TAINTMINI: Detecting Flow of Sensitive Data in Mini-Programs with Static Taint Analysis

Chao Wang
The Ohio State University
wang.15147@osu.edu

Ronny Ko
The Ohio State University
ko.410@osu.edu

Yue Zhang
The Ohio State University
zhang.12047@osu.edu

Yuqing Yang
The Ohio State University
yang.5656@osu.edu

Zhiqiang Lin
The Ohio State University
zlin@cse.ohio-state.edu

***Abstract*— Mini-programs, which are programs running inside mobile super apps such as WeChat, often have access to privacy-sensitive information, such as location data and phone numbers, through APIs provided by the super apps. This access poses a risk of privacy sensitive data leaks, either accidentally from carelessly programmed mini-programs or intentionally from malicious ones. To address this concern, it is crucial to track the flow of sensitive data in mini-programs for either human analysis or automated tools. Although existing taint analysis techniques have been widely studied, they face unique challenges in tracking sensitive data flows in mini-programs, such as cross-language, cross-page, and cross-mini-program data flows. This paper presents a novel framework, TAINTMINI, which addresses these challenges by using a novel universal data flow graph approach that captures data flows within and across mini-programs. We have evaluated TAINTMINI with 238,866 mini-programs and detected 27,184 that contain sensitive data flows. We have also applied TAINTMINI to detect privacy leakage colluding mini-programs and identify 455 such programs from them that clearly violate privacy policy.**

***Index Terms*—Mini-programs, Taint analysis, Privacy leaks detection, Security, Empirical Study**

I. INTRODUCTION

A new mobile computing paradigm, dubbed mini-app paradigm, has been growing rapidly in recent years among highly popular social apps, such as WECHAT, TIKTOK, and SNAPCHAT. In this paradigm, a host app allows its users to install and run mini-apps (or mini-program called by Tencent and Alibaba, quickapp by Huawei/Xiaomi, and smart mini program by Baidu, according to a W3C white paper [1]) inside the host itself [2]. The mini-apps, which behave just like native apps, have enabled the host app to build an ecosystem around (much like Google Play and Apple App Store), enrich the host-app's functionalities with various services (e.g., social e-commerce and ride-hailing), and offer mobile users elevated convenience [3]. For example, PINDUODUO, a social e-commence provider, has benefited significantly from this paradigm: now merchants sell their products directly through WECHAT using their mini-programs, allowing potential customers to browse the products, share interests with their social network, and make purchases without leaving WECHAT [4, 5]. Today, WECHAT has hosted more than four million mini-programs [6], whereas Google Pay has about three million mobile apps [7].

One reason for the success of mini-programs can be attributed to the abundant data collected by the host app. For instance, super apps such as WECHAT usually have collected a huge amount of privacy-sensitive data (e.g., phone number, user's home address, and location data). To further enhance user's experience, these sensitive data have become accessible to mini-programs by the super app's APIs [8]. To protect these sensitive data from being leaked, super-apps have introduced various security mechanisms such as permission-based access control [9] to allow the collected sensitive data to be accessed only by authorized mini-programs.

Now that privacy sensitive data can be directly accessed by mini-programs through super app provided APIs, unavoidably they can be leaked accidentally by carelessly programmed mini-programs or intentionally by malicious mini-programs. A well-known technique to solve this problem is taint analysis. However, existing taint analysis for mobile apps such as TaintDroid [10] and FlowDroid [11] cannot be directly applied, since taint analysis for mini-programs introduces new challenges. First, unlike programs solely implemented in a single programming language, mini-programs are developed with multiple languages. For example, a WECHAT mini-program usually contains at least two types of programming languages: WXML (a markup language for the design of the UI), and JavaScript. Second, since the purpose of most mini-programs is to serve interactive user requests, their programming logic heavily uses asynchronous event handlers, such as from the initial app launch (e.g., onLaunch) to the service-ready state (e.g., onReady). Once a mini-program becomes ready for service, its processing of user requests is mediated by developer-implemented callback functions registered under up to 195 mini-program API events and 128 WXML tag events [12]. Third, data flow analysis within a mini-program page is insufficient, because data can flow across different pages when a user navigates a mini-program. Finally, data can flow across different mini-programs when a mini-program redirects a user to another one (e.g., from a shopping mini-program to a payment mini-program).

To advance the state of the art, we present TAINTMINI, a static taint analysis framework to track the flow of sensitive data in inter-events, inter-pages, and inter-apps in mini-programs. At a high level, TAINTMINI runs taint tracking for data flows across 4

domains: i) data flows between the webview layer (WXML) and the logical layer (JavaScript); ii) data flows between asynchronous event handlers; iii) data flows between different pages within the same mini-program; and iv) data flows between different mini-programs. The key idea in TAINTMINI is to build a *universal* data flow graph, which is inspired by the JavaScript object dependency graph (ODG) [13] initially generated by statically analyzing a target program's JavaScript to illustrate data read/write dependencies among variables, objects, and object properties. However, ODGen does not support the analysis of asynchronous callback functions as well as the cross domain taint flows. As such, TAINTMINI significantly extends ODG by further considering the interactions between JavaScript and WXML, assigning new graph nodes for such interacting WXML tags and their attributes, and incorporating them into JavaScript's data flow graph. TAINTMINI also proposes a novel concept of *event groups* (a set of WXML & JavaScript data objects that are to be processed synchronously in a given mini-program's logic), based on which TAINTMINI generates the event group execution order graph. This graph is used to determine whether any given pair of asynchronous event handlers in a mini-program has a deterministic order of execution or a flexible order of execution. This collected knowledge is essential to deciding the possibility of various data flows across asynchronous event handlers. Finally, TAINTMINI tracks inter-page and inter-app data flows by analyzing the arguments of the mini-program APIs that process such actions (e.g., wx.navigateTo which navigates to a different page, or wx.naviagteToMiniProgram which navigates to a different mini-program).

Having the ability to track the flow of sensitive data, TAINTMINI can be used in many applications such as detecting cartographic key misues [14] or identifying potential privacy leaks, e.g., location data sent through the network. However, it is challenging to tell whether the leak is a violation of the user's intention [15], since many apps, e.g., location-based service apps, need to collect those data for better services, and the users are explicitly asked to grant the location access permissions. Therefore, automatic tools cannot directly detect the privacy leakage unless there are well-defined security and privacy policies. However, in this paper, we demonstrate how to use TAINTMINI to automatically detect the privacy leaks caused by colluding mini-programs given its "clear boundary". In particular, in collusion attacks, the user only gives the permission to the authorized mini-programs, and she has never expected that the data is transmitted to other mini-programs which do not have the corresponding permissions.

Contributions. In short, we make the following contributions:

- **Novel Techniques (§IV).** We propose TAINTMINI as the first static mini-program taint analysis framework to track the flow of sensitive data across different domains with a novel universal data flow graph based approach.
- **Empirical Evaluation (§V).** We have implemented TAINTMINI and demonstrated its performance and effectiveness for detecting 27,184 (11.38%) real-world mini-programs out of 238,866 mini-programs containing sensitive data flows.
- **Automated Application (§VI).** While TAINTMINI can have many applications, we particularly show how to apply it for the automated detection of collusion attacks where one mini-program legitimately accesses the sensitive data but delivers to others that do not have the permissions. We have identified 455 such colluding mini-programs in our tested dataset.

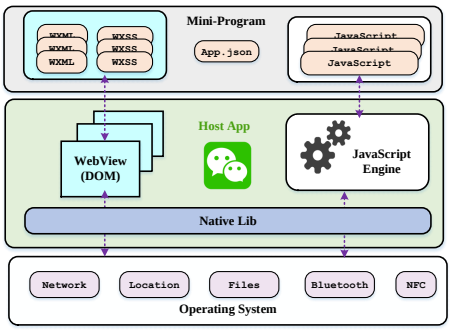

Fig. 1: The Architecture of Mini-Programs.

II. BACKGROUND

The Architecture of Mini-Programs. A mini-program typically consists of (1) a front-end running on the host app that interacts with users and accesses the resources offered by the host app as well as the underlying operating systems, and (2) a back-end communicating with the front-end and providing online services. As shown in Figure 1, similarly to native Android apps, the front-ends of mini-programs are compressed and distributed in files with specific formats, e.g., WECHAT mini-programs are formatted as WXAPKG files. A packed WXAPKG file mainly consists of four types of files: (i) a JSON file named app.json containing the basic information and configurations (e.g., , the permissions a mini-program requires) of the mini-program; (ii) one or multiple JavaScript (JS) files that use APIs to access the resources such as network communications and location data; (iii) one or multiple WXML (a Marking Language, WECHAT version of programmable HTML) files specifying the user interfaces (UIs) such as the layout of input boxes and clickable buttons; and (iv) one or multiple WXSS (Style Sheet, the WECHAT version's CSS) files to format the UIs (e.g., the font size and color).

