

Большие языковые модели в задачах анализа и обнаружения уязвимостей в программном коде

Чаругин Валентин Валерьевич, преподаватель кафедры КИБ ИИИ, аспирант

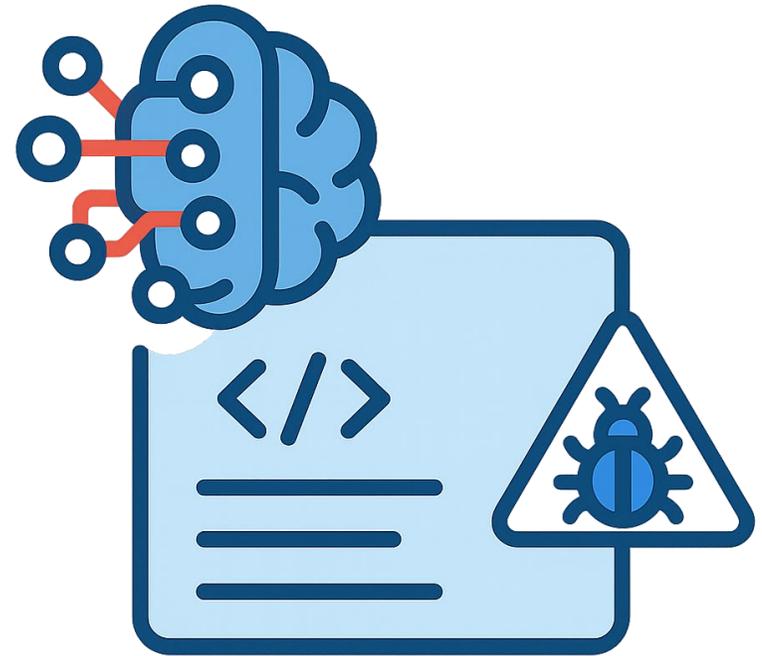
Использование больших языковых моделей в анализе кода

Проблема:

Традиционные подходы к анализу кода недостаточно эффективно справляются с новыми типами уязвимостей. Возникает необходимость в использовании более гибких и интеллектуальных инструментов, способных учитывать не только синтаксис, но и логику, структуру и поведение программ. Одним из таких решений являются большие языковые модели.

Цель работы:

- Исследовать возможности и ограничения применения больших языковых моделей для анализа уязвимостей в программном коде.
- Проанализировать, насколько точно и полно большие языковые модели и инструменты с ними могут выявлять уязвимости, классифицировать их по CWE, локализовать уязвимый код, предложить качественные исправления и дать обоснованные рекомендации.
- Провести подробный анализ выявления и устранения реальной уязвимости CVE-2025-25296 с помощью различных инструментов и моделей.



Использование больших языковых моделей в анализе кода

Инструменты на основе LLM для обнаружения уязвимостей программного обеспечения:



