THIS IS NOT GOING TO END

THE EVOLUTION OF ANDROID APP PACKING AND UNPACKING TECHNIQUES

YAJIN ZHOU

ABOUT ME

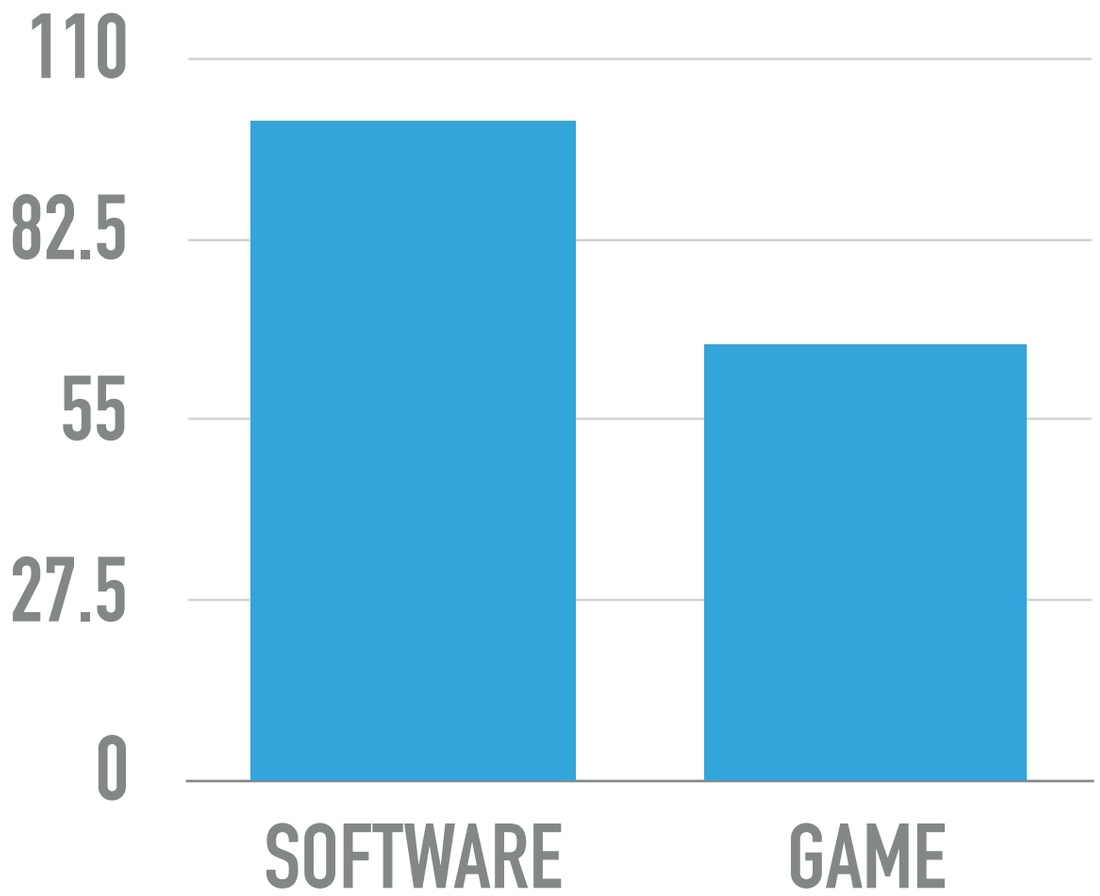
- Earned P.h.D. in Computer Science from NC State University
- Research mainly focuses on smartphone and system security
- Now involving in C0RE Team
- More information: http://yajin.org

AGENDA

- Why app packing services are becoming popular
- The main app packing/unpacking techniques
- New trends

APP REPACKAGING

- Given10,305 popular apps, 954,986 repackaged apps are found*



*Datasource Link

THE CONSEQUENCES OF APP REPACKAGING

- Developers
- Users

How easily to repackage an app?

Video Demo

APP PACKING SERVICE PROVIDERS

360加固保
jiagu.360.cn

应用加固
反静态分析，反动态调试，
反内存窃取，反恶意篡改...

阿里聚安全
应用更安全，用户更安心！

抗盗版神器
腾讯云应用加固
渠道监控 · 加固保护 · 漏洞检测
上传检测文件

FREE SERVICE

梆梆安全
BANGCLE

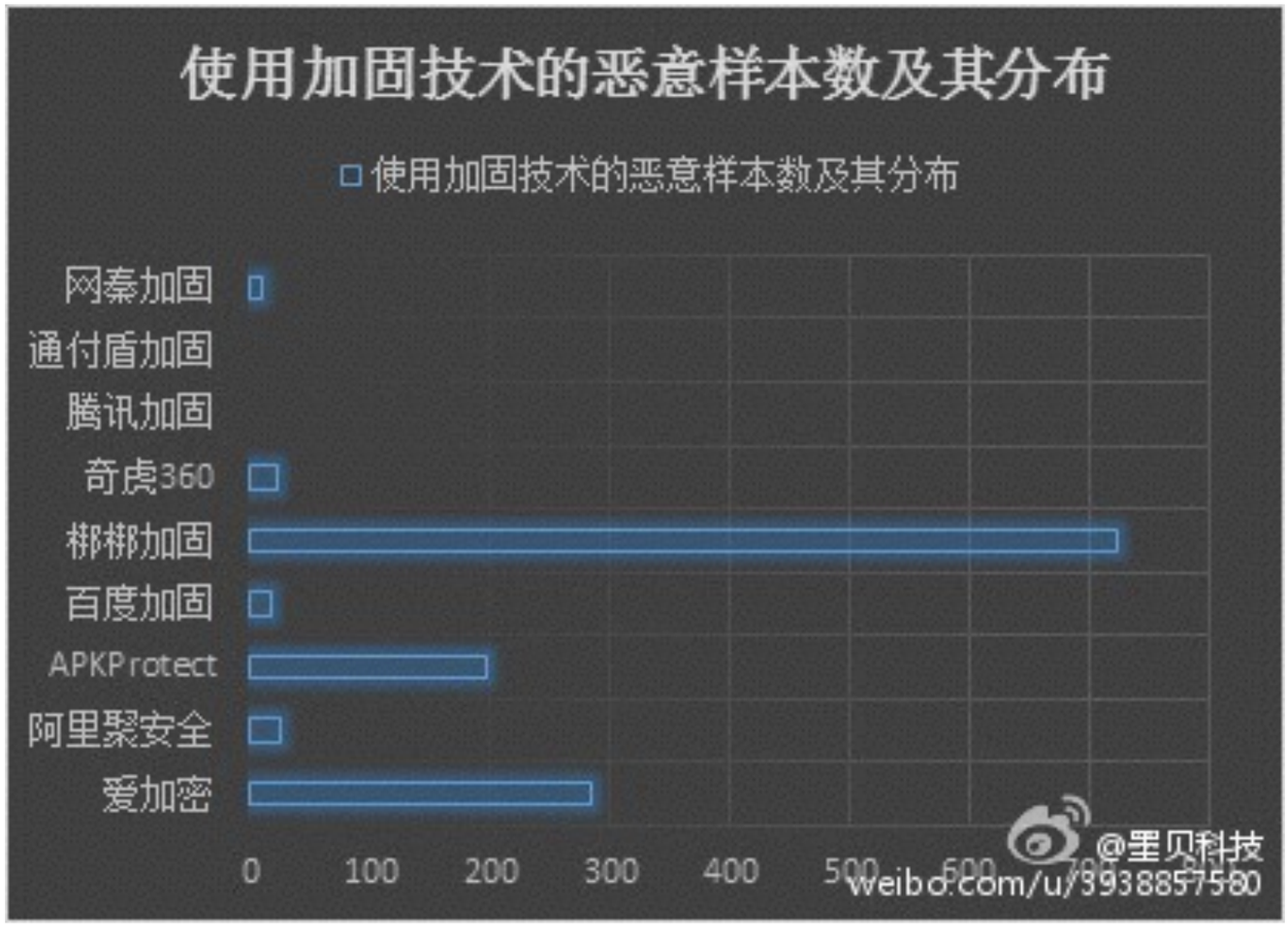
爱加密

DOUBLE-EDGED SWORD

- Packing services create problem for both good and bad guys
  - Bad guys: malware authors, (重)打包党
    - hard to repackage popular apps
  - Good guys: app markets maintainers, security researchers(??) …

