移动应用加固技术

梆梆安全
CTO 陈彪

加固的原因

- Android Java编写，容易逆向
- Android应用允许自签名，同时应用市场混乱，导致大量应用被二次打包，植入广告、木马等
- Root后，利用调试、Hook等技术手段对应用进行动态攻击

加固功能

- 代码加壳
- 反调试
- 完整性检查

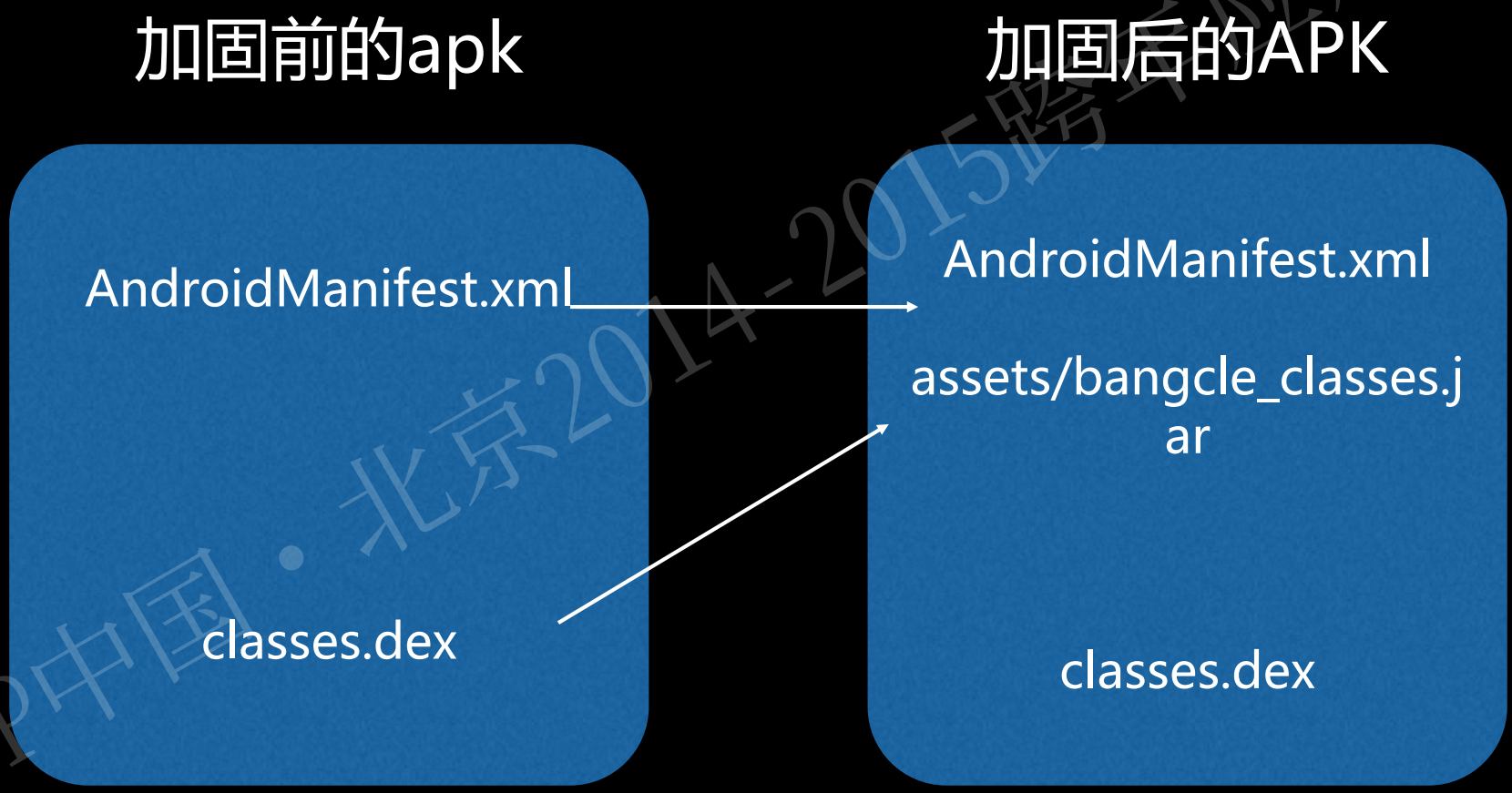
保护对象

- Java - Android Dex
- C/C++/ObjC - Android So、Apple mach-o
- 各种开发框架的文件
  - ✓ html、Js、Lua、C#等

