申请上海交通大学硕士学位论文

# Android 设备已公开漏洞的自适应修补方法研究


**论文作者** 张雪雯

**学　　号** 115033910032

**导　　师** 谷大武　教授

**专　　业** 计算机科学与技术



**答辩日期** 2018 年 1 月 9 日

Submitted in total fulfillment of the requirements for the degree of Master
in Computer Science

# Adaptive Patching for Public Vulnerabilities in Obsolete Android Devices


Zhang Xuewen

Advisor

Prof. Gu Dawu

Department of Computer Science and Engineering, School of Electronic Information and Electrical Engineering

Shanghai Jiao Tong University

Shanghai, P.R.China


Jan. 9th, 2018

# 上海交通大学
# 学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：_____

日　期：_____年_____月_____日

# 上海交通大学
# 学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

**保　密** □，在 _____ 年解密后适用本授权书。

**不保密** □。

(请在以上方框内打 ✓)

学位论文作者签名：_____　　指导教师签名：_____

日　期：_____年 ____月 ____日　　日　期：_____年 ____月 ____日

# Android 设备已公开漏洞的自适应修补方法研究

## 摘 要

随着移动生态高速发展，越来越多的人开始关注 Android 系统安全与漏洞研究。由于 Android 系统生态多样化，当一个安全更新发布之后，通常情况下只有少量的 Android 设备（例如 Google 的 Nexus 设备）会收到安全补丁的推送，而其他厂商往往需要花费大量的时间修改代码、进行测试，以兼容旗下定制的各种型号的移动设备。一些厂商甚至会忽略某些安全更新或者只对自己最新的设备进行更新，导致大量的 Android 设备面临严重的安全风险。因此，为了实现对不同 Android 设备的漏洞修补，需要一个第三方修补策略，将官方漏洞的修补源代码移植适配到各种设备上。

本文设计并实现了 Embroidery 系统，一个 Android 漏洞自适应修补系统。该系统基于二进制重写技术，可以在没有源码的情况下为不同 Android 设备上的框架层和内核层漏洞提供通用的安全更新。针对 Android 生态多样化导致的 Android 二进制代码差异性问题，Embroidery 提出了一种启发式的代码特征匹配策略，用于帮助匹配定位漏洞函数、插入代码的位置以及获取相关的修补信息 (修补相关的寄存器、内存地址等)。在修补过程中，Embroidery 针对框架层漏洞修补采用了静态二进制重写替换策略；针对内核层漏洞修补，Embroidery 能够处理内核代码不可写，可执行内核内存分配困难等因素，实施稳定的内核内存动态重写修补。

为了评估 Embroidery 系统的有效性，我们选取了不同厂商的 Android 设备 (系统版本 4.2-5.1) 作为实验对象，测试了所提出的自适应修补方法。具体地，我们通过 Android 系统碎片化实验，表明了由于 Android 系统生态多样化导致不同设备上二进制代码差异性极大以及不同内核防护措施的使用造成内核动态内存重写困难；我们通过漏洞修补实验，证明了 Embroidery 系统采取的启发式特征匹配策略可以适应二进制代码差异性，同时，针对内核层漏洞修补所采用的内核页表权限重置以及内存动态重写策略可以解决内核防护机制带来的修补困难问题。我们选取了两组广泛存在于多版本 Android 系统的公开漏洞进行了实验。所有被测试的漏洞都在实验中被修补系统精确检测定位，并最终被成功修补。修补后的设备可以抵御已知的漏洞攻击，同时正常功能不受影响。实验结果表明，Embroidery 系统能为多种 Android 设备的框架层和内核层漏洞提供自适应的安全更新维护。

**关键词：** 二进制漏洞　自适应漏洞修补　Android 安全

# Adaptive Patching for Public Vulnerabilities in Obsolete Android Devices

## ABSTRACT

With the rapid development of mobile ecology, more and more people are beginning to pay attention to Android system security research. In fact, due to the ecological diversity of the Android system, only few Android devices (official Nexus devices of Google) can receive the OTA updates related to latest Android's monthly security updates. For most manufacturers, it may take some time to make the patching code available thus the security update often tends to lag behind for a considerable long period. Even in some cases, some vendors ignore certain updates or focus more on promoting new devices, remaining millions of vulnerable obsolete Android devices in use. Therefore, a third-party vulnerability patching strategy is needed to port the official patch source code to a variety of obsolete devices.

To implement this, we propose EMBROIDERY, an adaptive binary rewriting based vulnerability patching system for obsolete Android devices without requiring the manufacturer's source code. To address the issue of Android fragmentation, EMBROIDERY applies heuristic matching strategies to help locate not only the appropriate places in binaries for patching code insertion but also the related patching information needed for patching process. During patching process, EMBROIDERY applies static binary rewriting strategies for framework vulnerabilities and fulfills a complex dynamic memory rewriting with kernel page table modification to implement kernel vulnerabilities patching.

