Android (Un)Packing Techniques

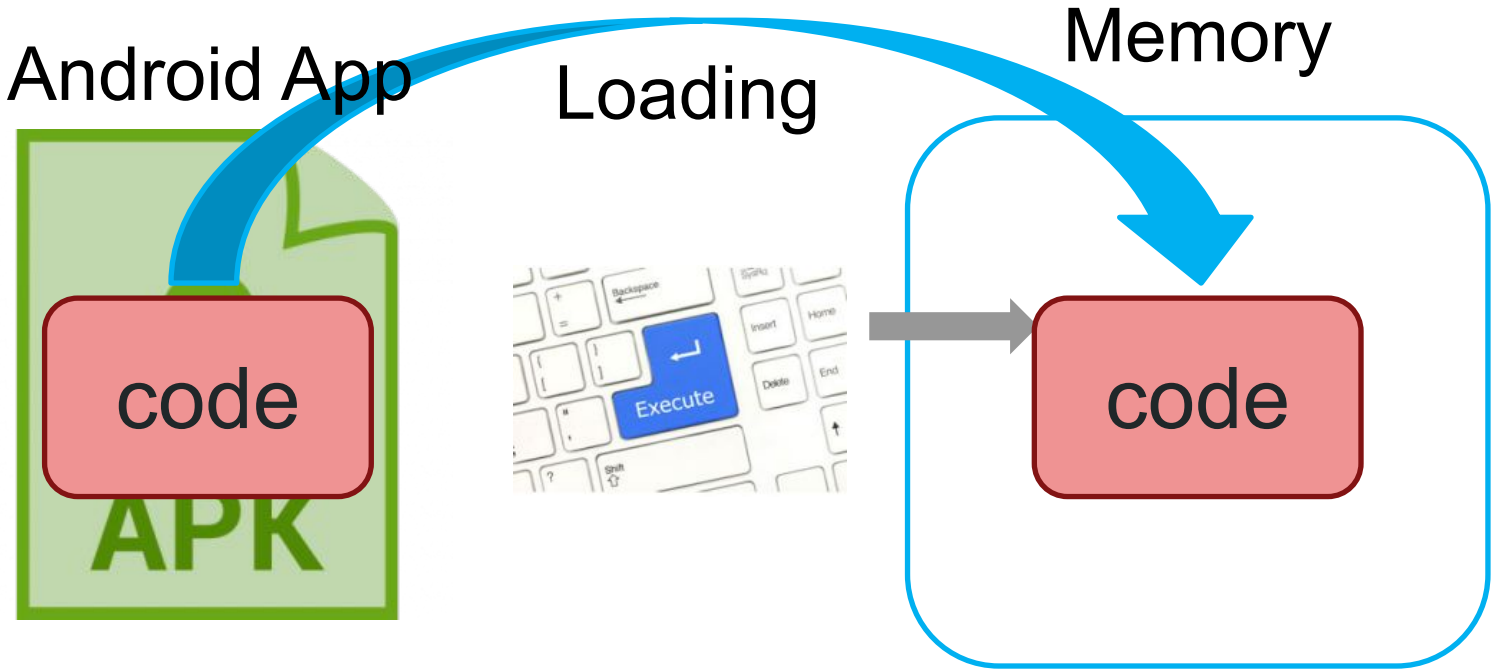
Yue Duan

Things You May Not Know About Android (Un)Packers: A Systematic Study based on Whole-System Emulation

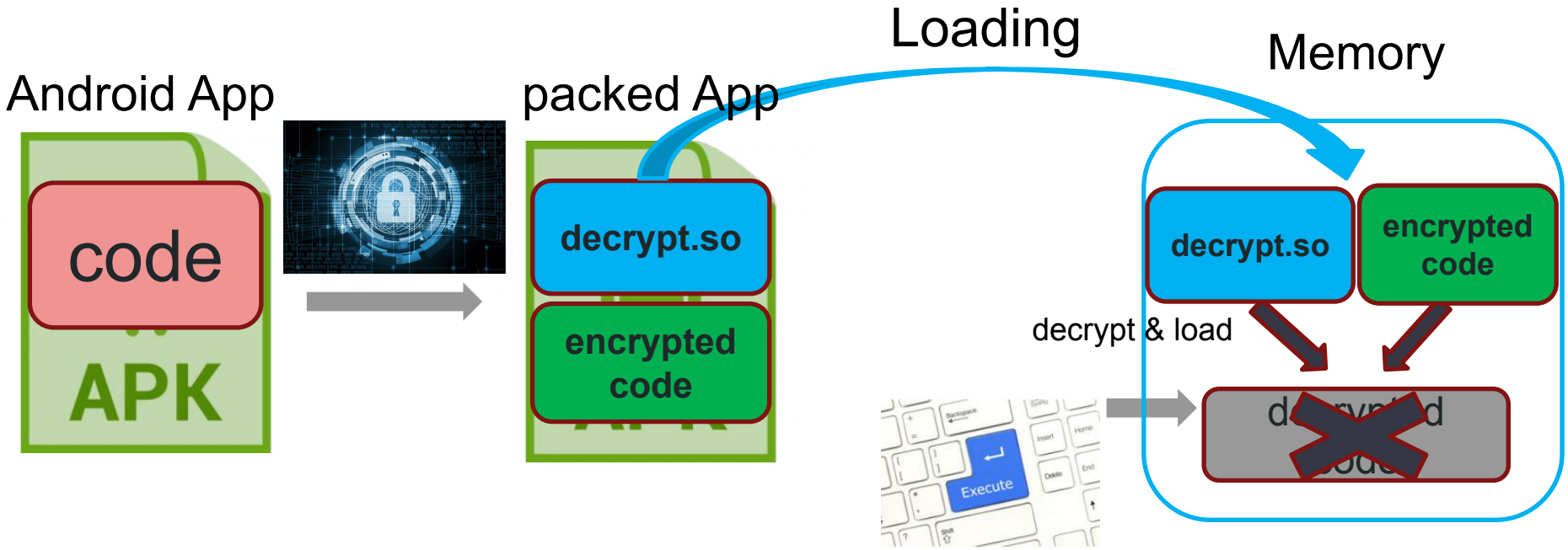
Yue Duan* , Mu Zhang† , Abhishek Vasisht Bhaskar‡ , Heng Yin* , Xiaorui Pan§ , Tongxin Li¶ , Xueqiang Wang§ , and XiaoFeng Wang§

