



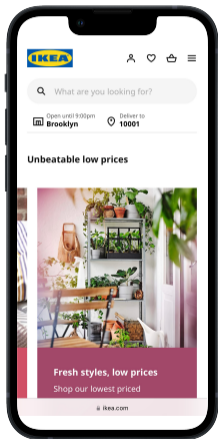
# TaintMini: Detecting Flow of Sensitive Data in Mini-Programs with Static Taint Analysis

Chao Wang, Ronny Ko, Yue Zhang  
Yuqing Yang, and Zhiqiang Lin

ICSE 2023

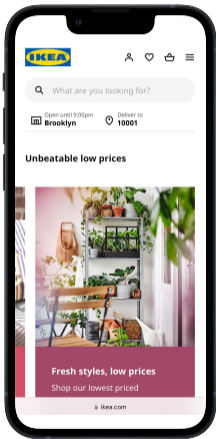


# The Evolution of Mobile App Development

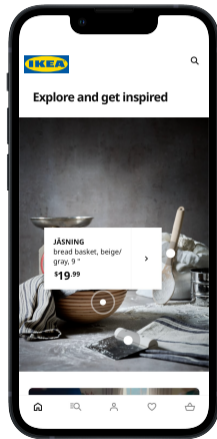


Web Apps

# The Evolution of Mobile App Development

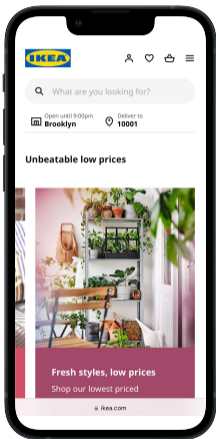


Web Apps

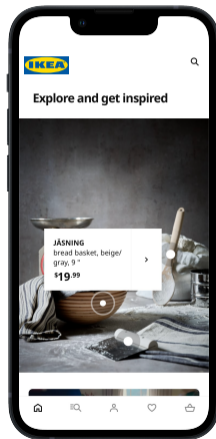


Native Apps

# The Evolution of Mobile App Development



Web Apps



Native Apps



Mini-Programs

# The Evolution of Mobile App Development



Apps	Web Apps	Native Apps	Mini-Programs
<b>End Users</b>			
Install-less?	✓	✗	✓
<b>Developers</b>			
Native APIs Access?	✗	✓	✓
Cross-platforms?	✓	✗	✓
Lightweight?	✗	✗	✓

# Our Focus: the Mini-Programs in WeChat Super App

There are many other super apps

- 1 Alipay
- 2 SnapChat
- 3 TikTok
- 4 ...



WeChat Mini-Program

# Our Focus: the Mini-Programs in WeChat Super App

There are many other super apps

- 1 Alipay
- 2 SnapChat
- 3 TikTok
- 4 ...

Why WeChat?

- ▶ Largest number of users (1.2 billion monthly active users)
- ▶ Pioneer of the Mini-Program paradigm
- ▶ Largest number of Mini-Programs (**millions**)

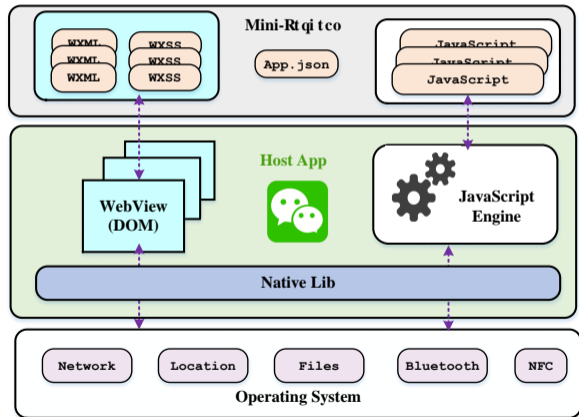


WeChat Mini-Program

# With Great Power Comes Great Responsibility

## The Great Power

- ▶ **Resources:** Network, Files, Bluetooth, NFC, etc.
- ▶ **Data:** Users' Location, Profiles, Phone numbers, etc.