IN REALITY

- App packing services are abused by bad guys



*Datasource Link

MAIN TYPES OF APP PACKING TECHNIQUES

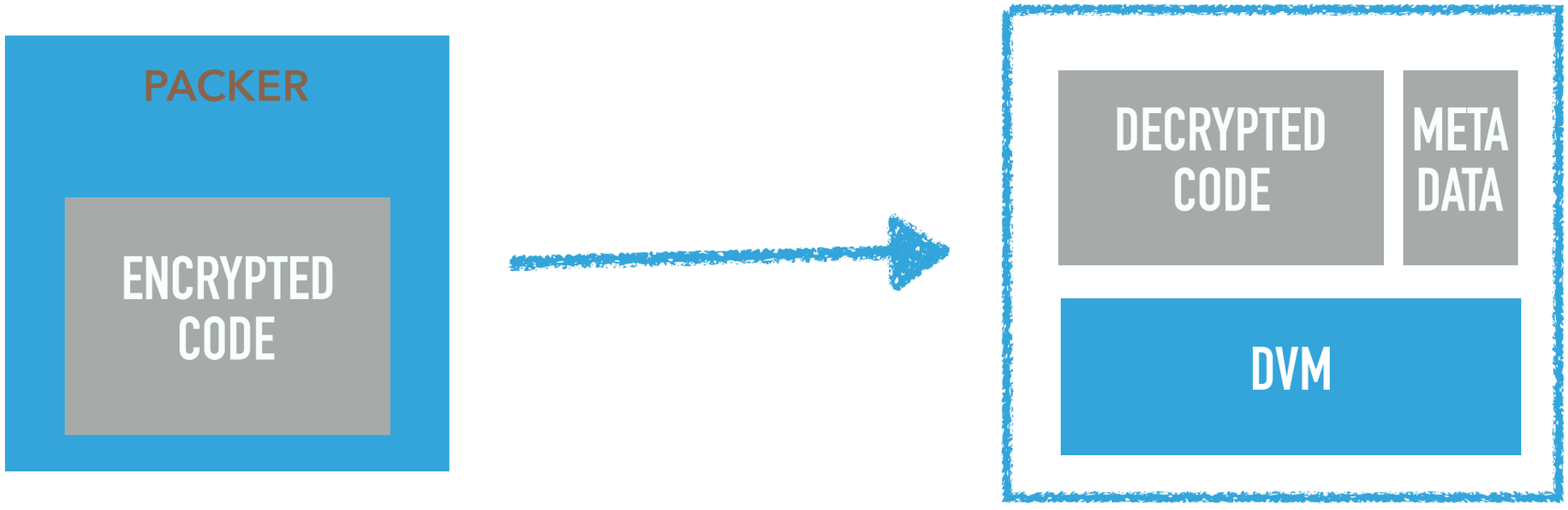
- Static: cheat statical analysis tools
- Dynamic
  - Memory dex loading: directly load encrypted dex file into memory and execute
- Anti-analysis: raise the bar for dynamic analysis

MAIN TYPES OF APP UNPACKING TECHNIQUES

- Static: reverse engineer the encryption algorithm
  - Pros: one method to kill all samples – protected by one packer
  - Cons: hard – usually the encryption algorithm is in the native code, and continuously changing

MAIN TYPES OF APP UNPACKING TECHNIQUES

- Dynamic: memory dump
  - Basic idea: the unencrypted bytecode will be eventually in memory
- Lack of self-modifying (and JITed bytecode) support



App Packing Techniques: Static

MANIFEST CHEATING

- Manifest file: define package name, permissions, components …
- When parsed, attributes are translated into ids
- If we insert an id to represent an undefined Java class
  - aapt: ignore this
  - apktool: honor this-> app repackaged by apktool will crash due to unimplemented Java classes

FAKE ENCRYPTION

- Apk file is indeed a normal zip file
- Set the encryption flag to true
- Old Android system does ***NOT*** check this flag, but static analyst tool does

BYTECODE-OBFUSCATION

- Depends on the disassembly algorithm
  - Linear
  - Recursive

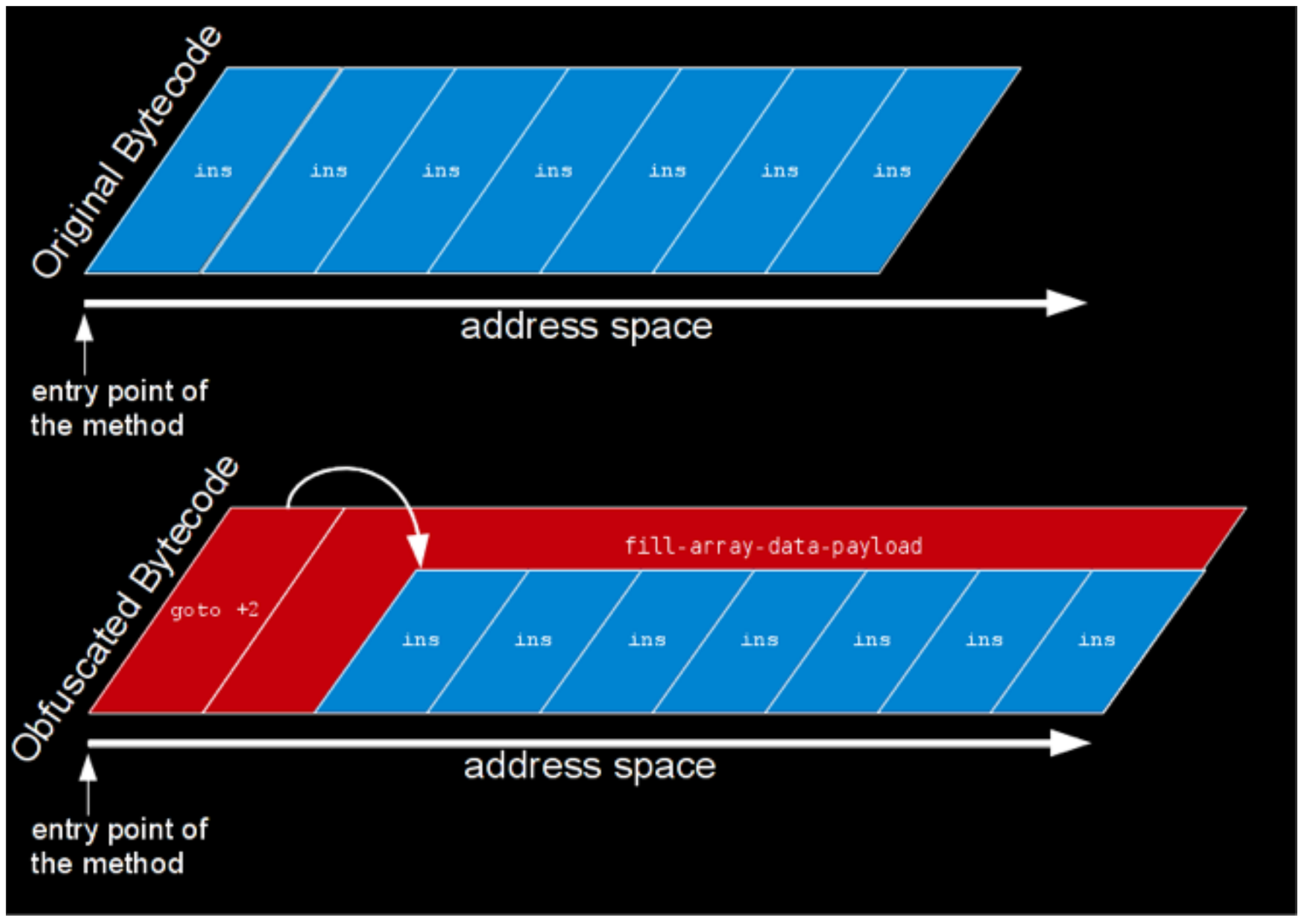


Image source: DexLabs

(LONG)FILE-NAME TRICKS

- Limited length of a file name

- 超长名字
  oooooooooooooo...
- 找茬
  Oo0o0OO00oooOOo0oo
  ijijijjiiiJiIIjii
- __$$_$$$$__$$_
- java语法关键字
  int int = 5;

