



武汉大学 国家网络安全学院

SCHOOL OF CYBER SCIENCE AND ENGINEERING·WHU

# DataCon2025 软件供应链安全赛道题解

武汉大学 - 哈基龙战队

2025 年 11 月 13 日

# 目 录

1	问题重述 .....	1
2	题目分析及相关工作 .....	1
2.1	题目数据集分析 .....	1
2.2	密钥分析 .....	2
2.3	密钥检测方法 .....	2
3	解题思路 1：基于开源工具的改进 .....	3
3.1	基于开源工具的解答 .....	3
3.2	对于 KEYSSENTINEL 的分析 .....	3
3.3	检测结果分析 .....	7
3.4	对于 KEYSSENTINEL 的改进 .....	9
4	解题思路 2：基于 LLM 的构造密钥检测 .....	10
4.1	背景 .....	10
4.2	实现方案 .....	11
4.3	实验验证 .....	12
	<b>附录 A 解题思路 1 关键代码 .....</b>	<b>15</b>
	<b>附录 B 解题思路 2 关键代码 .....</b>	<b>25</b>

## 1 问题重述

随着开源协作与持续集成流程的广泛应用，密钥泄露事件愈发频繁，各类文本文件中常常埋藏着 API Token、访问密钥等敏感凭证。我们需要分析数据集，开发自动化检测工具，尽可能发现潜在密钥并有效减少误报。

## 2 题目分析及相关工作

### 2.1 题目数据集分析

本次赛题提供的数据集贴近真实环境，具有以下特点：

- 数据规模大：包含海量文件（如源代码、配置文件、日志、数据等）；
- 分布广泛：密钥可能出现在任意文件类型或目录结构中；
- 类型复杂：数据集中密钥形式多样，还加入了若干构造性密钥，经过拼接、更长间距的上下文和特殊字符，难以通过简单正则匹配捕获。

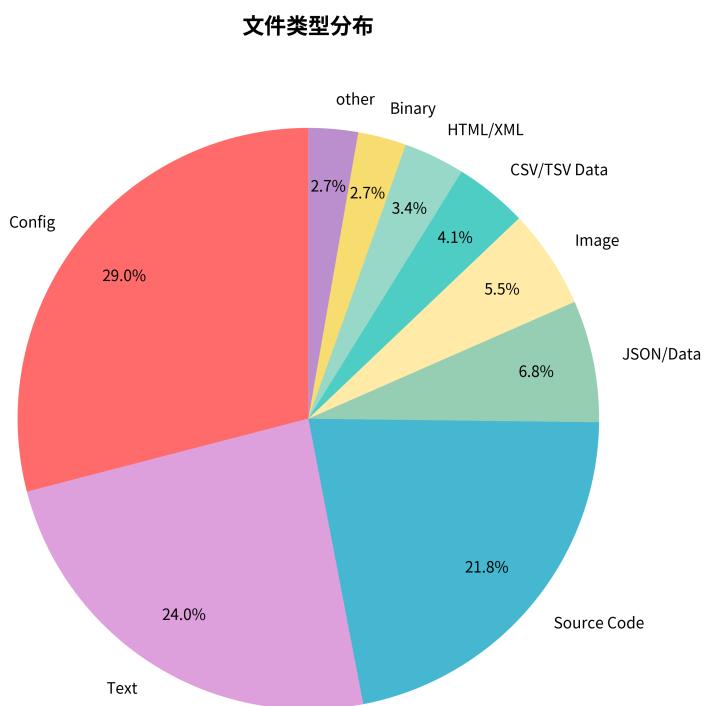


图 1 文件类型统计结果

我们使用了 `python-magic` 库和内容关键词来判断文件类型，结果如图。其中配置文件、文本类型和代码类型各占约 1/4，剩下的还有各类数据文件。

## 2.2 密钥分析

### 2.2.1 密钥格式特征

我们将用于身份验证、数据库访问、API 接口调用等功能的所有凭据（credentials）统称为“密钥”（secrets）。这些密钥主要可分为两类：**机器生成的密钥**和**人类选择的密钥**。机器生成的密钥通常由加密算法或随机数生成器创建，具有高度随机性且缺乏语义含义；而人类选择的密钥则由用户自行设定，往往包含可识别的信息或有意义的内容（如姓名、生日等）。<sup>[1]</sup>

机器生成的密钥可根据其格式特征划分为**结构化**与**非结构化**两类。结构化密钥具有明显的语法模式，例如 Qwen API 密钥通常以 sk- 开头。相比之下，部分机器生成的密钥（如 OpenCage 地理编码 API 使用的固定长度 30 位字母数字密钥）则完全随机，无明显结构特征。<sup>[2]</sup>

人类选择的密钥由于其语义性和主观性，通常也被归为非结构化类型。

### 2.2.2 密钥泄露方式

- 文本类文件：**密钥不仅可以存在于代码中，还会出现在注释、文档、配置文件等非传统位置。
- 二进制文件：**密钥也可以嵌入在二进制文件中，对于明文存储的可直接读出，加密存储的可以通过逆向工程等方式获得。
- AI 代码工具：**使用 copilot 等 AI 代码插件，或与在线 LLM 对话，会将代码直接发送到云端，其中就可能包含密钥。

## 2.3 密钥检测方法

### 2.3.1 检测工具

表 1 秘钥检测工具对比表

工具名称	多因子检测	仍在维护	公开规则数	Star 数	过滤方法
Ggshield	√	√	×	1,500	×
TruffleHog	√	√	√	13,200	熵值
Repo-supervisor	×	×	×	631	白名单, 熵值
Gitleaks	√	×	√	14,600	白名单, 熵值
Whispers	√	×	√	468	白名单, 相似度
Git-secret	√	×	×	11,800	未公开
KEYSENTINEL	√	√	√	34	语义分析, 熵值, 白名单

### 2.3.2 密钥提取

密钥提取涉及识别各种文件中的潜在秘钥。大多数工具使用正则表达式和熵检查来提取密钥。Gitleaks 和 TruffleHog 等工具依赖于熵和前缀匹配等，这通常会导致高误报率。<sup>[3]</sup> KEYSSENTINEL 通过结合多种检测方法（机器学习、语义分析和前缀匹配）能够提高提取的准确性。<sup>[4]</sup>

构造密钥的方法有很多种，例如拼接、更长间距的上下文、特殊字符等。这些方法都会使传统方法难以识别，需要借助 LLM。<sup>[5]</sup>

### 2.3.3 密钥过滤

由于并非所有提取的字符串都代表真正的密钥，因此需要进行密钥过滤才能从提取的候选字符串中找到真正的密钥。此外，人类生成的密码给基于熵的过滤器带来了挑战，因为它们通常缺乏可识别的模式。

## 3 解题思路 1：基于开源工具的改进

### 3.1 基于开源工具的解答

我们选用了主流的开源工具 KEYSSENTINEL、Gitleaks、TruffleHog，对样本进行扫描，并对结果进行处理然后提交。其中 KEYSSENTINEL 表现最佳，误报最少。由于后面方法的答案合并了前面方法的答案，所以与只使用该方法得到的结果会不一致（为了减小提交次数没有单独测试）。

表 2 检测工具性能对比

工具名称	F1 分数
KEYSENTINEL	0.56633
Gitleaks(+KEYSENTINEL)	0.32849
TruffleHog(+KEYSENTINEL+Gitleaks)	0.13221

### 3.2 对于 KEYSSENTINEL 的分析

为了获得更高的分数，我们阅读了 KEYSSENTINEL 的论文《Hey, Your Secrets Leaked! Detecting and Characterizing Secret Leakage in the Wild》，并对 KEYSSENTINEL 的实现进行了分析。

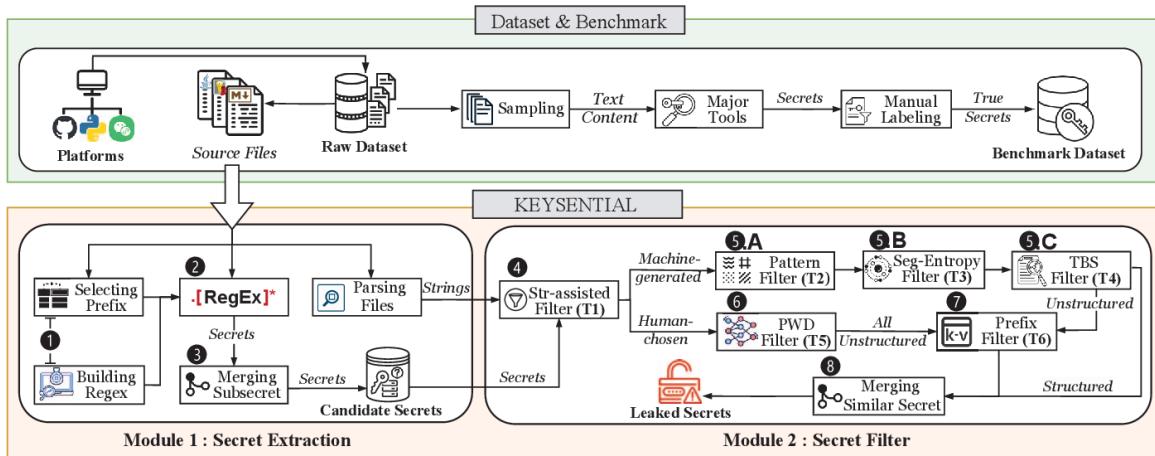


图 2 KEYSENTINEL 结构图

KEYSENTINEL 由两个主要模块组成：密钥提取和密钥过滤，包含六个过滤器（T1 到 T6）。密钥提取模块首先根据先前的研究和工具选择前缀并构建正则表达式（0）。然后，这些正则表达式用于扫描文件并提取潜在的机密（2）。之后，合并重复的密钥（3）。最后，KEYSENTINEL 提取一组候选密钥。密钥过滤模块从字符串辅助（Str-assisted）过滤器（T1）开始，该过滤器解析字符串并将其与候选密钥（0）匹配，然后根据其类别和结构进一步过滤。