NDSS 2018

What is Android packing



What is Android packing



What is Android packing

4.1 MB

```
.
./apktool.yml
./AndroidManifest.xml
./smali
./smali/com
./smali/com/example
./smali/com/example/hellojni
./smali/com/example/hellojni/R$color.smali
./smali/com/example/hellojni/R$layout.smali
./smali/com/example/hellojni/R$string.smali
./smali/com/example/hellojni/HelloJni.smali
./smali/com/example/hellojni/R$dimen.smali
./smali/com/example/hellojni/R$mipmap.smali
./smali/com/example/hellojni/R$integer.smali
./smali/com/example/hellojni/R.smali
./smali/com/example/hellojni/R$style.smali
./smali/com/example/hellojni/R$id.smali
./smali/com/example/hellojni/R$bool.smali
./smali/com/example/hellojni/R$anim.smali
./smali/com/example/hellojni/R$styleable.smali
./smali/com/example/hellojni/R$drawable.smali
./smali/com/example/hellojni/R$attr.smali
./smali/com/example/hellojni/BuildConfig.smali
./original
./original/META-INF
./original/META-INF/ALIAS_NA.SF
./original/META-INF/MANIFEST.MF
./original/META-INF/ALIAS_NA.RSA
./original/AndroidManifest.xml
./lib
./lib/armeabi-v7a
./lib/armeabi-v7a/libhello-jni.so
```

After packing

1 KB

```
.
./apktool.yml
./AndroidManifest.xml
./smali
./smali/com
./smali/com/ali
./smali/com/ali/fixHelper.smali
./smali/com/example
./smali/com/example/hellojni
./smali/com/example/hellojni/R$color.smali
./smali/com/example/hellojni/R$layout.smali
./smali/com/example/hellojni/R$string.smali
./smali/com/example/hellojni/HelloJni.smali
./smali/com/example/hellojni/R$dimen.smali
./smali/com/example/hellojni/R$mipmap.smali
./smali/com/example/hellojni/R$integer.smali
./smali/com/example/hellojni/R.smali
./smali/com/example/hellojni/R$style.smali
./smali/com/example/hellojni/R$id.smali
./smali/com/example/hellojni/R$bool.smali
./smali/com/example/hellojni/R$anim.smali
./smali/com/example/hellojni/R$styleable.smali
./smali/com/example/hellojni/R$drawable.smali
./smali/com/example/hellojni/R$attr.smali
./smali/com/example/hellojni/BuildConfig.smali
./original
./original/META-INF
./original/META-INF/ALIAS_NA.SF
./original/META-INF/MANIFEST.MF
./original/META-INF/ALIAS_NA.RSA
./original/AndroidManifest.xml
./lib
./lib/armeabi-v7a
./lib/armeabi-v7a/libpreverify1.so
./lib/armeabi-v7a/libdemolishdata
./lib/armeabi-v7a/libdemolish.so
./lib/armeabi-v7a/libdemolishdata.so
./lib/armeabi-v7a/libhello-jni.so
```

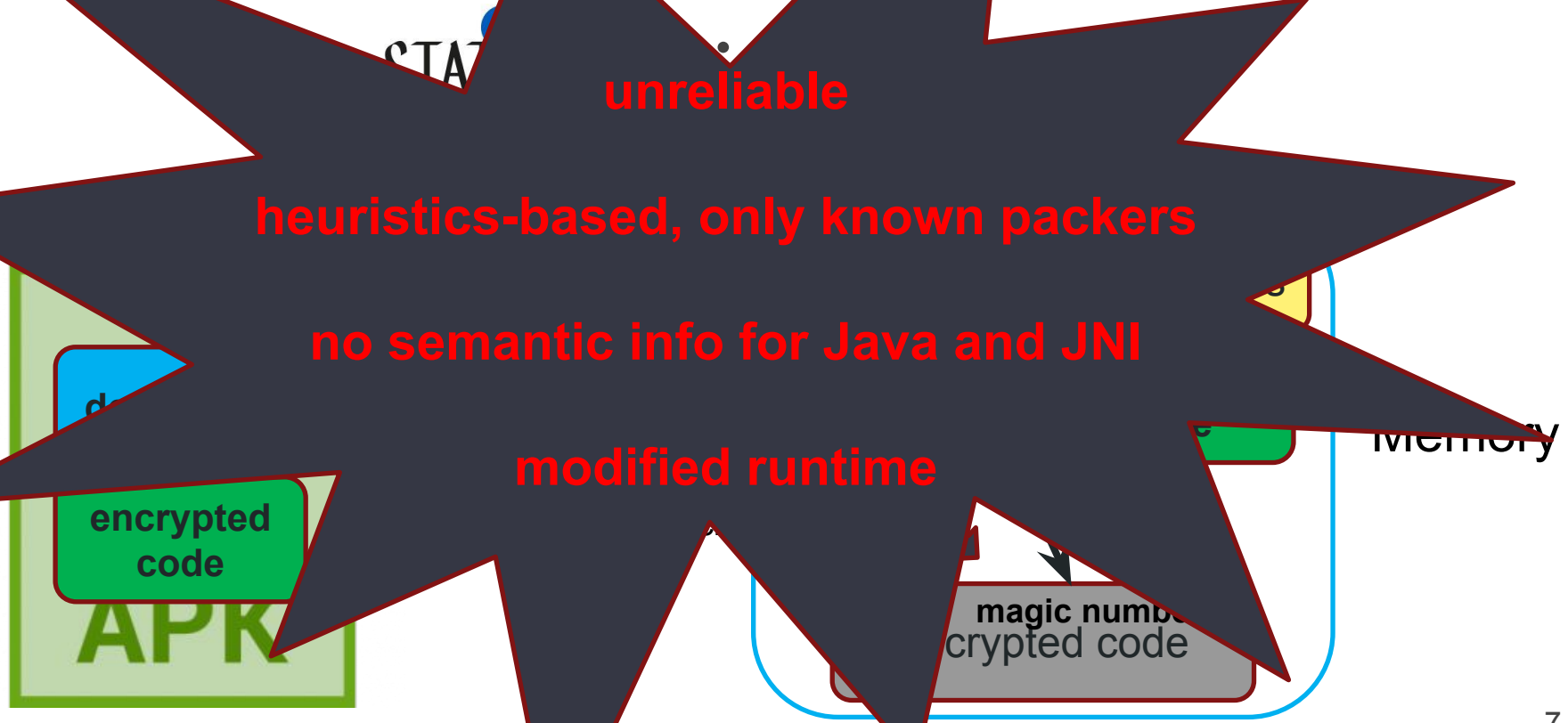
Why important

Android malware are evading detection

Check Po
SOFTWARE TECHNOLOGI

Charger, however, uses a l st compensate with other means. The developers of Charger gave it everything they had to boost its evasion capabilities and so it could stay hidden on Google Play for as long as possible.