Универсальные большие языковые модели



Специализированные большие языковые модели



Интеграция больших языковых моделей с инструментами статического анализа кода

Решаемые задачи и уязвимости



Определение наличия уязвимости в коде



Автоматическое создание исправленного кода



Определение фрагмента уязвимого кода



Выдача рекомендаций по устранению уязвимостей

Решаемые задачи и уязвимости

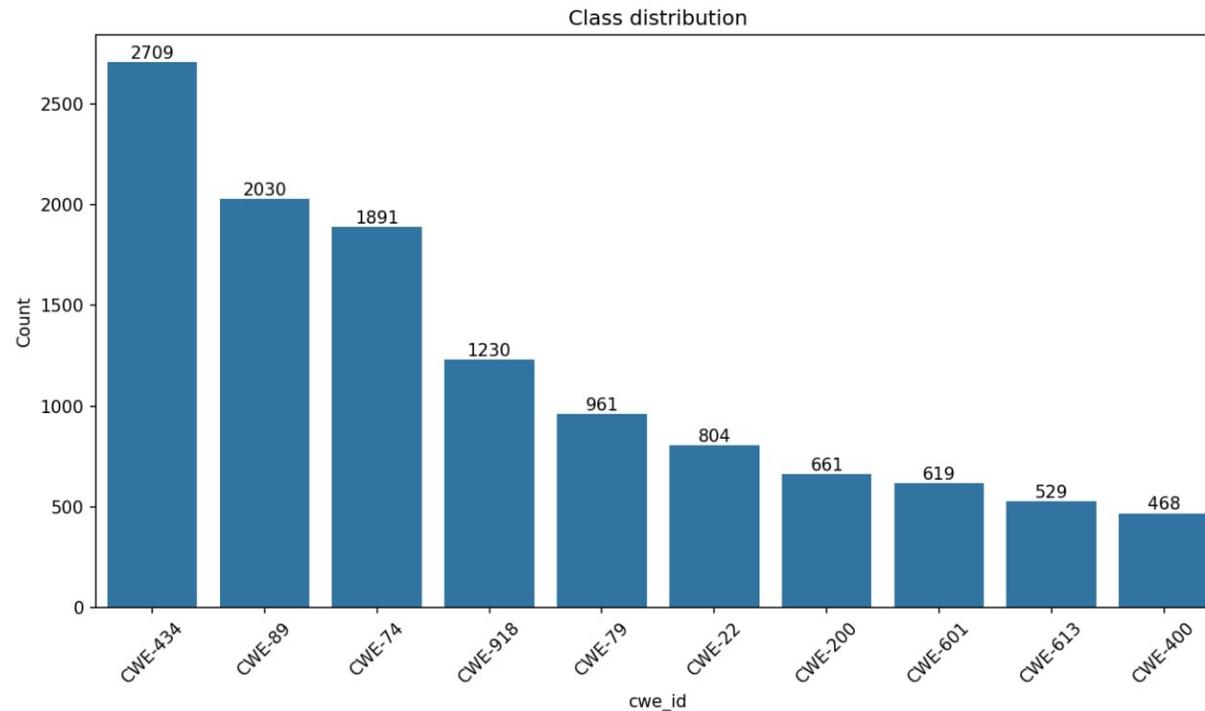
Решая задачи анализа и устранения уязвимостей, необходимо учитывать существующие стандарты, которые помогают систематизировать и приоритизировать угрозы.

Таким стандартом является **база CVE** – хранилище задокументированных уязвимостей, обнаруженных в программном обеспечении.

CWE – категории, которые описывают причины, приводящие к уязвимостям.



Решаемые задачи и уязвимости



Гистограмма распределения актуальных категорий уязвимостей для языка Python за 2021-2025 год

Вариант устранения уязвимости



CVE-2025-25296 **CWE-79**

Уязвимость, связанная с некорректной обработкой пользовательского ввода в веб-приложении на Python.

Первый вариант устранения уязвимости.

Для ее исправления необходимо ограничить метод запроса, то есть требуется удалить поддержку GET-запросов для эндпоинта, чтобы принимались только POST-запросы.

```
31 31 def playground_replacements(request, task_data):
32 32     if request.GET.get('playground', '0') == '1':
33 33         for key in task_data:
34 34             if '/samples/time-series.csv' in task_data[key]:
35 35                 task_data[key] = 'https://app.heartex.ai' + task_data[key]
36 36     return task_data
37 37
38 38
39 - @require_http_methods(['GET', 'POST'])
39 + @require_http_methods(['POST'])
40 40 def upload_example_using_config(request):
41 41     """Generate upload data example by config only"""
42 - config = request.GET.get('label_config', '')
43 - if not config:
44 -     config = request.POST.get('label_config', '')
42 + config = request.POST.get('label_config', '')
45 43
46 44     org_pk = get_organization_from_request(request)
47 45     secure_mode = False
48 46     if org_pk is not None:
49 47         org = generics.get_object_or_404(Organization, pk=org_pk)
50 48         secure_mode = org.secure_mode
51 49
52 50     try:
53 51         Project.validate_label_config(config)
54 52         task_data, __, _ = get_sample_task(config, secure_mode)
55 53         task_data = playground_replacements(request, task_data)
56 54     except (ValueError, ValidationError, lxml.etree.Error):
57 55         response = HttpResponse('error while example generating', status=status.HTTP_400_BAD_REQUEST)
58 56     else:
59 57         response = HttpResponse(json.dumps(task_data))
60 58     return response
```

Вариант устранения уязвимости



CVE-2025-25296 **CWE-79**

Уязвимость, связанная с некорректной обработкой пользовательского ввода в веб-приложении на Python.