机器生成的密钥经过模式过滤器（T2）、分段熵（Seg-Entropy）过滤器（T3）和基于标记化的语义（TBS）过滤器（T4）进行选择（5）。使用密码（PWD）过滤器（T5）筛选人类选择的密钥以排除非密钥（6），而使用前缀过滤器（T6）评估非结构化密钥以减少误报（7）。未通过任何筛选的密钥将标记为非机密，其余有效候选机密将按相似性合并（7）。

### 3.2.1 密钥提取

使用正则表达式用于扫描文件并提取潜在的密钥。之后，由于单个文本可能与多个模式匹配，识别最长的子字符串，并相应地合并密钥。

### 3.2.2 密钥过滤

我们对代码中各种数据文件在密钥检测中的匹配方式和用途进行了分析：

#### 1. 模式匹配

- 子字符串匹配
- 逐行读取文件中的模式
- 对每个模式检查是否在密钥值中出现（不区分大小写）

- 对于包含 PRIVATE KEY 的值，采用严格匹配（区分大小写）
- 对于其他值，采用不区分大小写的匹配

```

1 # 在 filter_patterns 函数中, 数据来自 patterns.txt
2 for pattern in patterns:
3     if ('PRIVATE KEY' in value):      # 对 private key
4         if pattern in value:          # 严格匹配
5             return True
6     else:
7         return False
8 elif pattern.lower() in value.lower(): # 不区分大小写匹配
9     return True

```

## 2. 关键词匹配

- 逐行读取 Excel 文件中的关键词
- 对每个关键词检查是否在密钥值中出现（不区分大小写）
- 忽略包含 PRIVATE KEY 或 http 的值

```

1 # 在 filter_words 函数中, 数据来自 target_words.xlsx
2 for word in words_list:
3     if word.lower() in value.lower(): # 不区分大小写匹配
4         return True

```

## 3. 英语词汇匹配

- 使用 wordninja 库分割密钥字符串为单词
- 对分割后的单词与词汇表进行精确匹配
- 计算真实单词在密钥中的占比，超过阈值则认为是有意义的文本而非密钥

```

1 # 在 count_dictionary_words 函数中, 数据来自 fixed_top_english_words_mixed_500000.json
2 words = split_word(secret_match, words_list) # 使用 wordninja 分割单词
3
4 for key in words_list:
5     if key in value_list: # 精确匹配
6         # 计算单词权重
7         if (secret['word_weight']/len(secret_match)) > 0.35:
8             have_meaning = True
9         return secret, have_meaning

```

## 4. 混淆词匹配

- 后缀匹配
- 用于键值对过滤器中
- 检查密钥是否以这些容易混淆的词结尾
- 如果密钥包含这些词且与前缀相关，则认为是合法密钥

```

1 # 在 filter_prefix 函数中, 数据来自 end_confuse_words.txt
2 for tmp in confusion_list:
3     if tmp.lower() in key_list_lower and prefix.lower() in tmp:
4         return True # 是混淆词, 过滤掉

```

## 5. 文件扩展名匹配

- 用于键值对过滤器中
- 检查密钥是否以文件扩展名结尾
- 如果是，则认为是文件路径而非密钥

```

1 # 在 key_value_filter_single 函数中, 数据来自 file_extension.txt
2 if any(key_value.endswith(ext) for ext in file_extention):
3     return False # 以文件扩展名结尾, 过滤掉

```

## 6. 合法结尾词匹配

- 包含常见的合法结尾词及其出现频率
- 用于键值对过滤器中，检查密钥结尾是否为这些合法词
- 如果是，则可能是合法的键值对而非密钥

```

1 # 加载并添加到 list_right 列表中, 数据来自 end_right_words.txt
2 list_right.extend([key for key, value in sorted_end_right_list_fasle_more20.items()])

```

## 7. 相似字符串过滤

- 判断交集是否是子串
- 判断通用类型密钥是否相似

```

1 def compare_secret_similarstr(group1, group2):
2
3 # 判断交集是否是子串——————反正都是直接用 index 来比较
4 start1, end1 = group1["index_start"], group1["index_end"]

```

```

5 start2, end2 = group2["index_start"], group2["index_end"]
6 if group1['value']==group2['value'] and start1==start2 :
7     if group2['series']=='generic':
8         return 2
9     else:
10        return 1
11 # 判断是否有交集，如果相似，如果 1 号是通用返回 1。否则返回 2（两者是不同的密钥）
12 elif start1 <= end2 and start2 <= end1:
13     if(compare_secret(group1['value'],group2['value'])):
14         if (group1['series']=='generic'):
15             return 1
16         else:
17             return 2

```

## 8. 熵值计算

机器生成的 API key、私钥因随机性高而熵值高，用户自设的通用密码熵值低。<sup>[6]</sup>据此可初步筛选密钥候选，再结合正则降低误报。

- 统计每个字符的出现次数
- 计算每个字符的概率
- 计算熵值

## 3.3 检测结果分析

为了提高检测效果，我们对检测结果进行了分析。

### 3.3.1 虚假密钥问题

虽然 KEYSENTINEL 已经考虑了这个问题，但在实际检测中，仍然有一些虚假密钥被检测出来。以下是 KEYSENTINE 检测结果中发现的一个虚假密钥。

```

1 {
2     "file": "/opt/dlami/nvme/dino/keyleak/all_files_hash/9584d6deba29095e37e34f9d35b7e354",
3     "value": "yourpsk",
4     "match": " password: yourpsk\n",
5     "prefix": "password",
6     "rule_name": "generic-password",
7     "is_mechanical": "human",
8     "series": "generic",
9     "filter_count": [],
10    "word_weight": 0,
11    "line_start": 304,
12    "line_end": 305,
13    "col_start": 2,

```

```

14 "col_end": 1,
15 "index_start": 5477,
16 "index_end": 5487,
17 "regex": "(?i)(?:[0-9a-z]{0,20})(?:[-_]{0,1})(passwd|password|pwd)(?:[0-9a-z-_\\t.]{0,20})(?:[
    ↳ |\\t|r|f|v|[ |\\t|r|f|v|]) {0,3}(?:>:{1,3}={|||:<|=|>|:||?=(?:'|\\\"|
    ↳ |\\t|r|f|v|=\\x60){0,5}{[0-9a-z-_!=!@#$%^&*+~\\({}\\})]{5,150}}(?:['\\\"|\\n|r|s|\\x60;]|$)",
18 "need_keyvalue": false
19 }

```

我们发现虚假密钥主要有两类：一类是用来存储密钥的变量名，如 key、secret、password 等，这些变量名是常见的，但不是密钥。一类将真实密钥替换成样例密钥的情况，如 example\_key、your\_key 等，这些替换为样例密钥的做法也是常见的，也不能算作密钥泄露。

虚假密钥可以通过对密钥值的关键词检测进行过滤。虽然 KEYSENTINEL 已经有类似的过滤器，但我们发现仍然有一些虚假密钥没有被过滤掉，我们通过人工标注，LLM 生成等方式增加了过滤关键词数据库，降低了此类问题的误报。

### 3.3.2 上下文提示的虚假密钥问题

在一些情况下，密钥值本身看不出问题，但其上下文提示了它并不是一个真实的密钥。例如：

```

1 {
2   "file": "/opt/dlami/nvme/dino/keyleak/all_files_hash/85cea76a95a8595495a5bf42584d91dc",
3   "value": "28P01",
4   "match": "invalid_password\" : \"28P01\"",
5   "prefix": "password",
6   "rule_name": "generic-password",
7   "is_mechanical": "human",
8   "series": "generic",
9   "filter_count": [],
10  "word_weight": 0,
11  "line_start": 154,
12  "line_end": 154,
13  "col_start": 10,
14  "col_end": 36,
15  "index_start": 6324,
16  "index_end": 6334,
17  "regex": "(?i)(?:[0-9a-z]{0,20})(?:[-_]{0,1})(passwd|password|pwd)(?:[0-9a-z-_\\t.]{0,20})(?:[
    ↳ |\\t|r|f|v|[ |\\t|r|f|v|]) {0,3}(?:>:{1,3}={|||:<|=|>|:||?=(?:'|\\\"|
    ↳ |\\t|r|f|v|=\\x60){0,5}{[0-9a-z-_!=!@#$%^&*+~\\({}\\})]{5,150}}(?:['\\\"|\\n|r|s|\\x60;]|$)",
18  "need_keyvalue": false
19 }

```

这些可能来源于登录日志，错误日志等，虽然密钥值本身看不出问题，但日志的上下文

提示了它不是一个真实的密钥，例如上面例子中的 `invalid_password`。我们可以通过增加对上下文的关键词过滤，过滤掉此类虚假密钥。

注意：我们认为此类虚假密钥的泄露是危险的，因为攻击者可以利用这些错误信息很容易的猜解出密钥，因为他们包含了使用者设置密钥的规律和部分正确的密钥值。

### 3.4 对于 KEYSSENTINEL 的改进

#### 3.4.1 正表达式数据扩充

我们查找了一些密钥检测工具的最新版本，包括 `gitleak` 和 `trufflehog` 等。我们将这些正则表达式合并去重到 KEYSSENTINEL 的正则表达式数据库中。当然，增加正则表达式的好处是提高召回率，后果是增加误报概率。我们对检测结果分析，删除了误报数量较多的正则表达式。对于误报率一般的表达式，我们可以通过增加对上下文的关键词、键值，密钥关键词等过滤机制来抑制误报率。

#### 3.4.2 密钥关键词过滤数据扩充

我们增加了密钥关键词过滤数据。我们通过参考其他密钥检测工具的数据、人工标注、LLM 生成等方式增加了过滤关键词数据库。例如 `keyvar`、`secretvar`、`yourkey`、`examplekey` 等。

#### 3.4.3 混淆词（后缀）过滤数据扩充

同样我们通过参考其他密钥检测工具的数据、人工标注、LLM 生成等方式增加了混淆词数据库。我们发现常见的混淆词主要有 `Component()`、`replace()`、`get()` 等函数调用，还有 `var`、`readme` 等词语。