- Unicode
- java \u0237
- CJK字符
- 难以阅读字符
  დამწერლობა
  אֶלֶף־בֵּית עִבְרִי
- 盲文点字模型
  2800-28FF

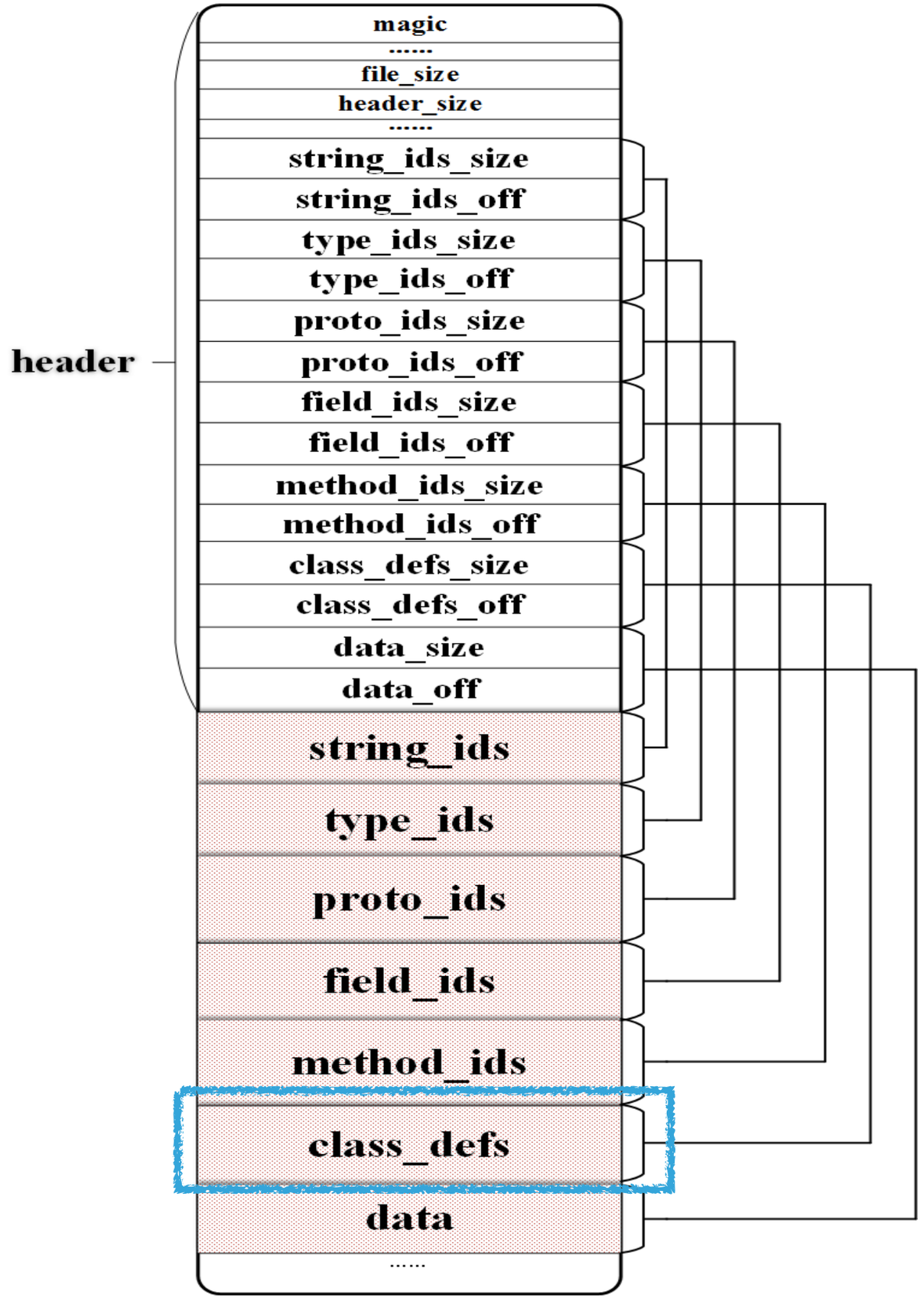
OVERVIEW

- Pros: easy to implement, better compatibility, low performance overhead
- Cons: easy to be bypassed,
- Small tricks, not a systematic way to protect app