A mini-program's WXML and WXSS files are processed by the rendering layer via a WebView Engine, whereas its JavaScript files are processed by the logical layer (i.e., the JavaScript Engine). However, WXSS is mainly used to format the UIs. Therefore, the data flows are usually observed between WXML and JavaScript files. From this perspective, the mini-program framework is analogous to that of the conventional client-side web framework which comprises HTML and JavaScript, and allows a webpage to orchestrate dynamic interactions between its DOM state (HTML) and JavaScript state (JavaScript). For example, a webpage's JavaScript can read a user's input from an HTML node, process it, and write the result to another HTML node for graphical display. Similarly, a mini-program also allows dynamic interactions between its webview layer (WXML) and the logical layer (JavaScript) for similar purposes.

The Sensitive Data and Access Control. Mini-programs can consume two categories of sensitive data through the APIs offered by the host app. The first category is the data

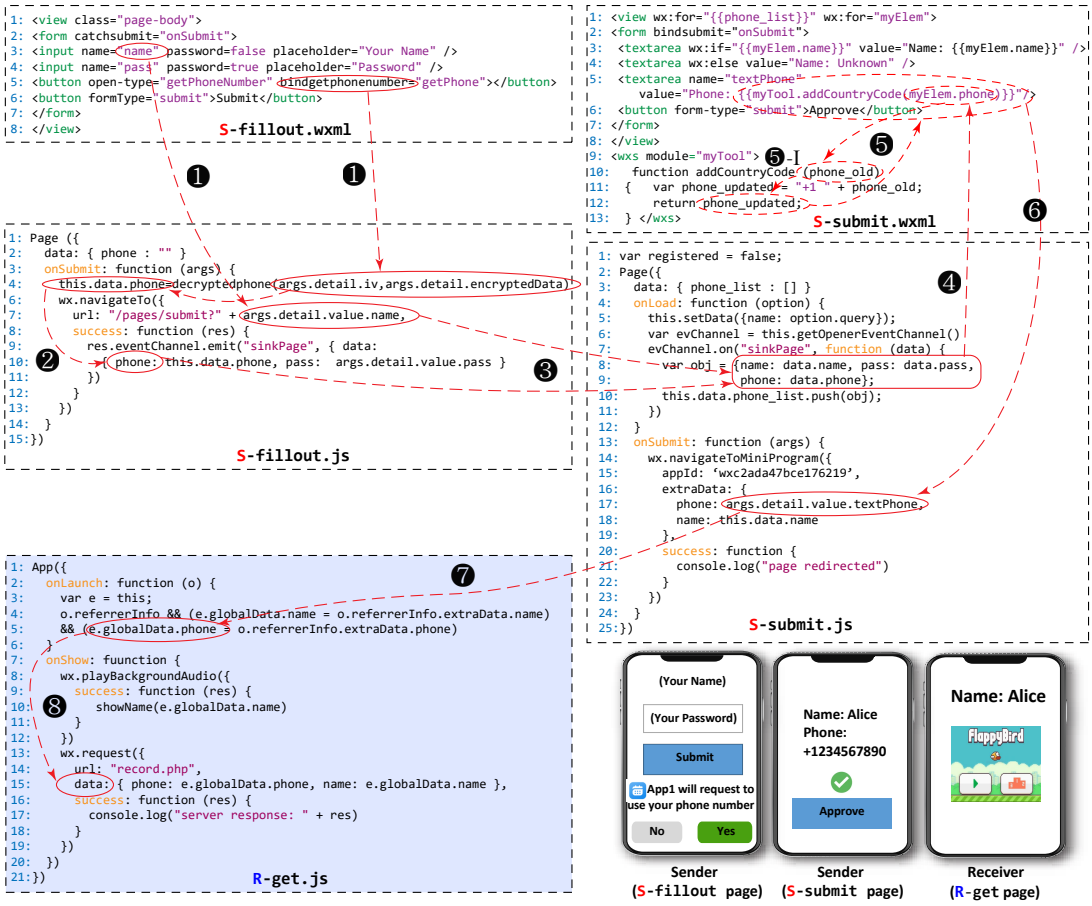

Fig. 2: Code Snippet of Cross Mini-Program Sensitive Data Leakage

(e.g., location or Bluetooth) guarded by the OS permission mechanisms: Before using it, the mini-program must request the authorization from the user, even though the host app has already been granted with the corresponding permissions (e.g., declared in `Androidmanifest.xml`). The second category is the data explicitly collected from the user inputs (e.g., phone numbers) or derived from the user's behaviors (e.g., steps walked daily) by the host app. These data are usually protected by both permission mechanisms and encryption. For instance, to access data such as `UserInfo`, a mini-program needs to declare permission `scope.userinfo` in its `app.json` file, or dynamically pop up a dialogue to ask the user to grant this permission.

Cross Mini-Program Communication. A mini-program is usually designed to achieve a specific task, and cannot integrate too many services due to its limited size [16]. Also, the larger a mini-program is, the longer the download time will be, which hurts the user's experience of using the mini-program (because users intend to use mini-programs for their install-less and storage-saving benefits [3]). To achieve the right balance, the super-apps have introduced cross-mini-program communication for data exchange. For example, a shopping mini-program allows users to browse goods, and when a product selection is made, the shopping mini-program will navigate the user to a payment mini-program via API navigateToMiniProgram to complete the purchase. A JSON object extraData containing key value pairs is used by this API to transmit the data.

Taint Analysis. Taint analysis [17] is a program analysis technique that tracks the data flows of interest in a given program. The primary elements of taint analysis are taint sources, taint propagations, and taint sinks. Taint sources are the entry points of the data flows where data starts to propagate, and taint sinks are their end points where the data finishes to propagate. Taint analysis is a technique that traces the flow of sensitive data, known as "taint", from its source to its final destination (or "sink"), in a given program. It has numerous applications, including the detection of exploits [18], privacy leaks [10, 19], and the misuse of cryptographic keys [14]. By identifying the path that taint data takes, taint analysis helps to safeguard against potential security threats and protect sensitive information.

III. Running Example and Challenges

In this section, we first present a running example of mini-programs (§III-A), and then discuss its key differences compared against mobile apps and web apps (§III-B) to clearly summarize the challenges we faced in designing our taint analysis for mini-programs (§III-C). Finally, we define the scope of this work (§III-D).

A. Running Example

To clearly illustrate the challenges of tracking sensitive data flows in mini-programs, we provide a running example of tracking the data flows between two communicating mini-programs, as shown in Figure 2. The sender mini-program S, which initiates a cross-mini-program request, first retrieves the user's phone number after being granted with the corresponding permission, and then asks the user to enter her username and password for the service sign-up. During this process, the user is aware that her phone number, username, and password will be collected by S and sent over to the network to sign up. In the following, we describe this process in seven steps, each of which represents a unique type of data flow.

❶ **User Input (WXML→JS).** The S-fillout.wxml page accepts the user's inputs (e.g., sensitive data) into the text box of the <input> tags and passes them to the JavaScript variables. First, as the user clicks the GetPhoneNumber button, S-fillout.js's getPhoneNumber callback function is called and decrypts the phone number. Then, the user clicks the Submit button and the entered strings are passed from S-fillout.wxml's <form> tag to the onSubmit callback function.

❷ **Data Transmission across Pages (JS→JS).** The username, phone number, and password are passed from S-fillout.js to S-submit.js via API navigateTo (page redirection within the same mini-program). During this process, the fillout page (comprised of S-fillout.wxml and S-fillout.js) visually closes the submit page (comprised of S-submit.wxml and S-submit.js) opens.

❸ **Data Transmission across Callbacks I (Handler→Handler).** The fillout page's onSubmit callback handler calls the navigateTo API to navigate to the submit page and passes the user's phone number to the navigateTo.success callback handler.

❹ **Data Display on a Page (JS→WXML).** S's submit page displays the received username and phone number to the same page's WXML tags by storing them in the mini-program framework's specially reserved Page.data property (bound to developer-specified WXML tags).

❺ **Data Processing within WXML (WXML→WXML).** S-submit.wxml runs its JavaScript module defined in its <wxs> tag to prepend the country code +1 to the received phone number.

❻ **Transmission of WXML's Processed Data (WXML→JS).** As the user clicks the Approve button, S-submit.wxml sends the updated (i.e., prepended with +1) phone number to S-submit.js as the onSubmit callback function's argument.

❼ **Data Transmission across Mini-programs (Mini-Program→Mini-Program).** S opens R, the receiver mini-program (i.e., the FlappyBird game), by calling API navigateToMiniProgram (context switch to another mini-program) and *stealthily* transmits the user's phone number to R. While the user sees that her name is displayed on the welcome page of R, she is not aware (and does not expect) that her phone number is transmitted to R during this process. This is clearly a privacy leak by the collusion of two mini-programs.

