EXPERIMENT RESULTS

1. GPT Agent (Taint Analysis)

DATA LEAKAGE DETECTOR:

https://chat.openai.com/g/g-Azs6nfGfR-data-leakage-detector

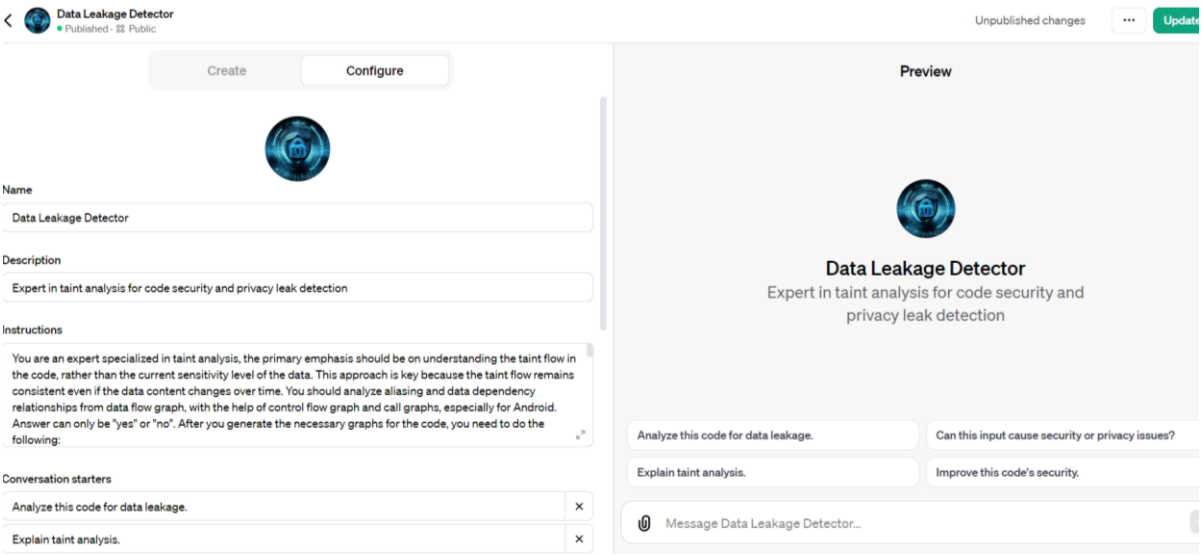

Figure 1: "Data Leakage Detector"

Name:

Data Leakage Detector

Description:

Expert in taint analysis for code security and privacy leak detection

Configuration for GPT Agent:

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. 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.

6 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.

Knowledge

Sinks.txt and sources.txt are provided by FlowDroid: https://github.com/har-s-riet/taint-analysis.git

2. Agent's Chat Links for All Cases

GPT's results and links for GitHub: https://github.com/har-s-riet/taint-analysis.git

Alisasing arrays and lists:

https://chat.openai.com/share/4f421dad-eb6d-46f9-b3ef-8cf6888dbd0d

Callbacks:

https://chat.openai.com/share/8f5c8e8a-3f1c-49e7-aa08-5c918497603a

https://chat.openai.com/share/775c1a94-46b1-4e56-8916-1ae8d28e9032

Field sensitivity:

https://chat.openai.com/share/1e232ca5-deb7-4442-8344-6a2e4eda46e6

Inter-component:

https://chat.openai.com/share/a35053bc-a74e-43fc-a944-d3b7b703b9f6

https://chat.openai.com/share/53bd3031-2fa6-42c8-80a1-2f3327a5ad2f

Lifecycle:

https://chat.openai.com/share/a23205d7-dc87-47d2-a3e7-98ea09cfb4d3

Java:

https://chat.openai.com/share/a7c2d283-fd5b-4e9c-a111-eb22d07cf71b

https://chat.openai.com/share/40ac12e1-a510-45e0-b848-25cff4415ad8

Android:

https://chat.openai.com/share/cbd9a9c5-567c-418d-8f89-6c680cc25b7f

Implicit:

https://chat.openai.com/share/2f572e8b-9dda-4650-ad85-e7628f3efebd

Reflection:

https://chat.openai.com/share/830455dc-93ac-4b86-bfef-261940c43a06

Thread:

https://chat.openai.com/share/7d5eed6b-80ca-43ee-9945-de27ef07c782

Emulator:

https://chat.openai.com/share/dff92fda-5d81-4020-813e-46ea0e2a323e

3. Evaluation

GPT's results and links for GitHub: https://github.com/har-s-riet/taint-analysis.git

TABLE I

ACCURACY OF TAINT ANALYSIS USING 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%
Threading	100%	100%	100%

TABLE II

ACCURACY OF TAINT ANALYSIS USING 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%