#### 3.4.4 上文过滤机制增强

```

1 # 位于 def key_value_filter_single(single_hit):
2
3 # 从文件读取无效上下文列表
4 target_path = os.path.abspath(os.path.join(current_path, './data/invalid_con.txt'))
5 try:
6     invalid_contexts = load_txt(target_path)
7     #invalid_pre = [prefix.lower() for prefix in invalid_pre] # 可选：转换为小写以便比较
8 except:
9     # 如果文件不存在，使用默认列表
10    invalid_contexts = ["invalid"]
11
12 # 添加上下文过滤逻辑
13 for invalid_context in invalid_contexts:

```

```

14     if invalid_context in key_value.lower():
15         print(f"filter: invalid pre '{key_value}' in {tmp['match']}") 
16     return False

```

## 4 解题思路 2：基于 LLM 的构造密钥检测

### 4.1 背景

#### 4.1.1 为什么使用大模型

- 处理复杂模式：**构造密钥可能经过字符串拼接、分散在不同变量中或包含特殊字符，传统的正则匹配难以捕获这些复杂模式，而 LLM 能够理解代码的语义和上下文，从而更有效地识别这些构造密钥。
- 误报较少：**相比传统的正则表达式方法，基于大语言模型的零样本学习方法能够更好地理解代码上下文，结合过滤机制显著减少误报率，同时保持高召回率。

#### 4.1.2 为什么使用零样本学习

- 数据稀缺性问题：**在代码安全检测领域，特别是构造密钥的样本非常有限且难以收集。零样本学习能够在仅有少量提示词样本的情况下实现有效的检测，这对于本次安全敏感的密钥检测任务至关重要。
- 成本低适应性强：**零样本学习使模型能够快速适应新的密钥构造模式和编码风格，而无需大量的重新训练数据。

#### 4.1.3 为什么选择 Qwen2.5-Coder-7B-Instruct-AWQ 模型

- 代码专精能力：**Qwen2.5-Coder 是专门针对代码任务优化的大型语言模型，具有强大的代码理解和生成能力，特别适合分析代码中的构造密钥泄露。
- 更好地执行指令：**Instruct 模型是为遵循指令或完成特定任务而设计和优化的模型，能够更好地完成我们按照指定格式输出的指令。
- 较长上下文支持：**该模型虽然小但支持 32768 tokens 的上下文，这覆盖了大部分正常代码文件，能够捕捉到跨多行、多文件的密钥构造逻辑。
- 本地化部署优势：**AWQ 量化技术将模型压缩到 4 位，大幅减少了内存占用和计算需求，使得在本地环境中高效处理大量代码文件成为可能。实测 25 万个代码文件（因为代码文件更可能有构造密钥），4\*5090 显卡未跑满只需要 8h 左右。

- **场景适应:**

- 效率考量: 在需要扫描大量代码文件的场景下, 调用 api 的速度慢且成本较高。我们使用 7B 参数的 AWQ 量化模型提供了极佳的性能/资源平衡, 可以在有限的硬件资源下快速处理成千上万的文件。
- 隐私安全: 本地部署避免了将敏感代码上传到云端的风险, 特别适合企业级代码安全审计。
- 定制化能力: 该模型可以通过零样本学习快速适应特定项目或组织的代码风格和密钥构造模式。

## 4.2 实现方案

### 4.2.1 文件选择

构造密钥最可能出现在代码文件中, 因此我们只对代码文件进行检测, 跳过配置文件、日志文件等非代码文件, 从而提高检测效率。我们选用的模型上下文是 32768 tokens, 换算成英文字符大概 100KB( $10^5$ B) 左右, 从图中可以看出覆盖了约 97% 的文件。

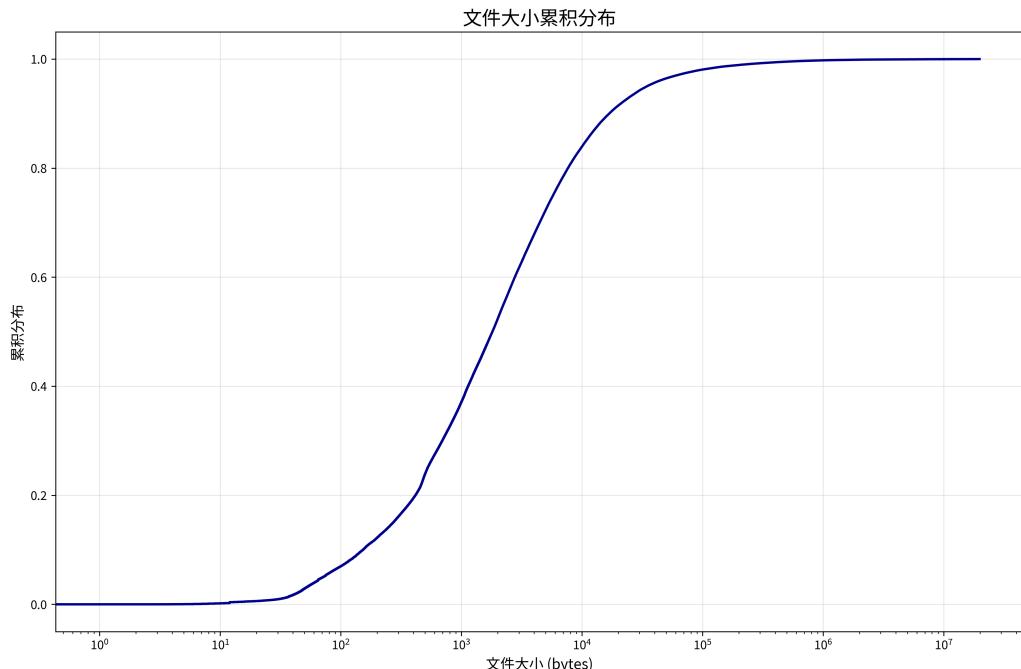


图 3 文件大小（累计）统计结果

### 4.2.2 提示工程设计

系统采用精心设计的提示模板引导模型识别复杂密钥:

```

1 system_prompt = """"
2 你是一个构造密钥泄露检测助手，你需要阅读代码并给出部分密钥可能经过拼接、更长间距的上下
   ↳ 文和特殊字符完整密钥，
3 并以 secret_key='value'的格式输出，若没有 value 可为空。例如：
4 ## 样例一
5 def connect(url, auth):
6     print(f'Connecting to {url} with auth token: {auth}')
7 a = "sk-hm6*****iob"
8 b = "xbhczx"
9 c = "bd*****bHwb"
10 secret = a + b + c
11 connect("service.example.com", secret)
12 答案是 secret_key='sk-hm6*****iobxbhczxbd*****bHwb'
13
14 ## 样例二
15 data: "secret_key=gna#3*****d&gdF4QaO&imei=0000000000000000&version=2.1.54",
16 答案是 secret_key='gna#3*****d&gdF4QaO'
17 """

```

#### 4.2.3 文件密钥检测流程

对于每个待检测文件，复用了 KEYSENTINEL 的预处理逻辑，从文件中提取出文本，然后将这些文本构造为提示词，输入到 Qwen2.5-Coder-7B-Instruct-AWQ 模型中进行分析。模型返回的结果被程序捕获并转换成结构化数据。

#### 4.2.4 多 GPU 并行处理架构

通过多 GPU 并行处理机制，系统具备高效的批量处理能力。实测 25 万个代码文件 4\*5090 显卡只需要 8h，且每个显卡的运算能力只使用了不超过 50%，说明同等算力还可以部署更多模型实现更快的速度。

- 支持多个 GPU 同时工作
- 每个 GPU 独立处理自己的文件批次
- 实时写入检测结果避免意外终止

### 4.3 实验验证

我们只取一个文件并在 content 中替换入测试内容：

```

1 content=""""
2 def api_call(endpoint, header):
3     print(666)

```

```
4 print(f'Calling {endpoint} with header {header}')
5 a='284'
6 b='356'
7 api_call("https://api.example.com/data", a+b)
8 """
```

程序运行结果 (file 是为了适应代码结构引入的，检测内容替换为了 content，可忽略)：

```
1 [
2   {"file": "/root/dino/keyleak/all_files_hash/00000a5a15ce08c6a3c13aa9ef6b2fcb",
3    "response": "secret_key='284356'",
4    "detected_keys": ["284356"]}
5 ]
```

以上结果说明 LLM 成功识别到了密钥并给出了指定格式的结果并被程序成功捕获，这是正则表达式做不到的。我们给出了部分原始检测结果在附录中，原始检测结果的 response 和程序成功提取出 value 最能证明 LLM 对构造密钥的检测的可行性。

## 参考文献

- [1] ZHOU J, ZHANG Z, YING L, et al. Hey, your secrets leaked! detecting and characterizing secret leakage in the wild[C/OL]//2025 IEEE Symposium on Security and Privacy (SP). 2025: 449-467. DOI: 10.1109/SP61157.2025.00122.
- [2] HE H, CHEN Y, HUANG X, et al. Deep learning-based channel reciprocity learning for physical layer secret key generation[J]. Security ; Communication Networks, 2022.
- [3] MELI M, MCNIECE M, REAVES B. How bad can it git? characterizing secret leakage in public github repositories[EB/OL]. 2019. DOI: 10.14722/ndss.2019.23418.
- [4] BASAK S K, COX J, REAVES B, et al. A comparative study of software secrets reporting by secret detection tools[J/OL]. CoRR, 2023, abs/2307.00714. <https://doi.org/10.48550/arXiv.2307.00714>. DOI: 10.48550/ARXIV.2307.00714.
- [5] LYKOUSAS N, PATSAKIS C. Decoding developer password patterns: A comparative analysis of password extraction and selection practices[J/OL]. Computers ; Security, 2024, 145: 103974. <https://www.sciencedirect.com/science/article/pii/S0167404824002797>. DOI: <https://doi.org/10.1016/j.cose.2024.103974>.
- [6] SAHA A, DENNING T, SRIKUMAR V, et al. Secrets in source code: Reducing false positives using machine learning[J/OL]. 2020 International Conference on COMmunication Systems ; NETworkS (COMSNETS), 2020: 168-175. <https://api.semanticscholar.org/CorpusID:212649107>.