	Mini-Programs	**Mobile apps**	**Web Apps**
UI Layer Language	WXML	XML	HTML
Supports Script Execution?	✓	✗	✓
Supports In-line Tag Logic?	✓	✗	✗
Data Flows Across UIs?	✓	✗	✗
Dynamically Modifiable?	✗	✗	✓
Logic Layer Language	JavaScript	Java/Objective-C	JavaScript
OOP Script?	✓	✗	✓
Compile-free?	✓	✗	✓
Dynamically Typed?	✓	✗	✓
Information Flow Types	JS↔JS JS↔WXML App↔App EH↔ EH **WXML↔WXML**	Java↔Java Java↔XML App↔App EH↔ EH	JS↔JS JS↔HTML App↔App EH↔EH

TABLE I: Comparison between mini-programs, mobile apps and web apps."EH" represents event handler.

❽ **Data Transmission across Callbacks II (Handler→Handler).** Finally, R sends the obtained phone number over the network via API wx.request. The data flow goes from one API event handler onLaunch (which is invoked every time a mini-program is opened by the user) to another API wx.playBackgroundAudio's callback handler success (which is invoked when the background music is successfully played). In that callback, wx.request sends the sensitive data to the network.

B. Comparison with Web Apps and Mobile Apps

As shown in the the workflow of the running example in §III-A, mini-programs share similarities with other apps (e.g., similar to web apps, mini-programs are programmed with JavaScript). However, there are still substantial differences among them. In the following, we discuss their key differences from the developer's perspective based on how the app is developed. At a high level, all three types of apps have the UI layer (which specifies how the interface visually looks) and the logic layer (which specifies how the app behaves). Therefore, we discuss the key differences based on these two aspects:

- **UI Layer** represents how the UI elements (e.g., buttons, input boxes) are laid out and specifies how users interact with them. Similar to web apps that use HTML and native apps that use XML, mini-programs in WeChat use a special type of markup language named WXML to describe their UIs. Although the formats and syntax of these three languages are similar (e.g., they all use tags such as buttons to represent specific UI elements), there are several differences. First, dynamically adding or removing UI elements is not allowed in mini-programs, but in web apps, the developers can do so (e.g., removing a button) from their logic layer by directly accessing the Document Object Model (DOM) tree of the UI layout. This is because the UI layer of web apps is handled by the logic layer (e.g., `JSCore`), which can update the DOM tree, but the UI layer of mini-programs is exclusively handled from the UI engine (e.g., `XWeb` of WeChat). Second, for web apps and native apps, the UI layers rely on the code from the logic layer to render the UI elements. Although the UI layers of web apps may contain a special tag named `script`, which directly includes JavaScript code, the extensible markup languages do not have the programming capabilities (i.e., in-line tag logic).

On the other hand, the mini-program's WXML can allow programmable branches and loops, and can even implement simple rendering logic (e.g., fetching data from a list and displaying them), as shown in lines 3-5 in *S*-submit.wxml in Figure 2. Due to such a new feature, there could be data flows directly generated between UIs.

- **Logic Layer** specifies how to execute the mini-programs when user inputs are provided, or callback functions are triggered. Mobile apps use Java or Objective-C to implement the logic layer, but mini-programs and web apps use JavaScript. There are multiple differences between these two types of languages. First, Java and Objective-C are Object-Oriented Programming (OOP) languages that need to be complied, whereas JavaScript is an interpretation-based OOP script language. Second, Java and Objective-C are strongly typed, which means variables have to be declared with types before being first used, but JavaScript allows the use of variables without declaring their types. Finally, as highlighted in Table I, since the mini-programs allow the UI layer to have branches and loops, there could be data flows processed through the UIs directly.

C. Objective and Challenges

Our goal is to track the privacy-sensitive data flows of mini-programs to identify their potential leaks. To achieve this, we need to perform taint analysis on 3 types of data flows, each of which has a unique challenge not addressed by existing taint-tracking techniques:

1) We need to track data flows between WXML tags and JavaScript, because a mini-program user often provides her interactive input(s) via WXML tags, which are in turn processed by JavaScript and written back to some WXML tags. Such examples are step ❶ and ❸ in §III-A. Tracking this type of data flows is challenging, because such flow spans different programming language domains (WXML and JavaScript). In particular, mini-program's view layer template (.wxml) provides richer programming features than iOS or Android's one (.xml). First, WXML supports in-line tag logic embedded into tag attributes to dynamically resolve the values of each WXML tag's attributes and inner WXML (e.g., line 1, 3, and 4 in S-submit.wxml in Figure 3), similar to EJS's templating language. Second, WXML supports module scripts (<wxs>) which intercommunicate under various scopes (e.g., line 9 S-submit.wxml). With these two features, the mini-program framework's view layer component (.wxml) actively creates data flows within itself and exchanges them with the logic layer component (.js). This is different from the prior mobile app frameworks where the logic layer component uses the view layer component to simply read/write data like an I/O interface (storage). This is why mini-program's static data flow analysis is technically more challenging, because it requires full coordination of the view layer's WXML tags, their in-line tag logic, the embedded script modules (i.e., <wxs> tags), and the logic layer's JavaScript code.
2) We need to track data flows between asynchronous JavaScript callback functions, because the mini-program framework often requires developers to use multiple asynchronous JavaScript handlers to implement a service and process the corresponding data. Such an example is step ❼ in §III-A. Tracking this type of data flows is challenging, because the execution order of many asynchronous handlers is nondeterministic (i.e., we do not know which asynchronous callback function will be executed first, and which one is the next), and depending on their actual execution order, certain segment of data flow may or may not occur, which affects our analysis result.
3) We need to track data flows between multiple mini-program pages as well as multiple mini-programs, because mini-program users often navigate different pages of a mini-program while using it (as shown in step ❺), and multiple mini-programs also exchange messages while working on a collaborative task (as shown in step ❻). Tracking data flows across different pages and different mini-programs is challenging, because the data flow tracking technique should identify which pages & mini-programs will interact with, and how they will interact (i.e., through which functions and variables in their code).

Another challenge for improving the correctness of data flow analysis is to properly handle JavaScript aliases (i.e., different variables that point to the same target object). This is a well-known problem in most of the program analysis [20–22] and we adopt the solution proposed in the object dependency graph [23], which enforces the nodes of different aliases referring to the same JavaScript object to point to the same graph object node.

D. Scope

While there are several super apps today such as WeChat, AliPay, TikTok, and SnapChat, we particularly focus on Wechat for two key reasons. First, WeChat has the largest number of users (with 1.2 billion monthly active users), and any security bugs and vulnerability in this super app can have a striking impact. Second, WeChat pioneered the concept of mini-program paradigm, and so far it has more than 4.3 million mini-programs, which is way more than any other platforms combined (e.g., as in 2021, AliPay has about 120 thousand mini-programs [24], and SnapChat has only 62 mini-programs).

IV. Design

This section provides the detailed design of TaintMini, a framework to automatically and comprehensively track the sensitive data flows. As described in §III-C, data flows in mini-programs occur across various components (i.e., WXML, JavaScript) at various granularity (i.e., event handlers, mini-program pages, mini-program programs) in both synchronous & asynchronous manners. To address these challenges, TaintMini's high-level approach is to segment a mini-program's data flows into the field granularity of JavaScript object & WXML tag node, and then carefully connect those segments to represent data flows across event routines, pages, and mini-programs. In particular, TaintMini generates a data flow graph for each JavaScript event-handling function and WXML tag in each page, and then merges those graphs into one, which represents a universal data flow graph (UDFG) across all pages and all cross-communicating mini-programs.

Then, by scanning the graph, TAINTMINI determines whether there is any sensitive user data flow leaked. At a high level, TAINTMINI's taint-tracking analysis is comprised of three steps:

1) **UDFG Generation (§IV-A).** TAINTMINI first scans the target mini-program's unpacked files (primarily comprised of *.js, *.wxml code files and *.json configuration files). Then, based on them, TAINTMINI generates graph nodes, where each node represents the smallest granularity of data flow target (i.e., JavaScript object or WXML tag). Finally, TAINTMINI groups the data nodes into the *event groups*, and all those *event groups* form UDFG. In particular, each event group is defined to be one of following three: (i) an asynchronous JavaScript event handler routine; (ii) a set of WXML tags binding to the same JavaScript event handler routine; (iii) a JavaScript routine defined in a <wxs> tag block as module.
2) **Data-Flow Propagation (§IV-B).** Within each event group, TAINTMINI generates inside-JS-event edges (blue arrows in Figure 3) between data nodes (square boxes) according to conventional taint propagation rules for basic operations (e.g., assignment, function call/return, branch) [10], where each edge represents the flow of data. TAINTMINI further generates cross-event-group edges (yellow, red, and black arrows in Figure 3) according to TAINTMINI's universal data flow analysis rules described in Table II.
3) **Data-Flow Resolution (§IV-C).** TAINTMINI identifies the paths in the graph which contain a taint source and a taint sink. We define the taint sources to be the start of sensitive user resource (i.e., mini-program's JavaScript APIs in Table III or WXML input tags), and taint sinks to be mini-program's JavaScript APIs for network, or navigation to another mini-program. If a mini-program participating in a data flow does not possess the permission for the source mini-program's sensitive user resource, TAINTMINI reports this data flow to be a possible data leakage.

