DeNAゲームセキュリティの攻と守

DeNA
nevermoe

アジェンダ

- 自己紹介兼仕事紹介
- 攻：プラクティカルスマホゲームデバッギング
- 守：チート対策ソリューション

© DeNA Co., Ltd.

自己紹介

- nevermoe
- GitHub：https://github.com/nevermoe
- ツイッター：@nevermoecom
- 発表資料：https://www.nevermoe.com/docs/

© DeNA Co., Ltd.

仕事紹介

- 診断（メイン）
  - ❖ ゲーム
  - ❖ ウェブ
  - ❖ 他(IoTなど)
- チート対策（メイン）
  - ❖ ツール、ライブラリ開発
  - ❖ チート情報調査
- プラベートSOC（サブ）
- Log管理運営（サブ）

© DeNA Co., Ltd.

一般的な診断

- クライアントに重要ロジックがない（のが多い）
- ホワイトボックス＋ブラックボックス

© DeNA Co., Ltd.

ゲーム診断の特徴

- クライアントに重いロジックが多い
- ブラックボックス診断が多い
  - ❖ 開発が社外などソースコードもらえない場合がある（同意を得た上で診断）
  - ❖ ソースコード読んでも、最終バイナリでないと弱みが分からない

© DeNA Co., Ltd.

ゲーム診断のスキル

- リバーシング！
  - 静的解析
  - 動的解析
    - デバッギング<ー今日のテーマ
    - フッキング
    - などなど

© DeNA Co., Ltd.

（攻）スマホゲームデバッギング

- なぜデバッギング？
  - ❖ 静的分析が難しい！

© DeNA Co., Ltd.

（攻）スマホゲームデバッギング

- 何ができる？なんでも出来る！
  - ファイルの暗号化を解く
    - セーブデータ
    - ゲームアセット、リソース
    - シンボルデータ
  - 検知ロジックを特定
    - Root / 脱獄検知
    - デバッガ検知
    - コード改ざん検知
  - ゲームロジック特定
    - 通信ペイロード暗号化ロジック

TIPS:
hardware breakpointと
watchpointを活用しよう

© DeNA Co., Ltd.

一般的なデバッギング手法

- Android (Root化端末): ndk-gdb

```
// on Android
# setenforce 0
# ./gdbserver tcp:9090 --attach [pid]
```

```
// on PC
$ adb forward tcp:9090 tcp:9090
$ gdb
(gdb) set architecture arm
(gdb) target remote :9090
```

TIPS:

1. arm32のゲームが多くて、gdbserverの32bit版使った方がsymbolがロードされる
2. breakpointを設置前にset arm force-mode [thumb|arm|auto]

© DeNA Co., Ltd.

一般的なデバッギング手法

- iOS (脱獄端末): lldb

```
// on iOS
# /Developer/usr/bin/debugserver 0.0.0.0:9090 -a [pid]
```

```
// on PC
$ iproxy 9090 9090
$ lldb
(lldb) platform select remote-ios
(lldb) process connect connect://localhost:9090
```

TIPS:

1. iproxyを使ってポートフォワーディングするとUSBからiphoneに繋がる

出典：debugserver - iPhone Development Wiki (http://iphonedevwiki.net/index.php/Debugserver)

※弊社では別の方法で実施

© DeNA Co., Ltd.

早期デバッガアタッチ

- こういうシナリオありませんか？
  - ゲーム**起動直後**に**一回しか走らない**関数をデバッグしたい
    1. ゲーム起動直後にRoot検知 / Anti-debuggingなどでクラッシュ
    2. Packerを解析したい
    3. 起動時ファイルを復号（*例）

通常手法でデバッガをアタッチしても間に合わない

早期デバッガアタッチしよう！

© DeNA Co., Ltd.

早期デバッガアタッチ

- 愚直なやり方
  - 無限ループでプロセスの起動を待つ

```
# sh wait.sh 'your.package.name'
```

```
// wait.sh, pseudo code
while true
do
    PID=`pgrep $1`
    if [ -z $PID ]; then
        echo "wait..."
    else
       $GDB_PATH tcp:9090 --attach $PID
        break
    fi
done
```

これも基本間に合わない

soが全部ロードされてしまう状態

© DeNA Co., Ltd.

