The Hong Kong
Polytechnic University
香港理工大學

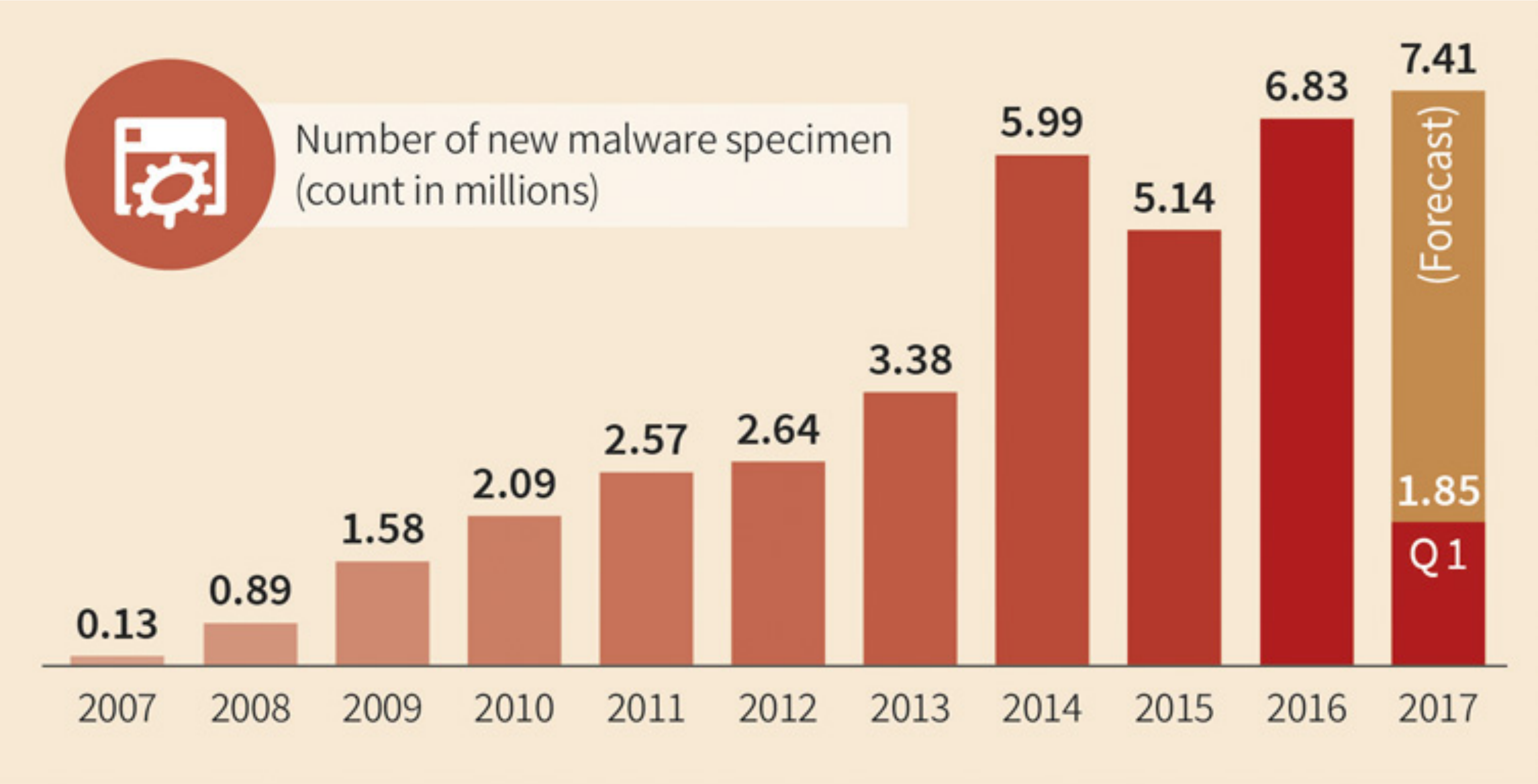
TEXAS A&M
UNIVERSITY

Malton: Towards On-Device Non-Invasive Mobile Malware Analysis for ART

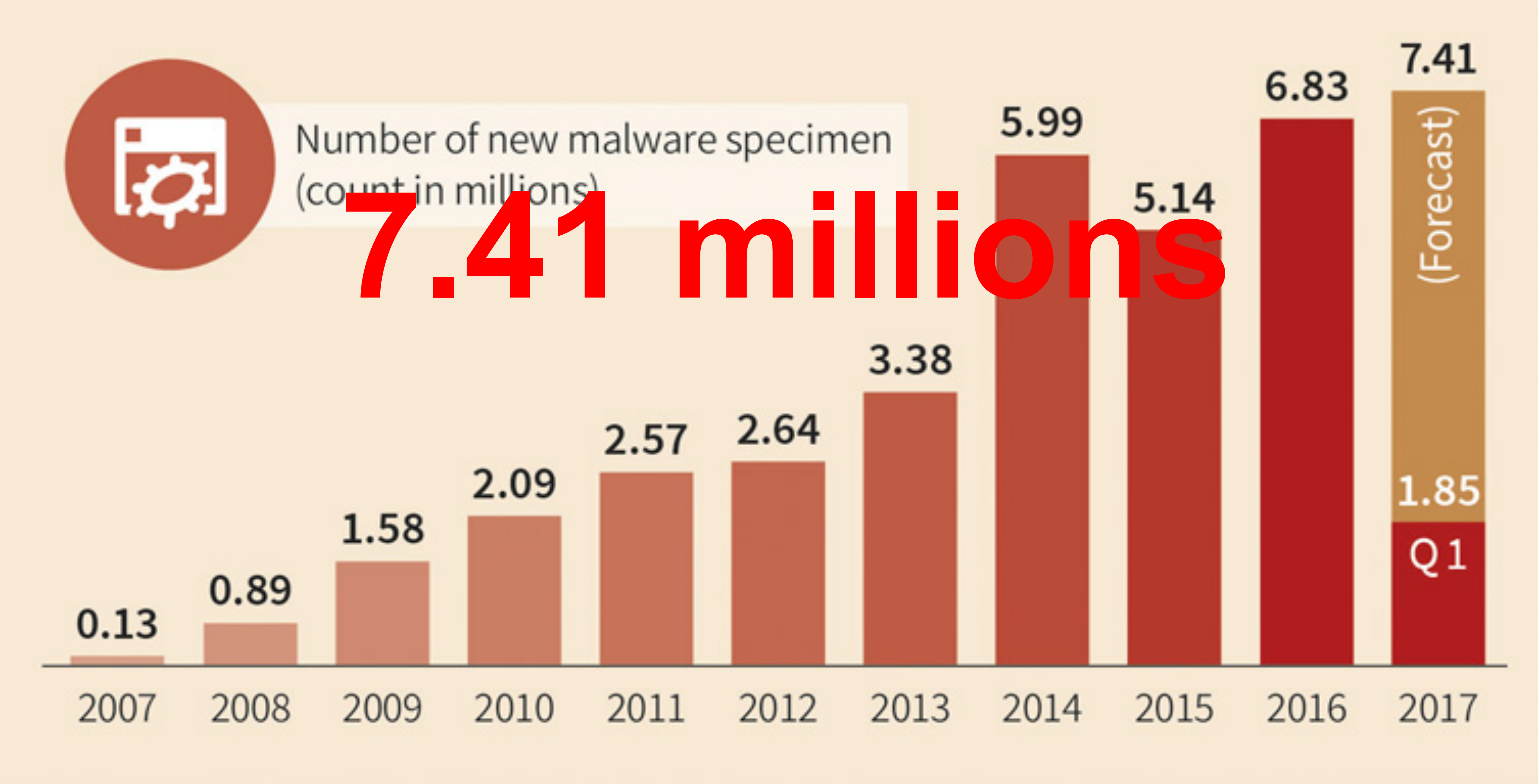
Lei Xue[1], Yajin Zhou, Ting Chen[1], Xiapu Luo[1], Guofei Gu[2]

[1]Department of Computing,
The Hong Kong Polytechnic University

[2]Department of Computer Science & Engineering,
Texas A&M University

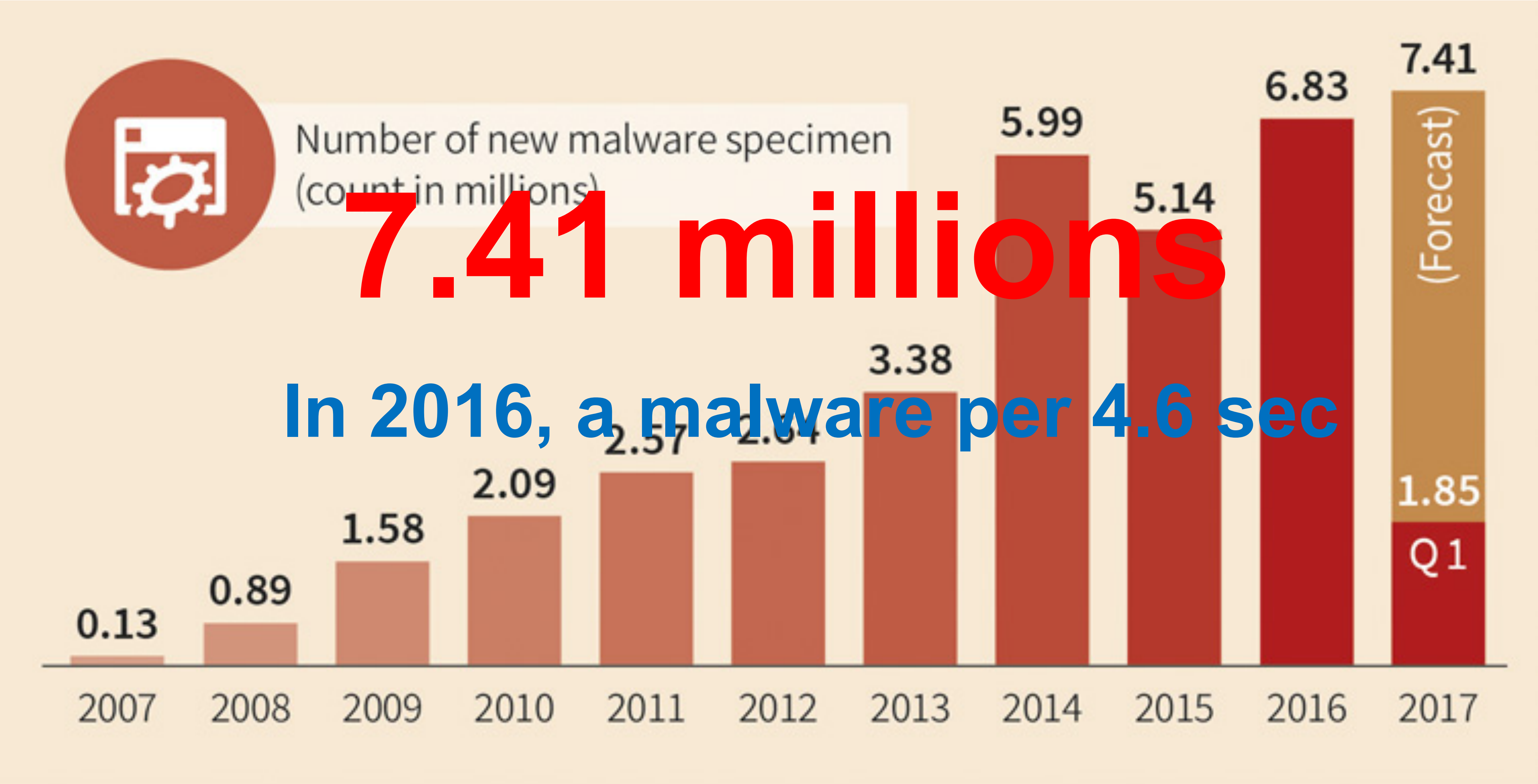


https://www.gdatasoftware.com/blog/2017/04/29666-malware-trends-2017



7.41 millions

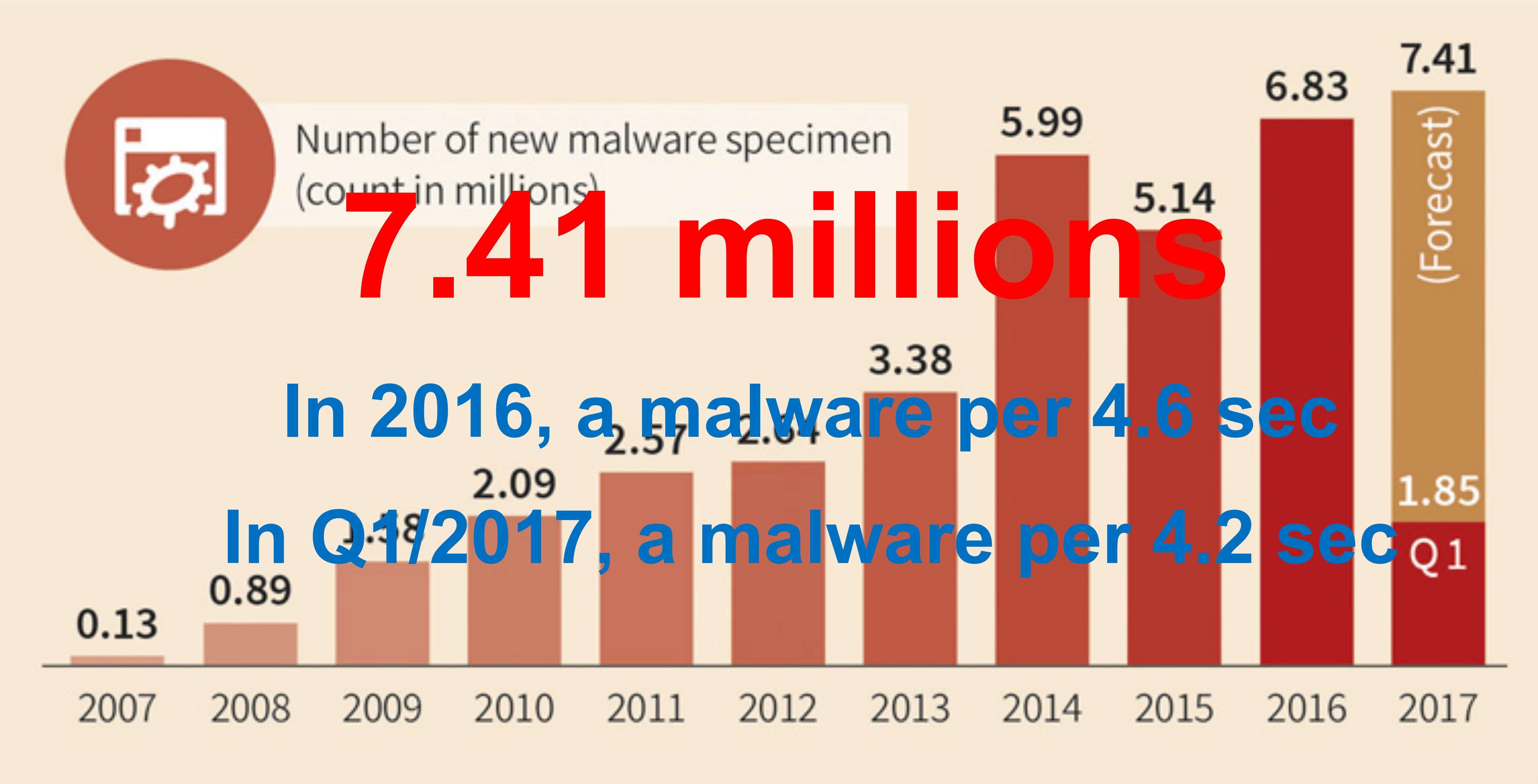
https://www.gdatasoftware.com/blog/2017/04/29666-malware-trends-2017



7.41 millions

In 2016, a malware per 4.6 sec

https://www.gdatasoftware.com/blog/2017/04/29666-malware-trends-2017



7.41 millions

In 2016, a malware per 4.6 sec

In Q1/2017, a malware per 4.2 sec

https://www.gdatasoftware.com/blog/2017/04/29666-malware-trends-2017

Existing Android Malware Analysis Tools

Existing Android Malware Analysis Tools

1

Focus on a specific layer

e.g., DroidBox (Android framework layer), DroidTrace (System layer)

Existing Android Malware Analysis Tools

1

Focus on a specific layer

e.g., DroidBox (Android framework layer), DroidTrace (System layer)

2

Run in the emulator

e.g., DroidScope, Copperdroid (QEMU)

Existing Android Malware Analysis Tools

1 **Focus on a specific layer**

e.g., DroidBox (Android framework layer), DroidTrace (System layer)

2 **Run in the emulator**

e.g., DroidScope, Copperdroid (QEMU)

3 **Modify the DVM or the compiler of ART**

e.g., TaindDroid (DVM), TaintART, ARTist (dex2oat of ART)

Existing Android Malware Analysis Tools

1 **Focus on a specific layer**

e.g., DroidBox (Android framework layer), DroidTrace (System layer)

2 **Run in the emulator**

e.g., DroidScope, Copperdroid (QEMU)

3 **Modify the DVM or the compiler of ART**

e.g., TaindDroid (DVM), TaintART, ARTist (dex2oat of ART)