Finally, we conduct experiments on devices from different manufacturers with installed OS ranging from Android 4.2 to 5.1. We test the system using two sets of vulnerabilities that affect most obsolete Android devices. The fragmentation experiment shows the great diversity of binary code on different devices due to the Android fragmentation. The patching evaluation demonstrates the heuristic matching strategies adopted by system can adapt to binary differences and page table modification strategies for kernel are effective. All of the vulnerabilities were located and the patched devices worked as usual after patching. As a result, the patching system is able to provide adaptive security updates for both framework and kernel vulnerabilities on

different obsolete devices.

**KEY WORDS:** Binary vulnerabilities, Adaptive patching, Android Security

# 目 录

# 插图索引

# 表格索引

# 主要符号对照表

| | |
|---|---|
| CVE | Common Vulnerabilities & Exposures |
| OEM | Original Equipment Manufacturer |
| MDM | Mobile Device Management |
| BYOD | Bring Your Own Device |
| SELinux | Security-Enhanced Linux |
| PoC | Proof of Concepts |
| LKM | Loadable Kernel Module |
| UAF | Use-After-Free |
| GLIBC | GNU C Library |
| NX | No-eXecute |
| PXN | Privilege Execute-Never |
| ASLR | Address Space Layout Randomization |
| KASLR | Kernel Address Space Layout Randomization |
| RKP | Real-Time Kernel Protection |
| ELF | Executable and Linkable Format |
| MPEG | Moving Picture Experts Group |
| PREC | Practical Root Exploit Containment |
| GCC | GNU Compiler Collection |
| TTBR | Translation Table Base Register |
| ret2dir | return-to-direct-mapped memory |
| OTA | Over The Air |

# 第一章 绪论

## 1.1 研究背景

移动终端具有尺寸小、移动性强的特点，其具备存储和处理数据、访问网络等能力，和用户身份紧密绑定，使其成为当下最主要的安全攻击目标。在民生和社会经济方面，每年由于移动智能设备安全问题引发的信息泄漏、系统破坏、诱骗欺诈造成的经济损失高达百亿。在国家和社会管理层面，移动智能作为国家基础设施中的重要一环，其安全面临更大的安全威胁：部分省市政府人员办公手机已经开始替换为专用手机；工控系统中也大量引入移动设备如平板电脑便于现场操作；作战系统和外勤工作也将大量依赖于移动智能设备；公共安全方面也引入了移动警务系统推广使用。这些关键职能部门的信息安全紧系国家利益，需要强有力的保障。

对于移动网络和终端的安全保护，最重要的防护措施之一是对其漏洞的及时修补。对于目前移动终端的主流操作系统——Android 操作系统，其漏洞的修补工作涉及不同层次，其中最重要的是框架层和内核层漏洞的修补。作为连接上层应用程序和底层系统资源的纽带，框架层的安全具有不容忽视的重要性，框架层出现的安全漏洞往往能够同时影响底层系统安全性和上层应用程序的安全，例如 Android 平台早期出现的组件暴露、ASLR 随机性缺失[aslr]、ADB 接口越权以及著名的 Stagefright[stagefrightdetail, stagefrightwiki] 等漏洞。该层次的漏洞多由框架层不同组件安全实现不一致、权限检查不严格、代码逻辑错误等原因引起。另一方面，Android 系统基于 Linux 开源内核构建自身的底层系统内核，Linux 开源内核代码中所包含的漏洞同样也会被 Android 所继承[brady2008anatomy]。内核漏洞所属的代码运行于内核态下，对于这些漏洞的攻击一般能够直接接触到系统的最底层，会对系统整体安全性造成极大的破坏。

针对 Android 设备上发现的大多数公开漏洞，其修补工作通常依赖于在源代码上进行修改并重新编译。然而现实中由于 Android 系统生态复杂化，通常情况下只有少量的 Android 设备（例如 Google 的 Nexus 设备）会收到安全补丁的推送，其他厂商往往需要大量的时间修改代码、进行测试，以兼容自己定制的移动设备。甚至于在某些情况下，一些厂商甚至会忽略某些更新或者只对自己最新的设备进行更新。例如，Motorola[motorupdate] 因为兼容更新的时耗和复杂性，宣布取消了安全月更新。因此大量设备在厂商没有及时提供安全更新的情况下面临严重的安全危机。另一方面，对于 Android 系统进行安全更新最大的障碍之一来自于 Android 系统碎片化。Android 系统的碎片化是指世界上所有 Android 用户实际使用的 Android 版本、配置、驱动等等有着非常显著的差异。Android

系统碎片化已经成为了业界共识，由于系统的开源性，用户、开发者、OEM 厂商、运营商都可以按照自己的想法进行改造，这种 “碎片化” 的程度异常严重。而且大多数厂商将自己设备相关的代码闭源，各厂商基于源码进行了大量修改或删除相关函数来生成自己某一固定的版本。此外有些厂商使用不同的编译器或编译选项对二进制代码进行优化，在这种情况下，二进制代码多样化，差异尤其明显。此外内核层漏洞修补涉及到内核二进制重写等工作，而不同厂商针对内核采用了不同的防护手段 (内核代码只读，内核模块校验[you2011android]，内核模块白名单，PXN[pxn]，NX[nx]，Samsung RKP[rkp] 等)，这些内核防护措施对重写工作造成了一定的阻碍。因此在这种情况下，安全研究人员无法在没有厂商支持的情况下提供一个通用的第三方安全更新。

