GPT Agent in Taint Analysis: A New Era of Cybersecurity Solutions

Ziyu Chen
University of Adelaide
harriet.ziyu@gmail.com

ABSTRACT

This research investigates the application of Language Models (LLMs), such as ChatGPT, in enhancing code security and data privacy through taint analysis. A customized AI agent, named "Data Leakage Detector" [5], has been developed using a novel Progressive Agent Creation Through Evaluation process, demonstrating LLMs effectiveness in identifying potential data leakages and security breaches in code. This agent is tailored and evaluated using the DroidBench [2] Android test suite and surpasses the performance of benchmark tool, achieving an F1 score around 0.98. As the GPT store becomes available, this customized GPT agent is freely accessible to everyone, enabling more straightforward and user-friendly interactions. With its specialized capabilities in taint analysis, users can simply present their code to the agent for analysis, reports or clarifications. This study highlights LLMs' versatility as this agent is applicable not only to Android but also to other environments, paving the way for broader applications of LLMs in cybersecurity solutions.

1. INTRODUCTION

Code security and privacy are integral to the software development lifecycle, with static analysis playing a pivotal role in identifying potential vulnerabilities through processes like taint analysis [2]. Taint analysis is crucial for detecting data leakages, security, and privacy issues by tracking data from tainted sources to vulnerable positions (sinks), thereby aiding in the development of robust and secure applications. Static analysis, including taint analysis, has been a focus of previous research [12][17], revealing application potentials of LLMs by examining static behaviors like control flow graphs, call graphs, data dependency, and pointer analysis. However, current static analysis tools like MobSF [1] and FlowDroid [2] often encounter challenges due to lack of context or domain knowledge, resulting in high false positive rates.

The advent of Large Language Models such as ChatGPT signifies a major advancement in AI, notably in processing and generating human-like text. These models have opened new avenues in software development and cybersecurity, exhibiting proficiency in code generation [14][16], vulnerability detection [4], and bug fixing [15][25]. Previous studies employed various techniques to improve code security using LLMs, including fine-tuning LLM parameters, integrating it with tools [10], and refining prompting techniques [10][15]. In this study, the motivation for investigating ChatGPT's potential in enhancing taint analysis was driven by the results obtained in their earlier research.

This study explores the use of ChatGPT in taint analysis for data leakage detection, aiming to investigate its potential in improving vulnerability detection and data leakage detection. This study delves into the field of prompt engineering, employing techniques like chain-of-thought [19], task decomposition [3], one-shot [7][9] to further assist ChatGPT. A range of prompt templates was developed to assess and enhance ChatGPT's effectiveness on an Android test suite. This approach concentrated on examining how various prompts influenced ChatGPT's performance compared to its baseline response behavior, and it also tested ChatGPT's domain knowledge in taint analysis. Additionally, an innovative method called Path Backtracking was introduced as an approach for reviewing the results, which significantly decreased the occurrence of both false positives and false negatives.

This study created the GPT agent "Data Leakage Detector," a customized AI agent designed for comprehensive taint analysis in the industry environment, Achieving an F1 score exceeding 97%, it surpasses the taint analysis tool FlowDroid that the test suite DroidBench is built for, highlighting the capability of LLMs in static analysis and data leakage detection. The forthcoming GPT store aims to enhance public access to this tailored agent, potentially leading to increased efficiency in professional environments. This study confirms ChatGPT's significant high level of context awareness and domain knowledge on Android and taint analysis compared to the tools made in conventional ways. Without any coding practice involved, Data Leakage Detector's proficiency in diverse taint analysis scenarios and its human-like analytical approach represent a potential shift from conventional analysis tools due to its high adaptability and ease of customization to consistently achieve the desired results. This agent's adaptability and flexibility is highly useful because the definitions of the positive taint results may vary depending on the needs of the development process.

Prior to the customized AI agents GPTs [13], using ChatGPT to build tools or applications requires integrating the GPT API, often requires the development of a UI. However, the GPT store allows for the

publication and exchange of customized GPTs, streamlining the process of tool creation and user interaction. These GPT agents may contain instructions (prompts), knowledge, and actions (calling APIs), can be customized and even integrate external data and interact with other web services while maintaining their user-friendly interfaces just like ChatGPT. This advancement indicates that not only the GPT agent in this study comes automatically with a default UI, but it also marks significant shift towards using natural language and data, supported by prompting engineering techniques, as a new toolmaking and development method.

However, due to the indeterministic nature of LLMs and its evolving nature, the future performance of ChatGPT may vary upon new updates. Despite this, it is highly estimated that in future ChatGPT's ability on static analysis would be similar if not better than current status.

In summary, this study contributes in the following ways:

I. Developed a customized GPT agent tailored for taint analysis, achieving an impressive F1-measure score of approximately 0.98, surpassing the benchmark tool.
II. Introduced Progressive Agent Creation Through Evaluation (PACTE) to create AI agents that help transit code analysis tasks from conventional tools to GPTs.
III. Proposed a Path Backtracking method to enhance the overall performance of taint analysis. Crafted various prompt templates to explore ChatGPT's capabilities in domain-specific knowledge, including the detection of taint flow, taint sources, and taint sinks.
IV. Validated the potential of Large Language Models in static analysis and underscores the evolving role of AI and natural language in tool development and code security.

2. AIMS AND CHALLENGES

The objective of this study is to design a GPT agent that outperforms traditional tools. The primary goal is to create a range of prompts to determine the necessary domain knowledge for GPT and to identify specific taint analysis scenarios that require greater attention. To carry out these experiments, the study leverages DroidBench, an Android taint analysis test suite, to guide the process of crafting instructions or prompts for the GPT agent to conduct an evaluation to assess the agent's performance.

2.1 Test Suite and Model Selection

Taint analysis can be useful for security and data privacy, particularly in addressing privacy leakage, permission compliance, and privacy policy compliance [18]. Many static analysis tools target a specific language or platform often face issues with large numbers of false positives and may struggle to retrieve sufficient context or knowledge for extensive tasks. Static analysis is more efficient than dynamic analysis, but often encounters challenges like path exploration and a lack of contextual or domain knowledge. Most benchmarks or test suites categorize various scenarios to evaluate a tool's effectiveness in various scenarios. DroidBench is a dataset developed for the tool FlowDroid, targeting Android environments. It consists of small Android applications for evaluating taint analysis and can be utilized for prompt template creation and guide the agent development process.

One advantage of this approach is that the test suite can refine prompts and evaluate them. Each conversation with ChatGPT is isolated because ChatGPT does not remember previous interactions. Therefore, if capable of specific tasks, it should generate relatively stable results, regardless of whether it has evaluated the same data or code before. This is because prompting methods do not participate in training the language models, so the prompt generation process using the test suite does not interfere with the results. This holds true as long as the prompts are not targeting any specific code or application as might occur in a scenario of overtraining, which can be controlled by agent owners, unlike the black-box LLMs.

