive Data Exposure
 - What, where, when, how?

GENGHIS KHAN - POST-MORTEM

app > controllers > chapters_controller.rb

```
12  def show
13    skip_authorization
14    @chapter_events = (
15      @chapter.events.includes(:organizers, :location).published_or_visible_to(current_user) + @chapter.external_events
16    ).sort_by(&:ends_at)
17    @show_organizers = true
18
19    return unless @chapter.leader?(current_user)
20
21    @organizer_rsvps = Rsvp
22      .group(:user_id, :user_type)
23      .joins([{ event: :chapter }])
24      .includes(:user)
25      .select('user_id, user_type, count(*) as events_count')
26      .where('chapters.id' => @chapter.id, role_id: Role::ORGANIZER.id, user_type: 'User')
27  end
28
29  def new
30    authorize Chapter
31    @chapter = Chapter.new
32  end
```

GENGHIS KHAN - POST-MORTEM

bridge_troll is dependent on the authorize function for access control protection. Where else do we look now?

```
app > controllers > application_controller.rb
  Michael Glass, 3 years ago | 4 authors (Travis Grathwell and others)
1   # frozen_string_literal: true
2
3   class ApplicationController < ActionController::Base
4     protect_from_forgery
5
6     include Pundit
7     rescue_from Pundit::NotAuthorizedError, with: :user_not_authorized
8     after_action :verify_authorized, unless: :devise_controller?
9
10    before_action :configure_permitted_parameters, if: :devise_controller?
11
12  <before_action do
13    Rack::MiniProfiler.authorize_request if current_user.try(:admin?)
```

GENGHIS KHAN

Pundit - Chapter Policy

app > policies > 📄 chapter_policy.rb

Michael Glass, 4 years ago | 2 authors (Travis Grathwell and others)

```
1 # frozen_string_literal: true Michael Glass, 4 years ago • run rubocop autocorrect
2
3 class ChapterPolicy < ApplicationPolicy
4   def new?
5     user && (user.admin? || user.organization_leaderships.present?)
6   end
7
8   def update?
9     record.leader?(user) || record.organization.leader?(user)
10  end
11
12  def create?
```

AUTHENTICATION

AUTHENTICATION REVIEW

- Authentication establishes user identity
- Examine the user identification process of the application.
- Available application resources include both unidentified and identified users
- User 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 any application functionality that identifies a user in this review

AUTHENTICATION REVIEW

How does an application confirm identity?

AUTHENTICATION REVIEW VULNERABILITIES

- Identification and Authentication Failures -
OWASP Top 10 A07:2021
- User Enumeration
- Session Management Issues
- Authentication Bypass
- Brute-Force Attacks

AUTHENTICATION REVIEW CHECKLIST

- Identify all the authentication flows
 - User Login
 - User Registration
 - Forgot Password
 - **Account Updates**
- How are users identified?
 - Username, email, password, MFA token, etc.
- How is authentication handled on each application endpoint?
 - Sensitive endpoints should require authentication
- Hard-coded accounts?
- Does application allow for easily-guessed, default, or common passwords?

```
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 };  
42 
```

SPOT THE BUG

AUTHENTICATION REVIEW CHECKLIST

- How are usernames determined, what naming conventions?
 - Does login/registration allow for easy enumeration?
- **Does the application allow for account enumeration through server responses or information disclosure?**
 - login/registration/forgot password flows
- Password hashing algorithms?
- Are security policies are 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?

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?](#)

SPOT THE BUG

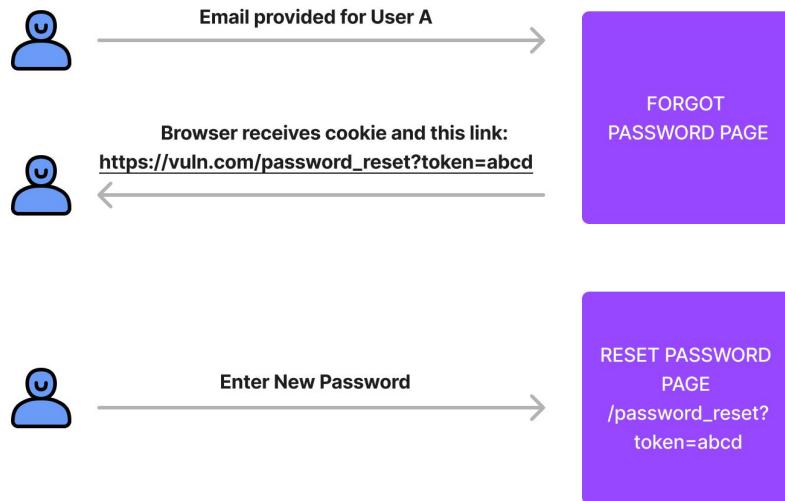
AUTHENTICATION REVIEW CHECKLIST

- 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?
- If applicable, **and in source code often it is not obvious**, 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?
- Can users elevate their initial access via mass assignment or business-logic bypasses?
- Are files and objects owned by a user removed or archived?
- **In cases where a validation email and link are required, how are the tokens generated?**

AUTHN REVIEW CHECKLIST - FLAWED CHANGE PASSWORD



Security Vulnerability:

1. Cookie is the user-supplied email address, encrypted
2. As long as the token is valid, the cookie is used to identify the user to reset

AUTHENTICATION CHECKLIST EXAMPLE

```
forms.py
models.py
tests.py
urls.py
views.py
6
skea_django
●
> __pycache__
__init__.py
settings.py
4
urls.py
wsgi.py
> venv
.gitignore
db.sqlite3
info.log
manage.py
README.md
requirements.txt

89     AUTH_PASSWORD_VALIDATORS = [
90         {
91             'NAME': 'django.contrib.auth.password_validation.
UserAttributeSimilarityValidator',
92         },
93         {
94             'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
95         },
96         {
97             'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
98         },
99         {
100             'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
101         },
102     ]
103
104     AUTH_USER_MODEL = 'intro.TodoUser'
105
106     LOGIN_REDIRECT_URL = 'home'
107     LOGOUT_REDIRECT_URL = 'home'
108     LOGIN_URL = '/intro/login/'
109
```

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?

