The Absolute AppSec Secure Code Review Framework

AppSec Day - 11/01/2019

TLDR; Seth & Ken’s Excellent Adventure (in Secure Code Review)

Introductions

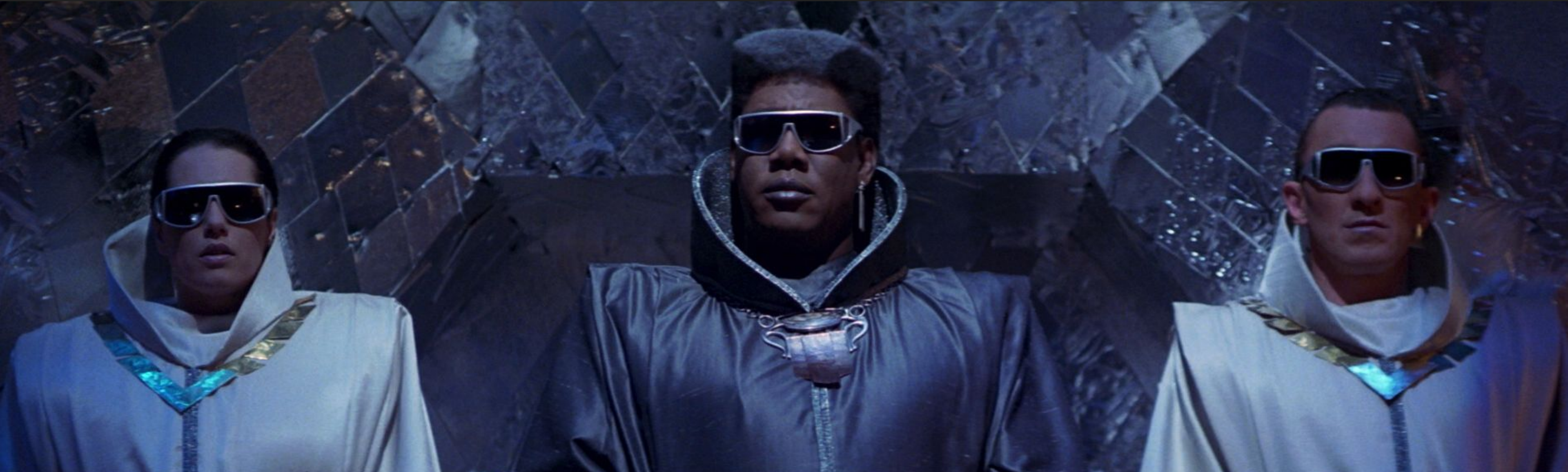
- Seth Law
- Application Security Consultant
- @sethlaw

ABSOLUTE AppSec

Redpoint Security

Overview

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



Philosophy

- Increase the likelihood of success. Not “find every bug imaginable”.
- Successful is defined as:
    - Focused
    - Comprehensive
    - Timely
    - Easily consumed, well drafted reporting
    - Detailed notes
- Language Agnostic Principles
- Repeatable and systematic approach to finding bugs

Focus on what's important

- What happens when you run into 3.2 million lines of code?
- And only have two weeks to look at it?

Approach

- Ideal World - As much time as is possible
- OMG, please no - No time at all
- Real World - Set time and scoped at least semi “okay”

Approach

NONE OF THESE REVIEWS ARE JUST “RUN A SCANNER AND ANALYZE THE RESULTS”

Absolute AppSec Code Review Methodology

Overview

Absolute AppSec Secure Code Review Methodology

1. Application Overview & Risk Assessment
2. Information Gathering
3. Checklist Creation
4. Perform Reviews

Checklists/Reviews

- Authorization
- Authentication
- Auditing (Logging)
- Injection
- Cryptographic
- Configuration
- Reporting

The Circle-K Framework

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

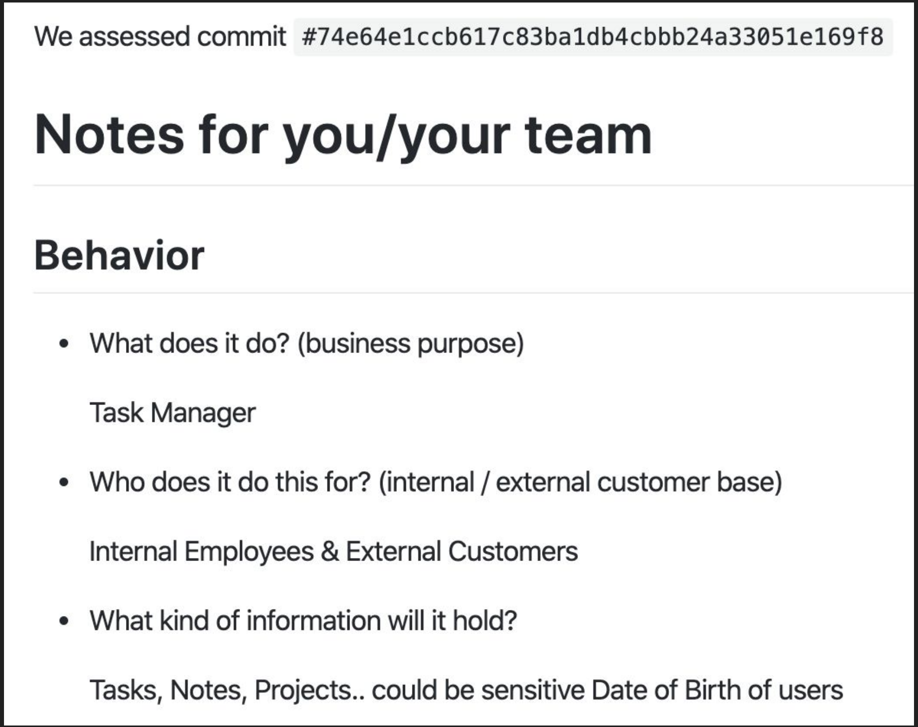
STRANGE THINGS ARE AFOOT AT THE CIRCLE K.
BILL AND TED'S EXCELLENT ADVENTURE

General Code Review 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 - Example



We assessed commit `#74e64e1ccb617c83ba1db4cbbb24a33051e169f8`

Notes for you/your team

Behavior

- What does it do? (business purpose)

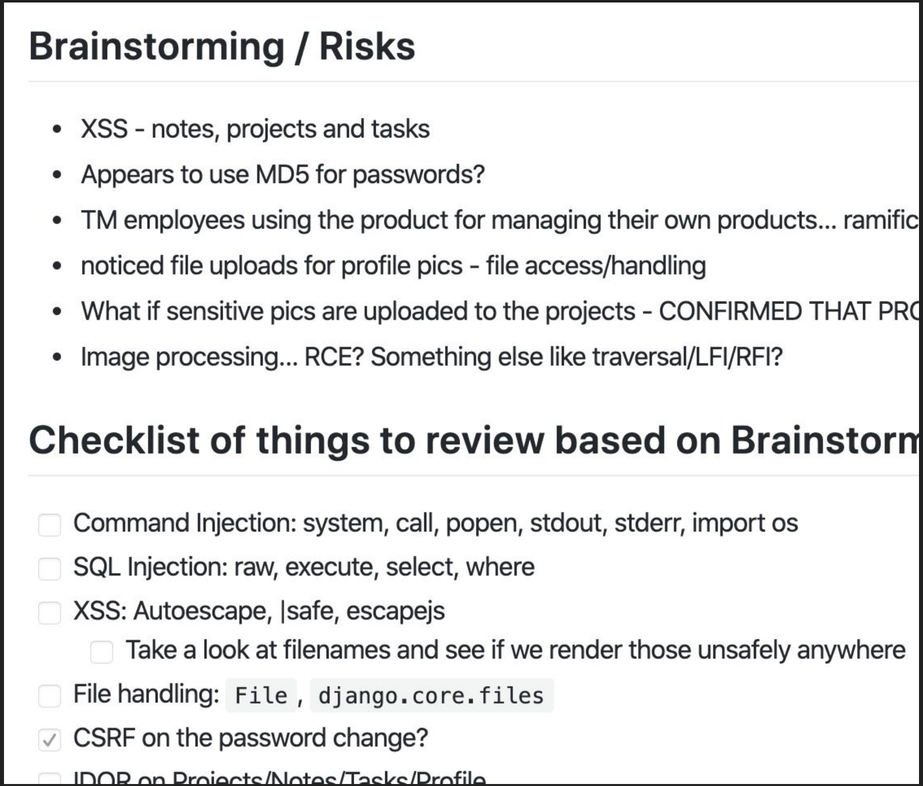
  Task Manager

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

  Internal Employees & External Customers

- What kind of information will it hold?