4 **Modify the target apps**

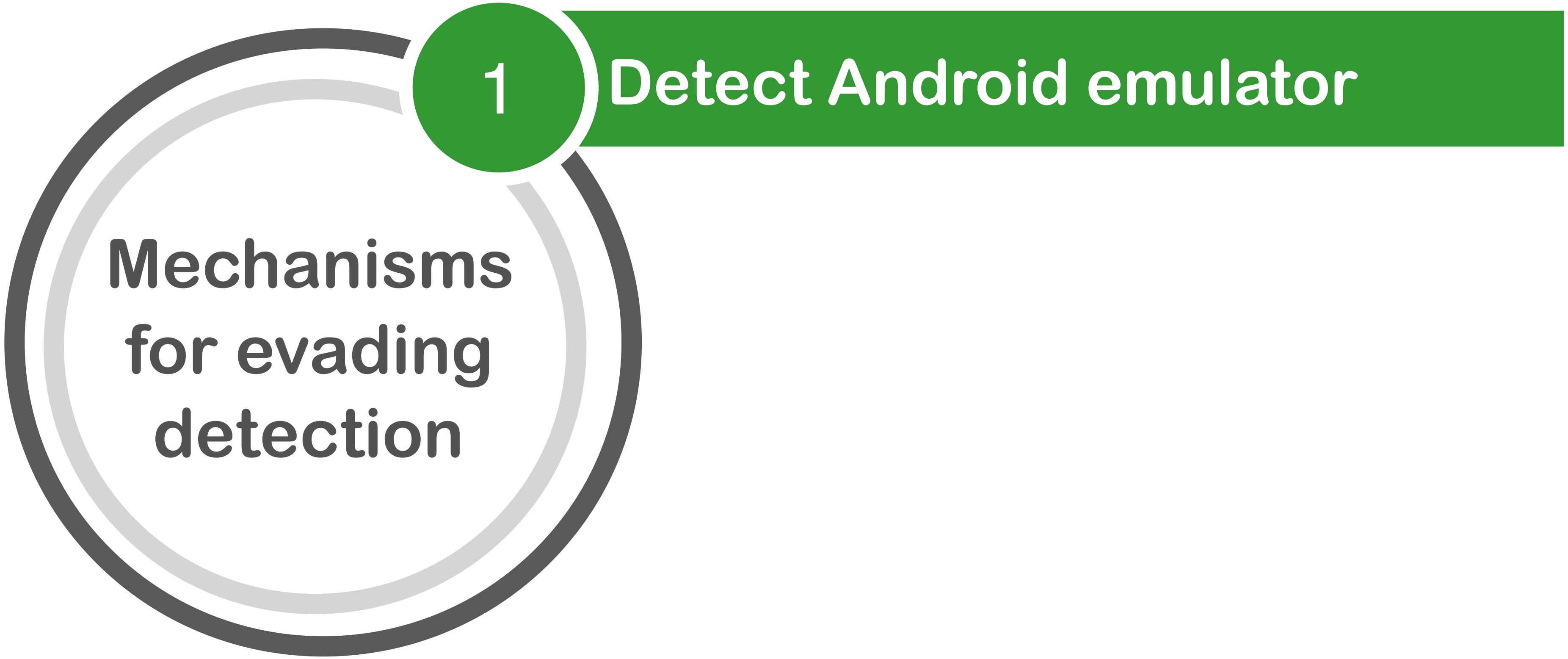
e.g., Aurasium

Malware

Malware

Mechanisms for evading detection

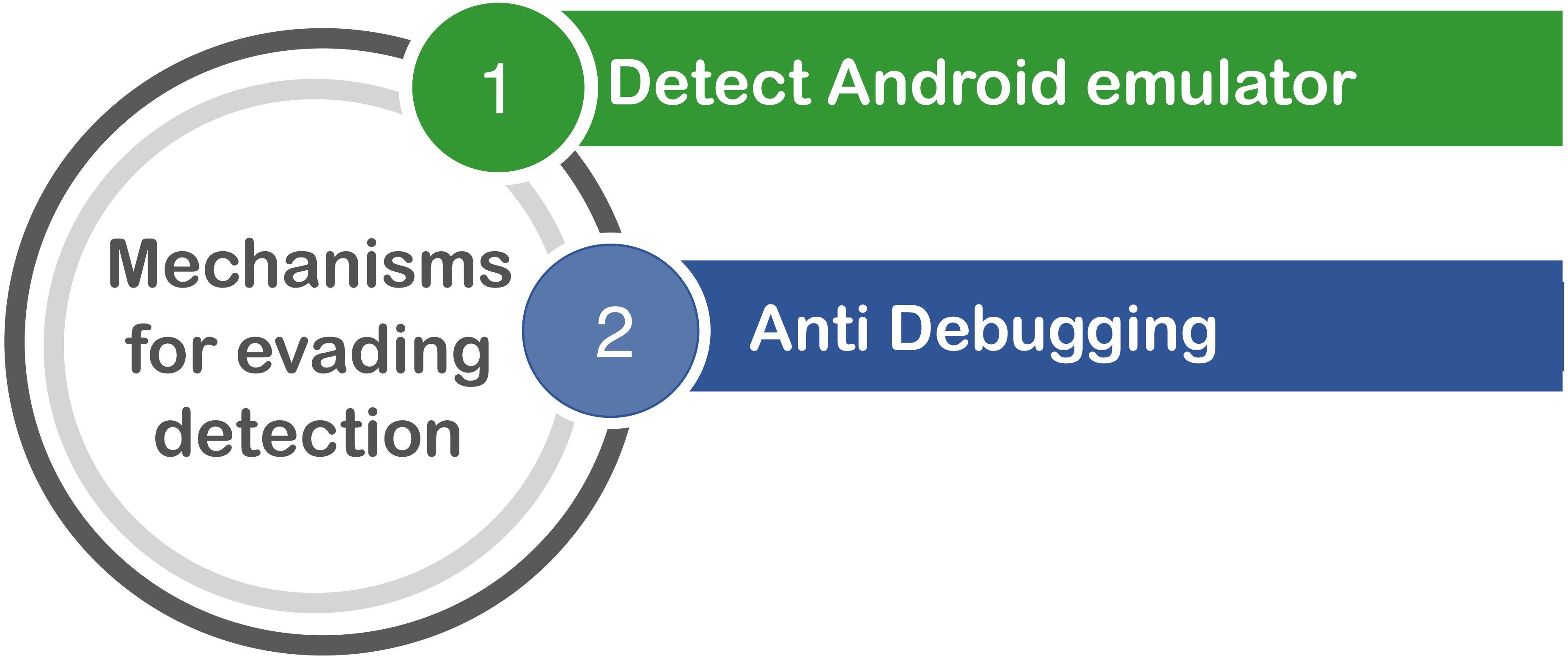
Malware



Mechanisms for evading detection

1 Detect Android emulator

Malware

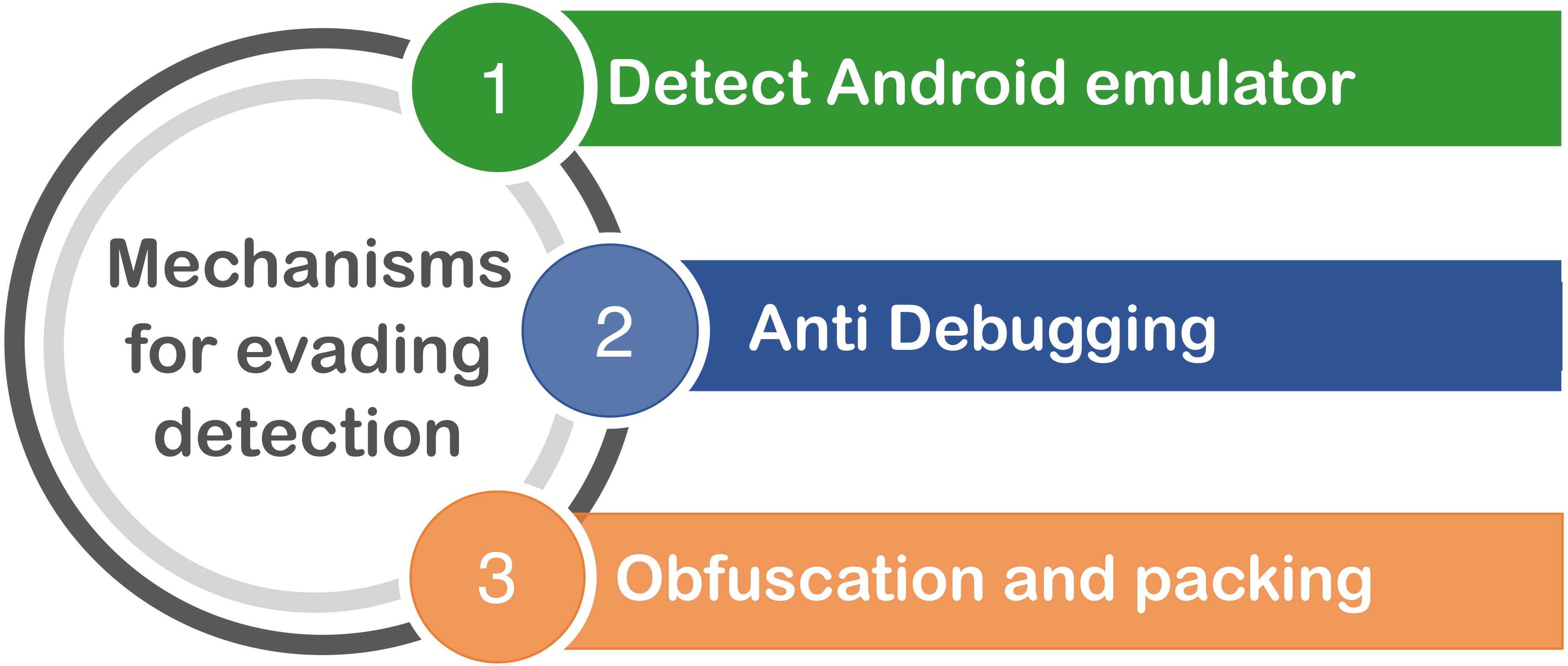


Mechanisms for evading detection

1 Detect Android emulator

2 Anti Debugging

Malware



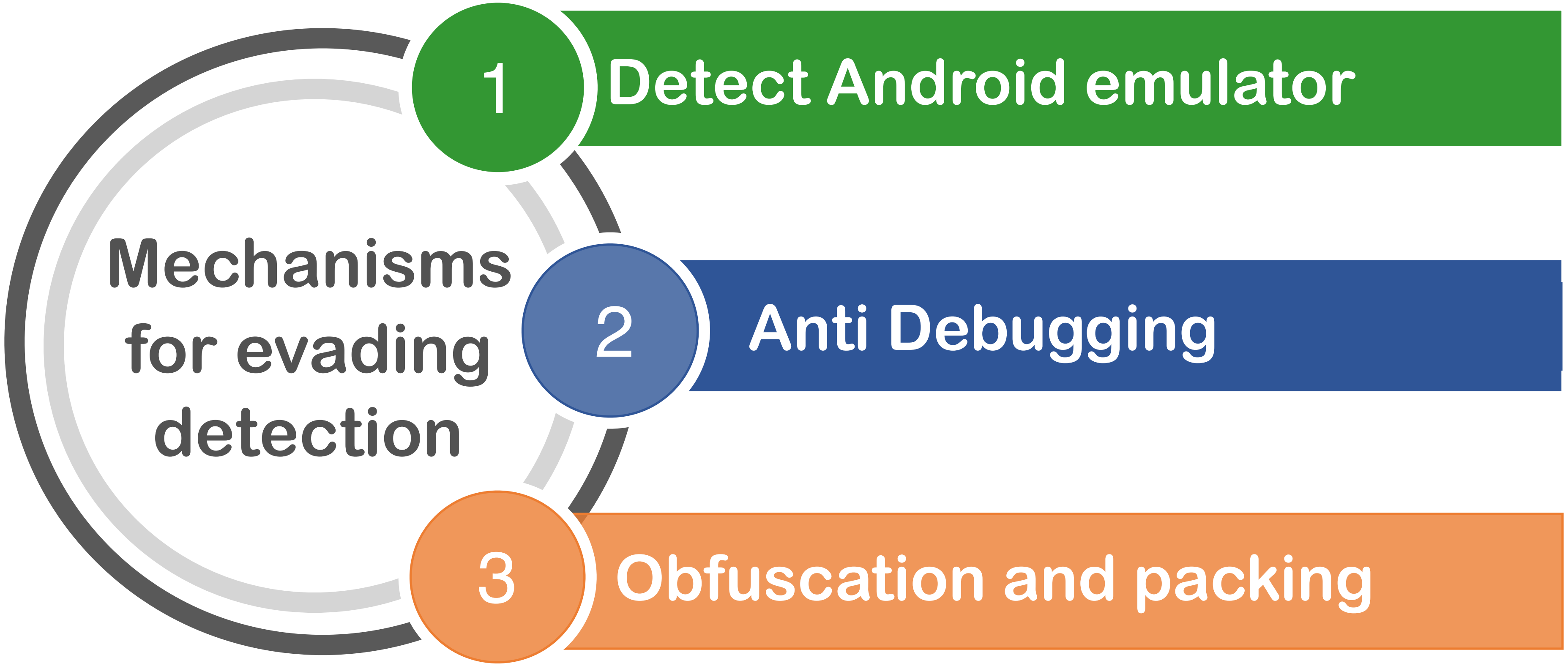
Mechanisms for evading detection

1 Detect Android emulator

2 Anti Debugging

3 Obfuscation and packing

Malware



Mechanisms for evading detection

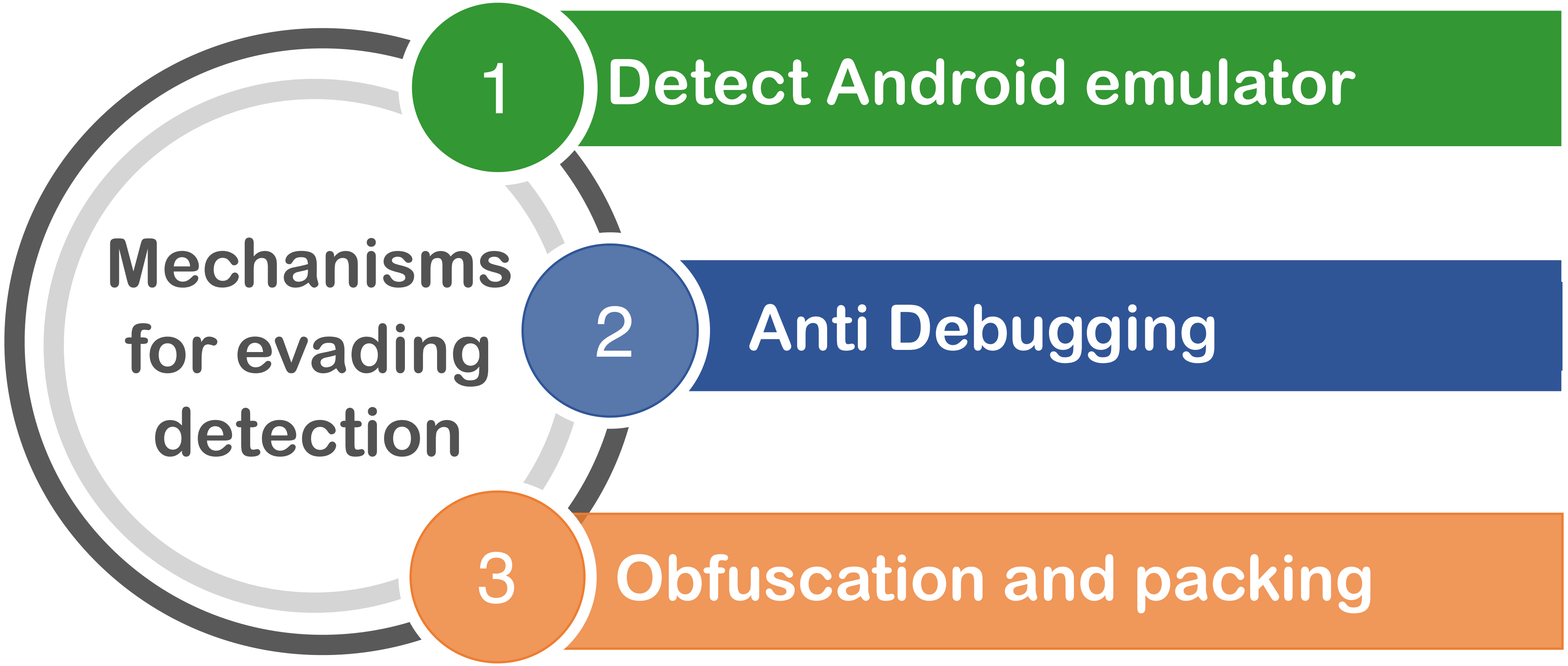
1 Detect Android emulator

2 Anti Debugging

3 Obfuscation and packing

Malton

Malware



Mechanisms for evading detection

1 Detect Android emulator

2 Anti Debugging

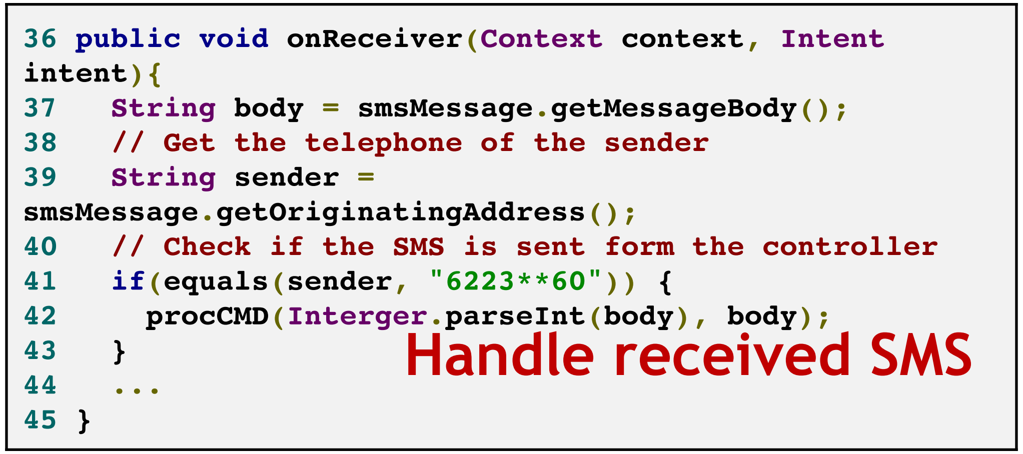
3 Obfuscation and packing

Malton On-device and Non-invasive Analysis for ART

Agenda

- ❑ Motivating Example
- ❑ The New Android Runtime (ART)
- ❑ Malton
- ❑ Evaluation
- ❑ Conclusion

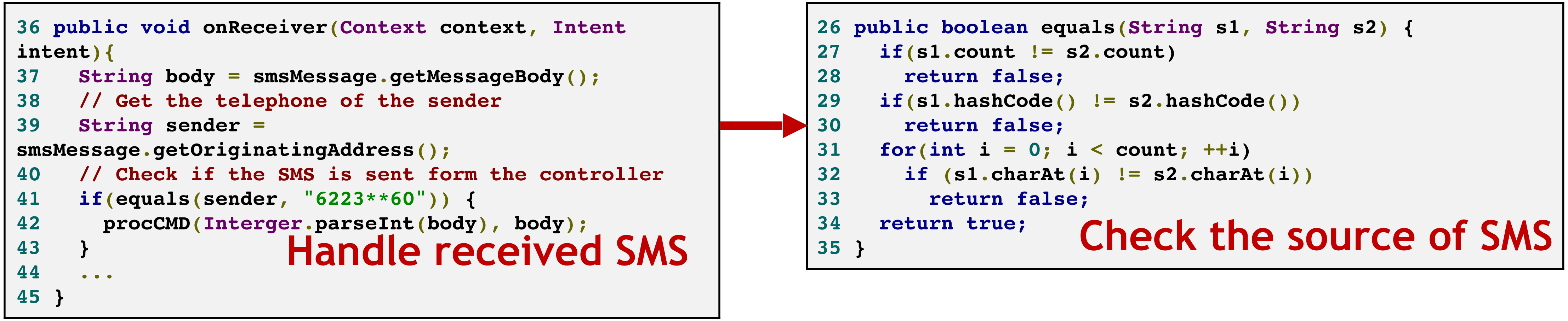
Motivating Example



```
public void onReceiver(Context context, Intent
intent){
  String body = smsMessage.getMessageBody();
  // Get the telephone of the sender
  String sender =
smsMessage.getOriginatingAddress();
  // Check if the SMS is sent form the controller
  if(equals(sender, "6223**60")) {
   procCMD(Interger.parseInt(body), body);
  }
  ...
}
```

Handle received SMS

Motivating Example



```
public void onReceiver(Context context, Intent
intent){
  String body = smsMessage.getMessageBody();
  // Get the telephone of the sender
  String sender =
smsMessage.getOriginatingAddress();
  // Check if the SMS is sent form the controller
  if(equals(sender, "6223**60")) {
    procCMD(Interger.parseInt(body), body);
  }
  ...
}
```

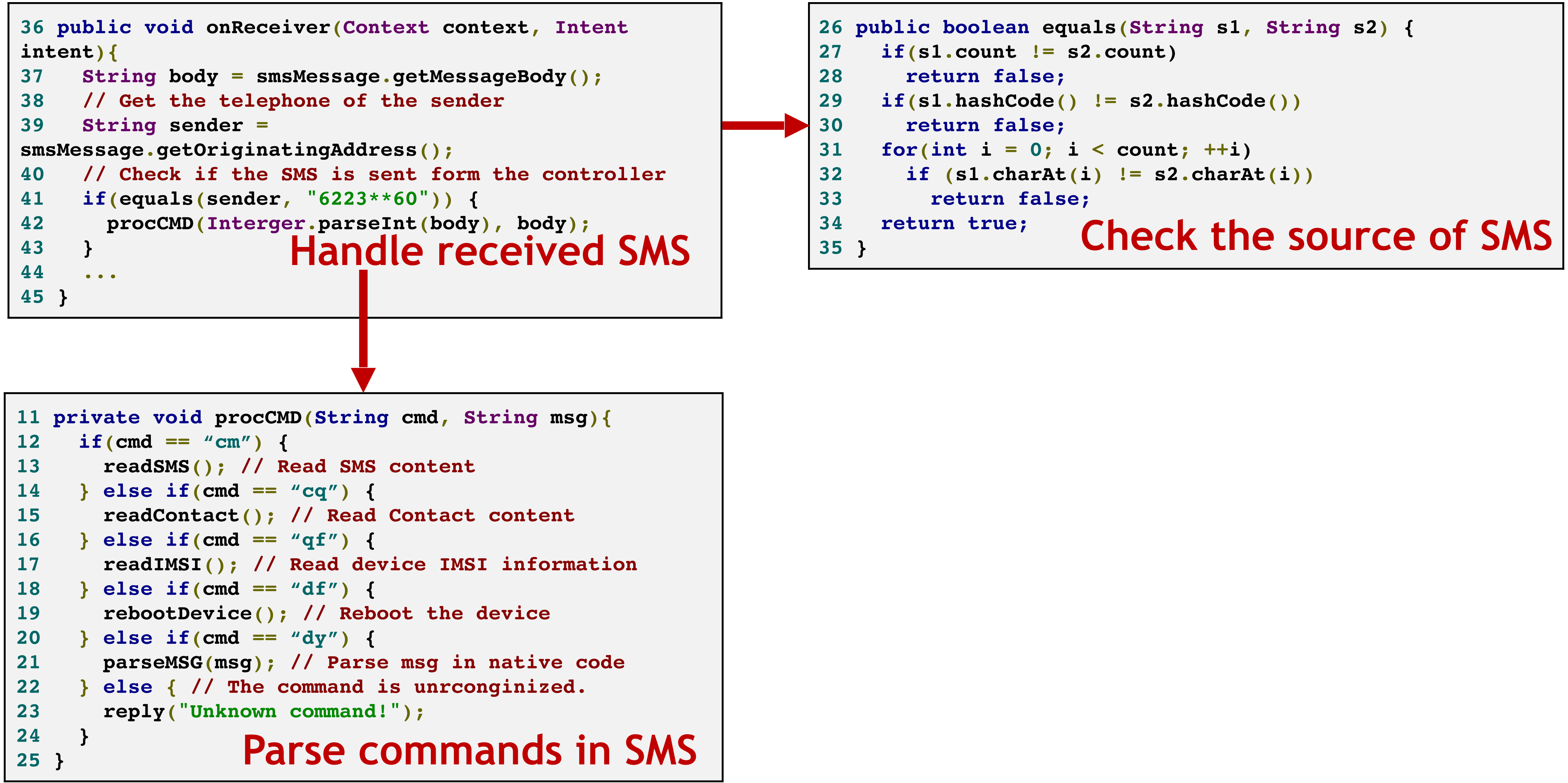
Handle received SMS

```
public boolean equals(String s1, String s2) {
  if(s1.count != s2.count)
    return false;
  if(s1.hashCode() != s2.hashCode())
    return false;
  for(int i = 0; i < count; ++i)
    if (s1.charAt(i) != s2.charAt(i))
      return false;
  return true;
}
```

Check the source of SMS

Motivating Example

Motivating Example



```
public void onReceiver(Context context, Intent intent){
  String body = smsMessage.getMessageBody();
  // Get the telephone of the sender
  String sender = smsMessage.getOriginatingAddress();
  // Check if the SMS is sent form the controller
  if(equals(sender, "6223**60")) {
    procCMD(Interger.parseInt(body), body);
  }
  ...
}
```

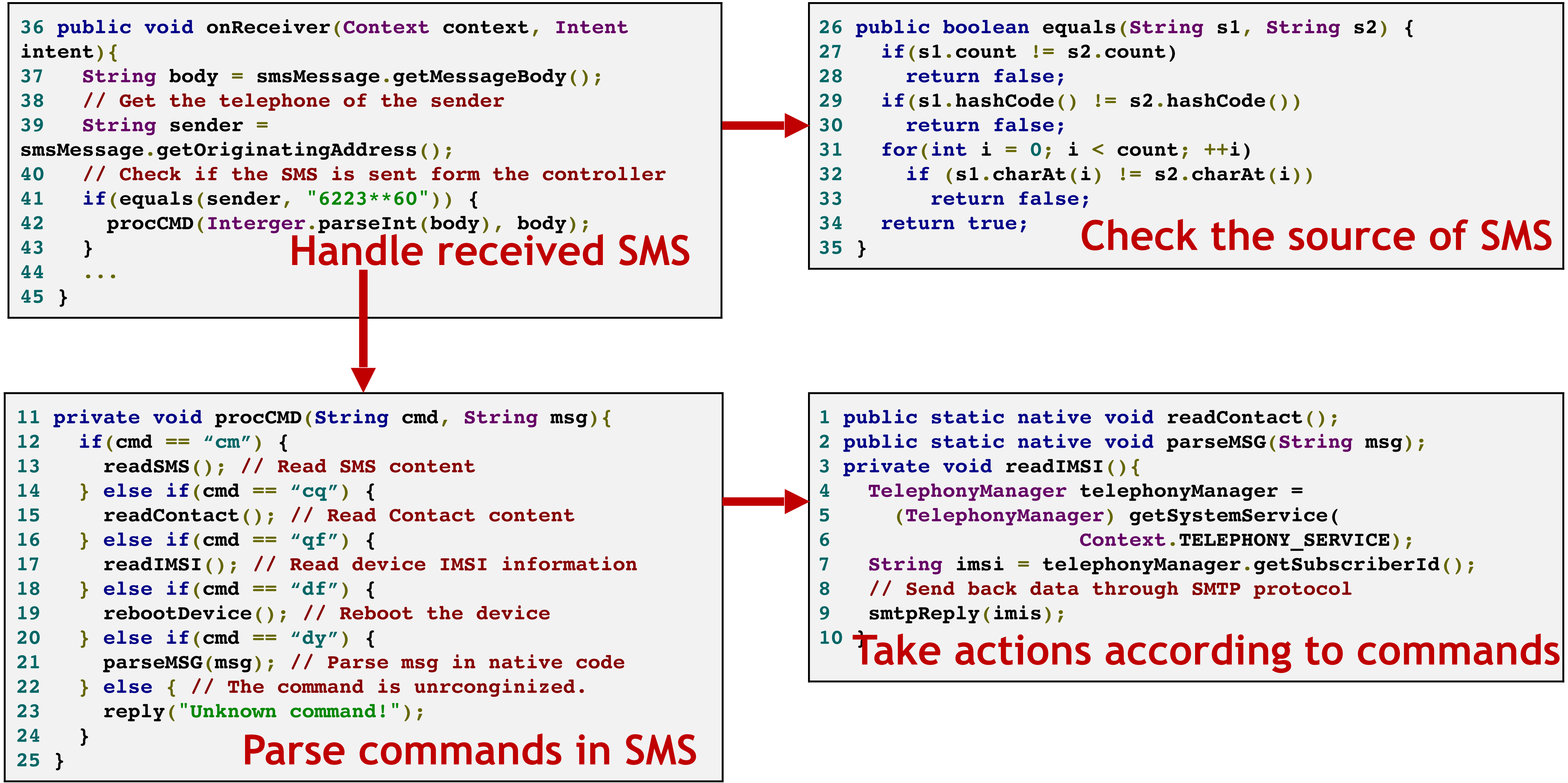
Handle received SMS

```
public boolean equals(String s1, String s2) {
  if(s1.count != s2.count)
    return false;
  if(s1.hashCode() != s2.hashCode())
    return false;
  for(int i = 0; i < count; ++i)
    if (s1.charAt(i) != s2.charAt(i))
      return false;
  return true;
}
```

Check the source of SMS

```
private void procCMD(String cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Parse commands in SMS



```
public void onReceiver(Context context, Intent
intent){
  String body = smsMessage.getMessageBody();
  // Get the telephone of the sender
  String sender =
smsMessage.getOriginatingAddress();
  // Check if the SMS is sent form the controller
  if(equals(sender, "6223**60")) {
    procCMD(Interger.parseInt(body), body);
  }
  ...
}
```

Handle received SMS