AUTHENTICATION EXERCISE

SOCRATES

1. **bridge_troll**
 - a. Build Authentication Checklist
 - i. Abstract Identification
 - ii. How and where in this application?
 - b. Review Authentication items:
 - i. Login
 - ii. Registration
 - iii. Sessions



SOCRATES - POST-MORTEM

Authentication Items

- Authentication flows/user enumeration
 - Login, Forgot Password, Registration
- Session Handling
 - Session hijacking, bypassing authentication
- Privilege Assignment
 - Registration process

SOCRATES - POST-MORTEM

```
seth@MacBook-Pro-3 bridge_troll % rake routes
                                              Prefix Verb     URI Pattern
Controller#Action
events#index
devise/sessions#new
devise/sessions#create
devise/sessions#destroy
devise_overrides/omniauth_callbacks#passthru
devise_overrides/omniauth_callbacks#facebook
devise_overrides/omniauth_callbacks#twitter
devise_overrides/omniauth_callbacks#github
devise_overrides/omniauth_callbacks#meetup
devise_overrides/omniauth_callbacks#passthru
```

✓ BRIDGE...

- cookies_serializer...
- cors.rb
- devise.rb**
- filter_parameter_l...
- inflections.rb
- mime_types.rb
- per_form_csrf_to...
- rack_profiler.rb
- request_forgery_...
- sentry.rb
- session_store.rb
- simple_form boo...
- simple_form.rb
- wrap_parameters...

> locales

- application.rb
- boot.rb
- ! database.yml
- environment.rb
- ! newrelic.yml

```
Michael Glass, 3 years ago | 6 authors (Sarah Allen and others)
# frozen_string_literal: true Michael Glass, 3 years ago • run rubocop auto

# Use this hook to configure devise mailer, warden hooks and so forth.
# Many of these configuration options can be set straight in your model.
Devise.setup do |config|
  # The secret key used by Devise. Devise uses this key to generate
  # random tokens. Changing this key will render invalid all existing
  # confirmation, reset password and unlock tokens in the database.
  config.secret_key = ENV['DEVISE_SECRET_KEY'] || ('x' * 30)

  # ==> Mailer Configuration
  # Configure the e-mail address which will be shown in Devise::Mailer,
  # note that it will be overwritten if you use your own mailer class
  # with default "from" parameter.
  config.mailer_sender = '"Bridge Troll" <troll@railsbridge.org>'

  # Configure the class responsible to send e-mails.
  # config.mailer = 'Devise::Mailer'

  # ==> ORM configuration
  # Load and configure the ORM. Supports :active_record (default) and
  # :mongoid (bson_ext recommended) by default. Other ORMs may be
  # available as additional gems.
  require 'devise/orm/active_record'
```

EXPLORER

...

devise.rb

registrations_controller.rb X

○

> OPEN EDITORS

BRIDGE_TROLL

> .github

app

> assets

controllers

> chapters

devise_overrides

omniauth_callbacks_controller.rb

registrations_controller.rb

> events

> regions

> users

admin_pages_controller.rb

application_controller.rb

chapters_controller.rb

checkinners_controller.rb

checkins_controller.rb

courses_controller.rb

event_sessions_controller.rb

app > controllers > devise_overrides > registrations_controller.rb

Michael Glass, 3 years ago | 3 authors (Michael Glass and others)

```
1 # frozen_string_literal: true
2
3 module DeviseOverrides
4   class RegistrationsController < Devise::RegistrationsController
5     # cf. https://github.com/plataformatec/devise/wiki/How-To:-Allow-users-to-edit-their-account-without-providing-a-password
6     def update
7       @user = User.find(current_user.id)
8
9       successfully_updated =
10      if assigning_password?(@user, params)
11        @user.update(user_params)
12      elsif needs_password?(@user, params)
13        @user.update_with_password(user_params)
14      else
15        # remove the virtual current_password attribute update_without_password
16        # doesn't know how to ignore it
17        filtered_params = user_params
18        filtered_params.delete(:current_password)
19        @user.update_without_password(filtered_params)
20      end
21
22      if successfully_updated
```

AUDITING

AUDITING REVIEW VULNERABILITIES

- Cryptographic Failures - OWASP Top 10 A02:2021
- Security Logging & Monitoring Failures - OWASP Top 10 A09:2021
- Debug Messages
- Error Handling
- Information Leakage

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

```
superadminService.manageOrganizationByUserUid(userUid, org);
ImpersonateHelper.configureImpersonateWebSettings(
    request, response, ImpersonateReturnMode.USERS_MODE);
apiResponse.setOrgUuid(org.getUuid());
apiResponse.registerSuccess(String.format("Switching to user '%s' successfully", userUid));

return new ResponseEntity<>(apiResponse, HttpStatus.OK);
```

SPOT THE BUG

AUDITING REVIEW CHECKLIST

- If an exception occurs, does the application fails 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

LOGGING EXAMPLE

```
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()
```

> OPEN EDITORS

SKEA_DJANGO

- admin.py
- apps.py
- forms.py
- models.py
- tests.py
- urls.py
- views.py 6

skea_django

- > __pycache__
- __init__.py

settings.py

- urls.py 4
- wsgi.py

> venv

.gitignore

db.sqlite3

info.log

manage.py

README.md

requirements.txt

> OUTLINE

skea_django > settings.py > ...

```
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': 'simple'
150             },
151         },
152         'loggers': {
153             'django': {
154                 'handlers': ['file'],
155                 'level': 'INFO',
156                 'propagate': True,
157             },
158         }
159     }
```

