Recorded Future®

# Hunting potential C2 commands in Android malware via Smali string comparison and control flow analysis

**JunWei Song, Senior Malware Researcher**

**Virus Bulletin, September 2025**

# About Me - JunWei Song

## Work

- Sr. Malware Researcher @ Recorded Future Triage Sandbox
- Analyze malware / Ensure our sandbox catches every sneaky malware

## Areas of Interest

- Malware analysis / Developing tools to aid malware analysis (mainly Android)

## Volunteer for PyCon TW

- Program Team since 2020

 @JunWei__Song  JunWei Song  krnick

# Agenda

1. Introduction: Why C2 Commands in Android Malware Matter
2. Common C2 Command Patterns in Android Malware
3. Methodology & Implementation
4. Conclusion & Future Work

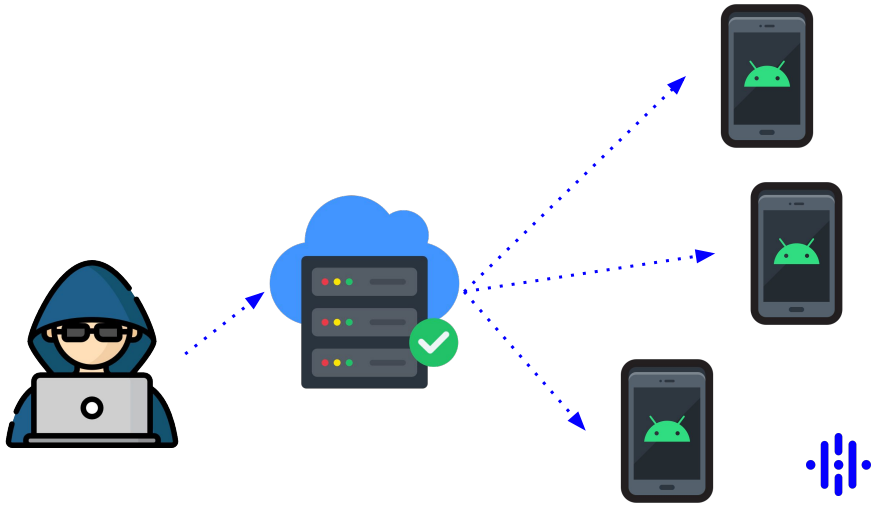


# Introduction: Why C2 Commands in Android Malware Matter

# Introduction: Why C2 Commands in Android Malware Matter

- The evolving threat of Android malware
- Their core functionality relies on Command and Control (C2) servers for instructions and data exfiltration.

One thing stays the same: **C2 Commands**



# Introduction: Why C2 Commands in Android Malware Matter

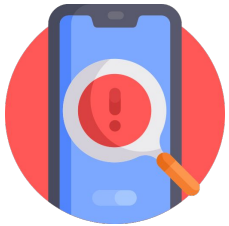
**But why do we look for the C2 commands?**

- The data they are targeting
- The malicious actions they are performing
- The way they communicate with the C2 server
- They serve as key indicators for classifying malware families

# Introduction: Why C2 Commands in Android Malware Matter

## Bottlenecks of Traditional Analysis in Finding C2 Commands

- Manual reverse engineering:
  - Extremely time-consuming
- Dynamic analysis often struggles with:
  - Encryption & Obfuscation
  - Unavailable C2 servers
  - Advanced anti-analysis techniques

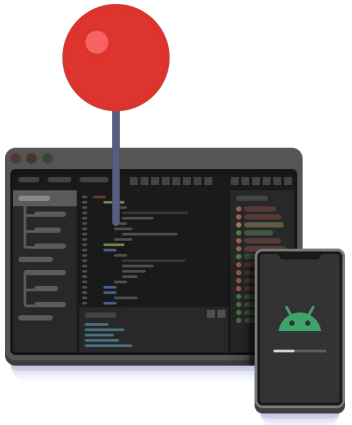


# Introduction: Why C2 Commands in Android Malware Matter

**This leads us to the following questions:**

1. How to effectively locate the function that handles the C2 commands
2. Finding out where the C2 commands are hiding
3. Or even identify an unknown malware in the wild

That is where **C2 command handling patterns** come into play!



# Common C2 Command Patterns in Android Malware

# Common C2 Command Patterns in Android Malware

## Let's check a common code structure for C2 command handling

get_sms

```
if (cmd.equals("get_sms")) {
    // Handle command to retrieve SMS messages
} else if (cmd.equals("get_contacts")) {
    // Handle command to retrieve the device's contact list
} else if (cmd.equals("upload_file")) {
    // Handle command to upload a specific file
} else if (cmd.equals("get_location")) {
    // Handle command to retrieve the device's GPS location
} else if (cmd.equals("take_screenshot")) {
    // Handle command to take a screenshot of the device's display
}
```


- Action: C2 Command Handling
- Where: Code Structures

# Common C2 Command Patterns in Android Malware

**By analyzing multiple recent malware families, we identified four primary C2 command handling structures:**

1. Dense if-else chains and switch-case blocks with multiple command string comparisons (e.g., TgToxic, AhRat, Hook, Alienbot, Octo)
2. HashMap mappings between command strings and functions (e.g., XLoader)
3. Command strings are stored in a large array (e.g., Copybara)
4. Multiple constant command strings embedded in a single class (e.g., Nexus, Cerberus, Crocodilus)

# #1: C2 command-handling pattern: if-else/switch

- **Description**

  Malware often uses dense if-else or switch-case structures to compare command strings and execute corresponding functions.