早期デバッガアタッチ

- Android (Root化端末)
  1. アプリをJavaデバッギングモードで起動
  2. native debug serverを起動 / アタッチ
  3. native debug clientからnative debug serverに繋ぐ
  4. jdbをアタッチする（プログラムを実行状態に戻す）

© DeNA Co., Ltd.

早期デバッガアタッチ

- Android (IDA Pro)

on PC

```
$ adb forward tcp:23946 tcp:23946
$ adb shell am start -D -n your.package.name/xxx.yyy.ActivityName
$ for pid in `adb shell "ps | grep your.package.name" | awk '{print $2}'`;
do adb forward tcp:8700 jdwp:$pid; done
```

on Android

```
# setenforce 0
# ./android_server
```

in IDA

1. IDA->Debugger->Attach->Remote ArmLinux/Android debugger
2. アタッチしたいプロセスを選ぶ

on PC

```
$ jdb -connect "com.sun.jdi.SocketAttach:hostname=localhost,port=8700"
```


© DeNA Co., Ltd.

早期デバッガアタッチ

- Android (ndk-gdb)

on PC

```
$ adb forward tcp:9090 tcp:9090
$ adb shell am start -D -n your.package.name/xxx.yyy.ActivityName
$ for pid in `adb shell "ps | grep your.package.name" | awk '{print $2}'`;
do adb forward tcp:8700 jdwp:$pid; done
```

on Android

```
# setenforce 0
# ./gdbserver tcp:9090 --attach [pid]
```

on PC

```
$ gdb
(gdb)set architecture arm
(gdb)target remote :9090
```

on PC
(another shell)

```
$ jdb -connect "com.sun.jdi.SocketAttach:hostname=localhost,port=8700"
```

TIPS:
IDA debuggerもバグりがちで、**ndk-gdb**を使った方が安定でしょう

© DeNA Co., Ltd.

早期デバッガアタッチ

- iOS (脱獄端末)

on PC

```
# /Developer/usr/bin/debugserver 0.0.0.0:9090 {APP_PATH}
```

on PC
(another shell)

```
$ iproxy 9090 9090
$ lldb
(lldb) platform select remote-ios
(lldb) process connect connect://localhost:9090
```

出典：debugserver - iPhone Development Wiki (http://iphonedevwiki.net/index.php/Debugserver)

※弊社では別の方法で実施

© DeNA Co., Ltd.

早期デバッガアタッチ

- iOS (入獄端末)

on PC

```
// 内製ツールapp-runner
$ ./app-runner -r {APP_PATH}
```

on PC
(another shell)

```
$ iproxy 9090 9090
$ lldb
(lldb) platform select remote-ios
(lldb) process connect connect://localhost:8080
```

原理：**imobiledevice**を使って**debugserver**を立ち上げ、さらにアプリを**debugserver**から立ち上げる。
https://github.com/libimobiledevice/libimobiledevice

*** 弊社では利用している手法**

© DeNA Co., Ltd.

早期デバッガアタッチ

- *例：起動時ファイルの復号
  - ❖ 先決条件
    1. ファイルの復号関数見つかったが、ロジックが複雑なので直接メモリからダンプしたい
    2. アプリが起動直後に復号関数が一回だけ呼ばれる
    3. 復号関数はlibgame.soにあるとする
  - ❖ 課題
    - libgame.soにある該当復号関数にbreakpointをかけて、メモリを読みたい！

© DeNA Co., Ltd.

早期デバッガアタッチ

- 解答

1. 早期デバッガアタッチする
2. dlopenにbreakpointを貼る
3. breakpointがヒットしたらr0/x0レジスターからライブラリの名前を確認
4. libgame.soだったらdlopen関数のretまで実行する
5. libgame.soにある復号関数にbreakpointをかける

ステップ3は反復実行されるので、IDAを使うか、gdbの場合は便利スクリプトを用意した：

https://github.com/nevermoe/debugger_scripts の

stop_at_load {libname} を使えば楽

© DeNA Co., Ltd.