App Packing Techniques: Dynamic

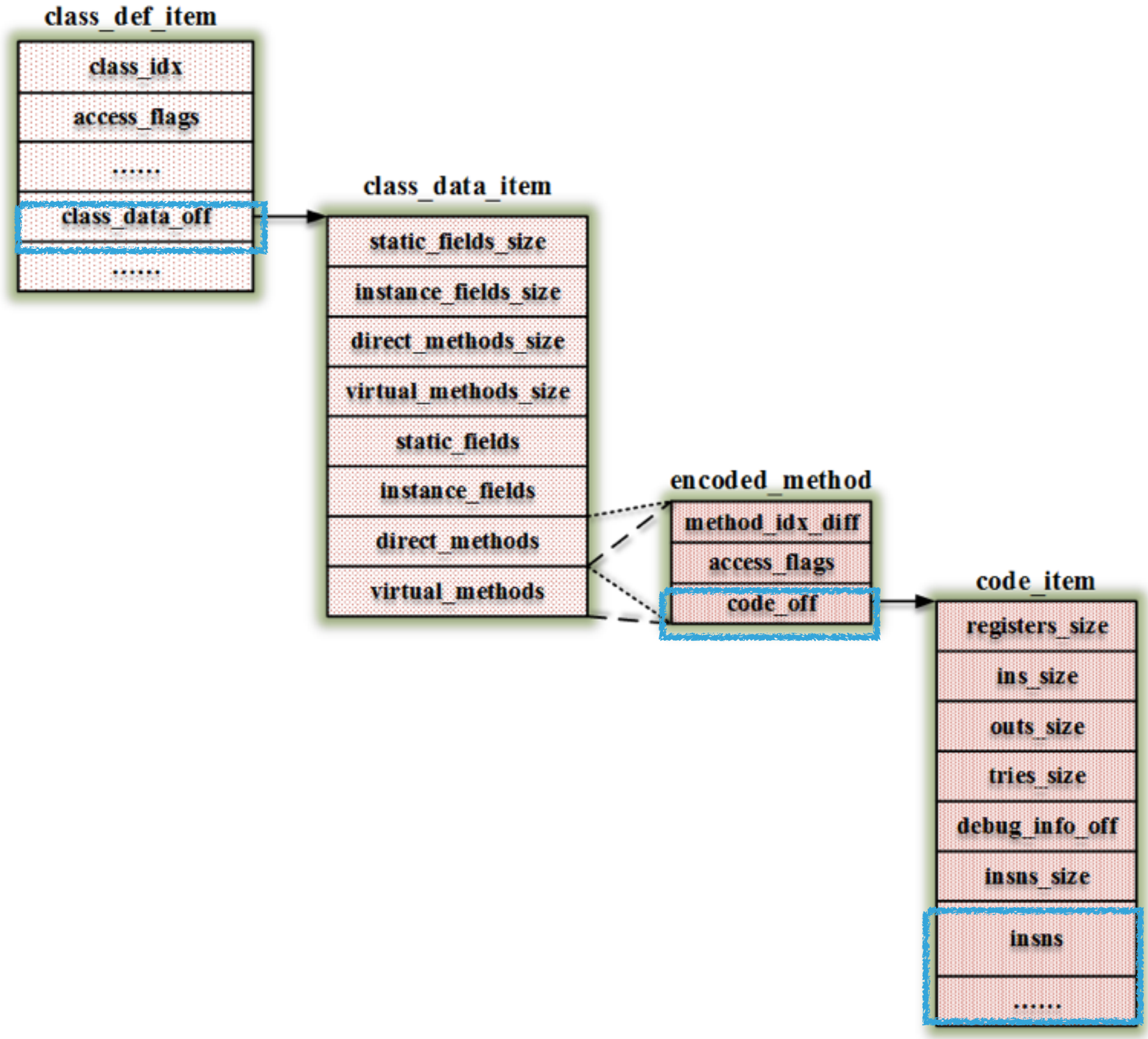
BACKGROUND

- Dex Header



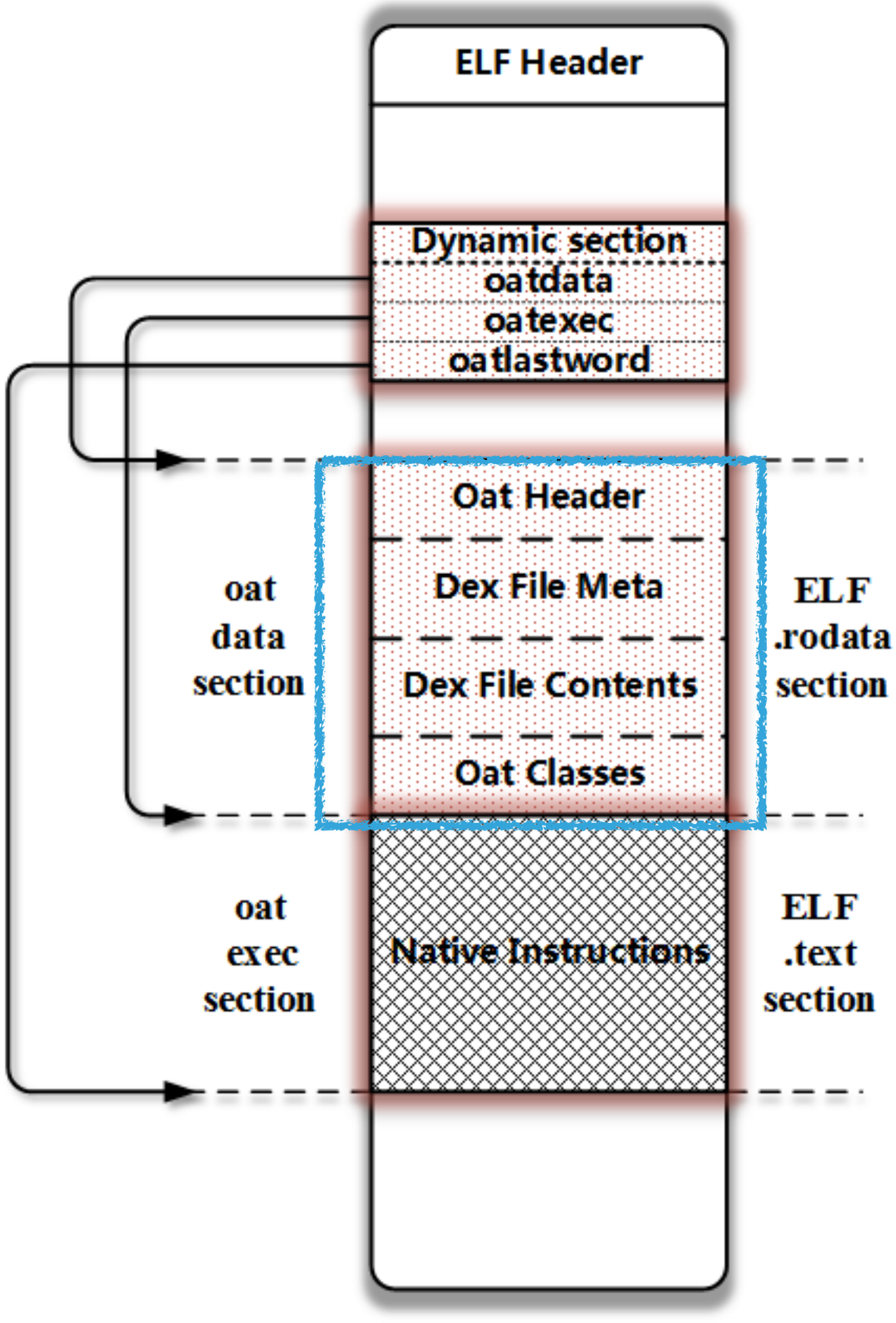
BACKGROUND

- class_def

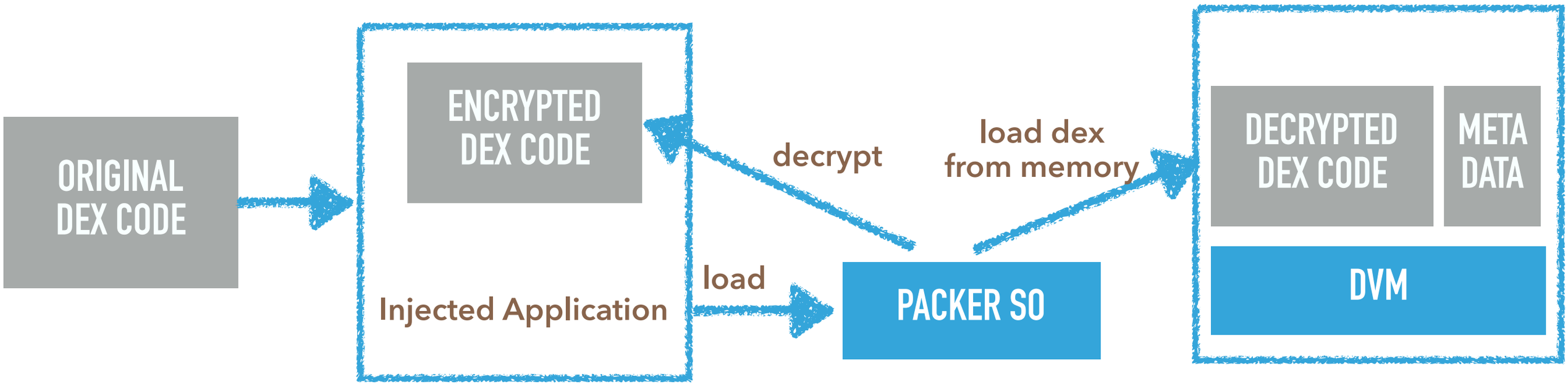


BACKGROUND

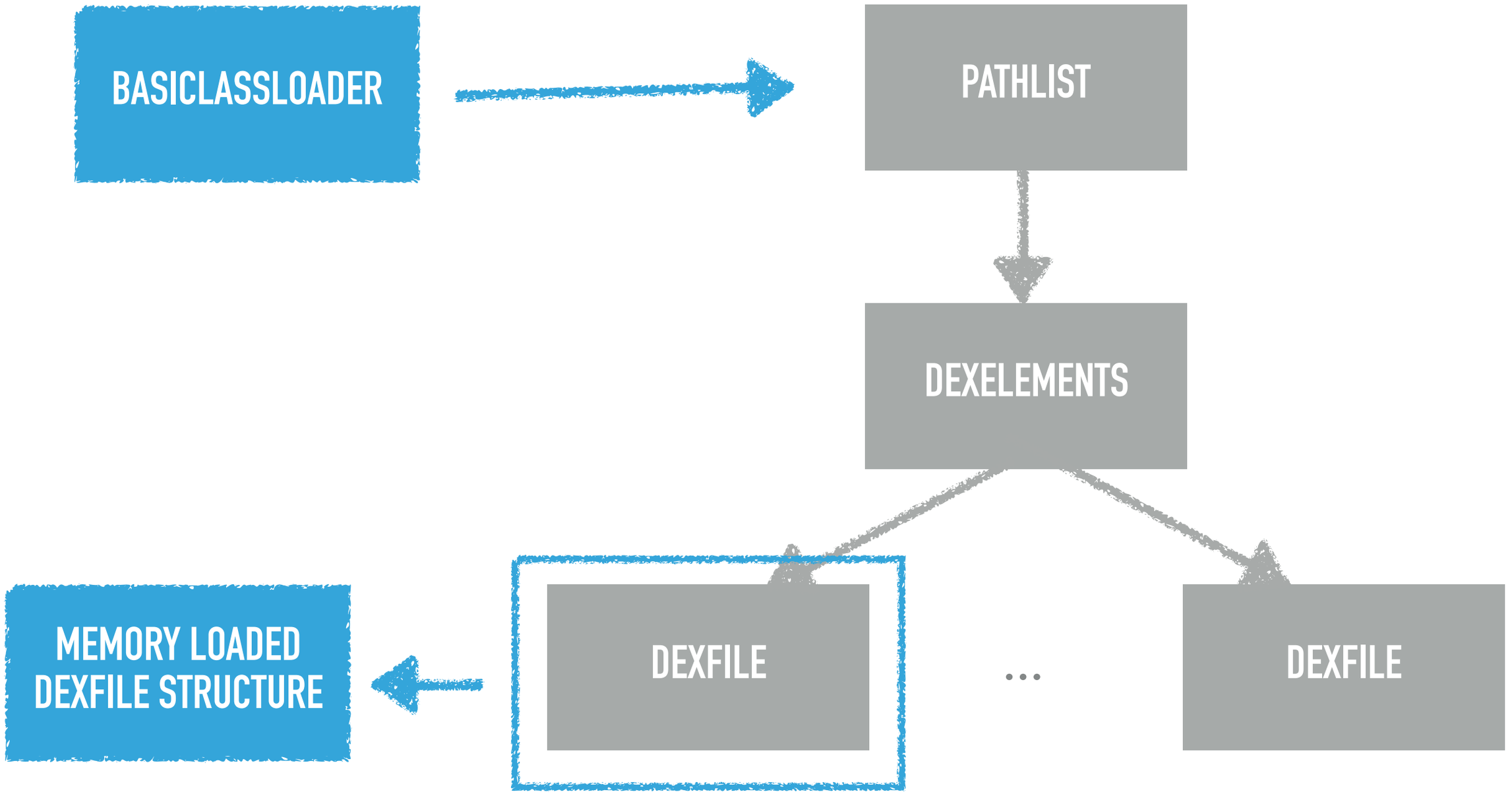
- Original dex file is embedded in the OAT file



THE BASIC IDEA OF APP PACKING



APP PACKING: DALVIK



APP PACKING: ART

- OAT file can still be executed in the interpreter mode – cost: performance loss
  - The embedded dex file
- Dex2oat is responsible for translating dex file into oat file when the app is being **installed**

APP PACKING: ART (CONTINUED)

- Propose I: run the app in the interpreter mode
- How
  - Create an empty dex file (with all classes but empty methods – real methods are encrypted) and the corresponding oat file will be created
  - Decrypt the real methods and make up the empty method structure in memory

APP PACKING: ART (CONTINUED)

- Propose II: Encrypt the generated oat file
- How
  - Hook the dex2oat tool: encrypt the oat file (LD_PRELOAD)
  - oat memory loading: Android fragmentation - Android L/M, custom ROMs

APP PACKER: PROTECT THE PACKER ITSELF

- Packer is usually in the format of so library
  - o-LLVM
  - upx
  - init functions
  - Based on custom so loader
  - VMP engine to protect key functions

App UNPacking Techniques: Static

APP UNPACKING: STATIC

- Understand the encryption/decryption logic of the packer
  - Pros: one effort to kill all (apps with one packer)
  - Cons: so packer (VMP engine), encryption method/key is continuously changing …
- But it is efficient if we have an insider

App UNPacking Techniques: Dynamic

THE KEY VULNERABLE POINT OF APP PACKING

- Dalvik VM
  - executes unencrypted dex code
  - requires the integrity of some meta data