AUDITING REVIEW CHECKLIST

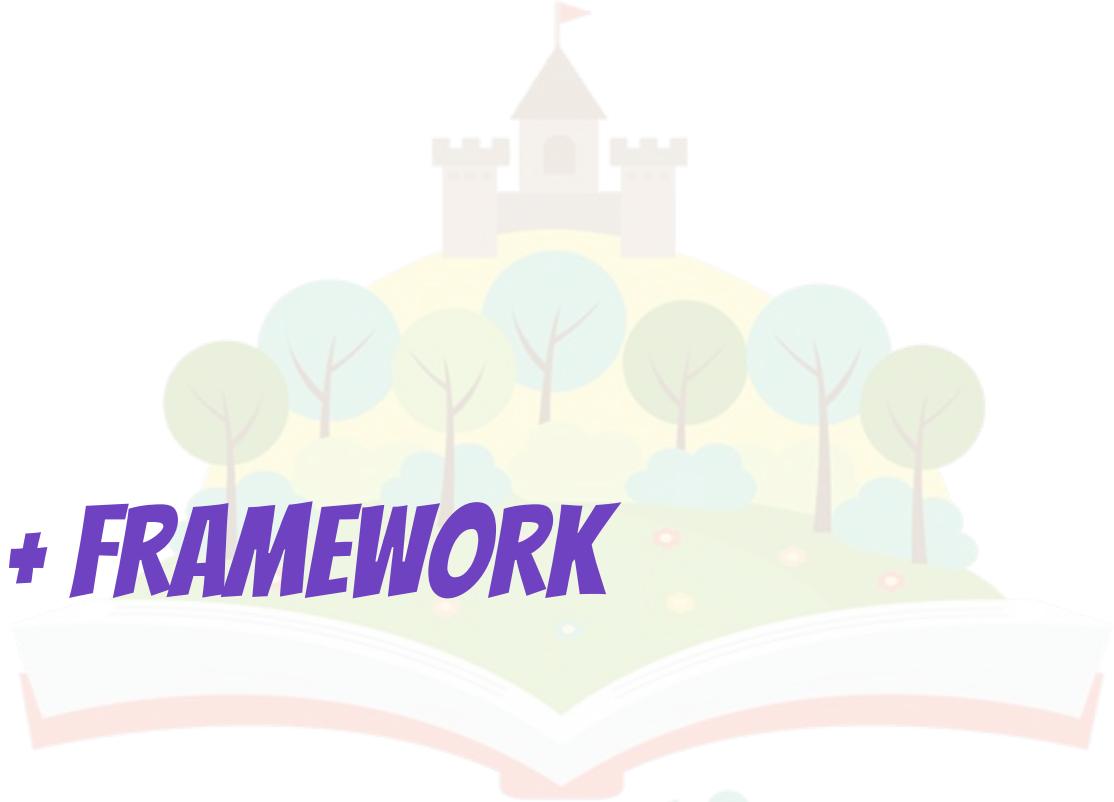
- Are unexpected errors and inputs logged?
 - Multiple login attempts, invalid logins, unauthorized access attempts
- Are log details specific enough to reconstruct events for audit purposes?
- Are logging configuration settings configurable through settings or environment variables and not hard-coded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

STORY TIME:

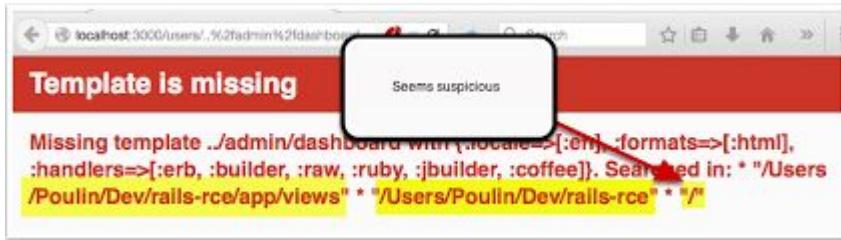
LOG INJECTION + FRAMEWORK

VULNERABILITY

Storytime



INJECTION + FRAMEWORK VULNERABILITY = RCE

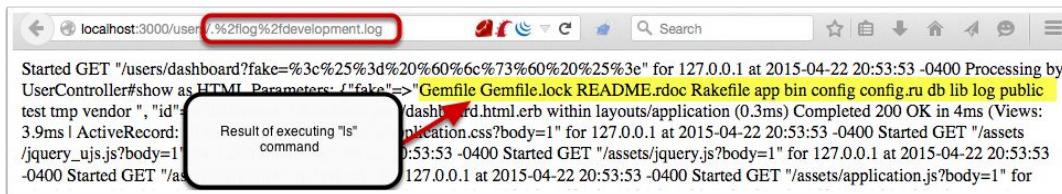


Odd place to look for erb
(html + ruby code) files

Request with ERB-style ruby code that would execute the "ls" command if executed by an ERB file



```
Started GET "/users/dashboard?fake=%3c%25%3d%20%60%6c%73%60%20%25%3e" for 127.0.0.1 at 04-22 20:53:53 -0400
Processing by UserController#show as HTML
Parameters: {"fake"=>"<= `ls` >", "id"=>"dashboard"}
Rendered user/dashboard.html.erb within layouts/application (0.3ms)
Completed 200 OK in 4ms (Views: 3.9ms | ActiveRecord: 0.0ms)
```



The final step was to have the render file pull in the log file which contains the valid ERB code and is executed as such...