- **Characteristics**
  - **High-Frequency API:** string.equals()
  - **Frequent Smali Opcodes:** const-string, invoke-virtual, move-result, if-eqz
  - **Specific Opcode Presence:** sparse-switch (non-contiguous string comparisons)

Case Study: TgToxic, an Android banking trojan, for its 2024 version (reported by Hatching), was first reported by Trend Micro [1] in 2023.



[1] https://www.trendmicro.com/en_us/research/23/b/tgtoxic-malware-targets-southeast-asia-android-users.html

# #1: C2 command-handling pattern: if-else/switch

## TgToxic code snippet for handling C2 commands

### Key Components

- C2 strings
- Switch statement
- If statement
- string.equals()

```
String POoIo0IIOi1 = PlI0lo0i00iO.POoIo0IIOi1(objArr[0].toString(), KszahaVmkrjij.HI0IOiOlo);
if (POoIo0IIOi1 != null) {
    try {
        JSONObject jSONObject2 = new JSONObject(POoIo0IIOi1);
        try {
            String string = jSONObject2.getString("action");
            switch (string.hashCode()) {
                case -1949226861:
                    if (string.equals("updateApk")) {
                        c = 29;
                        break;
                    }
                    c = 65535;
                    break;
                case -1884362643:
                    if (string.equals("stopCam")) {
                        c = '\n';
                        break;
                    }
                    c = 65535;
                    break;
                case -1858379769:
                    if (string.equals("restartMe")) {
                        c = '-';
                        break;
                    }
                    c = 65535;
                    break;
```

**Number of C2 commands: 94**

# #1: Smali Opcodes: TgToxic's C2 Command Handling

- 0026f432: sparse-switch v10, :s_switch_1278 ← **Starting Point**
- 0026f438: goto/16 :goto_0485
- 0026f43c: const-string v10, "installPermission" # string@4209
- 0026f440: invoke-virtual {v7, v10}, Ljava/lang/String;→equals(Ljava/lang/Object;)Z
- 0026f446: move-result v7
- 0026f448: if-eqz v7, :cond_0485
- 0026f44c: const/16 v7, 0×4c
- 0026f450: goto/16 :goto_0486
- 0026f454: const-string v10, "gestureB" # string@3b0f
- 0026f458: invoke-virtual {v7, v10}, Ljava/lang/String;→equals(Ljava/lang/Object;)Z
- 0026f45e: move-result v7
- 0026f460: if-eqz v7, :cond_0485
- ...

**Repeated Pattern of Opcodes**

# #2: C2 command-handling pattern: HashMap

- **Description**

  Malware uses a HashMap to link C2 command strings to their corresponding functional modules.

- **Characteristics**
  - **High-Frequency API:** HashMap.put()
  - **Frequent Smali Opcodes:** const-string, new-instance, invoke-virtual
  - **Specific Opcode Presence:** iget-object

Case Study: XLoader (also known as MoqHao [1]), an Android banking trojan.

[1] https://www.trendmicro.com/en_us/research/18/d/xloader-android-spyware-and-banking-trojan-distributed-via-dns-spoofing.html

# #2: C2 command-handling pattern: HashMap

XLoader code snippet for handling C2 commands.

```
J j2 = new J(this, 7);
d0 d0Var = this.f393f;
d0Var.f234c.put("sendSms", j2);
J j3 = new J(this, 8);
HashMap hashMap = d0Var.f234c;
hashMap.put("setWifi", j3);
hashMap.put("gcont", new J(this, 9));
hashMap.put("lock", new J(this, 10));
hashMap.put("bc", new J(this, 11));
hashMap.put("setForward", new J(this, 12));
hashMap.put("getForward", new J(this, 13));
hashMap.put("hasPkg", new J(this, 14));
hashMap.put("setRingerMode", new J(this, 15));
hashMap.put("setRecEnable", new J(this, i6));
hashMap.put("reqState", new J(this, i4));
hashMap.put("showHome", new J(this, i3));
hashMap.put("getnpki", E.f157c);
hashMap.put("http", E.f158d);
hashMap.put("call", new J(this, i5));
hashMap.put("get_apps", new J(this, 4));
hashMap.put("ping", new J(this, 5));
hashMap.put("getPhoneState", new J(this, i2));
```

**Number of C2 commands: 20**

```
public /* synthetic */ J(Loader loader, int i2) {
    this.f170a = i2;
    this.f171b = loader;
}
```

```
case 4:
    C.f.e((Object[]) obj, "it");
    Context context4 = loader.f388a;
    if (context4 == null) {
        C.f.g("ctx");
        throw null;
    }
    PackageManager packageManager = context4.getPackageManager();
    ArrayList arrayList = new ArrayList();
    List<PackageInfo> installedPackages = packageManager.getInstalledPackages(0);
    C.f.d(installedPackages, "getInstalledPackages(...)");
```