# 附录 A 解题思路 1 关键代码

## 潜在密钥提取 (关键代码片段)-hit\_git\_tf.py

```

1 def scan_filelist_tf(rules_single, file_list, single_dict, duplicate_dict, shared_variable, log_path,
2     ↳ timeout_seconds):
3
4     logger_instance = setup_logger(log_path)
5     for file_path in tqdm(file_list):
6
7         file_value_set = set()
8         hit_num = 0
9         hit_dict = []
10        shared_variable.value += 1
11        if (shared_variable.value % 3000) == 0:
12            print("Inside the function:", shared_variable.value)
13        try:
14            signal.signal(signal.SIGALRM, handler)
15            signal.alarm(timeout_seconds)
16
17            with open(file_path, 'r', encoding='utf-8') as f:
18                content = f.read()
19
20                for rule_info in rules_single:
21                    rule_name = rule_info['rule_name']
22                    rule_pattern = rule_info['regex']
23                    try:
24                        rule = re.compile(rule_pattern)
25                    except:
26                        continue
27
28                    for match in rule.finditer(content):
29
30                        start, end = match.span(1) if rule.groups == 1 else match.span(0)
31                        if rule_info["series"] == 'jdbc' or rule_info['series'] == 'private' or rule_info[
32                            'series'] == 'uri' or rule_info['series'] == 'jwt':
33                            start, end = match.start(), match.end()
34
35                        line_start = content.count('\n', 0, start) + 1
36                        line_end = content.count('\n', 0, end) + 1
37                        col_start = start - content.rfind('\n', 0, start)
38                        col_end = end - content.rfind('\n', 0, end)
39
40
41                        if rule_info['series'] != "uri":
42                            value = match.group(1) if rule.groups == 1 else match.group(0)
43                        else:
44                            # Parsing URLs
45                            parsed_url = urlparse(match.group(0))

```

```

46     # Get username and password
47     user_info = parsed_url.username, parsed_url.password
48
49     value = user_info[1]
50     if value == None: # The detected URL has no secret——
51         ↳ https://localhost:3000/jqueryui@1.2.3
52         continue
53     prefix = ""
54     if rule_info['series'] == 'generic':
55         prefix = match.group(1)
56         value = match.group(2)
57     if len(value) == 42 and value.startswith("0x"): # Special case representing address
58         continue
59     key_value = get_sub(match.group(), value)
60     key_value = re.sub(r'^a-zA-Z]*$', "", key_value)
61     start = start + len(key_value)
62
63     if rule_info['series'] == 'private':
64         if len(value) < 128:
65             continue
66         if len(value) < 256 and 'ECC' in match.group():
67             continue
68         if len(value) < 256 and 'RSA' in match.group():
69             continue
70         if value is None:
71             continue
72         if len(value) < 5:
73             continue
74         # 对于有熵的机械密钥
75         if 'entropy' in rule_info:
76             if entropy(value) < rule_info['entropy']:
77                 continue

```

### 多因素过滤检测-filter\_multireason.py

```

1 import os
2 import sys
3 sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
4 from base_func.base_func import read_dict_bin
5
6 def check_multi_reason(hit_dict):
7     series_have = {}
8     current_path = os.path.dirname(os.path.abspath(__file__))
9     target_path = os.path.abspath(os.path.join(current_path, '../data/regex/regex_multiple_rulename.bin'))
10    regex_multiple_rulename=read_dict_bin(target_path)
11    for tmp in hit_dict:
12        if tmp['series'] in regex_multiple_rulename: # 只判断多因素
13
14            if tmp['series'] not in series_have:
15                series_have[tmp['series']] = {tmp["rule_name"]} # 将字符串转换为集合
16            else:
17                series_have[tmp['series']].add(tmp["rule_name"]) # 将字符串作为一个元素添加到集合中

```

```

18 check_false=set()
19
20 for series in series_have:
21     if series_have[series]==regex_multiple_rulename[series]:    # 说明类别集齐满了
22         check_false.add(series)
23 end_result=[]
24 suffixes=["URL", "ID", "User", "Domain"]
25 for tmp in hit_dict:
26     if tmp['series'] not in regex_multiple_rulename:
27         end_result.append(tmp)
28     else:
29         if any(char.isupper() for char in tmp['rule_name']): # 在 tf 的正则里面判断
30             if tmp['series'] in check_false and (not any(tmp['rule_name'].endswith(suffix) for suffix in
31                   ↪ suffixes)):
32                 end_result.append(tmp)
33             else:
34                 end_result.append(tmp)
35 return end_result

```

### 熵值、单词、模式三个过滤器-filter\_pattern\_word.py

```

1 import os
2 import sys
3 sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
4
5 from base_func.base_func import read_json
6 import math
7 from base_func.save_state import *
8 from base_func.logger import setup_logger
9 from tqdm import tqdm
10 import wordninja
11 import re
12 # 计算单词字符串的熵
13 def entropy(string):
14
15     prob_dict = {}
16     for character in string:
17         if character in prob_dict.keys():
18             prob_dict[character] += 1
19         else:
20             prob_dict[character] = 1
21     probs = [float(prob_dict[character]) / len(string) for character in prob_dict]
22     return - sum([prob * math.log(prob, 2) for prob in probs])
23
24 def read_dict_bin(path):
25     # 从二进制文件中加载 hit_dict
26     with open(path, 'rb') as f:
27         return pickle.load(f)
28
29 def split_string_entropy(string, length=30): # 用于拆分长的字符串计算熵_对于 machine 的并且没有
30     ↪ private
31     if len(string) < length :

```

```

31     if entropy(string)< 2.4:
32         return True
33     else:
34         return False
35     step=len(string)//length
36     str_splited=[]
37     for i in range(step):
38         start_idx = i * length
39         end_idx = (i + 1) * length if i < step - 1 else len(string)
40         batch_str = string[start_idx:end_idx]
41         str_splited.append(batch_str)
42     value_entropy=[ entropy(secret_substr) for secret_substr in str_splited]
43     if any(x < 3 for x in value_entropy):
44         return True
45     else:
46         return False
47 def split_word(secret_match,words_list):
48
49     tokens = re.split(r'[_\(\)\{\}\-\#\@\$\%\^\&\*]', secret_match)
50     tokens = [token for token in tokens if len(token)>=1] # 删除空字符串
51     # 进一步拆解
52     new_tokens = []
53     for token in tokens:
54         if token not in words_list and len(token)>3:
55             split_tokens = [split_token for split_token in wordninja.split(token)if len(split_token)>2]
56             new_tokens.extend(split_tokens)
57         else:
58             new_tokens.append(token)
59     return new_tokens
60 def count_dictionary_words_long_value(secret, words_list,secret_fixed_prefixes):
61     secret['word_weight']=0
62     secret['filter_count']=[]
63     have_meaning=False
64     secret_match=secret['value']
65
66     more5=0
67     for key in words_list:
68         if len(key)<4:
69             continue
70         if key in secret_fixed_prefixes.lower(): # 正则表达式本身可能就会出现单词
71             continue
72         if key in secret_match: # 将关键词也转换为小写形式进行比较
73             count = secret_match.count(key)
74             for i in range(count):
75                 secret['filter_count'].append(key)
76                 secret['word_weight']+=[len(key)]
77             if len(key)>=6: # 直接由长度大于 6 的单词
78                 have_meaning=True
79             return secret,have_meaning
80         if (secret['word_weight']/len(secret_match))>=0.25:
81             have_meaning=True

```

```

82         return secret,have_meaning
83     return secret,have_meaning
84 def count_dictionary_words(secret, words_list,secret_fixed_prefixes):
85     secret['word_weight']=0
86     secret['filter_count']=[]
87     have_meaning=False
88     secret_match=secret['value']
89     words=split_word(secret_match,words_list)
90
91     more4=0
92     value_list=[]
93     for word in words:
94         # if len(word)>2 and len(word)<5:
95         #     value_list.append(word)
96         # else:
97         value_list.append(word.lower())
98     for key in words_list:
99         if key in secret_fixed_prefixes.lower(): # 正则表达式本身可能就会出现单词
100             continue
101         if key in value_list: # 将关键词也转换为小写形式进行比较
102             count = value_list.count(key)
103             for i in range(count):
104                 secret['filter_count'].append(key)
105                 secret['word_weight']+=[len(key)]
106             if len(key)>4:
107                 more4+=1
108             if len(secret_match)<=30:
109                 if more4>0 :
110                     have_meaning= True
111                     return secret,have_meaning
112             elif len(secret_match)>30 and len(secret_match)<=60:
113                 if more4>1 :
114                     have_meaning= True
115                     return secret,have_meaning
116             if (secret['word_weight']/len(secret_match))>0.35:
117                 have_meaning= True
118                 return secret,have_meaning
119             return secret,have_meaning
120 def filter_pattern_word_single(single_hit,words_list, patterns, rules_single,log_path):
121     logger_instance = setup_logger(log_path)
122     secret=single_hit
123     if secret['is_mechanical']=="human":
124         return secret
125
126     if secret['series']!='private' and split_string_entropy(secret['value']):
127         return False
128
129     #print(secret)
130     rule_name = secret['rule_name']
131     secret_match = secret['value']
132     secret['word_weight']=0
133     secret['filter_count']=[]

```

```

134 secret_fixed_prefixes=""
135 for tmp in rules_single:
136     if rule_name == tmp['rule_name']: # 特殊处理 general
137         if rule_name !='generic-api-key':
138             secret_fixed_prefixes=tmp["regex"] # 正则表达式本身可能就会出现单词
139
140     # ----- patrnn 过滤 -----
141     pattern_count=[] # 对每个类别的 word_ 可以计数-----> 可以统计一下私钥之类的情况,
142     #<-- 先弄 2 个的情况
143
144 for pattern in patterns: # 这里就不需要过滤 word 了它是 pattern
145     if pattern.lower() in secret_fixed_prefixes.lower(): # 排除前后缀
146         continue
147     if pattern in secret_match: #pattern 就不要 lower 了
148         pattern_count.append(pattern)
149     if(len(pattern_count)>0):
150         secret['filter_count'].extend(pattern_count)
151     return False
152 # 加特殊情况 ----- "api': 6.5.9 _biqbaboplfbrettd7655fr4n2y\n"
153 pattern_nums = r'^\d+\.\d+\.\d+'
154 # 检查字符串是否匹配模式
155 match = re.match(pattern_nums, secret['value'])
156 if match:
157     return False
158 secret['filter_count'].extend(pattern_count)
159 # 单词过滤 -----
160 # 不能将匹配的单词打印出来 if any(word.lower() in secret.lower() for word in words_list): # 不区
161     #<-- 分大小写
162 """if len(secret['value'])>60:
163     secret,have_meaning=count_dictionary_words_long_value(secret,words_list,secret_fixed_prefixes)
164 else: """
165 secret,have_meaning=count_dictionary_words(secret,words_list,secret_fixed_prefixes)
166 if have_meaning: # 单词长度占比大于密钥的百分之三十了
167     return False
168
169 return secret
170
171 def filter_pattern_word(hit_dict,words_list,Patterns,rules_single,log_path):
172     filtered_meaning=[]
173     for tmp in tqdm(hit_dict):
174         if filter_pattern_word_single(tmp,words_list,Patterns,rules_single,log_path):
175             filtered_meaning.append(tmp)
176     return filtered_meaning
177

