学　　号　2015301500150
密　　级

# 武汉大学本科毕业论文

# 移动应用的动态行为捕获


院 (系) 名 称:　国家网络安全学院

专 业 名 称:　信息安全

学 生 姓 名:　蹇奇芮

指 导 教 师:　傅建明　教授


二〇一九年五月

BACHELOR'S DEGREE THESIS
OF WUHAN UNIVERSITY

# Dynamic behavior capture for mobile applications


School (Department): School of Cyber Science and Engineering

Major: Information Security

Candidate: QiRui Jian

Supervisor: Prof. JianMing Fu



Wuhan University

May, 2019

# 郑 重 声 明

本人呈交的学位论文, 是在导师的指导下, 独立进行研究工作所取得的成果, 所有数据、图片资料真实可靠. 尽我所知, 除文中已经注明引用的内容外, 本学位论文的研究成果不包含他人享有著作权的内容. 对本论文所涉及的研究工作做出贡献的其他个人和集体, 均已在文中以明确的方式标明. 本学位论文的知识产权归属于培养单位.

本人签名: _____ 日期: _____

# 摘　　要

智能移动终端设备的盛行使得针对移动操作系统的恶意应用迅速增加. 目前的移动操作系统中, 安卓系统巨大的市场份额和其相对开放的应用分发和权限管理方式使得其成为的攻击者的主要目标. 为了识别出恶意应用并阻止其传播, 我们需要对应用的行为进行分析. 然而单纯的静态分析在如今安卓应用成熟的混淆和加壳机制的保护下无法很好的揭示应用的行为, 因此需要动态的对安卓应用的行为进行捕获和分析.

本文分析了目前已有的一些安卓系统应用行为监测系统的实现方式和优缺点, 并且通过 hook 技术以及对安卓 8.1 源代码的修改设计和实现了一个运行于 Nexus 5x(Google 的一款智能手机) 的高性能应用动态行为捕获系统. 该系统能够捕获到 Java 层的所有方法调用以及 Native 层的重要函数调用, 并且支持动态地调整需要监控的目标方法 (Java 层) 和函数 (Native 层). 本文使用常用应用对该系统进行了测试, 结果显示与同样能捕获到所有 java 层方法调用的 Android Device Monitor 相比本系统的性能开销明显更低.

本文设计思路结合了 hook 技术带来的灵活性和以及修改源代码的稳定性以及高性能, 对其他开源平台的类似工具设计有一定参考作用, 但在应用中应当注意两种方式可能的冲突问题.

**关键词**: 安卓应用; 动态行为; 高性能

# ABSTRACT

Malicious applications on mobile operating systems boom with the prevalence of smart mobile devices. The huge market share of Android, one of current mobile operation systems, and its relatively open application distribution and privilege management make it attackers' major target. In order to identify malicious applications and prevent them from spreading, we need to analyze the behavior of applications. However, only static analysis can not handle the mature obfuscation and packing mechanism of Android applications, so it is necessary to dynamically capture and analyze the behavior of Android apps.

In this paper, I analyze the implementation, advantages and disadvantages of some existing Android application behavior monitoring systems, and present a high-performance application dynamic behavior capture system, which can run on Nexus 5x and is implemented by using hook technology and modifying Android source code. The system captures all method invocations of Java layer and important function calls of Native layer, and supports dynamic adjustment of the target methods (Java layer) and functions (Native layer) that need monitoring. I evaluate the system with common applications and the result shows that the overhead is significantly lower than that of Android Device Monitor when capturing all java layer method invocations.