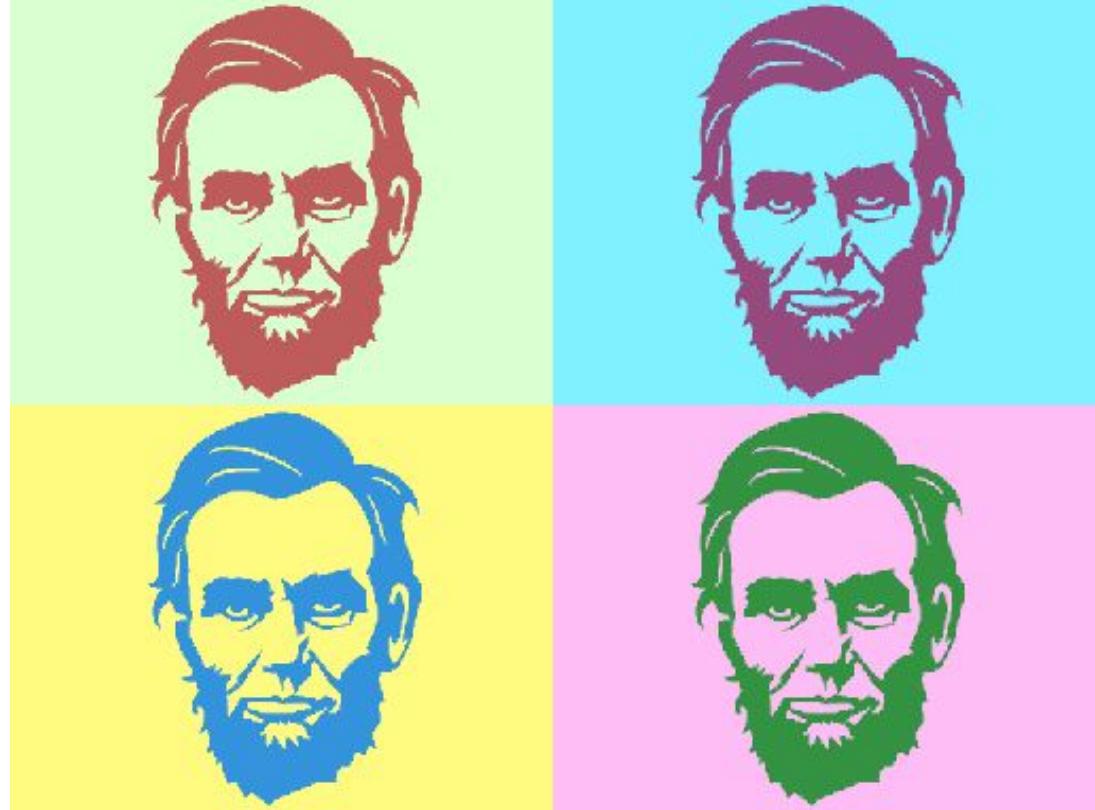
LOGGING - JAVA

```
    return new ResponseEntity<>(apiResponse, HttpStatus.OK);
} catch (final ServiceException e) {
    log.error("ERROR: " + e.getMessage());
    ErrorHelper.addErrorMessage(
        String.format("Failed to impersonate user '%s'", userUid), apiResponse, e);
}
```

AUDITING EXERCISE

ABRAHAM LINCOLN

- 1. bridge_troll**
 - a. Build/Review
Auditing Checklist
 - b. Review at least 5
checklist items



ABRAHAM LINCOLN - POST-MORTEM

Audit Items

- Security Event Logging
 - Devise, Pundit
- Debug/Error Messages

ABRAHAM LINCOLN - POST-MORTEM

The screenshot shows a code editor interface with a search results panel on the left and the file content on the right.

Search Results Panel:

- SEARCH icon
- Preserve Case (⌘P) checkbox
- logger search term
- Aa font icon
- Replace button
- AB find/replace buttons
- 50 results in 7 files - Open in editor link
- More options (three dots)
- Results list:
 - @logger.info('Done!')
 - development.rb config/en... 2 deprecation notices to the Rails I...
 - Bullet.bullet_logger = true
 - production.rb config/envir... 7
 - # Use a different logger for distri...
 - # require 'syslog/logger'
 - # config.logger = ActiveSupport::...
 - TaggedLogging.new(S... ✎ X
 - config.logger = ActiveSupport::T...
 - ActiveSupport::TaggedLogging.n...
 - config.log_formatter = "...logger..."

File Content:

```
production.rb ×  
config > environments > production.rb  
66 # when problems arise.  
67 config.log_level = :debug  
68  
69 # Prepend all log lines with the following tags.  
70 config.log_tags = [:request_id]  
71  
72 # Use a different logger for distributed setups.  
73 # require 'syslog/logger'  
74 # config.logger = ActiveSupport::TaggedLogging.new(Syslog::Logger.new 'app-name')  
75  
76 config.logger = ActiveSupport::TaggedLogging.new(Logger.new($stdout)) if ENV  
['RAILS_LOG_TO_STDOUT'].present?  
77  
78 # Use a different cache store in production.  
79 # config.cache_store = :mem_cache_store  
80  
81 # Use a real queuing backend for Active Job (and separate queues per environment)  
82 # config.active_job.queue_adapter = :resque  
83 # config.active_job.queue_name_prefix = "bridgetroll_#{Rails.env}"  
84 config.action_mailer.perform_caching = false
```

logger

Aa ab *

1



50 results - 7 files

.rspec_parallel:

```
1 --format progress  
2: --format ParallelTests::RSpec::RuntimeLogger --out tmp/parallel_runtime_rspec.log
```

app/services/account_merger.rb:

```
15
16:   Rails.logger.info(<<-USER_PROMPT.strip_heredoc)
17
```

app/services/database_anonymizer.rb:

```
4 class DatabaseAnonymizer
5   def initialize(logger = nil)
6     require 'faker'
7
8   @logger = logger || Logger.new($stdout)
9   @logger.formatter = proc do |_severity, _datetime, _progname, msg|
10    "#{msg}\n"
```

```
14     def anonymize_database
15         @logger.info('Anonymizing users...')
16         User.find_each { |u| anonymize_user(u) }
17
```

INJECTION

INJECTION



- Causes:
 - **Confusion between system code, and user-supplied**
- Typical prevention
 - Input Validation
 - Output Encoding
- Types
 - SQL Injection
 - HTML Injection (XSS)
 - LDAP, XML, Command ...

INJECTION VULNERABILITIES

- Injection - OWASP Top 10 A03:2021
- XML External Entities (XXE)
- Cross-Site Scripting (XSS)
- Redirects
- Server Side Request Forgery (SSRF) - OWASP Top 10 A10:2021

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 input is 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)?
- **Verify how security mechanisms performing validation behave via docs/review**

```
if (FormsAuthenticationHelper.IsRelativeUrl(redirectUrl))
{
    this.Response.Redirect(redirectUrl, true);
}
else
```

```
/// <summary>
/// Tests the incoming url to see if it relative.
/// </summary>
/// <param name="url"></param>
/// <returns></returns>
public static bool IsRelativeUrl(string url)
{
    Uri result;
    return Uri.TryCreate(url, UriKind.Relative, out result);
}
```

SPOT THE BUG

Uri.TryCreate Method

[Reference](#)[Feedback](#)

Definition

Namespace: [System](#)Assembly: [System.Runtime.dll](#)

Creates a new [Uri](#). Does not throw an exception if the [Uri](#) cannot be created.

