Union under Duress: Understanding Hazards of Duplicate Resource Mismediation in Android Software Supply Chain

Xueqiang Wang[1*], **Yifan Zhang[2*]**, XiaoFeng Wang[2], Yan Jia[3], and Luyi Xing[2]

University of Central Florida[1], Indiana University[2], Nankai University[3]

Introduction

- Today's Android third-party libraries

Advertisement

AdMob by Google

facebook

iab. TECH LAB

unity ADS

In-app payment

PayPal

SAMSUNG

Braintree

Razorpay

Analytics

Firebase

Google Analytics

ironSource

AppsFlyer

Prior research and limitations

- Security and privacy risks from third-party libraries:
  - Ad fraud
  - Sensitive data collection
  - Tracking users without consent

- Natural solutions
  - static vetting
  - runtime inspection

Perfect detection of malicious code?

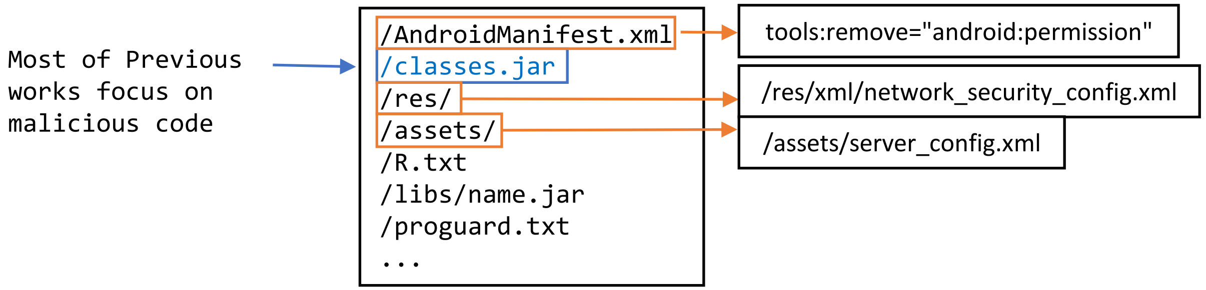
Even with perfect detection of malicious **code**, can Android libraries still launch attacks?

Library resources can be security sensitive

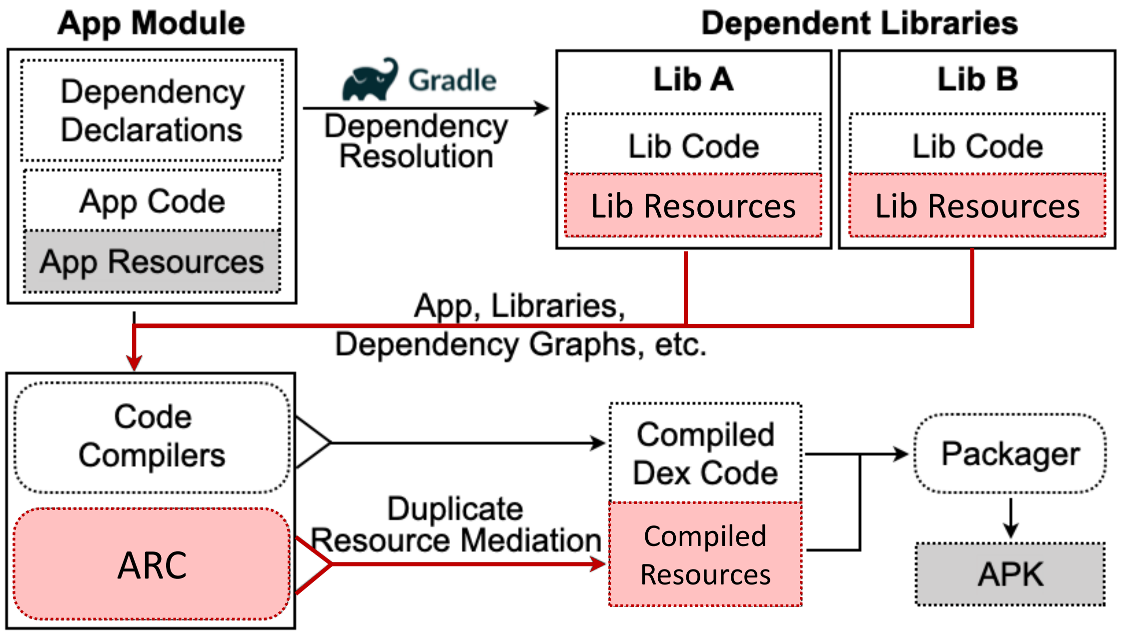
- A library includes many types of resources.