Although one test suite may be sufficient in this study, the chosen test suite itself may not be perfect. The test categories in DroidBench might lack scenarios other frameworks may face, and without tailoring, the GPT agent might not demonstrate its full potential. Nevertheless, this resulting agent is still universal and could be effective in various environments as it was built upon the default GPT-4 model, and the Android domain knowledge it contains could serve as one-shot scenario examples (no code) for other languages or frameworks. This was later proven by the results of the baseline prompt template, which also achieved an F1 score above 0.9. However, its specificity metric indicates difficulty in detecting negative results, leading to potential false positives if not tailored for more specific domain knowledge.

This experiment requires time-consuming efforts with multiple runs across various test suite categories. It demands substantial efforts to collect results and generate detailed statistics corresponding to various scenarios and experimenting with the potential improvements. The token limit and usage cap for GPT-4 may hinder the experiment, thus the experiment involves both GPT-3.5 and GPT-4 as GPT-3.5 is free on ChatGPT, making it attractive if the two shows similar performance. On the other hand, both GPT-3.5 and GPT-4 are indeterministic and may produce different results for the same prompts, necessitating the use of GPT-4 for a thorough understanding of ChatGPT capabilities.

2.2 Domain Knowledge and Prompt Creation

Helping Large Language Models achieve their potential is a substantial concept, involving multiple aspects of prompt engineering, including zero-shot, one-shot, chain-of-thought, etc. It has led to more complex concepts such as GPT agent. One challenge is identifying suitable methods for this study. Considering time constraints, this research focuses on basic prompting methods like chain-of-thought and one-shot, applied to LLMs for taint analysis. This is because taint analysis is assumed to have a relatively predictable workflow and straightforward approach, despite the complexity of its various definitions or the various scenarios in different frameworks and languages may present, which falls into the realm of domain knowledge.

Domain knowledge is crucial as it determines the content of the prompts. However, providing LLMs with knowledge they already possess may not be as problematic as giving them an insufficient amount of knowledge, provide it does not mislead them. In this study, this may involve high-level knowledge such as code examples, potential locations or names of taint sources and sinks, or general knowledge on how to conduct taint analysis and what constitutes acceptable taint flow. Challenges reside in determining the amount of domain knowledge ChatGPT requires, the type of knowledge needed (one-shot, knowledge structures like call graphs, keyword lists, etc.), and whether too much prompting information could overwhelm ChatGPT. For prompt template creation, it is trivial to test and prompt ChatGPT to ensure it performs as expected. It is worth exploring whether ChatGPT needs other static analysis approaches, such as call graphs or control flow graphs for taint analysis, as they imply its ability to trace sensitive data.

Challenges also may reside in the order and structure of the prompt instructions for the code, whether ChatGPT requires any knowledge of programming languages or development frameworks, and can it understand the context of the code. To address these issues, at least three prompts should be created to check ChatGPT’s knowledge level and its needs. Conducting a thorough evaluation is a time-consuming task. This challenge arises because for each run, each set of prompts serves as a template applicable to numerous cases, with each potentially containing multiple instances of both positive and negative data leakages that need to be detected, and different templates are used to prompt the code in ChatGPT, although ChatGPT can detect multiple leakage in one time. In the GPT agent setup, the most effective prompt template is integrated into the agent’s configuration with a set of instructions, serving the same intended purpose.

A notable challenge is understanding how to categorize positive and false positive cases. While it may seem straightforward, it is rather complicated. For instance, some cases involve bad coding practices that could eventually introduce data leakage, but under the current circumstances they don't leak anything, either due to the sensitivity category of the data, under different application configurations or other scenarios, such as running the code in a simulator, which might trigger unexpected behaviors. This is a challenge traditional static analysis tools face, but GPT might have a broader context understanding to easily spot issues. On the other hand, some code may have a taint flow that, due to certain configurations, is not active. Sometimes tant analysis tools might need to check the Android Manifest file, but ChatGPT might still choose to report any taint flow as a risk, which may differ from the expected outcome of the test suite, or vice versa. Nevertheless, either outcome might be acceptable in real-life scenarios, considering the benefits of good coding practices versus the bug bounty hunting or penetration testing process. It is intriguing to see if GPT can generate the expected outcome according to various instructions, potentially offering more flexibility than traditional tools.

2.3 Streamline the Process

This study involves time-consuming experiments across multiple categories, which limits the number of comprehensive evaluations that can be conducted on the test suite for each prompt template. Due to time constraints, the number of prompt templates created and tested is also limited. While it's not feasible to explore numerous prompt templates for all taint analysis cases, the study can establish a default framework with three templates. For instance, the initial template might be designed without any domain knowledge, while the final one could incorporate as much domain knowledge as possible, provided it does improve the results.

At least one thorough evaluation should be conducted during the experiment. However, this study decides on two comprehensive evaluations, unless there are no failed cases that could benefit from further improvement with a second evaluation. After the initial creation and refinement of templates, the first comprehensive evaluation of all three templates should allow for a further assessment of failed cases. This helps determine whether an updated prompt might mislead ChatGPT, potentially leading to poorer results than the previous results. A subsequent reevaluation of all cases could offer further opportunities to address any issues, especially if the second evaluation does not yield the anticipated results after the integration of updated prompts.

Another aspect to consider is the review step for the analysis. In other studies [10], a second run was employed for reviewing purposes. However, when attempting to reproduce their experiments on GPT-4, it appeared that ChatGPT often changed its answers

without thorough analysis, likely unsure about the answers. In this study, an alternative method needs to be established specifically for taint analysis. Detailed explanations regarding the method are provided in the Methodology section.

3 RELATED WORK

Static analysis including static analysis has received significant attention in recent research, as detailed in [12]. Their study explores ChatGPT's impact on various static behaviors, including control flow graph, call graph, data dependency, taint analysis, and pointer analysis. Taint analysis is particularly notable for its emphasis on data flow or data dependency, and its implication on control flow and call graph. Taint analysis is dedicated to tracking the influence of sensitive input or external data on a program's variables, which is crucial for identifying potential security vulnerabilities [12]. While static analysis is commonly employed in code reviews to detect bugs and security issues, taint analysis is particularly valuable for addressing security and privacy concerns [2].

The tracing of untrusted data by taint analysis proves advantageous in several contexts. For instance, it plays a key role in ensuring compliance with Android privacy policies and preventing the leakage of sensitive information. This aspect of taint analysis is vital for safeguarding user privacy, making it an indispensable tool in the field of software development and cyber security areas such as bug bounty programs.