Overloads

TryCreate(Uri, Uri, Uri)	Creates a new Uri using the specified base and relative Uri instances.
TryCreate(String, UriKind, Uri)	Creates a new Uri using the specified String instance and a UriKind .
TryCreate(String, UriCreationOptions, Uri)	Creates a new Uri using the specified String instance and UriCreationOptions .
TryCreate(Uri, String, Uri)	Creates a new Uri using the specified base and relative String instances.

INPUT VALIDATION - CHECKLIST

- 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 DenyLists or AcceptLists?
- 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)?

INPUT VALIDATION - CHECKLIST

- 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?

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)**

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])
```

SQL INJECTION - NODEJS

```
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

- What output encoding libraries are used? (updated?)
- 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?
- **What template tags are safe/unsafe? (html)**

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        </td>
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 CHECKLIST EXAMPLE

views.py 6 × urls.py 4 settings.py



intro > views.py > index

```
17     @login_required
18     def todo(request, todo_id):
19         if request.method == 'GET':
20             try:
21                 todo = Todo.objects.get(pk=todo_id, owner=request.user)
22                 form = TodoForm(initial={'todo_text':todo.todo_text,
23                                         'todo_date':todo.todo_date,
24                                         'completed':todo.completed})
25             except Todo.DoesNotExist:
26                 raise Http404("Todo does not exist")
27
28             logger.info("GET todo %s by %s" % (todo_id, request.user.username))
29             return render(request, "todo_update.html", {'todo': todo, 'form': form} )
30
31         elif request.method == 'POST':
32             form = TodoForm(request.POST)
33             if form.is_valid():
34                 try:
```



> OPEN EDITORS

✓ SKEA_DJANGO

- ↳ todo_update.html
- ↳ todo.html
- ↳ todos.html
- ✚ __init__.py
- ✚ admin.py
- ✚ apps.py
- ✚ forms.py
- ✚ models.py
- ✚ tests.py
- ✚ urls.py
- ✚ views.py

✓ skea_django

- > __pycache__
- ✚ __init__.py
- ✚ settings.py
- ✚ urls.py

intro > templates > ↳ todo_update.html > ...

You, 5 years ago | 1 author (You)

```
1  {% extends 'base.html' %} 
2  {% block content %} 
3  <div class="row"> 
4      <div class="col-3"></div> 
5      <div class="col-6"> 
6          {% if todo %} 
7              <form action="/intro/{{todo.id}}/" method="post"> 
8                  {% csrf_token %} 
9                  {{ form.as_p }} 
10                 <button type="submit" class="btn btn-primary">Update</button> 
11             </form> 
12         {% endif %} 
13     </div> 
14     <div class="col-3"></div> 
15 </div> 
16 {% endblock %} 
17 |
```

INJECTION EXERCISE

BILLY THE KID

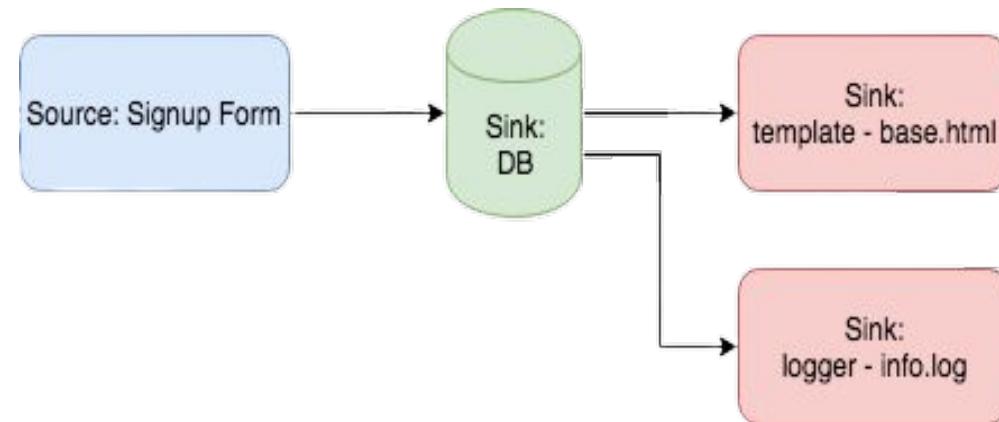
1. **bridge_troll**
 - a. Build Injection Checklist
 - b. Trace at least 5 sources to sinks



BILLY THE KID - POST-MORTEM

WebApp with a database Backend:

- a. SQL Injection (database storage)
- b. XSS (Stored/Reflected)
- c. Log Forging/Injection



> OPEN EDITORS

BRIDGE_TROLL



rsvps

<> _class_levels.html.erb

<> _dietary_restrictions.html.erb

<> _form.html.erb

<> edit.html.erb

<> new.html.erb

<> quick_destroy_confirm.html.e...

> shared

> static_pages

> surveys

users

<> index.html.erb

> volunteers

> bin

config

> environments

initializers

activerecord_concat.rb

application_controller_render...

assets.rb

app > views > rsvps > <> _class_levels.html.erb

Jennifer Konikowski, 6 years ago | 3 authors (Travis Grathwell and others)

```
1  <div class="level_buttons">
2    <% levels.each do |level| %>
3      <div class="class_level_line">
4        <label class='class_level' >
5          <% if defined?(f) %>
6            <%= f.radio_button :class_level, level.num %>
7          <% end %>
8          <span class="<%= "label btn-#{level.color}" %>"><%= level.color.capitalize %><
9            span>
10           <%= level.title %>
11           <ul>
12             <% level.description.each do |info| %>
13               <li><%= info.html_safe %></li>
14             <% end %>      Lillie Chilen, 9 years ago • Move class level descriptions in
15           </ul>
16           </label>
17         <% end %>
18         <% if show_no_preference %>
19           <div class="class_level_line">
20             <label>
21               <% if defined?(f) %>
22                 <%= f.radio_button :class_level, 0 %>
23               <% end %>
24               <span class="label btn-gray">Any</span>
25               No preference
26             </label>
27           </div>
28         <% end %>
29       </div>
30     <% end %>
31   </div>
```

html_safe

Aa ab *

Replace AB ⌘ab

...