Existing unpackers



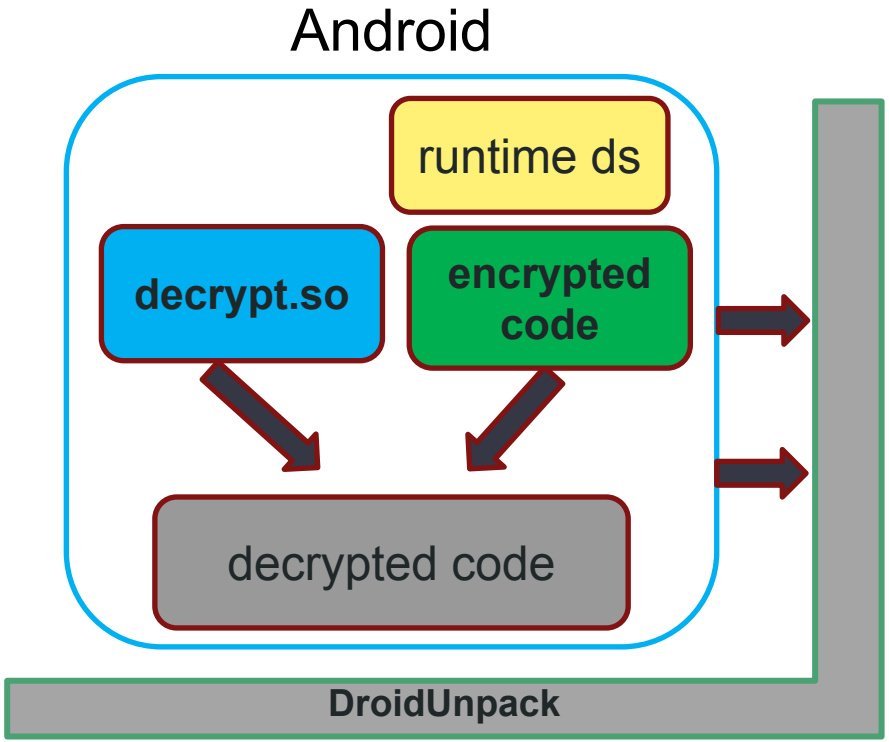
unreliable

heuristics-based, only known packers

no semantic info for Java and JNI

modified runtime

Our solution: DroidUnpack [NDSS'18]



Monitor from outside — **no modification**

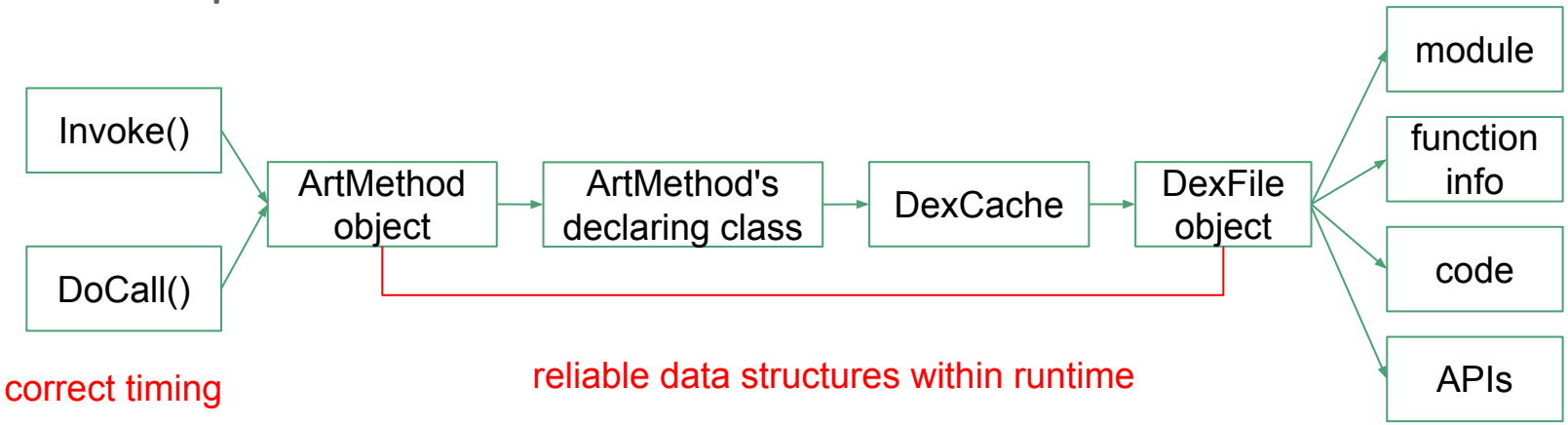
Intercept every function call — **multi-layer support**

Monitor java, native code and JNI — **semantic-level view**

Monitor memory ops for packing behaviors — **intrinsic characteristics**

Our solution: DroidUnpack [NDSS'18]

- Reconstruct ART Semantic View
  - Compiled Java functions
  - Interpreted Java functions



Evolution

Have Android packers evolved?

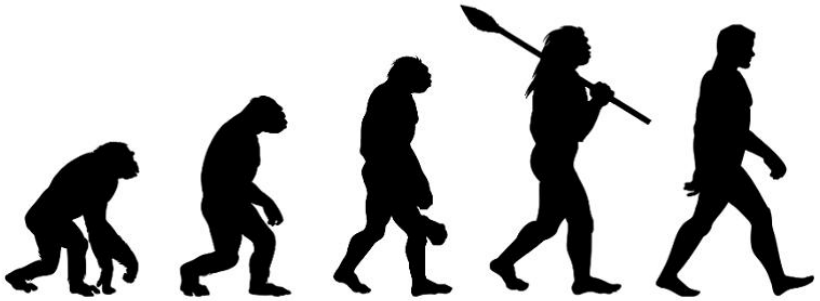


Fig. 5: Layer distribution.

Commercial packers

Commercial packers are supposed to protect your apps.
They do so by changing your apps.

Are the changes secure?

Commercial packers safety

Arbitrary code execution vulnerability

one vulnerable component is inserted stealthily

turn a secure app into a vulnerable one

acknowledged as highest priority vulnerability

won vulnerability bounty reward ~$8000

Privacy issue

Privacy leakage

adds six new permissions

collect sensitive user data

send back via an insecure connection

Impact

Gaode Navi: **500 million**

Qianniuniu finance: **3 million**

QQ: **800 million**

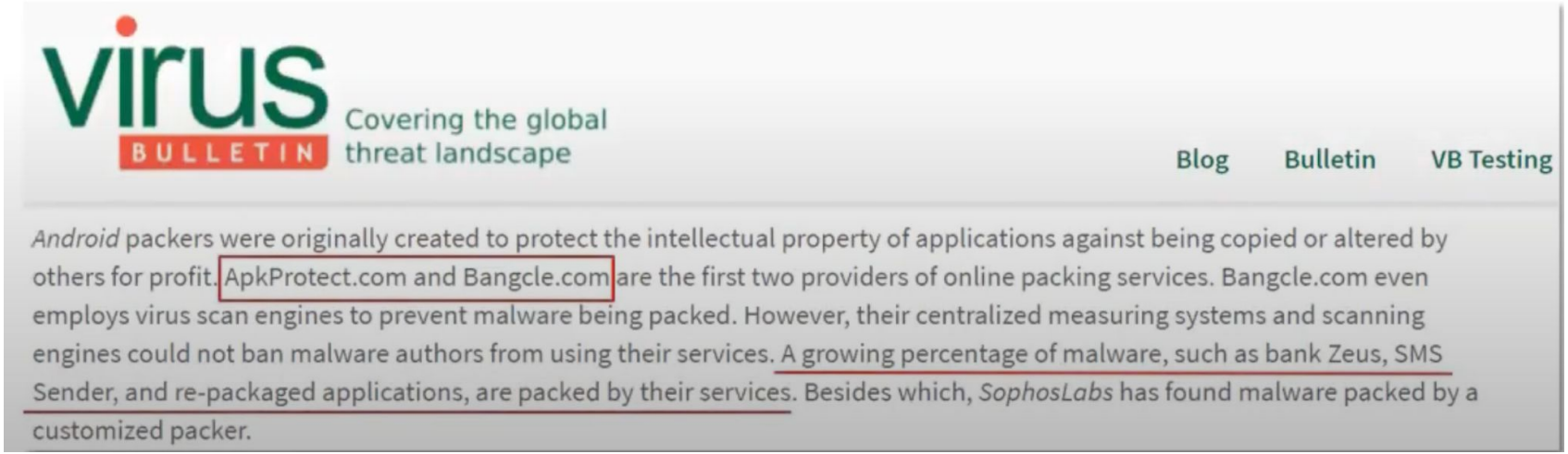
Happer: Unpacking Android Apps via a Hardware-Assisted Approach

