データセーフティセクションに対応したテイントソースの自動識別に向けて

稲吉 弘樹[1,a)]　齋藤 彰一[2]　門田 暁人[1]

概要：プライバシー保護の要求が強まり，ユーザや端末のデータを不正に収集する Android アプリの実態調査が実施されてきた．調査では，収集データを明らかにするために，アプリ内を流れるデータを追跡するテイント解析が利用される．このとき，追跡の開始条件（テイントソース）を指定する必要がある．数万の API メソッドを持つ Android フレームワークからテイントソースを人力で識別することは困難であるため，CoDoC や DocFlow といった自動識別手法がこの 1 年余りで提案された．一方で，近年アプリストアにデータセーフティセクション（DSS）が導入されたが，DSS 対応テイントソースの自動識別に取り組んだ研究はなく，性能評価用のデータセットも存在しない．既存の DSS に関する実態調査では，テイントソースはそれぞれの実施者が手動で識別している．本稿は，DSS 対応テイントソースの自動識別の第一歩として，CoDoC と DocFlow の汎化性能を調査し，また，437 個の API メソッドを含む DSS 対応テイントソースのデータセットを作成する．これを用いて CoDoC と DocFlow の正確性を評価し，DSS 対応テイントソースの識別に対する性能を明らかにする．

キーワード：テイントソース，テイント解析，プライバシー漏洩検出，Android

Towards Automatic Identification of Taint Sources Corresponding to the Data Safety Section

Hiroki Inayoshi[1,a)]　Shoichi Saito[2]　Akito Monden[1]

Abstract: With the increasing demand for privacy protection, researchers have investigated Android apps harvesting privacy data. In the investigation, taint analysis is executed to track data flow within an app after the starting points of the data flow (i.e., taint sources) are specified. Since manually identifying taint sources among tens of thousands of API methods provided by the Android framework is infeasible, automatic identification approaches such as CoDoC and DocFlow have been proposed lately. On the other hand, the Data Safety Section (DSS) has recently been introduced to the Android app store. Still, no studies or datasets for automatically identifying DSS-corresponded taint sources (DSSTSs) exist. In current DSS-focused investigations, DSSTSs are manually identified. In this paper, as a first step for automatic DSSTS identification, we evaluate the generalization performance of CoDoC and DocFlow. We also create a dataset including 437 DSSTSs and evaluate CoDoC and DocFlow to clarify their DSSTS identification performance.

Keywords: taint source, taint analysis, privacy leak detection, Android

1. はじめに

プライバシー保護に対する要求が高まる中，広告サービスによる過剰なユーザ追跡行為などを明らかにするために，Android アプリの実態調査が実施されてきた．アプリのプライバシーポリシーの分析 [1], [2], [3] や，プライバシーポリシーやプライバシーラベルの内容とアプリの実際のデータ収集動作との間の不一致の検出 [4], [5], [6], [7] が実施されてきた．

[1] 岡山大学 Okayama University
[2] 名古屋工業大学 Nagoya Institute of Technology
[a)] inayoshi@okayama-u.ac.jp

後者のようにアプリの実際の動作を解析する場合，アプリ内を流れるデータを追跡するテイント解析が利用されることがある．テイント解析は，アプリが調査対象のデータを読み込む地点（**図 1** の 2 行目）から，そのデータの追跡を開始し，アプリコード内の変数間を伝播していくデータを追いかけ（4 行目），そのデータがネットワークなどの外部へ送出される地点（8 行目）を検知する．解析者は，調査対象データの取得に関わる API メソッドを追跡開始条件（テイントソース）とし，ネットワーク通信などに関わる API メソッドを追跡終了条件（テイントシンク）として事前に設定する必要があり，これらはまとめてテイントスペシフィケーションと呼ばれる．

Android フレームワークは数万の API メソッドを持ち，年々更新されていくため，テイントスペシフィケーションとする API メソッドを人力で抽出する場合，不完全な結果となる可能性があり，また，更新に追随することが困難になる [8]．解決策の 1 つとして，テイントスペシフィケーションの自動識別手法が研究され，2013 年に SuSi [9], [10] が，さらに過去およそ 1 年の間にその改良に取り組んだ CoDoC [11] と DocFlow [12] が提案された．

一方で，特にテイントソースは解析者が注目するデータに依存するため，それに合った自動識別手法が必要である．2022 年 7 月に Android 公式のアプリストア Google Play にデータセーフティセクション（DSS）が導入され，DSS に関する実態調査に注目が集まっているが，DSS に対応したテイントソースの自動識別に取り組んだ研究はなく，性能評価用のデータセットも存在しない．DSS の申告対象として指定された 14 カテゴリ 38 データタイプへアクセスする具体的な API メソッドを Android フレームワーク中から自動的に抽出することが仮にできたとすれば，完全で更新にも追随したテイントソースを得られる．例えば，位置情報カテゴリに対応づく API メソッドの 1 つとして，Location クラスの *getLatitude()* メソッドが挙げられる．