（守）チート対策ソリューション

- 診断で使う攻撃手法無論チータも利用している
    - ==>クライアントセキュリティソリューションを導入するのが一般的

© DeNA Co., Ltd.

（守）チート対策ソリューション

- 取り組み方から分類
  - ライブラリ式
    - ❖ コーディング時に呼ぶ
  - コンパイラ式
    - ❖ コンパイル時適用
  - Packer式
    - ❖ 最終バイナリに適用

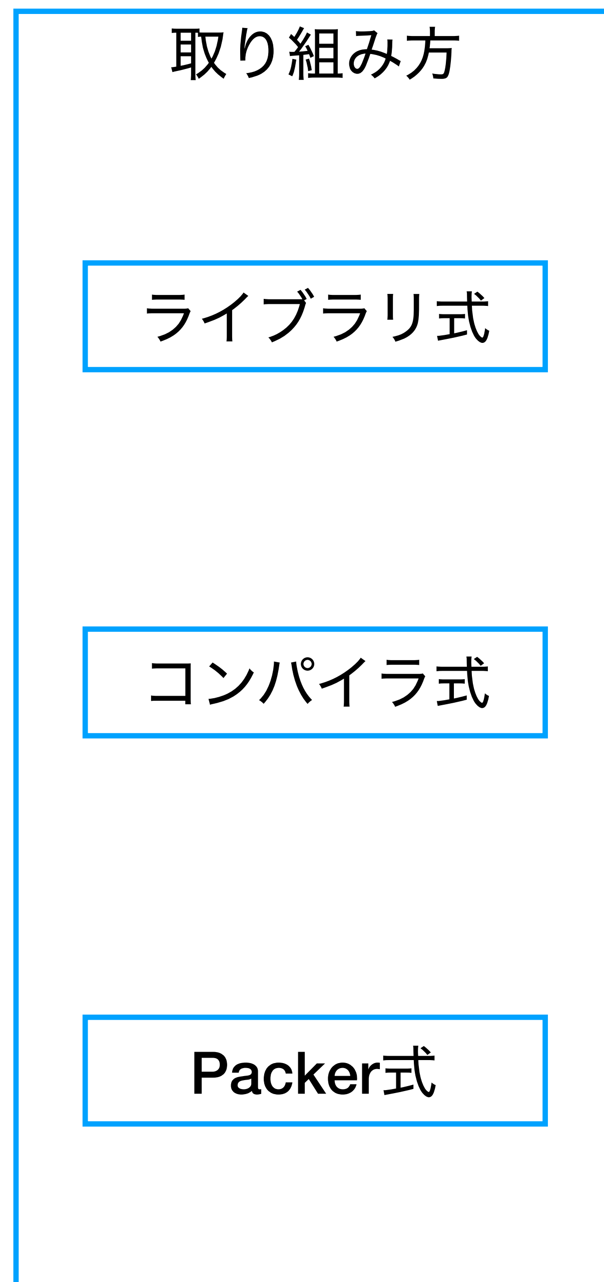
© DeNA Co., Ltd.