```
public boolean equals(String s1, String s2) {
  if(s1.count != s2.count)
    return false;
  if(s1.hashCode() != s2.hashCode())
    return false;
  for(int i = 0; i < count; ++i)
    if (s1.charAt(i) != s2.charAt(i))
      return false;
  return true;
}
```

Check the source of SMS

```
private void procCMD(String cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

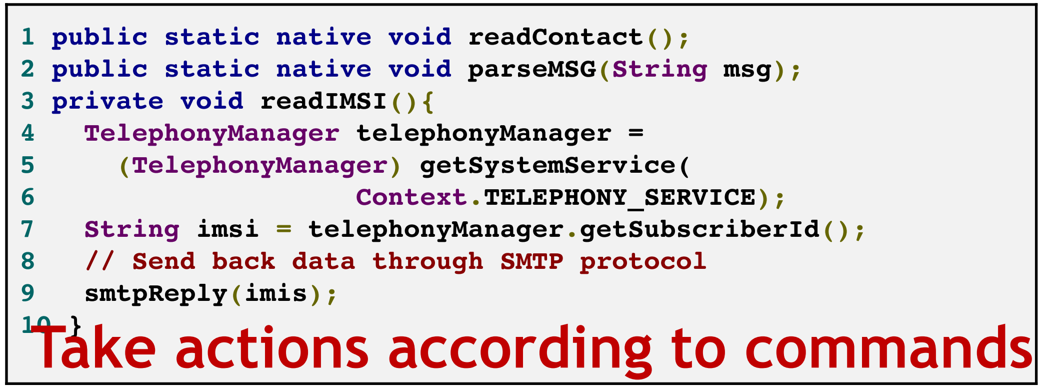
Parse commands in SMS

```
public static native void readContact();
public static native void parseMSG(String msg);
private void readIMSI(){
  TelephonyManager telephonyManager =
    (TelephonyManager) getSystemService(
                  Context.TELEPHONY_SERVICE);
  String imsi = telephonyManager.getSubscriberId();
  // Send back data through SMTP protocol
  smtpReply(imis);
}
```

Take actions according to commands

Motivating Example

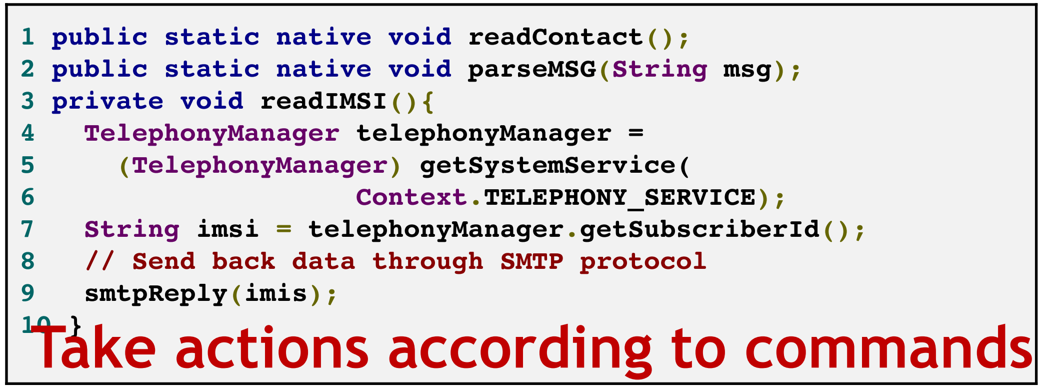
Challenges



```
public static native void readContact();
public static native void parseMSG(String msg);
private void readIMSI(){
  TelephonyManager telephonyManager =
    (TelephonyManager) getSystemService(
                     Context.TELEPHONY_SERVICE);
  String imsi = telephonyManager.getSubscriberId();
  // Send back data through SMTP protocol
  smtpReply(imis);
}
```

Take actions according to commands

Challenges

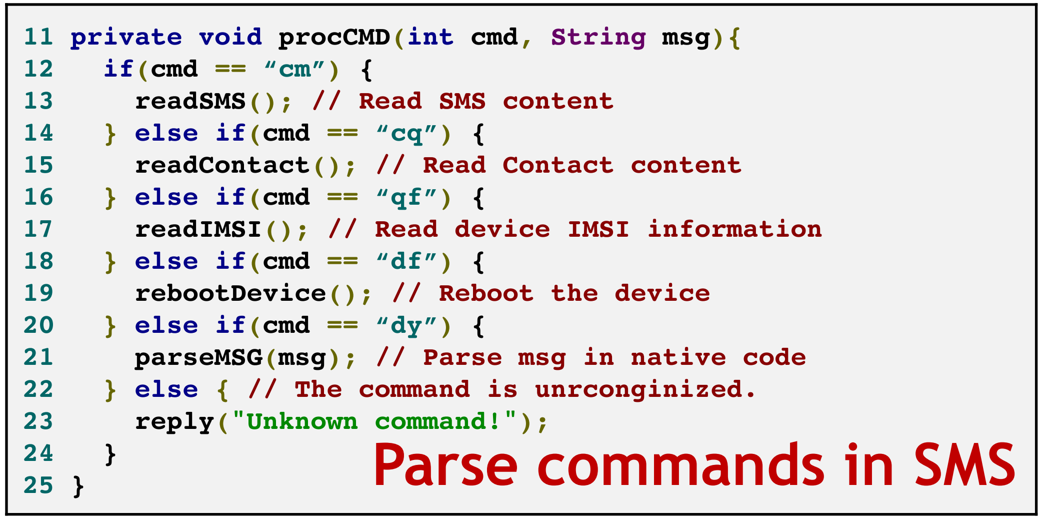


```
public static native void readContact();
public static native void parseMSG(String msg);
private void readIMSI(){
  TelephonyManager telephonyManager =
    (TelephonyManager) getSystemService(
                     Context.TELEPHONY_SERVICE);
  String imsi = telephonyManager.getSubscriberId();
  // Send back data through SMTP protocol
  smtpReply(imis);
}
```

Take actions according to commands

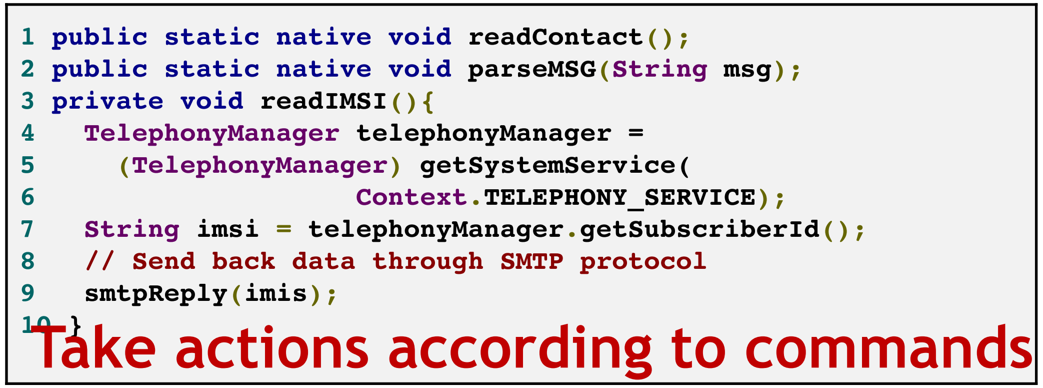
Identify the cross-layer information flow

Challenges



```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Parse commands in SMS

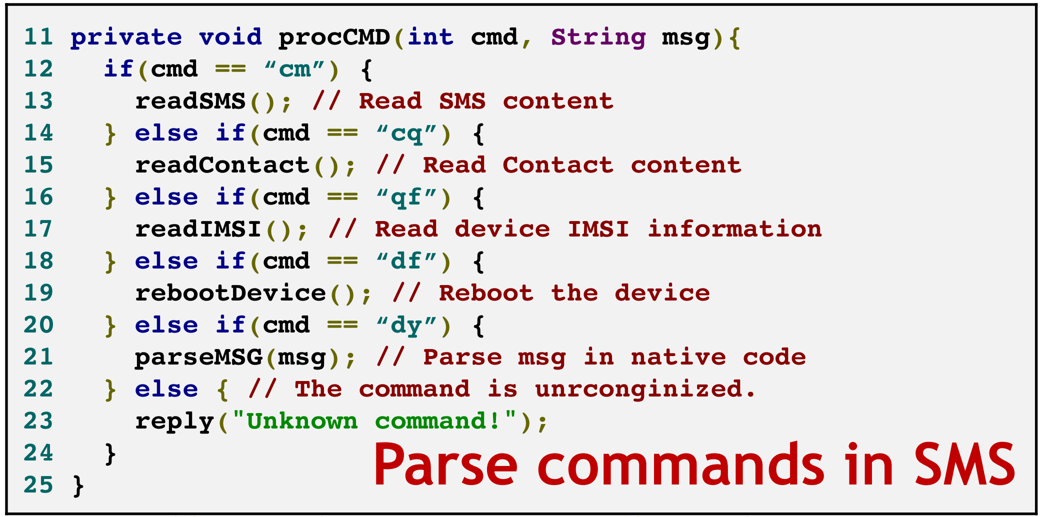


```
public static native void readContact();
public static native void parseMSG(String msg);
private void readIMSI(){
  TelephonyManager telephonyManager =
    (TelephonyManager) getSystemService(
                   Context.TELEPHONY_SERVICE);
  String imsi = telephonyManager.getSubscriberId();
  // Send back data through SMTP protocol
  smtpReply(imis);
}
```

Take actions according to commands

Identify the cross-layer information flow

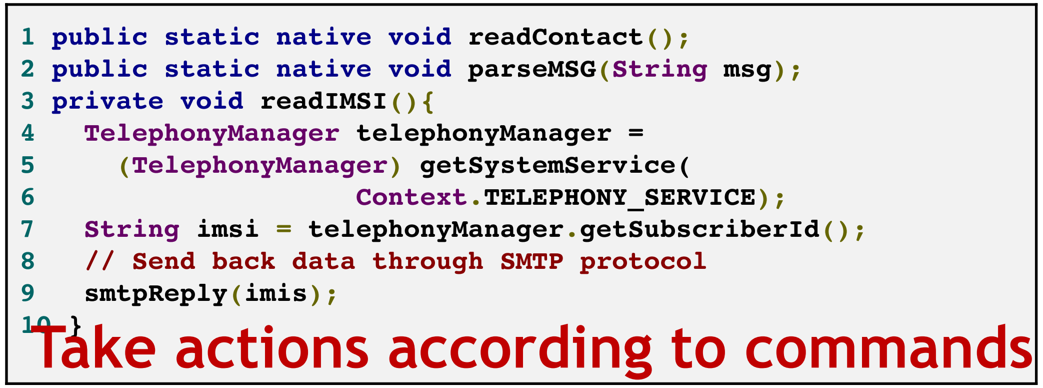
Challenges



```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Parse commands in SMS

Expose all malicious payloads triggered by various commands efficiently

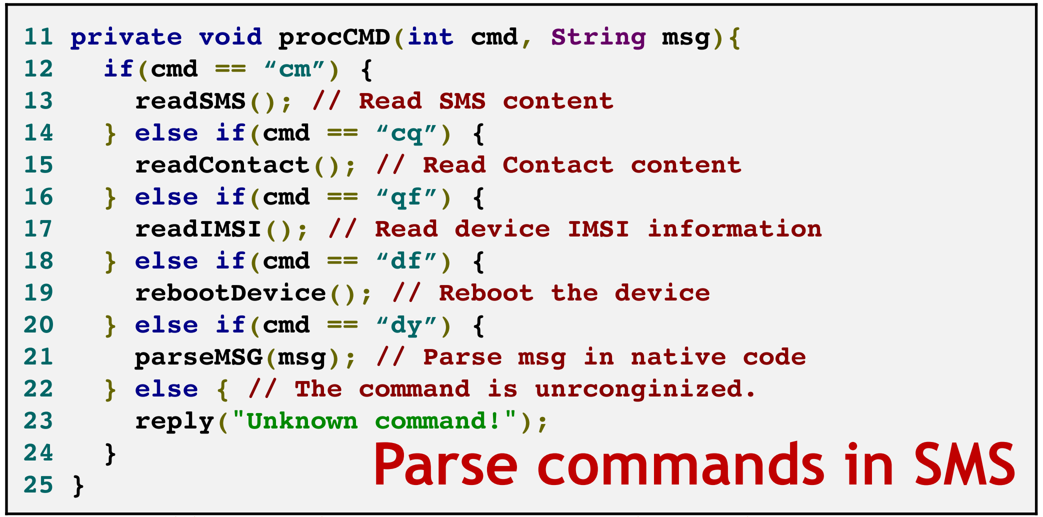


```
public static native void readContact();
public static native void parseMSG(String msg);
private void readIMSI(){
  TelephonyManager telephonyManager =
    (TelephonyManager) getSystemService(
                     Context.TELEPHONY_SERVICE);
  String imsi = telephonyManager.getSubscriberId();
  // Send back data through SMTP protocol
  smtpReply(imis);
}
```

Take actions according to commands

Identify the cross-layer information flow

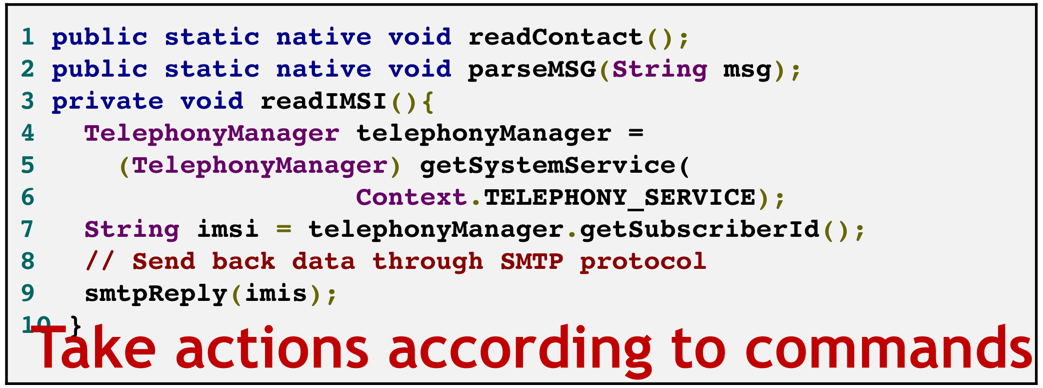
Challenges



```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Parse commands in SMS

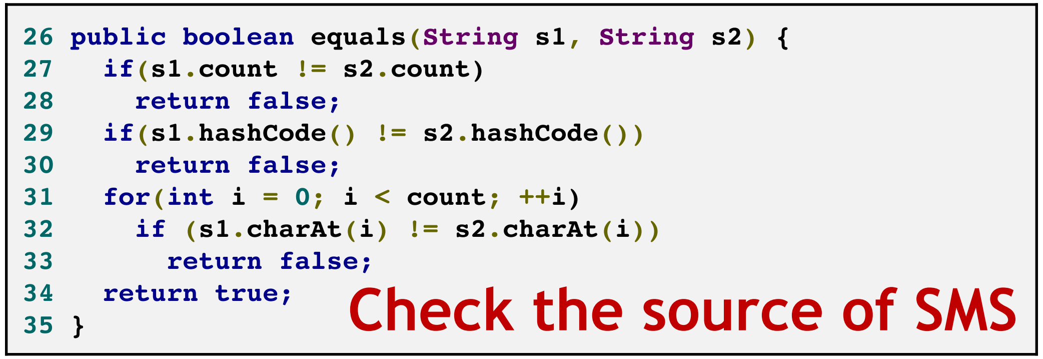
Expose all malicious payloads triggered by various commands efficiently



```
public static native void readContact();
public static native void parseMSG(String msg);
private void readIMSI(){
  TelephonyManager telephonyManager =
    (TelephonyManager) getSystemService(
                  Context.TELEPHONY_SERVICE);
  String imsi = telephonyManager.getSubscriberId();
  // Send back data through SMTP protocol
  smtpReply(imis);
}
```

Take actions according to commands

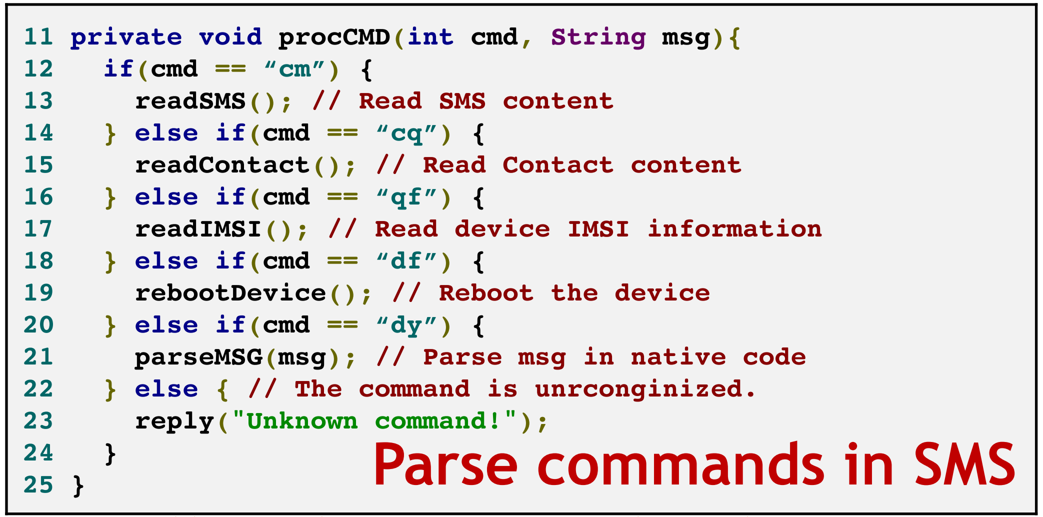
Identify the cross-layer information flow



```
public boolean equals(String s1, String s2) {
  if(s1.count != s2.count)
    return false;
  if(s1.hashCode() != s2.hashCode())
    return false;
  for(int i = 0; i < count; ++i)
    if (s1.charAt(i) != s2.charAt(i))
      return false;
  return true;
}
```

Check the source of SMS

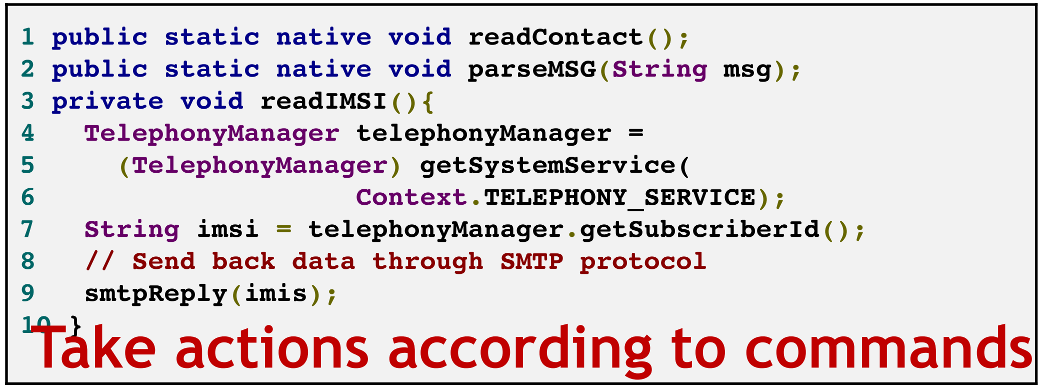
Challenges



```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Parse commands in SMS

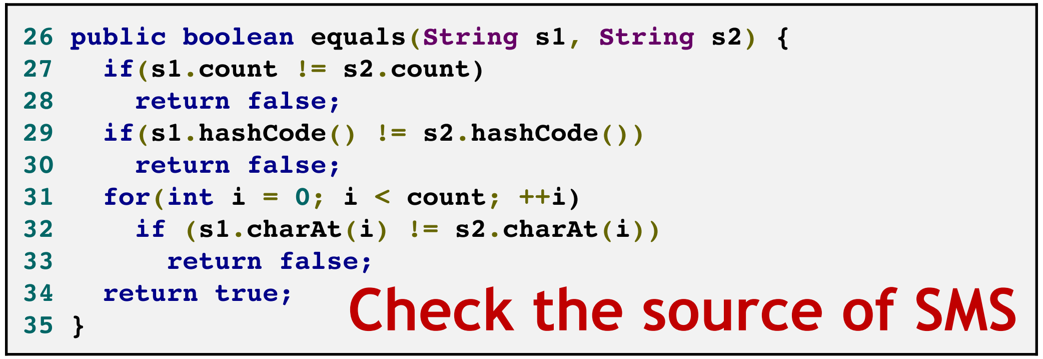
Expose all malicious payloads triggered by various commands efficiently



```
public static native void readContact();
public static native void parseMSG(String msg);
private void readIMSI(){
  TelephonyManager telephonyManager =
    (TelephonyManager) getSystemService(
                   Context.TELEPHONY_SERVICE);
  String imsi = telephonyManager.getSubscriberId();
  // Send back data through SMTP protocol
  smtpReply(imis);
}
```

Take actions according to commands

Identify the cross-layer information flow



```
public boolean equals(String s1, String s2) {
  if(s1.count != s2.count)
    return false;
  if(s1.hashCode() != s2.hashCode())
    return false;
  for(int i = 0; i < count; ++i)
    if (s1.charAt(i) != s2.charAt(i))
      return false;
  return true;
}
```

Check the source of SMS

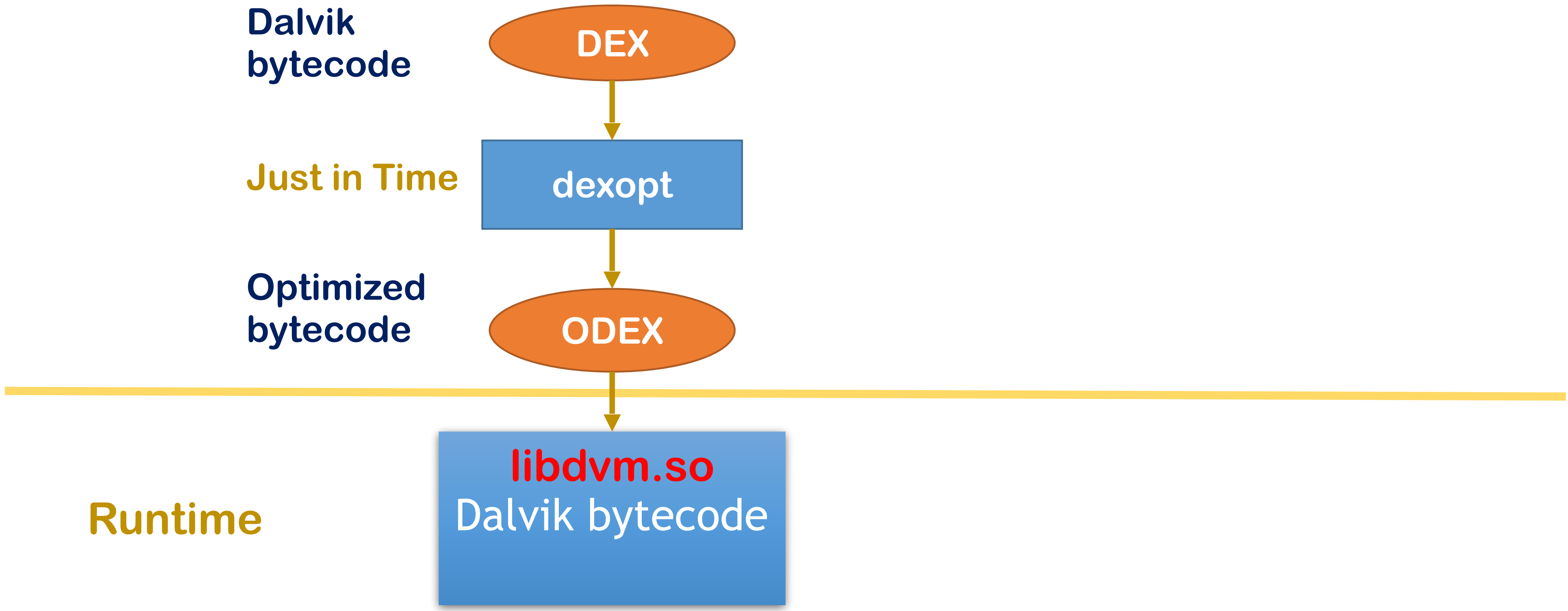
Force app to execute a certain path if the desired input cannot be generated.