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



Brainstorming / Risks

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

Checklist of things to review based on Brainstorm

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

Application Overview & Risk Assessment

Application Overview & Risk Assessment

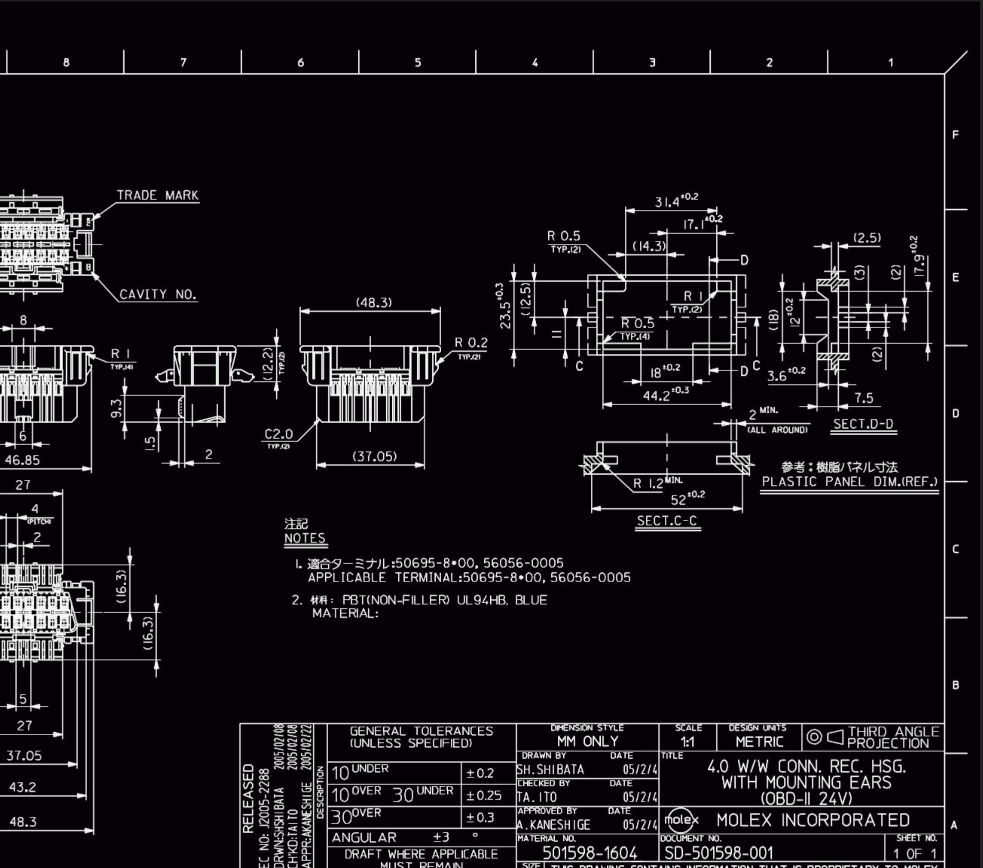
Build a portrait of the application

- Behavior Profile
- Technology Stack
- App Archeology

Behavior Profile

- 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/App Archeology



TRADE MARK
CAVITY NO.
NOTES
APPLICABLE TERMINAL:50695-8*00, 56056-0005
MATERIAL: PBT(NON-FILLER) UL94HB, BLUE
SECT.D-D
SECT.C-C
PLASTIC PANEL DIM.(REF.)
4.0 W/W CONN. REC. HSG. WITH MOUNTING EARS (OBD-II 24V)
MOLEX INCORPORATED
501598-1604
SD-501598-001

- Framework & Language
- 3rd party components
- Datastore
- How does the application accomplish its purpose?

Risk Assessment

- (mini) Threat Model / Prioritize Risks
- Utilize gain knowledge to identify risks, threats, and figure out where to spend time

Application Overview & Risk Assessment

Napoleon Walkthrough

Let’s figure out the following: determine:

- Tech stack
- Business purpose
- Application Risk
- Anything else you can dig up?

Information Gathering

Information Gathering

STILL TRYING TO UNDERSTAND THE APP

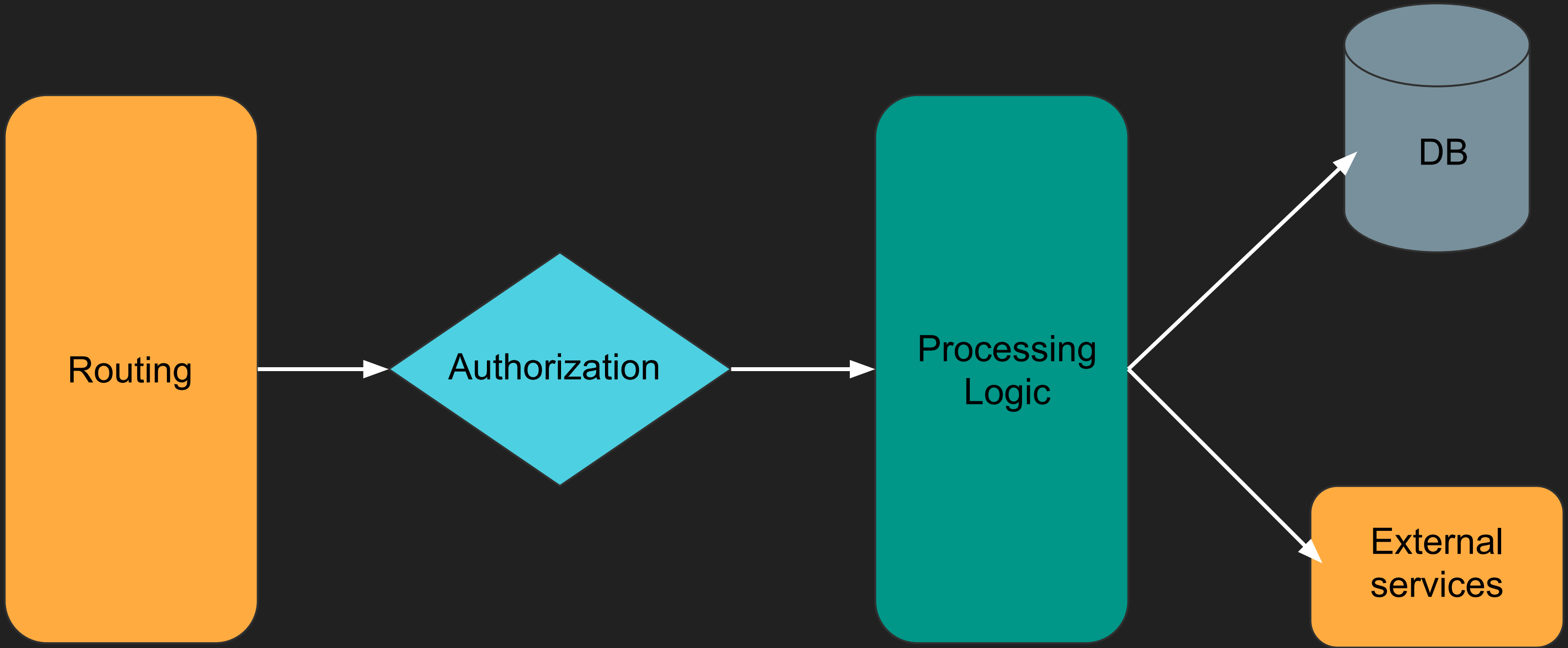
Information Gathering - Keep on Harvesting

1. Create Application Map
2. Identify Authorization Functions

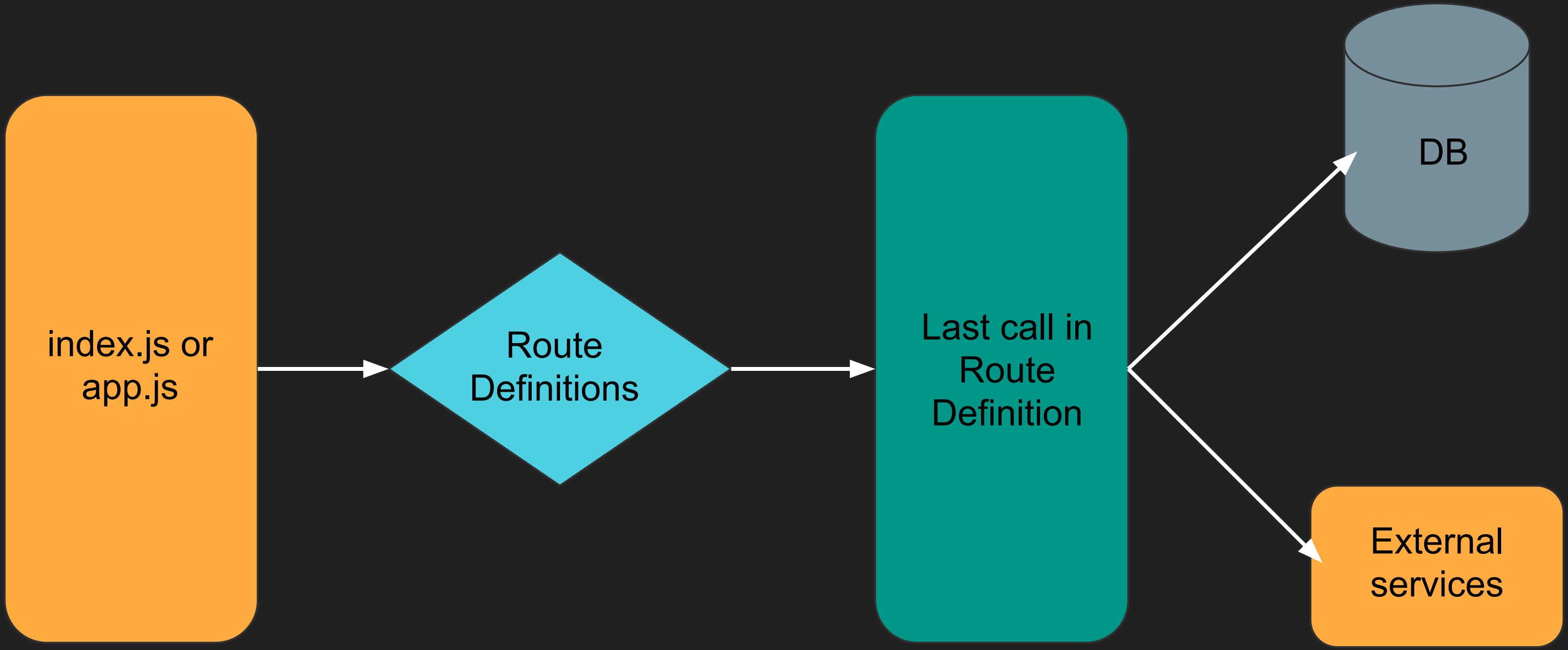
Mapping

Information Gathering - Create a Map

- Identify endpoints:
  - Rails = config/routes.rb - `rake routes`
  - Django= urls.py - `manage show_urls`
  - Node.js = index.js
  - Java Spring = *Controller.java
  - Android = AndroidManifest.xml
- Endpoints typically have at least three qualities
  - Authorization Filter
  - Logic processing
  - Datastore access