Lei Xue, Hao Zhou, Xiapu Luo, Yajin Zhou, Yang Shi, Guofei Gu, Fengwei Zhang, Man Ho Au

IEEE S&P 2021

Android packing

- App packing aims to prevent bytecode of apps from being analyzed.
  - Protect benign apps from being repackaged[1]
- Packers have been abused by malware to evade detection. [2]



Android packing

- App packing aims to prevent bytecode of apps from being analyzed.
  - Protect benign apps from being repackaged[1]
- Packers have been abused by malware to evade detection. [2]

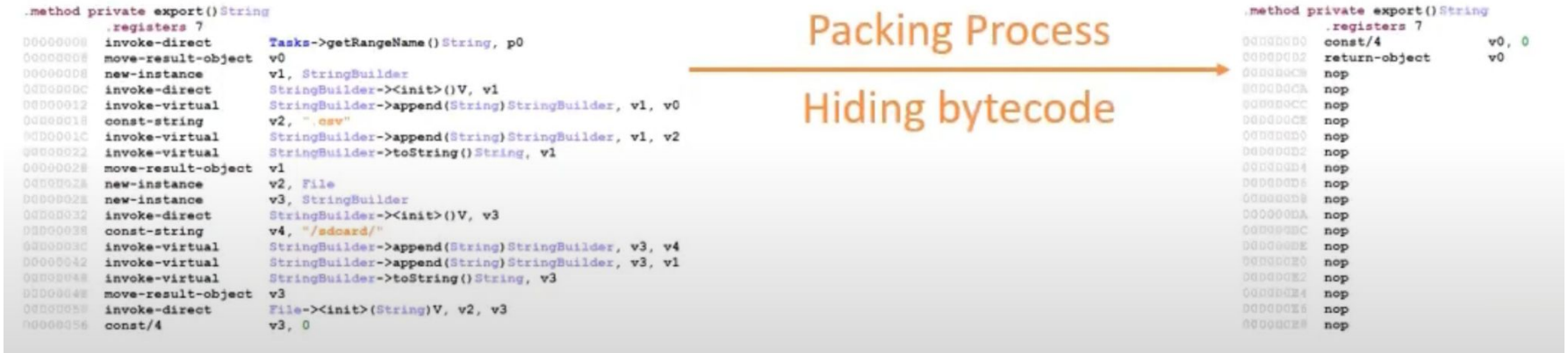
Forbes

How the apps bypassed Google Play's security controls

Specifically, the fraudsters used software called packers—which save space and obfuscate the final payload, then "unpack" their malicious code when the time comes — to bypass the Google Play Store's security controls.

Android packing

- Bytecode-hiding behavior.



Android packing

```
.method private export()String
        .registers 7
00000000  invoke-direct       Tasks->getRangeName()String, p0
00000006  move-result-object  v0
00000008  new-instance        v1, StringBuilder
0000000C  invoke-direct       StringBuilder-><init>()V, v1
00000012  invoke-virtual      StringBuilder->append(String)StringBuilder, v1, v0
00000018  const-string        v2, ".csv"
0000001C  invoke-virtual      StringBuilder->append(String)StringBuilder, v1, v2
00000022  invoke-virtual      StringBuilder->toString()String, v1
00000028  move-result-object  v1
0000002A  new-instance        v2, File
0000002E  new-instance        v3, StringBuilder
00000032  invoke-direct       StringBuilder-><init>()V, v3
00000038  const-string        v4, "/sdcard/"
0000003C  invoke-virtual      StringBuilder->append(String)StringBuilder, v3, v4
00000042  invoke-virtual      StringBuilder->append(String)StringBuilder, v3, v1
00000048  invoke-virtual      StringBuilder->toString()String, v3
0000004E  move-result-object  v3
00000050  invoke-direct       File-><init>(String)V, v2, v3
00000056  const/4             v3, 0
```

Packing Process

Hiding bytecode

Unpacking Process

Recovering bytecode

```
.method private export()String
        .registers 7
00000000  const/4             v0, 0
00000002  return-object       v0
00000004  nop
00000006  nop
00000008  nop
0000000A  nop
0000000C  nop
0000000E  nop
00000010  nop
00000012  nop
00000014  nop
00000016  nop
00000018  nop
0000001A  nop
0000001C  nop
0000001E  nop
00000020  nop
00000022  nop
00000024  nop
00000026  nop
00000028  nop
```

Android Unpackers

- Unpackers based on debuggers, e.g., Kisskiss [3].
  - Anti-analysis: detect ptrace.
- Unpackers based on emulators, e.g., DroidUnpack [4].
  - Anti-analysis: detect QEMU.
- Unpackers based on dynamic binary instrumentation (DBI) frameworks, e.g., ZjDroid [5], PackerGrind [6].
  - Anti-analysis: detect loaded binary files of DBI frameworks, …
- Unpackers based customized Android systems, e.g., AppSpear [7], TIRO [8].
  - Anti-analysis: hook system libraries, …

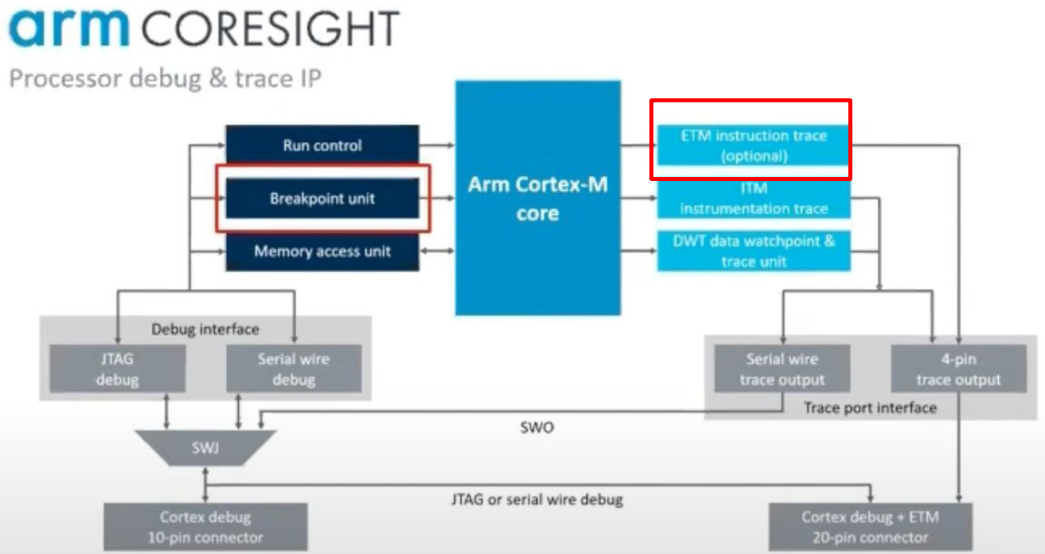
* Unfortunately, packed apps can easily detect and impede these software-based approaches.