Challenges

Agenda

- ❑ Motivating Example
- ❑ The New Android Runtime (ART)
- ❑ Malton
- ❑ Evaluation
- ❑ Conclusion

Android Runtime



Dalvik bytecode

DEX

Just in Time

dexopt

Optimized bytecode

ODEX

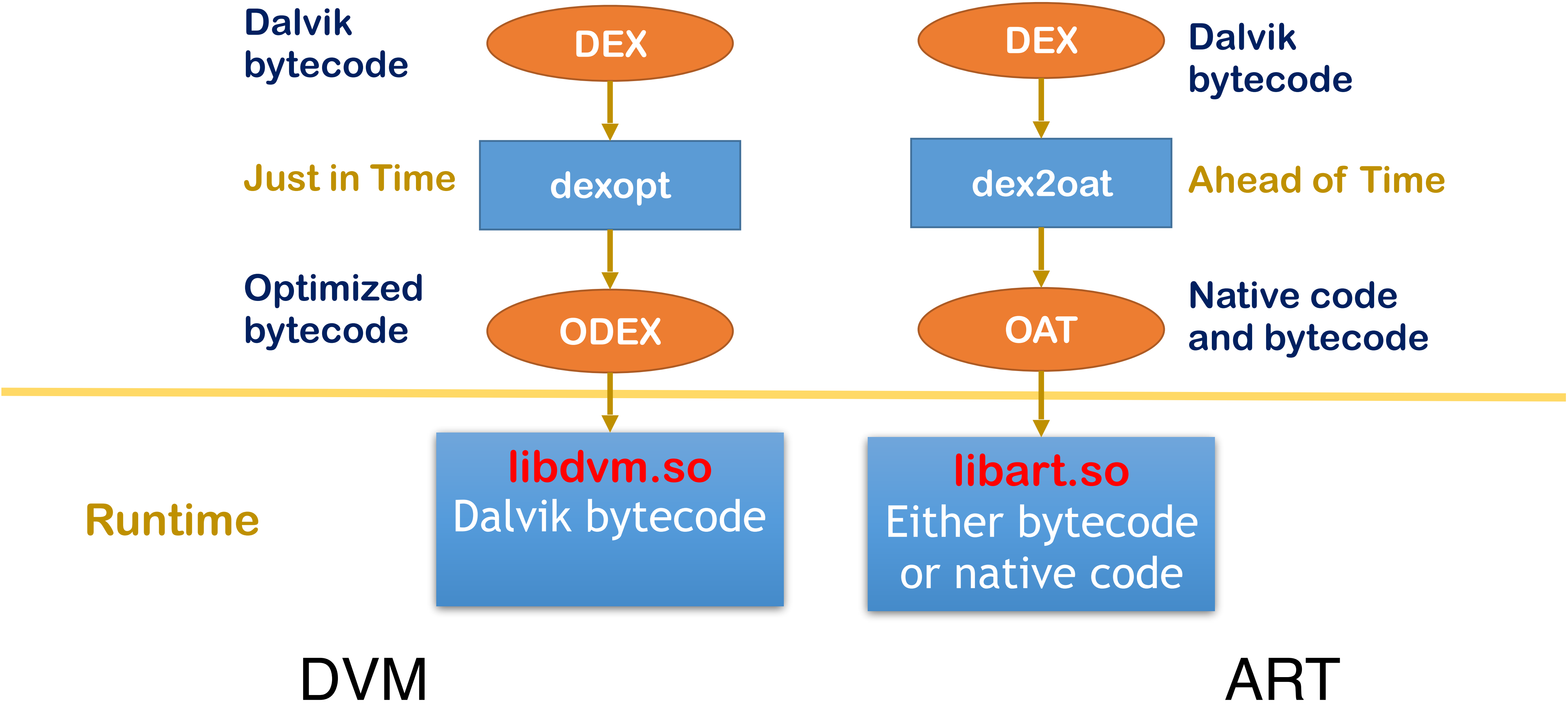
Runtime

libdvm.so

Dalvik bytecode

DVM

Android Runtime



Dalvik bytecode

DEX

Just in Time

dexopt

Optimized bytecode

ODEX

Runtime

libdvm.so
Dalvik bytecode

DVM

DEX

Dalvik bytecode

dex2oat

Ahead of Time

OAT

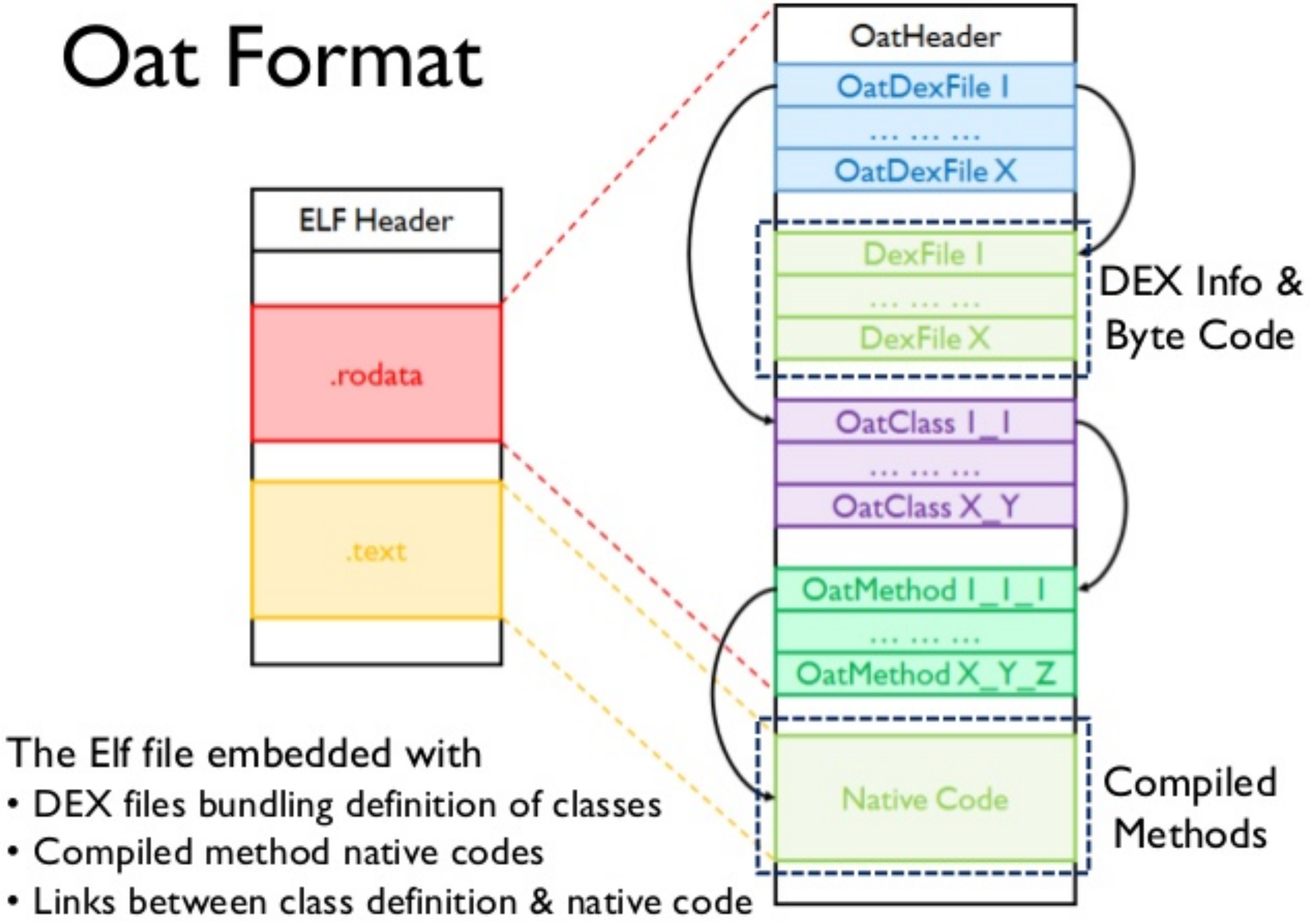
Native code and bytecode

libart.so
Either bytecode or native code

ART

Android Runtime

The OAT File



Oat Format

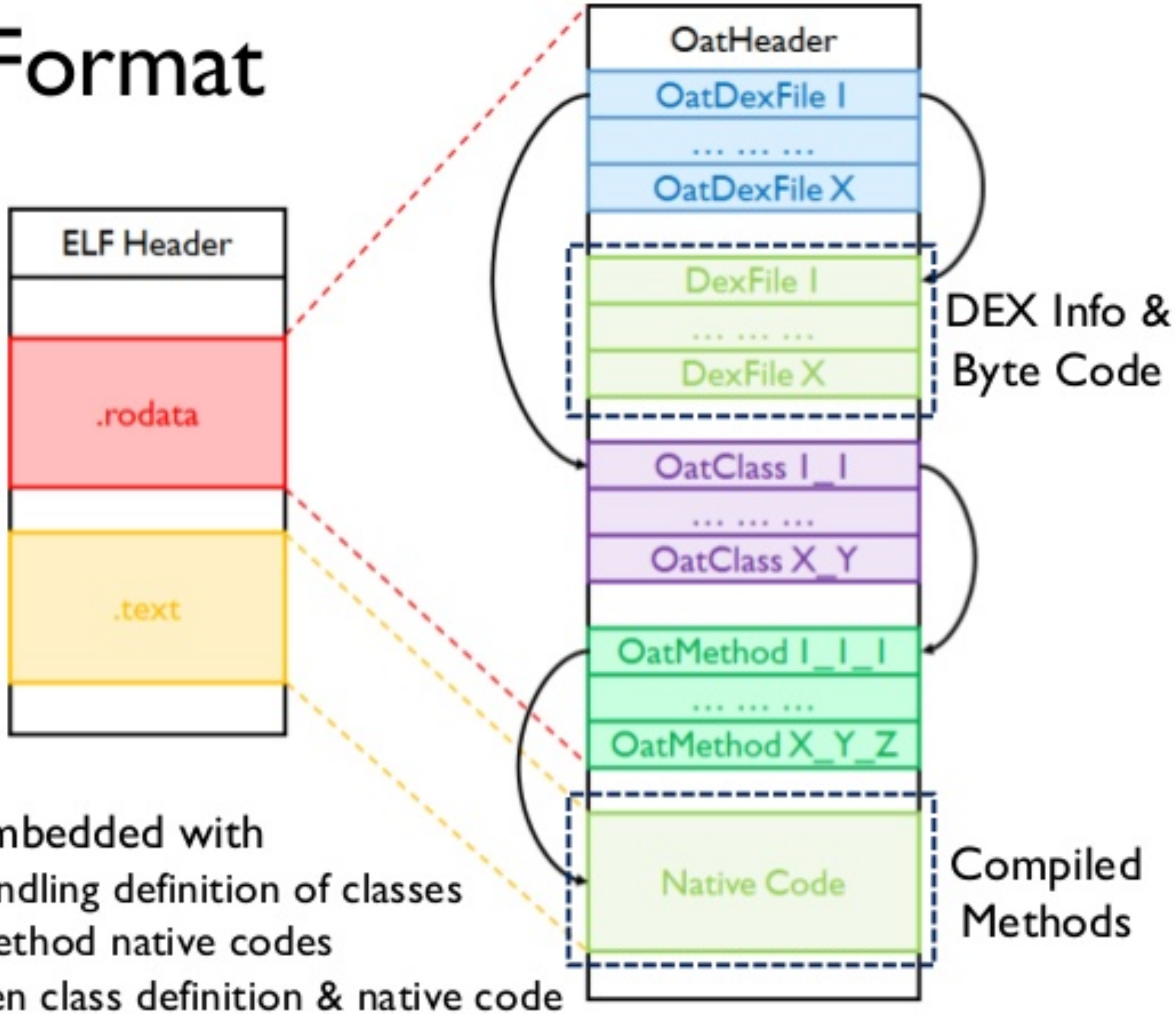
The Elf file embedded with

- DEX files bundling definition of classes
- Compiled method native codes
- Links between class definition & native code

The OAT File

Parse OAT files

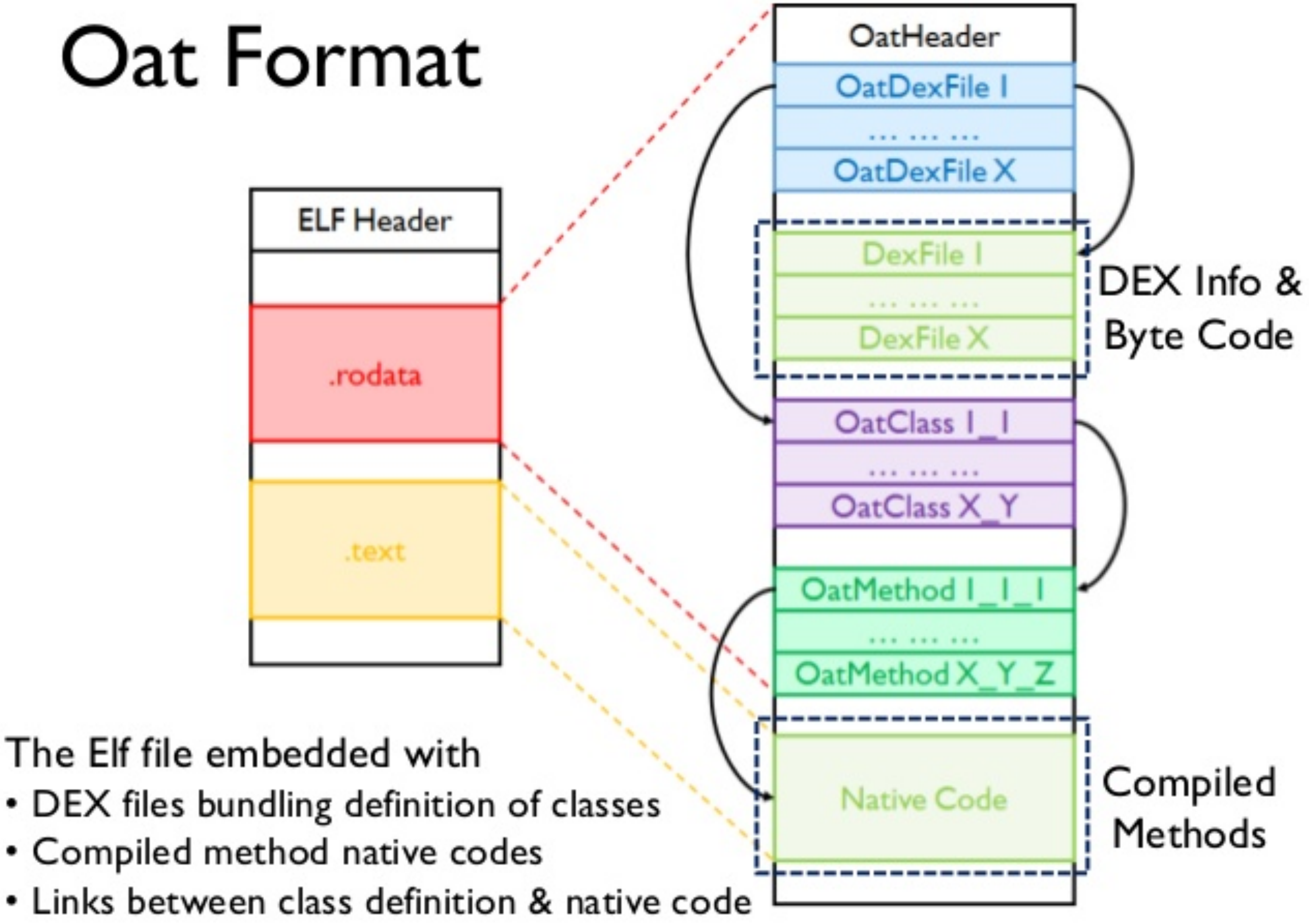
Oat Format



The Elf file embedded with

- DEX files bundling definition of classes
- Compiled method native codes
- Links between class definition & native code

The OAT File



Oat Format

The Elf file embedded with

- DEX files bundling definition of classes
- Compiled method native codes
- Links between class definition & native code

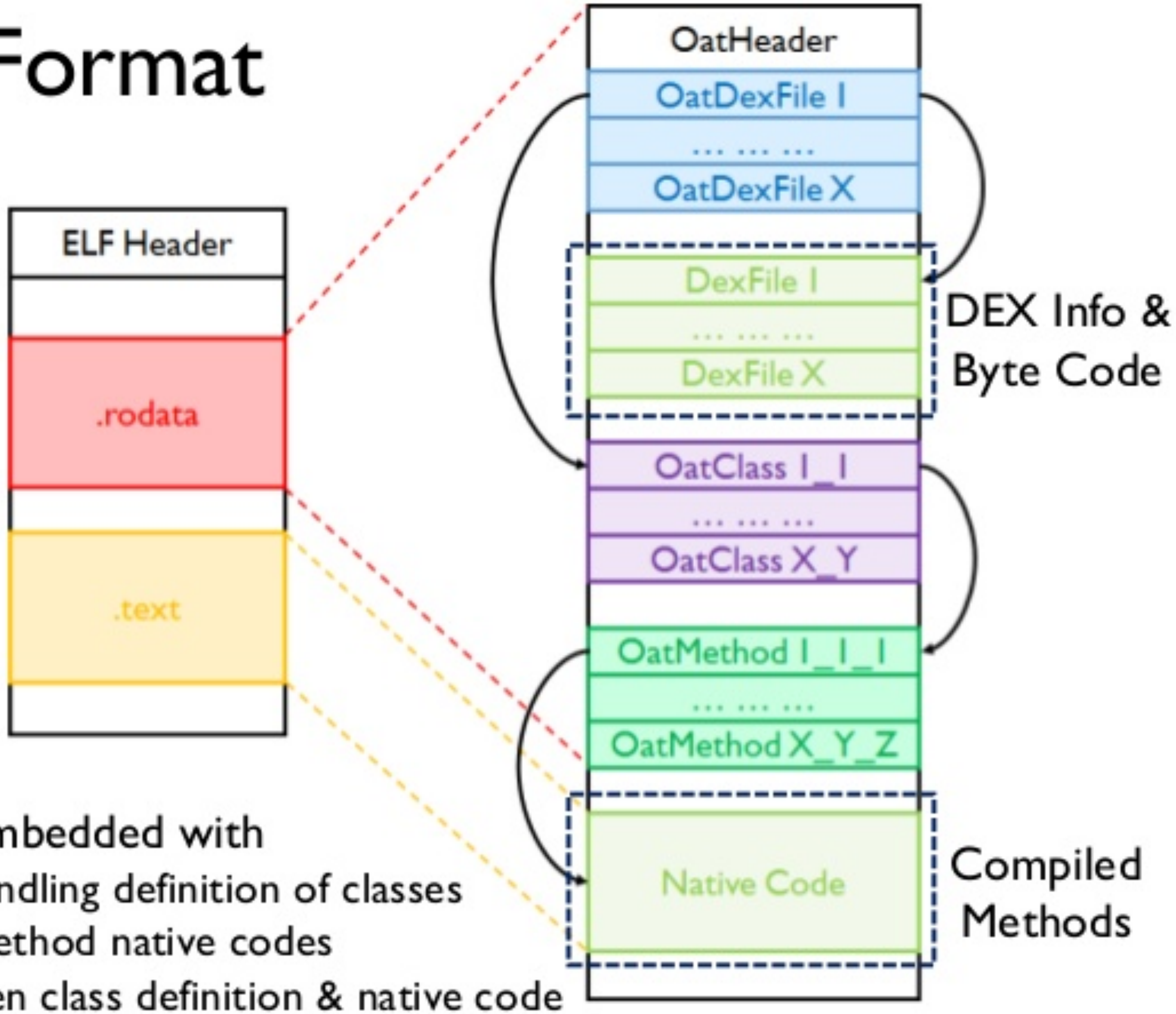
Parse OAT files

↓

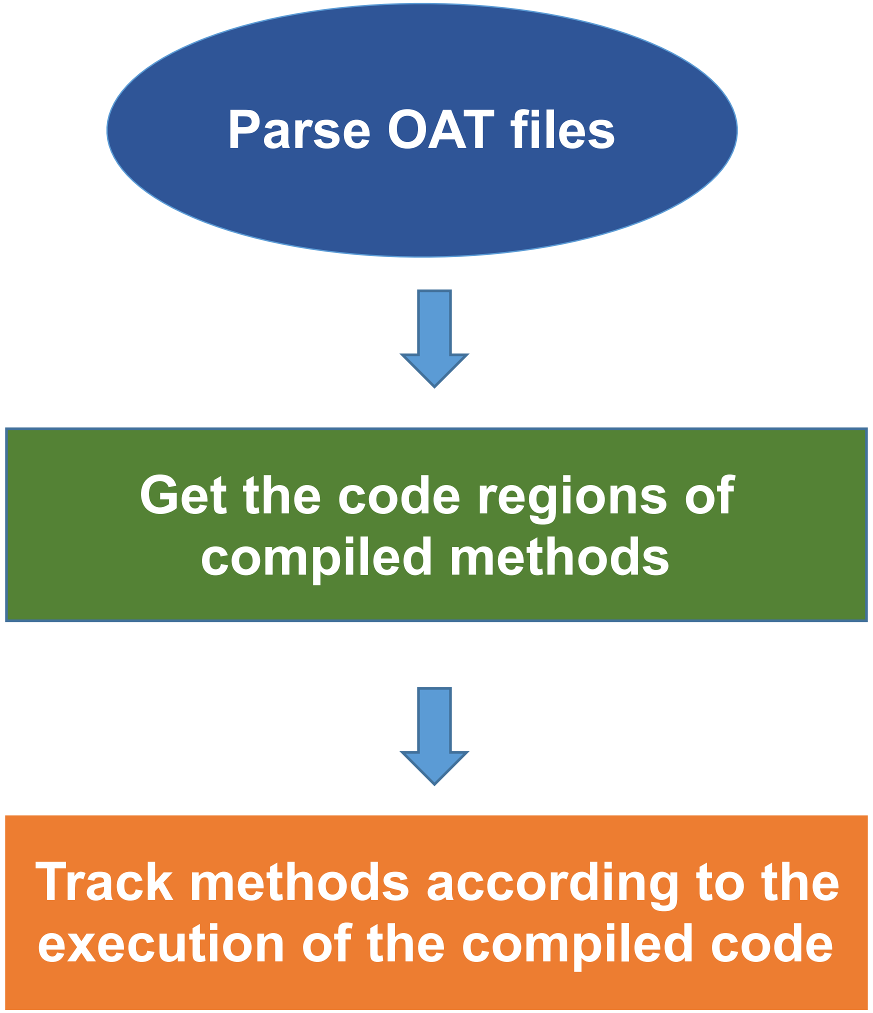
Get the code regions of compiled methods

The OAT File

Oat Format



The Elf file embedded with
- DEX files bundling definition of classes
- Compiled method native codes
- Links between class definition & native code



Parse OAT files

Get the code regions of compiled methods

Track methods according to the execution of the compiled code

Agenda

❑ Motivating Example

❑ The New Android Runtime (ART)