## 1.2 研究内容与成果

根据 1.1节所述，如何为不能及时获得安全更新的各种 Android 设备提供一个通用的修补方案成为目前 Android 漏洞修补的一个重要研究方向。本文针对 Android 设备的安全更新问题进行了研究，具体贡献如下:

1. 设计并实现了一种在没有厂商源码的情况下基于二进制重写的修补漏洞的系统。该系统基于 CVE[1]进行漏洞修补。系统根据官方的 CVE 修补源代码，将其移植适配到不同厂商的不同设备上。该系统针对 Android 框架层和内核层的漏洞的修补，提出了不同的修补方针。对于框架层漏洞，系统直接对二进制文件进行修补操作，在其中添加新的代码段，插入修补代码，然后替换原来有漏洞的文件；对于内核层漏洞，系统针对不同的防护措施，能够处理内核代码不可写，可执行内核内存分配困难等因素，实施稳定的内核内存动态重写修补。
2. 针对 Android 二进制代码差异性问题，本文提出了一种启发式的代码特征匹配策略。该策略采用代码相似性特征来帮助匹配定位漏洞函数、插入代码的位置以及获取相关的修补信息（修补相关的寄存器、内存地址等）。
3. 本文对开发的原型修补系统 Embroidery 进行了测试分析，论证了修补方法的可行性。同时进行了实验设备的差异性验证，论证了系统可以适应不同厂商不同 Android 版本的设备差异，提供通用的安全更新。实验表明，经过系统修补后的移动设备可以成功抵御已知的漏洞，并且功能无影响。

[1]Common Vulnerabilities and Exposures (CVE)，是广泛认同的信息安全漏洞或者已经暴露出来的弱点的公共命名规范。如 CVE-2015-3636 代表 2015 年被发现并公开定义的第 3636 号漏洞。对于部分 CVE，研究人员会公开针对性的攻击代码，例如针对 CVE-2015-3636，安全分析人员公布了著名的 Pingpong Root Exploit。

## 1.3 论文结构

本文的其他章节如下，第二章总结了目前 Android 设备漏洞修补研究的难点以及国内外的一些研究现状。第三章设计了一种在没有源码的情况下基于二进制重写的修补漏洞的系统，并给出了原型系统的实现细节。第四章针对本文设计实现的原型系统进行测试和评估，并给出实验结果以及样例分析。第五章总结全文并展望了未来的研究。