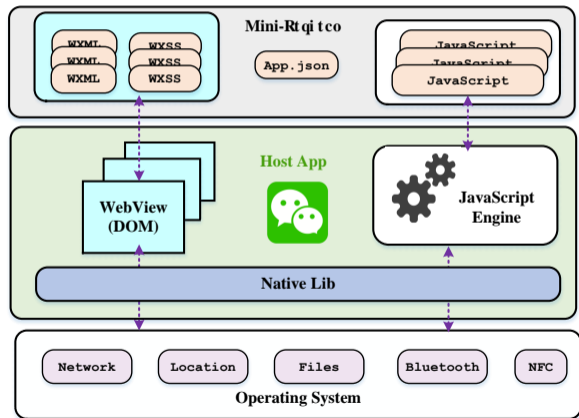
# With Great Power Comes Great Responsibility

## The Great Power

- ▶ **Resources:** Network, Files, Bluetooth, NFC, etc.
- ▶ **Data:** Users' Location, Profiles, Phone numbers, etc.

## The Great Responsibility

- ▶ Protect users' sensitive data
- ▶ Prevent data leakage



# When It Fails



## Data Leakage!

- ▶ Mini-Programs can now directly access **private and sensitive data** via APIs provided by the host app
- ▶ **Accidentally** leaked by careless programmers
- ▶ **Intentionally** leaked by malicious Mini-Programs

# When It Fails



## Data Leakage!

- ▶ Mini-Programs can now directly access **private and sensitive data** via APIs provided by the host app
- ▶ **Accidentally** leaked by careless programmers
- ▶ **Intentionally** leaked by malicious Mini-Programs

Then we must detect such data leakage in Mini-Programs?