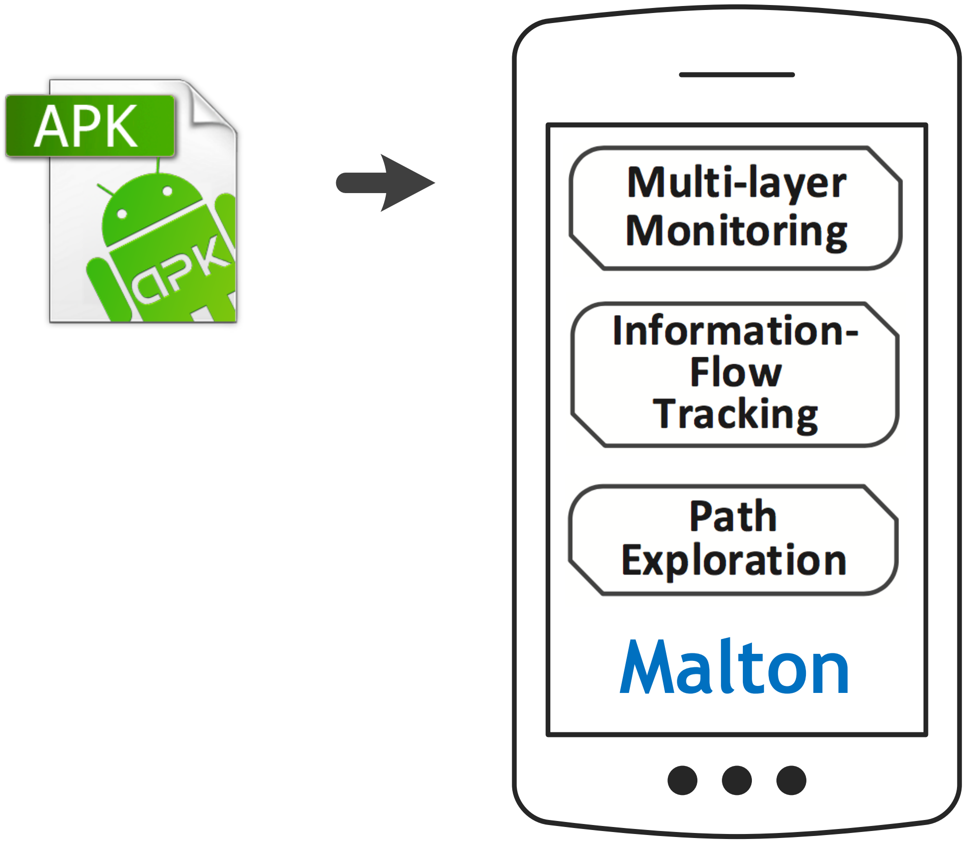
❑ **Malton**

❑ Evaluation

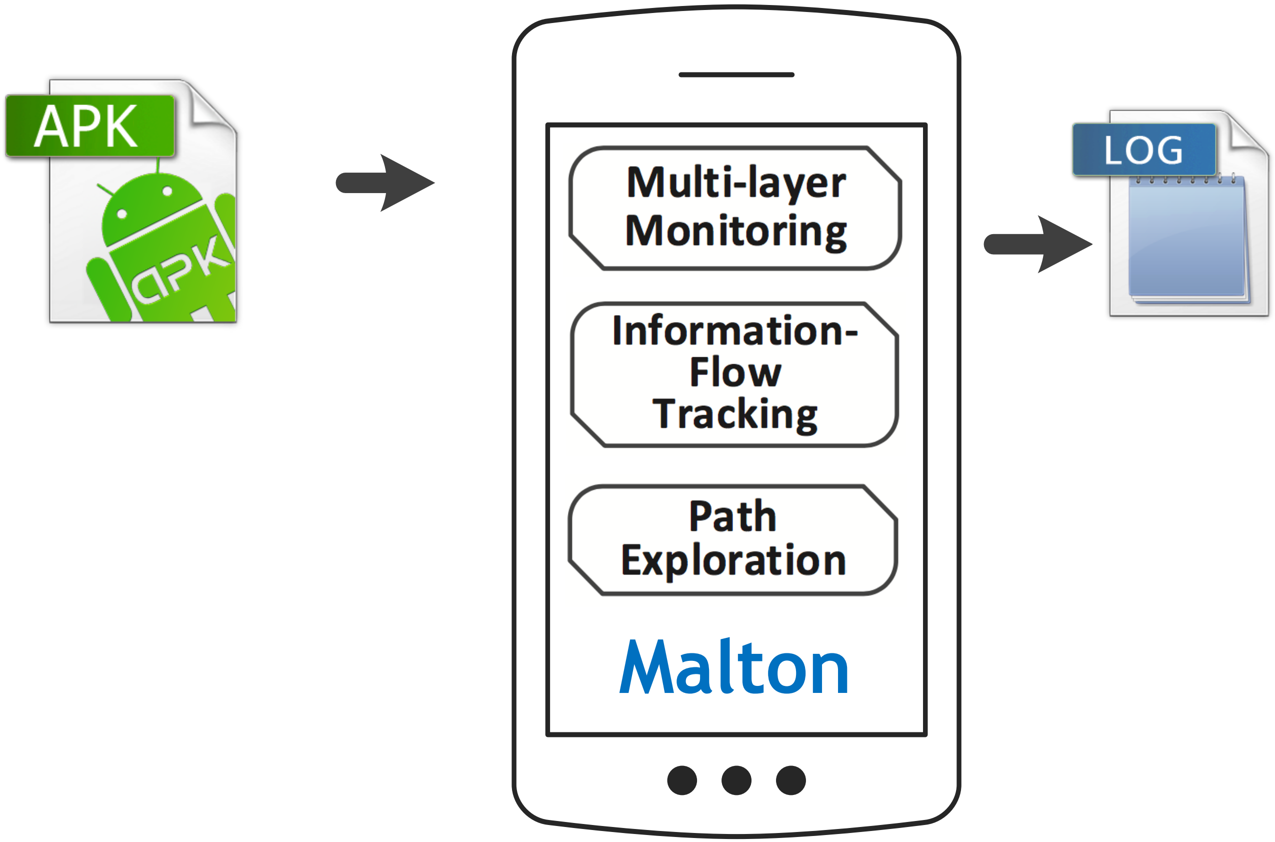
❑ Conclusion

APK
APK

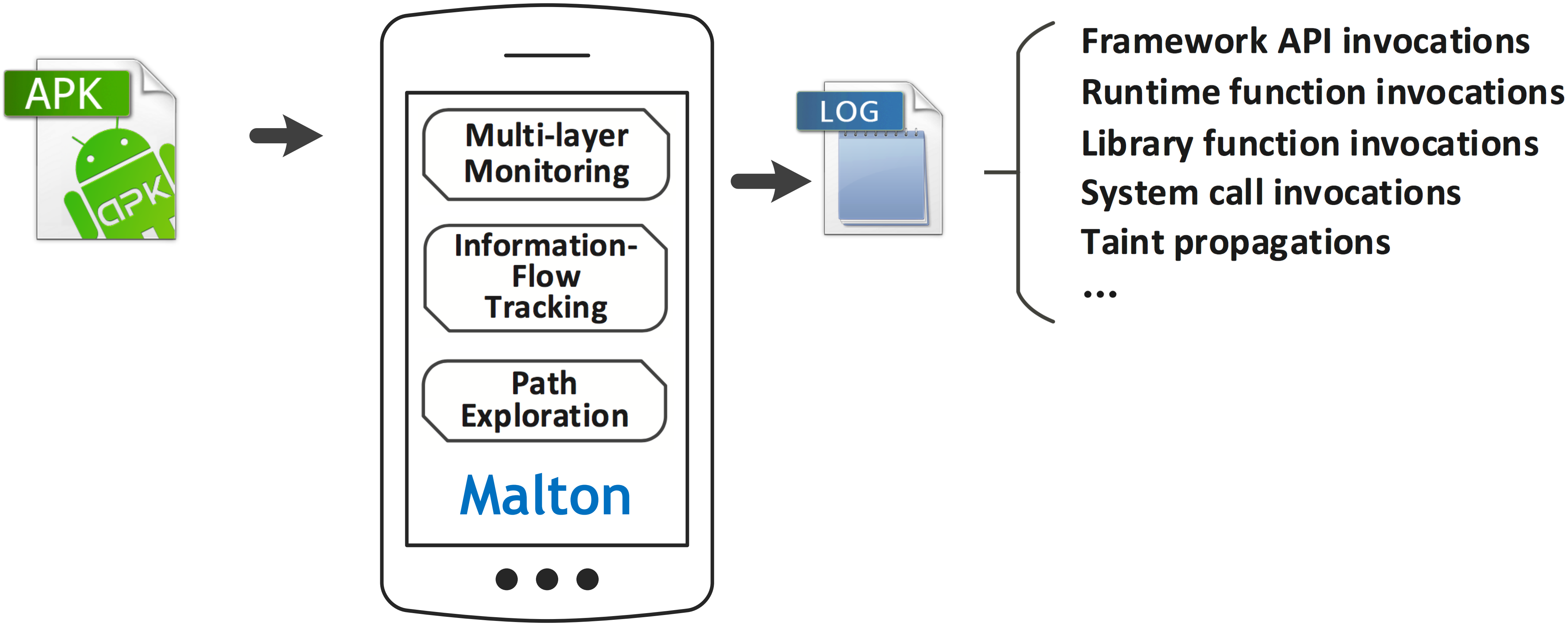
- **Running on a real device;**
- **Conducting cross-layer monitoring and information flow tracking;**
- **Doesn’t need to modify the app.**



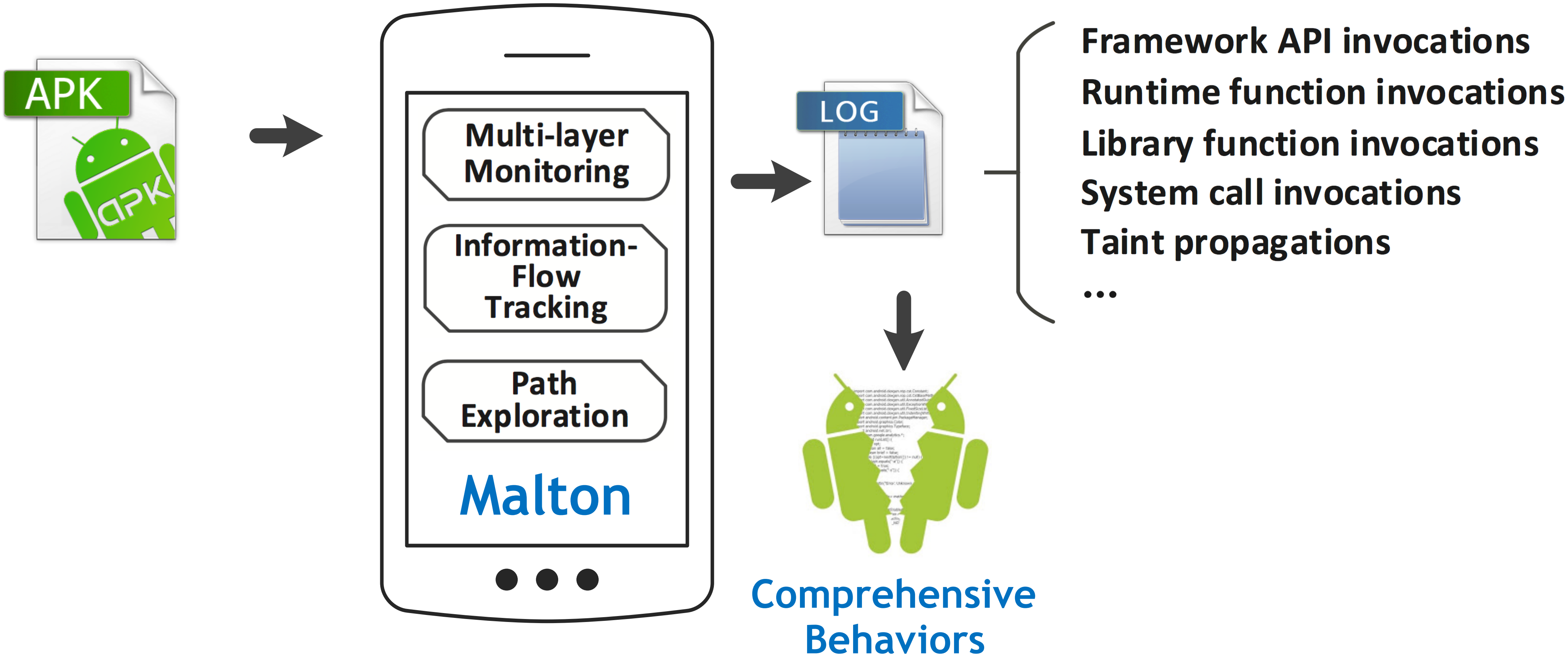
- Running on a real device;
- Conducting cross-layer monitoring and information flow tracking;
- Doesn’t need to modify the app.



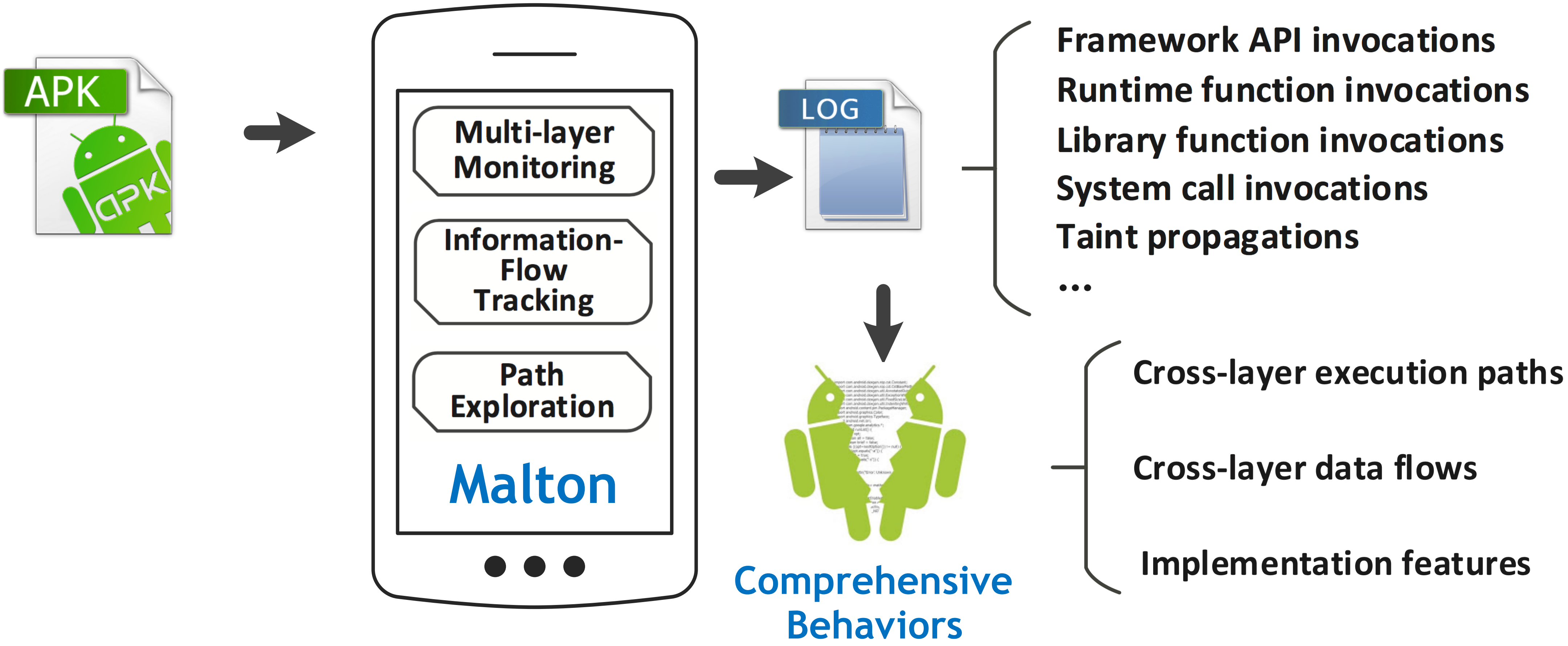
- Running on a real device;
- Conducting cross-layer monitoring and information flow tracking;
- Doesn’t need to modify the app.


APK
Multi-layer Monitoring
Information-Flow Tracking
Path Exploration
Malton
LOG
Framework API invocations
Runtime function invocations
Library function invocations
System call invocations
Taint propagations
…

- Running on a real device;
- Conducting cross-layer monitoring and information flow tracking;
- Doesn’t need to modify the app.

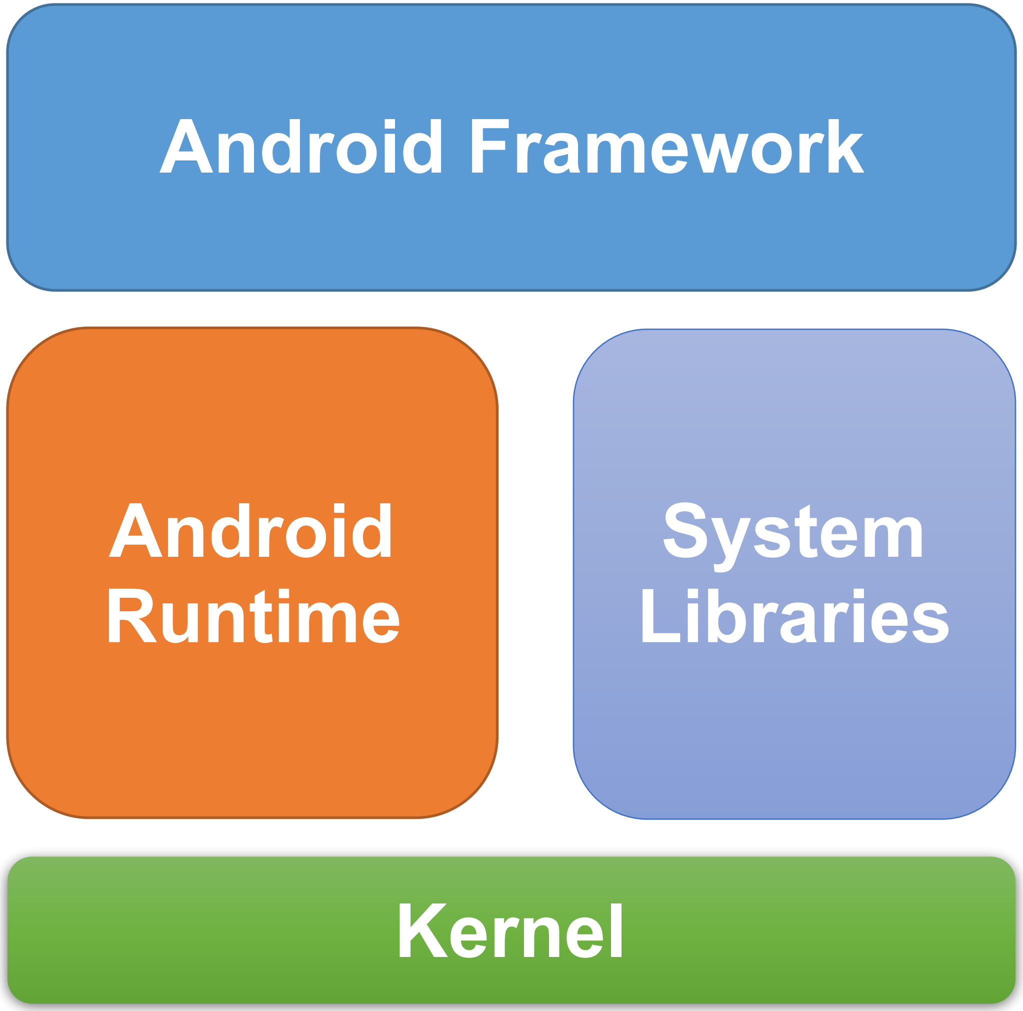


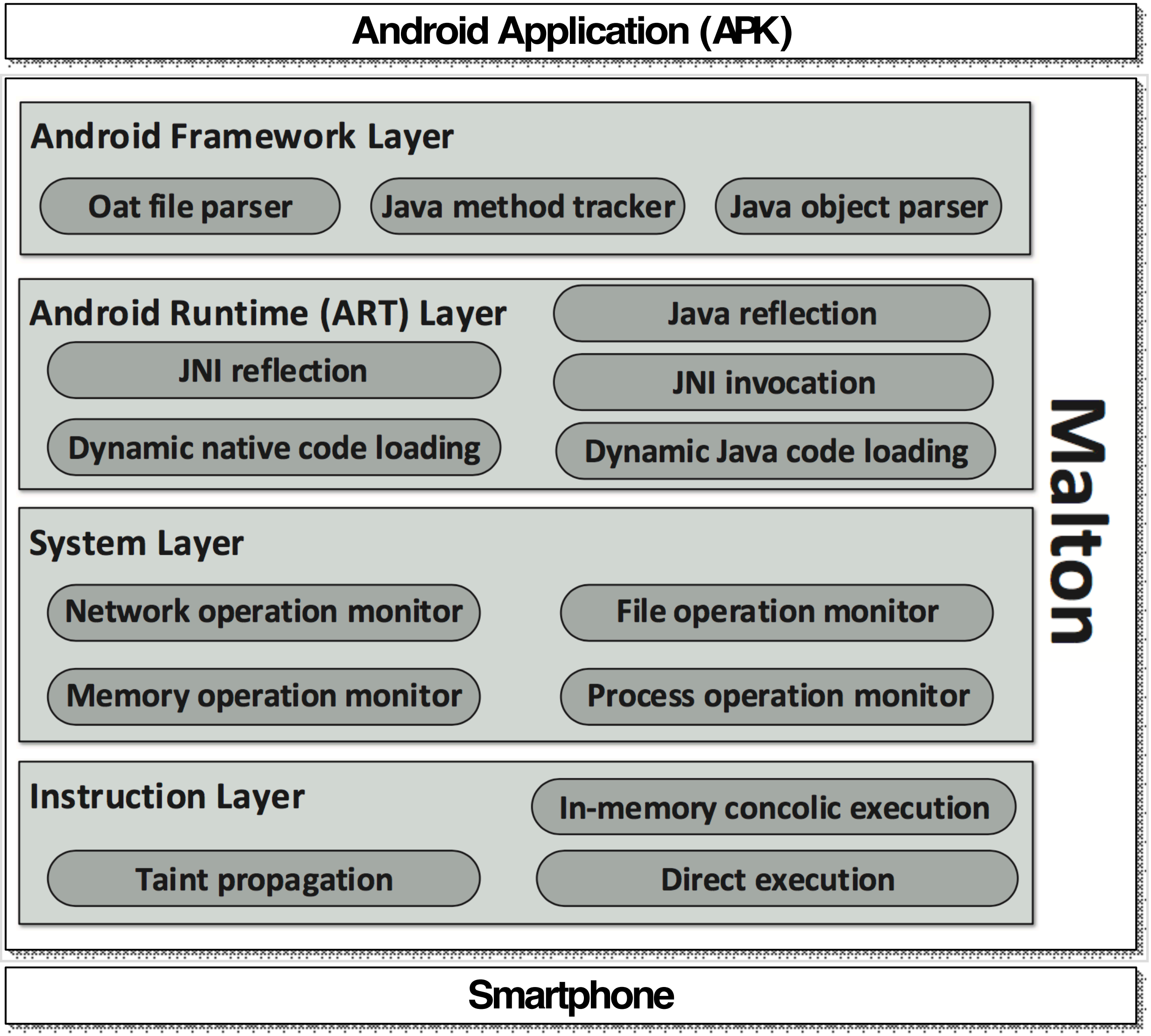
- Running on a real device;
- Conducting cross-layer monitoring and information flow tracking;
- Doesn’t need to modify the app.