# 第二章 研究现状

本章将介绍目前 Android 移动设备漏洞修补研究面临的难点以及框架层、内核层漏洞修补研究领域相关的研究现状。

## 2.1 研究难点

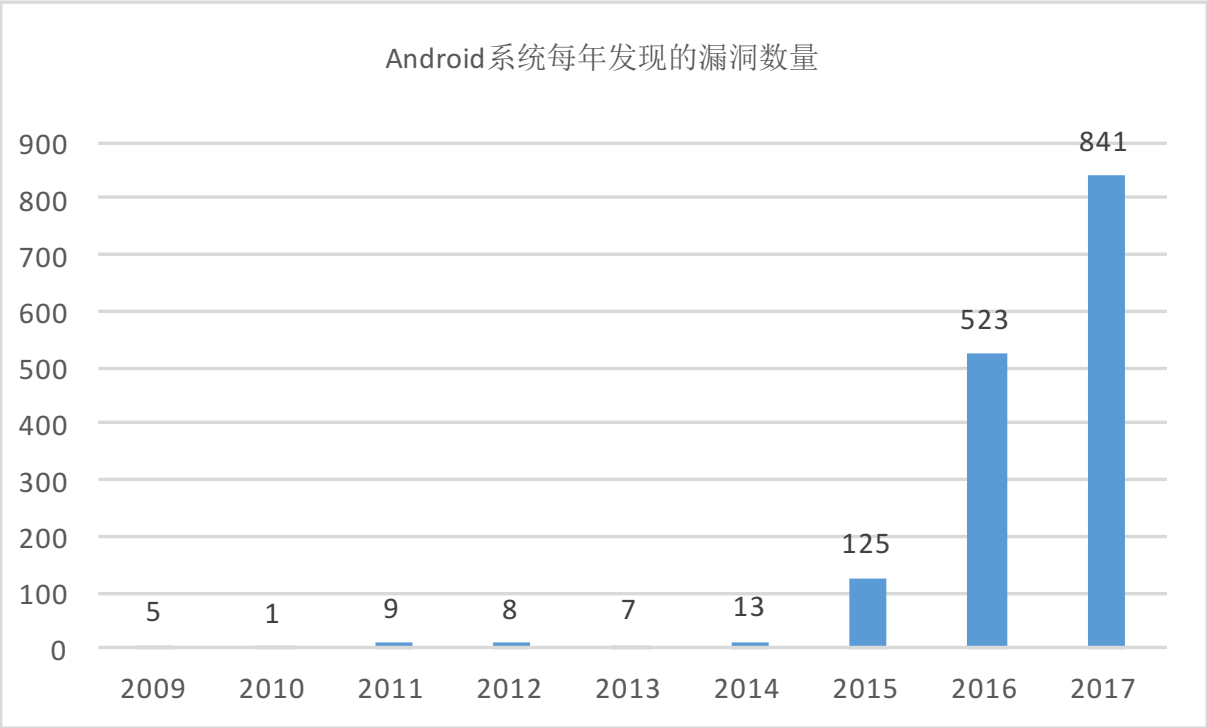


图 2–1 Android 系统每年发现的漏洞数量

Fig 2–1 Vulnerability trends over time in Android

随着移动生态的发展，越来越多的人开始关注 Android 系统安全与漏洞研究。根据 CVE 数据库[cve] 的信息（如图 2–1所示），每年发现的 Android 系统漏洞数量呈显著增长趋势，2017 年发现的漏洞数量甚至超过往年总和。如此众多的 Android 系统漏洞将增大各 Android 厂商安全更新的压力，导致大量设备得不到及时安全更新，面临严重的安全威胁。众多的 Android 漏洞涉及到了不同层次，其中我们重点关注框架层和内核层漏洞的修补。框架层的漏洞多由框架层不同组件安全实现不一致、权限检查不严格、代码

逻辑错误等原因引起。另一方面，Android 系统基于 Linux 开源内核构建自身的底层系统内核，Linux 开源内核代码中所包含的漏洞同样也会被 Android 所继承。内核漏洞所属的代码运行于内核态下，对于这些漏洞的攻击一般能够直接接触到系统的最底层，会对系统整体安全性造成极大的破坏。因此，对框架层和内核层的漏洞的及时修补至关重要。

表 2–1 运行指定 Android 平台版本的设备的相对数量

Table 2–1 The relative number of devices running a given version of the Android platform

| 版本 | 版本名 | API | 比例 |
|---|---|---|---|
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 0.4% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 0.5% |
| 4.1.x | Jelly Bean | 16 | 2.0% |
| 4.2.x | Jelly Bean | 17 | 3.0% |
| 4.3 | Jelly Bean | 18 | 0.9% |
| 4.4 | KitKat | 19 | 13.4% |
| 5.0 | Lollipop | 21 | 6.1% |
| 5.1 | Lollipop | 22 | 20.2% |
| 6.0 | Marshmallow | 23 | 29.7% |
| 7.0 | Nougat | 24 | 19.3% |
| 7.1 | Nougat | 25 | 4.0% |
| 8.0 | Oreo | 26 | 0.5% |

当一个影响几乎所有 Android 版本、危害性大的安全漏洞被公布的时候，一些有经验的攻击者可以根据漏洞细节很快写出利用代码对移动设备进行攻击。一次成功的攻击可能会导致恶意程序注入或者重要信息泄露（支付账号、密码等），严重的情况下，攻击者甚至可以远程控制移动设备。因此，及时地修补漏洞至关重要。Google 的 Android 安全公告**bulletin** 会每月发布重要的 Android 漏洞相关的 CVE 信息。CVE 信息公布后，更详细的漏洞细节或者攻击代码可能之后会被其他研究人员披露。现实中，当这样一个 CVE 被披露时，Google 只为自己官方的 Nexus 设备推送最新的安全更新（Google Nexus 设备由 Google 官方和与其合作的 OEM 制造厂商设计、开发、提供安全支持）。其他的厂商往往需要大量的时间来修改代码、测试，使其兼容旗下不同的移动设备。一些厂商甚至会忽略某些安全更新或者只对自己最新的设备进行更新。大部分情况下，只有这些厂商最新推出的 Android 设备才能及时获得安全更新。旧版本系统设备（例如，Android 系统 4.4）即使受到漏洞的危害，也不一定能接收或及时接收到相应的安全更新。这些

情况导致旧版本系统的 Android 设备得不到最新的安全支持，面临严重的安全威胁。根据 Google 官方发布的数据[google]（见表2–1），截止 2017 年 12 月，仍有 46.5% 的设备使用低于版本 6.0 的 Android 系统，甚至有超过 20% 的设备使用 Google 早已官方宣布永久停止更新的 4.x 和 2.x 的系统。因此，为了实现对不同 Android 设备的漏洞修补，需要一个第三方修补策略，将官方漏洞的修补源代码移植适配到各种设备上。

Android 移动设备之所以如此应用广泛的一个重要原因是由于 Android 系统本身的开源性。这使得 Android 系统成为众多厂商生产移动设备的第一选择。除了 Google 官方，众多其他的厂商基于原有 Android 系统进行多样化的定制并且采用不同的硬件规格。可以看到，现在 Android 阵营的主要领域厂商，包括三星、索尼、华为、小米、魅族等都没有采用 Android 原生的操作系统。这些原因导致了 Android 系统生态的多样性与复杂性，为 Android 移动设备漏洞修补带来诸多限制。传统 Android 移动设备漏洞修补存在以下一些难点：

```
ADD     R7, SP, #0x1C8+var_188
LDR     R0, [R7]
LDR     R3, [R4]
LDR.W   R12, =(_ZSt7nothrow_ptr - 0x110418)
ADD     R0, R3
LDR.W   R1, [R11, R12]  ; _ZSt7nothrow_ptr ...
BLX     _ZnajRKSt9nothrow_t
        ; operator new[](uint,std::nothrow_t const&)
```

```
LDR     R0, [SP, #0x1A8+var_168]
LDR     R3, [R7]
ADDS    R0, R0, R3 [R9,R1]
BLX     _Znaj ; operator new[](uint)
```