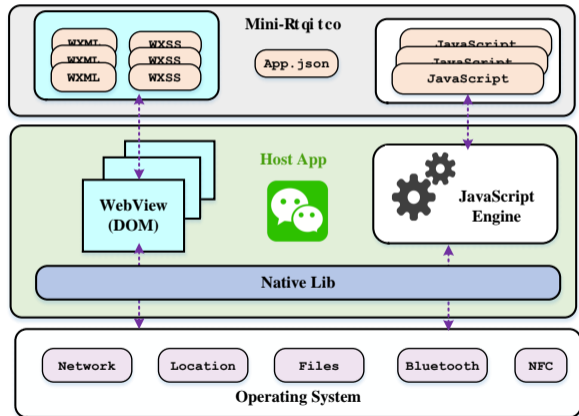
# How does a Mini-Program Look Like

## View - Rendering

- ▶ WXML
- ▶ WXSS

## Logic - Programming

- ▶ JavaScript



# How does a Mini-Program Look Like

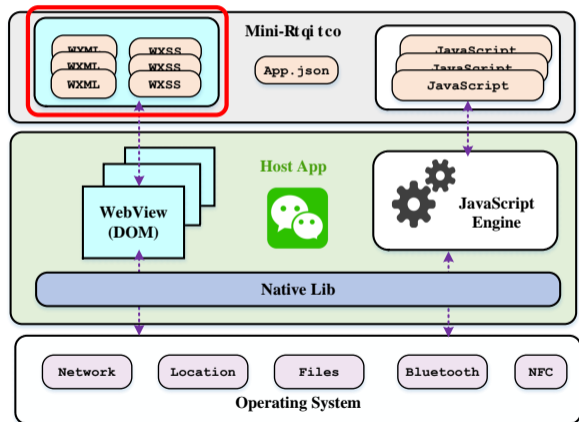
## View - Rendering

▶ WXML

▶ WXSS

## Logic - Programming

▶ JavaScript



# How does a Mini-Program Look Like

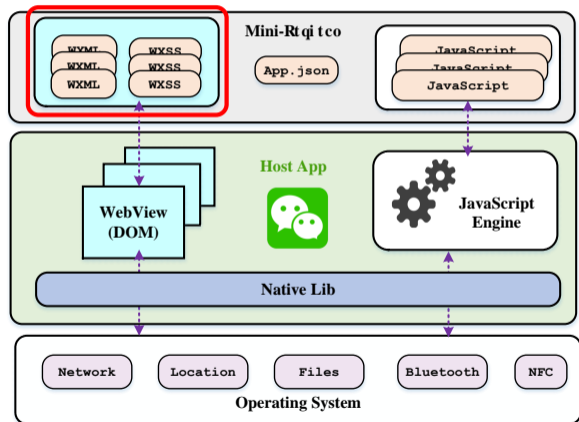
## View - Rendering

▶ WXML

▶ WXSS

## Logic - Programming

▶ JavaScript



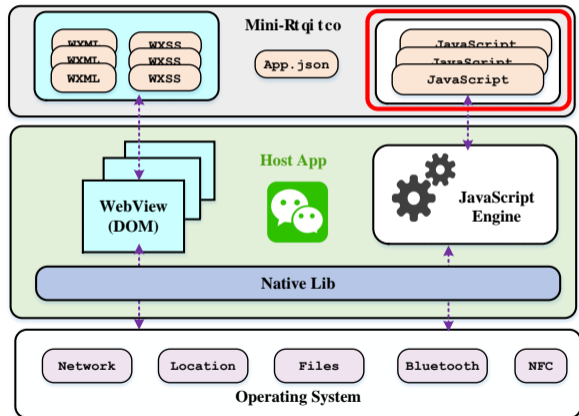
# How does a Mini-Program Look Like

## View - Rendering

- ▶ WXML
- ▶ WXSS

## Logic - Programming

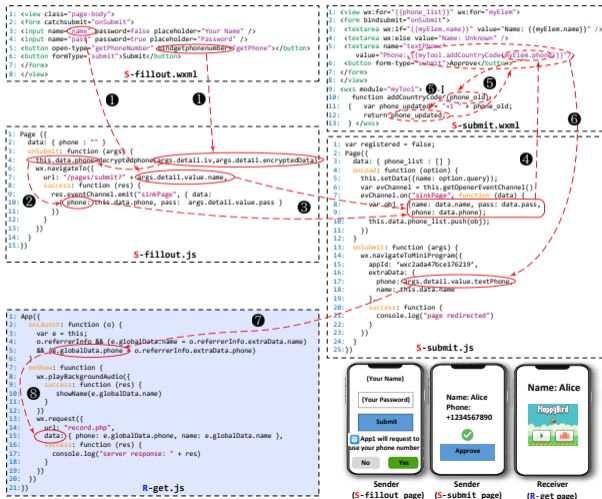
- ▶ JavaScript



# An Example to Illustrate Various Data Flows in Mini-Programs

## Various Data Flows

- ① JavaScript ↔ JavaScript
- ② JavaScript ↔ WXML
- ③ WXML ↔ WXML
- ④ EventHandler ↔ EventHandler
- ⑤ Mini-Programs ↔ Mini-Programs





# Why This is Challenging

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

# Our Solution

- 1 Track data flows between WXML tags and JavaScript

*Universal Data-Flow Graph (UDFG) Generation*

- 2 Track data flows between asynchronous JavaScript callback functions

*Data-Flow Propagation*

- 3 Track data flows between Mini-Program pages & Mini-Programs

*Data-Flow Resolution*

## Notations

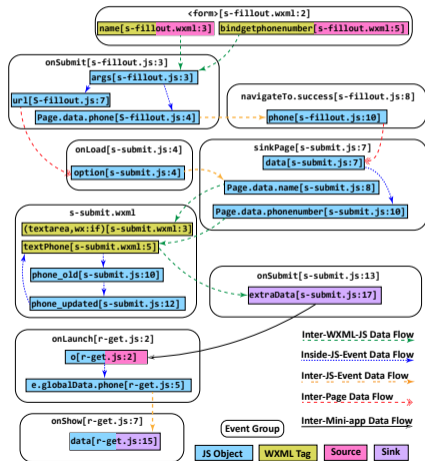
- $o_a \in \mathbb{O}$  : Data object  $o_a$  (in the data object universe  $\mathbb{O}$ )  
 $e_i \in \mathbb{E}$  : Event group  $e_i$  (in the event group universe  $\mathbb{E}$ )  
 $s_p^{(i,n)}$  : Event group  $e_i$ 's  $p$ -th code statement, where  $n$  is the order of statement execution within  $e_i$   
 $o_a \xrightarrow{s_p^{(i,n)}} o_b$  :  $s_p^{(i,n)}$  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.  $(e_i \ggg e_j) \wedge (e_j \ggg e_k) \implies e_i \ggg e_k$
2.  $(s_p^{(i,n)} \succ e_j) \wedge (s_q^{(j,m)} \succ e_k) \implies s_p^{(i,n)} \succ e_k$
3.  $(o_a \xrightarrow{s_p^{(i,n)}} o_b) \wedge (o_b \xrightarrow{s_q^{(j,m)}} o_c) \wedge (n < m) \implies o_a \rightarrow o_c$
4.  $(e_i \ggg e_j) \wedge (o_a \xrightarrow{s_p^{(i,*)}} o_b) \wedge (o_b \xrightarrow{s_q^{(j,*)}} o_c) \implies o_a \rightarrow o_c$
5.  $(s_p^{(i,n)} \succ e_j) \wedge (o_a \xrightarrow{s_r^{(i,*)}} o_b) \wedge (o_b \xrightarrow{s_q^{(j,*)}} o_c) \implies o_a \rightarrow o_c$
6.  $(s_p^{(i,n)} \succ e_j) \wedge (o_a \xrightarrow{s_q^{(j,*)}} o_b) \wedge (o_b \xrightarrow{s_r^{(i,m)}} o_c) \wedge (n < m) \implies o_a \rightarrow o_c$
7.  $(e_i \ggg e_j) \wedge (e_j \ggg e_i) \wedge (\forall p s_p^{(i,*)} \neq e_j) \wedge (\forall q s_q^{(j,*)} \neq e_i) \wedge (o_a \xrightarrow{s_p^{(i,*)}} o_b) \wedge (o_b \xrightarrow{s_q^{(j,*)}} o_c) \implies o_a \rightarrow o_c$