Typical Application Flow



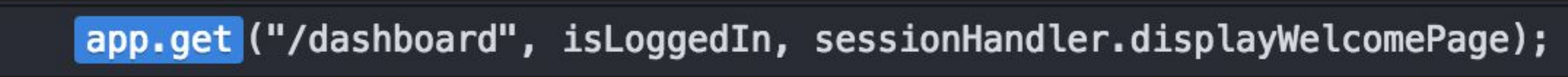
Node.js Application Flow

Node.js/Express - Map

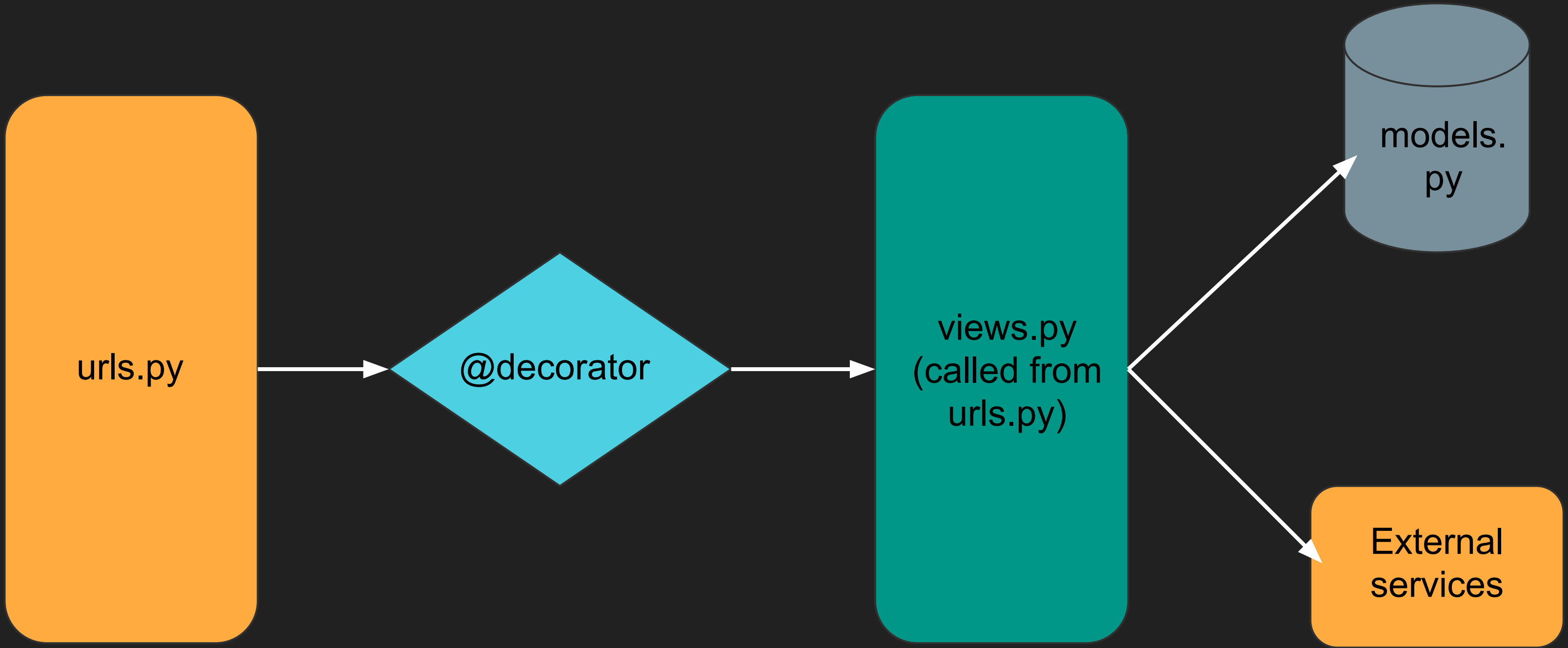
- Formula is basic, searching for:
  - (app/router/*).**get**
  - (app/router/*).**post**
  - (app/router/*).**delete**
- Annotate which of these is actually using middleware
- Create a checklist for tracing

Node.js/Express - Map

Take a closer look

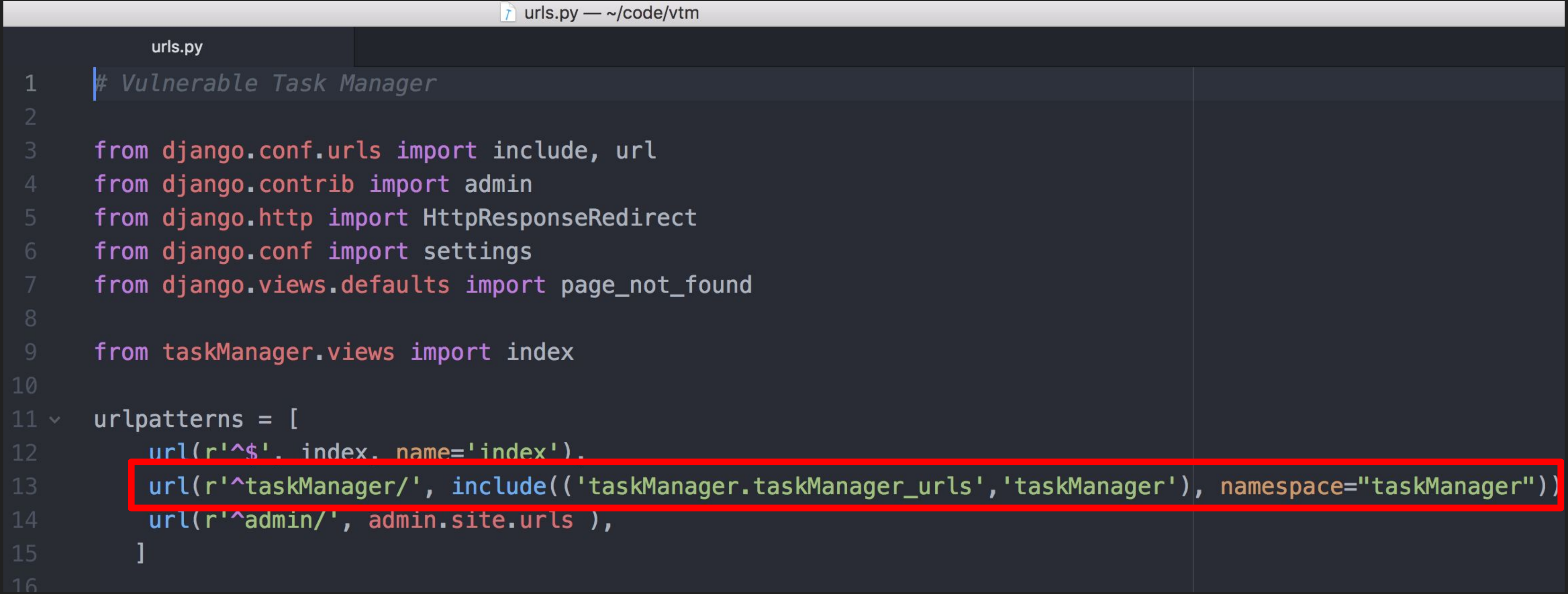


```
app.get("/dashboard", isLoggedIn, sessionHandler.displayWelcomePage);
```