图 2–2 CVE-2015-3864 中二进制代码差异性

Fig 2–2 The binary fragmentation of CVE-2015-3864

### 2.1.1 Android 二进制代码多样性

由于 Android 系统的开源性，用户、开发者、OEM 厂商、运营商都可以按照自己的想法对 Android 系统进行改造，导致 Android 系统 “碎片化” 的程度异常严重。各厂商基于 Android 原生系统进行了大量修改或删除相关功能函数来生成自己固定版本的 Android 系统。而且大多数厂商会将自己设备相关的代码闭源，所以无法通过修改源码、重新编译来修补漏洞。因此基于二进制代码的漏洞修补研究成为为不同厂商的 Android 设备提供通用的安全更新的唯一途径。此外有些厂商使用不同的编译器或编译选项对二进制代码进行优化，在这种情况下，Android 系统的二进制代码多样化，差异尤其明显。

以 CVE-2015-3864 为例，图 2–2列出了在两种不同版本的 Android 系统上的有漏洞的函数呈现出的汇编代码。可以看到，不同的汇编代码采用了不同的寄存器分配以及不同的加载参数的方式，甚至不同系统采用了不同的函数。在上面的汇编代码采用了 `std::nothrow operator new`（在内存不足时，`new (std::nothrow)` 并不抛出异常，而是将指针置 NULL），下面的汇编代码采用了 `operator new` 函数。同一种简单的操作，上面的汇编代码比下面多了三条指令。可以想到如果是更加复杂的操作，汇编代码的差异性也将越大。而漏洞修补需要适应这些二进制代码的差异性并准确定位到相应漏洞的位置。

代码 2.1 CVE-2015-3864 修补源代码

```
status_t MPEG4Extractor::parseChunk
(off64_t *offset, int depth) {
...
      size = 0;
    }
-   if (SIZE_MAX - chunk_size <= size) {
+   if ((chunk_size > SIZE_MAX) ||
+       (SIZE_MAX - chunk_size <= size)) {
      return ERROR_MALFORMED;
    }
    uint8_t *buffer = new uint8_t[size + chunk_size];
...
```

代码 2.1列出了 CVE-2015-3864 的修补代码。针对这个漏洞，修补代码需要在内存分配前添加对参数的校验。然而由于在二进制层面上汇编实现呈现多样化，因此修补过程中必须考虑到这些汇编层面上的差异性（例如，因为编译环境差异导致的不同寄存器的分配、指令乱序等）。同时还需要通过对二进制代码进行分析来获取相关的修补信息（如修补相关的寄存器、内存地址等）以便生成修补代码。以 CVE-2015-3864 为例，代码 2.1的修补方案还需要通过二进制代码分析找到存储 `size` 和 `chunk_size` 的相关寄

存器、校验失败时返回代码的地址（填入 ERROR_MALFORMED 返回值、栈调整、参数恢复等一系列操作的指令的开头地址)。

因此，在修补过程中，修补系统需要能够适应二进制代码差异性，进行准确的漏洞定位和和获取相关修补信息。以下是漏洞修补过程中需要克服的一些关于二进制代码差异性的问题:

**汇编差异性** 因为编译环境的差异，最终会生成不同形式的汇编代码（不同寄存器的分配、指令乱序等)。这些不同形式的汇编代码可能导致指令数量、类型的变化，阻碍二进制代码分析。

**Android 系统版本** 一些漏洞在某些特定版本的 Android 系统上不存在（厂商修改或替代或删除了该有漏洞的函数)。在这种情况下，必须先判断漏洞是否存在，依靠代码相似性策略找到有漏洞的函数的位置。

**结构体定义** 各厂商可基于源码进行了大量修改或删除相关函数来生成自己固定版本的 Android 系统，这其中可能涉及到一些数据结构体定义的变化。这将导致汇编代码在引用某些结构体中的成员函数或者成员变量时，偏移发生变化。因此，不能简单地使用硬编码的偏移等固定常量来进行二进制代码定位。

**乱序** 在某些情况下，基本块会发生乱序。例如，返回代码（进行栈恢复，函数返回操作的代码）的位置可能会因为编译器优化被放置在程序控制流的中间。所以在寻找漏洞的过程中不能仅仅根据源代码的逻辑顺序来进行搜索。

**函数调用** 随着编程语言的发展，有的函数可能有着不同的形式（如例子 2–2中的 `std::nothrow operator new` 和 `operator new` 函数)。一些二进制文件不直接调用函数，采用了不同的链接模式，使用 PLT 进行函数调用（PLT 会将位置无关的符号转移到绝对地址，当一个导出函数符号被调用时，PLT 会去引用 GOT 中对应的绝对地址，然后转入并执行)。