# Our Solution

- Track data flows between WXML tags and JavaScript  
*Universal Data-Flow Graph (UDFG) Generation*
- Track data flows between asynchronous JavaScript callback functions  
*Data-Flow Propagation*
- Track data flows between Mini-Program pages & Mini-Programs  
*Data-Flow Resolution*



# Dataset

- ▶ Downloaded by open source MiniCrawler [[ZTY<sup>+</sup>21](#)]
- ▶ **238,866** distinct Mini-Programs out of 3.3 million Mini-Programs
- ▶ **25.83** pages per Mini-Program on average
- ▶ **144.83 KiB** of JavaScript per Mini-Program on average

# Research Questions

## RQ1

Does TaintMini have any FPs and FNs?

## RQ2

How long does TaintMini take to analyze a Mini-Program on average?

## RQ3

What are those Mini-Programs that contain flow-sensitive data?

## RQ4

What is the data that may be leaked through those Mini-Programs that contain sensitive data flow?

# Research Questions

## RQ1: Does TaintMini have any FPs and FNs?

- ▶ Sampled **100** Mini-Programs in each set (FP and FN)
- ▶ Manually inspected by experts
- ▶ **0** FPs
- ▶ **5** FNs (5.00%): using non-standard APIs to get sensitive data

# Research Questions

## RQ1: Does TaintMini have any FPs and FNs?

- ▶ Sampled **100** Mini-Programs in each set (FP and FN)
- ▶ Manually inspected by experts
- ▶ **0** FPs
- ▶ **5** FNs (5.00%): using non-standard APIs to get sensitive data

## RQ2: How long does TaintMini take to analyze a mini-program on average?

- ▶ Recorded execution time of analysis for each Mini-Program
- ▶ **3.73** seconds on average to analyze one Mini-Program

# RQ3: What are those Mini-Programs that contain flow-sensitive data?

► **27,184 (11.38%)** Mini-Programs out of **238,866** contain sensitive data flows.

Category	Unrated Mini-apps			Rated Mini-apps								
				0.0 - 3.0			3.0 - 4.0			4.0 - 5.0		
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
<b>Education</b>	<b>3,531</b>	<b>31,057</b>	<b>11.4</b>	<b>17</b>	<b>105</b>	<b>16.2</b>	<b>79</b>	<b>590</b>	<b>13.4</b>	<b>497</b>	<b>3,661</b>	<b>13.6</b>
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
<b>Shopping</b>	<b>4,082</b>	<b>36,696</b>	<b>11.1</b>	<b>6</b>	<b>38</b>	<b>15.8</b>	<b>44</b>	<b>397</b>	<b>11.1</b>	<b>484</b>	<b>3,895</b>	<b>12.4</b>
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
<b>Tool</b>	<b>6,159</b>	<b>58,636</b>	<b>10.5</b>	<b>52</b>	<b>380</b>	<b>13.7</b>	<b>128</b>	<b>1,176</b>	<b>10.9</b>	<b>684</b>	<b>5,623</b>	<b>12.2</b>
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	-
<b>Total</b>	<b>23,504</b>	<b>210,702</b>	<b>11.2</b>	<b>159</b>	<b>1,014</b>	<b>15.7</b>	<b>575</b>	<b>4,379</b>	<b>13.1</b>	<b>2,946</b>	<b>22,771</b>	<b>12.9</b>