Django Application Flow

Django - Map

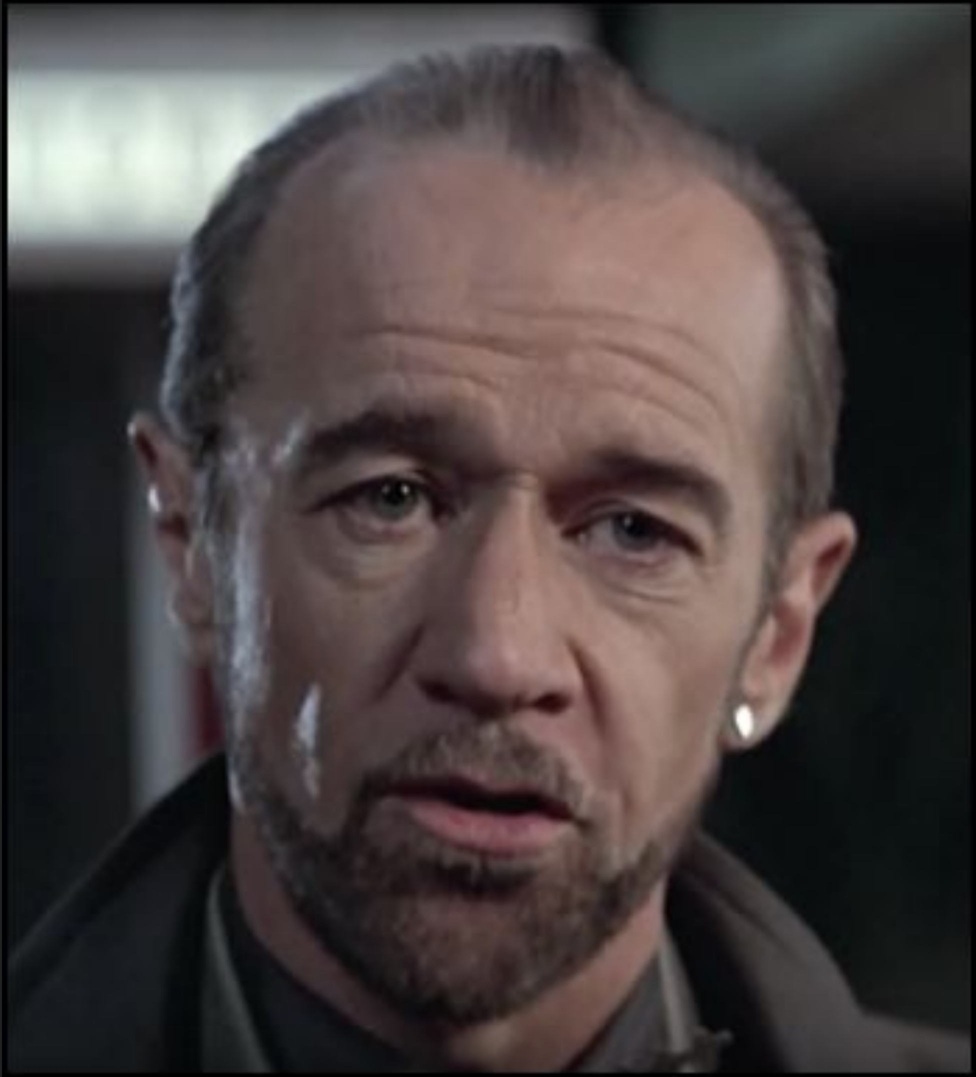


```python
# Vulnerable Task Manager

from django.conf.urls import include, url
from django.contrib import admin
from django.http import HttpResponseRedirect
from django.conf import settings
from django.views.defaults import page_not_found

from taskManager.views import index

urlpatterns = [
    url(r'^$', index, name='index'),
    url(r'^taskManager/', include(('taskManager.taskManager_urls','taskManager'), namespace="taskManager")),
    url(r'^admin/', admin.site.urls ),
   ]
```

Mapping

Exercise Rufus

What are the endpoints for the target application?



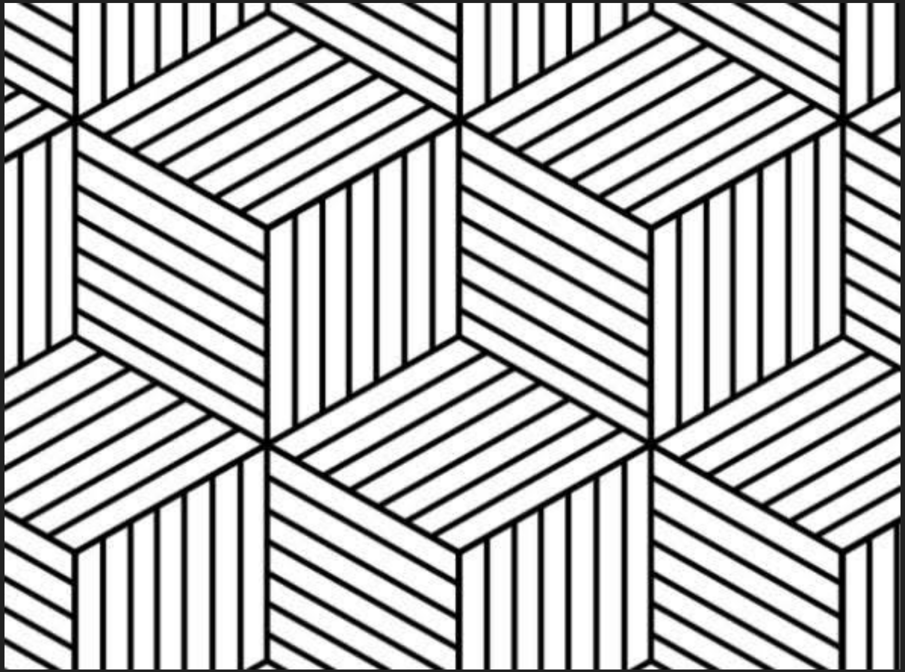
Authorization Functions

Information Gathering - Authorization Functions

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

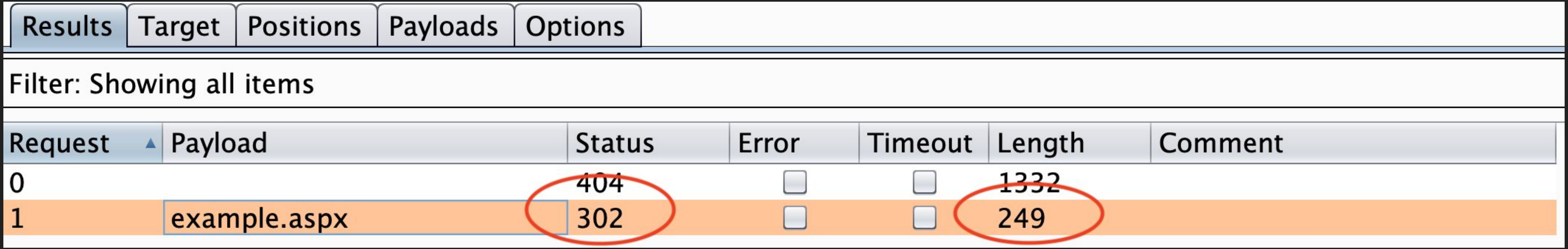
Information Gathering - Authorization Functions

- 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?



Redirection & Authorization Issue in .NET

Normal

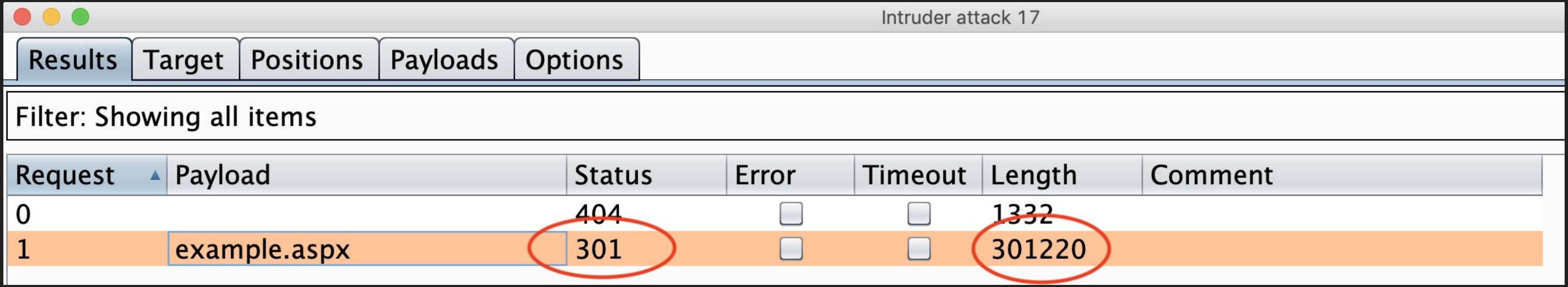


Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		404	☐	☐	1332	
1	example.aspx	302	☐	☐	249	