本稿は，DSS 対応テイントソースの自動識別への第一歩として，DSS 対応テイントソースのデータセットを作成し，これを用いて，CoDoC と DocFlow の DSS 対応テイントソース識別性能を明らかにする．ここで，CoDoC はネガティブリザルトとして発表されたものであり，未知のデータに対するテイントスペシフィケーション識別の自動化は困難であると主張している．一方で，DocFlow は高い正確性を達成したことが報告されている．これら 2 手法の主張が食い違っているため，予備調査として CoDoC と DocFlow をそれぞれ未知のデータで評価する．結果，DocFlow は F-score で CoDoC を大きく上回るが，Recall がかなり高い分 Precision が低く偏っており，テイントソースでない API メソッドも含めて，多くの API メソッドをテイントソースと判定する傾向が強いことが示された．

次に，DSS のデータカテゴリと API メソッドの正しい対応づけを手動で求めたデータセットを作成する．5 つの収集源から集めた API メソッドを 11 データカテゴリに対応づけ，437 個の API メソッドを採用した．これを用いた既存手法の性能評価では，データカテゴリによっては正確性が 70〜80%台と高いものがいくつかある一方で，20〜40%台と低いものも半数近くあり，得意・不得意がはっきりと分かれた．最後に，CoDoC と DocFlow を API レベル 30〜33 の Android フレームワークに対して適用し，各 API レベルの約 5 万メソッドすべてに対する分類を試みた結果，DocFlow は DSS 学習前で 37〜38%，DSS 学習後では 48%をテイントソースと識別した．また，CoDoC も少なくとも 21%をテイントソースと識別した．これらのテイントソースは，予備調査と DSS 対応テイントソースを用いた評価の結果を踏まえると，無関係な API メソッドを多く含むことが予想され，また多量のテイントソースはテイント解析への負荷の増大が懸念される．実際にテイント解析を動作させる評価を今後の課題とする．

以降，2 章で既存のテイントスペシフィケーション自動識別手法を紹介し，3 章で予備調査を説明する．4 章で DSS データセットの構築について述べる．5 章で評価を報告し，6 章で制限と今後の課題を説明する，7 章で関連研究を挙げ，8 章でまとめを述べる．

```
1  // データの読み込み
2  String lat = location.getLatitude();
3  // データの伝播
4  String data = "Latitude: " + lat;
5  // データの書き込み
6  OutputStream outputStream =
7      urlConnection.getOutputStream();
8  outputStream.write(data.getBytes());
```

図 1 プライバシーデータを収集するアプリのプログラムの例
Fig. 1 Example of app code collecting privacy data.

2. テイントスペシフィケーションの自動識別

Android フレームワークが提供する API メソッドに対して，そのソースコードやドキュメントに基づき，テイントソース，テイントシンク，その他の 3 分類を自動的に行う手法として，特にプライバシーにフォーカスしたものに，SuSi，CoDoC，DocFlow が挙げられる．2013 年に提案された SuSi [9], [10] は，API メソッドのソースコードに基づいて，Support Vector Machine（SVM）を用いた分類を行う．CoDoC は 2023 年，DocFlow は 2024 年に提案された最新の手法であり，SuSi を上回る性能を示しているため，本稿はこの 2 つに注目する．

2.1 CoDoC

CoDoC [11] は SuSi の後継を目指して開発され，2023 年

に発表された．SuSi との違いとして，API メソッドのソースコードに加えて，ドキュメントも用いる点が挙げられる．また，code2vec や Sentence-BERT といった新しい埋め込み手法を用いていおり，分類器も SVM ではなくディープラーニングを用いている．SuSi を上回る性能を示したが，未知のデータに対しては，SuSi と同様に誤検知が多く，実用レベルではないとして，ネガティブリザルトとされた．正確性の評価として，CoDoC が識別したテイントソースとシンクそれぞれから 100 件が検証され，テイントソースは 90%，シンクは 56%が誤検知という結果が報告された．

結果の改善は容易ではないことが説明されている．ドキュメントの追加や機械学習技術の改善，より良いトレーニングデータ，トレーニングの最適化などを行ったとしても，改善は難しいだろうと述べられている．個々の API メソッドと，ユーザのプライバシーのような抽象的な概念との間のセマンティックギャップは，教師あり機械学習では埋められそうにないことが主張されている．

2.2 DocFlow

DocFlow [12] は 2024 年に発表され，API メソッドのドキュメントのみに基づいて分類を行う点が CoDoC や SuSi と異なっている．ドキュメントの形式を複数種類提案して評価している．構成要素として，メソッド名，メソッドシグネチャ（クラス名，メソッド名，引数と戻り値の型），メソッドの説明，クラスの説明があり，これらを組み合わせる．DocFlow 作者らの評価の結果，メソッドシグネチャ，メソッドの説明，クラスの説明を組み合わせたもの（形式 D）が最も良い結果を出している．

ドキュメントの埋め込みを Sentence-BERT を用いて生成する点は CoDoC と同様だが，異なるモデルを利用している．また，Sentence-BERT のファインチューニングあり・なしの両方を評価しており，ファインチューニングを実施した方が高い性能を示している．分類器は，Logistic Regression（LR），SVM，Neural Network（NN），XGBoost（XGB）の 4 種類を評価している．XGBoost が最も性能が高く，XGBoost を用いた DocFlow と SuSi の比較では，SuSi を上回る性能が示されている．