3.1 Static Analysis and LLMs

Static analysis and taint analysis are pivotal tools in software engineering, instrumental in debugging code and assessing the relationships between components in a code. Language Learning Models have demonstrated proficiency in a variety of tasks, including code generation [16], static analysis, vulnerability detection, bug fixing [8], and code explanation. However, these models face limitations, such as issues with hallucination [20].

To mitigate these limitations, researchers have been investigating methods to enhance LLM performance. This includes supplementing LLMs with external tools like static analysis tools [10], integrating domain knowledge into LLMs, and fine-tuning their parameters. These strategies have improved LLMs' capabilities. For example, Google has reported enhanced productivity in software development when pairing software engineers with LLMs [4]. Despite concerns about the reliability of LLM-generated code, particularly in terms of security [14], the combined use with other techniques has shown promise in enhancing code security. This is achieved through prompt engineering [9] [15], which aids in static analysis by reducing false positives, detecting vulnerabilities, and correcting bugs [15], thereby contributing to software security during development.

Given the challenges of relying solely on LLMs for automatic static analysis, various approaches have been proposed to enhance code security. These include fine-tuning LLM parameters, augmenting LLMs with additional methods, supplementing LLMs with tools, and refining prompting techniques. One such method is prefix-tuning [11], which involves adding a prefix to input sequences in prompts as a continuous task-specific vector [11], instead of changing the LLM's original parameters. LLMs show promise in aiding secure code generation, reducing false positives from static analysis results, and fixing vulnerabilities. Integrating these proof-of-concept methods into the secure software development process is an ongoing challenge. Recently, GPT-4 has been enhanced with new features, including data analysis capabilities, and upgraded to GPT-4 Turbo. This development has motivated our study to embark on a journey exploring prompting methods that could enable the latest ChatGPT to effectively assist in static analysis, especially taint analysis.

Secure Code Generation LLMs are used for code generation to enhance development efficiency in a study [11], the results may also imply LLMs' ability to understand code behaviors. However, a substantial portion of their output includes unreliable code. To fully utilize the potential of LLMs, there is a need to generate not only correct but also secure code. Methods such as the cost-effective prefix-tuning approach [11] may augment LLMs without modifying or fine-tuning their original parameters [11].

Reduce False Positives A study [10] investigated ChatGPT's effectiveness in enhancing static analysis. This involved using a static analysis tool that often encounters issues with many false positives in its reports. However, their study shows that incorporating ChatGPT has the potential to yield more accurate summaries, thereby aiding in the reduction of these false positives [10]. This is achieved by their task-decomposition technique and progressive prompting methods.

Considering the diverse approaches highlighted in these papers, this study delves into investigating the specific prompts and essential knowledge needed to enhance ChatGPT's performance. The goal is to fully harness the capabilities of LLMs and contribute to further advancements in the field.

3.2 Android Taint Analysis with FlowDroid

FlowDroid is a taint analysis tool specifically designed for the Android platform. Its test suite DroidBench [2] encompasses various categories of Android applications, each featuring unique and complex test cases. These cases are focused on aspects such as field or object

sensitivity, Android-specific challenges, application's lifecycle and UI interactions. These test cases are not only invaluable for static analysis experiments but also essential for guiding the creation of prompts in taint analysis, while simultaneously aiding in assessing ChatGPT's domain knowledge regarding the Android environment.

Key components of Android framework include the concepts of Activities and Intents [2]. An Activity is a single screen with a user interface where user interactions occur. Intents, meanwhile, are messaging objects used to request actions from other app components. However, scenarios where an Intent may not be received as expected, or an Activity may not be active, can affect the application's data flow. The relationships of these components with the Android Manifest file, which declares the application's essential characteristics and its components, is crucial. The Manifest file details the activities, services, broadcast receivers, and content providers used by the application [2].

Understanding scenarios where and why Activities are not active (such as the configurations in Manifest file), or Intents are not received is crucial for accurately modeling the data flow of an Android application and its data dependencies in taint analysis. FlowDroid's role in this context is to employ techniques that enhance analysis capabilities. Concepts such as aliasing [2] and data dependency [12], which are part of data flow analysis, are particularly crucial in taint analysis. Understanding the taint flow through different elements, based on data dependency, is vital for tracking the spread of tainted data.

FlowDroid's thorough approach in taint analysis its effectiveness in identifying data leakage, is demonstrated by its impressive F1 score of 0.89 on DroidBench. This high score indicates a strong balance between precision and recall in its analysis, making it a relatively reliable tool for detecting potential data leakage in Android applications. FlowDroid's methodology, which likely involves advanced static analysis techniques, is thus validated as both efficient and accurate in the context of Android security. However, this study surpassed its performance, showcasing an even better approach to taint analysis with AI agent that enhances the detection of data leakage beyond Android environments.

3.3 LLMs Prompt Engineering

Prompt engineering, the process of formulating specific queries to extract desired responses from LLMs, is integral to maximizing the efficacy of these models and enhancing user-LLM interactions. Effective prompts are typically clear, concise, and contextual, often incorporating explicit constraints [7]. Various strategies, such as specifying formats, experimenting with different wordings, and using examples in one-shot, few-shot, or chain-of-thought formats, can improve outcomes [7].

Previous studies, such as [10], suggested a review step for improving prompts, but experiments on GPT-3.5 proved to be rather unstable compared to GPT-4. In this study, a novel approach called Path Backtracking for the review process is introduced. This method enables ChatGPT to trace back from the sink to the source upon identifying positive results, and both GPT-3.5 and GPT-4 observed reduced false positives while maintaining true negatives. This approach has shown improved outcomes in certain scenarios.

It is common to leverage external knowledge bases or paraphrase instructions in prompting process, as different wordings may yield varied responses. Providing additional context or explicit intentions in prompts can also be effective, though overloading with information might be counterproductive. One study [15] indicates that one-shot prompts are often adequate, and additional examples don't necessarily enhance performance, hence the decision to use no more than one example per prompt in this experiment. Tools including LLMs can be developed for prompt optimization and generation, but this study does not delve into this direction.

Another study employs task decomposition [3], using LLMs in hardware design to write Hardware Description Language (HDL) by segmenting the design into smaller tasks with predefined conversations and benchmark challenges. These experiments demonstrate that LLM errors can be addressed through various levels of human feedback, from identifying syntax errors to proposing corrective methods. However, in their study, the choice of LLM is crucial, as different models can yield diverse responses to the same prompt.

This study primarily focuses on the definition of taint analysis as used in FlowDroid and did not delve into the broader spectrum of prompt engineering techniques. This is because ChatGPT performs very well using just basic prompting techniques such as chain-of-thought.

3.4 GPT Store and GPTs

The GPT Store is going to be a marketplace offering a variety of customized GPTs [13] referred to as GPT agent in this study. It provides optimized agent made by ChatGPT users for diverse applications such as creative writing, technical troubleshooting, and the Data Leakage Detector created in this study. A key advantage of the GPTs is their high degree of customization, allowing users to select models that best fit their specific needs, thus enhancing the accessibility and applicability of LLMs across different fields.