## Key Components

- C2 strings
- Instance of a class
- HashMap.put()

# #2: Smali Opcodes: XLoader's C2 Command Handling

- 00021022: iget-object v5, v4, La/d0;→c:Ljava/util/HashMap;
- 00021026: const-string v6, "sendSms" # string@0895
- 0002102a: invoke-virtual {v5, v6, v14}, Ljava/util/HashMap;→put()
- 00021030: new-instance v14, La/J; # type@006a
- 00021034: const/16 v5, 0×8
- 00021038: invoke-direct {v14, p0, v5}, La/J;→<init>(Lcom/Loader;, I)V
- 0002103e: iget-object v4, v4, La/d0;→c:Ljava/util/HashMap;
- 00021042: const-string v5, "setWifi" # string@08cd
- 00021046: invoke-virtual {v4, v5, v14}, Ljava/util/HashMap;→put()
- 0002104c: new-instance v14, La/J; # type@006a
- 00021050: const/16 v5, 0×9
- 00021054: invoke-direct {v14, p0, v5}, La/J;→<init>(Lcom/Loader;, I)V
- ...

**Starting Point**

**Repeated Pattern of Opcodes**

# #3: C2 command-handling pattern: Array

- **Description**

  Defines all C2 commands within a large array.
- **Characteristics**
  - **Frequent Smali Opcodes:** const-string, aput-object
  - **Specific Opcode Presence:** new-array

Case Study: Copybara [1], an Android banking trojan that first appeared in November 2021 and which resurfaced in November 2023.

[1] https://www.zscaler.com/blogs/security-research/technical-analysis-copybara




# #3: C2 command-handling pattern: Array

## Copybara code snippet for handling C2 commands

### Key Components

- C2 strings
- Array
- Switch statement

```
public static String _mqtt_messagearrived(String str, byte[] bArr) throws Exception {
    try {
    } catch (Exception e) {
        processBA.setLastException(e);
        Common.LogImpl("3801373", BA.ObjectToString(Common.LastException(processBA)), 0);
        _send_myerror_tosrv(Common.LastException(processBA).getMessage(), "mqtt_MessageArrived");
        return "";
    }
    if (!str.equals("commands_FromPC")) {
        return "";
    }
    B4XSerializator b4XSerializator = new B4XSerializator();
    new Map();
    Map map = (Map) AbsObjectWrapper.ConvertToWrapper(new Map(), (java.util.Map) b4XSerializator.ConvertBytesToObject(bArr));
    globalparameters globalparametersVar = mostCurrent._vvvvvvvvvvvvvv3;
    if (!globalparameters._vvvv7.equals(BA.ObjectToString(map.Get("deviceid")))) {
        return "";
    }
    switch (BA.switchObjectToInt(map.Get("fun"), "open_app_setngs", "hid_app_icno", "send_admn_lckdvcs_on", "send_inj_lst", "send_custom_opencam",
        case 0:
            _open_app_setngs_data(map);
            return "";
        case 1:
            _hid_app_icno(map);
            return "";
        case 2:
            _send_admn_lckdvcs_on();
            return "";
        case 3:
            _send_inj_lst(map);
            return "";
```

**Number of C2 commands: 59**

# #3: Smali Opcodes: Copybara's C2 Command Handling

- 003e4d54: const-string v6, "fun" # string@9dab
- 003e4d58: invoke-virtual {v5, v6}, Lanywheresoftware/b4a/objects/collections/Map;→Get(Ljava/lang/Object;)...
- 003e4d5e: move-result-object v6
- 003e4d60: const/16 v7, 0×3e
- 003e4d64: new-array v7, v7, [Ljava/lang/Object;
- 003e4d68: const-string v8, "open_app_setngs"
- 003e4d6c: aput-object v8, v7, v4
- 003e4d70: const-string v8, "hid_app_icno"
- 003e4d74: const/4 v9, 0×1
- 003e4d76: aput-object v8, v7, v9
- 003e4d7a: const-string v8, "send_admn_lckdvcs_on"
- 003e4d7e: const/4 v10, 0×2
- 003e4d80: aput-object v8, v7, v10

**Starting Point**

**Repeated Pattern of Opcodes**

# #4: C2 command-handling pattern: Constant strings class (static field)

- **Description**

  Defines a large number of constant strings as C2 commands within a single class.

- **Characteristics**
  - **Frequent Smali Opcodes:** const-string, sput-object

Case Study: Nexus [1], an Android banking trojan associated with the SOVA Android malware family, which was first identified in 2023.