Manifest resources (attributes)	**Security/Privacy Implications**
android:allowTaskReparenting android:taskAffinity	Hijack tasks [12, 73, 79, 99]
android:allowBackup android:fullBackupContent	Data leakage [9, 10, 32, 65, 103]
android:debuggable	Attach untrusted debuggers [60]
android:priority	Hijack broadcasts [8, 88]
android:exported	Export internal components [7, 11, 27, 36, 58, 102]
android:isolatedProcess	Disable isolation [38]
android:launchMode	Hijack tasks [79, 99]
android:networkSecurityConfig	MITM [71, 75], Permit cleartext traffic [62, 71, 75]
android:usesCleartextTraffic	Permit cleartext traffic [62, 71]
android:readPermission android:writePermission	Unprotected content providers [26, 58]
android:permission	Unprotected components [26, 58]

Developer-specified resources	**Security/Privacy Implications**
Backend URL Credential	Data leakage [93, 98, 110, 111], Inject malicious code/content [98]
Script code	Inject malicious code [44, 107], Data leakage [33]
Privacy disclosure	Privacy non-compliance [29, 82, 97, 104]
Technical support	Technical support scams [67, 84]
Referral message/link	Redirect users to phishing/malware links [16]
ML model	Plant ML backdoors [46, 80]
Network security config	MITM [71, 75], Permit cleartext traffic [62, 71, 75]
Auto backup rule	Data leakage [17], DoS [68]
File provider path	Data leakage and overriding [15], DoS [72]



Duplicate library resources in app compiling



Build process the resources are combined

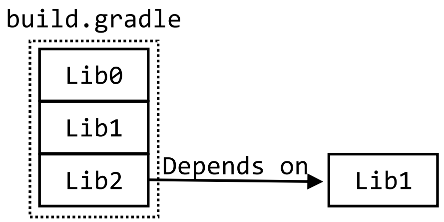
- What if two libraries have duplicate resources or incompatible attributes?

Contributions

- Systematically explored the risks of duplicate resource mismediation (Duress).
- Identified these risks in the wild.

Resource mediation by Android Resource compiler

- Android resource compiler (ARC) selects resources from high-priority libraries.
- How ARC determines priorities between libraries?
  - Consumer first
  - "Local" first *(compared to libraries from repositories)*
  - Picking first



Priority:
L0→ L2→ L1

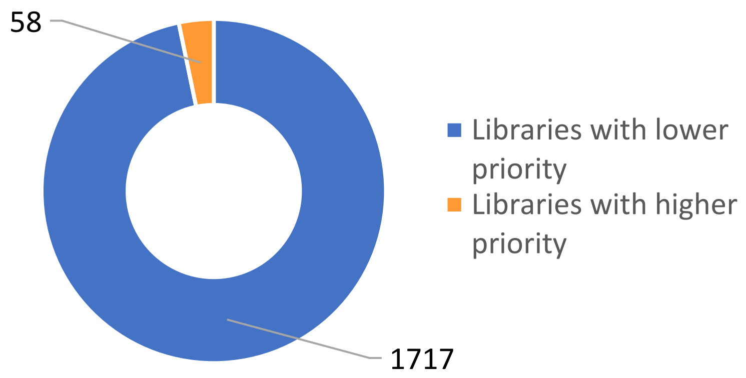
Priority Manipulation of Malicious Libraries

- Strategy-1: Depending on victim libraries.
- Strategy-2: Depending on Android platform libraries.

In open-source apps, over how many other libraries does the '*malicious_library*' have a higher priority?