```

## 键值过滤器-key\_value\_Filter.py

```

1 import sys
2 sys.path.append('../') # 将上一级目录添加到系统路径中
3 # import time
4 #
5 from base_func.base_func import *
6 # import multiprocessing
7 from multiprocessing import Pool, Manager
8 from tqdm import tqdm
9 import re
10 import wordninja
11
12 def count_matching_words(input_string, filter_key):
13     count = 0
14     input_string_lower = input_string.lower() # 将输入字符串转换为小写形式
15
16     for key in filter_key:
17         if key.lower() in input_string_lower: # 将关键词也转换为小写形式进行比较
18             count += 1
19     return count
20
21 def filter_prefix(key_list, left_spited_list, right_spited_list, confusion_list, list_left, list_right, prefix): # 前缀
    ↳ 拆词，前缀左拆、前缀右拆，混淆、左右白名单
22     #-----key 混淆-----
23     key_list_lower=[tmp.lower() for tmp in key_list]
24     confusion_list=[tmp.lower() for tmp in confusion_list]
25     for tmp in confusion_list:
26         if tmp in key_list_lower and prefix.lower() in tmp: # 后面的 and 是需要确认哪个 prefix 命中的
            ↳ 这个只能用于混淆
27         return True
28     #-----key 左边过滤-----
29     list_left_lower=[tmp.lower() for tmp in list_left]
30     left_spited_list_lower=[tmp.lower() for tmp in left_spited_list]
31     for tmp in list_left_lower:
32         if tmp in left_spited_list_lower: # 左边
33             return True
34     #-----key 右边过滤-----
35     list_right_lower=[tmp.lower() for tmp in list_right]
36     right_spited_list_lower=[tmp.lower() for tmp in right_spited_list]
37     for tmp in list_right_lower:
38         if tmp in right_spited_list_lower: # 右边
39             return True
40     return False
41 def split_words(secret_match, word_list):
42
43     tokens = re.split(r'[-\s]', secret_match)
44     tokens = [token.rstrip() for token in tokens if len(token)>=2] # 删除空字符串
45     # 进一步拆解
46     new_tokens = []
47     for token in tokens:
48         if token not in word_list and len(token)>3:
49             split_tokens = [split_token for split_token in wordninja.split(token)if len(split_token)>=2]

```

```

50     new_tokens.extend(split_tokens)
51 else:
52     new_tokens.append(token)
53 return new_tokens
54 def split_string_by_word(s, word):
55     if not word: # Check if the word is empty
56         return s, "" # Return the input string and empty strings
57     if word in s:
58         parts = s.split(word)
59         left_part = parts[0]
60         right_part = word.join(parts[1:])
61         return left_part, word, right_part
62     else:
63         return s, ""
64
65 def get_sub(a,b):
66     # 查找 b 在 a 中的起始位置
67     start_index = a.find(b)
68     # 如果 b 是 a 的子串
69     if start_index != -1:
70         # 提取 b 之前的那部分字符串
71         return a[:start_index]
72     else:
73         # 如果 b 不是 a 的子串, 可以设置一个默认值或抛出异常
74         return a
75
76 def key_value_filter_single(single_hit):
77     current_path = os.path.dirname(os.path.abspath(__file__))
78     target_path = os.path.abspath(os.path.join(current_path,
79                                     './data/fixed_top_english_words_mixed_500000.json'))
80     word_list=read_json(target_path)
81     end_filtered=[]
82     whole_value_list=["templateid:", "@tmp", "@example",'example','@test',
83                         "@hostname",'test:test','example.com','@somewhere','_key','@url','hello:world@','*****@localhost',
84                         "127.0.0.1:443@"]
85     value_placeholder=["changeit", "changeme", "change", "guest", "printf", "return", "test",
86                         "user",'pass',"password", "username",
87                         "secret",'test','bar','foobar','api_secret','TOKEN','pwd','password','apikey',"Parola", "Parolan",
88                         "Wachtwoord", "Salasana", "Pasahitzta", "boolean", "Lozinka", "before", "blahblah"]
89     value_machine=["ca-pub-"] # 机械密钥的
90     test_path=["test", "example", "demo"]
91     list_right=[ "Oid", "author", "hash", "Fingerprint", "keyword", "checksum", "addr", "type", "id"]
92     target_path = os.path.abspath(os.path.join(current_path,
93                                     './data/sorted_end_right_list_fasle_more20.json'))
94     sorted_end_right_list_fasle_more20=read_json(target_path)
95     list_right.extend([key for key,value in sorted_end_right_list_fasle_more20.items()])
96     list_left=[ 'PublicKey','public','fake','input','put','enter','invalid']
97     target_path = os.path.abspath(os.path.join(current_path, './data/end_confuse_words.txt'))
98     confusion_list=load_txt(target_path)

```

```

92 whole_key_list=["formkey", "?key", "h1", "GPG key", "SAPageKey", "pubkey", "key_hex",
93     ↪ "fake_",'Password_label','Password_action','Password_msg',"git-tree", "_id", "
94     ↪ id",'sha256','-sha','sha1',
95     ↪ h1','keyword','apiUsername','PublicKey','public','.js','.py','.yaml','_type','_addr','.txt']
96 target_path = os.path.abspath(os.path.join(current_path, './data/file_extension.txt'))
97 file_extention=load_txt(target_path)
98 #从文件读取无效上下文列表
99 target_path = os.path.abspath(os.path.join(current_path, './data/invalid_pre.txt'))
100 try:
101     invalid_contexts = load_txt(target_path)
102     #invalid_pre = [prefix.lower() for prefix in invalid_pre] #转换为小写以便比较
103 except:
104     #如果文件不存在，使用默认列表
105     invalid_contexts = ["invalid"]
106
107
108 tmp=single_hit
109 flag=False
110
111 #if tmp['series']=='generic'or tmp['match']!=tmp['value']:    #只有当key有别的语义时
112 if tmp['need_keyvalue']== True and tmp['match']!=tmp['value']:          #暂时只对通用密钥有效
113     key_value=get_sub(tmp['match'],tmp['value'])
114     key_value = re.sub(r'^a-zA-Z]*$', "", key_value) #去除非字母数字字符
115     if any(key_value.endswith(ext) for ext in file_extention):
116         print(f"filter:{tmp['match']}") #添加上下文过滤逻辑
117         return False
118     for invalid_context in invalid_contexts:
119         if invalid_context in key_value.lower():
120             print(f"filter: invalid pre '{key_value}' in {tmp['match']}") #除了generic的他们可能没有prefix,但是可能也有前缀
121             return False
122     key_list=split_words(key_value,word_list)
123     left_spited_list=split_words(pre_left,word_list)
124     right_spited_list=split_words(pre_right,word_list)
125     if
126         ↪ filter_prefix(key_list, left_spited_list, right_spited_list, confusion_list, list_left, list_right, tmp['prefix'])
127         ↪ or count_matching_words(key_value,whole_key_list) or
128         ↪ count_matching_words(tmp['value'],value_machine):
129             print(f"filter:{tmp['match']}") #除了generic的他们可能没有prefix,但是可能也有前缀
130             return False
131 elif tmp['match']!=tmp['value']:          #除了generic的他们可能没有prefix,但是可能也有前缀
132     if 'password' in tmp['rule_name'] or 'Jdbc' in tmp['rule_name'] or 'Uri' in tmp['rule_name'] or 'jwt' in
133         ↪ tmp['rule_name']:
134             pass
135     else:
136         key_value=get_sub(tmp['match'],tmp['value'])
137         key_value = re.sub(r'^a-zA-Z]*$', "", key_value) #添加上下文过滤逻辑
138         if any(key_value.endswith(ext) for ext in file_extention):
139             print(f"filter:{tmp['match']}") #除了generic的他们可能没有prefix,但是可能也有前缀
140             return False

```

```

136 if tmp['is_mechanical'] == 'human':
137     if 'password' in tmp['rule_name'] or 'Jdbc' in tmp['rule_name'] or 'Uri' in tmp['rule_name']:
138         #password
139         count_match = count_matching_words(tmp['value'], value_placeholder)
140         count_match += count_matching_words(tmp['match'], whole_value_list)
141         #count_match += count_matching_words(tmp['file'], test_path)
142         if count_match >= 1:
143             print(f"filter: {tmp['value']} ")
144             return False
145
146         split_value = re.split(r'\W+', tmp['value'])  # 拆分单词，过滤常见密钥
147         result_value = [s for s in split_value if len(s) >= 3]
148         if flag:
149             return False
150
151     else:  # 其它链接数据库的密钥
152         count_match = count_matching_words(tmp['value'], value_placeholder)
153         count_match += count_matching_words(tmp['match'], whole_value_list)
154         #count_match += count_matching_words(tmp['file'], test_path)
155         if count_match >= 2:
156             print(f"filter: {tmp['value']} ")
157             return False
158
159
160
161 def multiprocess(batch, share_dict):
162     for tmp in tqdm(batch):
163         if key_value_filter_single(tmp):
164             share_dict.append(tmp)
165