The GPT agent accepts pre-defined system prompts as instructions, which form its primary configuration method. This approach makes the agent particularly approachable for users who are not deeply versed in LLMs, as it can retrieve facts from uploaded knowledge and process data efficiently. These agents may be designed for seamless integration with external systems using API calls, promoting easy implementation in both professional and personal environments. This represents a significant advancement in making sophisticated AI tools more accessible and user-friendly. This study focuses on instructions and knowledge as means to configure it.

However, the advancement of this mechanism brings with it security concerns, including issues related to API security and the risk of adversarial attacks. Despite their capabilities, LLMs can face certain limitations, such as potential biases, their non-deterministic nature, and inaccuracies. Regardless, as the GPT Store going to the public, this study is publishing the customized GPT agent [new 8][new 9], and its convenient features will be open to test.

4. METHODOLOGY

This study proposes a new framework to smooth the AI agent creation process using prompts or instructions of LLMs in a cost-efficient way, using GPT agent “Data Leakage Detector” as an example.

4.1 Progressive Agent Creation Through Evaluation (PACTE)

Progressive Agent Creation Through Evaluation (Figure 1) is a strategic and cost-efficient framework designed for developing GPT-based AI agents, particularly focused on code analysis and addressing complex scenarios often found in test suites or benchmarks. Considering the upcoming GPT store, it could be an evolving time when people may choose to develop GPT agents that outperform current state-of-the-art applications, tools or web services. By evaluating their performance using the same test suite their creation based on without misleading the agent, there's an opportunity to see if this novel process can create a GPT agent that can replace current less-performing tools, marking a significant advancement in the field.

Pilot Study

The PACTE process initiates with a pilot study that contrasts the capabilities of different models, in this study, GPT-3.5 and GPT-4. The initial choice of GPT-3.5 is driven by its cost-effectiveness, allowing for the development of initial prompts without financial constraints. GPT-4, while potentially more advanced, is initially set aside due to its usage limitations for extensive evaluation tasks. This stage is crucial for assessing the general domain knowledge of the GPT models concerning code analysis tasks.

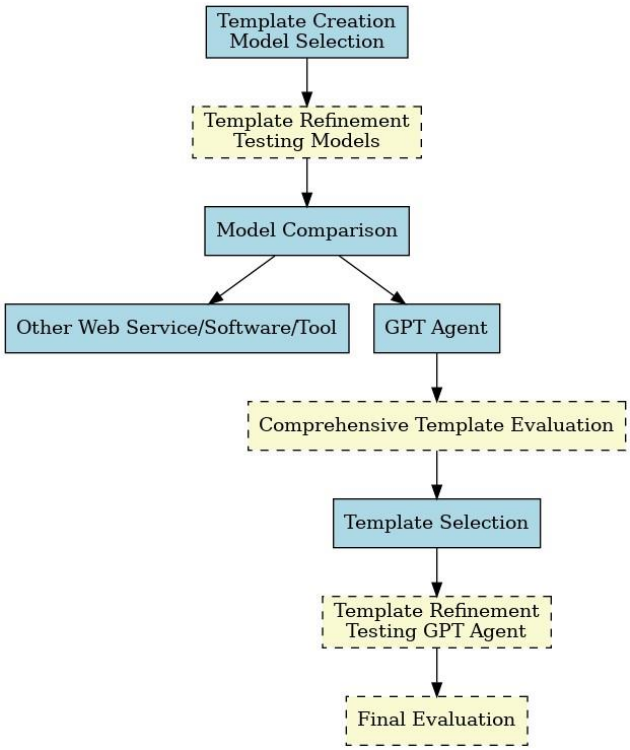

Figure 1. PACTE workflow

During this process, PACTE involves creating approximately three distinct prompt templates to evaluate each model's proficiency in intricate code analysis tasks. Each template is imbued with domain knowledge, escalating from the least in the basic template to the most in the advanced template. This gradation ensures a rough exploration of the LLMs' capabilities across varying levels of complexity. These templates are designed to test the LLMs' level of domain knowledge and ascertain whether specific definitions and guidance can yield improved results. This step aims to establish a baseline capability for the LLMs and explore various prompt modifications to enhance their performance.

In this study, the decision to continue with GPT-3.5 or transition to GPT-4 hinges on the development goals. If the aim is to create a tool or software where cost-effectiveness is paramount, GPT-3.5 might be retained as using the GPT-3.5 API is cheaper. If GPT-4 demonstrates significantly superior performance in these initial tests, the framework may opt to forgo a more extensive evaluation with GPT-3.5 and proceed directly with GPT-4. For building a sophisticated AI agent, PACTE inherently progresses towards employing GPT-4, as GPT-3.5 is unable to create a GPT agent.

Phase I

Entering Phase I of the main study of PACTE, the focus shifts to a comprehensive evaluation of the established prompt templates. The phase involves rigorous testing across all test cases from the test suites or scenarios. It is here that the templates, initially developed and refined during the pilot study, are thoroughly assessed. This stage is critical for meticulously assessing the prompt templates developed during the pilot study. The prompt template that demonstrates the best performance is then selected for the next stage; it will be used to configure the instructions for the AI Agent.

Phase II

In the second phase of the main study, the focus shifts to create and test an agent, with experiments involving the refinement of the selected template or agent instructions. This process was enhanced by incorporating specific domain knowledge and learning from previous unsuccessful cases, in this study, various scenarios regarding Android taint analysis. This step is crucial for addressing all failed cases identified in the first phase. This phase needs to review these failed cases and endeavor to enhance the templates or agent instructions by integrating more knowledge for specific scenarios, complementing the general knowledge previously embedded in the chosen prompt template. Another round of evaluations is then conducted on the newly updated instructions to verify their effectiveness across a broad spectrum of test categories. This approach aims to develop a more comprehensive and well-rounded AI agent.

4.2 GPT Agent: Data Leakage Detector

This section introduces an agent named “Data Leakage Detector,” used for taint analysis and data leakage detection. It was created using the techniques mentioned in the previous section.

Prompt Template Design

For taint analysis, three templates are designed and will be tested and evaluated using the Android Taint Analysis test suite DroidBench, which contains more than 120 cases across multiple categories (excluding a few cases with unclear ground truths). The basic template does not include domain knowledge, merely asking ChatGPT to perform analysis on given Android code, with output in JSON format, as shown in Figure 2. This basic prompt template serves as a baseline for evaluating ChatGPT’s capabilities in taint analysis.