### 2.1.2 内核防护措施

本文主要针对 Android 系统框架层和内核层漏洞修补。Android 系统基于 Linux 开源内核，相比框架层漏洞的库文件对象，内核对象的二进制代码差异性可能不那么明显。但内核漏洞修补还涉及到内核二进制重写问题。

常见的内核修补方法是利用内核可装载模块（Loadable Kernel Module，即 LKM）来修补内核漏洞（无需重写编译内核，操作系统将动态加载内核模块执行）。但因为 Android 碎片化和厂商代码闭源问题，很难为所有的 Android 设备生成通用的内核可装载模块。此外因为内核模块并不是独立进程，也没有独立的地址空间，而是与内核的其他部分共享内核地址空间，因此如果内核模块出现问题，则会影响损害整个系统内核。

很多设备包括 Google 官方设备 Nexus，在编译选项中默认不支持内核可装载模块；还有一些设备采取了可装载模块校验措施，一些厂商利用 TrustZone 或者内核模块白名单来防止不被认证的模块被加载进内核。因此，使用内核可装载模块来进行内核漏洞修补并不是一个较通用的方案。同时，因为 bootloader 加锁的关系，修补方案并不能采用直接静态重写内核对象并进行替换的策略。另一种策略是动态重写内核内存。由于不同的厂商对 Android 内核采用了不同的内核防护措施来进行自身加固和完整性校验，所以动态重写内核内存可能被内核数据代码段只读（`read-only kernel text/data`）防护措施(如 `CONFIG_STRICT_MEMORY_RWX`[rwx] 和 `CONFIG_DEBUG_RODATA`[rodata]）所阻碍。同时，在修补过程中还必须考虑到如何在内核中找到一块可用的执行代码区域放置修补代码，这其中也涉及到内核代码执行权限的防护。

因此，不同厂商采用的不同的内核防护手段会对内核二进制重写工作造成一定的困难。以下是一些内核漏洞修补涉及到的常见的内核防护手段：

- Android 4.3 引进了 SELinux（Security-Enhanced Linux）来加强了 Android 各进程的访问控制管理。SELinux 分为 `permissive` 和 `enforcing` 两种模式。在 Android 系统 4.3 上，SELinux 默认为 `permissive` 模式，不强制使用；而在 Android 系统 4.4 上，SELinux 只对部分 root 程序（`installd`, `netd`, `vold` and `zygote`）启用。在 Android 系统 5.0 版本中，SELinux 被开始全面强制执行。
- Android 系统支持 NX（No-eXecute），标记了 NX 的内存不支持代码执行。而在 Android 中，系统调用 `vmalloc_exec` 被用于提供可执行内存的分配。
- 内核漏洞修补涉及到对内核代码段的重写。内核防护措施——内核数据代码段只读（read-only kernel text/data）会对这一操作造成极大的阻碍。

## 2.2 框架层漏洞修补研究现状

对于 Android 框架层漏洞修补，最有名的一个系统是 2013 年 Mulliner et al 提出的 PatchDroid[patchdroid] 系统。PatchDroid 基于 `native code` 和 `Dalvik bytecode`，使用动态插桩，将修补的代码插入 Android 上运行的进程中。因为其基于进程级别，当系统重启后，必须重新插入修补代码。07 年提出的基于源码级别的 POLUS[polus] 系统也采用了与 PatchDroid 相似的策略。

另一个较有名的漏洞修补模式是通过移动设备管理系统（MDM）。自带设备办公（Bring Your Own Device）可以帮助公司减少 IT 设备的支出，因此被大量引入企业，但因为其可能涉及大量安全问题，成为传统的 IT 管理的一个重要问题。因此企业引进了 MDM 来管理这些设备，MDM 可以对移动设备的升级（及其他变更）进行部署和管理，在没有厂商参与的情况下自动推送安全更新。MDM 会重新编译打过补丁的源代码，替

换有漏洞的二进制库文件。但是这种方法通常只针对固定版本的 Android 系统。如果厂商没有提供源代码，MDM 则无法重写编译修补漏洞。还有一些开发者发布了自己针对某一漏洞的修改的某些设备的补丁[selfpatch]。他们重新编译源码或者直接修改二进制文件，生成没有漏洞的二进制文件，但这些文件同样只针对某些固定厂商固定版本的设备，并没有普适性。

此外还有一些关于漏洞检测与修补相关的研究方案。Enck et al. 2010 年提出了 TaintDroid[taintdroid]，一种基于 Dalvik VM 变量层的信息流追踪系统。而 Sarwar et al.[sarwar2013effectiveness] 在 2013 年发表了 TaintDroid 并不有效的后续研究。在 2014 年，Ho et al. 提出 PREC 框架[prec] (Practical Root Exploit Containment)。PREC 框架着重于对系统调用的识别。它对高层 (如第三方应用) 的系统调用进行监测，使用一个独立的线程来执行这些系统调用帮助检测和终止不正常的攻击。2010 年提出的 IntPatch[zhang2010intpatch] 系统可以在编译期间自动修补在 C/C++ 程序中的整数溢出漏洞。J. H. Perkins[perkins2009automatically] 则提出了 ClearView，一种基于 Windows X86 二进制文件的调试信息进行漏洞修补的方案。