Dex加固 - 第一代整体加密技术

- 基于Java本身提供的类加载技术
- classes.dex被完整加密，放到APK的资源中
- 运行时修改程序入口，将加密后的classes.dex在内存中解密，并让Dalvik虚拟机加载执行

加固前后的变化



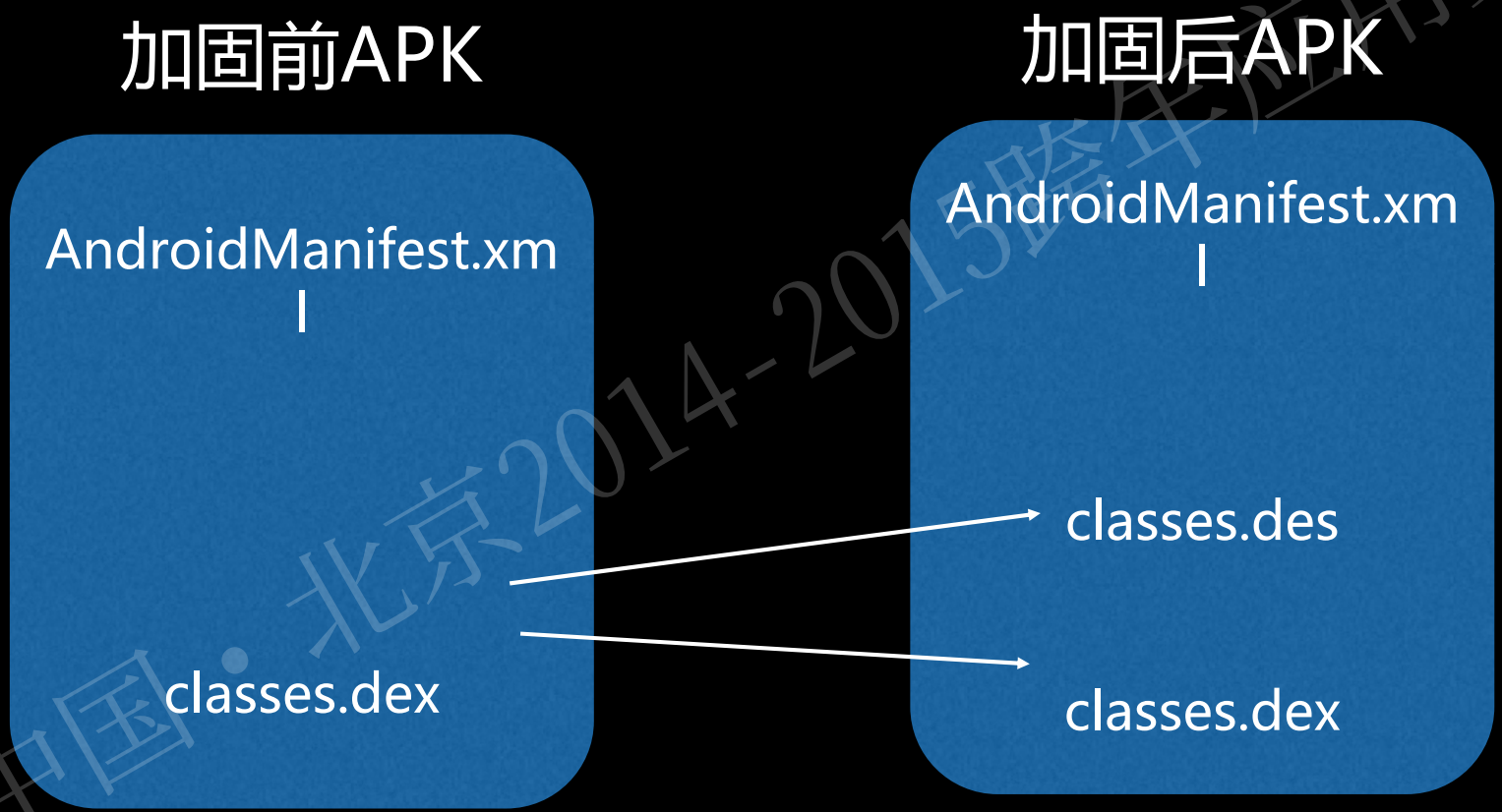
存在的问题

- 难以对抗动态分析
- 内存中存在连续完整的解密后的代码，可通过内存dump的方式得到解密代码
- 针对内存dump，各家加固厂商都会打一些patch:
  - ✓ Dex加载完毕后，抹掉或者混淆内存中dex的头部或者尾部信息
  - ✓ 检查ZJdroid是否存在，如存在，加固应用不运行
  - ✓ ...
- 这些patch都是治标不治本的措施，难以从本质上解决内存dump的问题
- 例如:通过修改Dalvik虚拟机，在载入dex时候进行dump，而不是在Dex已加载完成之后