3. 予備調査

CoDoC は未知のデータに対するテイントスペシフィケーション識別の自動化の困難さを指摘している一方で，DocFlow は高い正確性を達成できることが報告されている．これらの主張が食い違っているため，予備調査として CoDoC と DocFlow をそれぞれ未知のデータで評価し，汎化性能を明らかにする．

CoDoC は，ディープラーニングに関わるコード以外のコンポーネントやデータセットが公開 [13] されており，ディープラーニング部分は我々が再現した．DocFlow は，ファインチューニング済モデルなどを除いて，大部分が公開 [14] されており，ファインチューニングは我々が実施した．また，DocFlow のドキュメントの形式は，最も良い結果が報告された形式 D を用いる．事前にそれぞれの論文で実施された評価を再現し，報告された通りの性能を示すことを確かめている．

3.1 データセット

予備調査では，CoDoC のデータセットを用いて DocFlow を評価し，DocFlow のデータセットを用いて CoDoC を評価する．CoDoC のデータセットは 1,015 件のデータを持ち，データの重複や，同じ API メソッドに異なるラベルが付くといった問題はない．ここで，CoDoC のデータセットは API レベル 30 の Android フレームワークから作成されており，DocFlow のデータセットは API レベル 32 である．本調査では，より新しい API レベル 32 に対する CoDoC と DocFlow の汎化性能を比較することとし，CoDoC のデータセットを API レベル 32 の Android フレームワークから再生成する．API レベル 32 の Android フレームワークのソースコードを取得，ビルドし，CoDoC の抽出器を用いてドキュメントとコードを抽出した．これに対して，CoDoC のデータセットを対応づけた結果，API レベル 30 から 32 でメソッドが削除されたものが 4 件，メソッドのコードが変化したものが 137 件あった．メソッドのコードが変化したものは，その動作が変わりテイントソースではなくなった可能性もあるため，データセットから排除した．よって，これら 141 件を除外し，874 件が得られた．

一方で，DocFlow は 3,000 件のデータを持つが，データの重複や，同じ API メソッドに異なるラベルが付くものがあり，適切なものだけを取り出した結果，2,344 件得られた．さらに，CoDoC で必要となる API メソッドのソースコードの取得を試み，1,249 件が得られた．

最後にデータセット間で重複した 14 件を除外した．最終的に実験で用いるデータセットは，CoDoC に対しては，DocFlow のデータセットから得られた 1,235 件であり，テイントソースが 599 件，シンクが 351 件，その他が 285 件である．DocFlow に対しては，CoDoC のデータセット（API レベル 32）の 860 件であり，ソースが 203 件，シンクが 70 件，その他が 587 件である．

3.2 結果

テイントソース，シンク，その他の 3 分類において，CoDoC は F-score 0.39，DocFlow は最高で 0.56 とどちらも低い（**表 1**）．BERT の “default” はデフォルトの BERT を用いたことを示し，“fine-tuned” はファインチューニングを行った BERT を用いたことを示している．データセット中の 3 クラスの数に偏りがある点を考慮し，それぞれの評価指標は，重み付き平均を用いている．

表 1 CoDoC と DocFlow の 3 分類の汎化性能評価結果（A = Accuracy，P = Precision，R = Recall，F = F-score）

Table 1 Generalization performance of CoDoC and DocFlow in three-class classification.

	BERT	A	P	R	F
CoDoC	default	0.40	0.68	0.40	0.39
DocFlow NN	default	0.54	0.77	0.54	0.56
DocFlow NN	fine-tuned	0.46	0.81	0.46	0.46
DocFlow LR	default	0.48	0.76	0.48	0.48
DocFlow LR	fine-tuned	0.43	0.78	0.43	0.43
DocFlow SVM	default	0.51	0.76	0.51	0.53
DocFlow SVM	fine-tuned	0.43	0.78	0.43	0.42
DocFlow XGB	default	0.48	0.78	0.48	0.49
DocFlow XGB	fine-tuned	0.44	0.79	0.44	0.43

表 2 CoDoC と DocFlow の 2 分類の汎化性能評価結果（A = Accuracy，P = Precision，R = Recall，F = F-score）

Table 2 Generalization performance of CoDoC and DocFlow in two-class classification.

	BERT	A	P	R	F
CoDoC	default	0.65	0.87	0.33	0.48
DocFlow NN	default	0.83	0.59	0.90	0.72
DocFlow NN	fine-tuned	0.87	0.65	0.96	0.77
DocFlow LR	default	0.83	0.58	0.96	0.73
DocFlow LR	fine-tuned	0.86	0.63	0.97	0.77
DocFlow SVM	default	0.84	0.60	0.95	0.73
DocFlow SVM	fine-tuned	0.86	0.64	0.96	0.77
DocFlow XGB	default	0.82	0.57	0.97	0.72
DocFlow XGB	fine-tuned	0.85	0.62	0.96	0.75

本稿の対象であるテイントソースによりフォーカスするために，テイントソース，シンク，その他の 3 分類ではなく，テイントソースとその他の 2 分類を考える（**表 2**）．3 分類と比べて，CoDoC は Recall を除いて数値が高い．Precision が 87%と高いが，Recall が 33%と低く，多くのテイントソースを見逃す可能性が高いことを示している．