Happer

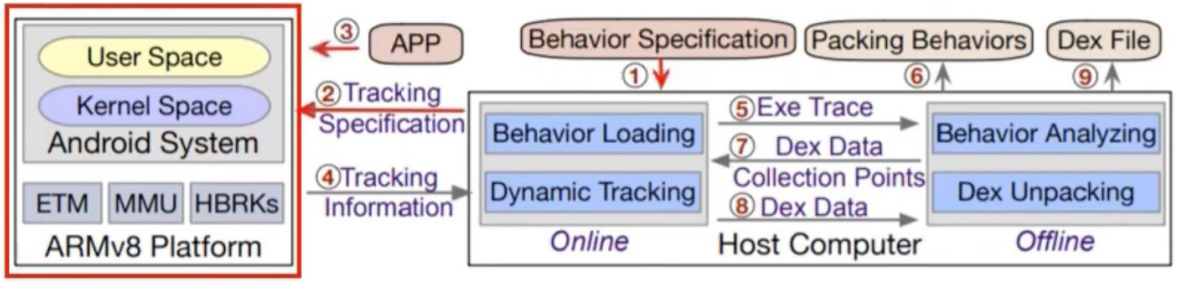
- A hardware-based Android app unpacker.
- Effectiveness
  - Target: retrieve hidden Dex data and assemble them to a Dex file.
  - Method: leverage the hardware features of the ARM platform.
- Adaptivity
  - Target: select proper unpacking strategies.
  - Method: select proper Dex data collection points according to packing behaviors.
- Extensibility
  - Target: support new packing behaviors.
  - Method: provide a domain-specific language.

Hardware Features

- Track Instructions
  - Embedded Trace Microcell (**ETM**): track the executed instructions.
- Fetch Memory Data
  - Hardware Breakpoints (**HBRKs**): interrupt normal execution.
  - Memory Management Unit (**MMU**): translate memory addresses.

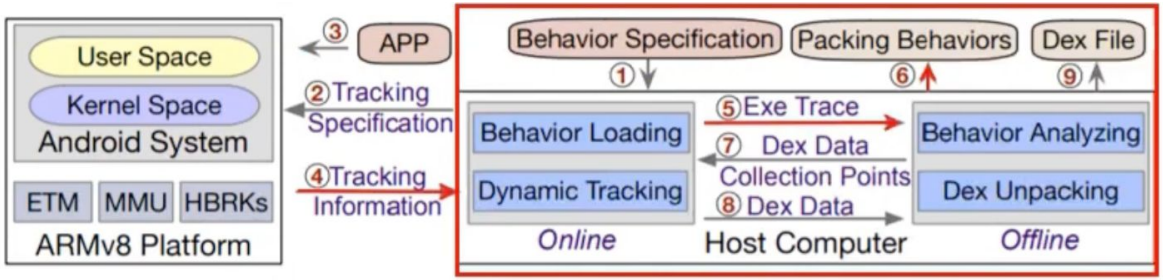


Happer



- Conduct dynamic tracking.
  - Take in the app and instructed by the input behavior specification.
  - Use ETM to record executed instructions and use HBRKs and MMU to retrieve the accessed memory data for determining packing behaviors.

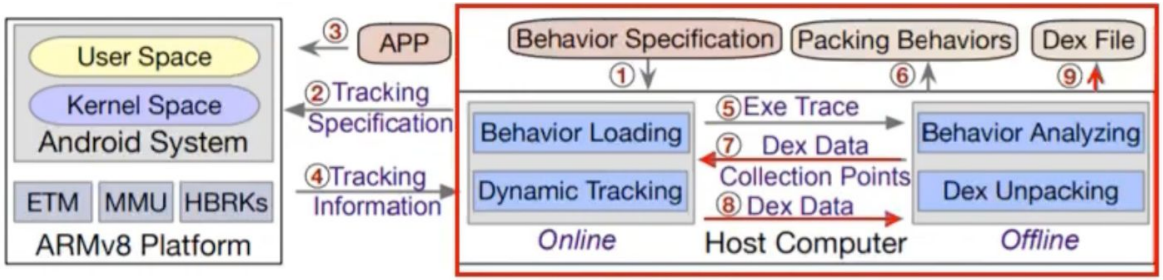
Happer



Address	Opcode		Detail
0xFFFFFF80081365C8		LDP	x6,x5,[x19,#8]
0xFFFFFF80081365CC		DMB	ISHLD
0xFFFFFF80081365D0		LDR	w2,[x20,#0]
0xFFFFFF80081365D4		CMP	w2,w22
0xFFFFFF80081365D8		B.NE	{pc}-52 ; 0xFFFFFF80081365A4
0xFFFFFF80081365DC		SUB	x0,x0,x6
0xFFFFFF80081365E0		MOV	w1,w1
0xFFFFFF80081365E4		AND	x0,x0,x5
0xFFFFFF80081365E8		LDP	x19,x20,[sp,#0x10]
0xFFFFFF80081365EC		MUL	x0,x0,x1
0xFFFFFF80081365F0		LDP	x21,x22,[sp,#0x20]
0xFFFFFF80081365F4		LSR	x0,x0,x4
0xFFFFFF80081365F8		ADD	x0,x0,x3
0xFFFFFF80081365FC		LDP	x23,x24,[sp,#0x30]

- Analyze packing behaviors.
- Take in ETM stream and memory data.
- Resolve ETM stream to recover runtime method invocations and then use them and dumped memory data to determine packing behaviors.