A. UDFG Generation

To generate the universal data flow graph (UDFG), TAINTMINI first scans the app.json configuration file in the unpacked mini-program's root directory, which specifies the path of the mini-program's entry file (e.g., app.js) and a list of paths for mini-program pages which contain each page's unpacked code files (i.e., *.wxml, *.wxss, *.js). Then, by scanning all JavaScript files, TAINTMINI generates a JavaScript abstract syntax tree (AST) for each mini-program page, and for each JavaScript object, TAINTMINI creates its corresponding data node (blue square box in Figure 3). Note that one mini-program page's logic may be implemented by multiple JavaScript files which include each other (by using the require and module.exports semantics), in which case TAINTMINI generates a single AST containing all of them because they constitute the same page. TAINTMINI also scans each mini-program page's WXML file, identifies WXML tags binding to JavaScript events, and creates their corresponding data nodes (yellow square boxes). Finally, TAINTMINI classifies each set of synchronously processed data nodes into the same event group (round boxes). Note that the same JavaScript object node can belong to multiple event groups (e.g., two event handler functions write to the same global variable). TAINTMINI ignores WXSS files, because those style files do not contribute to the data flow logic.

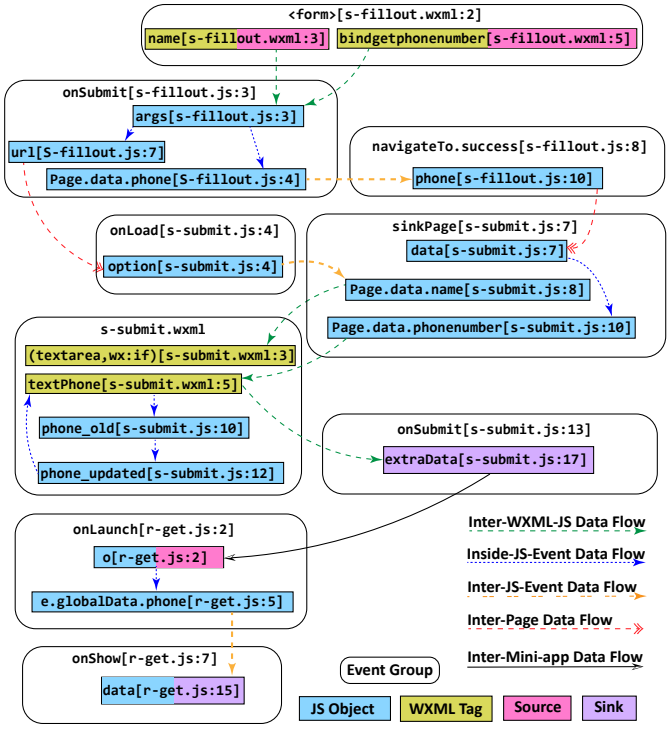

Fig. 3: The universal data flow graph for Figure 2's example.

By grouping all the events as groups, and connecting those events together, TAINTMINI produces the UDFG. Figure 3 is the universal data flow graph for Figure 2. The graph defines 4 types of data nodes: a source (pink box); a sink (purple box); a WXML tag (yellow box), and a JavaScript object (blue box). A round box containing multiple nodes represent an event group. Based on these data nodes, the graph also defines 5 types of data flow edges: (i) the data flows between a WXML tag and a JavaScript object (green arrow); (ii) the data flows between two JavaScript objects within the same JavaScript event handler function (blue arrow); (iii) the data flows between two JavaScript objects in different JavaScript event handler functions within the same mini-program page (yellow arrow); (iv) the data flows between two JavaScript objects across different mini-program pages (red arrow); and (v) the data flows between two JavaScript objects across different mini-programs (black arrow). If two event groups (round boxes) have a reachable path, their real-time execution order is deterministic; otherwise, their execution order is flexible.

B. Data-Flow Propagation

Having described the UDFG generation, we now explain how TAINTMINI tracks data propagation. Without loss of generality, Table II formalizes the procedure. At a high level, o_i is the data object (e.g., JavaScript variable, WXML tag). We also consider each field in an object or each bucket in an array as an object, which further reduces the granularity of data flow analysis and thereby avoids over-tainting. e_i is an event group. $s_p(i, j)$ is

Notations

$o_a \in \mathbb{O}$	: Data object o_a (in the data object universe $\mathbb{O}$)
$e_i \in \mathbb{E}$	: Event group e_i (in the event group universe $\mathbb{E}$)
$s_p^{(i,n)}$	: Event group e_i's *p-th* code statement, where n is the order of statement execution within e_i
$o_a \xrightarrow{s_p^{(i,n)}} o_b$	: $s_p^{(i,n)}$ creates the data flow from o_a to o_b
$e_i \ggg e_j$	: The application framework defines by design that e_j synchronously executes after e_i completes
$s_p^{(i,n)} \succ e_j$	: The application code's $s_p^{(i,n)}$ calls e_j

Rules

1. $(e_i \ggg e_j) \wedge (e_j \ggg e_k) \Longrightarrow e_i \ggg e_k$
2. $(s_p^{(i,n)} \succ e_j) \wedge (s_q^{(j,m)} \succ e_k) \Longrightarrow s_p^{(i,n)} \succ e_k$
3. $(o_a \xrightarrow{s_p^{(i,n)}} o_b) \wedge (o_b \xrightarrow{s_q^{(i,m)}} o_c) \wedge (n < m) \Longrightarrow o_a \rightarrow o_c$
4. $(e_i \ggg e_j) \wedge (o_a \xrightarrow{s_p^{(i,*)}} o_b) \wedge (o_b \xrightarrow{s_q^{(j,*)}} o_c)$ $\Longrightarrow o_a \rightarrow o_c$
5. $(s_p^{(i,n)} \succ e_j) \wedge (o_a \xrightarrow{s_r^{(i,*)}} o_b) \wedge (o_b \xrightarrow{s_q^{(j,*)}} o_c)$ $\Longrightarrow o_a \rightarrow o_c$
6. $(s_p^{(i,n)} \succ e_j) \wedge (o_a \xrightarrow{s_q^{(j,*)}} o_b) \wedge (o_b \xrightarrow{s_r^{(i,m)}} o_c) \wedge (n < m)$ $\Longrightarrow o_a \rightarrow o_c$
7. $(e_i \not\ggg e_j) \wedge (e_j \not\ggg e_i) \wedge (\forall p\, s_p^{(i,*)} \not\succ e_j) \wedge$ $(\forall q\, s_q^{(j,*)} \not\succ e_i) \wedge (o_a \xrightarrow{s_r^{(i,*)}} o_b) \wedge (o_b \xrightarrow{s_t^{(j,*)}} o_c)$ $\Longrightarrow o_a \rightarrow o_c$

TABLE II: TAINTMINI's universal data flow graph generation.

the *p-th* JavaScript code statement in the event group e_i. Each JavaScript statement may define data flow(s) between two data objects. We denote this as $o_a \xrightarrow{s_p^{(i,n)}} o_b$, which means that the p-th code statement in e_i has n as the order of statement execution in e_i and this statement creates the data flow from o_a to o_b. Note that all code statements within the same event group are executed synchronously, and thus their execution can be ordered. Inter-object data flows within a code statement occur in case of value assignments, function calls/returns, and read/write operations on WXML tags.

We also define notations for execution orders of event groups. The mini-program framework defines the execution order between certain page-handling callbacks (i.e., event groups) by design, such as: onLoad→onShow→onReady [25]. We denote such framework-defined execution orders between two event groups as $e_i \ggg e_j$. Meanwhile, the execution of some event groups are determined by the developer's hand-crafted mini-program code. For example, according to Figure 2's S-submit.js logic, the onShow callback handler is to be followed by the navigateToMiniProgram API call's success callback handler, because their calls are implemented as a nested structure. We denote such application-specific code-enforced execution order of event groups as $s_p^{(i,n)} \succ e_j$, meaning that the p-th statement in e_i calls e_j. This implies that e_j asynchronously executes in parallel with e_j's statements whose order of execution within e_j is greater than n.

Based on these notations, TAINTMINI tracks the data propagation based on 7 rules described in Table II. While the logical truth of each rule is straight-forward and self-explanatory, we also provide verbal description of each rule for clarity.