Redirection & Authorization Issue in .NET

Definitely **NOT** Normal

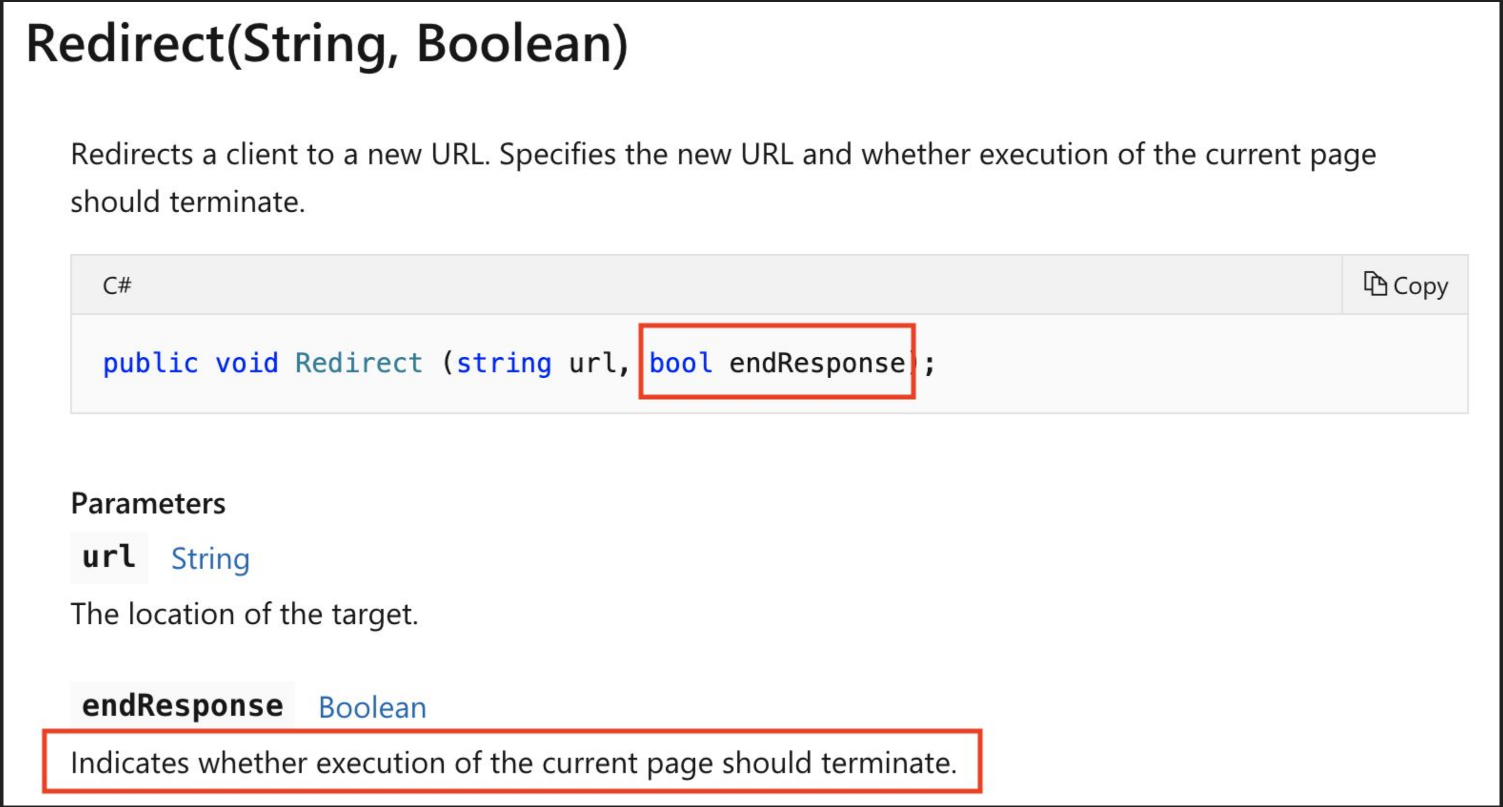

Intruder attack 17

Results | Target | Positions | Payloads | Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		404	☐	☐	1332	
1	example.aspx	301	☐	☐	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

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

Parameters

`url` String

The location of the target.

`endResponse` Boolean

Indicates whether execution of the current page should terminate.

Authz Functions

Exercise Sigmund Freud

What authorization functions are in place within the target application?

Checklists & Reviews

Authorization

Authorization Review

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

Authorization Review Vulnerabilities

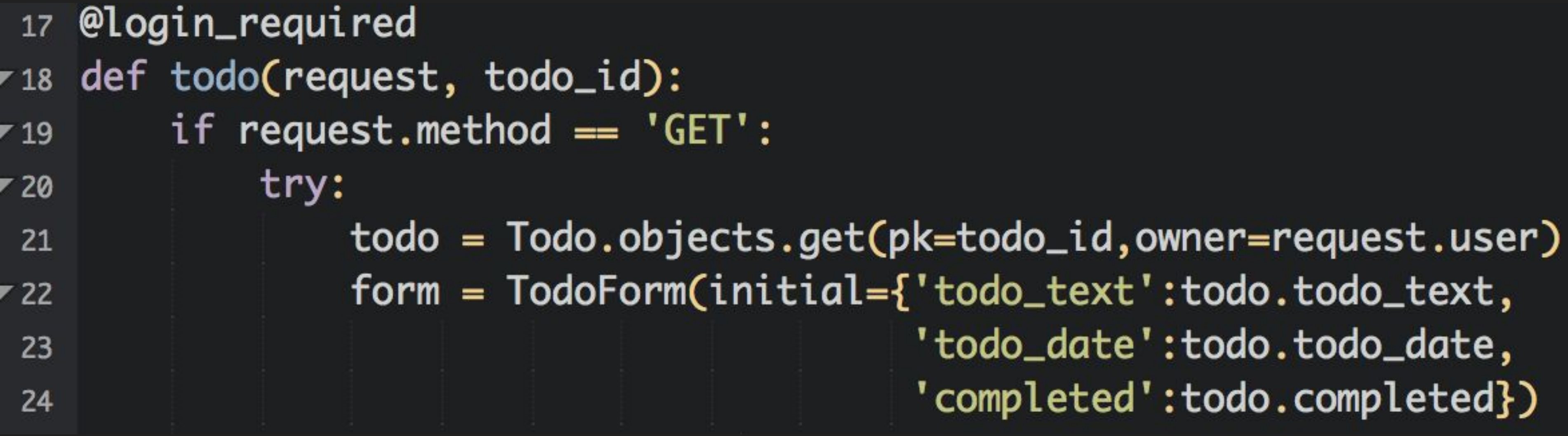
- Broken Access Control - OWASP Top 10 A5:2017
  - Privilege Escalation
  - Missing Function Level Access Control
  - Insecure Direct Object Reference
- Sensitive Data Exposure - OWASP Top 10 A3:2017
- Mass Assignment
- Business Logic Flaws

Mass-Assignment - Node

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

Genghis Khan Exercise

- What pieces of this code are authorization related?



```python
@login_required
def todo(request, todo_id):
    if request.method == 'GET':
        try:
            todo = Todo.objects.get(pk=todo_id,owner=request.user)
            form = TodoForm(initial={'todo_text':todo.todo_text,
                                     'todo_date':todo.todo_date,
                                     'completed':todo.completed})
```

Authentication

Authentication Review

How does an application confirm identity?