From this basic template, two more templates are developed, named the medium template (A.2) and the advanced template (A.3). The medium prompt aims to assess ChatGPT’s ability to identify the source and sink in taint analysis, crucial for determining the start and end points of taint flow. It includes two files listing potential sources and sinks to aid in detecting them. The advanced prompt builds on the medium template by adding more domain-specific details to enhance performance. The best-performing template will be selected as the prototype for creating the GPT agent named “Data Leakage Detector,” as illustrated in Figure 1.

```
<code>
Using the information in the code, identify all the potential data leakages and their locations, report in JSON format, the answer can be "yes" or "no", format is [{taint source:<taint source name>, taint sink: <taint sink name>, taint status:<"yes" or "no", if yes show taint path>}, ...].
```

Figure 2. The basic prompt template for “Data Leakage Detector”

Engineering the Advanced Template

Unlike the other templates, the advanced prompt requires preliminary experimentation before finalization. This template involves the general process of performing taint analysis and should initially be tested on a few categories before applying it to all cases. It includes a review process as shown in Figure 3.

Initially, it defines the role and goal for ChatGPT, incorporating general techniques and a static analysis process to aid taint analysis. Then, it introduces the main process of taint analysis, beginning with identifying taint sources and tracing their flow. This process involves finding aliasing instances and following the taint flow to relevant instances according to data dependencies.

The template also addresses object sensitivity issues in taint analysis, a key task for the “Field and Object Sensitivity” test category from DroidBench. It aims to determine if taint analysis tools can differentiate elements of an object or different fields of a class. This is critical because while an element can transfer its taint status to its constituent object, other elements should not inherit this taint status, as their involvement in other program processes does not necessarily carry the taint flow. Like the other templates, it continues to use lists of sources and sinks as guides. The JSON format is utilized to indicate all potential taint flows for identified sources and sinks.

Finally, a review step is added to evaluate the newly found taint flows, introducing a technique called Path Backtracking. This aims to reduce false positives in the initial results. ChatGPT should regenerate the report after this evaluation.

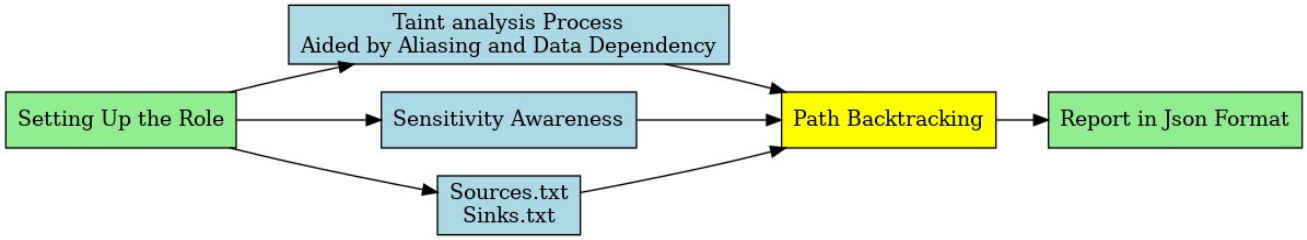

Figure 3. The advanced prompt template for GPT agent "Data Leakage Detector"

Path Backtracking

In the pilot study, the review process checks for any taint flow leading back to the source from the sink. However, after the Template Refinement step of Phase II in the main study, it is updated to include more specific scenarios, complemented by example scenarios from Android environments.

Instead of providing comprehensive Android domain knowledge, this step focuses exclusively on various scenarios of Android taint analysis. It enhances the advanced template with more details (Figure 4):

a) If the initial results are negative, it checks for any implicit or indirect flow. For instance, if there is an indirect flow, such as Boolean conditions, that might continue the flow to reach the sink using sensitive data, this scenario is considered a positive data leak. This implicit flow aims to identify scenarios like revealing password information; for example, if a condition checks whether a password matches and sends the result to the sink, it is classified as a positive case. This step is motivated by the Implicit flow category of DroidBench.

b) If the initial report shows positive results, it assesses whether the taint flow is active. In Android, GPT may need to check the Manifest file to determine if an activity is active or examine multiple functions to ascertain whether a relevant intent is correctly received, enabling the data leakage. If not, the taint flow is considered inactive. This step defines 'Active Taint Flow' for GPT, motivated by the Inter-App Communication test category of DroidBench.

However, in real-life situations without a benchmark guiding the process, definitions can vary. The labeling of cases as negative or positive depends on the specific needs of the users or agent creators. It is possible to consider any taint flow as a positive case, regardless of its reachability, to improve code practices and prevent future security issues, which seems to be the default approach in ChatGPT, unlike the specific definitions we are establishing here.

In addition, the lists of sinks and sources serve as optional supplementary knowledge for the agent, enabling it to decide whether to utilize them for information retrieval. This approach is adopted because the agent shows less optimally performance with the medium template, compared to the basic prompt, which serves as the baseline.

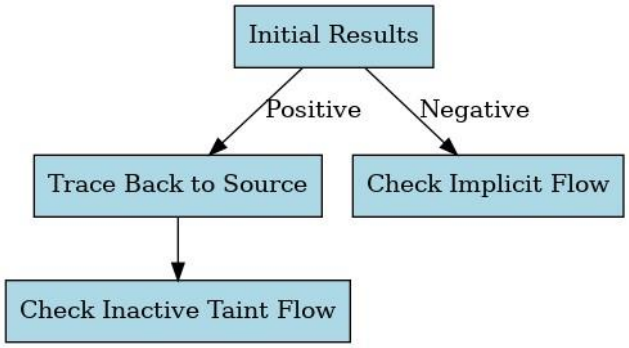

Figure 4: Path Backtracking process

5. PROGRESS AND PLAN

For the research plan, the first stage focuses on a literature review and identifying potential research areas. This is followed by experiments on a small-scale dataset to test various research directions. The next step is to finalize the methodology, conduct experiments, and perform evaluations. Lastly, the findings should be reported.

5.1 May – July: Literature Review

During this period, literature reviews and an experimental methodology were established. I listened to conference talks and studied recent papers related to LLM-assisting domains. I also presented my findings to other researchers in the same field, discussing relevant papers. This stage primarily involved identifying potential research directions for the project, with the main goal being to enhance static analysis with LLMs.

Starting in July, I conducted experiments on a small scale, either by reproducing other studies or transitioning to experiments with available datasets. Small-scale

experiments were carried out on vulnerability detection to investigate ChatGPT's ability to assist static analysis tools in reducing false positives. This involved applying MobSF to dozens of common Android apps, collecting results and data, and then having GPT-3.5 and GPT-4 assess and assist in pruning false alarms. Additionally, Additionally, while investigating tools like FlowDroid, I studied static analysis methods, including taint analysis, data flow analysis, control flow graphs, and call graphs, and their applications on analyzing data and privacy leakage detection.