点穴

RUNTIME MEMORY STATE

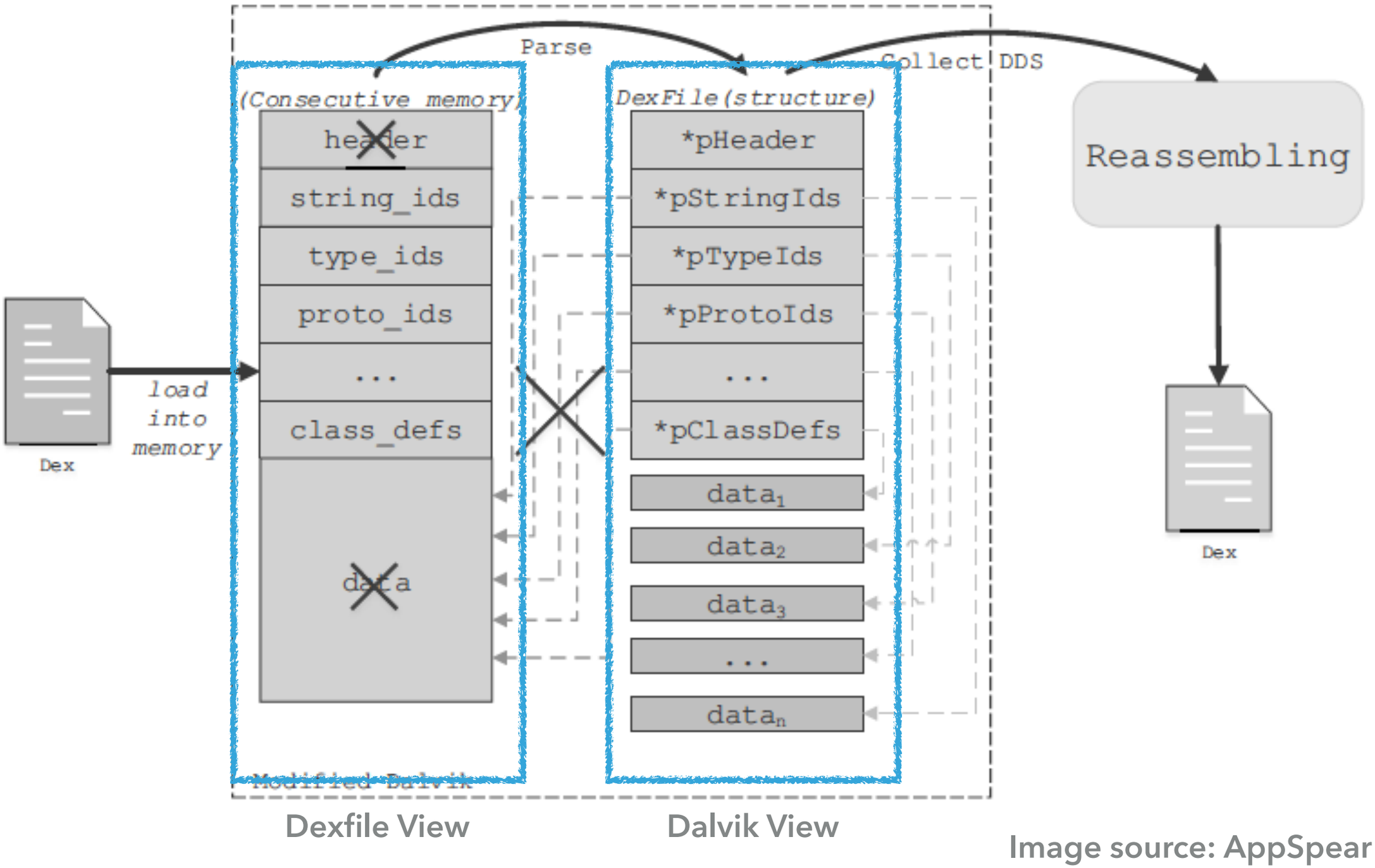


Image source: AppSpear paper

APP UNPACKING 101

- Basic idea: locate the dex file in memory and dump
- How: locate “dex. 035”
- When: hook key functions (mmap, strcmp and etc…)
- Countermeasure: corrupt the header, inline key functions

0000h:	64	65	78	0A	30	33	35	00	18	DF	1D	D9	C2	C8	7E	AA
0010h:	8D	5B	B4	03	BD	93	F1	8E	CB	A9	3D	78	03	B6	51	0B
0020h:	18	C4	2B	00	70	00	00	00	78	56	34	12	00	00	00	00
0030h:	00	00	00	00	B4	62	07	00	2B	4F	00	00	70	00	00	00
0040h:	77	0C	00	00	1C	3D	01	00	CD	11	00	00	F8	6E	01	00
0050h:	D0	2C	00	00	94	44	02	00	A0	51	00	00	14	AB	03	00
0060h:	55	09	00	00	14	38	06	00	64	61	24	00	B4	62	07	00

0000h:	dex.035..							ß	.	Ù	Â	È	~	ª
0010h:	.	[	´	.	½	"	ñ	Ž	Ë	©	=	x	.	¶	Q	.
0020h:	.	Ä	+	.	p	.	.	.	x	V	4	.	.	.	.	.
0030h:	.	.	.	.	´	b	.	.	+	O	.	.	p	.	.	.
0040h:	w	.	.	.	.	=	.	.	Í	.	.	.	ø	n	.	.
0050h:	Ð	,	.	.	"	D	.	.		Q	.	.	«	.	.	
0060h:	U	.	.	.	.	8	.	.	d	a	$	.	´	b	.	.

APP UNPACKING 102

- Basic idea: dump the memory and reconstruct the dex file ***without*** relying on the dex header – DexHunter, and AppSpear
- How: modify libdvm, dump memory, reconstruct dex