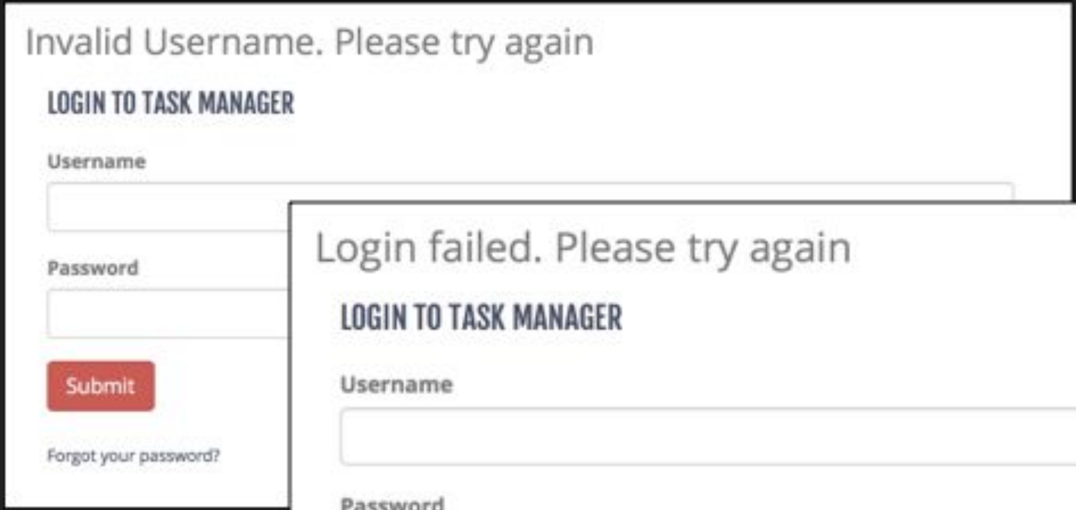
Authentication Review

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

Authentication Review Vulnerabilities

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

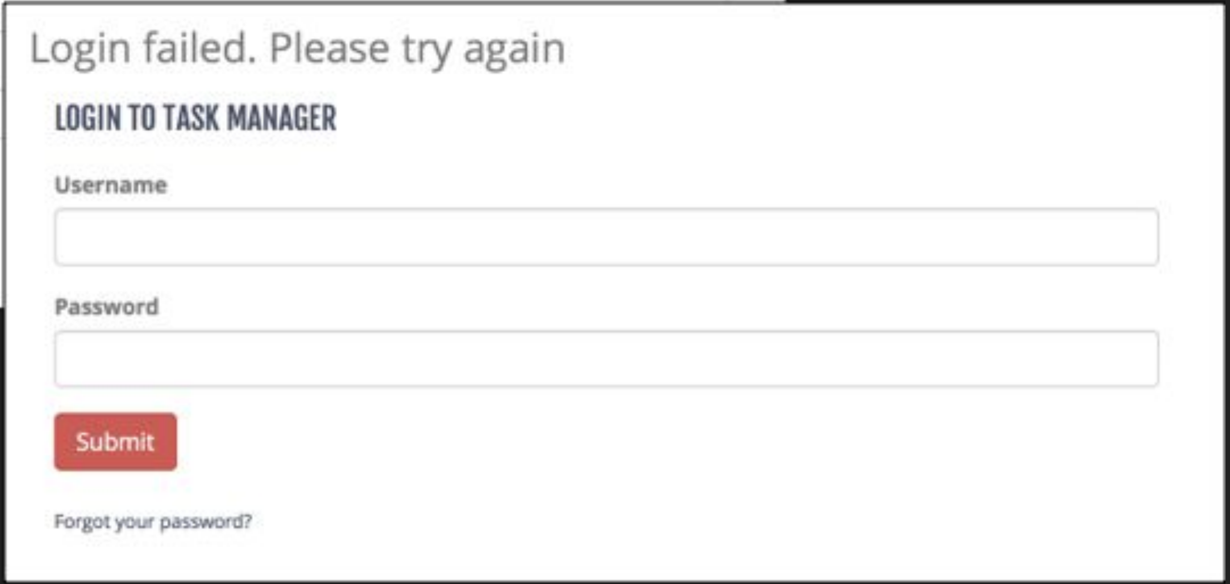
User Enumeration - Login Page


Invalid Username. Please try again

LOGIN TO TASK MANAGER

Username

Password

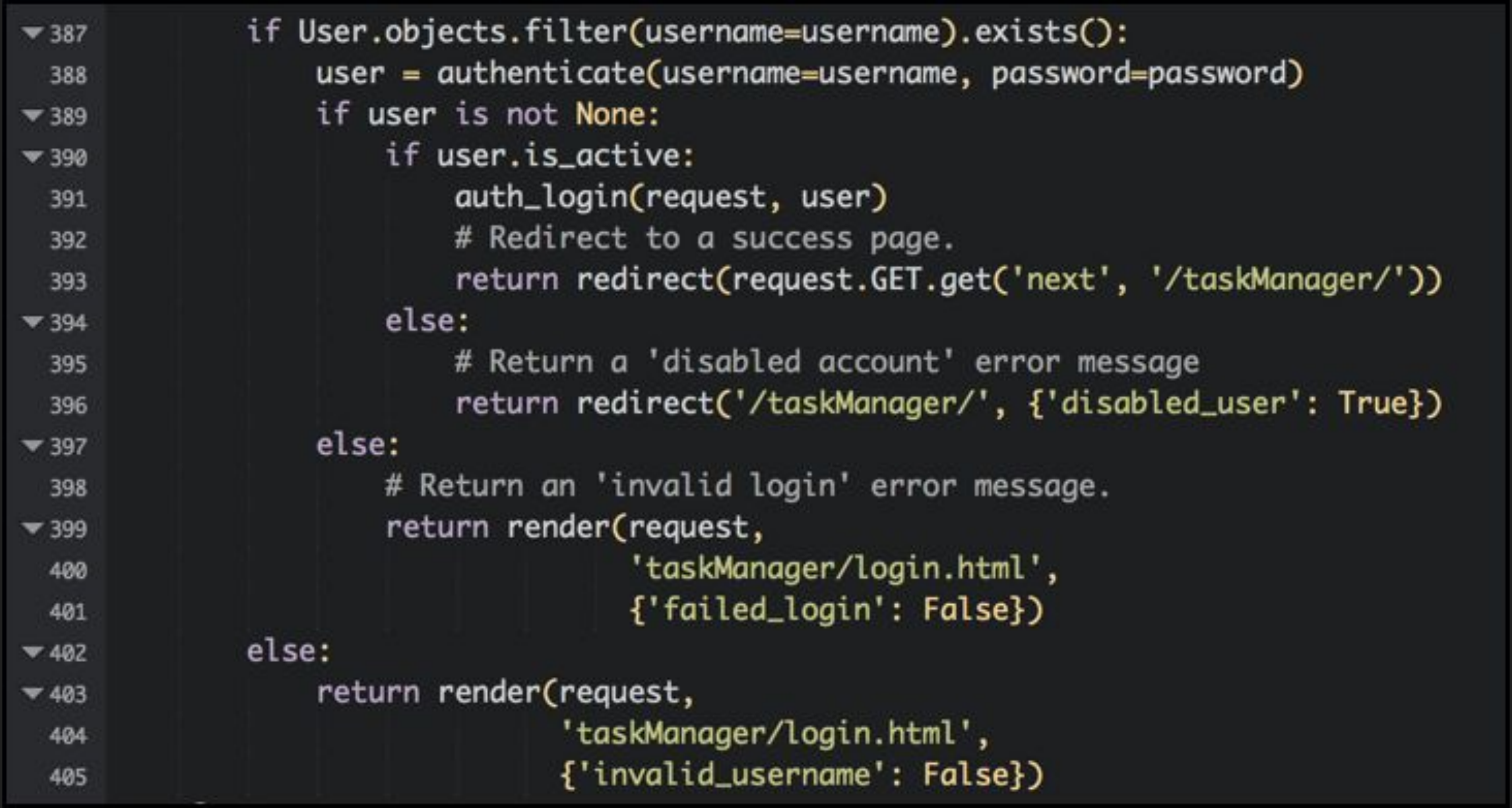

Login failed. Please try again

LOGIN TO TASK MANAGER

Username

Password

User Enumeration - Django



```python
        if User.objects.filter(username=username).exists():
            user = authenticate(username=username, password=password)
            if user is not None:
                if user.is_active:
                    auth_login(request, user)
                    # Redirect to a success page.
                    return redirect(request.GET.get('next', '/taskManager/'))
                else:
                    # Return a 'disabled account' error message
                    return redirect('/taskManager/', {'disabled_user': True})
            else:
                # Return an 'invalid login' error message.
                return render(request,
                              'taskManager/login.html',
                              {'failed_login': False})
        else:
            return render(request,
                          'taskManager/login.html',
                          {'invalid_username': False})
```

Auditing

Auditing Review

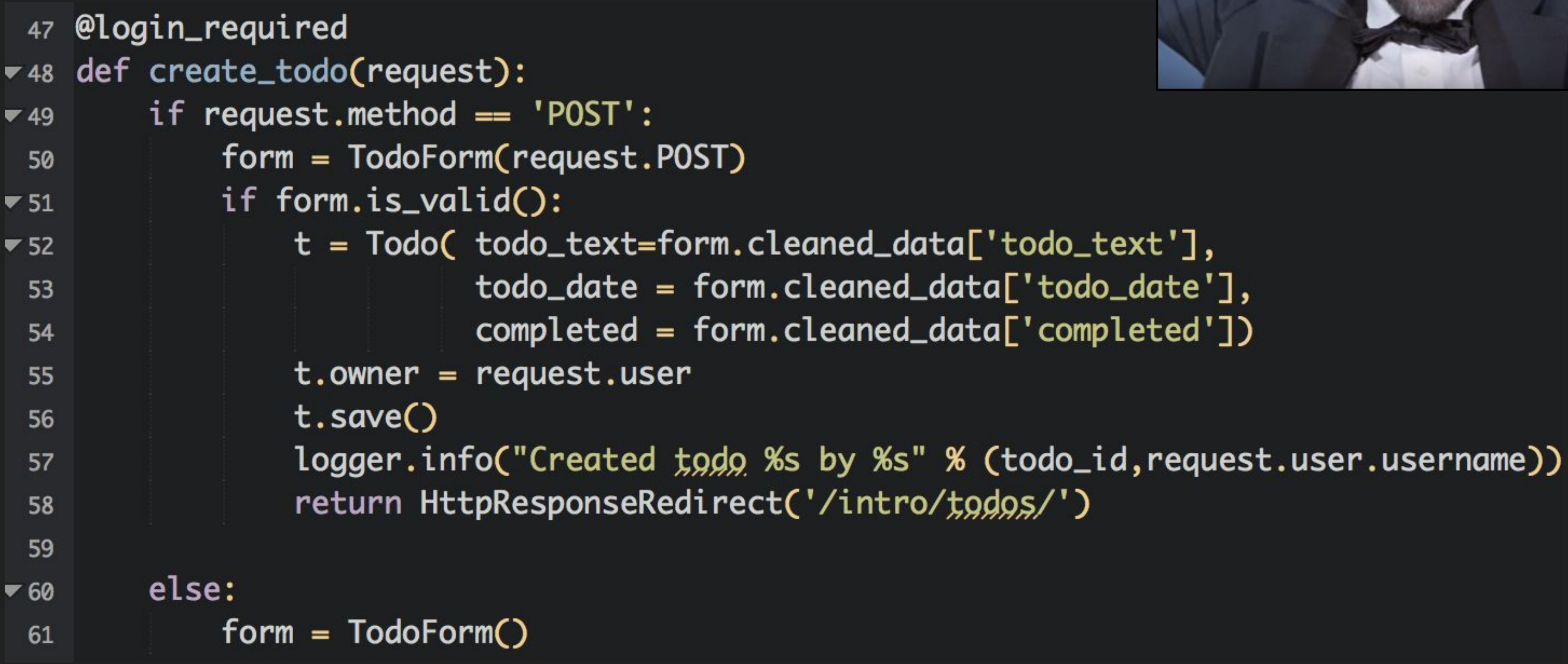
- Validate that appropriate logging and exception handling are handled within application source
- One path in the trace of sensitive data from source to sink
- Logging functions and error messages are considered a data sink
- Logging should happen in any endpoint that performs a state-changing operation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