一方，DocFlow も F-score が上がっており，最高で 0.77 と高いが，Precision と Recall の大小関係が逆転し，Recall へ大きく偏っている（表 2）．この結果は，DocFlow は API メソッドをテイントソースと判定する傾向が強いことを示している．API メソッドが誤ってテイントソースと識別され，テイントソースが不必要に多い場合，これを用いたテイント解析で誤検知が多く発生し，解析時間や結果検証コストへ悪影響を与えることが懸念される．以上より，どちらの手法も汎化性能には依然として課題が残っている．

4. DSS データセットの構築

DSS 対応テイントソースのデータセットを構築するために，これまでに実施された Android アプリのプライバシーに関する研究と Google が提供する情報から，テイントソースとなり得る API メソッドを集め，DSS のデータ分類に手動で対応づける．結果，5 つの収集源から重複を除いて 447 個の API メソッドを収集し，11 データカテゴリに対応づけた（**表 3**）．どのデータカテゴリにも対応づかないものが 10 個あり，最終的に 437 個が得られた．

Arkalakis ら [6] は，DSS とアプリの実際の動作との間の不一致を調査するために，DSS の 9 データカテゴリに対応するテイントソースを手動で作成した．成果物は公開 [16] されているが，公開時期が本稿の提出の直前であったため，本稿に含むことはできていない．Arkalakis らの原稿中では一部のテイントソースが掲載されているが，データカテゴリへのマッピングは示されていない．そこで，我々がマッピングを実施し，App info and performance に対応づけた（表 3）．5 つの API メソッドは，データベースのクエリを実行するためのものであり，データベース内のデータがわからなければデータカテゴリを対応づけられないため，除外した．

我々はこれまでに，位置情報と識別子にフォーカスし，サードパーティ SDK 開発元が DSS 申告の補助のためにアプリ開発者へ提供する情報と，SDK 本体の動作との間の不一致を調査した [7]．2 カテゴリ 19 メソッドがあり，除外したものはない（表 3）．

Kober ら [8] は，コミュニティの力でテイントソースを識別し蓄積することを提唱した．彼らのリポジトリ [17] には，執筆時点で位置情報や音声情報に関する API メソッドが登録されている．データ種類を表すラベルが付与されているが，DSS 向けのものではないため，我々はこのラベルを参考に，API メソッドのドキュメントに基づき，DSS のデータカテゴリへ対応づけた（表 3）．パスワードの取得に関わる API メソッドが 1 個あり，DSS の申告対象データカテゴリにはパスワードが属するものはないため除外した．また，クリップボードのデータ取得に関わるものが 2 個あり，これらはデータベースと同様に，データカテゴリを事前に対応づけることはできないため除外した．

Zhao ら [5] は，サードパーティライブラリのプライバシーポリシーと実際の動作との間の矛盾を検知するために，プライバシーに関わる 15 のデータカテゴリに対応づくテイントソースを作成した．このテイントソースは公開データ [18] の中に含まれている．しかし，API メソッドとデータカテゴリとの対応関係は示されていないため，我々がマッピングを実施した（表 3）．タイムゾーンを取得する API メソッドが 1 つあり，タイムゾーンは非常に広い範囲を示すため，位置情報と考えることは難しく，また，他のデータカテゴリにも合わないため，除外した．また，カメラのプレビューを設定する API メソッドが 1 つあり，これ自体は何も返さないため，除外した．

DSS に関して，Google がアプリ開発者向けに公開する情報として，主に 2 つのページがある．1 つは，DSS 作成のためのガイドライン [19] であり，申告対象となるデー

表 3 DSS データセット構築のために API メソッドを DSS データ分類へ対応づけた結果
Table 3 Mapping APIs to DSS data categories for the DSS dataset construction.

	Arkalakis ら [6]	Inayoshi ら [7]	Kober ら [8]	Zhao ら [5]	Google [15]	全体
API メソッド数	20	19	29	35	380	447
Device or other IDs	0	16	1	8	171	185
Contacts	0	0	0	2	108	108
Location	0	5	19	18	84	105
Personal info	0	0	0	4	98	99
App info and performance	15	0	0	0	9	24
Audio files	0	0	5	0	19	24
Messages	0	0	0	2	11	11
Files and docs	0	0	0	0	10	10
Photos and videos	0	0	1	0	8	9
App activity	0	0	0	0	6	6
Health and fitness	0	0	0	2	0	2
DSS 対象外	5	0	3	3	0	10
採用	15	19	26	32	380	437

タカテゴリが列挙されているが，具体的な API メソッドは登場しない．もう 1 つは，DSS への情報提供をサポートするためのドキュメントページ “Declare your app’s data use” [15] であり，申告対象の各データカテゴリについて，アプリが使用した場合に申告しなければならない API メソッドやパーミッションを掲載している．API メソッドは 3 個と少数ではあるが得られた．また，パーミッションに対応づく API メソッドを Android ソースコード中を探索し抽出した（表 3）．

5. 評価

構築した DSS データセットに対する評価と，Android フレームワークに対する評価について報告する．

5.1 DSS 対応テイントソースに対する分類性能

本稿で構築した DSS 対応テイントソースのデータセットに対して，CoDoC と DocFlow を適用し，現状のテイントソース識別性能を明らかにする．

5.1.1 セットアップ

DSS データセットの各 API メソッドのソースコードとドキュメントは，API レベル 30〜33 の Android フレームワークから抽出し，それぞれの API レベルに対する 2 手法の性能を評価する．DSS データセットの一部は，CoDoC や DocFlow のデータセットと重複しており，既に学習済であったため，DSS データセット全体に対する性能と，学習済を除いた場合の性能を報告する．また，DocFlow は BERT のファインチューニングあり・なしと 4 種類の分類器との組み合わせすべてを評価する．