## RQ4: What is the data that may be leaked through those Mini-Programs?

- ▶ **Storage, profile, and location** are the top categories that may be leaked
- ▶ **Network** is the top channel that data may be leaked through

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

# Applications: Automatic Detection of Collusion Attacks

## Leaking phone number

- ▶ The sender Mini-Program collects users' phone numbers
- ▶ The sender Mini-Program transfers them to another mini-program

```
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,
15         ↪ n.globalData.userInfo = {
16           phone_number: t }, console.log("Obtained Phone Number",
17           ↪ n.globalData.userInfo.phone_number), a.loginProcess(t);
18         } else n.wx_toast(e.data.message);
19       }); } },
20 wx.navigateToMiniProgram({
21   appId: e.linkAppid,
22   path: t,
23   extraData: {
24     mallId: this.data.mallId || "",
25     mallInfo: this.data.mallInfo || "",
26     memberId: this.data.memberId || "",
27     phoneNumber: this.data.phone_number || "", token:
28     ↪ this.data.token || "" } })
```

# Applications: Automatic Detection of Collusion Attacks

## Leaking location data

- ▶ The receiver Mini-Program does not have permission to retrieve users' location
- ▶ The receiver Mini-Program receives location data from other Mini-Programs

```
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 });
```

# Future Works

- ① Implicit data flows resolution
- ② Dynamic analysis
- ③ Extend to other mobile super apps (e.g., Alipay, Baidu, Tiktok, etc.)

## Related Works

- ▶ **Taint Analysis.** Widely used program analysis technique for numerous applications, such as exploit detection [NS05, CLZ21], privacy leakage detection [EKKV11, EGH<sup>+</sup>14], and cryptographic key misuse detection [ZCD<sup>+</sup>19, RY17]. In the past decades, numerous taint analysis systems have been proposed and applied to analyze different languages (e.g., Java [TPF<sup>+</sup>09, HDM14, HCF05, TPF<sup>+</sup>09] and C [MWX<sup>+</sup>15, FC20]) and frameworks (Android [DAL<sup>+</sup>17, EGH<sup>+</sup>14, ZTD19, YY12], iOS [EKKV11] and Microservices [ZLW<sup>+</sup>22]).

## Related Works

- ▶ **Mini-Program Security.** Lu *et al.* [LXX<sup>+</sup>20] 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.* [ZTY<sup>+</sup>21] developed MiniCrawler and studied the security practice of the mini-programs. Zhang *et al.* [ZZL<sup>+</sup>22] discovered identity confusion vulnerabilities against mini-programs, allowing the attacker to invoke high privileged capabilities. Most recently, Yang *et al.* also recently investigated the vulnerability of cross-mini-program redirection among mini-programs in WeChat and Baidu [YZL22].

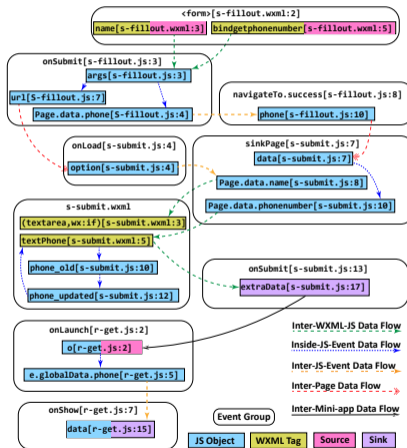
# Conclusion

## TaintMini

- ▶ First **open source**, static taint analysis framework for Mini-Programs

## Evaluated w/ 238K Mini-Programs

- ▶ **Empirical Results.** Identified 27,184 (11.38%) Mini-Programs contain sensitive data flows
- ▶ **Security Application.** Identified 455 colluding apps



Source code of TaintMini has been made available at: [github.com/OSUSecLab/TaintMini](https://github.com/OSUSecLab/TaintMini)