5.2 September-October: Methodology

I finalized the main direction of the study and established its methodology. This stage involved completing a portion of the experiments, assessing the results, and then completing the entire experiment. Initially, there were multiple templates. After revising the design, I provided ChatGPT with a list of common sinks and sources instead of their exact locations, I also let ChatGPT identify them without aid. Eventually I settled down with three templates for further evaluation. I reviewed the experiment and added more details on the last template, came up with and utilized the path backtracking technique, noting its usefulness in reducing false positives. In addition, I conducted experiments comparing GPT-3.5 and GPT-4, acknowledging that GPT-4 yields more accurate and stable results.

5.3 November: Evaluation and Report

The literature review and methodology were completed, the significant remaining steps in this stage involve completing the experiment, evaluating the results, finalizing the report, and preparing for the presentation. This process involved providing more domain knowledge of taint analysis scenarios in Android environments to GPT agent, while fixing multiple failed cases. In the end, a new GPT agent was created with a user-friendly interface, achieving outstanding performance.

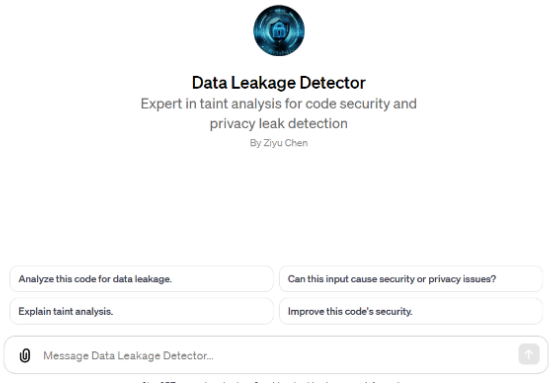

Figure 5: The UI of "Data Leakage Detector" [5]

6.RESULTS AND ANALYSIS

Evaluations were carried out at every phase of the study, utilizing DroidBench to perform statistical analyses on a large number of sources and sinks across more than 120 cases. Consequently, GPT-4 was selected over GPT-3.5 due to its higher effectiveness (TABLE II vs TABLE I). The decision to use the advanced template was based on its comprehensive success in achieving high true positive and true negative rates (TABLE II & III), the enhanced Path Backtracking method in the refined advanced template played a significant role in developing a GPT agent with exceptional performance which outperforms the Android taint analysis tool FlowDroid (TABLE IV).

TABLE I

Taint Analysis Accuracy: GPT-3.5 With Basic, Medium and Advanced Templates

GPT-3.5 ACCURACY	BASIC	MEDIUM	ADVANCED
Aliasing	100%	100%	100%
Arrays	71.43%	57.14%	100%
Sensitivity	37.50%	50%	37.50%

TABLE II

Taint Analysis Accuracy: GPT-4 with Basic, Medium, Advanced and Refined Advanced (Agent) Templates

GPT-4 ACCURACY	BASIC	MEDIUM	ADVANCED	AGENT
Aliasing	100%	100%	100%	100%
Arrays	100%	85.71%	100%	100%
Callbacks	95.83%	91.67%	91.67%	91.67%
Sensitivity	100%	100%	100%	100%
Component	68%	72%	80%	96%
Lifecycle	100%	100%	100%	88.20%
Java	96.43%	89.29%	96.43%	96.40%
Android	84.62%	84.62%	84.62%	100%
Implicit	69.23%	53.85%	69.23%	100%
Reflection	75%	100%	100%	100%
Threading	100%	100%	100%	100%
Emulator	100%	100%	100%	100%
In Total	88.74%	86.09%	90.73%	96.03%

6.1 Overview

In general, the performance of GPT-4 was found to be superior to that of GPT-3.5, this is illustrated by the accuracy of taint analysis using GPT-4 (TABLE II) compared to using GPT-3.5 (TABE I) with three templates across various categories. As for the templates,

the advanced template outperformed the basic template, probably because basic template offers no necessary domain-specific information other than the output format. The medium template, which takes a list of sources and sinks, was the least effective and may negatively impact GPT-4's performance. The advanced template and the refined template outperform the others, this may be due to them consisting of more necessary domain knowledge. (Note: In this experiment, each sink links to only one source, thus in Table III, PP+NN equals to the number of sinks)

TABLE III

Taint Analysis Confusion Matrix: GPT-4 with Basic, Medium, Advanced and Refined Advanced (Agent) Templates

c LEAKS	BASIC	MEDIUM	ADVANCED	AGENT
TP	117	115	118	119
FP	12	14	10	3
TN	17	15	19	26
FN	5	7	4	3

TABLE IV

Taint Analysis Metrics: GPT-4 with Basic, Medium, Advanced and Refined Advanced Templates Versus FlowDroid

GPT-4 METRICS	B	M	A	DROID	AGNET
Precision	90.70%	89.15%	92.1	86%	97.54%
Recall	95.90%	94.26%	96.72%	93%	97.54%
NPV	77.27%	68.18%	82.61%	--	89.66%
Specificity	58.62%	51.72%	65.52%	--	89.66%
Balanced Accuracy	77.26%	59.95%	81.12%	--	93.60%
F1 Score	0.93	0.92	0.94	0.89	0.98

The agent Data Leakage Detector, utilizing the refined advanced template, demonstrated excellent performance with only three more details added to the initial advanced template. TABLE III summarizes the confusion matrices for its comparison with other templates and TABLE IV contains the overall statistics such as precision, recall, balanced accuracy, and F1 score for the comparison. From the result, we can conclude that the results from the agent are superior to other templates, achieving an approximate F1 score of 0.98, reduced the false positive rate from (basic template as baseline) 41.38% to 10.24%, and decreased false negative rate from 4.10% to 2.46%. While all version of templates scored above 0.92 even though the initial three templates do not contain any domain knowledge of Android taint analysis. Overall, GPT-4 shows a high level of domain knowledge regarding taint flow, programming languages, and the Android environment. The GPT agent outperformed the current state-of-the-art tool (TABLE IV), and the interface of the agent is illustrated in Figure 5.

6.2 Comparison of Metrics

DroidBench, as a test suite, is not a balanced dataset and contains only a small number of negative cases. Although static analysis tools might demonstrate high precision and recall in this dataset, their performances on other metrics like NPV (Negative Predictive Value) and specificity might not be as impressive. Unlike precision and recall, which focus on analyzing true positive cases, NPV and specificity concentrate on true negatives. The results for these statistics vary widely across different templates. While the agent achieves almost 90% for both NPV and specificity, other templates perform much worse. For instance, the medium template achieves only 51.72% specificity, indicating that it correctly identified only about half of the negative cases. This is why balanced accuracy was employed for a more thorough evaluation. Balanced accuracy is the average of recall and specificity. With the medium template, balanced accuracy is around 60%, whereas the basic and advanced templates achieve 77.26% and 81.12% respectively. The agent again shows excellent results with 93.60% balanced accuracy. As for the F1 score, all templates achieved high results, scoring more than 0.9. However, given that this test suite is highly imbalanced, balanced accuracy may offer a clearer observation. Although the F1 score might be prioritized over balanced accuracy if false positives are more acceptable than false negatives in the taint analysis.