APP UNPACKING 102: DEXHUNTER

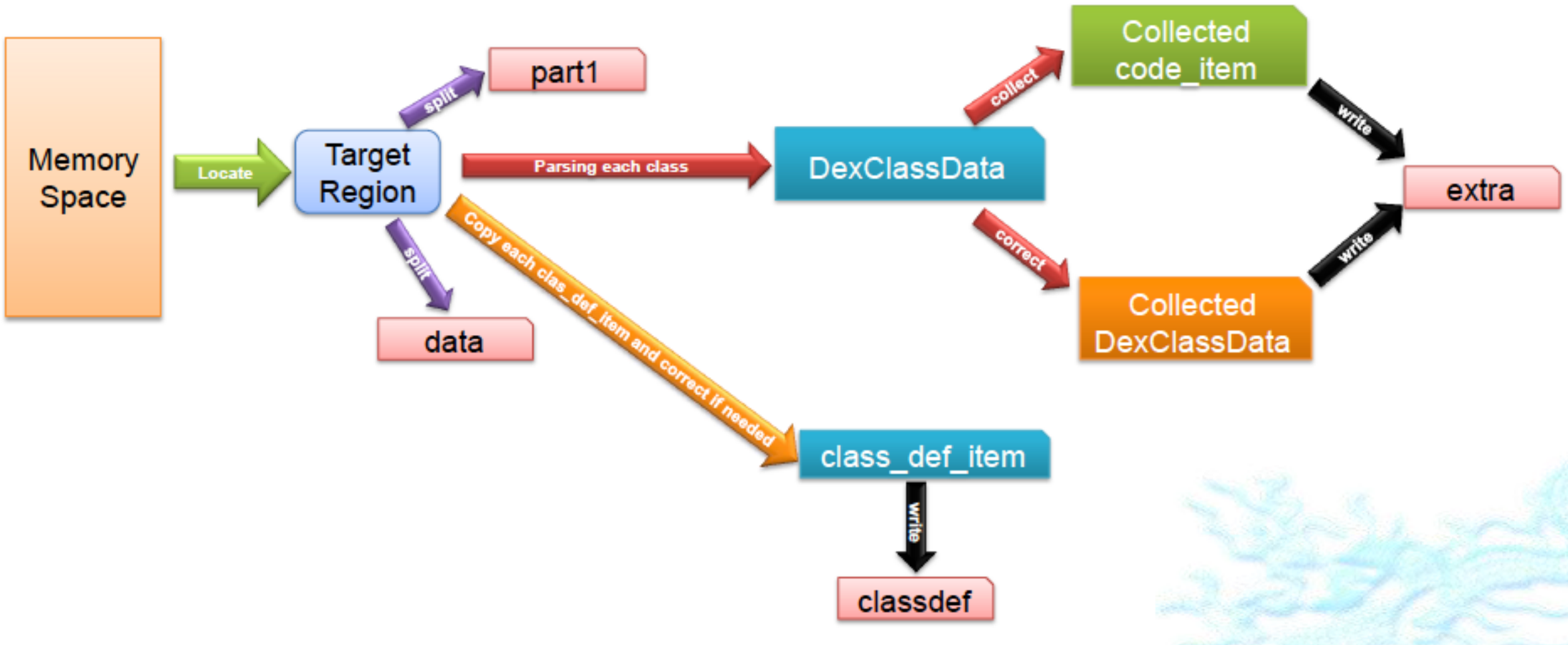


Image source: DexHunter HitCon2015 slides

APP UNPACKING 102: APPSPEAR

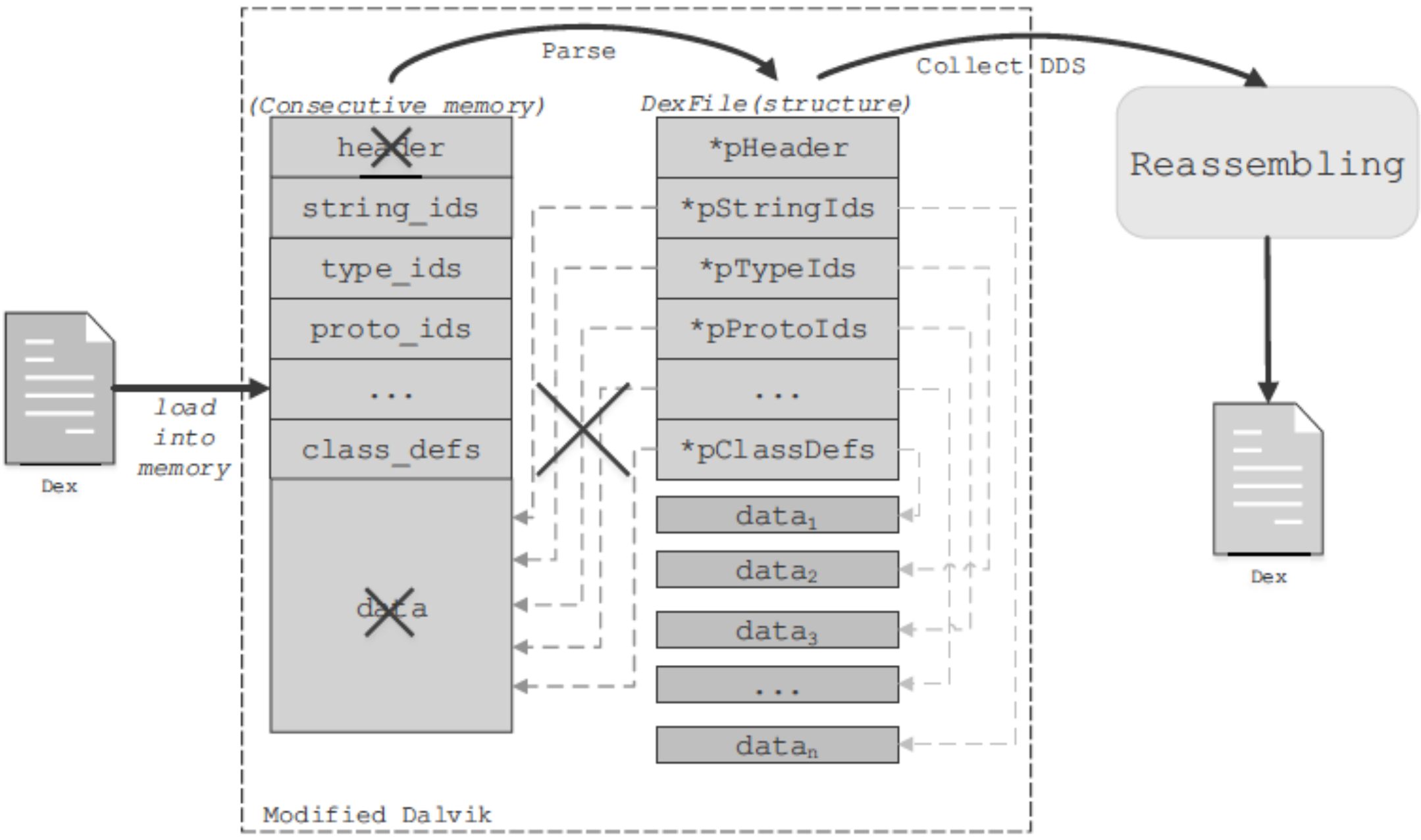


Image source: AppSpear paper

Countermeasures

INCREMENTAL UNPACKING

- When and where to refill these instructions?

struct encoded_method_list virtual_methods	1 methods
struct encoded_method method	protected void com.byd.aeri.caranywh
struct uleb128 method_idx_diff	0x3981
struct uleb128 access_flags	(0x4) ACC_PROTECTED
struct uleb128 code_off	0x15D174
struct code_item code	2 registers, 1 in arguments, 1 out a
ushort registers_size	2h
ushort ins_size	1h
ushort outs_size	1h
ushort tries_size	1h
uint debug_info_off	2843CAh
struct debug_info_item debug_info	
uint insns_size	9h
ushort insns[9]	
ushort insns[0]	0h
ushort insns[1]	0h
ushort insns[2]	0h
ushort insns[3]	0h
ushort insns[4]	0h
ushort insns[5]	0h
ushort insns[6]	0h
ushort insns[7]	0h
ushort insns[8]	Eh
ushort padding	0h
struct try_item tries	
struct encoded_catch_handler_list handlers	1 handler lists

ANTI-DISASSEMBLY

- Change the value of debug_info_off

struct encoded_method method[2]	
struct uleb128 method_idx_diff	0x1
struct uleb128 access_flags	(0x1) ACC_PUBLIC
struct uleb128 code_off	0xA40C0
struct code_item code	
ushort registers_size	3h
ushort ins_size	2h
ushort outs_size	2h
ushort tries_size	0h
uint debug_info_off	1031886Eh
struct debug_info_item debug_info	

ANTI-PTRACE/DEBUG

- Check files: /proc/$pid/status, etc …
- Check process name
- SIGTRAP
- Multi-process
- Inotify
- Hook read/write APIs

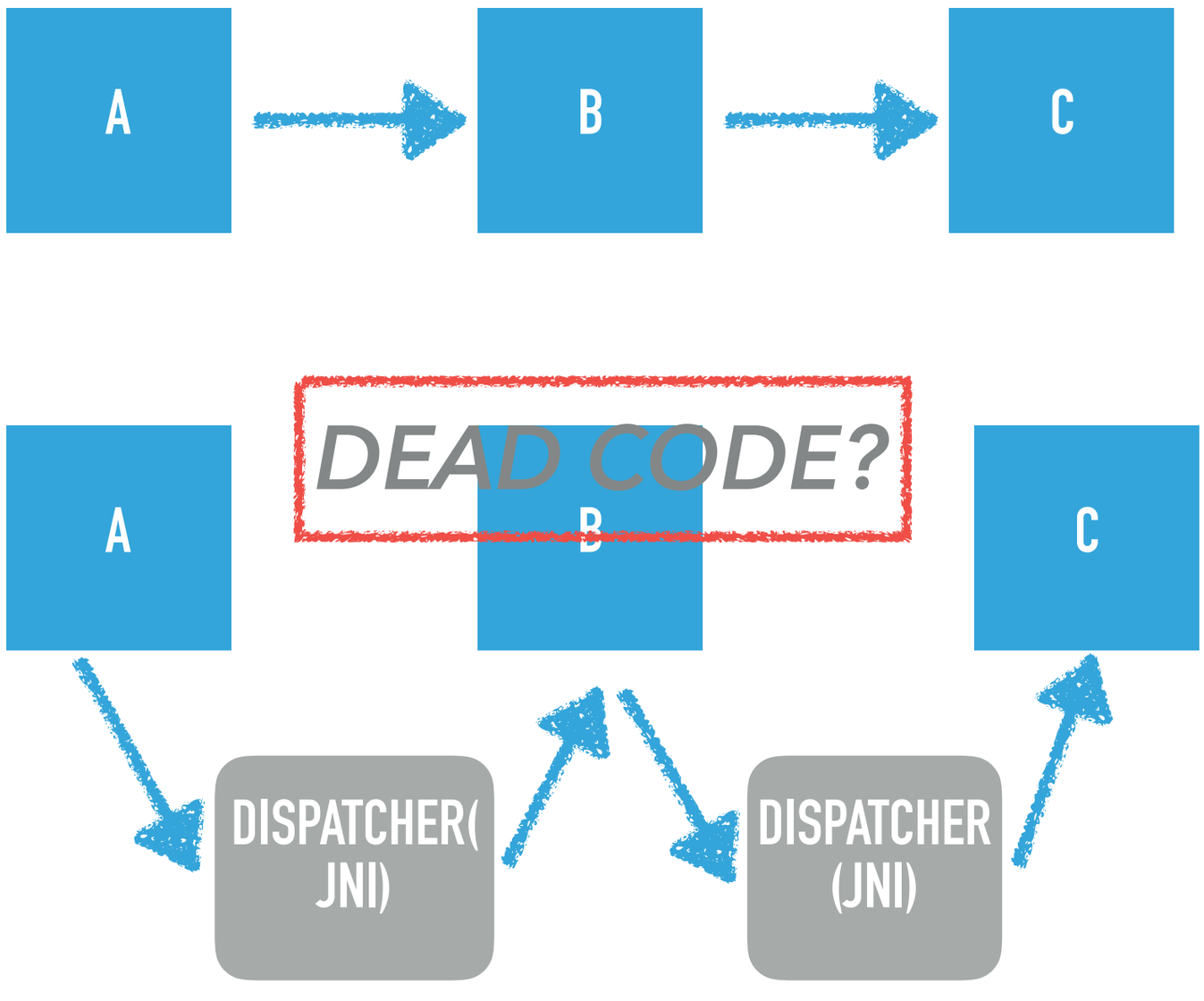
New Trends

DEX2NATIVE

- The dex code could be dumped from memory (as long as Dalvik is still used)
- Dex code could be recovered
- Native code is much harder to understand