Happer




```
.method private export()String
    .registers 7
    invoke-direct       Tasks->getRangeName()String, p0
    move-result-object  v0
    new-instance        v1, StringBuilder
    invoke-direct       StringBuilder-><init>()V, v1
    invoke-virtual      StringBuilder->append(String)StringBuilder, v1, v0
    const-string        v2, ".csv"
    invoke-virtual      StringBuilder->append(String)StringBuilder, v1, v2
    invoke-virtual      StringBuilder->toString()String, v1
    move-result-object  v1
    new-instance        v2, File
    new-instance        v3, StringBuilder
    invoke-direct       StringBuilder-><init>()V, v3
    const-string        v4, "/sdcard/"
    invoke-virtual      StringBuilder->append(String)StringBuilder, v3, v4
    invoke-virtual      StringBuilder->append(String)StringBuilder, v3, v1
    invoke-virtual      StringBuilder->toString()String, v3
    move-result-object  v3
    invoke-direct       File-><init>(String)V, v2, v3
    const/4             v3, 0
```

- Unpacking packed apps.
- Select Dex data collection points.
- Load app methods to collect hidden bytecode.
- Assemble the retrieved Dex data to a valid Dex file for unpacking.

Challenges and Solutions

- Tracking Executed Instructions
  - Issue: insufficient on-chip buffer (64KB) and irrelevant ETM information.
  - Solution: use a dedicated hardware and trace necessary instructions.

- Resolving Instruction Trace
  - Issue: gap between high-level semantic information and tracked instructions.
  - Solution: map instructions to called functions.

- Fetching Memory Data
  - Issue: required data is loaded into physical memory asynchronously.
  - Solution: force OS to load memory data (MMU).

Packing Behaviors

- Happer supports the identification of *27* behaviors in *10* categories.

Category of Packing Behavior	Description
Anti-Debugging (**ADG**)	Checking fingerprints of debuggers.
Anti-Emulator (**AEU**)	Checking fingerprints of emulators.
Anti-DBI (**ADI**)	Checking fingerprints of DBI frameworks.
Time Checking (**TCK**)	Checking time delays incurred by dynamic analysis tools.
System Library Hooking (**SLH**)	Hooking debug/unpack related system library methods.
Dynamic Dex File Loading (**DDL**)	Loading protected Dex files at runtime.
Dynamic Dex Data Modification (**DDM**)	Modifying content of protected Dex files.
Dynamic Object Modification (**DOM**)	Modifying relevant ART runtime objects.
Dex Data Fragmentation (**DDF**)	Dispersedly loading content of protected Dex files.
JNI Transformation (**JNT**)	Translating an app's functions to the native code.

Packing Behaviors: Anti-Emulation

```
01 // get the first parameter value of the fopen function in libc.so
02 var fopenArg1 <- getArgValue("libc::fopen", 1);
03 // get the first parameter value of the strncmp function in libc.so
04 var strncmpArg1 <- getArgValue("libc::strncmp", 1);
05 // get the second parameter value of the strncmp function in libc.so
06 var strncmpArg2 <- getArgValue("libc::strncmp", 2);

07 // if the following two conditions are satisfied, AEU-1 is found
08 var detected <- 0;
09 if (fopenArg1 == "/proc/tty/drivers"
    && (strncmpArg1 == "goldfish" || strncmpArg2 == "goldfish")) {
10     detected <- 1;
11 }
```

Evaluation

- 12 commercial packers.
  - Including Ali, Baidu, Ijiami, Qihoo, Tencent …
- More than 24,000 benign apps.
  - 1,710 packed apps are found.
  - All of them have bytecode-hiding behaviors.
- More than 1,700 malware.
  - 214 packed apps are found.
  - Most of them have anti-analysis behaviors.

Evaluation: Can Happer identify more behaviors?

Behaviors of commercial packers identified by Happer.

2016					2018						
Baidu	Ijiami	Qihoo	Tencent	Ali	Baidu	Ijiami	Qihoo	Tencent	Bangcle	Kiwi	Testin
ADG-1	ADG-1/3-6	–	ADG-2	–	ADG-1	ADG-1/3-6	–	–	ADG-2	–	–
–	AEU-1/2/3	–	–	–	–	AEU-1/2/3	–	–	–	–	–
–	ADI-1	–	–	–	–	ADI-1	–	–	–	–	–
–	TCK-1	TCK-2	–	–	–	TCK-1	TCK-2	–	–	–	TCK-1
–	SLH-1/3	–	–	–	–	SLH-1/3	–	–	SLH-1/2	–	–
DDL-1	–	DDL-2	DDL-2	DDL-2	DDL-1	–	DDL-2	DDL-2	DDL-3	–	DDL-3
DDM-1	DDM-4	–	DDM-1	DDM-2/3	–	–	–	–	–	–	–
–	–	–	–	–	–	DOM-1	–	–	–	DOM-2	–
DDF-1	–	–	–	–	–	DDF-2	–	–	–	DDF-2	–
JNT-1	–	JNT-1	–	–	JNT-1	–	JNT-1	–	–	–	JNT-1

27 different packing behaviors are monitored by Happer.

Behaviors identified by others.

Other Unpackers[1]					
DU-18	TR-18	AS-18	PG-17	AS-15	DH-15
–	–	ADG-2	ADG-2	–	ADG-2
–	–	–	–	–	–
–	–	–	ADI-1	–	–
–	–	TCK-1	TCK-1	–	–
SLH-2	–	–	–	–	–
DDL-*	DDL-2	DDL-*	DDL-*	DDL-*	DDL-*
DDM-*	DDM-3	DDM-*	DDM-1/3	DDM-1/4	DDM-*
–	DOM-1	–	–	–	–
–	–	DDF-*	–	DDF-*	–
–	JNT-1	JNT-1	JNT-1	–	–
3	4	6	7	4	3

Evaluation: Effectiveness of Unpacking

- Completely recover apps packed by 7 commercial packers.
  - Ali-16, Bangcle-18, Ijiami-16/18, Tencent16/18, and Kiwi-18.
  - Ijiami-18 and Kiwi-18 cannot be recovered by existing unpackers because they release hidden bytecode when the app methods are loaded.
- Partially recover apps packed by 5 commercial packers.
  - Baidu-16/18, Qihoo-16/18, and Testin-18.
  - They reimplement specific app methods using native code and thus packed bytecode will never be released to memory.

Evaluation: Can Happer facilitate Malware Analysis?

The average number of sensitive APIs found in packed/unpacked malware.

Packers	Ali	Baidu	Bangcle	Ijiami	Kiwi	Qihoo
$\#App$	5	80	45	49	7	28
PS_{avg}	0.0	0.0	0.5	0.7	5.9	0.6
RS_{avg}	11.6	10.9	4.3	7.4	8.6	14.6
ΔS_{avg}	+11.6	+10.9	+3.8	+6.7	+2.7	+14.0

* Almost no sensitive APIs are found in the malware packed by Ali, Baidu, Bangcle, Ijiami, and Qihoo.

* Many sensitive APIs are found in the unpacked bytecode.