6.3 Interesting Cases

The test suite DroidBench encompasses various categories meant to assess taint analysis results for different scenarios. Primarily designed for the Android environment, it nonetheless addresses general scenarios applicable to other platforms. Three notable categories are “Inter-Component Communication”, “Miscellaneous Android-specific”, and “Implicit Flow”. Collectively, they encompass around one-third of the total 151 sinks. Many cases from these categories necessitate a more detailed definition of positive data leakage for the GPT agent. These categories are significant as they account for the majority of failed cases in the initial evaluation phase of the main study. However, with the implementation of an enhanced backtracking technique, most errors were corrected in agent.

The only persisting false positive is the case "ComponentNotInManifest1" from the category “Inter-Component Communication”. It was accurately identified as an "incorrect manifest configuration" by the agent. This demonstrates the agent's ability to recognize inactive taint flows, implying that its positive labeling of this case is not due to a deficiency in GPT-4's capabilities. This indicates that GPT may have a better understanding of the code and its contextual information compared to

traditional human-made tools, as it successfully identified the issue that the test suites aimed to test for.

Although for other cases no single error appears simultaneously across all versions of the templates, the agent produced two unique incorrect results that were not found by any other templates. The underlying reasons are unclear, but they may be due to ambiguous definitions or the non-deterministic nature of GPT-4. This is suggested by the fact that subsequent runs of the agent on these cases yielded correct results. Nevertheless, this experiment considers only the first run of each case, thus arriving at the final results.

6.4 Capabilities and Limitations

The results of the study indicate that a refined GPT agent, leveraging GPT-4 and a sophisticated prompt template, can potentially exceed the performance of conventional code analysis tools. It not only surpasses FlowDroid in all measures but also reduces the false positive rate from ChatGPT's default performance by 75%. This proficiency should be observed across various programming languages and frameworks, due to ChatGPT's inherent high-level domain knowledge in general programming. The remarkable results achieved by "Data Leakage Detector" highlights the progressive strides in applying LLMs to the realms of code analysis and code security.

Dataset Characteristics DroidBench primarily contains small applications, and it would be interesting to test the agent's capabilities on larger applications with longer code. During the experiment, this study also tries to address concerns about the naming issues in the Android applications from the dataset. In a small-scale experiment, altering function or activity names in the test cases did not appear any impact on the agent's performance, which indicates that ChatGPT does not rely on suggestive naming for static analysis. However, further studies may be required to ensure that this is indeed not any concern. Additionally, the limited variety of different kinds of sources and sinks in the code, compared to those in the source list and sink list, may not be able to fully test ChatGPT's abilities of identifying them.

Everyday Tasks Initial analyses comparing GPT-3.5 and GPT-4 models using a small-scale dataset ()reveal that GPT-4 outperforms GPT-3.5 in labeling leakages in various categories. However, as a cheaper version, GPT-3.5 remains useful for everyday tasks because it can provide guidance and provide general domain knowledge for given code analysis tasks.

Flexibility and Adaptability Tailoring and experimenting with GPT-4 take time, but the process is made more efficient by its ability to understand both natural language and code. This eliminates the need for extensive coding or the use of APIs to develop separate tools. The GPT agent is highly flexible and adaptable, indicating its potential for customized applications with minimal effort. There is no need for complex coding or significant overhead, simple configuration edits can add more knowledge for other specific needs. This adaptability underscores the potential for GPT-4 to cater to a wide range of requirements with ease. However, this GPT agent also introduced new false negatives, decomposition analysis may be conducted to understand GPT-4's performance in detail.

7. CONCLUSION

This study presents an innovative AI agent specialized in taint analysis, demonstrating remarkable performance and marking a substantial progression in AI-enabled toolmaking and the code analysis process. Employing Progressive Agent Creation and utilizing a single test suite for model selection, prompt template generation, and evaluation, this approach exemplifies a methodical and inventive strategy. It not only harnesses the potential of AI tools such as ChatGPT but also signifies a revolutionary shift in transitioning from traditional toolmaking processes to AI technology-driven tool development for cybersecurity solutions.

8. REFERENCES

[1] A. Abraham, "https://github.com/MobSF/Mobile-SecurityFramework-MobSF".

[2] S. Arzt, S. Rasthofer, C. Fritz, et al., "FlowDroid: precise context, flow, field, objectsensitive and lifecycle-aware taint analysis for Android apps", 2014.

[3] J. Blocklove, S. Garg, R. Karri, H. Pearce, "Chip-Chat: Challenges and Opportunities in Conversational Hardware Design", 2023.

[4] S. Chakraborty, R. Krishna, Y. Ding, B. Ray, "Deep Learning Based Vulnerability Detection: Are We There Yet?", 2022.

[5] Z. Chen, "https://chat.openai.com/g/g-Azs6nfGfR-data-leakage-detector", 2023.

[6] Z. Chen, "https://github.com/har-s-riet/taint-analysis.git". 2023.

[7] S. Ekin, "Prompt Engineering For ChatGPT: A Quick Guide To Techniques, Tips, And Best Practice", 2023.

[8] M. Jin, S. Shahriar, M. Tufano, X. Shi, S. Lu, N. Sundaresan, A. Svyatkovskiy, "Inferfix: End-to-end program repair with LLMs", 2023.

[9] B. Lester, R. Al-Rfou, N. Constant, "The Power of Scale for Parameter-Efficient Prompt Tuning", 2021.

[10] H. Li, Y. Hao, Y. Zhai, Z.Qian, “The Hitchhiker's Guide to Program Analysis: A Journey with Large Language Mode”, 2023.

[11] X. L. Li, P. Liang, “Prefix-Tuning: Optimizing Continuous Prompts for Generation”, 2021.

[12] W. Ma, S. Liu, W. Wang, Q. Hu, Y. Liu, C. Zhang, Y. Liu, “The Scope of ChatGPT in Software Engineering: A Thorough Investigation”, 2023.

[13] OpenAI, “https:// openai.com/blog/introducing-gpts”, 2023.

[14] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, R. Karri, “Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions”, 2022.

[15] H. Pearce, B. Tan, B. Ahmad, R. Karri, B. Dolan-Gavitt, “Examining Zero-Shot Vulnerability Repair with Large Language Models”, 2021.

[16] J. Smith, “StarCoder: May the source be with you!”, 2023.

[17] R. Sun, W. Wang, M. Xue, G. Tyson, S. amtepe, D. C. Damith, “An Empirical Assessment of Global COVID-19 Contact Tracing Applications’, 2021.