15 results in 10 files - [Open in editor](#)

- ↳ index.html.erb app/views/... 1
event.course.levels.to_json.html...
- ↳ show.html.erb app/views/... 1
<%= @event.course.d... ⌘b X
- ↳ new.html.erb app/views/ev... 1
end.to_json.html_safe %>;
- ↳ organize_sections.html.e... 4
rsvps_with_checkins.to_json.ht...
<%= @event.sections.to_json.ht...
event.event_sessions.to_json.ht...
end.to_json.html_safe
- ↳ from_organizer.html.erb... 1
)gsub(/(\r?\n|
)/) html_sa

app > views > events > ↳ show.html.erb

```
64      <% end %>
65    </div>
66    <% end %>
67  </div>
68  <% if @event.course %>
69    <div id="course">
70      <h2>Course</h2>
71      <p>
72        <%= @event.course.description.html_safe %>
73      </p>
74      <p>
75        <%= link_to "Click here for more information about class levels in this
76        course!", levels_event_path(@event) %>
77      </p>
78    </div> Michael Dreyfus, 10 years ago • added course selection for organizer
79    <% end %>
80    <div class="details">
81      <%= simple_format_with_html(@event.details) %>
82      <% if current_user %>
83        <% if @event.volunteer?(current_user) || @can_edit %>
84          <h3>Volunteer Details</h3>
85        <% else %>
86          <h3>Organizer Details</h3>
87        <% end %>
88      <% end %>
89    </div>
90  <% end %>
91  <div class="links">
92    <%= link_to "Edit this event", edit_event_path(@event) %>
93    <%= link_to "Delete this event", event_path(@event), method: :delete %>
94  </div>
95<% end %>
```

```
2
3  class OrganizationsController < ApplicationController
4    before_action :authenticate_user!, only: [:download_subscriptions]
5
6    def index
7      skip_authorization
8      @organizations = Organization.all
9      chapter_last_event_ids = Event
10         .published
11         .select('max(id) as event_id, chapter_id')
12         .group(:chapter_id)
13         .map(&:event_id)
14
15      @chapter_locations = Event
16        .includes(:location, :chapter)
17        .where(id: chapter_last_event_ids)
18        .map { |e| ChapterEventLocation.new(e) }
19
20    def show
21      skip_authorization
22      @organization = Organization.find(params[:id])
23
24
```

VULNERABLE?

Michael Glass, 3 years ago

CRYPTOGRAPHIC ANALYSIS

CRYPTOGRAPHIC ANALYSIS

- Analyze code for encryption flaws, including:
 - outdated protocols
 - custom-developed algorithms
 - weak encryption
 - Misuse
 - Signature mechanisms
- **Automated tools will uncover some of this, including**
 - **Use of older hashing algorithms (ex: 3DES, SHA-1)**

CRYPTOGRAPHIC ANALYSIS

- 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:
 - base64, encrypt, decrypt, secure, md5, sha1, etc.

CRYPTOGRAPHIC ANALYSIS VULNERABILITIES

- Cryptographic Failures - OWASP Top 10 A02:2021
- Lack of Encryption
- Improper Encryption
- Insecure Token Generation/Randomness/Validation

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 CHECKLIST - JWT ATTACKS

<https://portswigger.net/web-security/jwt>

JSON Web Tokens (JWTs):

- Header/payload hashed & verified
- Flexible by design
- Self-contained information
- Vital that the signature is correctly verified

CRYPTOGRAPHIC ANALYSIS CHECKLIST - JWT ATTACKS

Common attacks:

- Incorrect verification func invoked **decode()** vs. **verify()**
- **alg** parameter set to **nOnE**
- Brute force key used to generate signature
- JWT Header Parameter Injection:

- **jwk** (JSON Web Key) - Provides an embedded JSON object representing the key.
- **jku** (JSON Web Key Set URL) - Provides a URL from which servers can fetch a set of keys containing the correct key.
- **kid** (Key ID) - Provides an ID that servers can use to identify the correct key in cases where there are multiple keys to choose from. Depending on the format of the key, this may have a matching **kid** parameter.

CRYPTOGRAPHIC ANALYSIS CHECKLIST - JWT

- JSON Web Tokens (JWTs)
 - Tokens should be validated during each interaction
 - Cryptographic Signature Algorithm - is it None?
 - Use of default or weak secret keys
 - Algorithm confusion
 - How does the code handle unrecognized or unsupported signature algorithms
 - What JWT library is used? What are the defaults and recommendations to using JWTs securely?

CRYPTOGRAPHIC REVIEW EXAMPLE

skeia_django >  settings.py > ...

```
85
86     # Password validation
87     # https://docs.djangoproject.com/en/2.1/ref/settings/#auth-passwordValidators
88
89 AUTH_PASSWORD_VALIDATORS = [
90     {
91         'NAME': 'django.contrib.auth.password_validation.
92             UserAttributeSimilarityValidator',
93     },
94     {
95         'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
96     },
97     {
98         'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
99     },
100    {
101        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
102    },
```



CRYPTOGRAPHIC REVIEW EXAMPLE

For verifying passwords, Django will find the hasher in the list that matches the algorithm name in the stored password. If a stored password names an algorithm not found in **PASSWORD_HASHERS**, trying to verify it will raise **ValueError**.

The default for **PASSWORD_HASHERS** is:

```
PASSWORD_HASHERS = [  
    "django.contrib.auth.hashers.PBKDF2PasswordHasher",  
    "django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher",  
    "django.contrib.auth.hashers.Argon2PasswordHasher",  
    "django.contrib.auth.hashers.BCryptSHA256PasswordHasher",  
    "django.contrib.auth.hashers.ScryptPasswordHasher",  
]
```

This means that Django will use **PBKDF2** to store all passwords but will support checking passwords stored with PBKDF2SHA1, **argon2**, and **bcrypt**.