Dex加固 - 第二代

- 基于方法替换方式
  - ✓ 原理：Java虚拟机在第一次执行某个类的某个方法前，才需要真正加载这个方法的代字节码
  - ✓ 将原APK中的所有方法的代码提取出来，单独加密
  - ✓ 当Dalvik要执行某个方法时，加固引擎才解密该方法，并将解密后的代码交给虚拟机执行引擎执行

加固前后的变化



加固前后的变化

加固前APK

SAEditText.class

```
private void hideSystemInput()
{
  setOnTouchListener(new View.OnTouchListener()
  {
    public boolean onTouch(View paramAnonymousView, MotionEvent paramAnonymousMotionEvent)
    {
      int i = SAEditText.this.getInputType();
      SAEditText.this.setInputType(0);
      SAEditText.this.onTouchEvent(paramAnonymousMotionEvent);
      SAEditText.this.setInputType(i);
      return true;
    }
  });
}

private void onCloseActivity()
{
  if (this.infoReceiver == null)
    return;
  this.saETContext.unregisterReceiver(this.infoReceiver);
  this.infoReceiver = null;
}

private void onSetContentWithMask()
{
  String str = "";
  for (int i = 0; ; i++)
  {
    if (i >= this.sakbd.getPlaitextLen())
    {
      setText(str);
      setSelection(length());
      return;
    }
    str = str + this.maskStr;
  }
```

加固后APK

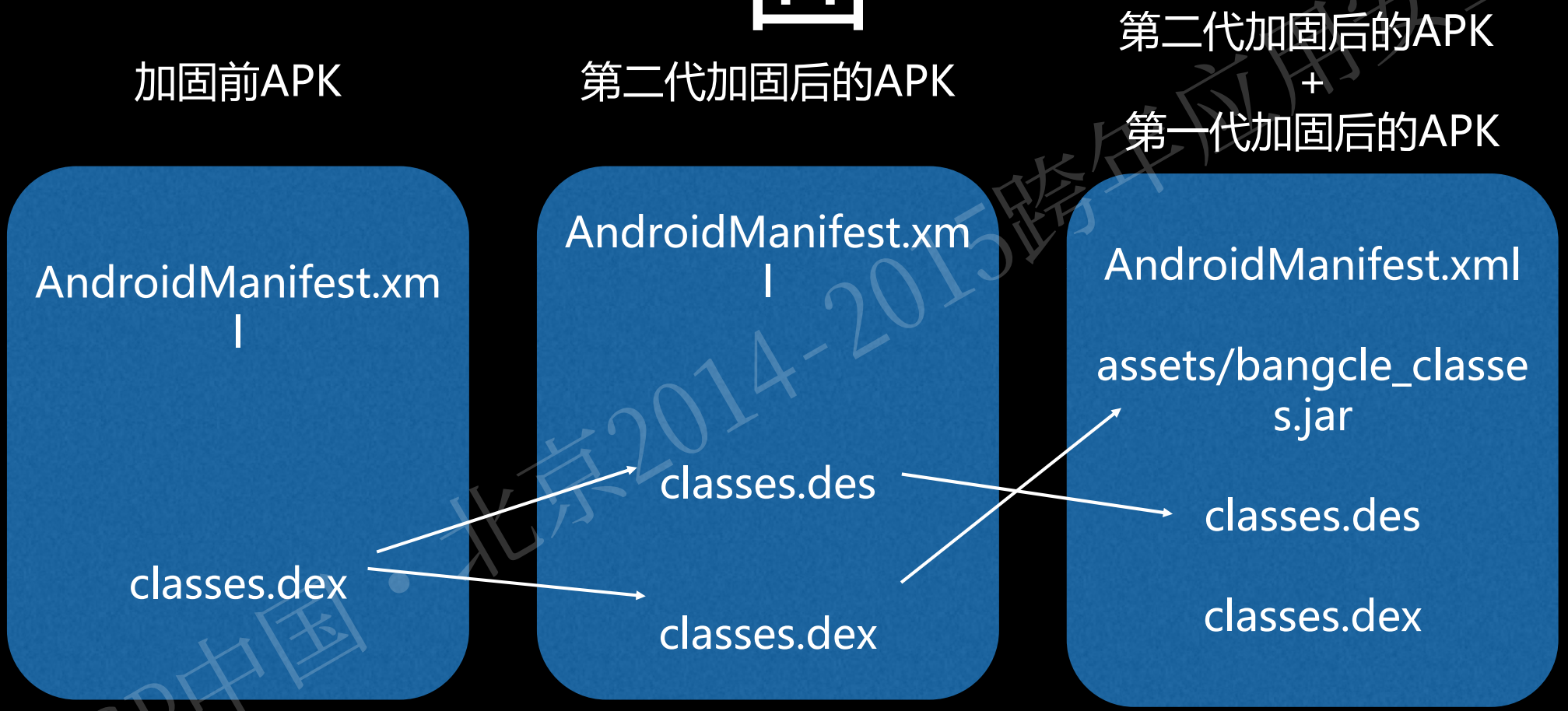
SAEditText.class

```
private void hideSystemInput()
{
}

private void onCloseActivity()
{
}

private void onSetContentWithMask()
{
}

private void onStartActivity()
{
}

private void openSystemSoftKeyboard()
{
}

public void clear()
{
}

public void closeSAKbd()
{
}

public char getComplexDegree()
{
  return '\000';
}

public byte[] getEncryptedPinCode()
{
  return null;
}
```