Auditing Review Vulnerabilities

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

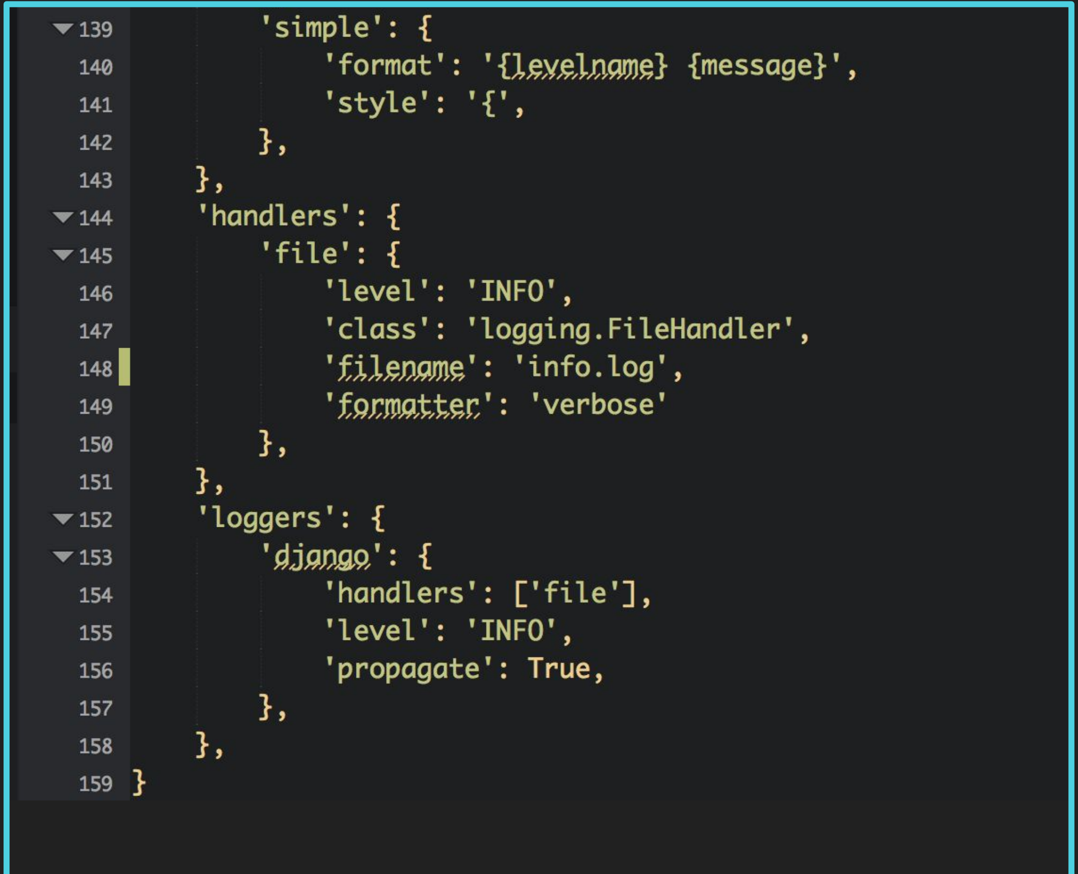
Abraham Lincoln Exercise

- So where should we look? Sensitive functions.



```python
@login_required
def create_todo(request):
    if request.method == 'POST':
        form = TodoForm(request.POST)
        if form.is_valid():
            t = Todo( todo_text=form.cleaned_data['todo_text'],
                      todo_date = form.cleaned_data['todo_date'],
                      completed = form.cleaned_data['completed'])
            t.owner = request.user
            t.save()
            logger.info("Created todo %s by %s" % (todo_id,request.user.username))
            return HttpResponseRedirect('/intro/todos/')

    else:
        form = TodoForm()
```

Abraham Lincoln Exercise



```
        'simple': {
            'format': '{levelname} {message}',
            'style': '{',
        },
    },
    'handlers': {
        'file': {
            'level': 'INFO',
            'class': 'logging.FileHandler',
            'filename': 'info.log',
            'formatter': 'verbose'
        },
    },
    'loggers': {
        'django': {
            'handlers': ['file'],
            'level': 'INFO',
            'propagate': True,
        },
    },
}
```

Injection

Injection

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

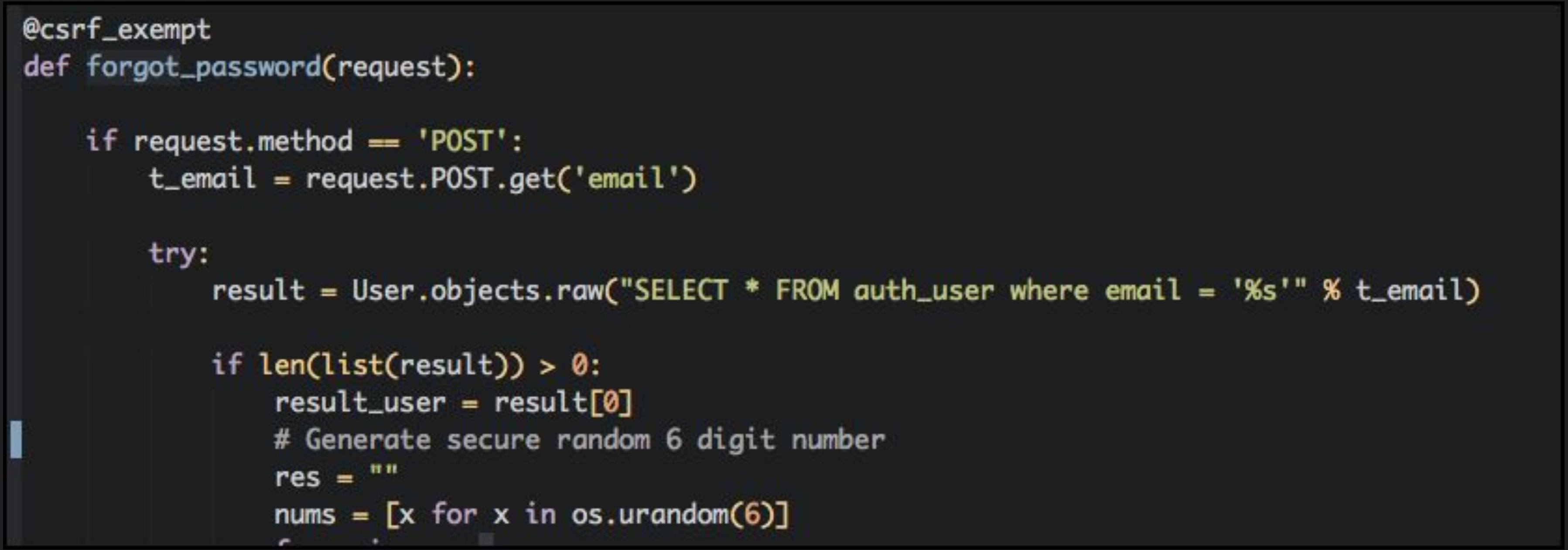
Injection Vulnerabilities

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

Input Validation

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

SQL Injection - Django

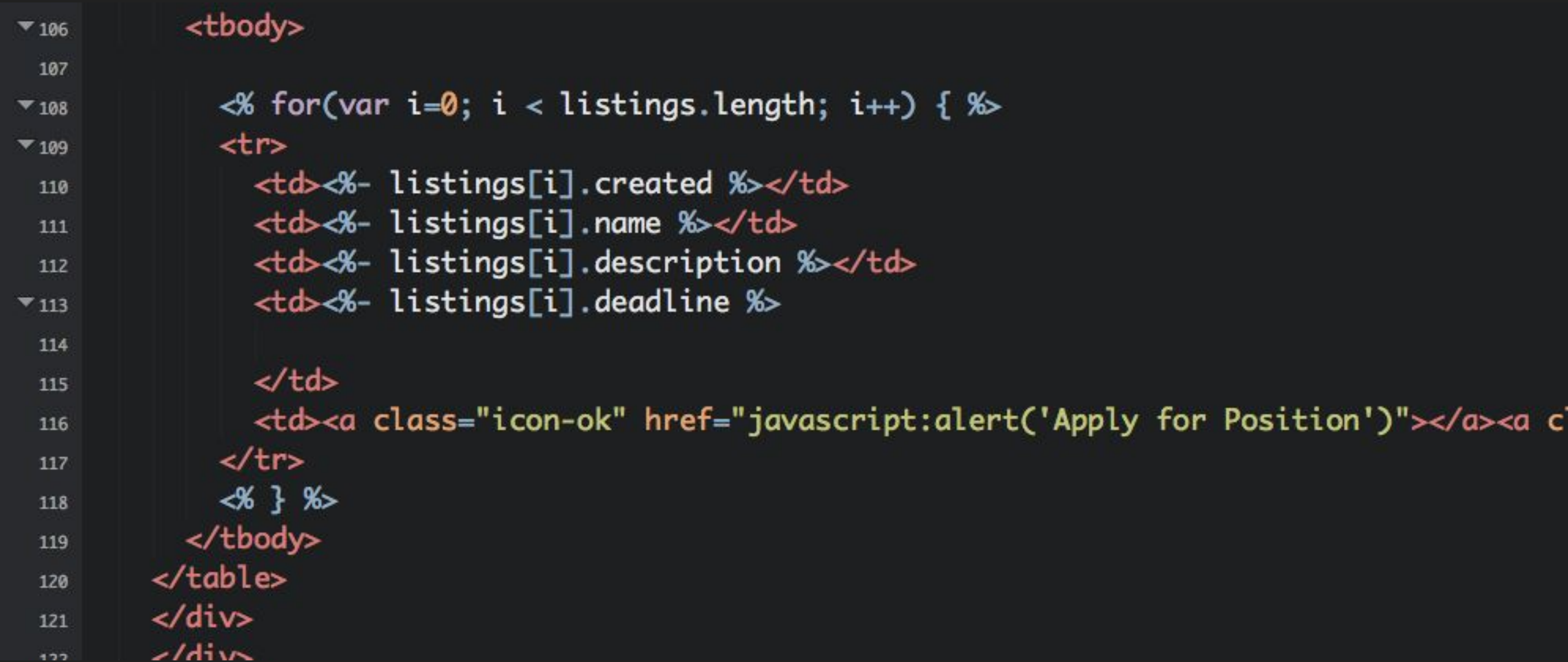


```
@csrf_exempt
def forgot_password(request):

    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)]
```