CRYPTOGRAPHIC ANALYSIS EXERCISE

BEETHOVEN

1. bridge_troll

- a. Build Crypto Checklist
- b. How are passwords stored/tokens generated



BEETHOVEN - POST-MORTEM

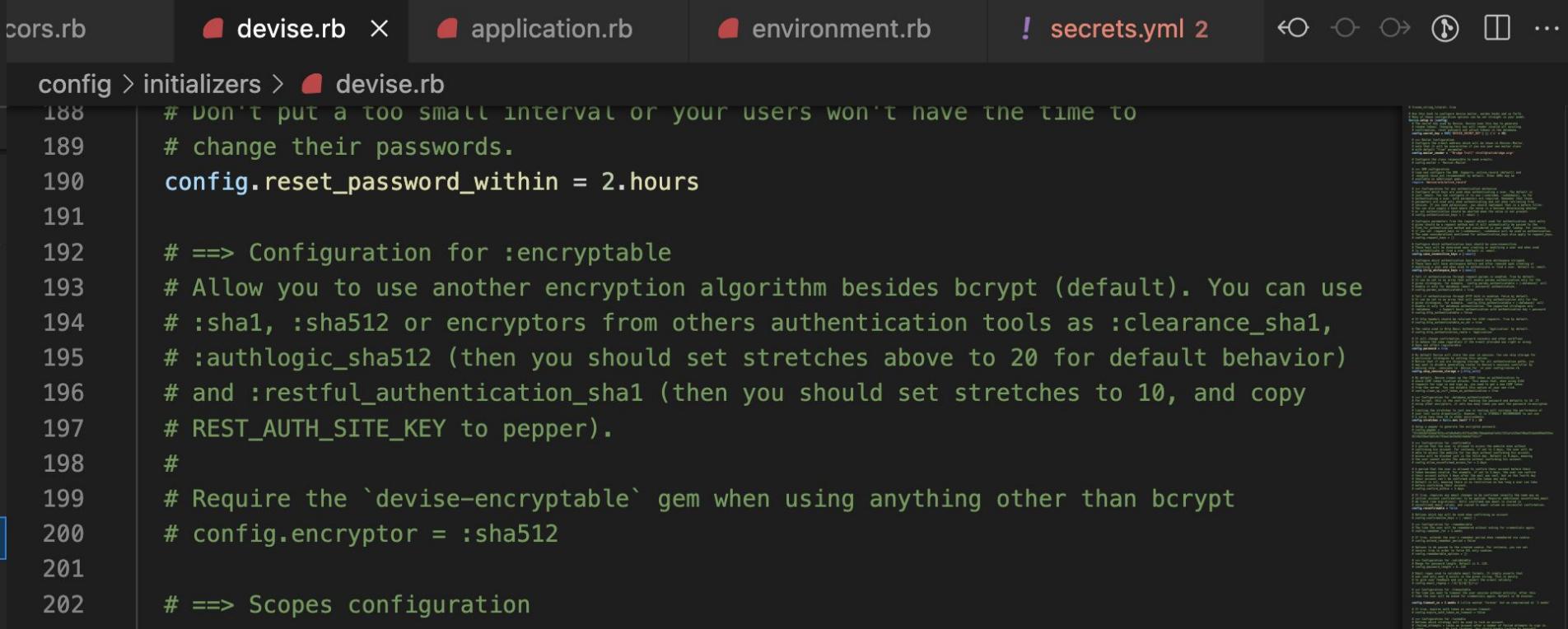
- Possible cryptographic issues (outside of encryption routines):
 - a. Hardcoded values stored in source
 - b. Same key used for dev/staging/production
 - c. Password hashing

BEETHOVEN - POST-MORTEM

```
! newrelic.yml
puma.rb
routes.rb
! secrets.yml          2
spring.rb
! storage.yml
> db
> doc
> lib
> log
> nix
> node_ ~/Desktop/SKEA/bridge_troll/nix
> public
> script
> spec
> tmp
> vendor

12
13   development:
14     secret_key_base:
15       fa074d9121f177b2b88ba095ed5e524f55510c1e0c8ab5ee38a71c8893344e682e82f3eb6a
16       b1f41c75d2144fbe99b180cab667e2684362378af
17
18   test:
19     secret_key_base:
20       955d16363ae7f73ee24f9b36be5c61410c88d869d27ddf8986d771e83dd5ebd65ab511e2dd
21       0e66447823885ba35e8414776084af1dc41db2840
22
23   # Do not keep production secrets in the repository,
24   # instead read values from the environment.
25   # keep 'staging' as close to production as possible
26   staging:
27     secret_key_base: <%= ENV["SECRET_TOKEN"] %>
28
29   production:
30     secret_key_base: <%= ENV["SECRET_TOKEN"] %>
```

BEETHOVEN - POST-MORTEM



The screenshot shows a code editor interface with several tabs at the top: cors.rb, devise.rb (selected), application.rb, environment.rb, and secrets.yml 2. The current view is on the devise.rb file under the config/initializers directory. The code is as follows:

```
config > initializers > devise.rb
188 # Don't put a too small interval or your users won't have the time to
189 # change their passwords.
190 config.reset_password_within = 2.hours
191
192 # ==> Configuration for :encryptable
193 # Allow you to use another encryption algorithm besides bcrypt (default). You can use
194 # :sha1, :sha512 or encryptors from others authentication tools as :clearance_sha1,
195 # :authlogic_sha512 (then you should set stretches above to 20 for default behavior)
196 # and :restful_authentication_sha1 (then you should set stretches to 10, and copy
197 # REST_AUTH_SITE_KEY to pepper).
198 #
199 # Require the `devise-encryptable` gem when using anything other than bcrypt
200 # config.encryptor = :sha512
201
202 # ==> Scopes configuration
```

A vertical scrollbar is visible on the right side of the code editor.

CONFIGURATION REVIEW

CONFIGURATION REVIEW VULNERABILITIES

- Security Misconfiguration - OWASP Top 10 A05:2021
 - Insecure defaults
 - Incomplete configurations
 - Open cloud storage
- Vulnerable and Outdated Components - OWASP Top 10 A06:2021
 - Any dependencies, libraries, services

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

Stopping

By default, the  requires an administrator password to initiate a server shutdown. The default (and permanent) administrator account is ADMIN with an initial default password of ADMIN.