- **Rule 1** describes the transitivity of the framework-enforced execution order between event groups: if e_i executes before e_j and e_j executes before e_k, then e_i executes before e_k.
- **Rule 2** describes the transitivity of the application-code-enforced caller-callee relationship between event groups: if $s_p^{(i,n)}$ calls e_j and $s_q^{(j,m)}$ calls e_k, then it's also true that $s_p^{(i,n)}$ calls e_k.
- **Rule 3** describes the transitive inter-object data flows within the same event group: if an event group's statement creates the data flow $o_a \rightarrow o_b$ and a post-ordered statement creates the data flow $o_b \rightarrow o_c$, then $o_a \rightarrow o_c$. An example of this is Figure 2's step ❺-I where phone_old→phone_updated within the addCountryCode function. Note that Rule 3 is an in-event-group propagation rule, whereas Rules 4∼7 described next are cross-event-group propagation rules.
- **Rule 4** describes the transitive inter-object data flows across two event groups whose execution order is enforced by the application framework by design: if e_i is guaranteed to complete before e_j starts and e_i creates the data flow $o_a \rightarrow o_b$ and e_j creates the data flow $o_b \rightarrow o_c$, then $o_a \rightarrow o_c$. An example of this is Figure 2's step ❸ where onLaunch's e.globalData.phone propagates to onShow's data.
- **Rule 5** describes the transitive inter-object data flows across two event groups that have a caller-callee relationship enforced by application code: if the caller e_i's statement $s_p^{(i,n)}$ calls the callee e_j and the caller e_i creates the data flow $o_a \rightarrow o_b$ and the callee e_j creates the data flow $o_b \rightarrow o_c$, then $o_a \rightarrow o_c$. An example of this is Figure 2's step ❷ where onSubmit's this.data.phone flows to wx.nagivateTo's phone.
- **Rule 6** describes the transitive inter-object data flows similar to Rule 5 but in a reversed direction: if the caller e_i's statement $s_p^{(i,n)}$ calls the callee e_j and the callee e_j creates the data flow $o_a \rightarrow o_b$ and the caller e_i creates the data flow $o_b \rightarrow o_c$ after calling e_j, then $o_a \rightarrow o_c$. Rule 6 is true, because the two flows, $o_a \rightarrow o_b$ and $o_b \rightarrow o_c$, occur *after* e_j is started by $s_p^{(i,n)}$, which implies that these two flows can occur asynchronously in either order. Note that Table II's rules optimistically detects all data flows that can possibly occur when the application executes.
- **Rule 7** describes that if neither the application framework nor the application code enforces the execution order between e_i and e_j (i.e., e_i and e_j can be executed in any order in real time) and one of these event groups define the data flow $o_a \rightarrow o_b$ and the other event group creates the data flow $o_b \rightarrow o_c$, then $o_a \rightarrow a_c$ can possibly occur in real time.

C. Data-Flow Resolution

The goal of TAINTMINI is to track the sensitive data flow in mini-programs, and ultimately identify data leaks. In our running example, we can see that Figure 2 contains a data leakage flow, which is comprised of two sub-flows. The first part is generated by the Sender (S) mini-program and the second part is generated by the Receiver (R) mini-program. The S's data flow's source is bindgetphonenumber in S-fillout.wxml, and its sink is navigateToMiniProgram in S-submit.js, which in turn transfers the data to R which does not have the permission to access

the user's phone number. As the S explicitly specifies appId of R , TAINTMINI can further trace the data flow to inspect how the transferred phone number is used in R. The R's data flow's source is onLaunch, and its sink is the request API (which transfers the data to a remote server). As the final sink of the data flow (request) belongs to R that does not have the permission for the initial source of the data flow (i.e., user's phone number), TAINTMINI concludes that this is a data leakage.

V. EVALUATION

A. Experiment Setup

Implementation. We have implemented a prototype of TAINTMINI on top of open source DoubleX [23]. The prototype consists of two modules: (i) WXJS analyzer, which produces basic UDFG; and (ii) WXML analyzer, which binds flows to UDFG for flow propagation analysis. In support of open science, we have made the source code of TAINTMINI available at https://github.com/OSUSecLab/TaintMini.

Dataset. We used MiniCrawler [3]—we developed earlier and have made it open source—to download mini-programs from the app market of WECHAT. We have downloaded 3.3 million mini-programs. However, numerous mini-programs were developed from the same templates [3], resulting in almost the same code. We thus first conducted a similarity analysis to eliminate those that were too alike, and then eventually we obtained 238,866 distinct mini-programs, which consumed a total of 318.15 GiB of disk space. On average, each mini-program contains 25.83 pages, and the size of JavaScript file is 144.83 KiB.

Configuring the Taint Sources and Sinks. TAINTMINI focuses on detecting the flow of sensitive data in mini-programs. Particularly, the flows of interest are the ones that may leak sensitive data to the network or the another mini-program. Therefore, we configure the taint source APIs to be the ones that generate sensitive information (e.g., `wx.getLocation`), and the taint sinks are networking APIs (e.g., `wx.request`) or the cross-mini-program API (i.e., `navigateToMiniprogram`). Table III reports the number of mini-programs in our dataset that use the corresponding taint source or taint sink APIs.

Running Environment. We performed all our experiments on a server with two 16-core Intel Xeon 4314 CPUs at 2.40 GHz and 256 GiB of RAM, running Debian Bullseye.

B. Evaluation Results

RQ1. Does TAINTMINI have any False Positive (FP) and False Negative (FN)?

TAINTMINI detects the case where a mini-program sends out the sensitive data through the cross-mini-program channel or network channel. As such, a FP is the case where we mistakenly identify a mini-program that does not sends out any sensitive data, and a FN is the case where we fail to identify a mini-program that sends out the sensitive data. To verify whether our results are valid, we have sampled 100 mini-programs in each set, unpacked them, and inspected their code manually. Our manual analysis shows that there is zero FPs but 5 FNs (5.00%).

Category	APIs (Example)	# of Mini-apps
Taint Source APIs		
Storage	`wx.getStorageSync`	198,846
	`wx.getStorage`	6,263
Profile	`wx.getUserInfo`	144,749
	`wx.getUserProfile`	12,142
Location	`wx.getLocation`	146,163
	`wx.onLocationChange`	1,272
	`wx.startLocationUpdate`	958
	`wx.startLocationUpdateBac*`	330
Device	`wx.getNetworkType`	34,614
	`wx.createCameraContext`	6,295
	`wx.getBLEDeviceCharacteri*`	5,366
	`wx.getConnectedWifi`	1,883
Media	`wx.chooseVideo`	27,298
	`wx.chooseMedia`	8,207
	`wx.getFileInfo`	7,246
	`wx.startRecord`	2,522
Address	`wx.chooseAddress`	24,249
Open-API	`wx.getWeRunData`	3,504
	`wx.chooseInvoiceTitle`	1,598
	`wx.chooseInvoice`	109
	`wx.chooseContact`	62
Taint Sink APIs		
Request	`wx.request`	203,981
Upload	`wx.upload`	99,697
Navigate	`wx.navigateToMiniProgram`	42,352
WebSockets	`wx.sendSocketMessage`	1,428
UDP	`wx.createUDPSocket`	8

TABLE III: The statistics of the taint sources and sinks used in the tested mini-apps.

We further inspected these samples to understand the reason why there are FNs. We find that FNs occur when the mini-programs use non-standard APIs to collect the user's sensitive information. Specifically, we have observed that some mini-programs do not use the official API `getPhoneNumber` to collect the user's phone number, and instead they used input boxes to require the user to explicitly enter their phone numbers. This case is not considered under our current detection policy. While it is true that we can taint all the input boxes in order to capture this case, we believe this will introduce false positives, which is not desirable.

RQ2. How long does TAINTMINI take to analyze a mini-program on average?

We use TAINTMINI to analyze the collected 238,866 mini-programs. We record the execution time of each mini-program. Based on our results, the average cost of time to analyze one mini-program is 3.73 seconds. We believe the performance of our tool is acceptable given that tainting is a heavy program analysis technology by nature.

RQ3. What are those mini-programs that contain flow-sensitive data?