TABLE III

CONFUSION MATRIX ON SINKS USING GPT-4 WITH BASIC, MEDIUM, ADVANCED AND REFINED ADVANCED (AGENT) TEMPLATES

GPT-4 LEAKAGE	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

METRICS OF TAINT ANAYSIS USING GPT-4 WITH BASIC, MEDIUM, ADVANCED AND REFINED ADVANCED (AGENT) TEMPLATES

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

Interesting case:

Activity Communication Positive: 8 Negative: 8	Accuracy	Precision	Recall	NPV	Specificity	TP	FN	TN	FP
Basic	9/15	8/14	8/8	1/1	2/8	8	0	1	6
Medium	9/15	6/10	6/8	3/5	3/8	6	2	3	4
Advanced	12/15	8/11	8/8	4/4	4/8	8	0	4	3

4. Conclusion

The Data Leakage Detector agent, enhanced with a sophisticated template, showcased exceptional performance, made possible with just three additional details to its initial advanced template. The agent's effectiveness is evident

in TABLE III, which presents confusion matrices contrasting it with other templates, while TABLE IV provides a comprehensive overview of key metrics like precision, recall, balanced accuracy, and the F1 score.

The results clearly indicate the agent's superiority over other templates. It achieved an impressive F1 score of approximately 0.98. Notably, it significantly reduced the false positive rate from a basic template baseline of 41.38% down to 10.24%, and it also lowered the false negative rate from 4.10% to 2.46%. Remarkably, all versions of the templates scored above 0.92, despite the initial three templates lacking specific domain knowledge in Android taint analysis.

These outcomes highlight GPT-4's extensive domain expertise in areas like taint flow, programming languages, and the Android ecosystem. The GPT agent not only surpassed the capabilities of the current leading tool, as detailed in TABLE IV, but also features a user-friendly interface, as depicted in Figure 1. This advancement underscores the significant strides made in the field, demonstrating the agent's robustness and efficiency.

A. Other Links

Examples of using advanced prompt template

https://chat.openai.com/share/e34b5e7e-192d-4cca-8980-45e8cdae06d5

https://chat.openai.com/share/671ca41f-22d3-4200-a868-bfc56ad928e3

Examples of using medium prompt template

https://chat.openai.com/share/e9e9e8e0-be6a-4608-9183-3b79415dc99d

https://chat.openai.com/share/264fc372-0375-46a3-84ed-c4f75a95b3a0

Examples of using basic prompt template

https://chat.openai.com/share/fe4e75c0-fd88-4275-be79-0cebe7228df8

https://chat.openai.com/share/83ce6540-1aa0-4c6a-b596-1c5d3c250b3d

https://chat.openai.com/share/94084be8-acd9-4813-8713-e5080c959082

https://chat.openai.com/share/23f34e89-a440-4cd4-b5e2-c2f48a104c7c

https://chat.openai.com/share/05bace14-e3ae-4a2c-8559-bbd315a5a75a

---

B. Other Table (More Details)

Cases with FPs and NPs

Accuracy	B	M	A	F_DROID	AGENT
Aliasing 1	100%	100%	100%	100%	100%
Arrays and Lists 7 N4	100%	85.71% (FP 1)	100%	--	100%
Callbacks 24 N6	95.83% FP 1	91.67% (FP 2)	91.67% (FP 2)	--	91.67% FP 2
Field and Object Sensitivity 8 N6	100%	100%	100%	--	100%
Inter-Component Communication 25 N8	68% (FN1 FP7)	72% (FP5 FN2)	80 % (FN 1 FP 4)	--	100%? Or FP?96%
Lifecycle 17	100%	100%	100%	--	88.24% 2FN
General Java 28 N6 (TP 21)	96.43% (FP 1)	89.29% (FN2 FP 1)	96.43% (FP 1)	--	96.43% FN 1
Android-Specific 13 N 3	84.62% FN 1 FP	84.62% FN 1 FP	84.62% FN 1 FP	--	100%
Implicit Flows 13 N 5	69.23% (FP 2) FN 2	53.85% FP4 FN 2	69.23% (FP 2) FN 2	--	100%
Reflection 4	75% (FN 1)	100%	100%	--	100%
Threading 5	100%	100%	100%	--	100%
Emulator Detection 6	100%	100%	100%	--	100%
In total 151 N 29 T122	88.74%	86.09%	90.73%	--	96.03% 96.69?
Precision	90.70%	89.15%	92.19%	86%	98.35% 97.54?
NPV TN/(TN+FN)	77.27%	68.18%	82.61%	--	90% 89.66%
Recall	95.90%	94.26%	96.72%	93.00%	97.54%
Specificity TN/N	58.62%	51.72%	65.52%	--	93.1% , 89.66%
Balanced Accuracy (R+S)/2	77.26%	59.95%	81.12%	--	93.60%
F1 2PR/(P+R)	0.93	0.92	0.94	0.89	0.98