Второй вариант устранения уязвимости.

Для исправления нужно использовать JsonResponse, который не даёт браузеру обработать JSON как HTML-страницу и запустить скрипты.

```
11 + from django.http import JsonResponse
11 12 from django.shortcuts import render
12 13 from django.views.decorators.http import require_http_methods
13 14 from organizations.models import Organization

@@ -36,25 +37,25 @@
39 40 @require_http_methods(['GET', 'POST'])
40 41 def upload_example_using_config(request):
41 42     """Generate upload data example by config only"""
42 43     config = request.GET.get('label_config', '')
43 44     if not config:
44 45         config = request.POST.get('label_config', '')
45 46
46 47     org_pk = get_organization_from_request(request)
47 48     secure_mode = False
48 49     if org_pk is not None:
49 50         org = generics.get_object_or_404(Organization, pk=org_pk)
50 51         secure_mode = org.secure_mode
51 52
52 53     try:
53 54         Project.validate_label_config(config)
54 55         task_data, _, _ = get_sample_task(config, secure_mode)
55 56         task_data = playground_replacements(request, task_data)
56 57     except (ValueError, ValidationError, lxml.etree.Error):
57 58         response = HttpResponse('error while example generating', status=status.HTTP_400_BAD_REQUEST)
58 59     else:
59 - response = HttpResponse(json.dumps(task_data))
60 + response = JsonResponse(task_data)
60 61     return response
```

Вариант устранения уязвимости



CVE-2025-25296 CWE-79

Уязвимость, связанная с некорректной обработкой пользовательского ввода в веб-приложении на Python.

Третий вариант устранения уязвимости.
Добавление фильтрации пользовательского ввода с помощью библиотеки bleach. Перед тем как сформировать URL, значение из task_data проходит очистку - удаляются опасные HTML-теги и код JavaScript.

```
5 + import bleach
5 6
6 7 import lxml.etree
7 8 from core.label_config import get_sample_task
@@ -12,27 +13,28 @@
31 32 def playground_replacements(request, task_data):
32 33     if request.GET.get('playground', '0') == '1':
33 34         for key in task_data:
34 35             if '/samples/time-series.csv' in task_data[key]:
35 - task_data[key] = 'https://app.heartex.ai' + task_data[key]
36 + cleaned_value = bleach.clean(task_data[key])
37 + task_data[key] = 'https://app.heartex.ai' + cleaned_value
36 38     return task_data
37 39
38 40
39 41 @require_http_methods(['GET', 'POST'])
40 42 def upload_example_using_config(request):
41 43     """Generate upload data example by config only"""
42 44     config = request.GET.get('label_config', '')
43 45     if not config:
44 46         config = request.POST.get('label_config', '')
45 47
46 48     org_pk = get_organization_from_request(request)
47 49     secure_mode = False
48 50     if org_pk is not None:
49 51         org = generics.get_object_or_404(Organization, pk=org_pk)
50 52         secure_mode = org.secure_mode
51 53
52 54     try:
53 55         Project.validate_label_config(config)
54 56         task_data, _ = get_sample_task(config, secure_mode)
55 57         task_data = playground_replacements(request, task_data)
56 58     except (ValueError, ValidationError, lxml.etree.Error):
57 59         response = HttpResponse('error while example generating', status=status.HTTP_400_BAD_REQUEST)
58 60     else:
59 61         response = HttpResponse(json.dumps(task_data))
60 62     return response
```

Готовые решения для обеспечения безопасности программного обеспечения



Применение готовых инструментов для анализа уязвимостей



ChatGPT

ChatGPT 4o

Are there any coding vulnerabilities in this code? If yes, which CWE ID(s) apply to them?

