G-3

Trusted Application における LLM を活用した静的テイント解析手法の提案

Proposal of a Static Taint Analysis Method Utilizing LLM in Trusted Applications

○吉永達哉 [1]，依田みなみ [2]，松野裕 [3]
*Tatsuya Yoshinaga[1], Minami Yoda[2], Yutaka Matsuno[3]

This paper presents a static taint-analysis tool for OP-TEE TAs that leverages LLMs to assess reachability to dangerous sinks and contextual validity. The pipeline performs function and external-API extraction, sink candidate identification, call-chain reconstruction from entry points to sinks, and stepwise LLM deliberation to determine propagation and mitigations. Results are reported in JSON/HTML. Initial case studies show the ability to surface context-dependent issues that rule-based systems such as DITING may miss.

１．はじめに

スマートフォンや PC の機密処理は TEE（Trusted Execution Environment）が担うことで安全性を確保している．通常領域と機密領域間の境界で生じる脆弱性は，デバイスの機密情報の漏洩につながる．スマートフォンの世界出荷台数は 2030 年に約 15 億台と見込まれており[1]，その内部で動作する TEE が脆弱である場合，広範な機器にリスクが及び得る．

既存研究のルールベース検知では，境界で生じる平文出力や未検証入力，共有メモリの不適切な書込みは複雑であり見逃されやすい．本研究は，LLM を静的テイント解析に統合し，関数呼び出しチェーンとコード文脈に基づいて平文出力と未検証入力，共有メモリの不適切な利用を判定するツールを提案する．実験では既知の脆弱性 6 つを追加したサンプル TA に対して先行研究では見つけられなかった 3 件の脆弱性を検出できた．

２．原理

２.１．TEE（Trusted Execution Environment）

TEE は，REE（Rich Execution Environment）から TA（Trusted Application）をハードウェアレベルで分離して実行する技術である．TA は TEE 内部で鍵管理・暗号処理・認証などの機密処理を担う．REE と呼び出しインタフェースと共有メモリを介してのみ連携する．本研究では，Arm TrustZone を基盤とする OP-TEE 向け TA を対象とする．

先行研究の DITING では TEE-REE 境界の扱いが不適切であると，機微情報の漏えい，未検証入力の受容，共有メモリの不正な書込みといったパーティショニング問題が生じることを定式化しており，本研究も同様の問題を検出対象とする．[2]

２.２．静的テイント解析

テイント解析は，信頼できない入力を受け付ける関数をソースとしてラベル付けし，代入や引数渡しを通じたラベルの伝播を静的に追跡を行い，ラベルがセキュリティ上問題となりうるシンクに到達するかを判定するデータフロー解析手法である．本研究では，ソースを REE 由来のパラメータや共有メモリとし，シンクは LLM が判定を行い，ソースからシンクまでの関数チェーンを再構成し，各関数の段階ごとに LLM に質問を行うことでテイント解析で必要なルール設定を排除する．

３．システム実装

システムの概要を Fig.1 に示す．関数・外部 API 情報の抽出，エントリからシンクへの呼び出しチェーン再構成，段階的対話による到達性と緩和策の判定を統合する．対象脆弱性は DITING の 3 つに（平文出力，入力検証不足，共有メモリの上書き／直接使用）を基軸とし，さらに LLM の文脈理解で境界チェック欠落，サイズ不整合の構造的リスクも検出する．

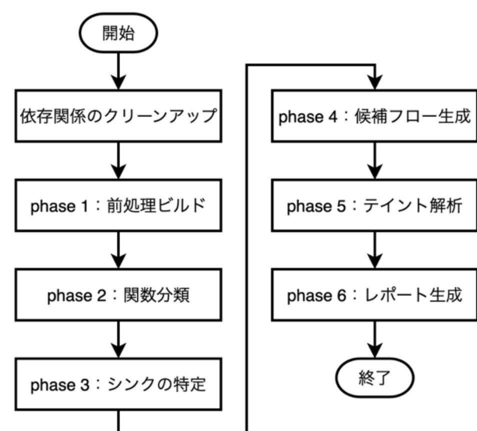

Fig. 1 Overall processing flow of the proposed method

1：日大理工・学部・情報 2：日大理工・教員・情報 3：日大理工・教員・情報

Phase 1 で関数名・位置を中間表現に格納し，Phase 2 でユーザ関数と外部宣言を確定する．Phase 3 で外部関数呼び出しのシンク該当性と脆弱引数を LLM が判定し，Phase 4 で逆方向探索により冗長サブチェーンを除去したチェーン集合を得る．Phase 5 ではチェーン要約，関数点検，再投影の 3 ステップで LLM に質問し，シンク未到達でも上記の構造的リスクを併記する．最後に Phase 6 で JSON 判定と対話ログを HTML にまとめる．