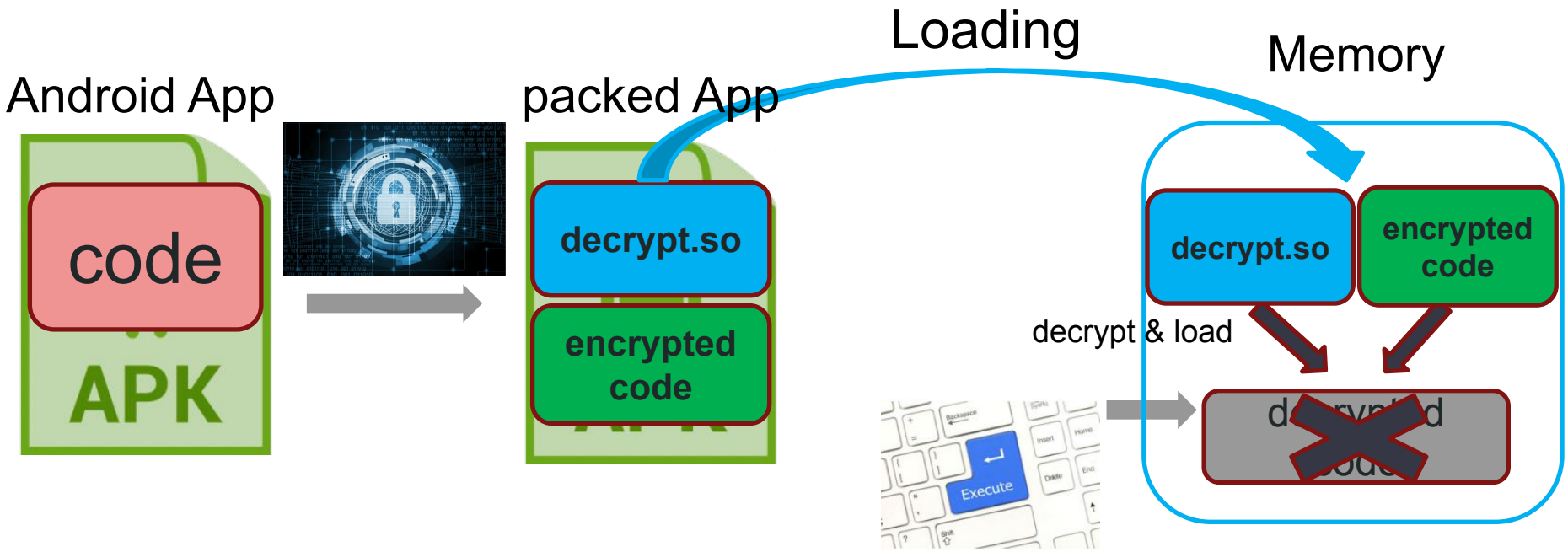
DeepVMUnpack: Neural Network-based Semantic Recovery from VM-Protected Android Apps for Malware Detection

Xin Zhao, Mu Zhang, Xiaopeng Ke, Yu Pan,

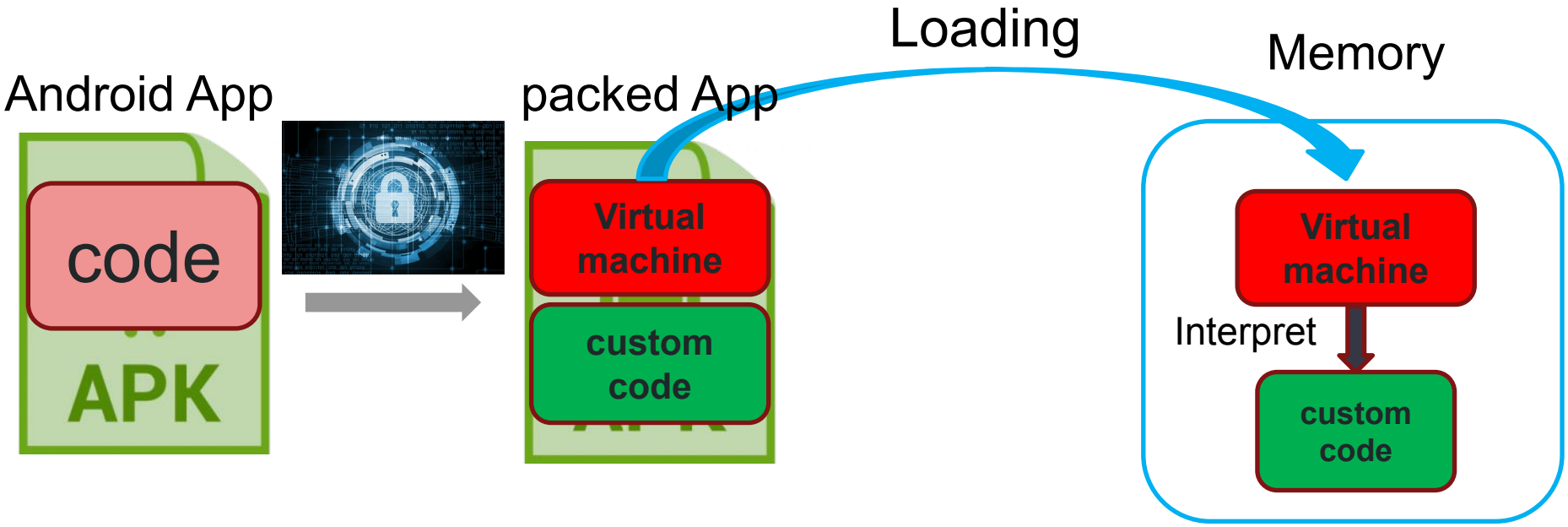
Yue Duan, Jun Xu, Sheng Zhong, Fengyuan Xu