Yes, there are a few potential security and robustness issues in the provided code. Below is an analysis of the key vulnerabilities and their corresponding *Common Weakness Enumeration (CWE)* identifiers:

- 1. Use of `lxml` without Secure XML Parser Configuration**
 - **Location:** Exception handling mentions `lxml.etree.Error`, implying `lxml` may be used in the `get_sample_task()` function (from `core.label_config`), though its implementation isn't shown here.
 - **Risk:** If external or untrusted XML input is parsed without disabling DTD and external entity expansion, it may allow:
 - XXE (XML External Entity Injection)
 - Billion Laughs Attack (DoS)
 - **CWE IDs:**
 - **CWE-611:** Improper Restriction of XML External Entity Reference (XXE)
 - **CWE-776:** Improper Restriction of Recursive Entity References in DTDs (Billion Laughs)
- 2. Use of `json.dumps()` Without `safe=True` in `HttpResponse`**
 - **Location:** `response = HttpResponse(json.dumps(task_data))`
 - **Risk:** While `json.dumps()` helps serialize data, this `HttpResponse` should explicitly set `content_type='application/json'` to avoid ambiguity or XSS vectors in some contexts.
 - **CWE ID:** **CWE-79:** Improper Neutralization of Input During Web Page Generation (Cross-site Scripting) (applies if rendered improperly on the frontend)

Спросите что-нибудь...

Поиск Глубокое исследование Создать изображение

ChatGPT может допускать ошибки. Проверяйте важную информацию. См. настройки конфиденциальности.



YandexGPT

Про YandexGPT 5 Pro

Are there any coding vulnerabilities in this code? If yes, which CWE ID(s) apply to them?

After analyzing the provided Python code, I've identified several potential vulnerabilities:

- 1. CWE-400: Uncontrolled Resource Consumption**
 - The `MAX_LABEL_CONFIG_LENGTH` limit is set to 10000 characters, which might still be too high and could potentially lead to resource exhaustion under certain circumstances.
- 2. CWE-79: Cross-Site Scripting (XSS)**
 - In the `playground_replacements` function, there's URL manipulation without proper sanitization:
 - This could potentially lead to XSS if user-provided data is not properly sanitized before concatenation.

Как вам помочь?

Алиса старается, но может ошибаться — проверяйте важное. Подробнее

Применение готовых инструментов для анализа уязвимостей



GitHub Copilot

```
python_code.py X
python_code.py
15 from rest_framework import generics, status
16 from rest_framework.exceptions import ValidationError
17
18 logger = logging.getLogger(__name__)
19
20
21 @login_required
22 def project_list(request):
23     return render(request, 'projects/list.html')
24
25
26 @login_required
27 def project_settings(request, pk, sub_path):
28     return render(request, 'projects/settings.html')
29
30
31 def playground_replacements(request, task_data):
32     if request.GET.get('playground', '0') == '1':
33         for key in task_data:
34             if '/samples/time-series.csv' in task_data[key]:
35                 task_data[key] = 'https://app.heartex.ai' + task_data[key]
36     return task_data
37
38
39 @require_http_methods(['GET', 'POST'])
```

CHAT

copilot

Are there any coding vulnerabilities in this code? If yes, which CWE ID(s) apply to them?

python_code.py:15-51

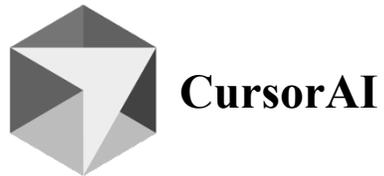
GitHub Copilot

> Used 1 reference

Yes, there are potential coding vulnerabilities in the provided code. Below are the identified issues along with their corresponding CWE (Common Weakness Enumeration) IDs:

- Cross-Site Scripting (XSS) (CWE-79)**
 - Issue:** The `playground_replacements` function modifies `task_data` based on user input (`request.GET.get('playground')`) and returns it without ensuring that the data is properly sanitized. If this data is rendered in a template or returned to the client, it could lead to XSS attacks.
 - CWE ID:** [CWE-79: Improper Neutralization of Input During Web Page Generation \('Cross-Site Scripting'\)](#)
- Insecure Direct Object Reference (IDOR) (CWE-639)**
 - Issue:** In `upload_example_using_config`, the `org.pk` is retrieved from the request and used to fetch an `Organization` object without verifying that the user has permission to access it. This could allow unauthorized users to access or manipulate data they shouldn't have access to.
 - CWE ID:** [CWE-639: Authorization Bypass Through User-Controlled Key](#)