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

XSS - Node.js (ejs templates)



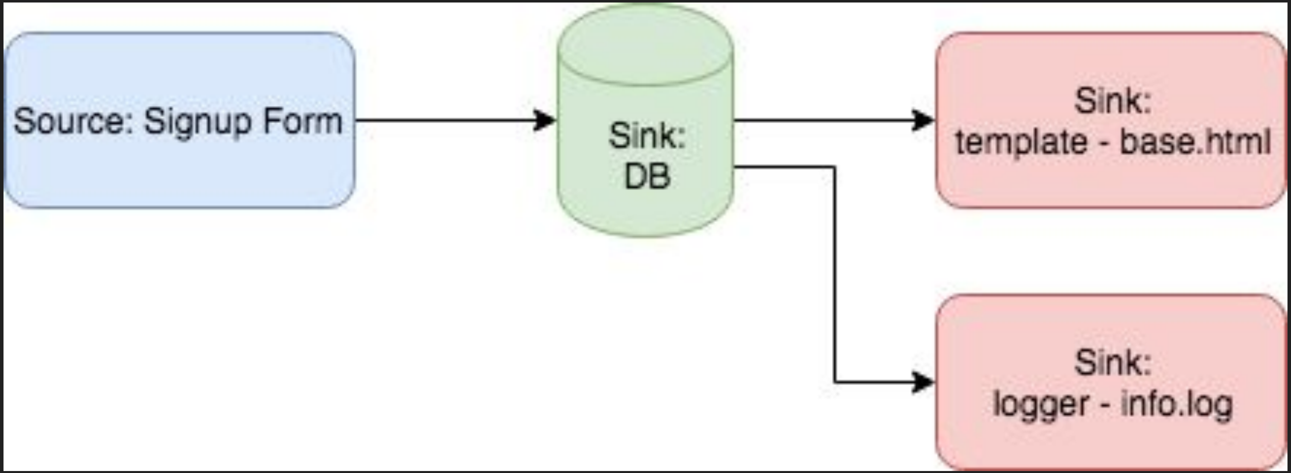
```
      <tbody>

        <% for(var i=0; i < listings.length; i++) { %>
        <tr>
          <td><%- listings[i].created %></td>
          <td><%- listings[i].name %></td>
          <td><%- listings[i].description %></td>
          <td><%- listings[i].deadline %>

          </td>
          <td><a class="icon-ok" href="javascript:alert('Apply for Position')"></a><a c
        </tr>
        <% } %>
      </tbody>
    </table>
    </div>
    </div>
```

Exercise Billy the Kid

- Perform a Source to Sink Trace for usernames.
- Some sources have already been identified. Are there others?

Exercise Billy the Kid - Post-Mortem

- Sources/sinks coming from the Signup Form could result in:
  a. SQL Injection (database storage)
  b. XSS (Stored/Reflected)
  c. Log Forging/Injection



Cryptographic Analysis

Cryptographic Analysis

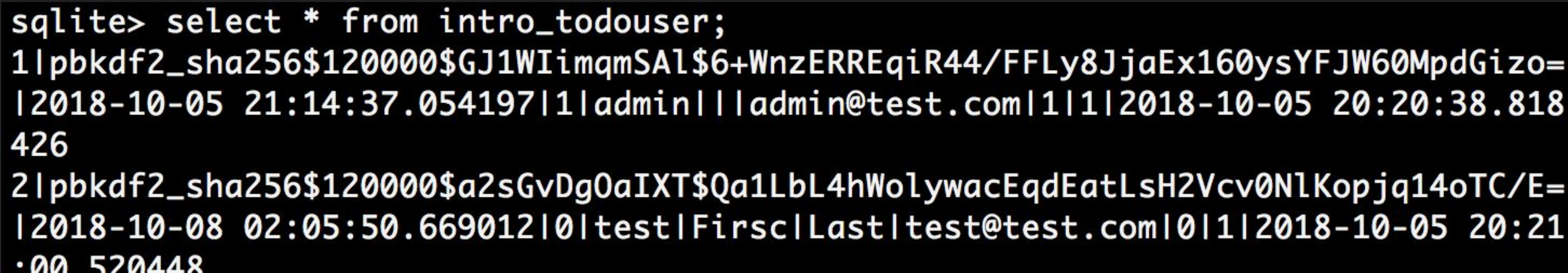
- Analyze code for encryption flaws, outdated protocols, custom-developed algorithms, weak encryption, and misuse
- Automated tools will uncover some of this, including
  - Use of older hashing algorithms (MD5, SHA-1, etc)
- Code and routes that handles sensitive information specifically should be reviewed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

Cryptographic Analysis Vulnerabilities

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

Exercise Beethoven

- How are passwords being stored?



```
sqlite> select * from intro_todouser;
1|pbkdf2_sha256$120000$GJ1WIimqmSAl$6+WnzERREqiR44/FFLy8JjaEx160ysYFJW60MpdGizo=
|2018-10-05 21:14:37.054197|1|admin||admin@test.com|1|1|2018-10-05 20:20:38.818
426
2|pbkdf2_sha256$120000$a2sGvDgOaIXT$Qa1LbL4hWolywacEqdEatLsH2Vcv0NlKopjq14oTC/E=
|2018-10-08 02:05:50.669012|0|test|Firsc|Last|test@test.com|0|1|2018-10-05 20:21
:00.520448
```

Configuration Review

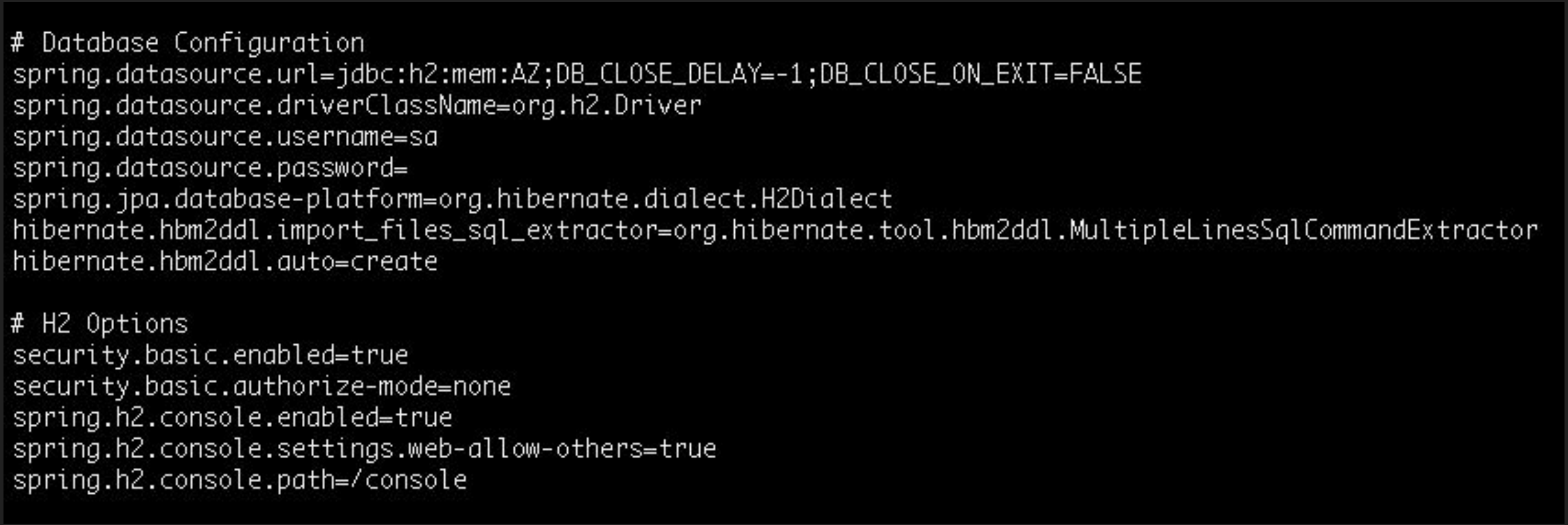
Configuration Review

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

Configuration Review Vulnerabilities

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

Security Misconfiguration - Java Spring



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

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

Additional Resources

Additional Resources

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

Absolute AppSec Secure Code Review Methodology

1. Application Overview & Risk Assessment
2. Information Gathering
3. Checklist Creation
4. Perform Reviews

Checklists/Reviews

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

BE EXCELLENT TO EACH OTHER AND
PARTY ON DUDES
made on imgur