- Running on a real device;
- Conducting cross-layer monitoring and information flow tracking;
- Doesn’t need to modify the app.

The Design of Malton





Android Framework Layer

Android Framework Layer

❑ OAT File Parser

- Get the beginning addresses of the compiled methods.
- Get the types of the methods’ parameters and return values.

Android Framework Layer

- ❑ **OAT File Parser**
  - Get the beginning addresses of the compiled methods.
  - Get the types of the methods’ parameters and return values.
- ❑ **Java Method Tracker**
  - Get information about the invoked methods with their parameters.
  - Get information about the returned methods with results.

Android Framework Layer

❑ **OAT File Parser**

- Get the beginning addresses of the compiled methods.
- Get the types of the methods’ parameters and return values.

❑ **Java Method Tracker**

- Get information about the invoked methods with their parameters.
- Get information about the returned methods with results.

❑ **Java Object Parser**

- Get the content stored in Java class instance.
  (i.e., result of *TelephonyManager.getDeviceId()*)

Android Framework Layer

❑ OAT File Parser

- Get the beginning addresses of the compiled methods.
- Get the types of the methods’ parameters and return values.

❑ Java Method Tracker

- Get information about the invoked methods with their parameters.
- Get information about the returned methods with results.

❑ Java Object Parser

- Get the content stored in Java class instance.
  (i.e., result of *TelephonyManager.getDeviceId()*)

StringObject
*object_.klass_*
*object_.monitor_*
…..
*value: “**6534900622308366**”*

Android Framework Layer

□ OAT File Parser

- Get the beginning addresses of the compiled methods.
- Get the types of the methods' parameters and return values.

□ Java Method Tracker

- Get information about the invoked methods with their parameters.
- Get information about the returned methods with results.

□ Java Object Parser

- Get the content stored in Java class instance.
  (i.e., result of *TelephonyManager.getDeviceId()*)

StringObject
*object_.klass_*
*object_.monitor_*
…..
*value: **"6534900622308366"***

Android Runtime Layer

Android Runtime Layer

❑ **Native Code Loading**

- Track the dynamically loaded native code

Android Runtime Layer

❑ **Native Code Loading**

- Track the dynamically loaded native code

❑ **Java Code Loading**

- Track the dynamically loaded Java code

Android Runtime Layer

❑ **Native Code Loading**

- Track the dynamically loaded native code

❑ **Java Code Loading**

- Track the dynamically loaded Java code

❑ **JNI Invocation**

- Track the native methods invoked by Java code

Android Runtime Layer

- ❑ **Native Code Loading**
  - Track the dynamically loaded native code
- ❑ **Java Code Loading**
  - Track the dynamically loaded Java code
- ❑ **JNI Invocation**
  - Track the native methods invoked by Java code
- ❑ **JNI Reflection**
  - Track the Java methods invoked by native code

Android Runtime Layer

- ❑ **Native Code Loading**
  - Track the dynamically loaded native code
- ❑ **Java Code Loading**
  - Track the dynamically loaded Java code
- ❑ **JNI Invocation**
  - Track the native methods invoked by Java code
- ❑ **JNI Reflection**
  - Track the Java methods invoked by native code
- ❑ **Java Reflection**
  - Track the Java methods invoked through Java reflection

Android Runtime Layer

- ❑ **Native Code Loading**
  - Track the dynamically loaded native code
- ❑ **Java Code Loading**
  - Track the dynamically loaded Java code
- ❑ **JNI Invocation**
  - Track the native methods invoked by Java code
- ❑ **JNI Reflection**
  - Track the Java methods invoked by native code
- ❑ **Java Reflection**
  - Track the Java methods invoked through Java reflection

Malton can be easily extended to support the tracking of new behaviors.

System Layer

System Layer

Android NDK

❑ **Network Operations**

- Monitor information leaked through network.
- Monitor the received remote commands.

System Layer

Android NDK

- **Network Operations**
  - Monitor information leaked through network.
  - Monitor the received remote commands.
- **File Operations**
  - Monitor the access of personal files in storage.
  - Monitor dynamically released and deleted data.

System Layer

Android NDK

- ❑ **Network Operations**
  - Monitor information leaked through network.
  - Monitor the received remote commands.
- ❑ **File Operations**
  - Monitor the access of personal files in storage.
  - Monitor dynamically released and deleted data.
- ❑ **Memory Operations**
  - Monitor dynamically memory modification .

System Layer

Android NDK

- ❑ **Network Operations**
  - Monitor information leaked through network.
  - Monitor the received remote commands.
- ❑ **File Operations**
  - Monitor the access of personal files in storage.
  - Monitor dynamically released and deleted data.
- ❑ **Memory Operations**
  - Monitor dynamically memory modification .
- ❑ **Process Operations**
  - Monitor protection behaviors (e.g., anti-emulator and anti-debugging)

Instruction Layer

Instruction Layer

- **Taint Propagation**
  - Track the information leakage flow.

Instruction Layer

❑ **Taint Propagation**

- Track the information leakage flow.

❑ **In-memory Concolic Execution**

- Explore more execution paths.

Instruction Layer

- **Taint Propagation**
  - Track the information leakage flow.
- **In-memory Concolic Execution**
  - Explore more execution paths.
- **Direct Execution**
  - Explore execution path, of which no input is generated.

Taint Propagation

We propagate taint tags according to the logics of 9 IR statements and 11 IR expressions.

```
……
t12 = Load(0xabcd1234)
Put(8) = t12
……
```

Execution

Taint Propagation

We propagate taint tags according to the logics of 9 IR statements and 11 IR expressions.

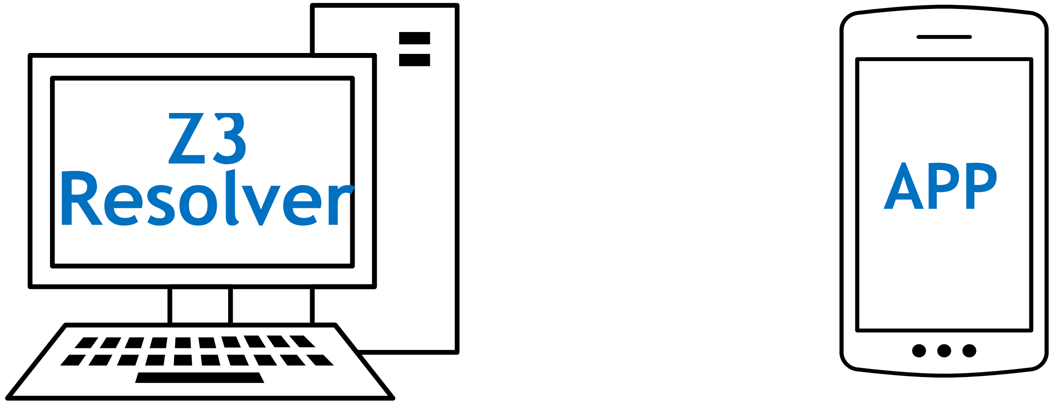
```
……
t12 = Load(0xabcd1234)
Put(8) = t12
……
```

Execution

```
……
Taint(t12) = Taint(0xabcd1234)
Taint(8) = Taint(t12)
……
```

Taint propagations

Path Exploration

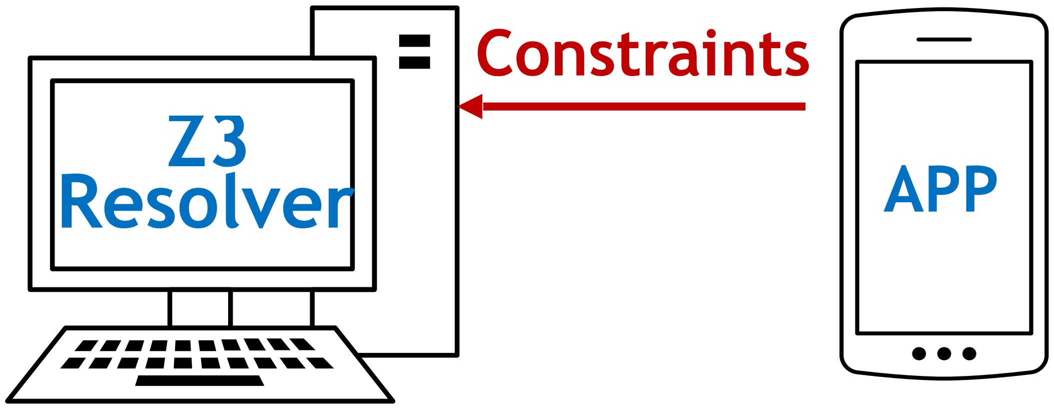


Desktop

Phone

Concolic Execution with Offloading Mechanism

Path Exploration



Z3 Resolver

Constraints

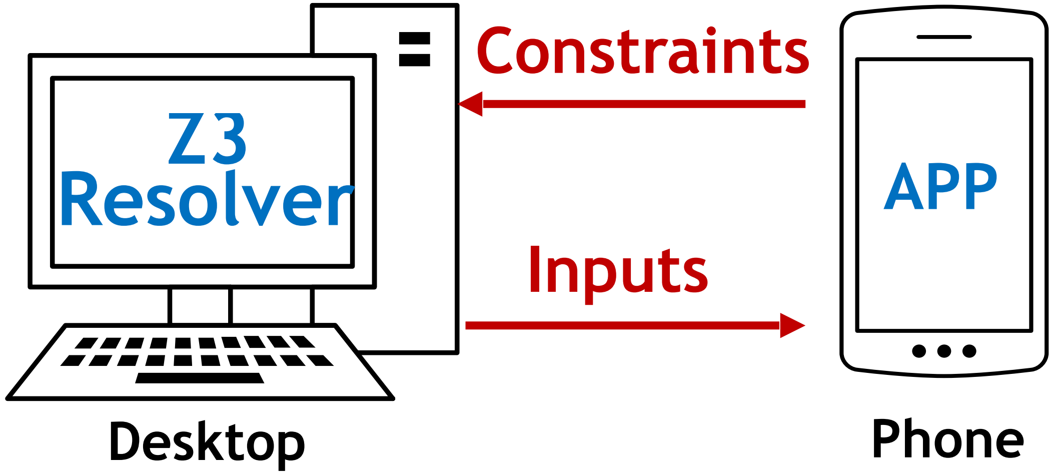
APP

Desktop

Phone

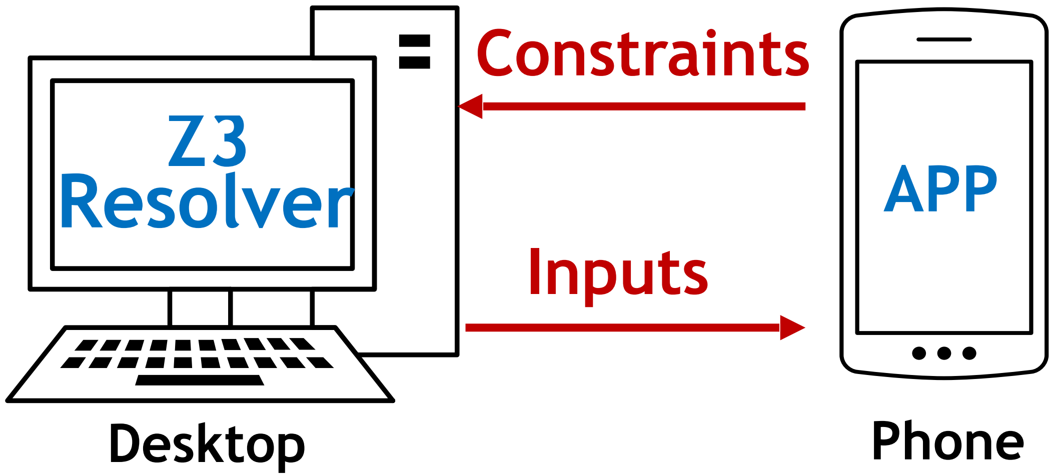
Concolic Execution with Offloading Mechanism

Path Exploration

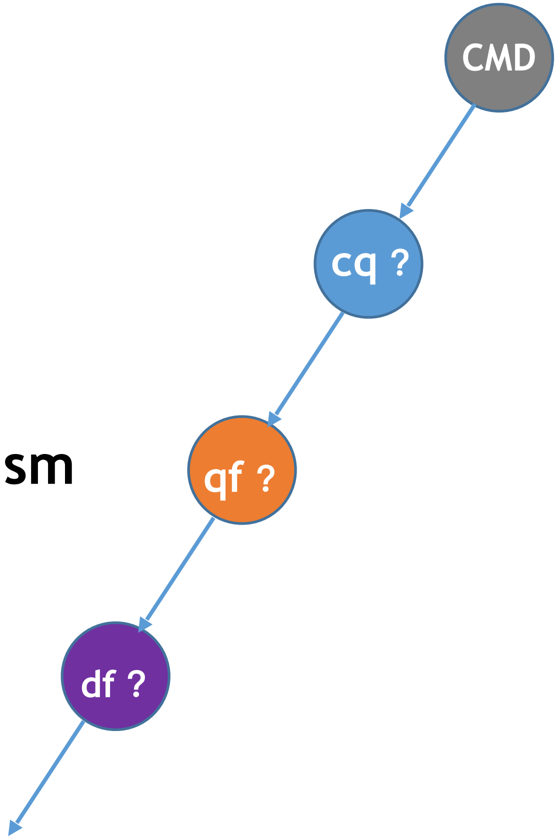


Concolic Execution with Offloading Mechanism

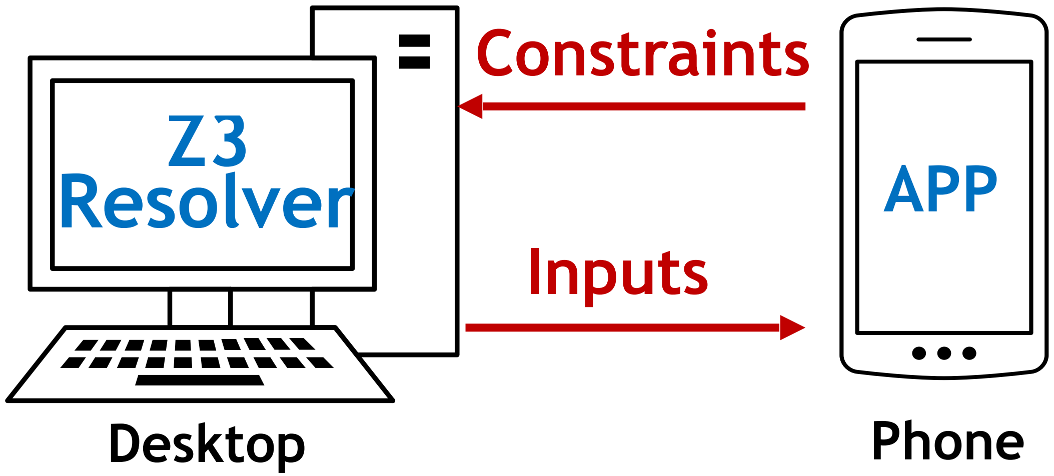
Path Exploration



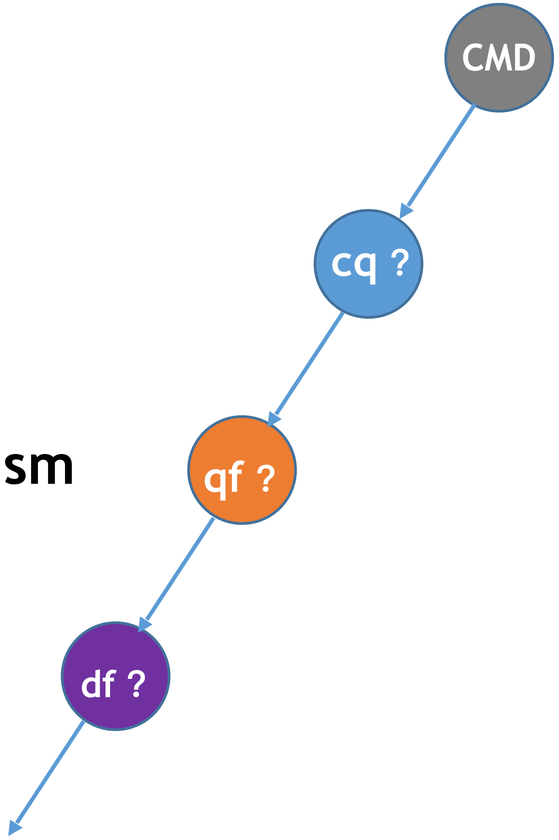
Concolic Execution with Offloading Mechanism



Path Exploration



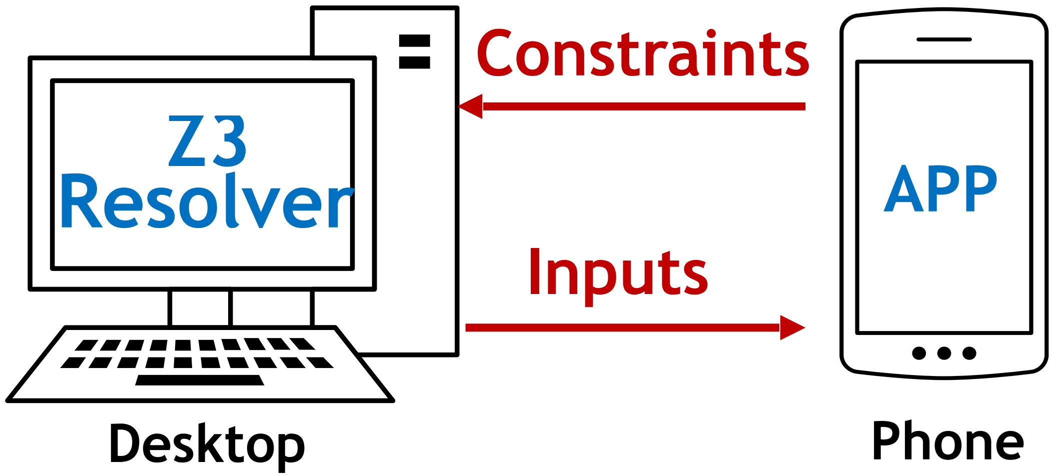
Concolic Execution with Offloading Mechanism



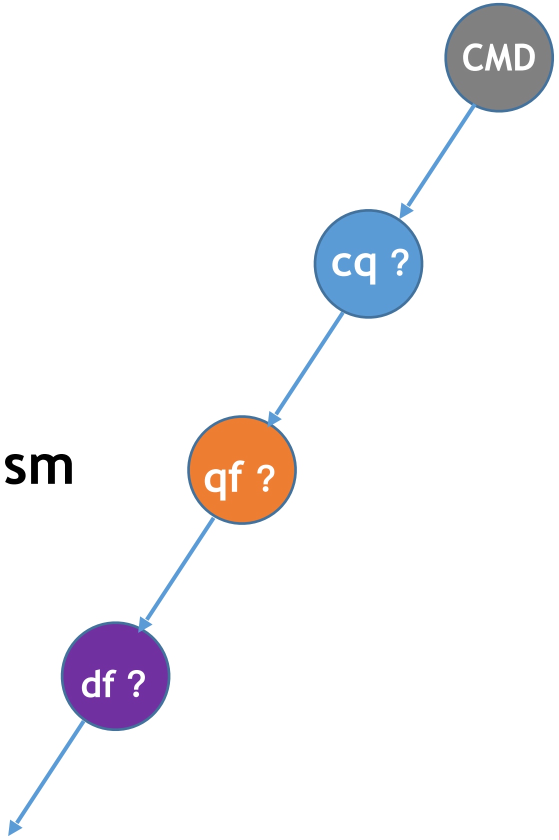
Constraints:

```
 CMD == “cq” ?
 CMD == “qf” ?
 CMD == ”df” ?
 …
```

Path Exploration



Concolic Execution with Offloading Mechanism

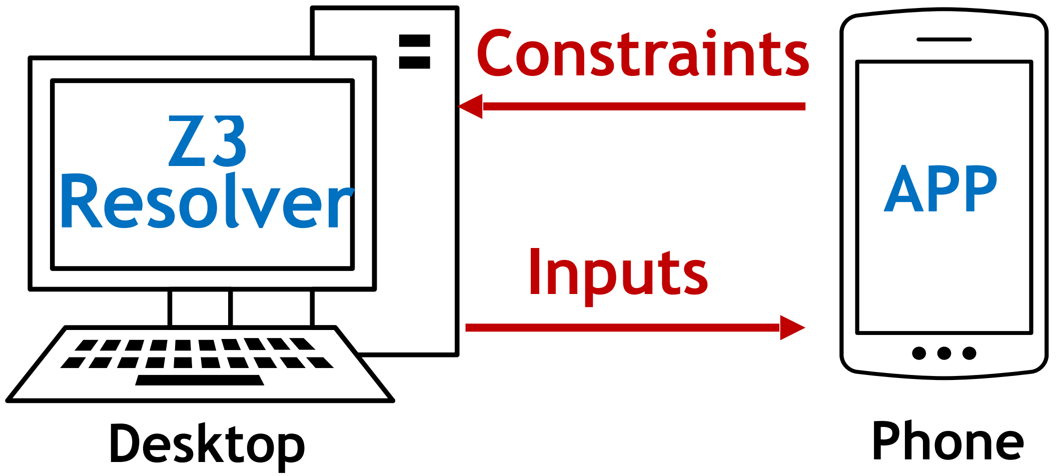


```
Inputs:
 CMD = “cq”
 CMD = “qf”
 CMD = “df”
 …
```

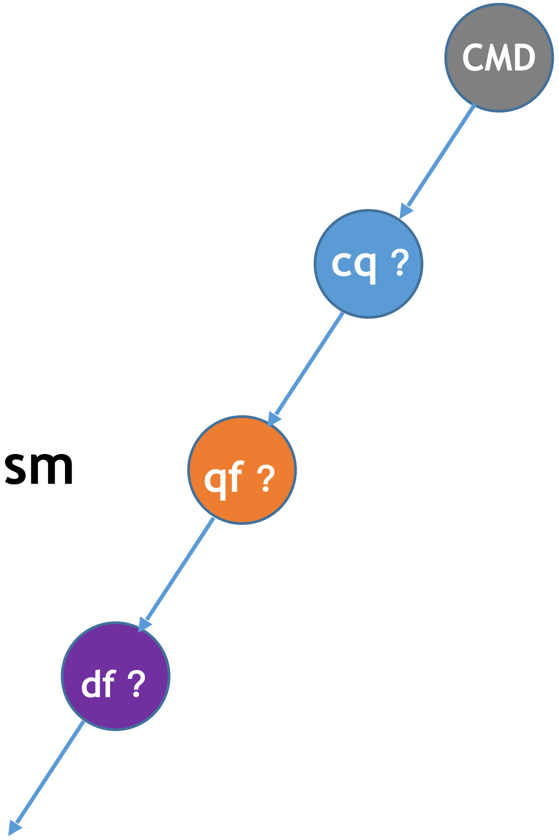
Constraints:

```
 CMD == “cq” ?
 CMD == “qf” ?
 CMD == ”df” ?
 …
```

Path Exploration



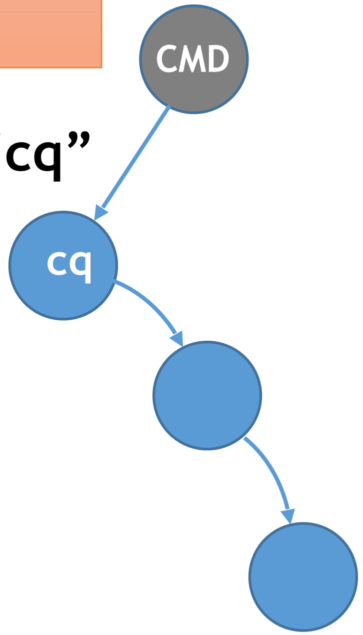
Concolic Execution with Offloading Mechanism