5.1.2 結果

表 4 は，各データカテゴリについて，CoDoC と DocFlow それぞれがソースと判定した数と，カッコ内にその割合を示している．紙面の制約上，API レベル 32 に対する結果のみを載せており，DocFlow は最も高かったファインチューニング済 BERT とニューラルネットワークの結果を載せている．学習済を除いた場合は，除いた後のデータ数をスラッシュの右側に示している．

DocFlow は CoDoC をほとんどのデータカテゴリで上回っており，データセット全体を用いた場合で 12 ポイント，学習済を除く場合で 14 ポイント上回っている．データセット全体を用いた場合の App info and performance と Health and fitness，学習済を除いた場合の Health and fitness の 3 つは引き分けであり，学習済を除いた場合の Messages のみで CoDoC がわずかに上回っている．

DocFlow の学習済を除いた場合に注目すると，正解率 69%と比較的高く，Device or other IDs や Contacts ではそれぞれ 74%，79%，Personal info と App info and performance では 80%を超えている．Health and fitness は 100%であるが，データ数が 1 個と少ないため，十分な評価とは考えられない．一方で，Audio files，Files and docs，Photos and videos，App activity の 4 データカテゴリでは，20〜40%台と低い．以上より，70%を超えるデータカテゴリが半数近くある一方で，40%台以下のデータカテゴリも半数近くあり，得意なデータカテゴリと不得意なデータカテゴリがはっきりと分かれている．アプリ解析者が調査対象とするデータカテゴリが，CoDoC や DocFlow にとって得意なものであった場合，CoDoC や DocFlow を用いてテイントソースを自動的に識別してテイント解析へ利用できる可能性がある．実際にテイント解析を動かす評価は今後の課題とし，これに向けて，次の 5.2 節で Android 全体からテイントソースを抽出した結果を報告する．

表 4　DSS 対応テイントソースの自動識別結果（API レベル 32）
Table 4　Classification result against the DSS dataset (API level 32).

データカテゴリ	#	CoDoC		DocFlow	
		全体	学習済除く	全体	学習済除く
Device or other IDs	185	103 (0.56)	96/176 (0.55)	137 (0.74)	136/184 (0.74)
Contacts	108	76 (0.70)	72/100 (0.72)	85 (0.79)	85/108 (0.79)
Location	105	50 (0.48)	40/ 94 (0.43)	57 (0.54)	53/ 96 (0.55)
Personal info	99	78 (0.79)	73/ 93 (0.78)	86 (0.87)	85/ 98 (0.87)
App info and performance	24	20 (0.83)	16/ 20 (0.80)	20 (0.83)	20/ 24 (0.83)
Audio files	24	9 (0.38)	7/ 22 (0.32)	11 (0.46)	9/ 22 (0.41)
Messages	11	5 (0.45)	5/ 8 (0.63)	6 (0.55)	6/ 11 (0.55)
Files and docs	10	2 (0.20)	1/ 9 (0.11)	5 (0.50)	3/ 8 (0.38)
Photos and videos	9	2 (0.22)	1/ 8 (0.13)	4 (0.44)	2/ 7 (0.29)
App activity	6	1 (0.17)	1/ 6 (0.17)	2 (0.33)	2/ 6 (0.33)
Health and fitness	2	2 (1.00)	1/ 1 (1.00)	2 (1.00)	1/ 1 (1.00)
全体	437	248 (0.57)	220/402 (0.55)	300 (0.69)	292/424 (0.69)

5.2　Android フレームワークに対する分類

CoDoC と DocFlow を用いて Android フレームーク中のすべての API メソッドからテイントソースを抽出する．これまでのアプリ解析で利用されるテイントソースは多くても数十個程度である．CoDoC や DocFlow がこれより大幅に多くのテイントソースを抽出した場合，実際の解析において多量の情報フローの追跡が発生することが予想され，テイント解析の所要時間の増加や，テイント解析結果の検証コストの増大が懸念される．本稿では，CoDoC と DocFlow がどの程度の数の API メソッドをテイントソースとして出力するのかを調査し報告する．

5.2.1　セットアップ

Android フレームワークの API レベルは，CoDoC の対象である API レベル 30 から，CoDoC が適用可能な API レベル 33 までの 4 バージョンとする．現時点で API レベル 34（Android 14）が最新であるが，CoDoC の抽出器が非対応であったため，除外した．DocFlow は，5.1 節に合わせて，ファインチューニング済 BERT とニューラルネットワークの結果を報告する．

CoDoC と DocFlow は，それぞれ自身のデータセットで学習を行ったものと，本稿で構築した DSS データセットを含めて学習を行ったものの 2 種類を用いて，DSS データセットの学習前後の結果を比較する．学習における DSS データセットの API レベルは，CoDoC と DocFlow それぞれのデータセットの API レベル 30 と 32 に合わせた．

5.2.2　結果

表 5 は，API レベル 30 から 33 の Android フレームワークに対して，それぞれのアプローチが DSS の学習前後でテイントソースとして識別した API メソッドの数と，カッコ内に割合を示している．DSS 対応テイントソースの学習前後で，CoDoC と DocFlow 共に，フレームワーク全体に対してテイントソースと識別された API メソッドの割合は 10 ポイント以上増加しており，大きく変化している．

表 5　Android フレームワーク全体に対する分類結果
Table 5　Classification result against the Android framework.