方法替换的特点

- 特点
  - ✓ 加密粒度从Dex文件变为方法级别
  - ✓ 按需解密
  - ✓ 解密后代码在内存中不连续
- 优点
  - ✓ 对抗动态分析，内存dump成本高

第一代+第二代混合加固



C/C++/ObjC的保护

- 编译成汇编
  - ✓ Android NDK -> ELF（ARM/X86)
  - ✓ Xcode -> mach-o (ARM)
- 保护方式
  - ✓ binary-level
  - ✓ source-level

So保护-binary level

- 类似与PC上的加壳软件
  - ✓ 代码加密或者压缩
  - ✓ 畸形ELF
  - ✓ 自定义加载器(linker)
  - ✓ ....
  - ✓ 虚拟机保护（类PC上的VMProtect）
    - 移动平台是否可行？

So保护 - source level

- 基于源代码级别的混淆
- 核心技术 - 控制流平坦化(control flow flatten)

控制流平坦化

```
i = 1;
s = 0;


while (i <= 100) {


  s += i;
  i++;


}
```

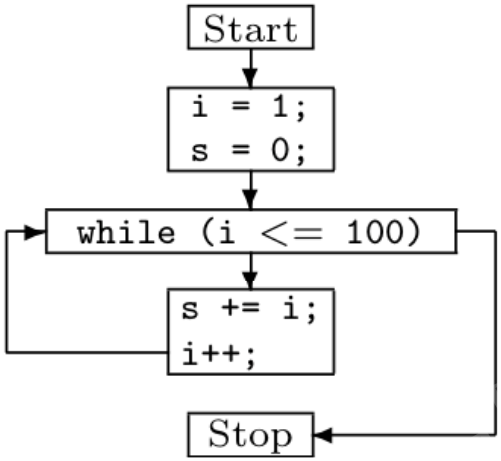
(a)

```
int swVar = 1;
while (swVar != 0) {
  switch (swVar) {
    case 1: {
      i = 1;
      s = 0;
      swVar = 2;
      break;
    }
    case 2: {
      if (i <= 100)
        swVar = 3;
      else
        swVar = 0;
      break;
    }
    case 3: {
      s += i;
      i++;
      swVar = 2;
      break;
    }
  }
}
```