[1] https://www.cleafy.com/cleafy-labs/nexus-a-new-android-botnet




# #4: C2 command-handling pattern: Constant strings class (static field)

## Nexus code snippet for handling C2 commands

### Key Components

- C2 strings
- Constant strings class
  - (static field)

```
static {
    PERMISSION_LIST = Build.VERSION.SDK_INT >= 26
    get2fa = "get2fa";
    start2faactivator = "start2faactivator";
    stop2faactivator = "stop2faactivator";
    delbot = "delbot";
    openUrl = "openurl";
    startlock = "startlock";
    stoplock = "stoplock";
    admin = "getperm";
    delapp = "delapp";
    clearappdata = "clearappdata";
    startextraverbose = "startextraverbose";
    stopextraverbose = "stopextraverbose";
    starthidenpush = "starthidenpush";
    stophidenpush = "stophidenpush";
    hidesms = "starthidesms";
    stophidensms = "stophidesms";
    scancookie = "scancookie";
    stopcookie = "stopcookie";
    scaninject = "scaninject";
    stopscan = "stopscan";
    getsms = "getsms";
```

**Number of C2 commands: 39**

# #4: Smali Opcodes: Nexus's C2 Command Handling

Constant strings class (static field)

- 004d9234: sput-object v0, Lcom/tapston/burgerking/Const;→PERMISSION_LIST:Ljava/util/List;
- 004d9238: const-string v0, "get2fa"
- 004d923c: sput-object v0, Lcom/tapston/burgerking/Const;→get2fa:Ljava/lang/String;

- 004d9240: const-string v0, "start2faactivator"
- 004d9244: sput-object v0,Lcom/tapston/burgerking/Const;→start2faactivator:Ljava/lang/String;

- 004d9248: const-string v0, "stop2faactivator"
- 004d924c: sput-object v0, Lcom/tapston/burgerking/Const;→stop2faactivator:Ljava/lang/String;
- ...

**Repeated Pattern of Opcodes**

# #4: C2 command-handling pattern: Constant strings class (instance field)

- **Description**

  Defines a large number of constant strings as encrypted strings within a single class.
- **Characteristics**
  - **Frequent Smali Opcodes:** const-string, invoke-virtual, move-result-object, iput-object

Case Study: The Cerberus [1], an Android banking trojan. In this variant, RC4 encryption with Base64 encoding is employed.

[1] https://www.threatfabric.com/blogs/cerberus-a-new-banking-trojan-from-the-underworld

# #4: C2 command-handling pattern: Constant strings class (instance field)

## Cerberus code snippet for handling encrypted constant strings.

### Key Components

- C2 strings
- Constant strings class
  - (instance field)

```
public class a {
    public boolean a = false;
    public String b = b("pejofjdnxlapNGU2NjNlZWI1NQ==");
    public String c = b("cnugorsjkayfOThhY2YxZjJhYjBkZDAyN2Y2Y2UyMmY4MzE2Mw==");
    public String d = b("dbebfxtwqutzYzA0YjcwNWI2NmFkOTYxNDRkZWMwZWY4YmM=");
    public String e = b("lnlanhaussdsMjlhYmQxN2E2MjdmNmVlYmY3ZTBhMjUyOWI1YQ==");
    public String f = b("nonyzppwyyjtNjBkOWM2ZjU3NmFmZWJjZDQ4ZWZlZmU1YjU=");
    public String g = b("tgjspamxouujYzg0ODBmYjA=");
    public String h = b("ocicvxpcwamrYTRkMmRkY2FlZDFhNTE3YmM0YjE3NzdmOWU2ZDNi");
    public String i = b("mkrpbwlqemmgYmVjNGRlMzA=");
    public String j = b("dmnlxjqwgsfjNjJmYTI2Zjg4Y2EzODg2NGM5ZTg3YWUzMTBkZDk1NzU1NzM5ZDk3YTY3ZWY1MWI1MmY2YTYyYTg=");
    public String k = b("pgegyigmdejmYmU0MTYwY2UxZDMzZDI5MDQ1ZTFhMGMxNjM2Mw==");
    public String l = b("ioxwkjcjsjerNDU0YWEyZWU3Y2NjNDc5YWNhY2VmOGQxZTM2YTI0NDU5Yzk1ZmE1Zg==");
    public String m = b("eraillolkwtrOWVhMGEyYmJmYjRiNTE5YmM2NTc0NjQ1NmJjZDA4MjA0OWUzZWYxMTIwZGM0ZGJk");
    public String n = b("zniscbjkwrtzODQ4YzY0YTk3MjUyZDNlODgwYWRlMTBmNGFmYTVmMWUwNzU2MzdmNzE3");
    public String o = b("cdgbwyvzuxxgNzVkZDA3NWUxNzY5YTFmMWE1M2QzMjMzYzhhMjBkZGRjOWEzY2FlY2U1");
    public String p = b("rvjozvnjeqpxMDNlNjU2NWEzNzk5");
    public String q = b("vwmkltdfezpqN2IzNmYzNmI2ZTdlMTk2OWEyZDM3MTkxOGFkMmVkZDU3MWFjZmE1YzNhYzIxZWM5NDdjNTAw");
    public String r = b("opuojndvvrgnMDZjMzc2OTBmN2M1YTFhN2Mw");
    public String s = b("lvmbvywdqipiZDdiZWM0YTZiNWFkMDZiYmYzMGM=");
    public String t = b("crohfnyrtuvhZjI2NGUyOGVhMmFlMDY5ZDMz");
    public String u = b("wiedsnqtytrdNTM1MGUwYjI0ZjFjNDdmYWQ1ODY1Zg==");
    public String v = b("ngceyqkycffnODY3YzkyMDBlOTRiMDljZDg3N2U=");
    public String w = b("ynreaupxyxgnODBhMTg4ZDFmMjMxOWM5ZDc5NzAyMWMwNmZkMDM3MDA=");
    public String x = b("xiuurgqvrzvuMTc1MDYxMjU2YTA0YTY=");
    public String y = b("vitolkujkfalN2U3MGQ0NmRmOGZlYTJjNjVh");
    public String z = b("wdpipmndgddjZmFkY2E2ZmZlNzYwMWU0ZTJk");
    public String A = b("dxkaydobvtvxNzAwYTNiMDUxNzlhNmEwOWU4");
    public String B = b("cqbimlyvylstYmZkYTEyOTlmZThkNjVmMTNjOTdmOGRm");
```