Among the tested 238,866 mini-programs, TAINTMINI has identified 27,184 (11.38%) mini-programs that contain sensitive data flows. The distribution of these detected mini-programs is

Category	Unrated Mini-apps			Rated Mini-apps								
				0.0 - 3.0			3.0 - 4.0			4.0 - 5.0		
	L	T	%	L	T	%	L	T	%	L	T	%
Business	1,290	12,547	10.3	2	9	22.2	10	62	16.1	73	645	11.3
E-Learning	131	1,625	8.1	4	9	44.4	8	56	14.3	20	185	10.8
Education	3,531	31,057	11.4	17	105	16.2	79	590	13.4	497	3,661	13.6
Entertainment	338	3,661	9.2	14	74	18.9	29	289	10.0	50	671	7.5
Finance	58	951	6.1	0	4	0.0	4	35	11.4	40	313	12.8
Food	633	4,775	13.3	0	3	0.0	4	31	12.9	75	453	16.6
Games	288	3,438	8.4	25	204	12.3	45	422	10.7	25	265	9.4
Government	793	6,525	12.2	0	10	0.0	9	76	11.8	64	542	11.8
Health	543	5,320	10.2	1	6	16.7	8	57	14.0	81	598	13.5
Job	445	3,375	13.2	7	21	33.3	24	119	20.2	56	431	13.0
Lifestyle	3,358	27,670	12.1	12	63	19.0	99	520	19.0	443	2,916	15.2
Photo	99	1,444	6.9	5	30	16.7	15	109	13.8	14	209	6.7
Shopping	4,082	36,696	11.1	6	38	15.8	44	397	11.1	484	3,895	12.4
Social	509	4,004	12.7	2	19	10.5	31	193	16.1	93	769	12.1
Sports	375	2,531	14.8	1	3	33.3	8	50	16.0	82	489	16.8
Tool	6,159	58,636	10.5	52	380	13.7	128	1,176	10.9	684	5,623	12.2
Traffic	629	4,634	13.6	11	34	32.4	26	177	14.7	144	911	15.8
Travelling	236	1,755	13.4	0	1	0.0	4	20	20.0	21	195	10.8
Ungrouped	7	58	12.1	0	1	0.0	0	0	-	0	0	-
Total	23,504	210,702	11.2	159	1,014	15.7	575	4,379	13.1	2,946	22,771	12.9

TABLE IV: The distribution of the detected mini-programs among the tested ones w.r.t. their different categories and ratings. "L" represents "Leakage" (i.e., the number of mini-programs that contain sensitive flows), and "T" represents "Total" (i.e., the total number of mini-programs)

Category	APIs (Example)	Total	Network Channel		Cross-App Channel	
			L	%	L	%
Storage	`wx.getStorageSync`	205,109	20,719	10.10	420	0.20
Profile	`wx.getUserInfo`	156,891	7,552	4.81	20	0.01
Location	`wx.getLocation`	145,205	3,737	2.57	4	0.00
Address	`wx.chooseAddress`	24,249	627	2.59	0	0.00
Open-API	`wx.getWeRunData`	5,273	295	5.59	0	0.00
Device	`wx.getNetworkType`	85,358	112	0.13	11	0.01
Media	`wx.chooseVideo`	89,351	39	0.04	0	0.00

TABLE V: The distribution of detected mini-programs w.r.t. their taint sources and taint sinks.

presented in Table IV. The rating shown in the table is designed on a five-point scale, where zero is the lowest and five is the highest. For each column representing the range of the scores, e.g., (0, 3], (3, 4], (4, 5], the left boundary does not include the endpoint (it is an open interval), but the right boundary does. We can see from Table IV that the majority of them is unrated, and the top 3 categories are tool, shopping, and education. Note that WeChat introduces a mechanism to allow users to rate a mini-program, and this rating will be available only when a certain number of users have rated it [26]. Therefore, these mini-programs are likely not extremely popular (note that unlike Google Play, there is no number of download metadata available in WeChat app market; we can only use the app rating to estimate its popularity).

Category	Address	Location	Open-API	Storage	Profile
Business	14 (0.9)	158 (9.9)	10 (0.6)	1,055 (66.4)	352 (22.2)
E-Learning	1 (0.5)	8 (4.2)	1 (0.5)	121 (63.7)	59 (31.1)
Education	66 (1.4)	370 (7.7)	26 (0.5)	3,149 (65.3)	1,213 (25.1)
Entertainment	8 (1.6)	44 (8.6)	3 (0.6)	310 (60.9)	144 (28.3)
Finance	1 (0.8)	20 (16.5)	3 (2.5)	74 (61.2)	23 (19.0)
Food	38 (4.3)	144 (16.4)	5 (0.6)	490 (55.8)	201 (22.9)
Games	3 (0.7)	30 (6.9)	4 (0.9)	292 (66.8)	108 (24.7)
Government	3 (0.3)	100 (10.0)	16 (1.6)	663 (66.6)	214 (21.5)
Health	15 (2.0)	66 (8.7)	15 (2.0)	480 (62.9)	187 (24.5)
Job	3 (0.5)	86 (13.9)	4 (0.6)	385 (62.3)	140 (22.7)
Lifestyle	94 (2.0)	734 (15.3)	20 (0.4)	2,903 (60.5)	1,044 (21.8)
Photo	3 (2.0)	9 (6.0)	0 (0.0)	97 (64.7)	41 (27.3)
Shopping	280 (4.9)	618 (10.8)	27 (0.5)	3,394 (59.2)	1,415 (24.7)
Social	4 (0.5)	78 (10.0)	11 (1.4)	479 (61.6)	205 (26.4)
Sports	10 (1.7)	63 (10.6)	91 (15.2)	331 (55.4)	102 (17.1)
Tool	69 (0.9)	956 (11.8)	52 (0.6)	5,245 (64.8)	1,775 (21.9)
Traffic	14 (1.4)	194 (19.7)	3 (0.3)	610 (62.0)	163 (16.6)
Travelling	1 (0.3)	52 (16.2)	4 (1.2)	187 (58.4)	76 (23.8)
Ungrouped	0 (0.0)	2 (28.6)	0 (0.0)	4 (57.1)	1 (14.3)

TABLE VI: The distribution of detected mini-programs w.r.t their accessed data and categories.

RQ4. What is the data that may be leaked through those mini-programs that contain sensitive data flow?

We further inspect how the data is accessed by the mini-programs, and how the data can be potentially leaked. Knowing specific categories of mini-programs collecting specific kind of data can benefit both users and super app developers. For the users, if they know that, they may get alerted when their data are improperly collected by the mini-programs (e.g., a game mini-program collects the user's location data). Super app developers can also use the information to guide their malware detection (i.e., their vetting). We can see from Table V that the mini-programs send out location data, user profile, werun (fitness) data, and so forth. Among them, most of the mini-programs send out sensitive information through network channel (not the cross-mini-program channel). Particularly, we also notice that most of the mini-programs send out stored information and location data. Next, we seek to understand the association between the specific type of leaked sensitive information (particularly the top ones) and the specific category of mini-programs. Table VI shows this result. To provide better illustration, we associate the accessed data with the corresponding mini-programs category, and plot a heatmap. The greater the number, the darker the color of that cell. For example, shopping mini-programs may access the user's profile (to know who the user is), user's location and address (to know where the user is for order delivery), and locally stored information (e.g., to collect the user's searching history for recommendation). Note that there are only 115 mini-programs that access the "device" and 38 mini-programs that access "media". We have eliminated those two columns for brevity in Table VI.

VI. Application

TaintMini can be used in many applications such as facilitating programmers to identify bugs or detecting privacy leakages. However, privacy leakage is typically application specific and depends on the context (e.g., a map mini-program will send the location data to the server to fetch the corresponding maps, and an analyst often needs to be involved to determine the

context). Interestingly, while in general it is hard to be automated, we notice that TAINTMINI can be used for automatic detection of privacy leakage in the case of collusion attacks. In particular, as specified in WECHAT's policy [27]: "*The user data collected in a specific Weixin Mini Program can only be used in that specific Weixin Mini Program, and shall not be used outside the specific Weixin Mini Program or for any other purpose, even if you have registered more than two Weixin Mini Programs at the same time.*" Therefore, any mini-program violates this policy will be banned and considered malware by WECHAT, and such programs can be automatically detected by TAINTMINI by inspecting whether any privacy sensitive data flows to cross mini-program request APIs.

As shown in Table V (highlighted in red), we have identified 455 mini-programs that leak sensitive information through the cross-mini-program channel. To understand concretely the impacts of the collusion attacks, we have performed a few case studies by inspecting the identified colluding mini-programs. In particular, we first check the taint sources of the senders to understand the specific sensitive information sent through the cross-mini-program channel, and then unpack the code to see why and how the sensitive data gets leaked. If the sender happens to have a receiver that falls into our dataset, we further inspect the receiver's code to see how the receiver consumes the data correspondingly. In the following, we present three such case studies.

Case Study-I: Leaking User Info. We noticed that many of the sender mini-programs transfer `userinfo` to other mini-programs. The `userinfo` is an object defined by WECHAT, which contains a set of privacy-sensitive information collected from users (e.g., the username, gender, the language the user speaks, and the home address). For example, we notice a sender mini-program (named "Pufferfish Shopping") directly transferred `userinfo` to a receiver mini-program. We further launched the sender mini-program on our mobile phone and noticed that there is no warning message that alerts the user about the possible leakage of information, which is clearly a violation of the user's intention.

Case Study-II: Leaking Phone Number. Interestingly, we noticed that some sender mini-programs transfer the user's phone number to another mini-program. However, the phone numbers are encrypted by WECHAT, and in order to get the decrypted phone number, the mini-programs may require its remote backend to fetch a key from the WECHAT server, decrypt the cipher remotely, and then send the phone number back. This is a clear violation of the privacy protection enforced by WECHAT, as WECHAT specifies in whatever cases the phone number should not be leaked to a mini-program's front-end [28]. Figure 5 shows an example, where the sender mini-program named "Impression Star" (a parking mini-program) obtains the encrypted information (line 6), sends it to the remote server for decryption (line 9∼10 and 13∼14), and finally sends the decrypted data to a second mini-program. During this process, the user is not informed that her phone number is shared with the other mini-programs.