```

## 附录 B 解题思路 2 关键代码

### 使用 LLM 检测复杂密钥-llmdetect.py

```

1 import os
2 import sys
3 import json
4 import torch
5 from transformers import AutoModelForCausalLM, AutoTokenizer
6 from base_func.base_func import get_files
7 import multiprocessing
8 from multiprocessing import Pool
9 import argparse
10 from tqdm import tqdm
11 from joblib import Parallel, delayed
12
13 # 初始化模型
14 def init_model(device_id=0):
15     model_name = "Qwen/Qwen2.5-Coder-7B-Instruct-AWQ"
16
17     # 指定特定的 CUDA 设备
18     device_map = {"": f"cuda:{device_id}"}
19
20     try:
21         model = AutoModelForCausalLM.from_pretrained(
22             model_name,
23             torch_dtype=torch.float16,
24             device_map=device_map
25         )
26     except Exception as e:
27         print(f"使用 device_map 加载模型时出错: {e}")
28         print("尝试使用传统方式加载模型...")
29     # 如果 device_map 方式失败，使用传统方式加载
30     model = AutoModelForCausalLM.from_pretrained(
31         model_name,
32         torch_dtype=torch.float16,
33         low_cpu_mem_usage=True
34     )
35     model = model.to(f"cuda:{device_id}")
36
37     tokenizer = AutoTokenizer.from_pretrained(model_name)
38     return model, tokenizer
39
40 # 检测单个文件中的复杂密钥
41 def detect_complex_keys_in_file(file_path, model, tokenizer, output_file=None):
42     try:
43         with open(file_path, 'r', encoding='utf-8') as f:
44             content = f.read()
45     except Exception as e:
46         try:

```

```

47     with open(file_path, 'r', encoding='latin-1') as f:
48         content = f.read()
49     except Exception as e:
50         with open(file_path, 'rb') as f:
51             content = f.read().decode('utf-8', errors='ignore')
52
53     # 如果文件太大, 只取部分内容
54     max_content_length = 100000 #131,072 个上下文 token 约 100k 代码
55     # content="""def api_call(endpoint, header):
56     #     .....
57     #     print(f"Calling {endpoint} with header {header}")
58
59     # api_call("https://api.example.com/data", "sec*****ret")"""
60     content = content[:max_content_length]
61
62     system_prompt = """
63 你是一个构造密钥泄露检测助手, 你需要阅读代码并给出部分密钥可能经过拼接、更长间距的上
64 ↳ 下文和特殊字符完整密钥, 并以 secret_key='value'的格式输出, 若没有 value 可为空。例如:
65 ## 样例一
66 def connect(url, auth):
67     print(f"Connecting to {url} with auth token: {auth}")
68     a = "sk-hm6*****iob"
69     b = "xbhczx"
70     c = "bd*****bHwb"
71     secret = a + b + c
72     connect("service.example.com", secret)
73     答案是 secret_key='sk-hm6*****iobxbhczxbd*****bHwb'
74
75 ## 样例二
76 data: "secret_key=gna#3*****d&gdF4QaO&imei=0000000000000000&version=2.1.54",
77     答案是 secret_key='gna#3*****d&gdF4QaO'
78     """
79     prompt=f"""
80     请检测:
81     {content}
82     需要你根据理解告诉我密钥是多少, 不要编写代码, 按 secret_key='value'的格式输出, 若找不到
83     ↳ value 可为空。
84     """
85     messages = [
86         {"role": "system", "content": system_prompt},
87         {"role": "user", "content": prompt}
88     ]
89     text = tokenizer.apply_chat_template(
90         messages,
91         tokenize=False,
92         add_generation_prompt=True
93     )
94
95     model_inputs = tokenizer([text], return_tensors="pt", truncation=True).to(model.device)
96

```

```

97     generated_ids = model.generate(
98         **model_inputs,
99         max_new_tokens=512
100    )
101
102    generated_ids = [
103        output_ids[len(input_ids):] for input_ids, output_ids in zip(model_inputs.input_ids, generated_ids)
104    ]
105
106    response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0]
107    #print(content[:200], '>>>', response)
108    # 解析响应中的密钥
109    if "secret_key=" in response:
110        result = {
111            "file": file_path,
112            "response": response,
113            "detected_keys": extract_keys_from_response(response)
114        }
115
116        # 实时写入结果到文件
117        if output_file and len(result["detected_keys"]) > 0:
118            with open(output_file, 'a', encoding='utf-8') as f:
119                f.write(json.dumps(result, ensure_ascii=False) + '\n')
120
121    return result
122
123    return None
124
125 # 从 LLM 响应中提取密钥
126 def extract_keys_from_response(response):
127     keys = []
128     lines = response.split('\n')
129     for line in lines:
130         if "secret_key=" in line:
131             # 提取密钥值
132             try:
133                 key_part = line.split("secret_key=", 1)[1]
134                 if "" in key_part:
135                     key_value = key_part.split("", 1)[1].split("", 1)[0]
136                 elif "" in key_part:
137                     key_value = key_part.split("", 1)[1].split("", 1)[0]
138                 else:
139                     key_value = key_part.strip()
140                     if key_value and key_value != "" and key_value != "":
141                         keys.append(key_value)
142             except:
143                 pass
144     return keys
145
146 # 从 src_filenames.txt 读取文件列表
147 def get_source_files_from_txt(txt_path, base_path="/root/dino/keyleak/all_files_hash"):
148     """

```

```

149 从 src_filenames.txt 读取文件名并构造成完整路径
150 """
151 try:
152     with open(txt_path, 'r', encoding='utf-8') as f:
153         filenames = [line.strip() for line in f.readlines() if line.strip()]
154
155     # 构造完整路径
156     full_paths = [os.path.join(base_path, filename) for filename in filenames]
157
158     # 过滤掉不存在的文件
159     existing_files = [path for path in full_paths if os.path.exists(path)]
160
161     print(f"从 {txt_path} 读取到 {len(filenames)} 个文件名")
162     print(f"其中 {len(existing_files)} 个文件在 {base_path} 中存在")
163
164     return existing_files
165 except Exception as e:
166     print(f"读取文件列表时出错: {e}")
167     return []
168
169 # 处理文件批次
170 def process_file_batch(file_batch, device_id, output_file_prefix):
171     """
172     在指定的设备上处理一批文件，每个设备写入独立的文件
173     """
174     # 为每个 GPU 创建独立的输出文件
175     output_file = f"{output_file_prefix}_{device_id}.json"
176
177     # 初始化文件
178     with open(output_file, 'w', encoding='utf-8') as f:
179         f.write('[')
180
181     results = []
182     # 为每个批次初始化一次模型
183     model, tokenizer = init_model(device_id)
184
185     # 使用 tqdm 显示进度条
186     for file_path in tqdm(file_batch, desc=f"GPU {device_id} 处理进度", leave=False):
187         try:
188             result = detect_complex_keys_in_file(file_path, model, tokenizer, output_file)
189             if result and len(result["detected_keys"]) > 0:
190                 results.append(result)
191         except Exception as e:
192             print(f"处理文件 {file_path} 时出错: {e}")
193
194     # 完成后关闭 JSON 数组
195     with open(output_file, 'a', encoding='utf-8') as f:
196         f.write(']')
197
198     # 清理模型以释放内存
199     del model
200     del tokenizer

```

```

201 torch.cuda.empty_cache()
202
203 return results
204
205 # 将列表划分为 n 份
206 def split_list(lst, n):
207     """
208     将列表划分为 n 个子列表
209     """
210     k, m = divmod(len(lst), n)
211     return [lst[i*k + min(i, m):(i+1)*k + min(i+1, m)] for i in range(n)]
212
213 # 多 GPU 并行处理函数
214 def process_files_with_multiple_gpus(file_list, num_gpus=4, output_file=None):
215     """
216     使用多个 GPU 并行处理文件
217     """
218     # 将文件列表划分为 num_gpus 个子列表
219     file_batches = split_list(file_list, num_gpus)
220
221     # 使用 joblib 并行处理，但确保每个进程独立加载模型
222     results = Parallel(n_jobs=num_gpus, backend="loky")(
223         delayed(process_file_batch)(file_batch, i, output_file)
224         for i, file_batch in enumerate(file_batches)
225     )
226
227     # 展平结果列表
228     flattened_results = []
229     for batch_result in results:
230         flattened_results.extend(batch_result)
231
232     return flattened_results
233
234 # 主函数
235 def main():
236     parser = argparse.ArgumentParser(description='使用 LLM 检测复杂密钥')
237     parser.add_argument('--path', dest='path', type=str, help='要检测的文件根路径',
238                         default="/root/dino/keyleak/all_files_hash")
239     parser.add_argument('--size_limit', dest='size_limit', type=int, help='文件大小限制', default=100 *
240                         1024 * 1024)
241     parser.add_argument('--output', dest='output', type=str, help='输出文件路径',
242                         default="./complex_keys_results.json")
243     parser.add_argument('--num_gpus', dest='num_gpus', type=int, help='使用的 GPU 数量', default=4)
244
245     args = parser.parse_args()
246
247     print(f'正在扫描路径: {args.path}')
248     print(f'使用 GPU 数量: {args.num_gpus}')
249
250     # 从 src_filenames.txt 读取文件列表
251     txt_file_path = "/root/dino/keyleak/src_filenames.txt"
252     if os.path.exists(txt_file_path):

```

```

250     file_list = get_source_files_from_txt(txt_file_path, args.path)
251     print(f"从 src_filenames.txt 读取到 {len(file_list)} 个文件")
252 else:
253     print(f"未找到 {txt_file_path}, 使用默认文件列表")
254     file_list = get_files(args.path, args.size_limit)
255
256     print(f"实际处理文件数: {len(file_list)}")
257
258 # 初始化输出文件, 创建 JSON 数组开始
259 with open(args.output, 'w', encoding='utf-8') as f:
260     f.write('[')
261
262     # 使用多个 GPU 并行处理文件
263     print("正在使用多 GPU 并行处理文件...")
264     results = process_files_with_multiple_gpus(file_list, args.num_gpus, args.output)
265
266
267     print(f"检测完成, 找到 {len(results)} 个可能包含复杂密钥的文件")
268     print(f"结果已保存到: {args.output}")
269
270 if __name__ == '__main__':
271     main()