The design of the system in this paper combines the flexibility brought by hook technology and the stability and high performance brought by modification of source code, which can be used for designing similar tools of other open source platforms, but we

should pay attention to the possible conflicts between the two approaches;

Key words: Android application; dynamic behavior; high performance

# 目　　录

# 1 绪论

## 1.1 研究背景与意义

随着移动互联网和物联网的蓬勃发展, 智能移动终端设备迅速普及。截止 2018 年 9 月, 国内智能手机用户数量已达到 7.8 亿[5], 占总人口数的 55% 以上。如此众多的用户极大地促进了移动应用的发展, 2018 年的数据显示[7], 谷歌公司的 Google Play 上已经有超 260 万应用软件, 苹果公司的 App Store 上也有超过 200 万应用软件可供下载使用。这些应用软件涵盖了娱乐, 社交, 购物, 出行, 金融服务, 身份服务等等领域, 极大地便利了人们的生活。但与此同时, 各种服务通过应用软件集中于智能手机使得智能手机与个人隐私, 财产安全甚至人身安全的联系变得更加紧密, 从而不可避免地吸引了大量攻击者开发和传播恶意应用来牟取不正当利用。

目前市场上的智能移动终端设备运行的移动操作系统几乎均为 Android 和 Apple iOS[6]。其中 Android 以其免费, 开源的特点吸引了大量智能手机厂商, 占据了超过 75% 的市场份额[6]. 最新数据显示, 在 2019 年 Q1 国内智能手机销售量中搭载 Android 的手机销量占比达 78.2%[4]。然而, Android 本身宽松的权限管理以及开放的应用分发方式使其很容易受到攻击, 加上巨大的市场体量, 造成了绝大多数恶意应用把 Android 作为攻击目标的局面。虽然近年来 Android 的权限管理和安全机制不断加强, 同时工信部各大应用分发平台的监管加强, 一定程度上遏制了恶意应用的发展, 但数据显示[11]2018 年 Android 新增恶意软件达 800.62 万个, 感染用户数近 1.13 亿, 数量仍然庞大。另外, 恶意软件的类型也持续朝着多样化隐秘化方向发展，新式的恶意软件通过更加难以分析的加壳和混淆技术隐藏自己的恶意行为, 绕过安全软件的查杀。因此, Android 平台的安全问题依然严峻。

为了阻止恶意应用被下载运行, 保护用户手机的信息安全, 各大应用分发平台需要能够精确有效地判断开发者提交的应用是否为恶意应用, 而捕获应用的行为是分析一个新应用是否为恶意应用的必要前提。对应用软件的行为获取方法有两

个大类: 静态方式和动态方式。静态方式即在不运行应用软件的情况下对应用软件内部的资源文件、代码、数据等进行分析, 获取应用的特征, 代码逻辑等; 动态方式则是运行应用软件, 在执行过程中对软件的代码执行路径, 数据访问等进行监控和记录。在目前 Android 平台的应用加壳和混淆技术成熟的情况下, 单独的静态分析无法获取包含应用的真正逻辑的代码和数据, 因而无法获取到应用行为, 必须通过动态的方式才能捕获到包含应用真实目的的行为, 获取相应的数据, 从而判断应用是否为恶意应用。另外, 动态分析还能够在运行中捕获到执行应用真正逻辑的代码和数据 (脱壳), 从而结合静态分析揭示更加完整的应用行为。因此, 对 Android 平台移动应用的动态行为捕获技术进行研究, 有助于识别和分析隐蔽性越来越强的恶意应用, 从而遏制恶意应用的传播, 提升 Android 平台的安全性。

## 1.2 国内外研究现状和发展方向

Android 系统从发布至今已有 10 年, 目前国内外已有许多 Android 应用动态分析相关的研究成果发表。这些成果借鉴了传统 PC 平台的动态分析方法, 并结合了 Android 平台的自身特点, 在本地指令层面, 系统调用层面, 本地函数层面, Java 指令层面, Java 方法层面中部分或全部层面对应用的运行进行跟踪记录, 并在此基础上结合污点传播技术实现了隐私数据泄露的检测功能, 结合对应用加壳混淆机制的研究实现了脱壳和去混淆功能, 给恶意应用的分析提供了许多强有力的工具。下文介绍了一些有代表性的成果。