Case Study-III: Leaking Location Data. There are also mini-programs that transfer the user's real-time location to other mini-programs. Being the receiver, some of them do not require

```
1  getPhoneNumber: function(e) {
2      ...
3      n.globalData.btnCanClick = !0, wx.hideLoading();
       ↪ else {
4      var o = {
5      iv: e.detail.iv,
6      encryptedData: e.detail.encryptedData,
7      type: n.globalData.appType
8      };
9      var l = n.getUrl("info");
10     t.default.post(l, o).then(function(e) {
11     if (200 == e.data.status) {
12     wx.hideLoading();
13     var t = e.data.data.phoneNumber;
14     this.data.phone_number = e.data.data.phoneNumber,
       ↪ n.globalData.userInfo = {
15      phone_number: t }, console.log("Obtained Phone
        ↪ Number", n.globalData.userInfo.phone_number),
        ↪ a.loginProcess(t);
16      } else n.wx_toast(e.data.message);
17      }); } },
18 wx.navigateToMiniProgram({
19     appId: e.linkAppid,
20     path: t,
21     extraData: {
22     mallId: this.data.mallId || "",
23     mallInfo: this.data.mallInfo || "",
24     memberId: this.data.memberId || "",
25     phoneNumber: this.data.phone_number || "", token:
       ↪ this.data.token || "" })
```

Fig. 4: Code snippet of a sender mini-program Impression Star

```
1   var a = e.referrerInfo.extraData.location, n =
    ↪ e.referrerInfo.extraData.imp_data, o =
    ↪ e.referrerInfo.extraData.src_path;
2     void 0 != a && void 0 != n && null != a && null != n
3     "" == this.data.location ? (wx.showLoading({
4     title: "Loading ..."
5      }), this.locationApp(a, n, o)) :
       ↪ this.data.location = "");
6      } catch (e) {}
7       t.eventReady(function() {
8       t.beginTime();
9       });
10      ...
11     wx.request({
12         url: "https://********.***.***",
13     data: location
14  });
```

Fig. 5: Code snippet of a receiver mini-program Good Games

the location permission at all, and we also do not know why those mini-programs require the user's location to run. For example, there is a mini-program named "Good Games" (which is a game), and it does not have `scope.location` to access the user's location, but fetches the location data from other mini-programs. Being a game by nature, we do not see the intention of collecting the user's location data. Further, it also sends out the collected location information, and we infer that this mini-program might have malicious intention that stealthily collects user's locations for other purposes.

VII. DISCUSSION

Limitation and Future Works. Although TAINTMINI is the first static taint analysis framework for mini-programs, it is certainly not perfect and is subject to a few limitations. For instance, we did not consider implicit data flows [29], where a tainted variable appears in the predicate (true/false). Although it is resolvable by assuming that the taints propagate to both branches, in practice it may introduce a taint explosion. Second, our detection can also have false negatives. To address this, one possible direction is to

resort to dynamic analysis. Finally, while we only analyzed the WeChat mini-programs, an immediate future work is to extend TaintMini to analyze the mini-programs in other platforms. We anticipate the extension will be minimal since nearly all the mini-programs are programmed with JavaScript, and have similar architectures (e.g., they all have the data flows between JavaScript files, JavaScript to the web page files). For example, Baidu has SWAN files, which have a similar syntax as WXML. Similarly, Tiktok has TTML and Alipay has AXML.

Ethics and Responsible Disclosure. During our study, we carefully addressed ethical concerns. For example, we never launched attacks against any other victims and conducted the experiment only in a controlled environment. When using our MiniCrawler [3] to download the mini-programs, we never exceed WeChat's rate limit (we limited our number of requests just six per minute to the WeChat server). Also, we have identified 455 malware based on the security and privacy policy of WeChat [27]. We followed the community practice by reporting these malicious mini-programs to *Tencent*, who has acknowledged and confirmed our findings.

Threat to the Validity. There could be some human errors in our study, particularly from our manual confirmation of the FPs and FNs. Recall to verify the FPs and FNs, we randomly selected 100 mini-programs from 18 categories, and therefore, this inspection involved manual efforts. Since there is no ground truth to verify our results, we relied on three domain experts (also the three co-authors of this paper) who are familiar with JavaScript and mini-program programming to reverse engineer those mini-programs. During the inspection, when disagreements occur, we would discuss the case to reach a consensus to reduce the errors caused by manual efforts. Since all manual analysis was performed by reverse engineering and inspecting the code, the results may still be biased. For example, we may make mistakes, as JavaScript heavily relies on callbacks, which can make the code difficult to follow, especially in complex mini-programs with multiple levels of nested functions. Meanwhile, JavaScript is an asynchronous language, which means that multiple things can happen at the same time. This can make it difficult to keep track of what is happening in the code.

VIII. Related Works

Taint Analysis. Taint analysis [17] is a widely used program analysis technique tracking flow of sensitive data, and it has numerous applications, such as exploit detection [18, 30], privacy leakage detection [10, 19], and cryptographic key misuse detection [14, 31]. In the past decades, numerous taint analysis systems have been proposed and applied to analyze different languages (e.g., Java [32, 32–34] and C [35, 36]) and frameworks (Android [10, 37–39], iOS [19] and Microservices [40]). However, our work is still quite different compared with the prior efforts. For example, none of the existing works can detect privacy leaks in mini-programs, and TaintMini is the first such a framework that can automatically detect cross-mini-program privacy leaks.

JavaScript Analysis. There is also a large body of efforts focusing on web and JavaScript program analysis. For instance, DOMinator [41] modifies Firefox to support dynamic taint analysis (for detecting XSS). Kang *et al.* propose PROBETHEPROTO [42], to measure the client-side prototype pollution of JavaScripts among websites. There are a few tools (e.g., [43, 44]) that also modified the browsers and used dynamic taint analysis to detect the possible vulnerabilities existing in web applications. Other than those dynamic tools, there are also static frameworks that analyze JavaScript for bug hunting and errors detection. For example, Madsen *et al.* [45] presented the a bug analyzer based on an event-based call graph, and Feldthaus *et al.* [46] used Approximate Call Graphs for JavaScript IDE Services. VEX [47] and DoubleX [23] detected browser extension vulnerabilities. JAW [48] used a hybrid structure to detect client-side CSRF attacks. Iqbal *et al.* [49] detect Web tracking by modelling the relationship of client side objects. Compared with those tools, TaintMini not only considers the mini-program specific data flow with JavaScript but also considers the data flows between WXML files.

Mini-Program Security. The mini-program paradigm has been adopted by multiple super apps today, and its security is under active scrutiny. For example, Lu *et al.* [2] have studied the resource management in mini-programs, and identified a few flaws that enable the attackers to steal sensitive user information. Zhang *et al.* [3] developed MiniCrawler and studied the security practice of the mini-programs (e.g., whether the mini-programs have enabled obfuscation and what kind of security related APIs the mini-programs may invoke). Zhang *et al.* [50] discovered identity confusion vulnerabilities against mini-programs, allowing the attacker to invoke high privileged capabilities (e.g., stealing the user's information, and downloading and installing the malicious apps). Most recently, Yang *et al.* also recently investigated the vulnerability of cross-mini-program redirection among mini-programs in WeChat and Baidu [51]. Unlike these works, we developed TaintMini, a general taint analysis framework, and applied it to detect collusion attacks among mini-programs.

IX. Conclusion

We have presented TaintMini, a static taint analysis framework that can detect the flow of sensitive data in mini-programs. The framework builds a universal data flow graph by considering data flows across multiple languages (i.e., JavaScript and WXML), pages, and mini-programs. Our evaluation of 238,866 mini-programs on the WeChat platform showed that 11.38% of them contain privacy sensitive data flows, with 455 of them capable of leaking privacy data through collusion. Our findings have been responsibly disclosed to the vendor, who has acknowledged and confirmed them. We believe that TaintMini provides a valuable tool for detecting and mitigating privacy risks in mini-programs.

Acknowledgment

We thank the anonymous reviewers for their insightful feedbacks. This research was supported in part by DARPA award N6600120C4020. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of DARPA.

REFERENCES

[1] W3C, "Miniapp standardization white paper," https://w3c.github.io/miniapp/white-paper/, 2020.

[2] H. Lu, L. Xing, Y. Xiao, Y. Zhang, X. Liao, X. Wang, and X. Wang, "Demystifying resource management risks in emerging mobile app-in-app ecosystems," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 569–585.

[3] Y. Zhang, B. Turkistani, A. Y. Yang, C. Zuo, and Z. Lin, "A measurement study of wechat mini-apps," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 2, pp. 1–25, 2021.

[4] H. Cao, Z. Chen, F. Xu, T. Wang, Y. Xu, L. Zhang, and Y. Li, "When your friends become sellers: an empirical study of social commerce site beidian," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 14, 2020, pp. 83–94.