API	CoDoC		DocFlow	
	DSS 学習前	DSS 学習後	DSS 学習前	DSS 学習後
30	9,918 (0.21)	16,047 (0.34)	17,673 (0.37)	22,888 (0.48)
31	11,282 (0.21)	19,048 (0.36)	19,775 (0.38)	25,482 (0.48)
32	11,500 (0.22)	19.793 (0.37)	19,883 (0.38)	25,627 (0.48)
33	11,522 (0.21)	19,471 (0.35)	20,759 (0.37)	26,710 (0.48)

テイントソースの割合は，DocFlow の方が CoDoC より 15 ポイント前後高い．DocFlow は，DSS 学習前で 37〜38%，DSS 学習後では 48%をテイントソースと識別しており，テイントソースと判定する傾向が強いことが表れている．この傾向は，予備調査（表 2）でも表れている．DSS 対応テイントソースの評価（5.1 節）では，DocFlow は CoDoC を上回っていたが，テイントソースを正確に識別できるのではなく，テイントソースと判定する傾向が強いためであると考えられる．この傾向が実際のテイント解析に与える影響について調査し，DocFlow と CoDoC の実用性をさらに明らかにすることが今後の課題である．

6.　制限と今後の課題

4 章で述べた 5 つの収集源から DSS データセットを構築した結果，DSS 申告対象の 14 データカテゴリ中 3 つに関して API メソッドを得られなかった．また，例えば Health and fitness カテゴリは API メソッドが 2 個と少ない．多数の API メソッドが得られたデータカテゴリであっても，そのカテゴリの特徴を表すのに十分な API メソッドが得られていない可能性がある．これらへの対応は，DSS データセット構築における今後の課題である．また，Google 公式のドキュメントから抽出したパーミッションについて，対

応する API メソッドを Android フレームワークから抽出してテイントソースとした．しかし，DSS 関連のパーミッションで保護されている API メソッドであっても，重要なデータを返さない場合も考えられる．パーミッションから収集したテイントソースの扱いについて今後検討する．また，我々が実施した API メソッドと DSS データカテゴリとの対応づけが不適切な可能性を考慮し，DSS データセットを公開して第三者による検証を促す．

DSS データセットの構築において除外したデータベースやクリップボード関連の API メソッド（4 章）は，場合によってはテイントソースとなり得る．また既存研究では，ユーザ入力フォーム [20] やサードパーティ SDK が独自生成の識別子を外部ストレージへ保存し永続化するケース [21]，プライバシーデータがサーバ側から送られるケース [22]，アプリやサードパーティライブラリが独自に定義した API [23] などが検討され，我々の今後の課題とする．

DocFlow はドキュメントのみに基づいて識別を行うため，オープンソースではない Google Play Services ライブラリなどにも適用できる [12]．また，クロスプラットフォームフレームワーク [24] や Android デバイスメーカーが定義するプライベート API [25] なども存在する．本稿は Android フレームワークのみにフォーカスしており，今後スコープを広げる予定である．

SuSi，CoDoC，DocFlow の中で，SuSi と DocFlow はテイントソースに紐づくデータのカテゴリ（例．位置情報や識別子）を識別できる．本稿は，まずは基本となるテイントソースかそうでないかの判定にフォーカスしている．

7. 関連研究

テイント解析はバックドアや脆弱性の検出といったセキュリティ目的にも用いられ，security-relevant method（SRM）を自動的に識別する研究が発展してきた．Android プラットフォームを対象としたものとして，Rodrigues ら [26] は SRM 分類問題に対して，モノリシック分類器や複数分類器システム，2 値-実数値特徴量変換埋め込みアルゴリズムの性能を評価した．

SWAN [27] は Java プログラムを対象とした，機械学習を用いた全自動の SRM 識別手法である．SWAN ASSIST [27], [28] はアプリ開発者と連携してフィードバックを統合し，アプリ開発者を支援する半自動の SRM 識別ツールである．また，Dev-Assist [29] は SWAN の拡張であり，SRM 間の依存関係を考慮したマルチラベル機械学習を用いている．Java ライブラリに対して SRM の自動抽出に取り組んだ研究 [30] では，SVM に加えて，k 近傍法や Gradient Boosting を評価している．

JavaScript ライブラリのテイントスペシフィケーションの自動的生成のために，Staicu ら [31] は動的解析に基づく手法を提案し，Dutta ら [32] は既存の機械学習アプローチと手動のテイントモデリングを組み合わせた．Seldon ら [33] は Python の大規模なプログラムからテイントスペシフィケーションを推測する手法を提案した．