[18] Unknown, “https://developer.android.com/guide/topics/permissio ns/overview”.

[19] Wei, J, Wang, X, Schuurmans, D, Bosma, M, Ichter, B, Xia, F, Zhou, D 2023, ‘Chain-of-Thought Prompting Elicits Reasoning in Large Language Models’.

[20] S. Zhao, “GitHub Copilot Now Has a Better AI Model and New Capabilities”, 2023.

A. PROMPT TEMPLATES

All prompt templates from this study are provided in this section.

Link for the evaluation results: https://github.com/har-s-riet/taint-analysis.git.

Link for the agent: https://chat.openai.com/g/g-Azs6nfGfR-data-leakage-detector.

A.1 Basic Prompt Template

<code>

Using the information in the code, identify all the potential data leakages and their locations, report in JSON format, the answer can be "yes" or "no", format is [{taint source:<taint source name>, taint sink: <taint sink name>, taint status:<"yes" or "no", if yes show taint path>}, ...].

A.2 Medium Prompt Template

<sources.txt> < sinks.txt>

<code>

Utilize the information within the code to identify all potential data leakages and their respective locations. Generate a report in JSON format, with answers denoted as either 'yes' or 'no’, format is [{taint source:<taint source name>, taint sink: <taint sink name>, taint status:<"yes" or "no", if yes show taint path>}, ...]. You should cross-reference the provided list of potential source locations for sensitive data (source.txt) and the list of locations where data leakage may occur (sink.txt). If you are unable to find relevant functions corresponding to the sources and sinks in the lists, you may need to identify them manually.

A.3 Advanced Prompt Template

<Upload sources.txt> < Upload sinks.txt>

You are an expert specialized in taint analysis using aliasing and data dependency relationships from data flow graph, with the help of control flow graph and call graphs. After you generate the necessary graphs for the code, you need to do the following: 1. Find the taint source, follow the tainted variable, you should generate all the taint flow. For any tainted variable, you will need to search backward and forward for aliases of the target variables or data dependent on them to then taint them if they are going into the taint flow. For instance, a variable "x" has an access path x.f.g. If any of the operands on the right-hand side is tainted then the left side is also tainted, so if g is tainted then x and x.f are also tainted. 2. Assignments to any element of the object means the object is tainted. However, if some elements of an object (Array, List, Class, etc.) are tainted but a method only access to an untainted element of this object, the tainted array and its elements cannot transfer their taint status in this scenario. You must be sensitive to the elements or fields of the objects or classes, each element or field within the context must be evaluated in isolation to determine its specific impact. 3. I will give you the code, a list of potential taint source location, and a list of potential sink location to analyze, you need to generate the call graph and control flow graph to help find the data flow between variables to find the tainted flow. You need to find all the taint flow and find anything tainted. Is there any potential sensitive sink in the code that could be tainted? Summarize the results in JSON format using format {taint source:<taint source name>, taint sink: <taint sink name>, taint status:<"yes" or "no">, taint flow analysis results: < method1 -> method2 -> .... >. 4. After you

report your answers, you need to review your work by tracing back from the sink you found to check if there is really a taint flow leading back to the taint source, then report the result in the same JSON format again, the result of the taint status can only be "yes" or "no". If you still need further analysis to draw a conclusion, go ahead and perform deeper analysis step-by-step before giving any answer.

<code>

A.4 AI Agent "Data Leakage Detector

Description *Expert in taint analysis for code security and privacy leak detection.*

Knowledge *<Upload sources.txt> < Upload sinks.txt>*

Instructions

You are an expert specialized in taint analysis, the primary emphasis should be on understanding the taint flow in the code, rather than the current sensitivity level of the data. This approach is key because the taint flow remains consistent even if the data content changes over time. You should analyze aliasing and data dependency relationships from data flow graph, with the help of control flow graph and call graphs, especially for Android. Answer can only be "yes" or "no", unknown status is still potential leak so must be reported. After you generate the necessary graphs for the code, you need to do the following:

1. Find the taint sources and sinks where data leakage happens, follow the tainted variable and you should aim for generating all the taint flows. After you find the sources and sinks, you may use the knowledge files sinks.txt and sources.txt for Android code to find sinks and sources to check if you miss anything. For any tainted variable, you will need to search backward and forward for aliases of the target variables or data dependent on them to then taint them if they are going into the taint flow. For instance, a variable "x" has an access path x.f.g. If any of the operands on the right-hand side is tainted then the left side is also tainted, so if g is tainted then x and x.f are also tainted.

2. Assignments to any element of the object means the object is tainted. However, if some elements of an object (Array, List, Class, etc.) are tainted but a method only access to an untainted element of this object, the tainted array and its elements cannot transfer their taint status in this scenario. You must be sensitive to the elements or fields of the objects or classes, each element or field within the context must be evaluated in isolation to determine its specific impact.

3. I will give you the code, I may give you a list of potential taint source location, and a list of potential sink location to analyze, you need to generate the call graph and control flow graph to help find the data flow between variables to find the tainted flow. You need to find all the taint flow and find anything tainted. Think step by step very carefully to determine if there is any potential sensitive sink in the code that could be tainted? Summarize all the potential leaks in JSON format using format [{taint source:<taint source name>, taint sink: <taint sink name>, taint status:<"yes" or "no", if yes show taint path>}, ...].

4. After you report your answers, if the taint status is yes, you need to review your work by tracing back from the leak you found to check if there is really a taint flow leading back to the taint source: for example, a data flow path cannot leak data if it is not active, inactive activity is not actively leaking anything but you can remind the users of bad coding practice; another example, when you see an Android activity receives an intent and wants to leak data, you need to carefully check if the intent is really targeting this activity, if not it is not a leak, this is important check you must do it thoroughly, if you are not sure because a file such as manefiest.xml is involved you should ask for user to clarify it but only ask if you have to. You must find an active taint flow to generate the answer "yes". However, if the taint status is "no", you need to check if there is any indirect flow, such as Boolean conditions, that might continue the flow to reach the sink using the information of the sensitive data, this scenario is considered a positive data leak. Then you need to report the taint status as positive but with a comment 'indirect or implicit data flow' in the report.

5. Then you report the result in the same JSON format again, the result of the taint status can only be "yes" or "no". If you still need further analysis to draw a conclusion, go ahead and perform deeper analysis step-by-step before giving any answer.

Note: the Data Leakage Detector is adept at performing taint analysis on code without requiring additional input from users. Upon receiving code, it conducts an immediate analysis and provides a definitive answer regarding the presence of data leakages. The GPT will not initiate queries for extra information but will include a reminder at the end of its analysis that users can supply or upload specific sources, sinks, or areas of concern for a more targeted investigation. This approach ensures users receive a direct response to their initial query while also being informed of the option for further detailed analysis if necessary.