# #4: Smali Opcodes: Cerberus's encrypted strings

## Constant strings class (instance field)

- 000173cc: invoke-direct {v6}, Ljava/lang/Object;→<init>()V
- 000173d2: const/4 v0, 0
- 000173d4: iput-boolean v0, v6, Lcom/konybqplijaqcazz/nwclqe/a;→a:Z
- 000173d8: const-string v1, "pejofjdnxlapNGU2NjNlZWI1NQ=="
- 000173dc: invoke-virtual {v6, v1},Lcom/konybqplijaqcazz/nwclqe/a;→b(Ljava/lang/String;)Ljava/lang/String;
- 000173e2: move-result-object v1
- 000173e4: iput-object v1, v6, Lcom/konybqplijaqcazz/nwclqe/a;→b:Ljava/lang/String;
- 
- 000173e8: const-string v1, "cnugorsjkayfOThhY2YxZjJhYjBkZDAyN2Y2Y2UyMmY4MzE2Mw=="
- 000173ec: invoke-virtual {v6, v1},Lcom/konybqplijaqcazz/nwclqe/a;→b(Ljava/lang/String;)Ljava/lang/String;
- 000173f2: move-result-object v1
- 000173f4: iput-object v1, v6, Lcom/konybqplijaqcazz/nwclqe/a;→c:Ljava/lang/String;
- 000173f8: const-string v1, "dbebfxtwqutzYzA0YjcwNWI2NmFkOTYxNDRkZWMwZWY4YmM="
- ...

**Repeated Pattern of Opcodes**

# Methodology & Implementation

# Methodology: Extract Patterns from Smali Instructions

**We propose a lightweight static method to identify functions that may contain C2 commands by recognizing C2 command structure patterns:**

- **Parse** Smali opcodes from APK/DEX
- **Count** characteristic opcodes and API calls
- **Flag** potential function that may contain C2 commands based on thresholds

# Methodology: Extract Patterns from Smali Instructions

**#1: C2 command-handling pattern: if-else/switch** (e.g., TgToxic):

- **Key Smali Opcodes:** sparse-switch, const-string, invoke-virtual, move-result, if-eqz
- **Key Android API:** String.equals()
- **Threshold for Opcodes and Android API Occurrences:** e.g., 10
  - (The threshold is based on the expected number of C2 commands)

# Methodology: Opcode and Android API pattern rule

```
{
  "opcode": {
    "sparse-switch": 1,
    "const-string": 10,
    "invoke-virtual": 10,
    "move-result": 10,
    "if-eqz": 10
  },
  "api": {
    "Ljava/lang/String;->equals(Ljava/lang/Object;)Z": 10
  }
}
```