[5] H. Cao, Z. Chen, M. Cheng, S. Zhao, T. Wang, and Y. Li, "You recommend, I buy: How and why people engage in instant messaging based social commerce," *Proc. ACM Hum. Comput. Interact.*, vol. 5, no. CSCW1, pp. 1–25, 2021. [Online]. Available: https://doi.org/10.1145/3449141

[6] T. Graziani, "What are wechat mini-programs? a simple introduction," https://walkthechat.com/wechat-mini-programs-simple-introduction/, 2021.

[7] "Biggest app stores in the world 2022," https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, (Accessed on 04/30/2022).

[8] T. inc., "Wechat payment official document," https://pay.weixin.qq.com/index.php/public/wechatpay_en, 2020.

[9] "Authorization (miniapp)," https://developers.weixin.qq.com/miniprogram/en/dev/framework/open-ability/authorize.html, (Accessed on 04/30/2022).

[10] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1–29, 2014.

[11] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *Acm Sigplan Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[12] Tencent, "Base library update logs," https://developers.weixin.qq.com/miniprogram/dev/framework/release/.

[13] "Mining node.js vulnerabilities via object dependence graph and query," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/li-song

[14] L. Zhang, J. Chen, W. Diao, S. Guo, J. Weng, and K. Zhang, "{CryptoREX}: Large-scale analysis of cryptographic misuse in {IoT} devices," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019, pp. 151–164.

[15] H. LU, Q. ZHAO, Y. CHEN, X. LIAO, and Z. LIN, "Detecting and measuring aggressive location harvesting in mobile apps via data-flow path embedding," in *Proceedings of the 2023 ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, 2023.

[16] "The total size of all subpackages of a Mini Program cannot exceed 12 MB," https://developers.weixin.qq.com/miniprogram/en/dev/framework/subpackages.html, 06 2020, (Accessed on 04/30/2022).

[17] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)," in *2010 IEEE symposium on Security and privacy*. IEEE, 2010, pp. 317–331.

[18] J. Newsome and D. X. Song, "Dynamic taint analysis for automatic detection, analysis, and signaturegeneration of exploits on commodity software." in *NDSS*, vol. 5. Citeseer, 2005, pp. 3–4.

[19] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "Pios: Detecting privacy leaks in ios applications." in *NDSS*, 2011, pp. 177–183.

[20] A. Kogtenkov, B. Meyer, and S. Velder, "Alias calculus, change calculus and frame inference," *Science of Computer Programming*, vol. 97, pp. 163–172, 2015, special Issue on New Ideas and Emerging Results in Understanding Software. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642313002906

[21] A. Diwan, K. S. McKinley, and J. E. B. Moss, "Type-based alias analysis," ser. PLDI '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 106–117. [Online]. Available: https://doi.org/10.1145/277650.277670

[22] D. G. Clarke, J. M. Potter, and J. Noble, "Ownership types for flexible alias protection," in *Proceedings of the 13th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 48–64. [Online]. Available: https://doi.org/10.1145/286936.286947

[23] A. Fass, D. F. Somé, M. Backes, and B. Stock, "Doublex: Statically detecting vulnerable data flows in browser extensions at scale," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1789–1804.

[24] "The race to create the world's next super-app - bbc news," https://www.bbc.com/news/business-55929418, (Accessed on 08/28/2022).

[25] T. inc., "Wechat mini-program's official document (component)," https://developers.weixin.qq.com/miniprogram/en/dev/component, 2022.

[26] Allison, "Wechat mini-programs 2020: What your brand should know about this daily-life essential," https://daxueconsulting.com/wechat-mini-programs-2020-report/, 2020.

[27] T. inc., "15. rules for user privacy and data," https://developers.weixin.qq.com/miniprogram/en/product/#14-User-Privacy-and-Data-Specifications, 2022.

[28] ——, "Wechat mini-program's official document (data verification and encryption)," https://developers.weixin.qq.com/miniprogram/en/dev/framework/open-ability/signature.html, 2020.

[29] M. G. Kang, S. McCamant, P. Poosankam, and D. Song, "Dta++: dynamic taint analysis with targeted control-flow propagation." in *NDSS*, 2011.

[30] S. Chen, Z. Lin, and Y. Zhang, "SelectiveTaint: Efficient data flow tracking with static binary rewriting," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1665–1682. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/chen-sanchuan

[31] S. Rahaman and D. Yao, "Program analysis of cryptographic implementations for security," in *2017 IEEE Cybersecurity Development (SecDev)*. IEEE, 2017, pp. 61–68.

[32] O. Tripp, M. Pistoia, S. J. Fink, M. Sridharan, and O. Weisman, "Taj: effective taint analysis of web applications," *ACM Sigplan Notices*, vol. 44, no. 6, pp. 87–97, 2009.

[33] W. Huang, Y. Dong, and A. Milanova, "Type-based taint analysis for java web applications," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2014, pp. 140–154.

[34] V. Haldar, D. Chandra, and M. Franz, "Dynamic taint propagation for java," in *21st Annual Computer Security Applications Conference (ACSAC'05)*. IEEE, 2005, pp. 9–pp.

[35] J. Ming, D. Wu, G. Xiao, J. Wang, and P. Liu, "{TaintPipe}: Pipelined symbolic taint analysis," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 65–80.

[36] X. Fu and H. Cai, "Scaling application-level dynamic taint analysis to enterprise-scale distributed systems," in *ICSE*

'20: 42nd International Conference on Software Engineering, Companion Volume, Seoul, South Korea, 27 June - 19 July, 2020, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 270–271. [Online]. Available: https://doi.org/10.1145/3377812.3390910

[37] L. N. Q. Do, K. Ali, B. Livshits, E. Bodden, J. Smith, and E. R. Murphy-Hill, "Cheetah: just-in-time taint analysis for android apps," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*, S. Uchitel, A. Orso, and M. P. Robillard, Eds. IEEE Computer Society, 2017, pp. 39–42. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.20

[38] J. Zhang, C. Tian, and Z. Duan, "Fastdroid: efficient taint analysis for android applications," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 236–237. [Online]. Available: https://doi.org/10.1109/ICSE-Companion.2019.00092

[39] Z. Yang and M. Yang, "Leakminer: Detect information leakage on android with static taint analysis," in *2012 Third World Congress on Software Engineering*. IEEE, 2012, pp. 101–104.

[40] Z. Zhong, J. Liu, D. Wu, P. Di, Y. Sui, and A. X. Liu, "Field-based static taint analysis for industrial microservices," in *44th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2022, Pittsburgh, PA, USA, May 22-24, 2022*. IEEE, 2022, pp. 149–150. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP55303.2022.9794096

[41] R. Gera, "On the dominator colorings in bipartite graphs," in *Fourth International Conference on Information Technology (ITNG'07)*. IEEE, 2007, pp. 947–952.

[42] Z. Kang, S. Li, and Y. Cao, "Probe the proto: Measuring client-side prototype pollution vulnerabilities of one million real-world websites."

[43] S. Lekies, B. Stock, and M. Johns, "25 million flows later: large-scale detection of dom-based xss," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 1193–1204.

[44] W. Melicher, A. Das, M. Sharif, L. Bauer, and L. Jia, "Riding out domsday: Towards detecting and preventing dom cross-site scripting," in *2018 Network and Distributed System Security Symposium (NDSS)*, 2018.

[45] M. Madsen, F. Tip, and O. Lhoták, "Static analysis of event-driven node. js javascript applications," *ACM SIGPLAN Notices*, vol. 50, no. 10, pp. 505–519, 2015.

[46] A. Feldthaus, M. Schäfer, M. Sridharan, J. Dolby, and F. Tip, "Efficient construction of approximate call graphs for javascript ide services," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 752–761.

[47] S. Bandhakavi, S. T. King, P. Madhusudan, and M. Winslett, "{VEX}: Vetting browser extensions for security vulnerabilities," in *19th USENIX Security Symposium (USENIX Security 10)*, 2010.

[48] S. Khodayari and G. Pellegrino, "{JAW}: Studying client-side {CSRF} with hybrid property graphs and declarative traversals," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2525–2542.

[49] U. Iqbal, P. Snyder, S. Zhu, B. Livshits, Z. Qian, and Z. Shafiq, "Adgraph: A graph-based approach to ad and tracker blocking," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 763–776.

[50] L. Zhang, Z. Zhang, A. Liu, Y. Cao, X. Zhang, Y. Chen, Y. Zhang, G. Yang, and M. Yang, "Identity confusion in webview-based mobile app-in-app ecosystems," in *31st USENIX Security Symposium (USENIX Security'22)*, 2022.

[51] Y. Yang, Y. Zhang, and Z. Lin, "Cross miniapp request forgery: Root causes, attacks, and vulnerability detection," in *Proceedings of the 29th ACM Conference on Computer and Communications Security*, 2022.