Enck William 等构建了名为 Taintdroid[3] 的隐私数据跟踪系统。该系统采用了污点传播技术, 通过修改 Android 系统 Java 层与隐私数据获取相关的 API 给隐私数据添加标记, 通过修改 Android Runtime 的 Dalvik 虚拟机运行机制实现了带标记隐私数据在虚拟机内部的透明传播, 通过修改 Android 系统进程间通信的接口实现了带标记隐私数据跨进程传播, 通过修改 Java 层文件和网络的 API 实现记录带标记的隐私数据去向, 从而能够检测到应用泄露隐私数据的行为。不过该系统有以下局限性: 1. 没有对应用的所有敏感行为进行监控, 例如拨打电话, 发送短信等; 2. 没有对 Native 层的函数进行监控, 无法检测到应用通过 JNI 接口调用自身 Native 模块泄露隐私数据的行为; 3. 针对特定 Android 版本, 并且不再支持 Android4.3 以后的版本使用;

Desnos Anthony 等构建了名为 Droidbox 的[2] 动态分析系统。该系统使用了 Taintdroid[3] 来监控隐私数据泄露, 另外通过修改 Android 系统源代码中敏感 API 的方式实现对 Java 层的电话, 短信, 网络, 文件, Java 类动态加载, 加密等 API 调用的监控, 能够记录应用在 Java 层的敏感行为。之后为避免频繁修改系统以适应 Android 版本变化, 该系统更改了监控 Java 层敏感 API 调用的方式, 通过反编译需要监控的应用并在敏感 API 调用前插入监控代码, 再重新打包生成修改后的应用的方式实现监控, 并为将实现新的监控方案的工具命名为 APIMonitor[1]。但该系统只涉及了 Java 层预定义的敏感 API 的监控, 没有实现对应用自身的 Java 方法调用的监控, 并且没有实现对应用本地函数层调用的监控, 因此无法完整的揭示应用的行为; 另外该系统也针对特定 Android 版本开发, 并且不支持 Android4.1 之后的系统使用; 对于采用修改应用本身实现监控的 APIMonitor, 由于目前加壳和混淆以及应用完整性检查技术的成熟, 已经几乎失效。

Yan Lok Kwong 等构建了名为 DroidScope[10] 的动态分析系统。该系统通过修改运行 Android 系统的 qemu 虚拟机以及 Android 系统中的 Dalvik 虚拟机实现了了对本地指令层面和 Java 指令层面的监控追踪, 并在此基础上实现了污点传播分析数据泄露, 监控 Java 和本地次层敏感 API 调用等功能。该系统在底层实现了对应用运行的全面监控, 并提供了接口用以在特定事件 (例如执行系统调用, 执行本地函数, 读写内存等) 发生时添加自定义的处理逻辑, 可以用来开发特定用途 (例如脱壳) 的工具。但是该系统也有一些局限性: 1. 提供了对底层执行的监控功能因而性能开销较大; 2. 依赖于虚拟机环境, 而部分恶意应用会检测虚拟机运行环境从而隐藏自己的恶意行为, 使得分析结果不准确 3. Java 部分的监控通过修改 Android 系统中 Dalvik 虚拟机来完成, 对特定 Android 版本有效, 然而目前 Android 系统已经使用 Art 虚拟机代替了 Dalvik 虚拟机, 该系统无法应用于目前的应用分析。

Tam Kimberly 等构建了名为 CopperDroid[8] 的动态分析系统。该系统基于 qemu 虚拟机, 通过 VMI 技术捕获应用运行时调用的系统调用序列及相应参数, 然后通过解析记录的系统调用序列和对应参数重建出应用在本地函数层面和 Java 方法层面的具体行为, 例如发送短信, 拨打电话, 进行网络传输, 进行文件读写, 启动进程, 进程间通信等。由于只需要系统调用序列, 该系统不需要对 Android 系统进行修改, 能够较好的兼容 Android 版本的升级, 但仍由一些局限性: 1. 系统基于虚拟机环境, 受