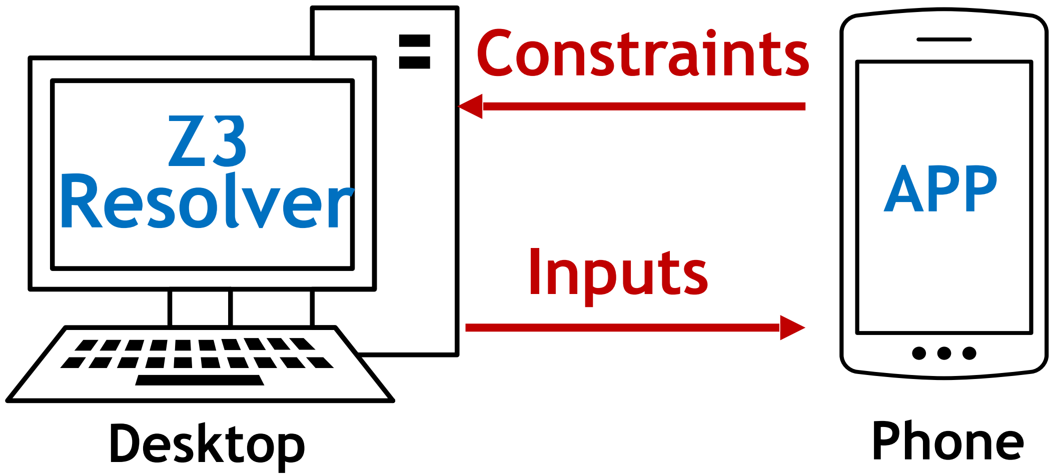
Constraints:
```
 CMD == “cq” ?
 CMD == “qf” ?
 CMD == ″df″ ?
 …
```

Inputs:
```
 CMD = “cq”
 CMD = “qf”
 CMD = “df”
 …
```

CMD = “cq”



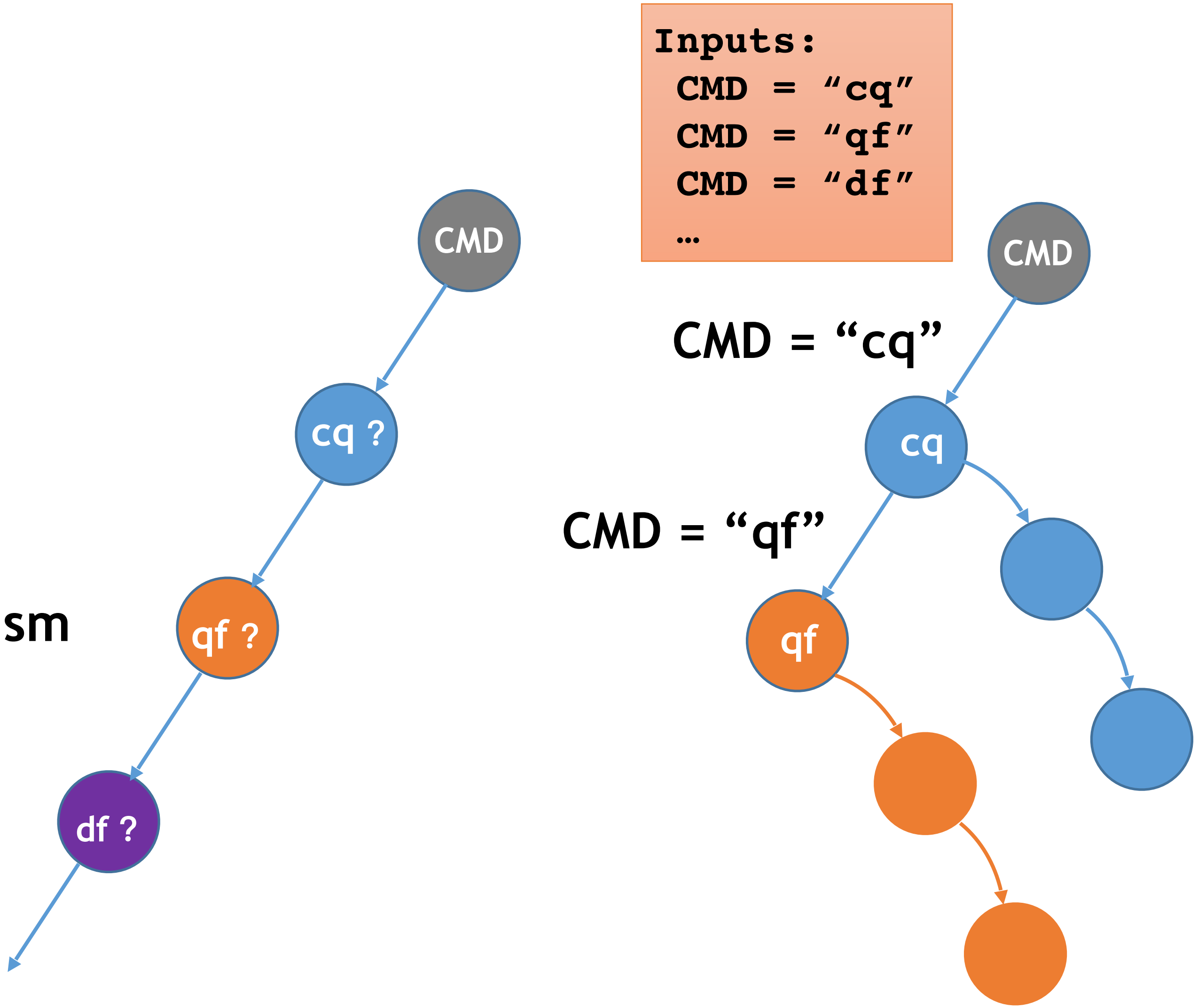
Path Exploration



Concolic Execution with Offloading Mechanism

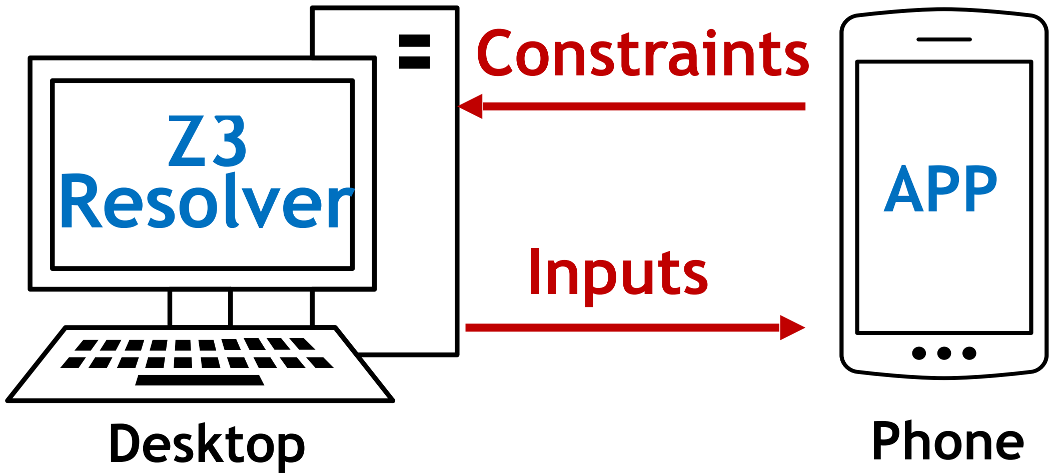
Constraints:

```
 CMD == “cq” ?
 CMD == “qf” ?
 CMD == ”df” ?
 …
```

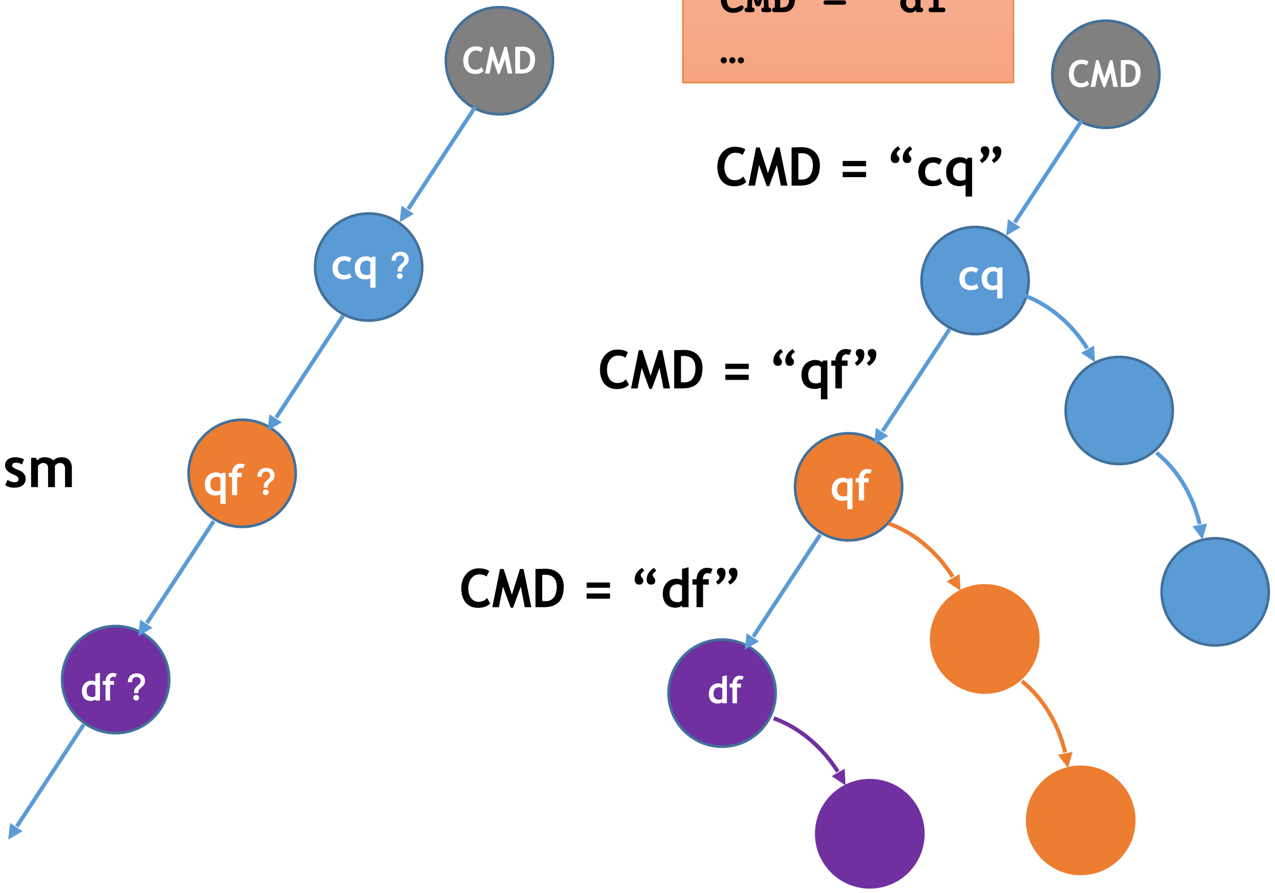


```
Inputs:
 CMD = “cq”
 CMD = “qf”
 CMD = “df”
 …
```

Path Exploration



Concolic Execution with Offloading Mechanism

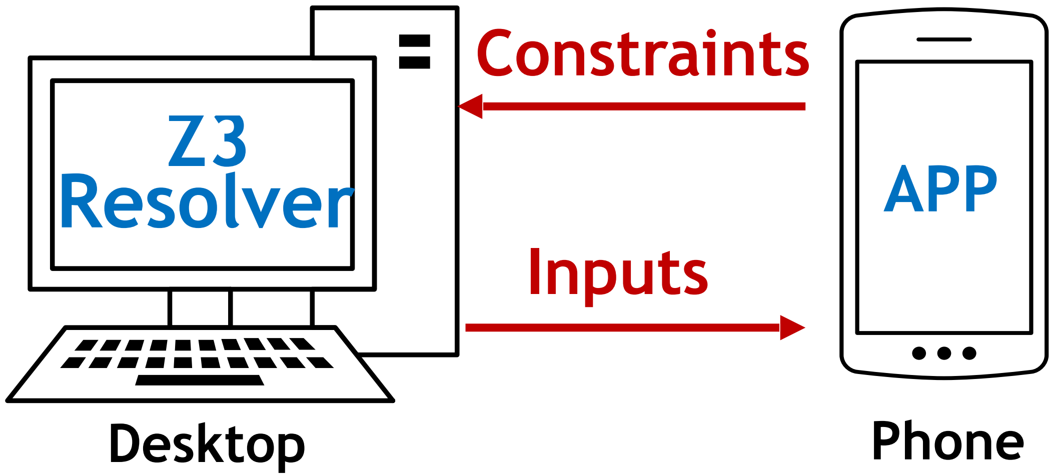


```
Inputs:
 CMD = "cq"
 CMD = "qf"
 CMD = "df"
 …
```

Constraints:

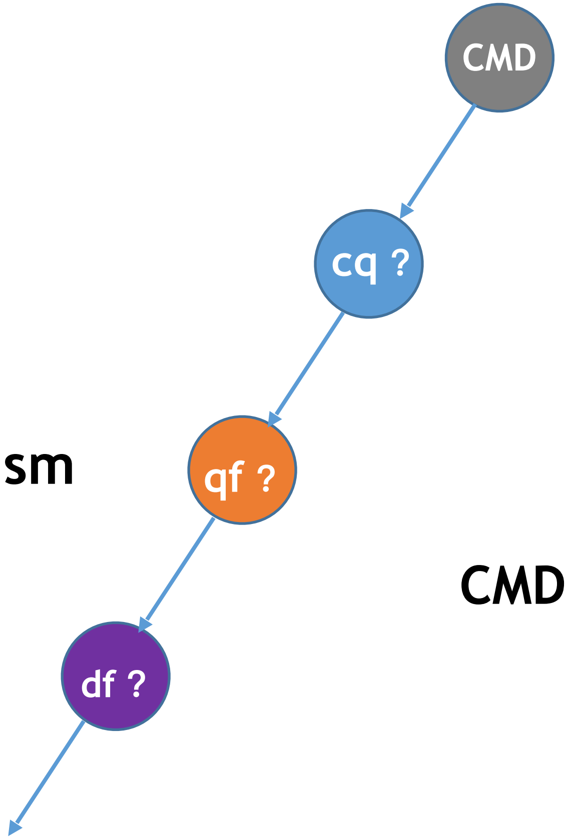
```
 CMD == "cq" ?
 CMD == "qf" ?
 CMD == "df" ?
 …
```

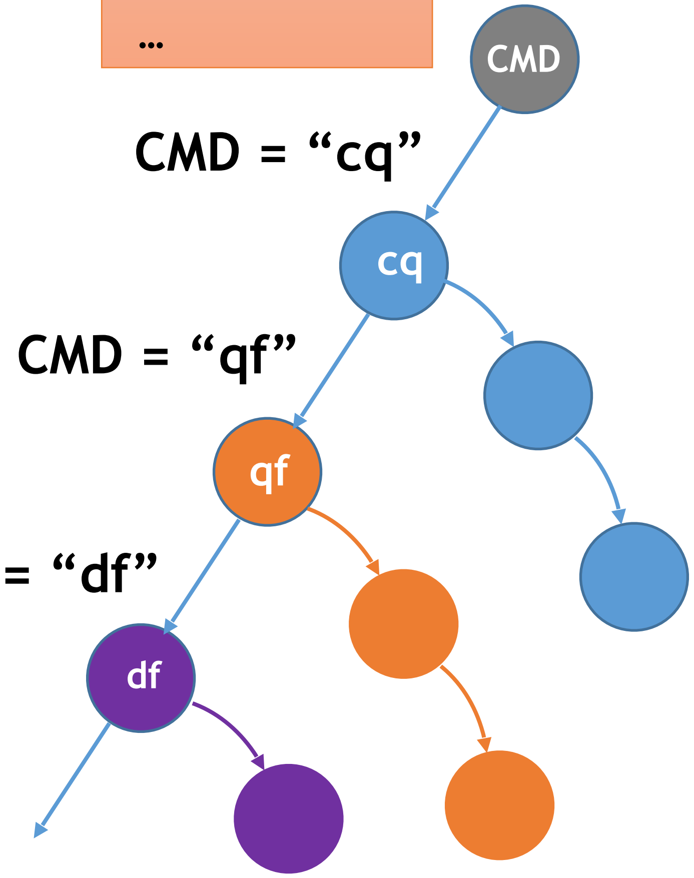
Path Exploration



Concolic Execution with Offloading Mechanism

```
Inputs:
 CMD = “cq”
 CMD = “qf”
 CMD = “df”
 …
```





CMD = “cq”

CMD = “qf”

CMD = “df”

```
Constraints:
 CMD == “cq” ?
 CMD == “qf” ?
 CMD == ”df” ?
 …
```

Path Exploration

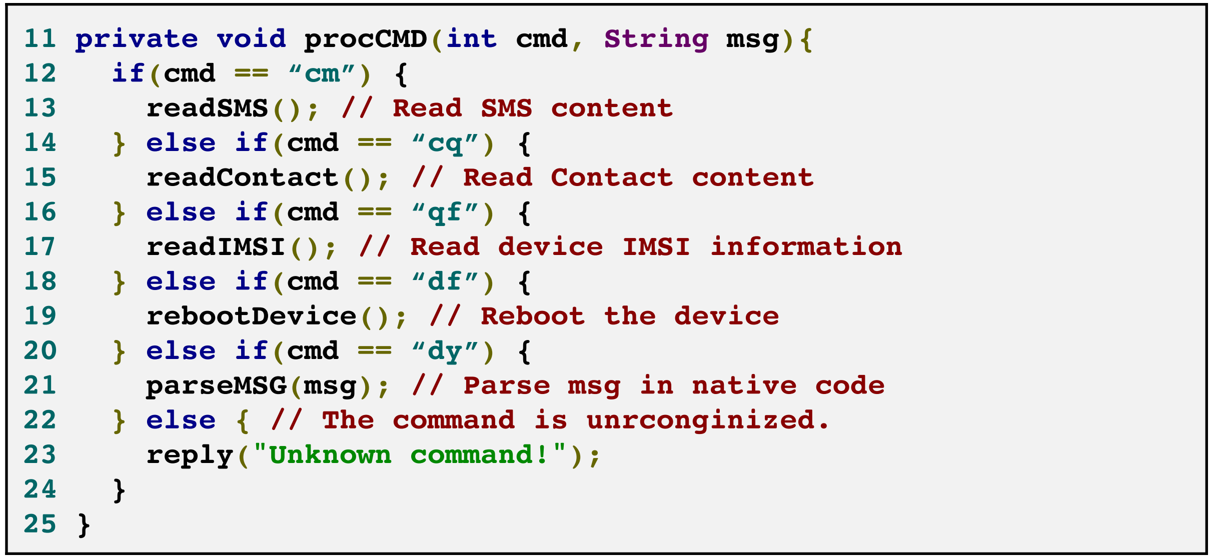
In-memory optimization：

Run specified code regions iteratively with different inputs

Path Exploration

In-memory optimization：

Run specified code regions iteratively with different inputs

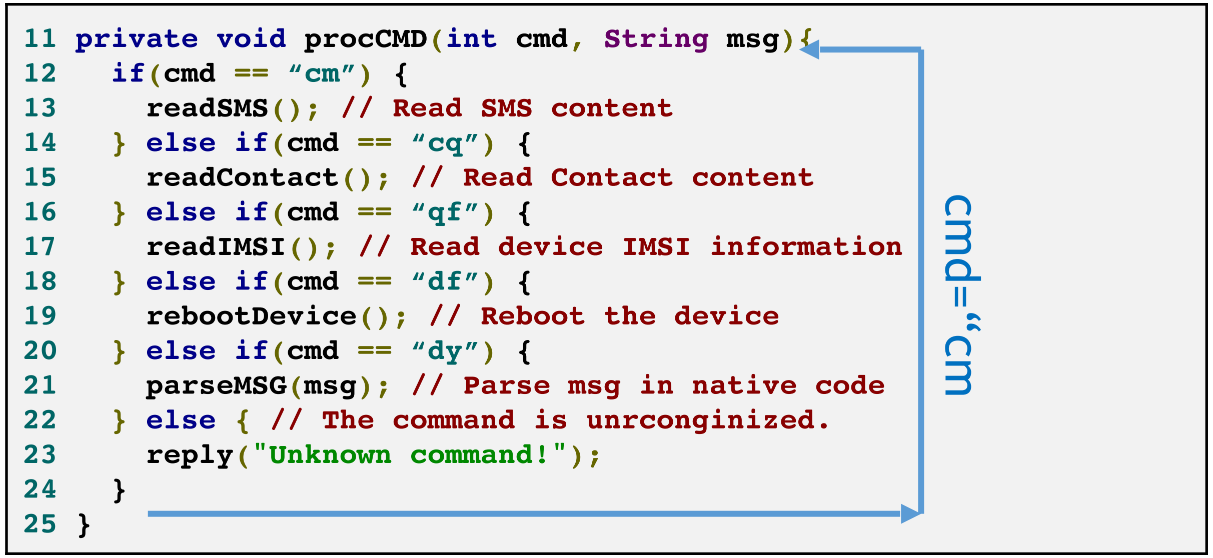


```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Path Exploration

In-memory optimization：

Run specified code regions iteratively with different inputs

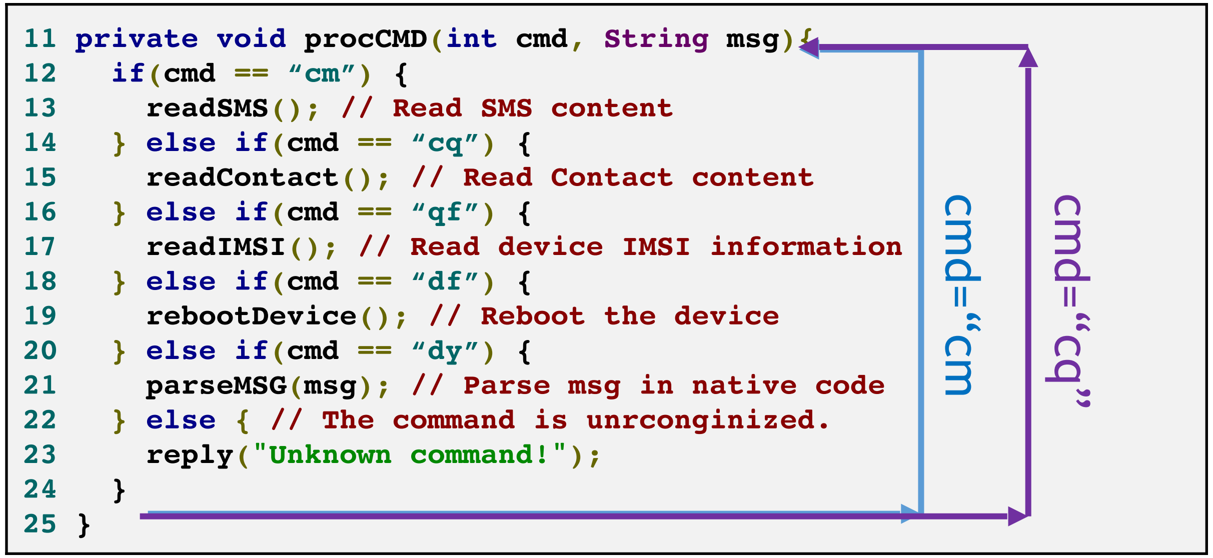