SinkFinder [34] は，潜在的にバグを引き起こす可能性のある関数を Linux や OpenSSL などの大規模なシステムから発見するために，サブワード単語埋め込みにより少量の学習サンプルを増量し，SVM を学習させる．類似して，Huang ら [35] は，バグが発生しやすいが呼び出し回数が少ないセンシティブ関数を検出するために，関数とコンテキストを 1 対のベクトルに埋め込む手法を提案した．また，Chiang ら [36] も少量の API リストを元に，Knowledge Graph や機械学習を用いてテイントスペシフィケーションを生成し，Amazon CodeGuru Reviewer サービスでのバグや脆弱性の検出に取り組んだ．

以上に挙げたセキュリティ目的で提案された手法を，プライバシー向けに応用することで，テイントソース自動識別性能を向上できる可能性がある．

8. まとめ

本稿は，DSS 対応テイントソースの自動識別に向けて，DSS データセットを構築し，近年発表された CoDoC と DocFlow の性能を評価した．結果，特に DocFlow は，いくつかのデータカテゴリに対して高い再現率で識別できることが明らかとなった．今後は，2 手法により識別されたテイントソースの実用性を評価するために，これらを用いて実際にテイント解析を動かし，手動で識別したテイントソースを用いた場合と比較する．加えて，DSS データセットの収集源を増やし，改良を進める．

謝辞　本研究は JSPS 科研費 JP24K23863，JP20H05706 の助成を受けた．

参考文献

[1] Andow, B., Mahmud, S. Y., Wang, W., Whitaker, J., Enck, W., Reaves, B., Singh, K. and Xie, T.: PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play, *28th USENIX Secur. Symp.*, pp. 585–602 (2019).

[2] Xiang, A., Pei, W. and Yue, C.: PolicyChecker: Analyzing the GDPR Completeness of Mobile Apps' Privacy Policies, *Proc. 2023 ACM SIGSAC Conf. Computer Communications Secur.*, p. 3373–3387 (2023).

[3] Zhou, L., Wei, C., Zhu, T., Chen, G., Zhang, X., Du, S., Cao, H. and Zhu, H.: POLICYCOMP: Counterpart Comparison of Privacy Policies Uncovers Overbroad Personal Data Collection Practices, *32nd USENIX Secur. Symp.*, pp. 1073–1090 (2023).

[4] Slavin, R., Wang, X., Hosseini, M. B., Hester, J., Krishnan, R., Bhatia, J., Breaux, T. D. and Niu, J.: Toward a framework for detecting privacy policy violations in Android application code, *Proc. 38th Int. Conf. Softw. Eng.*, p. 25–36 (2016).

[5] Zhao, K., Zhan, X., Yu, L., Zhou, S., Zhou, H., Luo, X., Wang, H. and Liu, Y.: Demystifying Privacy Policy of

Third-Party Libraries in Mobile Apps, *Int. Conf. Softw. Eng.*, pp. 1583–1595 (2023).

[6] Arkalakis, I., Diamantaris, M., Moustakas, S., Ioannidis, S., Polakis, J. and Ilia, P.: Abandon All Hope Ye Who Enter Here: A Dynamic, Longitudinal Investigation of Android's Data Safety, *USENIX Secur. Symp.*, (online), available from ⟨https://www.usenix.org/system/files/sec24fall-prepub-399-arkalakis.pdf⟩ (2024).

[7] Inayoshi, H., Kakei, S. and Saito, S.: Detection of Inconsistencies between Guidance Pages and Actual Data Collection of Third-party SDKs in Android Apps, *Int. Conf. Mobile Softw. Eng. Syst.*, p. 43–53 (2024).

[8] Kober, M., Samhi, J., Arzt, S., Bissyandé, T. F. and Klein, J.: Sensitive and Personal Data: What Exactly Are You Talking About?, *Int. Conf. Mobile Softw. Eng. Syst.*, pp. 70–74 (2023).

[9] Arzt, S., Rasthofer, S. and Bodden, E.: SuSi: A Tool for the Fully Automated Classification and Categorization of Android Sources and Sinks, Technical report, University of Darmstadt (2013).

[10] Rasthofer, S., Arzt, S. and Bodden, E.: A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks, *Proc. Network Distributed Syst. Secur. Symp.*, (online), available from ⟨http://dx.doi.org/10.14722/ndss.2014.23039⟩ (2014).

[11] Samhi, J., Kober, M., Kabore, A. K., Arzt, S., Bissyandé, T. F. and Klein, J.: Negative Results of Fusing Code and Documentation for Learning to Accurately Identify Sensitive Source and Sink Methods : An Application to the Android Framework for Data Leak Detection, *Int. Conf. Softw. Analysis Evolution Reengineering*, pp. 783–794 (2023).

[12] Tileria, M., Blasco, J. and Dash, S. K.: DocFlow: Extracting Taint Specifications from Software Documentation, *Proc. Int. Conf. Softw. Eng.* (2024).

[13] JordanSamhi: CoDoC, `https://github.com/JordanSamhi/CoDoC`. (Accessed 2024-08-06).

[14] (S3Lab), S. . S. S. L.: docflow, `https://gitlab.com/s3lab-rhul/android/docflow`. (Accessed 2024-08-06).

[15] Google: Declare your app's data use, `https://developer.android.com/privacy-and-security/declare-data-use`. (Accessed 2024-05-24).

[16] GiannisArk: USENIX24_DataSafety, `https://github.com/GiannisArk/USENIX24_DataSafety`. (Accessed 2024-05-24).