**Threshold:**

**How many times does this opcode or API occur within a function**



# Methodology: The Result

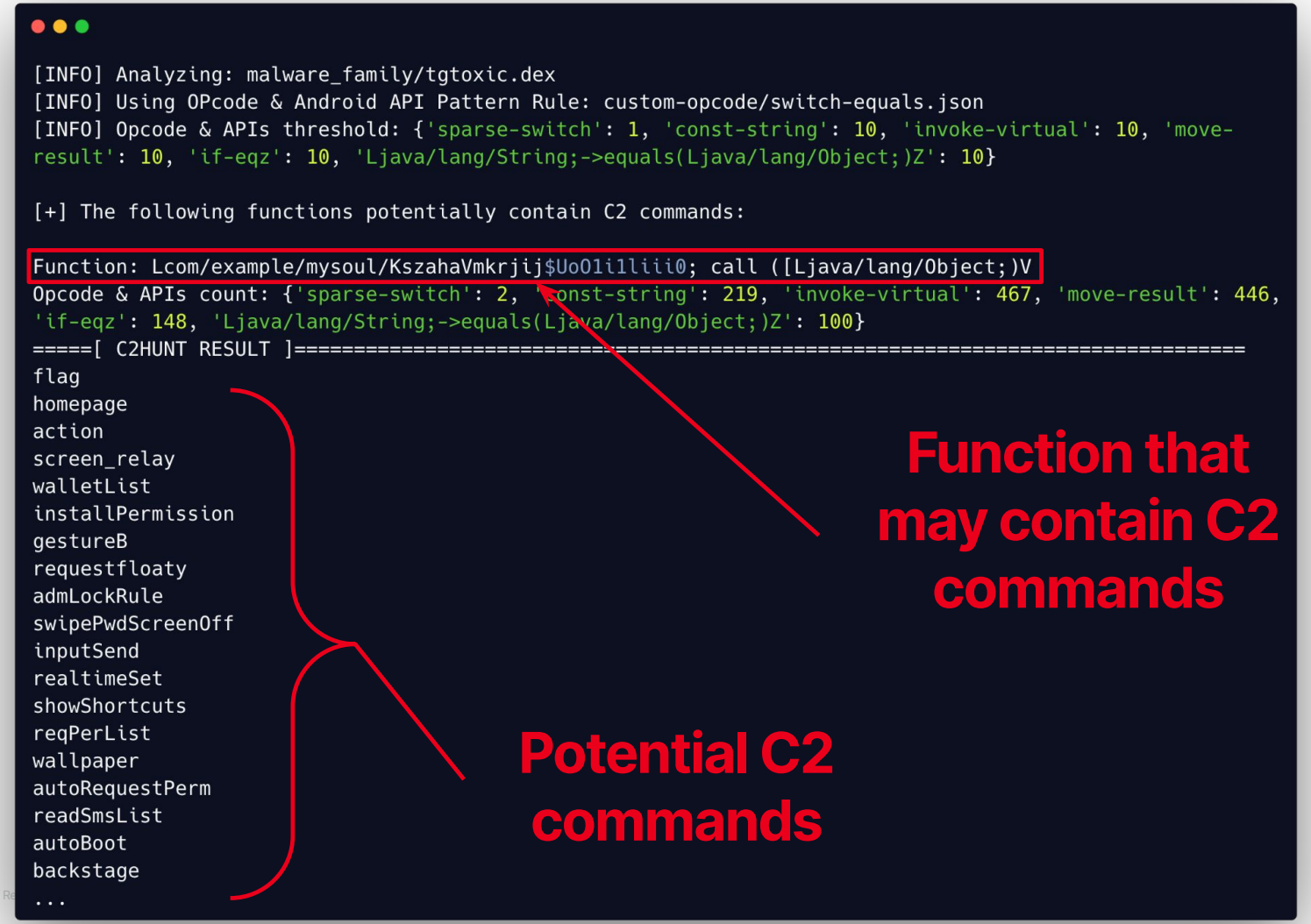
**Once the rule matches, flag as a potential C2-command-containing function**

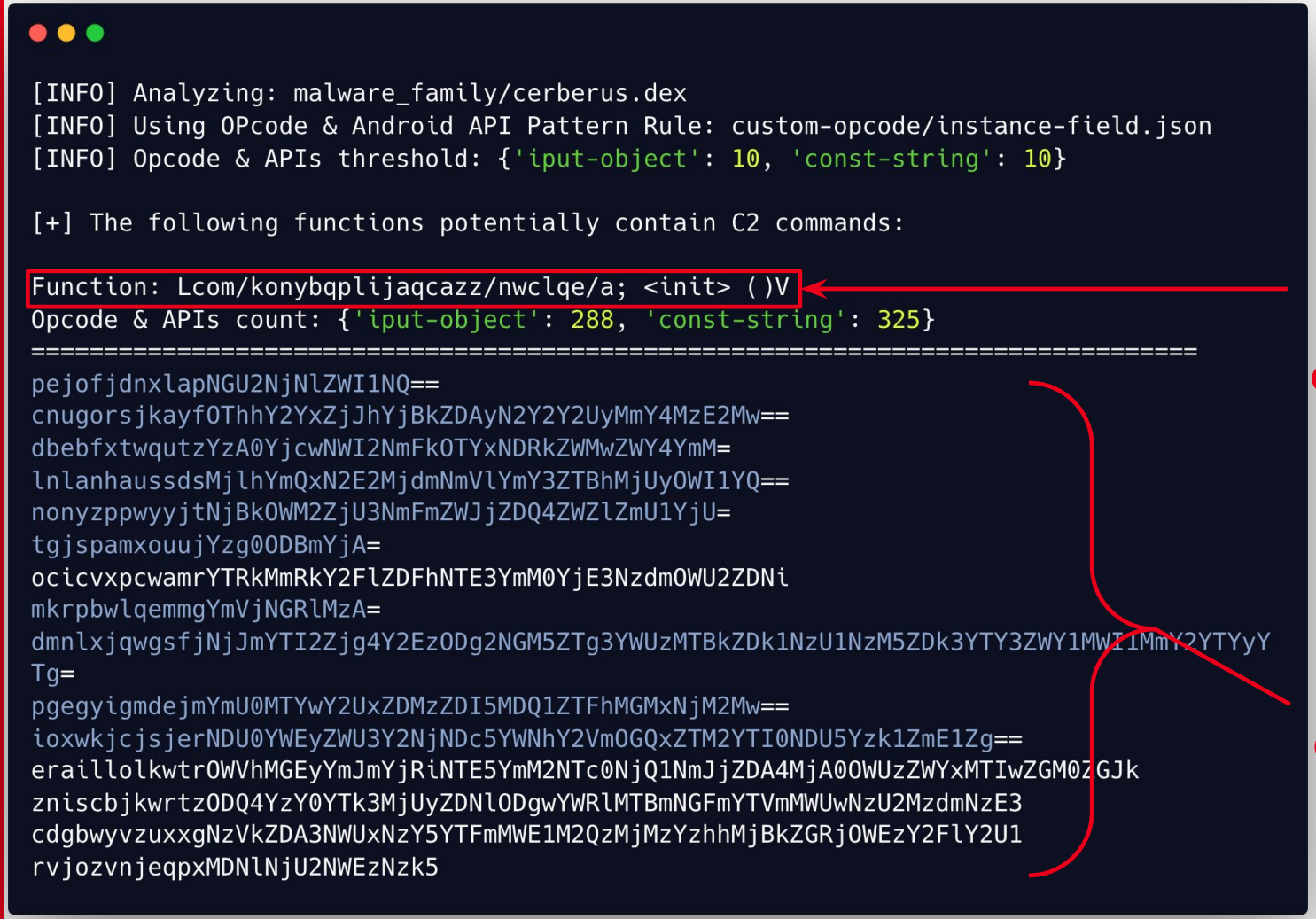
1. Extract the function name (Method Signature)
2. Extract all strings from the function