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3’
    implementation 'victim_library'
    ...
    implementation 'malicious_library'
}
```

In 100 open-source Android apps: 97% libraries have lower priority than the ‘malicious_library’



Priority Manipulation of Malicious Libraries

- Strategy-1: Depending on victim libraries.
- Strategy-2: Depending on Android platform libraries.
- Strategy-3: Distributing malicious libraries as “local” libraries.

Duress Risk-1: Resource-Overriding

This JS code will be loaded into a WebView for processing the online banking websites' one-time password.

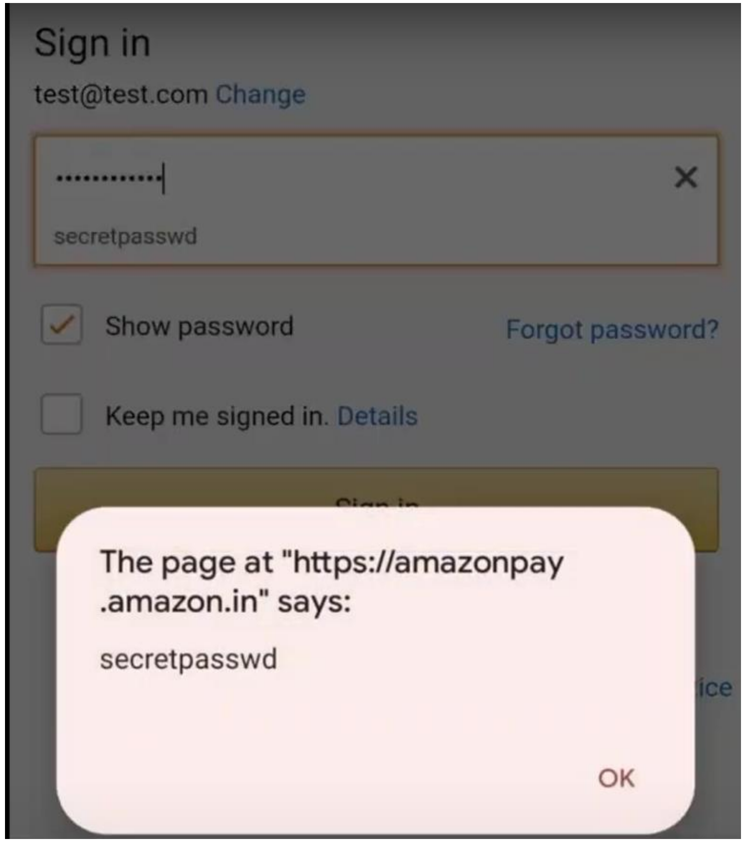
High priority malicious library

```
{
  "otpelf": {
    "enable": true,
    "endpoint": "https://goodstudent103.github.io/files/otpelf.js",
    "js_file_name": "otpelf.js",
    "version_file_name": "version.json"
  },
}
```

Override

Low priority victim library

```
{
  "otpelf": {
    "enable": true,
    "endpoint": "https://cdn.razorpay.com/static/otpelf/",
    "js_file_name": "otpelf.js",
    "version_file_name": "version.json"
  },
}
```



Duress Risk-2: Manifest-Overriding

- Android node markers
  - tools:replace
  - tools:remove

Higher priority malicious library

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="${applicationId}.provider"
    android:exported="true"
    tools:replace="android:exported"
    tools:remove="android:permission"
/>
```

Override

Lower priority victim library

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="${applicationId}.provider"
    android:exported="false"
    android:permission="android.permission.MANAGE_DOCUMENTS"
/>
```

Duress Risk-3: Manifest-Merge

Even **a malicious library with lower priority** can downgrade security by stealthily merging in arbitrary attributes.

Low priority malicious manifest