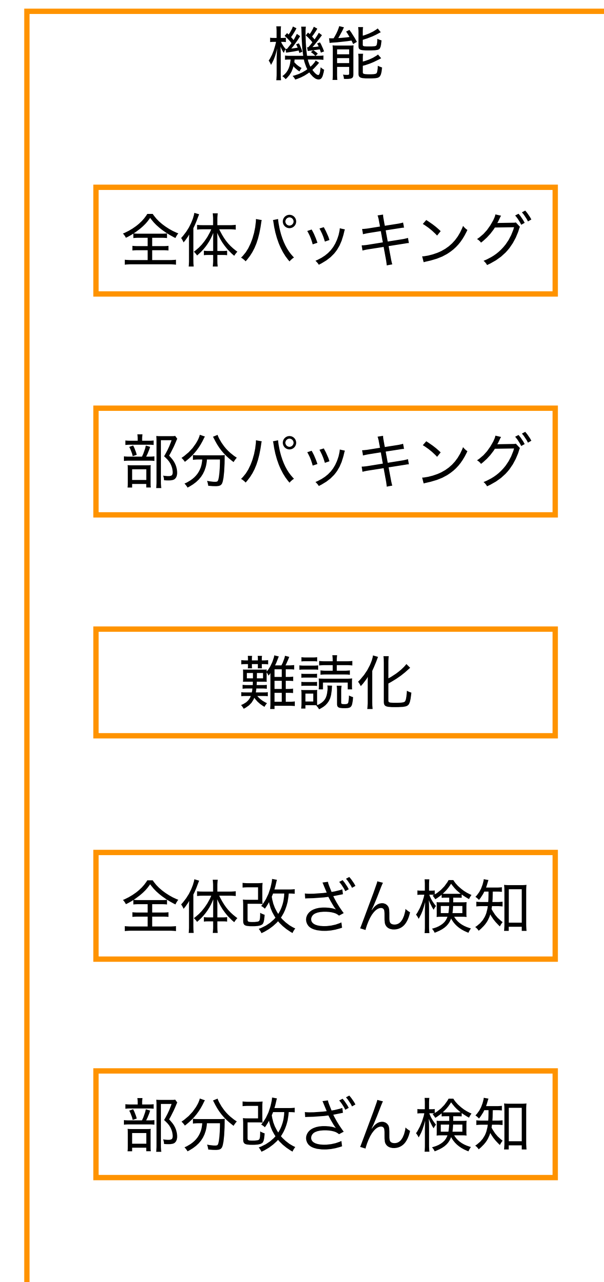
（守）チート対策ソリューション

- 機能から分類
  - パッキング
    - ❖ 実行バイナリを暗号化
    - ❖ 部分的なコードを暗号化
  - 難読化
    - ❖ 一般的な難読化：Control Flow Flattening、デッドコード挿入などなど
    - ❖ VM難読化: ネイティブコードを独自の命令セットを解釈できるインタプリタを介して実行する
  - 改ざん検知
    - ❖ コード領域全体のチェックサムを照合
    - ❖ 部分的なコードのチェックサムを照合
  - Anti-decompile
    - ❖ Lua/Python/C#などのインタプリタにおけるopcodeを置換

© DeNA Co., Ltd.