Применение готовых инструментов для анализа уязвимостей

A screenshot of the CursorAI interface. On the left, a file explorer shows a project with files 'cpp_code.cpp' and 'python_code.py'. The main editor displays the content of 'python_code.py', which includes a license header and several imports and function calls. On the right, a chat window titled 'Identifying Coding Vulnerabilities and...' contains a prompt: 'Are there any coding vulnerabilities in this code? If yes, which CWE ID(s) apply to them?'. Below the prompt, the AI response identifies two vulnerabilities: '1. CWE-79: Cross-Site Scripting (XSS)' and '2. CWE-89: SQL Injection', each with a list of specific code-related details.

```
1 """This file and its contents are licensed under the Apache License 2
2 """
3 import json
4 import logging
5 from django.core.exceptions import ValidationError as DjangoValidation
6
7 import lxml.etree
8 from core.label_config import get_sample_task
9 from core.utils.common import get_organization_from_request
10 from django.contrib.auth.decorators import login_required
11 from django.http import HttpResponse
12 from django.shortcuts import render
13 from django.views.decorators.http import require_http_methods
14 from organizations.models import Organization
15 from projects.models import Project
```

Identifying Coding Vulnerabilities and...

python_code.py

Are there any coding vulnerabilities in this code? If yes, which CWE ID(s) apply to them?

Restore checkpoint

Let me analyze the code for potential security vulnerabilities. I'll identify several issues and their corresponding CWE IDs:

- CWE-79: Cross-Site Scripting (XSS)**
 - The `playground_replacements` function directly modifies task data without proper sanitization
 - The modified data is later returned in the HTTP response, which could lead to XSS if the data contains malicious JavaScript
 - The modified data is later returned in the HTTP response, which could lead to XSS if the data contains malicious JavaScript
- CWE-89: SQL Injection**
 - While not directly visible in this code, the `get_organization_from_request` and subsequent database queries could be vulnerable if proper parameterization isn't used in the underlying implementation

Применение готовых инструментов для анализа уязвимостей



The screenshot displays the Snyk Code IDE interface. On the left, a sidebar shows a summary of issues: 1 total, 0 new. It lists 'All scans are completed', '1 issues found in your open projects', and '1 issues are fixable with Snyk DeepCode AI'. Under 'OPEN SOURCE SECURITY', it shows 'code-analysis' with 1 issue. Under 'CODE SECURITY', it shows '1 issues found by Snyk' and '1 issue can be fixed by Snyk DeepCode AI'. A specific issue is highlighted: 'Cross-site Scripting (XSS) python_code.py[75, 20]'. Under 'CONFIGURATION ISSUES' and 'CODE QUALITY', it shows 'Congrats! No issues found!'. The central code editor shows a Python function `upload_example_using_config` with a comment '# Validate input before processing' and a call to `validate_label_config_input`. The function returns an `HttpResponse` object. The right sidebar shows a detailed view of the 'Cross-site Scripting (XSS)' issue, including its position (line 75), priority score (784), and CWE-79. It provides a fix analysis: 'Unsanitized input from an HTTP parameter flows into django.http.HttpResponse, where it is used to render an HTML page returned to the user. This may result in a Cross-Site Scripting attack (XSS)'. Below this, it offers 5 AI-generated solutions, with the second solution highlighted:

```
response = JsonResponse(task_data)
```

Применение готовых инструментов для анализа уязвимостей



Risk of XSS from Direct Use of HttpResponseRedirect

python_code.py

High

Open

```
54     task_data, __, __ = get_sample_task(config, secure_mode)
55     task_data = playground_replacements(request, task_data)
56     except (ValueError, ValidationError, lxml.etree.Error):
57         response = HttpResponseRedirect('error while example generating', status=status.HTTP_400_BAD_REQUEST)
58     else:
59         response = HttpResponseRedirect(json.dumps(task_data))
60     return response
61
```

Issue Trace

Node

Description	Mitigation	AI Assistant	Audit	History	Details
-------------	------------	--------------	-------	---------	---------

Identified Vulnerability:
Risk of Cross-Site Scripting (XSS) from direct use of HttpResponseRedirect when returning unescaped or unvalidated content (here, task_data) in the response.