```

## LLM 检测部分原始结果

```

1 [
2 {"file": "/root/dino/keyleak/all_files_hash/401f1475d0ccb35d216c0e226d6fd1c", "response":
3     "secret_key='3q^83^86Xo'u>\n3È1B)", "detected_keys": ["3q^83^86Xo'u>"]},
4 {"file": "/root/dino/keyleak/all_files_hash/403f0383412139230632ac8662400966", "response":
5     "secret_key='zbLsuDDL4CS2U0M4KezOZZbGUY9iWtVf", "detected_keys":
6     ["zbLsuDDL4CS2U0M4KezOZZbGUY9iWtVf"]},
7 {"file": "/root/dino/keyleak/all_files_hash/405103674df7b7e0e60f8a60cea6cd21", "response":
8     "```plaintext\nsecret_key='sk-12345'\n``", "detected_keys": ["sk-12345"]},
9 {"file": "/root/dino/keyleak/all_files_hash/413f92bf719b14b74c80cd20b429d079", "response": "secret_key='secret'",
10    "detected_keys": ["secret"]},
11 {"file": "/root/dino/keyleak/all_files_hash/41c379ae899736528cba437426c0146f", "response":
12     "secret_key='cEvewMJFX4OwbsYVaMt2Os6GxxGgDUIBGILs2RySgoeoNKE61ygbz1vhaCVynGERaRrlviPB...",
13     "detected_keys":
14     ["cEvewMJFX4OwbsYVaMt2Os6GxxGgDUIBGILs2RySgoeoNKE61ygbz1vhaCVynGERaRrlviPB..."]},
15 {"file": "/root/dino/keyleak/all_files_hash/41d797089874bc076d485b082125b9b4", "response":
16     "```plaintext\nsecret_key='devkey, should be in a file'\n``", "detected_keys": ["devkey, should be in a file"]},
17 {"file": "/root/dino/keyleak/all_files_hash/41fc027952382be8e50b37fbb406fc0", "response":
18     "secret_key='1234567890:AAAABBBBCCCCDDDDDeeeeFFFFgggGHHHH'", "detected_keys":
19     ["1234567890:AAAABBBBCCCCDDDDDeeeeFFFFgggGHHHH"]},
20 {"file": "/root/dino/keyleak/all_files_hash/4265795394c4c84ab33f94dbbe2384a5", "response":
21     "```plaintext\nsecret_key='TEST_AWS_ACCESS_KEY_ID'\nsecret_key='TEST_AWS_SECRET_ACCESS_KEY'\n``",
22     "detected_keys": ["TEST_AWS_ACCESS_KEY_ID", "TEST_AWS_SECRET_ACCESS_KEY"]},
23 {"file": "/root/dino/keyleak/all_files_hash/42f7992f8347ad853370a6d5ba03a55e", "response": "secret_key='Bearer
24     6c6adc18-22a9-929f-648d-786eb20ebcf4'", "detected_keys": ["Bearer 6c6adc18-22a9-929f-648d-786eb20ebcf4"]},
25 {"file": "/root/dino/keyleak/all_files_hash/4306851994e0f2ae1148e441b1a8e06", "response":
26     "secret_key='optional_api_key'", "detected_keys": ["optional_api_key"]},
27 {"file": "/root/dino/keyleak/all_files_hash/4383b599581de437407e7a59bc52c98f", "response":
28     "secret_key='TWILIO_ACCOUNT_SID=your_account_sid&TWILIO_AUTH_TOKEN=your_auth_token'",
29     "detected_keys": ["TWILIO_ACCOUNT_SID=your_account_sid&TWILIO_AUTH_TOKEN=your_auth_token"]},

```

```

13 {"file": "/root/dino/keyleak/all_files_hash/43abb93487d59b57cef6501225486832", "response": "secret_key='abc'",  
→ "detected_keys": ["abc"]},  
14 {"file": "/root/dino/keyleak/all_files_hash/442198909b99dd7852b0ee0ef1b0de30", "response":  
→ "secret_key='ANTHROPIC_API_KEY'", "detected_keys": ["ANTHROPIC_API_KEY"]},  
15 {"file": "/root/dino/keyleak/all_files_hash/447536c01954b19ca00e405acc5a0e55", "response":  
→ "secret_key='K2T6k1eRNwH3kwNqiXk51aSelzqBTdN0-4K0XhsAZ0'", "detected_keys":  
→ ["K2T6k1eRNwH3kwNqiXk51aSelzqBTdN0-4K0XhsAZ0"]},  
16 {"file": "/root/dino/keyleak/all_files_hash/44c377c2741b19ea9ad204b8d363d244", "response":  
→ "secret_key='92db86.....'", "detected_keys": ["92db86....."]},  
17 {"file": "/root/dino/keyleak/all_files_hash/44df6ca85a9ce095d4e4d34ed11d00cd", "response":  
→ "secret_key='azffw7zSQU6KJq4DrWrwhw'", "detected_keys": ["azffw7zSQU6KJq4DrWrwhw"]},  
18 {"file": "/root/dino/keyleak/all_files_hash/44f03bbcc1f1a8ac9fb1a6af25ea66b8", "response":  
→ "secret_key='fake-super-secret-token'", "detected_keys": ["fake-super-secret-token"]},  
19 {"file": "/root/dino/keyleak/all_files_hash/458a0cf26a6962cfe8e8a721bff0d848", "response":  
→ "secret_key='phc_oENDjGgHtmIDrV6puUiFem2RB4JA8gGWulfdulmMdZP'", "detected_keys":  
→ ["phc_oENDjGgHtmIDrV6puUiFem2RB4JA8gGWulfdulmMdZP"]},  
20 {"file": "/root/dino/keyleak/all_files_hash/45a4d1a0423421eab8c5a4c104f85213", "response":  
→ "secret_key='E6wRbVhD0IBeCiGJ'", "detected_keys": ["E6wRbVhD0IBeCiGJ"]},  
21 {"file": "/root/dino/keyleak/all_files_hash/45d340767418e3edadda558e42f8a8c3", "response":  
→ ```python\nsecret_key='cNo1Fekr1kVEWcAw4P2Gg6MXWsRBtYy5W8idzMojifCgkpPfLMfj'\n```",  
→ "detected_keys": ["cNo1Fekr1kVEWcAw4P2Gg6MXWsRBtYy5W8idzMojifCgkpPfLMfj"]},  
22 {"file": "/root/dino/keyleak/all_files_hash/4608957577dc59fbe3b86aacb8f6ce20", "response":  
→ "secret_key='yasserp:your_bintray_api_key_here'", "detected_keys": ["yasserp:your_bintray_api_key_here"]},  
23 {"file": "/root/dino/keyleak/all_files_hash/46734730c6ad06a943fb7121d54ebdfe", "response":  
→ "secret_key='0lhozfib5410mp'", "detected_keys": ["0lhozfib5410mp"]},  
24 {"file": "/root/dino/keyleak/all_files_hash/48a7cf52f561a939de1363a1bb48595f", "response":  
→ "secret_key='KwnbpKJNe6gOjC2X5ilxaFFvxNppiIfGejB2h1Y'", "detected_keys":  
→ ["KwnbpKJNe6gOjC2X5ilxaFFvxNppiIfGejB2h1Y"]},  
25 {"file": "/root/dino/keyleak/all_files_hash/48e73717b47fba305124d2e60940fc2", "response":  
→ ```plaintext\nsecret_key='secret0key000000'\n```, "detected_keys": ["secret0key000000"]},  
26 {"file": "/root/dino/keyleak/all_files_hash/49059c3b73176d45a6633768084721d7", "response":  
→ "secret_key='eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9."  
→ eyJpc3MiOiJzdXBhYmFzZSIsInJlZi6ImhocXBwY3FsdGtsemZwZ2dkb2NiIwicm9sZSI6ImFub24iL...",  
→ "detected_keys": ["eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9."],  
→ eyJpc3MiOiJzdXBhYmFzZSIsInJlZi6ImhocXBwY3FsdGtsemZwZ2dkb2NiIwicm9sZSI6ImFub24iL..."],  
27 {"file": "/root/dino/keyleak/all_files_hash/49409bc5f5c875360ecbb8b82bfe29eb", "response": "secret_key='AdmSbl8'",  
→ "detected_keys": ["AdmSbl8"]},  
28 {"file": "/root/dino/keyleak/all_files_hash/495065977bcb8b252161bc07ef9672c2", "response":  
→ "secret_key='MicroPyth0nRules'", "detected_keys": ["MicroPyth0nRules"]},  
29 {"file": "/root/dino/keyleak/all_files_hash/49661bea739ec82b6add5d44a914c386", "response":  
→ "secret_key='NDE0MzIyMDQ1MzA0OTYzMDcy.DWl2qw.nTxSDf9wIcf42te4uSCMuk2VDA0'", "detected_keys":  
→ ["NDE0MzIyMDQ1MzA0OTYzMDcy.DWl2qw.nTxSDf9wIcf42te4uSCMuk2VDA0"]},  
30 {"file": "/root/dino/keyleak/all_files_hash/497354f448cc7c85289cab169b525f94", "response": "secret_key='this is my  
secret key'", "detected_keys": ["this is my secret key"]},  
31 {"file": "/root/dino/keyleak/all_files_hash/4995efa29190cd6745032fa70594c208", "response": "secret_key='prosieben'",  
→ "detected_keys": ["prosieben"]},  
32 {"file": "/root/dino/keyleak/all_files_hash/49cb17549b1eed7bd89732461042669c", "response": "secret_key='This is my  
little secret ^_~'", "detected_keys": ["This is my little secret ^_~"]},  
33 {"file": "/root/dino/keyleak/all_files_hash/49d73f3f23e553e8796feb5aca29fa2", "response":  
→ "secret_key='wzNMApO6LpLdqSSS45Q~TjKeA'", "detected_keys": ["wzNMApO6LpLdqSSS45Q~TjKeA"]},  
34 {"file": "/root/dino/keyleak/all_files_hash/4a0512787c49e6af98b0b4a6eb03a2d4", "response": "secret_key='Bearer  
6c6adc18-22a9-929f-648d-786eb20ebcf4'", "detected_keys": ["Bearer 6c6adc18-22a9-929f-648d-786eb20ebcf4"]},  
35 {"file": "/root/dino/keyleak/all_files_hash/4bee821a3e45938daf27087df141f6c5", "response":  
→ "secret_key='6LeWLCYeAAAAAL1caYzkrIY-M59Vu41vIblXQZ48'", "detected_keys":  
→ ["6LeWLCYeAAAAAL1caYzkrIY-M59Vu41vIblXQZ48"]},  
36 {"file": "/root/dino/keyleak/all_files_hash/4c57e854351a969664dde3e99b480563", "response":  
→ "secret_key='09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7'", "detected_keys":  
→ ["09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"]},  
37 {"file": "/root/dino/keyleak/all_files_hash/4c6d1849771d851cb8082076852815ae", "response":  
→ "secret_key='YOUR_API_KEY'", "detected_keys": ["YOUR_API_KEY"]},