# Implementation (Tool Demo)

**The implementation is available as open-source code at**

- **https://github.com/krnick/c2hunt**
- Pattern Rules: https://github.com/krnick/c2hunt/tree/main/custom-opcode
- Malware: https://github.com/krnick/c2hunt/blob/main/malware_family.zip
- Malware C2 command counts and handling patterns





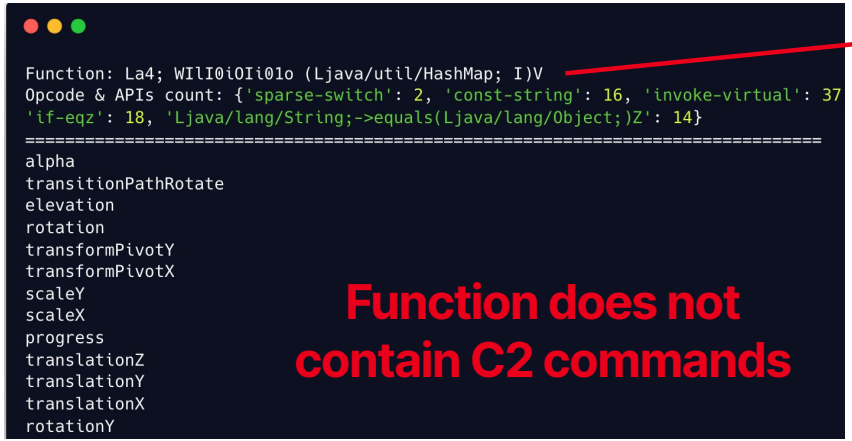

Function that may contain encrypted constant strings

Potential encrypted constant strings

# Note

**While our method may produce some false positives, the primary goal is to:**

- To significantly reduce the time and effort for malware analysts
- By narrowing down thousands of functions to a handful of suspicious candidates




```
public final void WIlI0iOIi01o(HashMap<String, hh> hashMap, int i) {
    for (String str : hashMap.keySet()) {
        hh hhVar = hashMap.get(str);
        Objects.requireNonNull(str);
        char c = 65535;
        switch (str.hashCode()) {
            case -1249320806:
                if (str.equals("rotationX")) {
                    c = 0;
                    break;
                }
                break;
            case -1249320805:
                if (str.equals("rotationY")) {
                    c = 1;
                    break;
                }
                break;
            case -1225497657:
                if (str.equals("translationX")) {
                    c = 2;
                    break;
```

# With this tool, and back to our questions

1. How to effectively locate the function that handles the C2 commands
2. Finding out where the C2 commands are hiding
3. Or even identify an unknown malware in the wild