```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Path Exploration

In-memory optimization：

Run specified code regions iteratively with different inputs

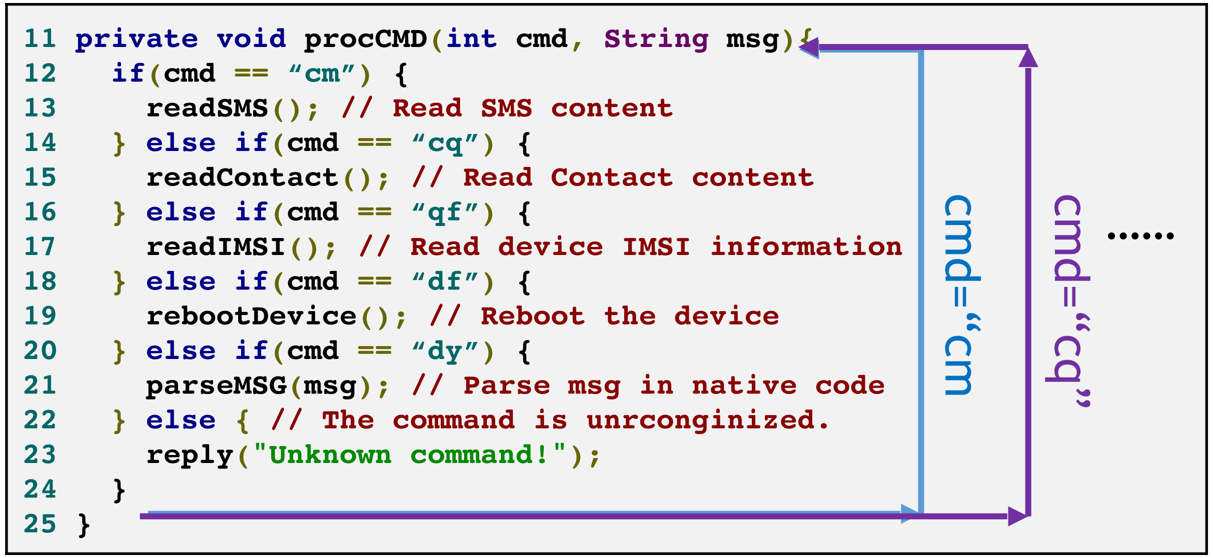


```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Path Exploration

In-memory optimization：

Run specified code regions iteratively with different inputs

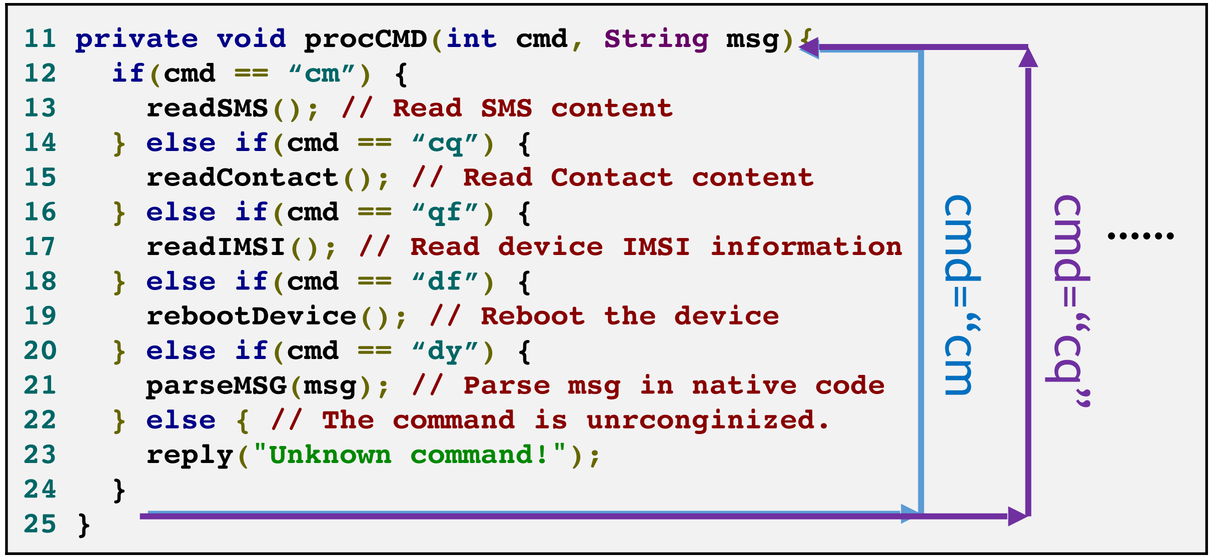


```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Path Exploration

In-memory optimization：

Run specified code regions iteratively with different inputs



```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

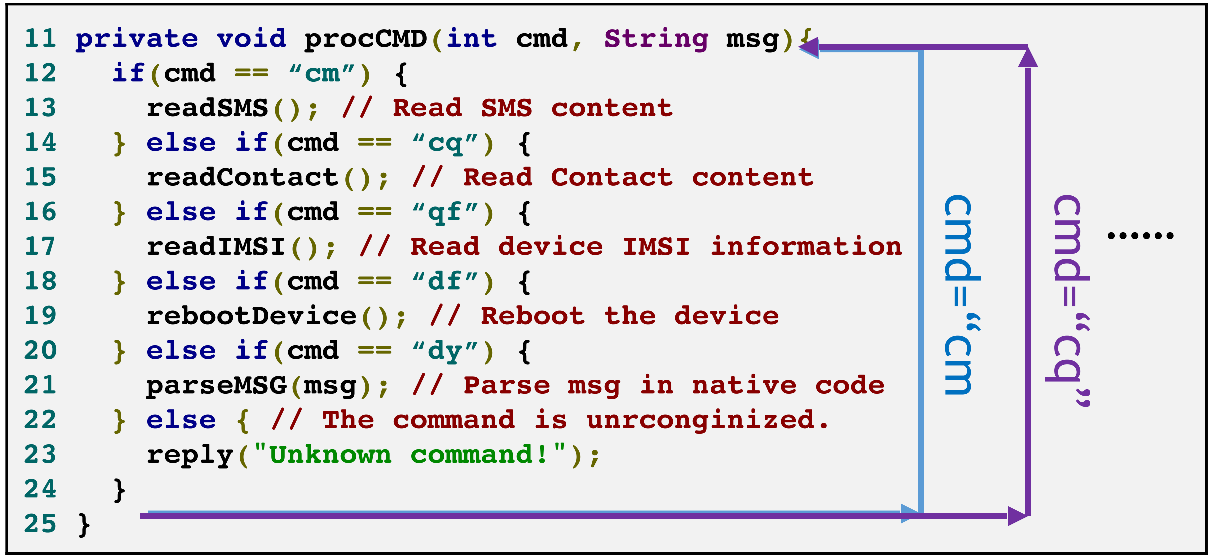
Direct Execution

Force an app execute a path without input

Path Exploration

In-memory optimization：

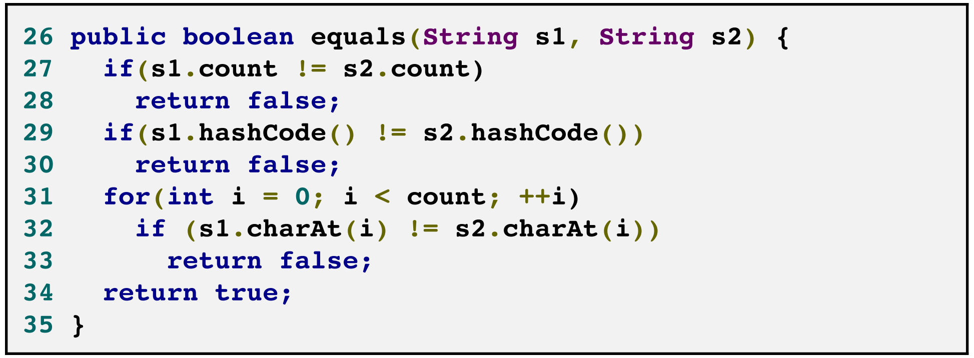
Run specified code regions iteratively with different inputs



```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Direct Execution

Force an app execute a path without input

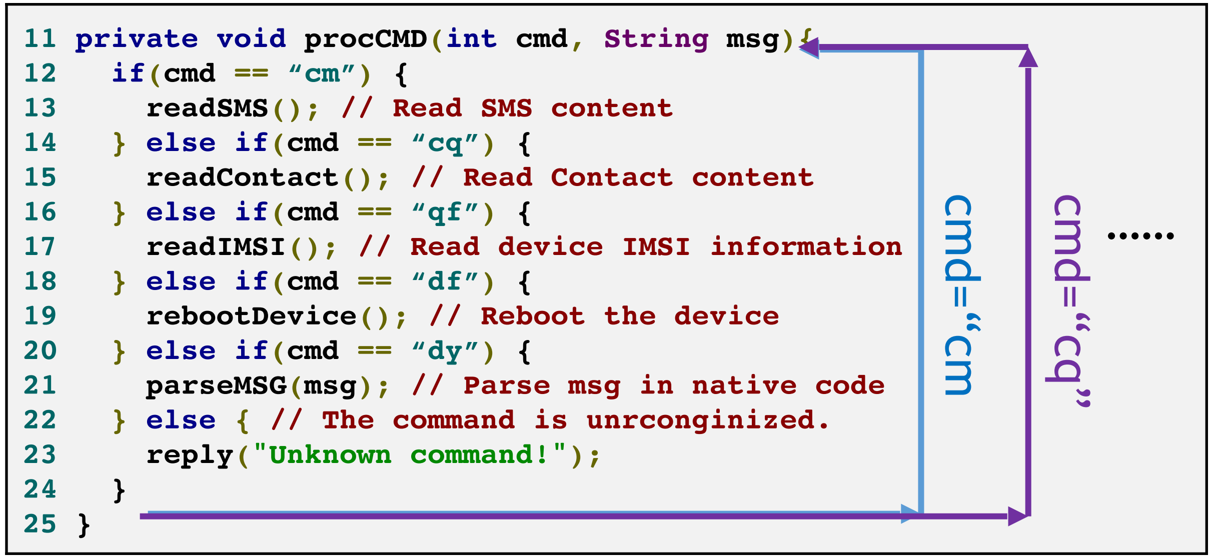


```
public boolean equals(String s1, String s2) {
  if(s1.count != s2.count)
    return false;
  if(s1.hashCode() != s2.hashCode())
    return false;
  for(int i = 0; i < count; ++i)
    if (s1.charAt(i) != s2.charAt(i))
      return false;
  return true;
}
```

Path Exploration

In-memory optimization：

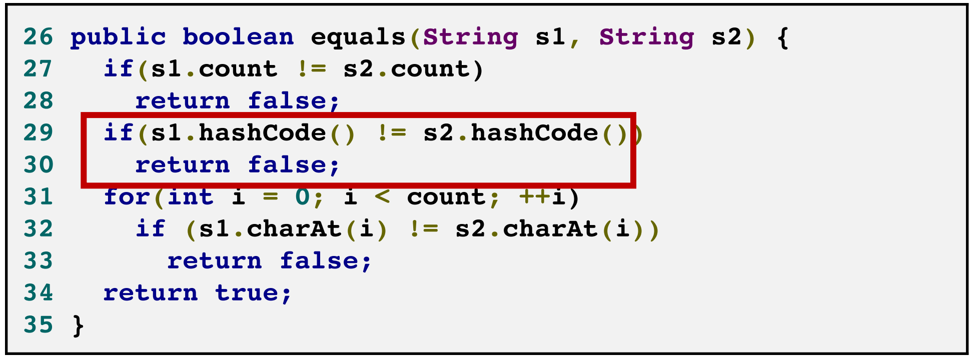
Run specified code regions iteratively with different inputs



```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Direct Execution

Force an app execute a path without input

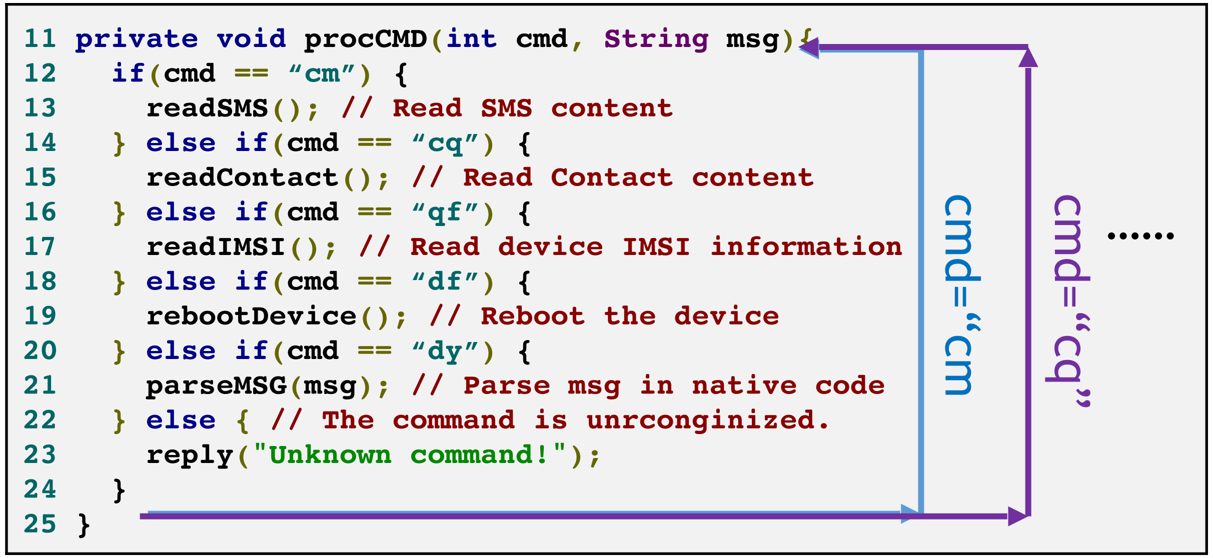


```
public boolean equals(String s1, String s2) {
  if(s1.count != s2.count)
    return false;
  if(s1.hashCode() != s2.hashCode())
    return false;
  for(int i = 0; i < count; ++i)
    if (s1.charAt(i) != s2.charAt(i))
      return false;
  return true;
}
```

Path Exploration

In-memory optimization：

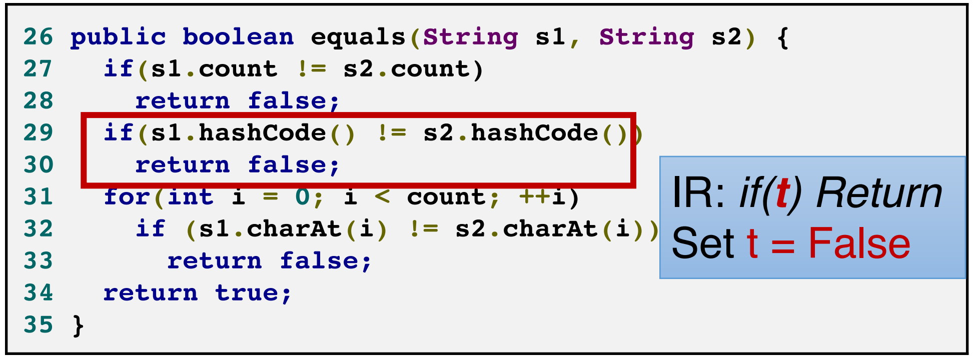
Run specified code regions iteratively with different inputs



```
private void procCMD(int cmd, String msg){
  if(cmd == “cm”) {
    readSMS(); // Read SMS content
  } else if(cmd == “cq”) {
    readContact(); // Read Contact content
  } else if(cmd == “qf”) {
    readIMSI(); // Read device IMSI information
  } else if(cmd == “df”) {
    rebootDevice(); // Reboot the device
  } else if(cmd == “dy”) {
    parseMSG(msg); // Parse msg in native code
  } else { // The command is unrconginized.
    reply("Unknown command!");
  }
}
```

Direct Execution

Force an app execute a path without input



```
public boolean equals(String s1, String s2) {
  if(s1.count != s2.count)
    return false;
  if(s1.hashCode() != s2.hashCode())
    return false;
  for(int i = 0; i < count; ++i)
    if (s1.charAt(i) != s2.charAt(i))
      return false;
  return true;
}
```

IR: *if(**t**) Return*

Set t = False

Agenda

- ❑ Motivating Example
- ❑ The New Android Runtime (ART)
- ❑ Malton
- ❑ **Evaluation**
- ❑ Conclusion

Discovering Sensitive Operations

Behavior	CopperDroid	DroidBox	Malton
Personal Info	435 (85.0%)	135 (26.4%)	511 (99.8%)
Network access	351 (68.5%)	211 (41.2%)	445 (86.9%)
File access	438 (85.5%)	509 (99.4%)	512 (100%)
Phone call	52 (10.1%)	1 (0.2%)	59 (11.5%)
Send SMS	26 (5.1%)	15 (2.9%)	28 (5.5%)
Java code loading	NA	509 (99.4%)	512 (100%)
Anti-debugging	4 (0.8%)	NA	4 (0.8%)
Native code loading	NA	NA	160 (31.2%)

- 512 samples and results of CopperDroid are downloaded from its web servers.

Discovering Sensitive Operations

Behavior	CopperDroid	DroidBox	Malton
Personal Info	435 (85.0%)	135 (26.4%)	511 (99.8%)
Network access	351 (68.5%)	211 (41.2%)	445 (86.9%)
File access	438 (85.5%)	509 (99.4%)	512 (100%)
Phone call	52 (10.1%)	1 (0.2%)	59 (11.5%)
Send SMS	26 (5.1%)	15 (2.9%)	28 (5.5%)
Java code loading	NA	509 (99.4%)	512 (100%)
Anti-debugging	4 (0.8%)	NA	4 (0.8%)
Native code loading	NA	NA	160 (31.2%)

- 512 samples and results of CopperDroid are downloaded from its web servers.

Discovering Sensitive Operations

Behavior	CopperDroid	DroidBox	Malton
Personal Info	435 (85.0%)	135 (26.4%)	511 (99.8%)
Network access	351 (68.5%)	211 (41.2%)	445 (86.9%)
File access	438 (85.5%)	509 (99.4%)	512 (100%)
Phone call	52 (10.1%)	1 (0.2%)	59 (11.5%)
Send SMS	26 (5.1%)	15 (2.9%)	28 (5.5%)
Java code loading	NA	509 (99.4%)	512 (100%)
Anti-debugging	4 (0.8%)	NA	4 (0.8%)
Native code loading	NA	NA	160 (31.2%)

- 512 samples and results of CopperDroid are downloaded from its web servers.

Result: Malton can capture more sensitive behaviors thanks to its on-device and cross-layer inspection.

Path Exploration

Command	Detected behavior	No. of executed blocks
“cq”	Read information SMS contents, contacts, device model and system version, then send to 292019159c@fcvh77f.com with password “aAaccvv11” through SMTP protocol.	32k/20443k
“qf”	Send SMS to all contacts with no SMS content.	7k/20537k
“df”	Send SMS to specified number, and both the number and content are specified by the command SMS.	5k/22970k
“zy”	Set unconditional call forwarding through making call to “**21* targetNum%23”.	8k/22848k
“by”	Set call forwarding when the phone is busy through making call to “%23%23targetNum%23”.	15k/20639k
“ld”, “fd”, “dh”, “cz”, “fx”, “sx”, “dc”, “bc”	Modify the its configuration file zzxx.xml.	5k-18k/20403k-20452k
Others	Tell the controller the command format is error by replying an SMS.	15k/20443k

Path Exploration

Command	Detected behavior	No. of executed blocks
“cq”	Read information SMS contents, contacts, device model and system version, then send to 292019159c@fcvh77f.com with password “aAaccvv11” through SMTP protocol.	32k/20443k
“qf”	Send SMS to all contacts with no SMS content.	7k/20537k
“df”	Send SMS to specified number, and both the number and content are specified by the command SMS.	5k/22970k
“zy”	Set unconditional call forwarding through making call to “**21* targetNum%23”.	8k/22848k
“by”	Set call forwarding when the phone is busy through making call to “%23%23targetNum%23”.	15k/20639k
“ld”, “fd”, “dh”, “cz”, “fx”, “sx”, “dc”, “bc”	Modify the its configuration file zzxx.xml.	5k-18k/20403k-20452k
Others	Tell the controller the command format is error by replying an SMS.	15k/20443k

Path Exploration

Command	Detected behavior	No. of executed blocks
“cq”	Read information SMS contents, contacts, device model and system version, then send to 292019159c@fcvh77f.com with password “aAaccvv11” through SMTP protocol.	32k/20443k
“qf”	Send SMS to all contacts with no SMS content.	7k/20537k
“df”	Send SMS to specified number, and both the number and content are specified by the command SMS.	5k/22970k
“zy”	Set unconditi [illegible] n%23”.	8k/22848k
“by”	Set call forwarding when the phone is busy through making call to “%23%23targetNum%23”.	15k/20639k
“ld”, “fd”, “dh”, “cz”, “fx”, “sx”, “dc”, “bc”	Modify the its configuration file zzxx.xml.	5k-18k/20403k-20452k
Others	Tell the controller the command format is error by replying an SMS.	15k/20443k

With in-memory optimization: 5k

Path Exploration

Command	Detected behavior	No. of executed blocks
“cq”	Read information SMS contents, contacts, device model and system version, then send to 292019159c@fcvh77f.com with password “aAaccvv11” through SMTP protocol.	32k/20443k
“qf”	Send SMS to all contacts with no SMS content.	7k/20537k
“df”	Send SMS to specified number, and both the number and content are specified by the command SMS.	5k/22970k
“zy”	Set unconditi n%23”.	8k/22848k
“by”	Set call forwarding when the phone is busy through making call to “%23%23targetNum%23”	15k/20639k
“ld”, “fd”, “dh”, “cz”, “fx”, “sx”, “dc”, “bc”	Modify the its configuration file 22xx.xml.	5k-18k/20403k-20452k
Others	Tell the controller the command format is error by replying an SMS.	15k/20443k

With in-memory optimization: 5k

Without in-memory optimization: 22,970k

Path Exploration

Command	Detected behavior	No. of executed blocks
“cq”	Read information SMS contents, contacts, device model and system version, then send to 292019159c@fcvh77f.com with password “aAaccvv11” through SMTP protocol.	32k/20443k
“qf”	Send SMS to all contacts with no SMS content.	7k/20537k
“df”	Send SMS to specified number, and both the number and content are specified by the command SMS.	5k/22970k
“zy”	Set unconditi [illegible] n%23”.	8k/22848k
“by”	Set call forwarding when the phone is busy through making call to “%23%23targetNum%23”.	15k/20639k
“ld”, “fd”, “dh”, “cz”, “fx”, “sx”, “dc”, “bc”	Modify the its configuration file zzxx.xml.	5k-18k/20403k-20452k
Others	Tell the controller the command format is error by replying an SMS.	15k/20443k

With in-memory optimization: 5k

Without in-memory optimization: 22,970k

Result: Malton can explore paths effectively and efficiently because of the in-memory optimized concolic execution.

Agenda

- ❑ Motivating Example
- ❑ The New Android Runtime
- ❑ Malton
- ❑ Evaluation
- ❑ **Conclusion**

1

Propose and develop Malton, a novel on-device non-invasive analysis tool for ART.

1 Propose and develop Malton, a novel on-device non-invasive analysis tool for ART.

2 Malton can provide a comprehensive view of the Android malware behaviors through multi-layer tracking and path exploration.

1. Propose and develop Malton, a novel on-device non-invasive analysis tool for ART.

2. Malton can provide a comprehensive view of the Android malware behaviors through multi-layer tracking and path exploration.

3. In future, we will automate the in-memory optimisation and the recovery from crashes during direct execution.

Thank
You!!!