SPOT THE BUG

CONFIGURATION REVIEW - CHECKLIST

- Are any endpoints enabled through configurations properly protected with authentication and authorization?
- Are security protections implemented in framework properly configured?
- **If an environment variable is used to specify things like keys, tokens, webhook endpoints, etc. ensure the app fails if not present**

CONFIGURATION REVIEW - CHECKLIST

- Does the target language and framework version have any known security issues?
- Are configuration-controlled security headers implemented according to recommended best practices?

CONFIGURATION REVIEW EXAMPLE

skeia_django > 📄 settings.py > ...

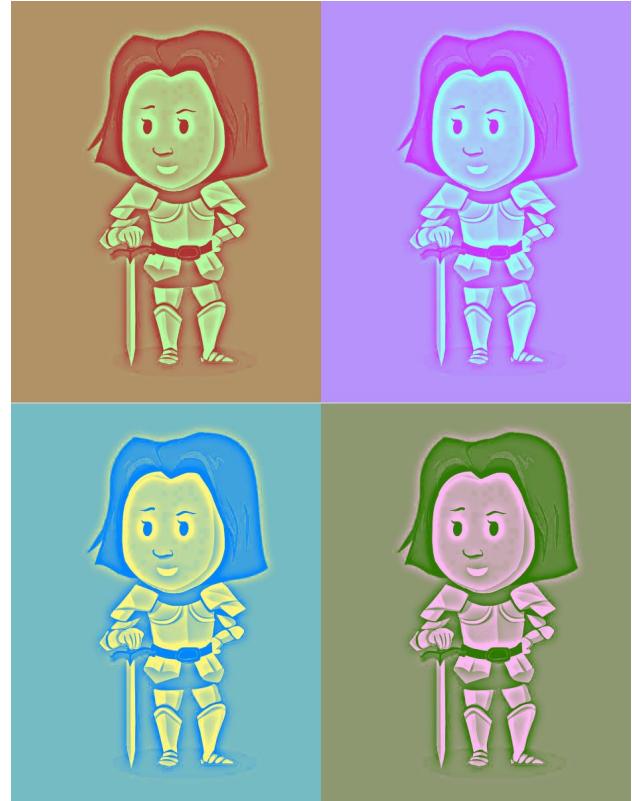
```
12
13     import os
14
15     # Build paths inside the project like this: os.path.join(BASE_DIR, ...)
16     BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
17
18
19     # Quick-start development settings - unsuitable for production
20     # See https://docs.djangoproject.com/en/2.1/howto/deployment/checklist/
21
22     # SECURITY WARNING: keep the secret key used in production secret!
23     SECRET_KEY = '*@ty00=62-recu@lsczr-o0(%y97c(ll1gnv9pu7o^)h%#e17d'
24
25     # SECURITY WARNING: don't run with debug turned on in production!
26     DEBUG = True
27
28     ALLOWED_HOSTS = []
29
```



CONFIGURATION ANALYSIS EXERCISE

JOAN OF ARC

1. **bridge_troll**
 - a. Build Config Checklist
 - b. Run/review brakeman



JOAN OF ARC - POSTMORTEM

Possible Vulnerabilities

- Out of date third party libraries/dependencies
- Insecure Configurations of 3rd Party Components
 - Devise
 - Pundit
 - ???

Category: Unmaintained Dependency

Check: EOLRails

Message: Support for Rails 5.2.5 ended on 2022-06-01

File: Gemfile.lock

Line: 262

Confidence: Weak

Category: Cross-Site Scripting

Check: SanitizeConfigCve

Message: rails-html-sanitizer 1.3.0 is vulnerable to cross-site scripting when `select` and `style` tags are allowed (CVE-2022-32209). Upgrade to 1.4.3 or newer

File: Gemfile.lock

Line: 287

Confidence: Weak

Category: Cross-Site Scripting

Check: LinkToHref

Message: Potentially unsafe model attribute in `link_to` href

Code: link_to("Code of Conduct", Event.find(params[:id]).code_of_conduct_url, :target => :blank, :id => "coc_url")

File: app/views/events/_form.html.erb

Line: 369

Confidence: Weak

Category: Dynamic Render Path

Check: Render

Message: Render path contains parameter value

Code: render(action => EventEditor.new(current_user, params).create.render, {})

File: app/controllers/events_controller.rb

Line: 91

Check: SanitizeConfigCve
Message: rails-html-sanitizer 1.3.0 is vulnerable to cross-site scripting when `select` and `style` tags are allowed (CVE-2022-32209). Upgrade to 1.4.3 or newer
File: Gemfile.lock
Line: 287

Confidence: Weak
Category: Cross-Site Scripting
Check: LinkToHref
Message: Potentially unsafe model attribute in `link_to` href
Code: link_to("Code of Conduct", Event.find(params[:id]).code_of_conduct_url, :target => :blank, :id => "coc_url")
File: app/views/events/_form.html.erb
Line: 369

Confidence: Weak
Category: Dynamic Render Path
Check: Render
Message: Render path contains parameter value
Code: render(action => EventEditor.new(current_user, params).create.render, {})
File: app/controllers/events_controller.rb
Line: 91

Confidence: Weak
Category: SQL Injection
Check: SQL
Message: Possible SQL injection
Code: relation.select(:id, :first_name, :last_name).where("#{relation.connection.concat("lower(first_name)", "' '", "lower(last_name)")} like ?", "%#{query.downcase}%")
File: app/services/user_searcher.rb
Line: 13

(END)

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
```

SECURITY MISCONFIGURATION - .NET

```
<sessionState mode="Off"  
stateConnectionString="tcpip=localhost:42424"  
sqlConnectionString="data source=localhost;user  
id=sa;password=" cookieless="false" timeout="20"/>
```

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 & RETESTING

REPORTING & 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

GROUP PROJECTS

THANK YOU!!!!



CONTACT

ABSOLUTE APPSEC - ABSOLUTEAPPSEC.COM

@ABSOLUTEAPPSEC

KEN - @CKTRICKY - KEN@ABSOLUTEAPPSEC.COM

SETH - @SETHLAW - SETH@ABSOLUTEAPPSEC.COM