組み合わせ

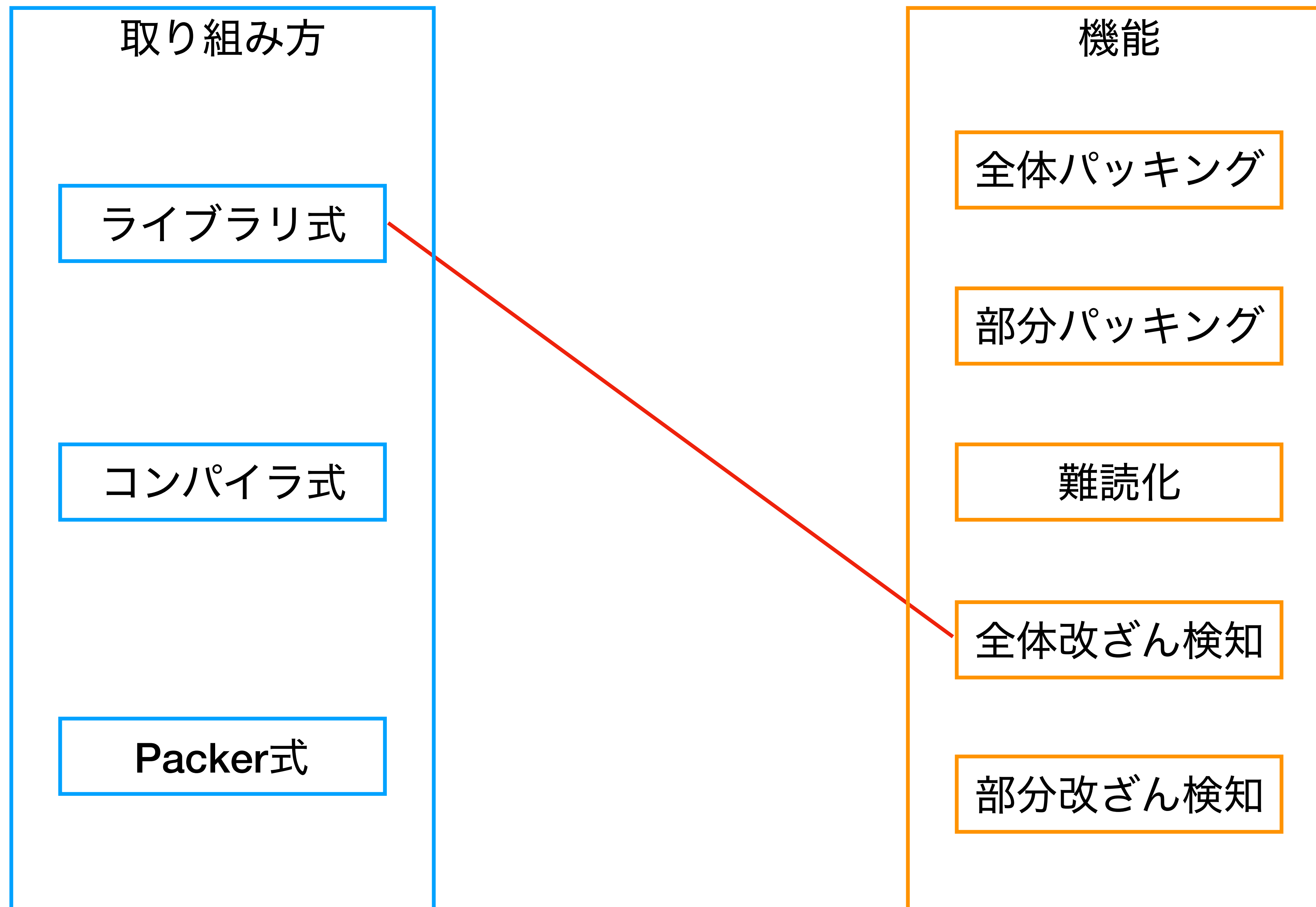




*ネイティブに着目するのでAnti-decompileを割愛します。

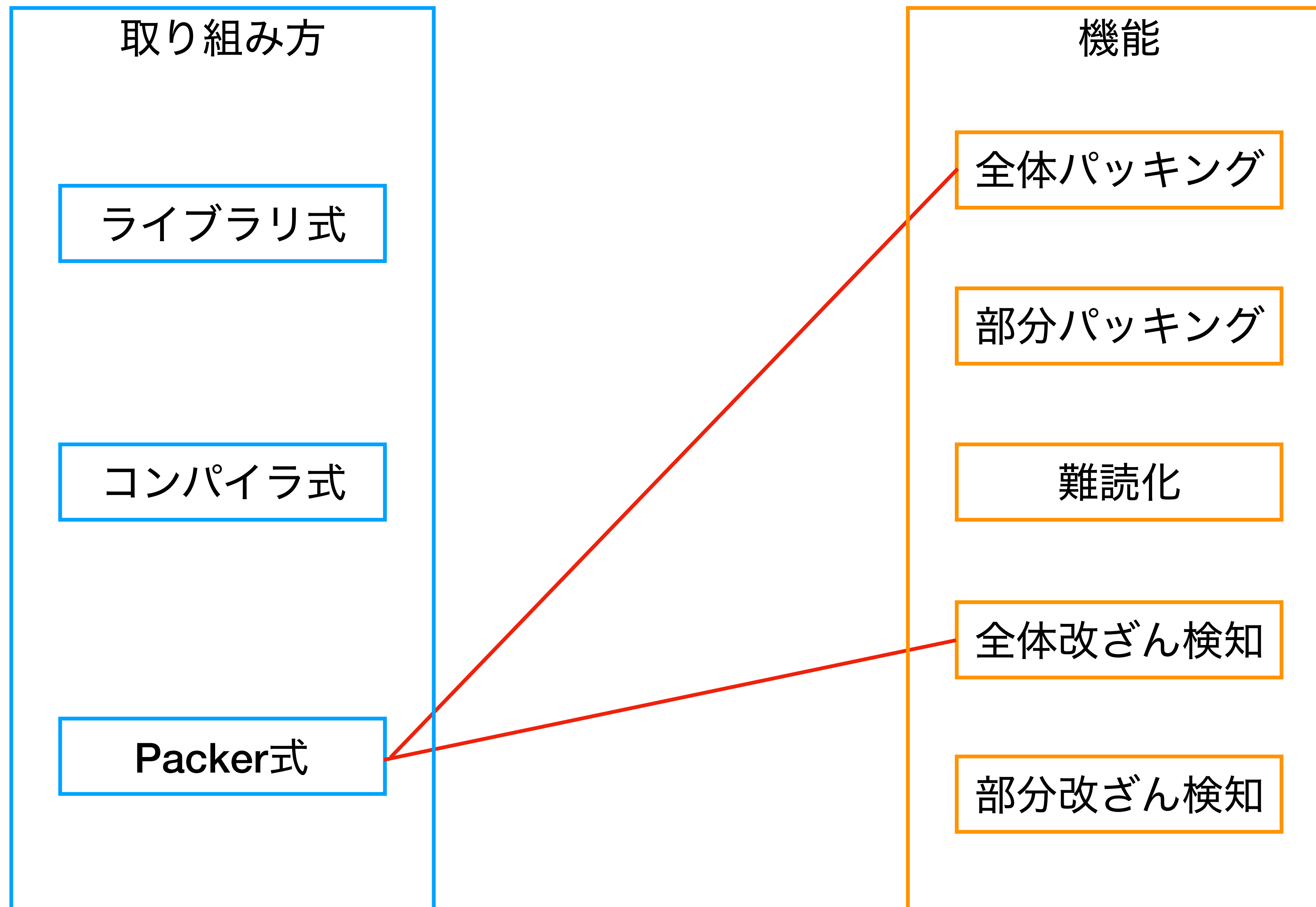
© DeNA Co., Ltd.