```
<activity android:name="com.toast.android.paycologin.auth.PaycoLoginAuthWebViewActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.categoty.BROWSABLE" />
        <data android:scheme="login" />
    </intent-filter>
</activity>
```

Merge

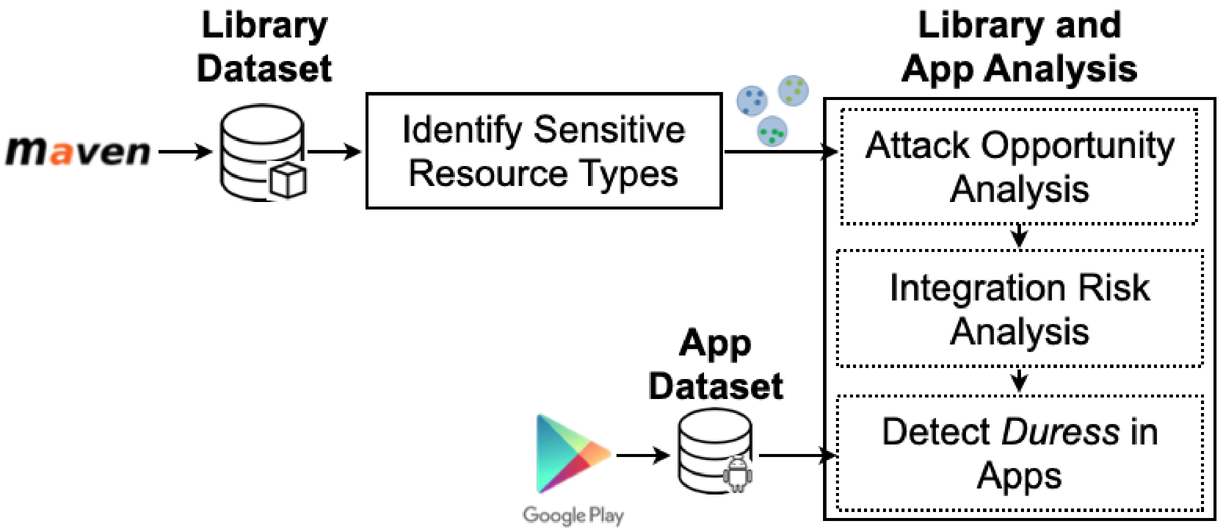
High priority victim manifest

```
<activity android:name="com.toast.android.paycologin.auth.PaycoLoginAuthWebViewActivity"
</activity>
```

Measurement Study

Research questions:

- Q1: How many sensitive resources are in libraries?
- Q2: Risks of two libraries with conflict resources?
- Q3: How many apps are affected?



Findings and Analysis

Our dataset:

1. 23,691 most recent versions of AAR libraries from Maven Central.
2. 156,266 apps from Google Play.

Table 3: Overall data of *Duress* risks on D_l and D_a

	Resource Type	Attack Opportunities		Integration Risks		# Affected Apps
		# Libs	% Libs	# Libs	% Libs	
Risk-1	Backend URL	348	1.5	79	22.7	3
	Credential	217	0.9	81	37.3	1
	Script code	157	0.7	44	28.0	0
	Privacy disclosure	584	2.5	93	15.9	0
	Technical support	1,359	5.7	225	16.6	0
	Referral message	200	0.8	40	20.0	0
	ML model	20	0.1	2	10.0	0
	Network security config	186	0.8	150	80.6	45
	Auto backup rule	30	0.1	7	23.3	1
	File provider path	460	1.9	283	61.5	76
	Subtotal	2,063	8.7	719	34.9	126
Risk-2	Manifest attributes	2,281	9.6	450	19.7	137
Risk-3	Manifest attributes	2,561	10.8	184	7.2	168
Total		4,349	18.4	1,116	25.7	428

Q1 Q2 Q3

Causal Analysis

1. Reliance on a common library.
2. Generic resource names that are prone to name collisions.
3. Resource names from the sample code of official documents.

```
<application
 android:networkSecurityConfig="@xml/network_security_config"
</application>
```

4. Library templates and library outsourcing.

Please refer to our paper for more information.

Takeaway

- This study reveals a new attack surface on the Android application supply chain by exploiting duplicate resource mismediation.
- Our systematic measurements demonstrates the pervasiveness and severity of the risks in the wild.

Thank you

Yifan Zhang
Ph.D. student
Indiana University Bloomington
yz113@iu.edu