実装とプロンプト一式は GitHub で公開しており，必要なスクリプトや結果も順次整備する．[3]

４．実験

DITING が公開している PartitioningE-Bench[2] の randomTA に DITING が検知できなかった 3 つの脆弱性を追加したサンプル TA に対して，gpt-5-nano を用いて検証を行った．既存ツールである DITING と本システムの判定結果を行単位で整理し，正しく脆弱性を検出した場合を TP（True Positive），誤って脆弱性と判定した場合を FP（False Positive），脆弱性と判定されるべき箇所を脆弱でないと判定した場合を FN（False Negative）として集計した結果を Table 1 に示す．DITING は，ユーザー定義関数内の呼び出しや手動実装のメモリ書込みに起因する脆弱性を見落とす．一方で，本システムはそれらを検知でき、DITING と比較して追加で 3 件の脆弱性を検出した．

Table. 1 Comparison of this tool and DITING

	THIS TOOL	DITING
TP	6	3
FP	1	1
FN	0	3

Fig.2 は for ループでのメモリ書込み例であり，確保済みバッファ容量と書込み量の整合が検査されていないため，境界超過の危険がある．この脆弱性は DITING が主に明示的な TEE_MemMove 呼び出しや固定長コピーを前提としたルールを採用しており，ループ境界の妥当性やサイズ計算の文脈を扱わないために未検出となったケースである．

```
112      char str_ov[1024] = {0};
113      const uint8_t *p2 = (const uint8_t *)params[2].memref.buffer;
114 ⌄    for(uint32_t i = 0; i < params[2].memref.size; i++) {
115          str_ov[i] = (char)p2[i]; //←ルールベースのDITINGでは不検出
116      }
117      TEE_Free(buf);
```

Fig. 2 Vulnerable code that cannot be detected by DITING

一方，本システムは関数境界での段階的対話と呼び出しチェーンの整合検査を通じて，当該ループの長さ計算とコピー範囲の不一致を設計上の脆弱性として指摘できる．段階的対話で得られた最終出力を Fig.3 に JSON 形式で示す．


```json
{
  "file": "/workspace/benchmark/random/ta/random_example_ta.c",
  "line": 115,
  "function": "random_number_generate",
  "rule": "weak_input_validation",
  "why": "Per-byte copy from tainted input using a tainted length;
      potential bounds issue into local stack array",
  "sink_function": "array_write",
  "rule_matches": {
    "rule_id": [
      "weak_input_validation"
    ],
    "others": []
  },
  "code_excerpt": "str_ov[i] = (char)p2[i];"
}
```


Fig. 3 Output in the proposed method

５．関連研究

DITING は境界問題をルールで検出するが，網羅性や文脈依存の問題がある．LATTE[4]は LLM で文脈を補うが TEE 特化ではない．本研究は DITING の脆弱性 3 つを，シンクの呼出しチェーンで LLM と段階的対話を行い，サイズの不整合や手書きコピー等の構造的リスクを検知する．評価では PartitioningE-Bench[2]に脆弱性を追加した TA に対し DITING 未検出の 3 件を検出し，境界依存の見逃しを埋める有効性を実証した．

６．まとめと今後の課題

本稿では，TEE 向け段階的テイント解析手法を提案し，有効性を示した．一方で，LLM 利用に伴う再現性低下や誤判定が課題として残る．今後は出力手順の固定化や根拠提示の必須化により揺らぎを抑制し，公開ベンチマークに基づく定量評価を整備する．

７．参考文献

[1] 総務省：“令和 7 年版　情報通信白書の概要”，https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r07/html/nd215220.html，閲覧日：2025/09/25

[2] Chengyan Ma et al : “DITING: A Static Analyzer for Identifying Bad Partitioning Issues in TEE Applications”，arXiv, eprint={2502.15281}，2025

[3]吉永達哉：“GitHub tee-flow-inspector”，https://github.com/ta-061/tee-flow-inspector/tree/AcademicStudies

[4] Puzhuo Liu et al : “LLM-Powered Static Binary Taint Analysis”，ACM Trans. Softw. Eng. Methodol. Vol. 34, No.3, Article 83, 2025.