組み合わせ



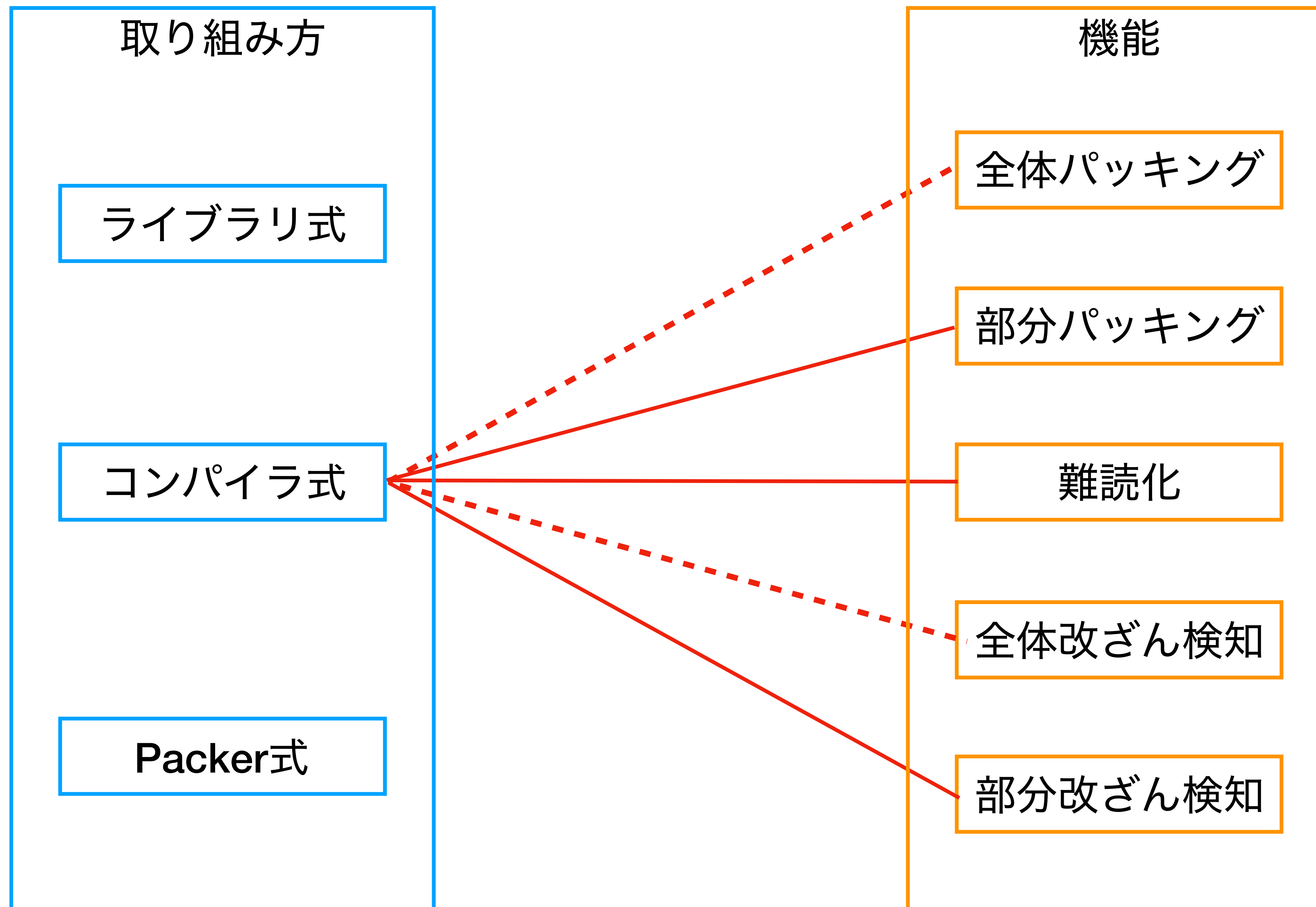
© DeNA Co., Ltd.

組み合わせ



© DeNA Co., Ltd.

組み合わせ



© DeNA Co., Ltd.

ソリューション考察

ざっくり：

ライブラリ式

低
高
中

コンパイラ式

高
中
低

Packer式

中
低
高

© DeNA Co., Ltd.