```

```

38 {"file": "/root/dino/keyleak/all_files_hash/4c703257410b4ffbb77630d3c62f145e", "response":
→ "secret_key='smtp_login=your_smtp_login&smtp_password=your_smtp_password'", "detected_keys":
→ ["smtp_login=your_smtp_login&smtp_password=your_smtp_password"]},
39 {"file": "/root/dino/keyleak/all_files_hash/4da0c864729b6f7939fdcc62a0edf288", "response": "secret_key='1234'", "detected_keys": ["1234"]},
40 {"file": "/root/dino/keyleak/all_files_hash/4e8f369dfee8101cdbc8364a75659a64", "response":
→ "secret_key='PUVKp9WgGUb3-JUw6EqafLx8tFVP6VKZTbUOR-HOm__g4fNDt1bCsm_LgYf_k9H'", "detected_keys": ["PUVKp9WgGUb3-JUw6EqafLx8tFVP6VKZTbUOR-HOm__g4fNDt1bCsm_LgYf_k9H"]},
41 {"file": "/root/dino/keyleak/all_files_hash/50557d10cd4bb6955ab2092634a94552", "response":
→ "secret_key='my-minio-key:OSMM+vkKUTCvQs9YL/CVMIMt43HFhkUpqJxTmGl6rYw=='", "detected_keys": ["my-minio-key:OSMM+vkKUTCvQs9YL/CVMIMt43HFhkUpqJxTmGl6rYw=="]},
42 {"file": "/root/dino/keyleak/all_files_hash/508dacb228ffb81a85d044ebddd46c0", "response":
→ "secret_key='FakeExamplePassword'", "detected_keys": ["FakeExamplePassword"]},
43 {"file": "/root/dino/keyleak/all_files_hash/50a881fe7d177a743e739d22a2c2e95a", "response": "secret_key='Bearer 6c6adc18-22a9-929f-648d-786eb20ebcf4'", "detected_keys": ["Bearer 6c6adc18-22a9-929f-648d-786eb20ebcf4"]},
44 {"file": "/root/dino/keyleak/all_files_hash/50cba45c1ec4038fa5e196ee8512f7cf", "response":
→ "```plaintext\nsecret_key='DJANGO_WEROBOT_TOKEN'\n```", "detected_keys": ["DJANGO_WEROBOT_TOKEN"]},
45 {"file": "/root/dino/keyleak/all_files_hash/50d288d2e6c766a04927160e60573ef6", "response":
→ "secret_key='yh^%ZA^$Xh+%FxUv%NucZ(N+POPN%'", "detected_keys": ["yh^%ZA^$Xh+%FxUv%NucZ(N+POPN%"]},
46 {"file": "/root/dino/keyleak/all_files_hash/513f7a51bf645064d4024966cc65a266", "response":
→ "secret_key='eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.\neyJDbGllbnRfSUQiOixIiwiQWNjZXNzIjoiZUZudmdBT0NvZjB5em1zUXZ4WjcfQ.\n9J6_xKNG-6L_RnJS7Woq20BZRZpUboBMQknSX7E0YJE'", "detected_keys": ["eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.\neyJDbGllbnRfSUQiOixIiwiQWNjZXNzIjoiZUZudmdBT0NvZjB5em1zUXZ4WjcfQ.\n9J6_xKNG-6L_RnJS7Woq20BZRZpUboBMQknSX7E0YJE"]},
47 {"file": "/root/dino/keyleak/all_files_hash/5154311c42c5265e3fe1d3b210a6556b", "response":
→ "secret_key='coneofsilence'", "detected_keys": ["coneofsilence"]},
48 {"file": "/root/dino/keyleak/all_files_hash/520899916818546f76aa90e5cde9736b", "response":
→ "secret_key='X@IAIgafLvMSnSnBx()UiRb&().GglNDzqHVlZUgggMZhX^IK'", "detected_keys": ["X@IAIgafLvMSnSnBx()UiRb&().GglNDzqHVlZUgggMZhX^IK"]},
49 {"file": "/root/dino/keyleak/all_files_hash/532411b7fadaa33c7267556581f2e0ae", "response":
→ "secret_key='fuiKNFp9vQFvjLNvx4sUwti4Yb5yGutBN4Xh10LXZhhRKjWIV4'", "detected_keys": ["fuiKNFp9vQFvjLNvx4sUwti4Yb5yGutBN4Xh10LXZhhRKjWIV4"]},
50 {"file": "/root/dino/keyleak/all_files_hash/53c147b5d65d59f789c8d530b10cb5ad", "response":
→ "secret_key='ABCSECRETKEY'", "detected_keys": ["ABCSECRETKEY"]},
51 {"file": "/root/dino/keyleak/all_files_hash/53dc66ef6b2b8db8c69ca538a77a5622", "response":
→ "secret_key='ODIN_KEY_ID=your_value_here ODIN_SECRET_KEY=your_value_here'", "detected_keys": ["ODIN_KEY_ID=your_value_here ODIN_SECRET_KEY=your_value_here"]},
52 {"file": "/root/dino/keyleak/all_files_hash/5431200468659dd07a61d0f91470f237", "response":
→ "```plaintext\nsecret_key='367e9b21fef313f187026320016962b47b74ca4ada7d64d551c43c51e195d7a5'\n```", "detected_keys": ["367e9b21fef313f187026320016962b47b74ca4ada7d64d551c43c51e195d7a5"]},
53 {"file": "/root/dino/keyleak/all_files_hash/5443fae5b04b8dece593952b9e7c600b", "response":
→ "secret_key='zEgTKP4vzVI0aWE3sC1EsMybKkSPHH'", "detected_keys": ["zEgTKP4vzVI0aWE3sC1EsMybKkSPHH"]},
54 {"file": "/root/dino/keyleak/all_files_hash/5506edd0d0a72791b72ea693e9a6e667", "response":
→ "secret_key='settings.YANDEX_CAPTCHA_SERVER_KEY'", "detected_keys": ["settings.YANDEX_CAPTCHA_SERVER_KEY"]},
55 {"file": "/root/dino/keyleak/all_files_hash/55b9927be42265bbbd617f1141a7157a", "response":
→ "secret_key='dtcon_ZnAoXhZcPtnobnVCLSgg'", "detected_keys": ["dtcon_ZnAoXhZcPtnobnVCLSgg"]},
56 {"file": "/root/dino/keyleak/all_files_hash/563f36484c2c661216271e422bdd60ec", "response": "secret_key='sekrit'", "detected_keys": ["sekrit"]},
57 {"file": "/root/dino/keyleak/all_files_hash/565dd158d48faf21cdd47c1a58e2bbce", "response":
→ "secret_key='fake-super-secret-token'", "detected_keys": ["fake-super-secret-token"]},
58 {"file": "/root/dino/keyleak/all_files_hash/568d38644e8b0d21a66c027c6f0efd3a", "response": "secret_key='ru9-JsUrWiNLjWpLT0LeNoZzagirpTtFVa2yj9jIHv8='", "detected_keys": ["ru9-JsUrWiNLjWpLT0LeNoZzagirpTtFVa2yj9jIHv8="]},

```

```

59 {"file": "/root/dino/keyleak/all_files_hash/5759847844cd6cef86e249040cef5bd3", "response":
  ↪ "secret_key='qSQLiHLwStBWT3nncdrN241UgdZdPM5H'", "detected_keys":
  ↪ ["qSQLiHLwStBWT3nncdrN241UgdZdPM5H"]},
60 {"file": "/root/dino/keyleak/all_files_hash/5843faed6478236d9f5f457e0eafbc", "response": "secret_key='secret'",
  ↪ "detected_keys": ["secret"]},
61 {"file": "/root/dino/keyleak/all_files_hash/58b7011927f105c10af88a2d5dd1d497", "response":
  ↪ "secret_key='e05dabf6fe0e-2c18-abf4-496d-1d010490'", "detected_keys":
  ↪ ["e05dabf6fe0e-2c18-abf4-496d-1d010490"]},
62 {"file": "/root/dino/keyleak/all_files_hash/58c8b8fa82408d9f39fd5a11800c1e5c", "response":
  ↪ "secret_key='h4s4bZMtjz1kJbJmj44Vip#ivQDrS.H6NTg0'", "detected_keys":
  ↪ ["h4s4bZMtjz1kJbJmj44Vip#ivQDrS.H6NTg0"]},
63 {"file": "/root/dino/keyleak/all_files_hash/58ce191d6df1fb490c33ac9b34232ba7", "response": "secret_key='asdf'",
  ↪ "detected_keys": ["asdf"]},
64 {"file": "/root/dino/keyleak/all_files_hash/5901c60a88351708811a7530380f2774", "response": "secret_key='faketoken'",
  ↪ "detected_keys": ["faketoken"]},
65 {"file": "/root/dino/keyleak/all_files_hash/592068642fae41970891ba50b722681b", "response":
  ↪ "```plaintext\nsecret_key='some-long-long-secret-only-for-wtforms-string-be-brave-if-you-use-it-on-prod'\n``",
  ↪ "detected_keys": ["some-long-long-secret-only-for-wtforms-string-be-brave-if-you-use-it-on-prod"]},
66 {"file": "/root/dino/keyleak/all_files_hash/594a010e7863fd07dfad770416d480ce", "response":
  ↪ "secret_key='09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7'", "detected_keys":
  ↪ ["09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"]},
67 {"file": "/root/dino/keyleak/all_files_hash/5994a445f56474371824cb8f8c9eec01", "response": "secret_key='0x20, 0x15'",
  ↪ "detected_keys": ["0x20, 0x15"]}]
68

```