Code Fix:
Use JsonResponse instead of HttpResponseRedirect for returning JSON data. JsonResponse automatically sets the correct content type (application/json) and escapes data appropriately to prevent XSS.

Explanation:
Directly returning JSON data using HttpResponseRedirect can lead to XSS if task_data contains untrusted or user-controlled content, as it may not be properly escaped or serialized. JsonResponse safely serializes data and sets the appropriate headers, mitigating the risk of XSS.

Fixed Code:

```
from django.http import JsonResponse, HttpResponseRedirect
from rest_framework import status

try:
    task_data, __, __ = get_sample_task(config, secure_mode)
    task_data = playground_replacements(request, task_data)
except (ValueError, ValidationError, lxml.etree.Error):
    response = HttpResponseRedirect('error while example generating', status=status.HTTP_400_BAD_REQUEST)
else:
    # Use JsonResponse to safely serialize and return JSON data
    response = JsonResponse(task_data, safe=False)
return response
// Use JsonResponse to prevent XSS and set the correct Content-Type header
```

Summary:
The vulnerability occurs because HttpResponseRedirect does not automatically escape or serialize JSON data, which can allow XSS if the data contains malicious scripts. JsonResponse ensures the data is properly serialized and escaped, preventing XSS attacks.

Результаты готовых инструментов для анализа уязвимостей

Критерии оценивания:

К1 – наличие уязвимости

К2 – точность сопоставления CVE

К3 – точность указания источника уязвимости

К4 – качество предлагаемого исправления

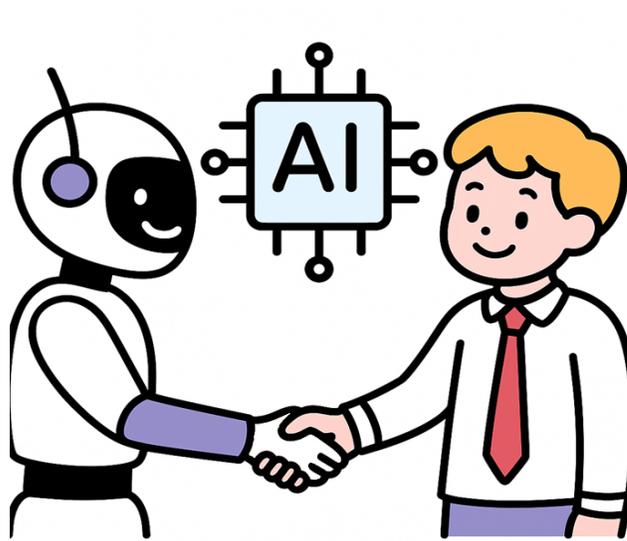
К5 – полнота рекомендаций

	К1	К2	К3	К4	К5	Итог
ChatGPT	0.5	0.5	0.5	0.5	0.5	2.5
YandexGPT	0.5	0.5	0.5	0.5	0.5	2.5
GitHub Copilot	1	1	0.5	0.5	0.5	3.5
CursorAI	1	1	0.5	0.5	0.5	3.5
Snyk Code	1	1	0.5	0.5	1	4.0
CodeThreat	1	1	0.5	0.5	0.5	3.5

Сравнительная оценка инструментов по качеству анализа уязвимости CVE-2025-25296

Доверие к большим языковым моделям, применяемым в задачах анализа и обнаружения уязвимостей в программном коде

Результаты анализа показывают, что полностью полагаться на такие модели в критически важных задачах нельзя. Их следует рассматривать как дополнительные инструменты, способные повысить скорость и эффективность анализа. Принятие окончательных решений в обеспечении безопасности остаётся за специалистом ИБ.



Спасибо за внимание!