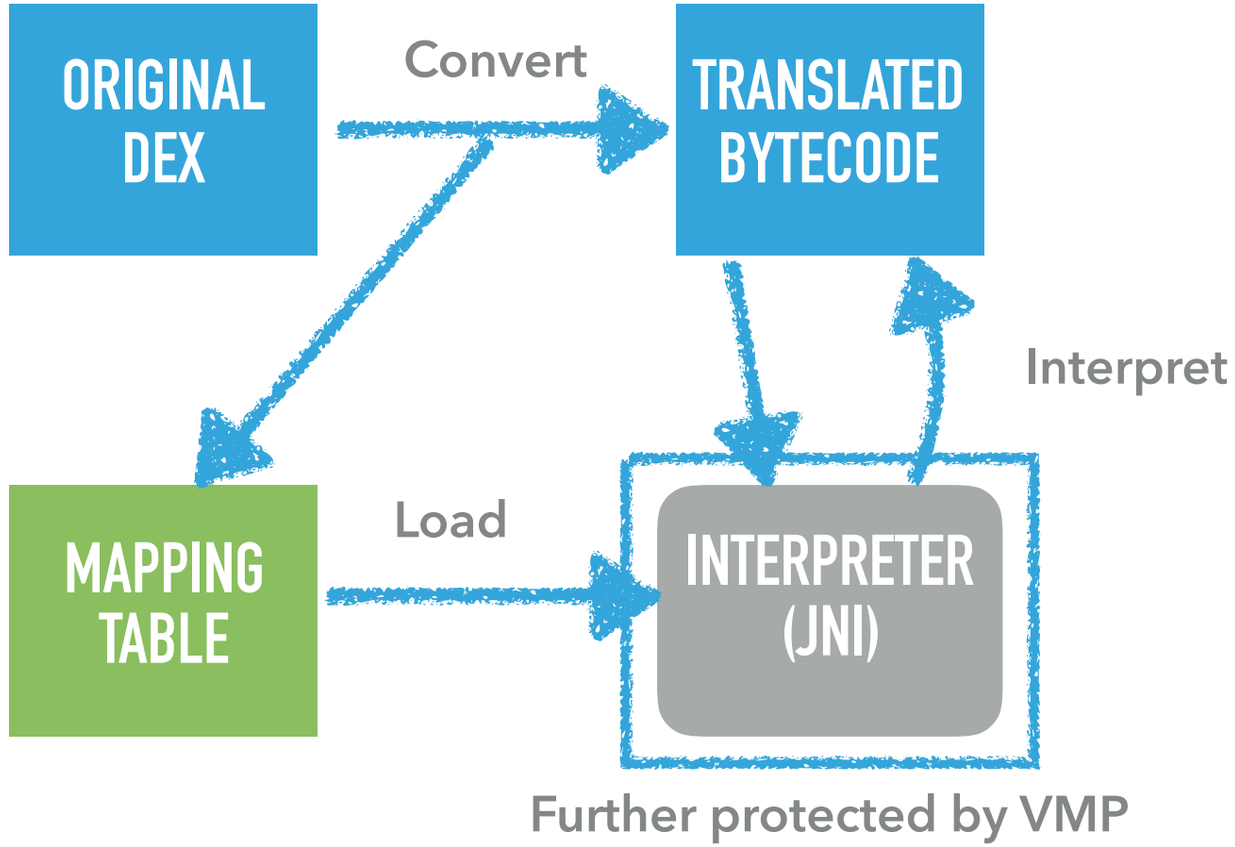
DEX2NATIVE: 101

- Hide the control flow



DEX2NATIVE 102

- Completely convert the bytecode to another format of bytecode: how to maintain the semantics of the bytecode?



DEX2NATIVE 102

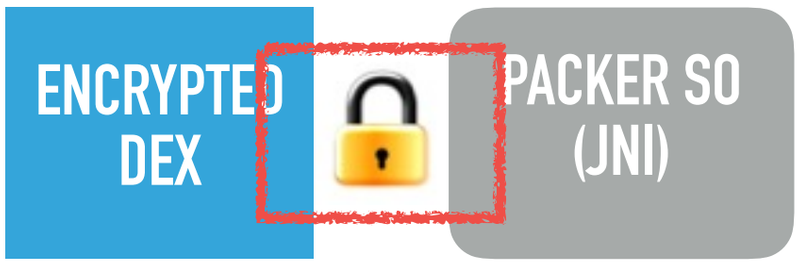
```
.class public Lcom/example/ApiTest/MyActivity;

.method protected native onCreate(Landroid/os/Bundle;)V

.end method
```

DEEP COUPLING

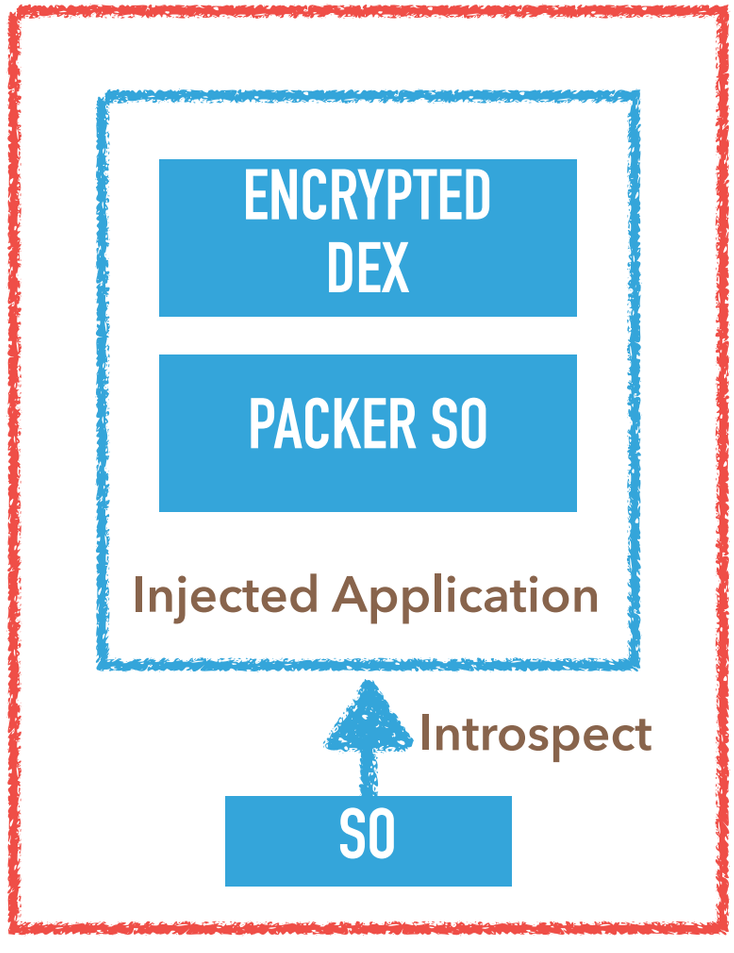
- Put some app logic into the packer– the dumped dex cannot be repackaged without the packer
- The packer checks the integrity of the dex code



FROM UNPACKING TO REPACKAGING

HOW TO RELIABLY REPACKAGE THE PACKED APP

- Load the packed app using the similar packing technique: Matrix?
- DroidPlugin



THANKS