[17] JordanSamhi: SensitiveData, `https://github.com/JordanSamhi/SensitiveData`. (Accessed 2024-05-24).

[18] Zhao, K.: Artifacts for Demystifying Privacy Policy of Third-Party Libraries in Mobile Apps, `https://zenodo.org/records/8321054`. (Accessed 2024-05-24).

[19] Google: Provide information for Google Play's Data safety section, `https://support.google.com/googleplay/android-developer/answer/10787469`. (Accessed 2024-05-24).

[20] Wang, X., Qin, X., Hosseini, M. B., Slavin, R., Breaux, T. D. and Niu, J.: GUILeak: tracing privacy policy claims on user input data for Android applications, *Proc. 40th Int. Conf. Softw. Eng.*, p. 37–47 (2018).

[21] Dong, Z., Liu, T., Deng, J., Wang, H., Li, L., Yang, M., Wang, M., Xu, G. and Xu, G.: Exploring Covert Third-party Identifiers through External Storage in the Android New Era, *USENIX Secur. Symp.*, (online), available from ⟨https://www.usenix.org/system/files/sec24summer-prepub-442-dong.pdf⟩ (2024).

[22] Nan, Y., Yang, Z., Wang, X., Zhang, Y., Zhu, D. and Yang, M.: Finding Clues for Your Secrets: Semantics-Driven, Learning-Based Privacy Discovery in Mobile Apps, *Proc. Network Distributed Syst. Secur. Symp.*, (online), available from ⟨http://dx.doi.org/10.14722/ndss.2018.23092⟩ (2018).

[23] Zhang, X., Heaps, J., Slavin, R., Niu, J., Breaux, T. and Wang, X.: DAISY: Dynamic-Analysis-Induced Source Discovery for Sensitive Data, *ACM Trans. Softw. Eng. Methodol.*, Vol. 32, No. 4 (2023).

[24] Chen, H., Chen, D., Liu, Y., Sun, X. and Li, L.: Are Your Android App Analyzers Still Relevant?, *Proc. Int. Conf. Mobile Softw. Eng. Systems*, p. 69–73 (2024).

[25] Vyas, P., Waheed, A., Aafer, Y. and Asokan, N.: Auditing Framework APIs via Inferred App-side Security Specifications, *32nd USENIX Secur. Symp.*, pp. 6061–6077 (2023).

[26] Rodrigues, W. M., Walmsley, F. N., Cavalcanti, G. D. C. and Cruz, R. M. O.: Security Relevant Methods of Android's API Classification: A Machine Learning Empirical Evaluation, *IEEE Transactions on Computers*, Vol. 72, No. 11, pp. 3273–3285 (2023).

[27] Piskachev, G., Do, L. N. Q. and Bodden, E.: Codebase-adaptive detection of security-relevant methods, *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Testing Analysis*, p. 181–191 (2019).

[28] Piskachev, G., Nguyen Quang Do, L., Johnson, O. and Bodden, E.: SWAN$_{\text{ASSIST}}$: Semi-Automated Detection of Code-Specific, Security-Relevant Methods, *Int. Conf. Automated Softw. Eng.*, pp. 1094–1097 (2019).

[29] Johnson, O., Piskachev, G., Krishnamurthy, R. and Bodden, E.: Detecting Security-Relevant Methods using Multi-label Machine Learning, *2nd IDE Workshop* (2024).

[30] Sas, D., Bessi, M. and Arcelli Fontana, F.: Automatic Detection of Sources and Sinks in Arbitrary Java Libraries, *IEEE 18th Int. Working Conf. Source Code Analysis Manipulation*, pp. 103–112 (2018).

[31] Staicu, C.-A., Torp, M. T., Schäfer, M., Møller, A. and Pradel, M.: Extracting taint specifications for JavaScript libraries, *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, p. 198–209 (2020).

[32] Dutta, S., Garbervetsky, D., Lahiri, S. K. and Schäfer, M.: InspectJS: leveraging code similarity and user-feedback for effective taint specification inference for JavaScript, *Proc. 44th Int. Conf. Softw. Eng.: Softw. Eng. Practice*, p. 165–174 (2022).

[33] Chibotaru, V., Bichsel, B., Raychev, V. and Vechev, M.: Scalable taint specification inference with big code, *Proc. 40th ACM SIGPLAN Conf. Programming Language Design and Implementation*, p. 760–774 (2019).

[34] Bian, P., Liang, B., Huang, J., Shi, W., Wang, X. and Zhang, J.: SinkFinder: harvesting hundreds of unknown interesting function pairs with just one seed, *Proc. 28th ACM Joint Meeting European Softw. Eng. Conf. Symp. Foundations of Softw. Eng.*, p. 1101–1113 (2020).

[35] Huang, J., Nie, J., Gong, Y., You, W., Liang, B. and Bian, P.: Raisin: Identifying Rare Sensitive Functions for Bug Detection, *Proc. Int. Conf. Softw. Eng.* (2024).

[36] Chiang, W.-H., Li, P., Zhou, Q., Banerjee, S., Schaef, M., Lyu, Y., Nguyen, H. and Tripp, O.: Inference for Ever-Changing Policy of Taint Analysis, *Proc. Int. Conf. Softw. Eng.: Softw. Eng. Practice*, p. 452–462 (2024).