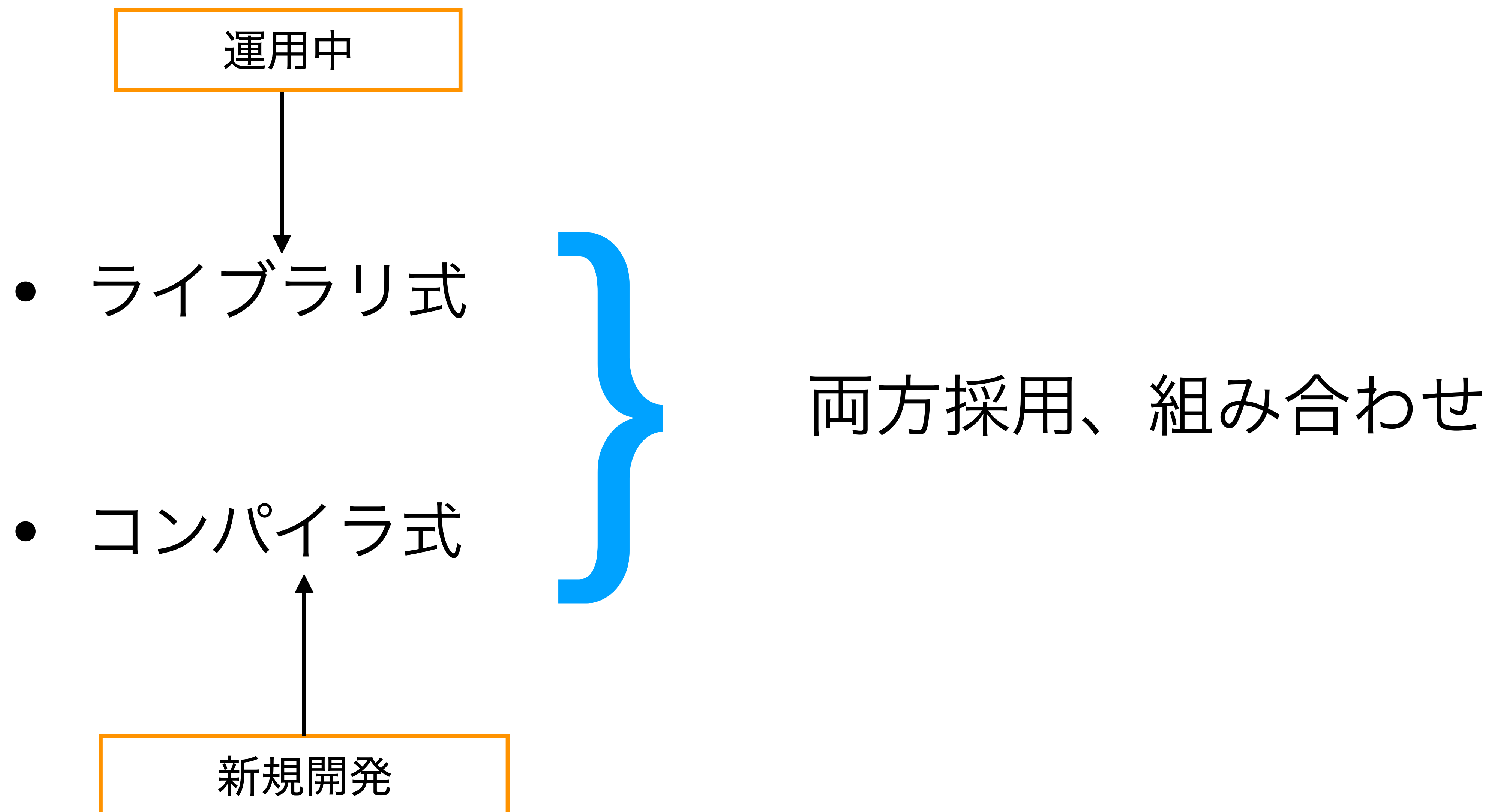
ソリューション考察

もうちょっと補足：

- ライブラリ式
  - ❖ 簡単に作れる
- コンパイラ式
  - ❖ パフォーマンスの微調整できる
  - ❖ iOSに対応できるメリットは大きい
- Packer式
  - ❖ iOSに適用できない

© DeNA Co., Ltd.

チート対策ソリューション



*参考：https://engineer.dena.jp/2018/12/anti-cheat-tool-using-llvm.html

© DeNA Co., Ltd.

コンパイラ式ソリューション

- 課題点
  1.既存の(CIを含む)ビルドプロセスに統合できるか
  2.ビルド速度にどれぐらい影響あるか
  3.開発者にどれぐらい負担かけるか

© DeNA Co., Ltd.

コンパイラ式ソリューション

- 課題点
  1. 既存の(CIを含む)ビルドプロセスに統合できるか
  2. ビルド速度にどれぐらい影響あるか
  3. 開発者にどれぐらい負担かけるか

- 解決案
  - ❖ カスタマイズClangを作る！
    1. Clangバイナリを置き換えるだけでビルドプロセスに影響無し
    2. 余計なコンパイルしないのでビルド速度への影響が小
    3. Makefileなどを書く必要がない、configファイルを書くだけ

© DeNA Co., Ltd.

Clang改造

- 現状
  - iOSに適用できるPoCが完成、絶賛開発中。
    - Control Flow Flattening ✓
    - 部分的改ざん検知 ✓
    - 部分的パッキング
    - 文字列暗号化
    - などなど

© DeNA Co., Ltd.

サマリー

- デバッギングできたらほぼなんでもできる
- 逆にコンパイラ改造できたら？なんでもできる

© DeNA Co., Ltd.

ありがとうございます！

© DeNA Co., Ltd.