Under Submission

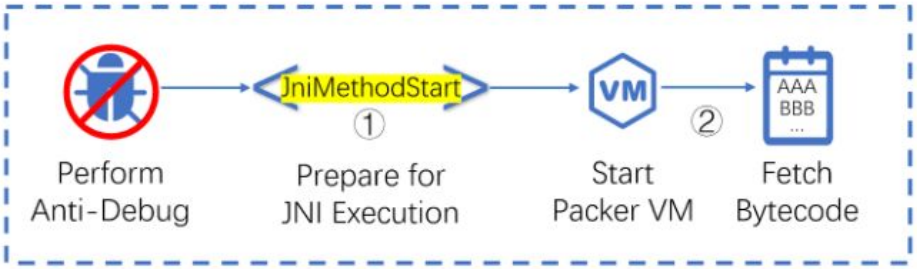
Android packing



Android VM-packing



VM Execution



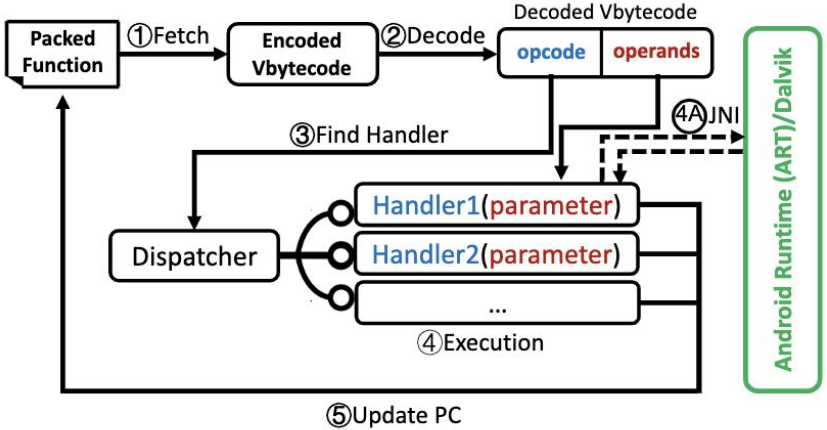
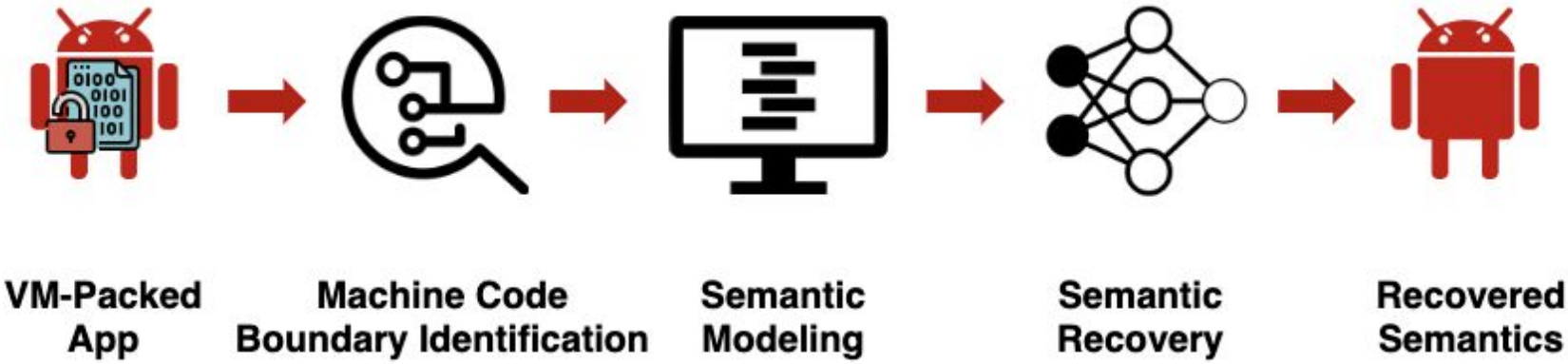
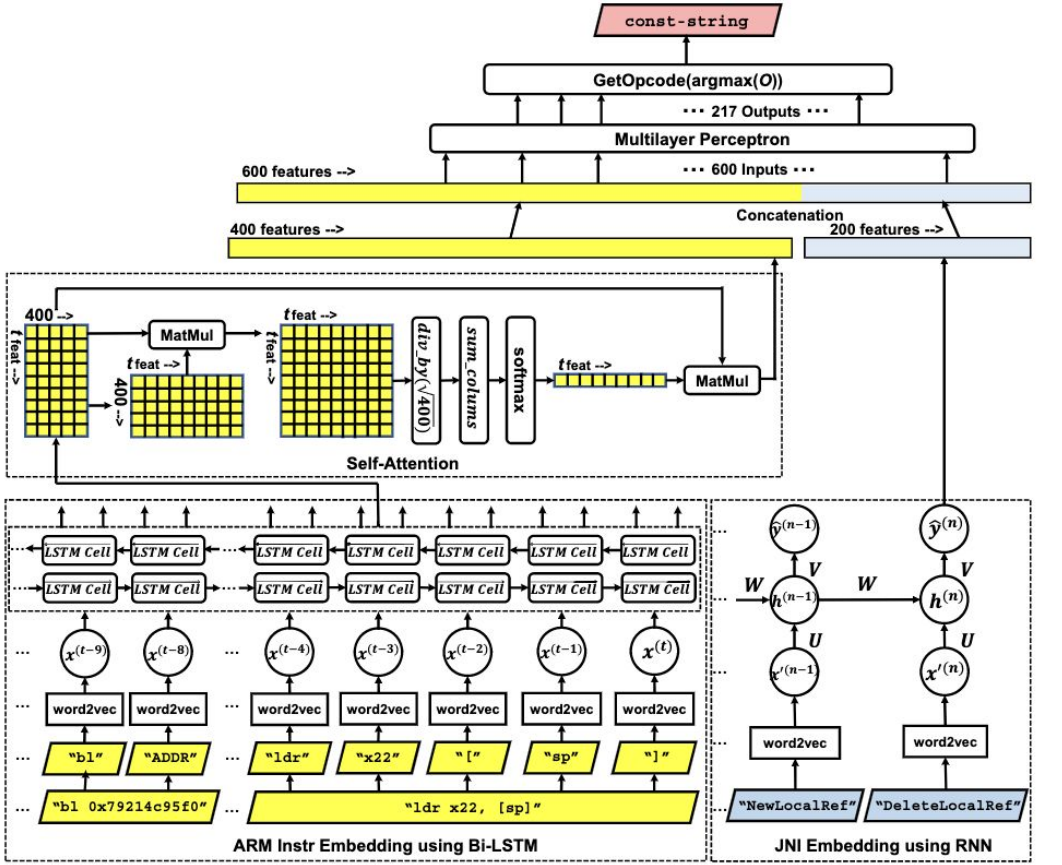


Figure 2: Execution Flow of Packer VM

DeepVMUnpack Overview

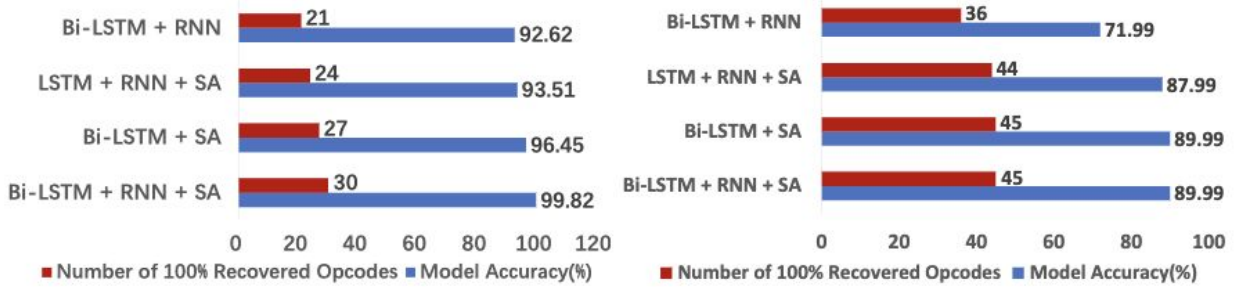


Deep Learning Model



Evaluation

	Accuracy	Precision	Recall	F1 Score
DEEPVMUNPACK	83.53%	50.63%	70.18%	14.71%
Parema	26.76%	13.08%	59.65%	5.36%



(a) Opcodes w/ JNIs (b) Opcodes w/o JNIs

Figure 10: Opcode-Level Accuracy for Different Models. Blue bars represent recovery accuracy; red bars indicate the amounts of opcodes that can be 100% recovered.

Thank you!

Question?