## 2.3 内核层漏洞修补研究现状

对于 Linux 内核层漏洞修补，学术和工业界提出了热补丁的策略，允许用户在不重启内核的情况下应用安全补丁。

**Ksplicee** Ksplice[arnold2009ksplice] 由 Oracle 开发，以原本的内核源码和 diff 信息作为输入，使用内核模块根据源码的修补代码进行动态修补。Ksplice 把内核中有漏洞的函数替换为新的修补过的函数。Ksplice 在每个函数的开头添加一个跳转，从而实现修补函数的调用。因此，内核中增加了新的函数代码和几个函数调用重定向指令。要想完成内核漏洞修补，Ksplice 要求必须有内核的源代码、生成的修补代码以及编译器。同时编译器必须将内核中的每个函数和数据结构进行隔离。例如，GNU Compiler Collection (GCC) 的标志 `-ffunction-sections` 和 `-fdata-sections` 分别提供了这些功能。但 ARM 相关的编译器并未提供这些功能。

**KGraft** kGraft[kgraft] 由 SUSE 实验室开发，它使用 diff 文件来找到内核漏洞需要修改更新的部分。同样 Kgraft 采取热补丁策略，不需要停止内核。在进行修补的时候，运行的函数使用旧版本或新版本对应的部分，等更新结束后就完全切至新版本执行。

**Kpatch** Kpatch[kpatch] 由 Redhat 开发，包含四个组件：kpatch-build、hot patch module 把 diff 文件转换成热补丁的内核模块；kpatch core module 为注册新函数的内核模块；kpatch utility 为为用户管理模块热补丁的命令行工具。

**Livepatch** Livepatch[livepatch] 由 Canonical 公司开发，原理类似上面三个工具。

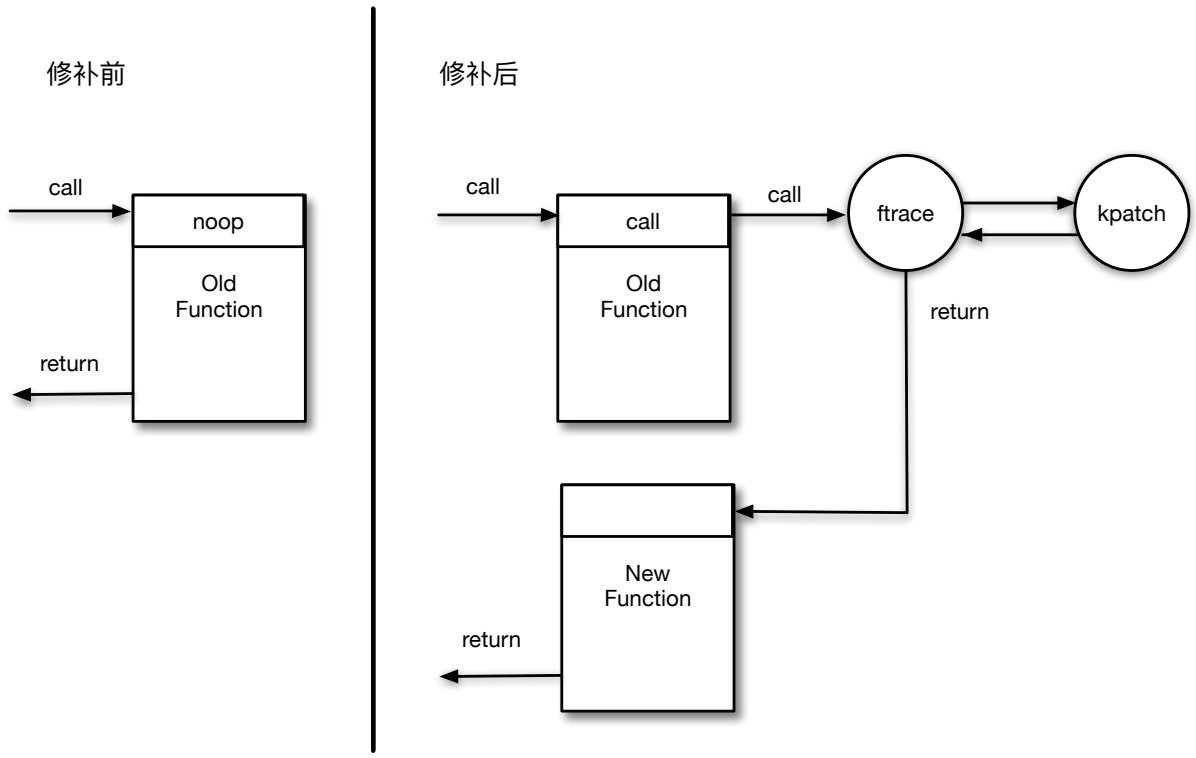


图 2–3 Linux 内核漏洞热修补策略

Fig 2–3 The hot patching process in Linux Kernel

这些方法的工作原理类似，如图 2–3所示，在原有的函数前面加上跳转到新函数的指令。它们通过内核模块替换了整个有漏洞的函数。但这些方法都要求拥有相应厂商对应版本的内核源代码。同时这些方法都是基于 X86 系统，并不兼容 ARM，因此也并不适用于 Android 内核漏洞的修补。此外它们使用内核可装载模块（Linux Kernel Module）的方案，但是使用内核模块需要通过相关校验。如果厂商编译内核时关闭内核模块选项或者启用内核模块白名单保护等策略，该方案将失效。