**Our solution:**

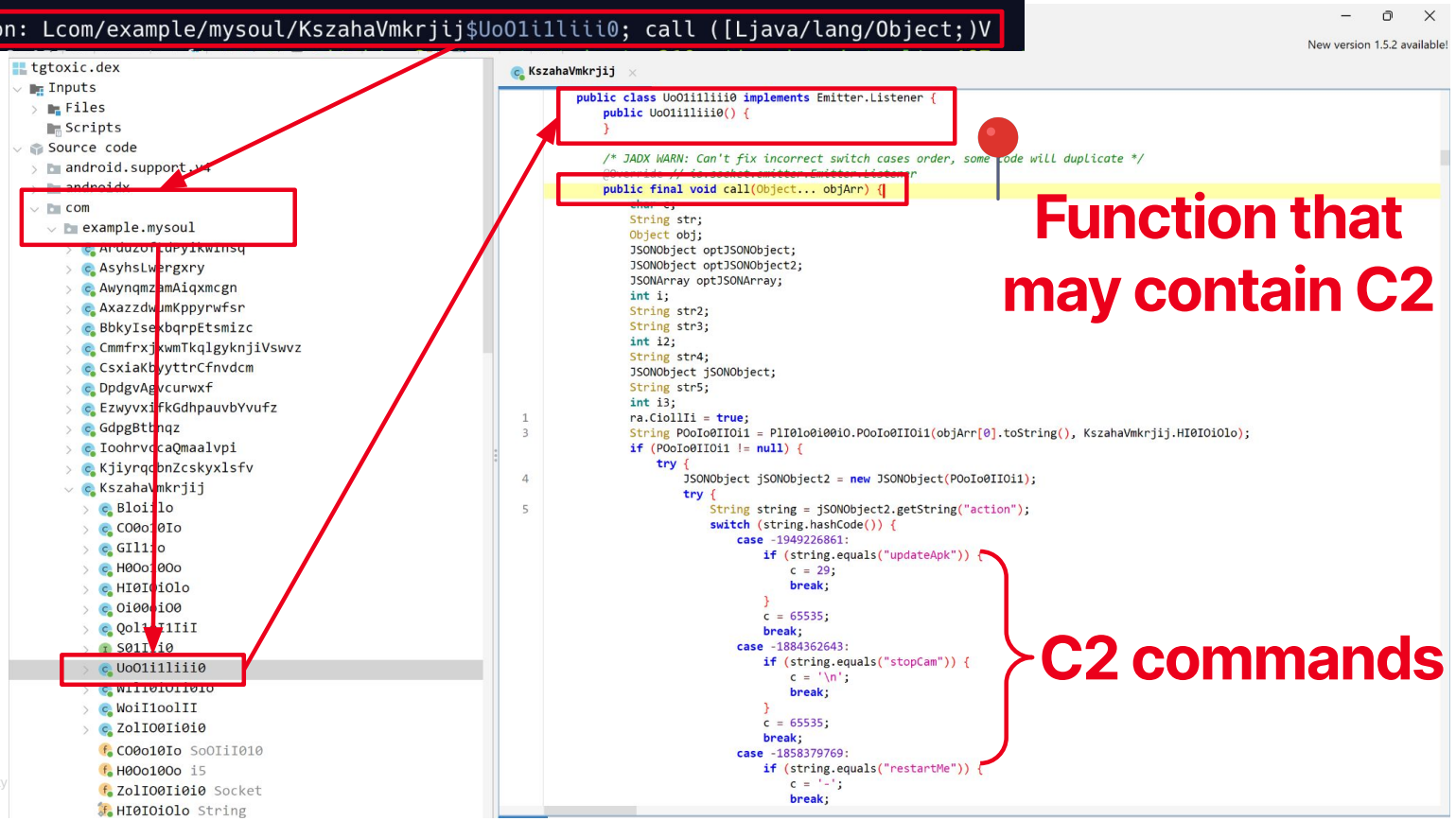
Focus on matching C2 command structural patterns, rather than specific strings or APIs. This allows us to build a scalable detection mechanism that is independent of specific malware families.




# Q1 & Q2: Pinpoint the function that contain C2 commands

```
[+] The following functions potentially contain C2 commands:

Function: Lcom/example/mysoul/KszahaVmkrjij$UoO1i1liii0; call ([Ljava/lang/Object;)V
```



# Q3: Identify an unknown malware in the wild

This tool allows us to investigate the following:

**Unknown Malware**

- Identify previously unreported malware samples that show well-defined C2 command structures.

**New Variants of Known Malware Families**

- By comparing C2 commands to identify potential links to known malware families.
  - In our research, we identified a new version of the TgToxic [1]

[1] https://hatching.io/blog/triage-insights-ep3/

# Conclusion & Future Work

# Conclusion

## Our Approach

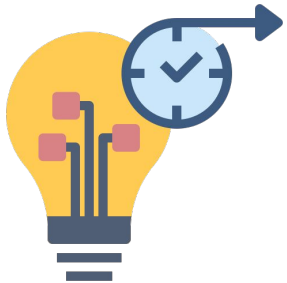
- **Scalable & Independent:** A detection mechanism independent of specific malware families.
- **Cross-Platform Framework:** Adaptable to various CPU architectures (e.g., X86, ARM).
- **Customizable & Extensible:** A flexible framework for diverse analysis needs.

## Challenges: Defining and Optimizing Thresholds

- Developing reliable detection thresholds that are effective across different malware families.

# Future Work

- Discover more C2 command handling patterns
- Enhance rules to support API cross-references (xrefs)
- Automate & Optimize Thresholds



Know what matters.

Act first.

# Thank you so much!

Recorded Future®