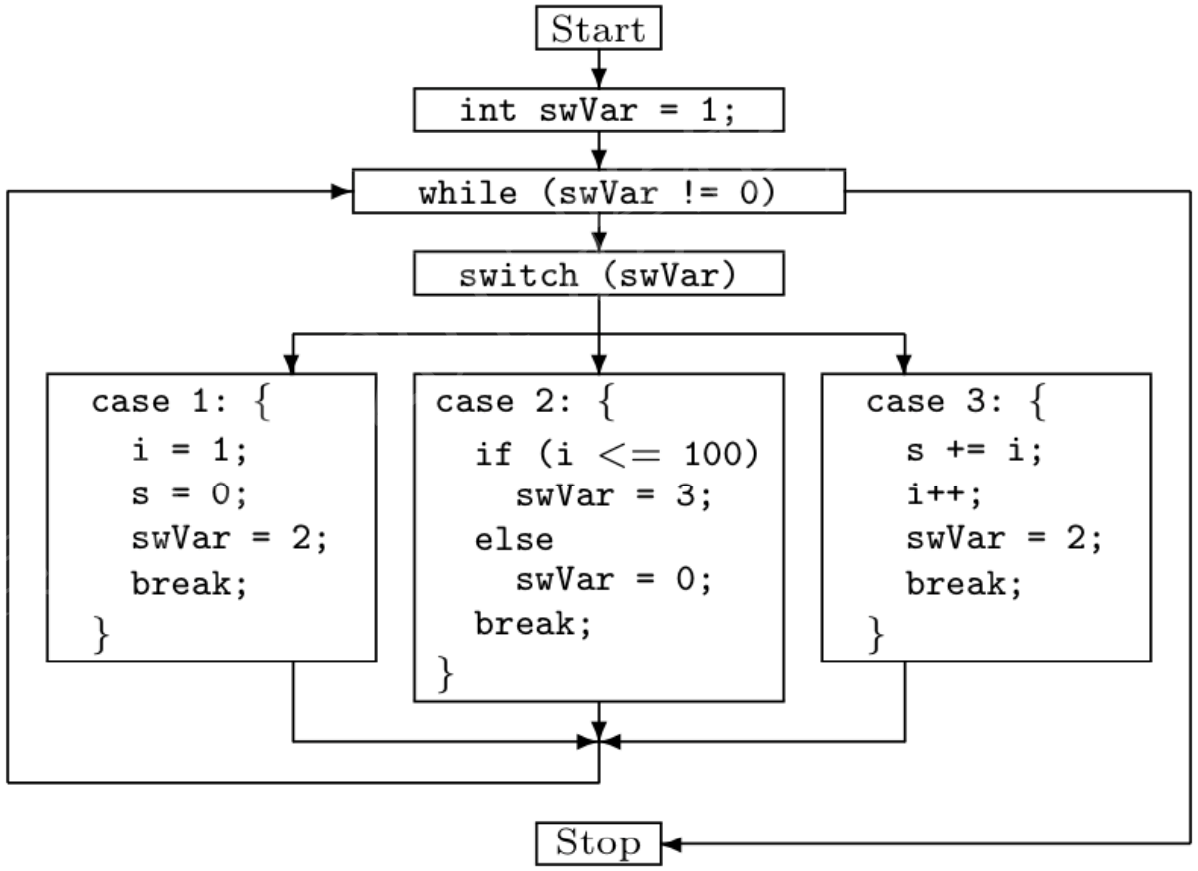
(b)

T László and Á Kiss, Obfuscating C++ programs via control flow flattening, Annales Univ. Sci. Budapest., Sect. Comp. 30 (2009)

控制流平坦化



(c)



(d)

T László and Á Kiss, Obfuscating C++ programs via control flow flattening, Annales Univ. Sci. Budapest., Sect. Comp. 30 (2009)

控制流平坦化的实现

- 基于LLVM或者Clang
- 开源项目
  - ios平台
    - https://github.com/obfuscator-llvm/obfuscator
  - Android平台
    - https://github.com/Fuzion24/AndroidObfuscation-NDK

梆梆C/C++/ObjC保护方案

- Android平台 - 源码级别的混淆 + 二进制加壳
- IOS平台 - 源码级别的混淆
- 源码级别
  - ✓ 控制流混淆
  - ✓ 各种花指令等
  - ✓ 代码/数据完整性校验
  - ✓ 代码多样性
  - ✓ 反调试、越狱检查等

梆梆C/C++/ObjC保护方案

- DEMO

开发框架的保护

- App开发者使用各类跨平台开发框架进行开发，通常都放在资源中
- 采用编程框架开发的程序进行加密保护
  - ✓ DLL（C# Unity3D）
  - ✓ Lua
  - ✓ HTML/Javascript(IBM worklight、PhoneGap等)
  - ✓ Flash
  - ✓ …
- 与Dex、so保护绑定在一起，形成广度的防御

总结

- 应用加固技术主要是针对代码保护的，是应用安全的基础
  - ✓ 如果没有安全加固的保护，上层的各种安全措施，如软键盘、加密密钥等，都可能被轻易拿到
- 但安全加固不是万能药，开发人员还是需要提高安全开发意识，遵循安全规范进行客户端的开发
  - ✓ 如各种程序漏洞、组件漏洞、业务逻辑漏洞等，加固是无能为力的