此外 Kprobes[kprobes] 是 Linux 内核提供的一个调试模块。开发者可以使用 Kprobes 对内核函数进行动态 hook、收集处理器寄存器和全局数据结构等调试信息。所以在一些情况下，也有人使用 Kprobes 进行内核漏洞修补，但同样 Kprobes 对于 Android 内核漏洞修补有着诸多限制条件，并不适用。因此，目前学术界并没有一个针对 Android 的通用的内核漏洞修补方案。

## 2.4 本章小结

本章对当前移动设备漏洞修补技术的难点以及研究现状进行了分析总结。因为 Android 系统生态多样化，导致了 Android 二进制代码的差异性以及内核漏洞二进制重写的复杂性。同时通过对现有的框架层和内核层漏洞修补技术的分析，为后续章节的研究工作提供了参考和方向。

# 第三章 ©¶´¹ϵ µ

## 3.1 ϵ ¼

¢ Embroideryϵ £¬ ¸ Android©¶´ ¹ϵ ¡£¸ϵ » ½ ¼¼ û · Ɪ²»Android 豸 ϵĿ ©¶´ ¦ õİ²õ¸ ¡£

### 3.1.1 ϵ

Embroideryϵ µ 3–1 º£º

² ¯ǵ¿ Embroideryϵ ¹ Googleÿ µİ²õ¸ ¯ ¹ µ CVEµ £¨CVE± š¢CVE ¹ ©¡£ ¶¨λǵ¿ ϵ¸ ¾ CVE ☒ · ºű ©¶´¶¨λ£¬» ©¶´µ λ á¢ ¹ ¡£ Androidϵ ¶ Ķ ½ ☒ λ¡¢ ·

³ ǵ¿ ¬ ¹² Embroidery°¹ǵ ¹µ CVE µ 壬 º» 豸 µ ¹ ǵ° ³ ¹´ ☒ Embroidery ´¹ ½ l Ļ Ɪ ´¶©»½ ☒

¶ ½ ǵ¿ ¶ ½ ǵ¿ Ɪ¶ ¹¤ ö¯ ¹ ·½· ´¹ ¿ µĿ ½ £¬²¢ ±

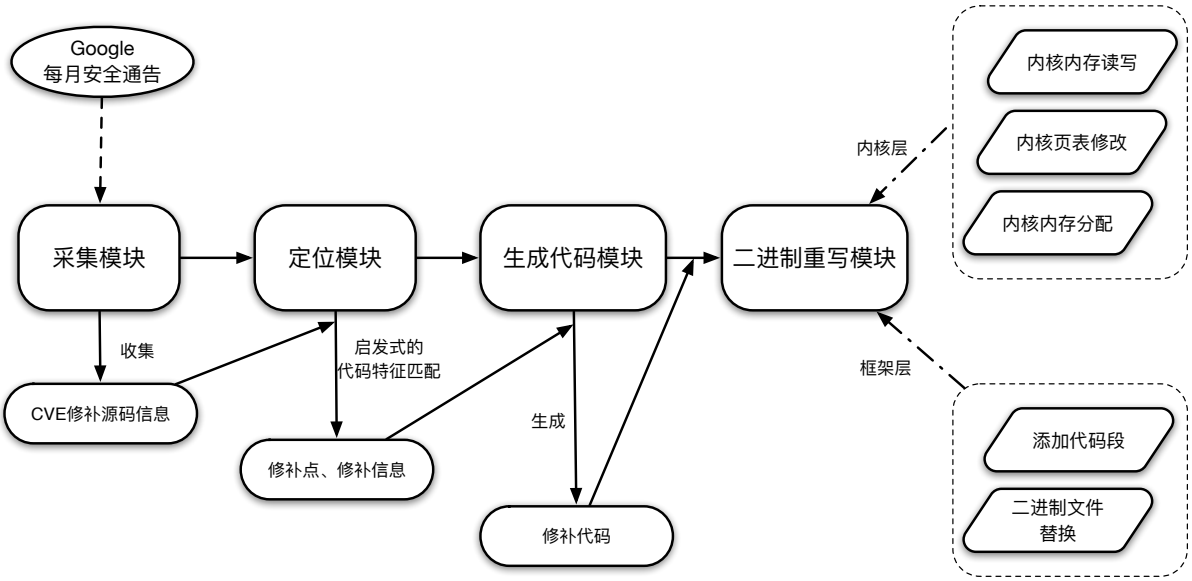


图 3–1 Embroideryϵ µ

Fig 3–1 Overview of Embroideryx