# References I

-  Sanchuan Chen, Zhiqiang Lin, and Yinqian Zhang, *SelectiveTaint: Efficient data flow tracking with static binary rewriting*, 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, August 2021, pp. 1665–1682.
-  Lisa Nguyen Quang Do, Karim Ali, Benjamin Livshits, Eric Bodden, Justin Smith, and Emerson R. Murphy-Hill, *Cheetah: just-in-time taint analysis for android apps*, Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume (Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard, eds.), IEEE Computer Society, 2017, pp. 39–42.
-  William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth, *Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones*, ACM Transactions on Computer Systems (TOCS) **32** (2014), no. 2, 1–29.
-  Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna, *Pios: Detecting privacy leaks in ios applications.*, NDSS, 2011, pp. 177–183.
-  Xiaoqin Fu and Haipeng Cai, *Scaling application-level dynamic taint analysis to enterprise-scale distributed systems*, ICSE '20: 42nd International Conference on Software Engineering, Companion Volume, Seoul, South Korea, 27 June - 19 July, 2020 (Gregg Rothermel and Doo-Hwan Bae, eds.), ACM, 2020, pp. 270–271.
-  Vivek Haldar, Deepak Chandra, and Michael Franz, *Dynamic taint propagation for java*, 21st Annual Computer Security Applications Conference (ACSAC'05), IEEE, 2005, pp. 9–pp.
-  Wei Huang, Yao Dong, and Ana Milanova, *Type-based taint analysis for java web applications*, International Conference on Fundamental Approaches to Software Engineering, Springer, 2014, pp. 140–154.



# References II

-  Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang Wang, *Demystifying resource management risks in emerging mobile app-in-app ecosystems*, Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 569–585.
-  Jiang Ming, Dinghao Wu, Gaoyao Xiao, Jun Wang, and Peng Liu, *{TaintPipe}: Pipelined symbolic taint analysis*, 24th USENIX Security Symposium (USENIX Security 15), 2015, pp. 65–80.
-  James Newsome and Dawn Xiaodong Song, *Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software.*, NDSS, vol. 5, Citeseer, 2005, pp. 3–4.
-  Sazzadur Rahaman and Danfeng Yao, *Program analysis of cryptographic implementations for security*, 2017 IEEE Cybersecurity Development (SecDev), IEEE, 2017, pp. 61–68.
-  Omer Tripp, Marco Pistoia, Stephen J Fink, Manu Sridharan, and Omri Weisman, *Taj: effective taint analysis of web applications*, ACM Sigplan Notices **44** (2009), no. 6, 87–97.
-  Zhemin Yang and Min Yang, *Leakminer: Detect information leakage on android with static taint analysis*, 2012 Third World Congress on Software Engineering, IEEE, 2012, pp. 101–104.
-  Yuqing Yang, Yue Zhang, and Zhiqiang Lin, *Cross miniapp request forgery: Root causes, attacks, and vulnerability detection*, Proceedings of the 29th ACM Conference on Computer and Communications Security, 2022.
-  Li Zhang, Jiongyi Chen, Wenrui Diao, Shanqing Guo, Jian Weng, and Kehuan Zhang, *{CryptoREX}: Large-scale analysis of cryptographic misuse in {IoT} devices*, 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019), 2019, pp. 151–164.

# References III

-  Zexin Zhong, Jiangchao Liu, Diyu Wu, Peng Di, Yulei Sui, and Alex X. Liu, *Field-based static taint analysis for industrial microservices*, 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.
-  Jie Zhang, Cong Tian, and Zhenhua Duan, *Fastdroid: efficient taint analysis for android applications*, Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019 (Joanne M. Atlee, Tevfik Bultan, and Jon Whittle, eds.), IEEE / ACM, 2019, pp. 236–237.
-  Yue Zhang, Bayan Turkistani, Allen Yuqing Yang, Chaoshun Zuo, and Zhiqiang Lin, *A measurement study of wechat mini-apps*, Proceedings of the ACM on Measurement and Analysis of Computing Systems 5 (2021), no. 2, 1–25.
-  Lei Zhang, Zhibo Zhang, Ancong Liu, Yinzhi Cao, Xiaohan Zhang, Yanjun Chen, Yuan Zhang, Guangliang Yang, and Min Yang, *Identity confusion in webview-based mobile app-in-app ecosystems*, 31st USENIX Security Symposium (USENIX Security'22), 2022.