恶意应用针对虚拟机攻击的影响 2. 没有对 Java 层进行具体的监控, 会丢失掉一些 Java 层的行为 3. 重构应用行为依赖于特定 Java 层行为与系统调用的映射关系, 而这个关系可能发生变化而使得结果不准确

Xue Lei 等构建了名为 Malton[9] 的动态分析系统。

## 1.3 论文主要工作

本文分析了目前已有的一些安卓系统应用行为监测系统的实现方式和优缺点, 并且通过 hook 技术以及对安卓 8.1 源代码的修改设计和实现了一个运行于 Nexus 5x(Google 的一款智能手机) 的高性能应用动态行为捕获系统. 该系统能够捕获到 Java 层的所有方法调用以及 Native 层的重要函数调用, 并且支持动态地调整需要监控的目标方法 (Java 层) 和函数 (Native 层). 本文使用常用应用对该系统进行了测试, 结果显示与同样能捕获到所有 java 层方法调用的 Android Device Monitor 相比本系统的性能开销明显更低.

## 1.4 论文组织结构

根据本文研究的特点, 本文的内容按如下方式组织:

第一章为绪论, 主要说明了本文课题的研究背景和研究意义, 简述了国内外对本文课题的研究成果及相关工具和技术, 介绍了本文的主要工作内容和文章组织结构。

第二章为背景技术介绍, 主要讲述了 Android 系统的基本架构, Android 应用的基本结构, Android 平台的常用动态分析技术和应用保护技术, Android Runtime 的运行机制, 以及本系统用到的一个 hook 框架—Frida。

第三章为系统设计和实现, 详细说明了本文提出的应用动态行为捕获系统的设计实现方案。

第四章为实验与结果分析, 描述了对本系统进行测试的实验环境, 实验方法并对实验结果进行了分析和总结。

第五章为总结与展望, 主要是整理本文所做的工作，并简要分析了本文提出系统的局限性和改进方案。

# 2　背景技术分析

## 2.1　Android 系统架构

Android 系统由多个软件层次构成, 这些层次功能分明, 每一层都对其上的一层提供服务, 构成一个 5 层的软件栈。软件栈最底层为 Linux 内核层, 其上为硬件抽象层, 之后为本地函数库层, 该层次包括了 Android Runtime 和其他的一些本地函数库, 再上层为 Android 框架层, 该层包括了提供给应用程序的 API 和系统管理服务程序, 最上层为应用层, 该层次为用户直接交互的应用程序运行的层次。图2.1给出了各层次的组件和关系。


图 2.1　Android 系统架构

**Linux 内核层**

Android 基于修改的 Linux 内核构建, Linux 内核为 Android 系统提供了操作系统的基本功能, 包括进程管理, 内存管理, 文件管理, 进程间通信 (共享内存和 Binder), 网络协议栈, 电源管理, 许多设备驱动程序 (音频, 视频, 蓝牙, 相机, 键盘, USB, WIFI 等) 以及访问控制机制 (基于用户和用户组的访问控制和 selinux)。这些功能通过系统调用的方式提供给上层使用, 因此, 监控系统调用的使用情况能够获取到应用对关键资源的访问行为。

**硬件抽象层**

硬件抽象层是定义了用于更高层次调用对应硬件驱动的接口, 屏蔽了不同厂商的同种设备驱动的差异, 降低 Android 系统与硬件的耦合度, 便于 Android 系统的移植。硬件抽象层包含多个库模块，其中每个模块都为特定类型的硬件组件实现一个接口，例如相机或蓝牙模块。当更高层次要求访问设备硬件时，Android 系统将为该硬件组件加载库模块。

**本地库层**

本地库层由许多由 C/C++ 开发的系统运行库组成。这些运行库主要分为两部分, 第一分部为 Android 运行时环境相关的库, 第二部分为其他系统运行库。

Android 运行时环境由给应用提供 Java 运行环境的虚拟机实现和实现 Java API 的核心运行时库组成。虚拟机实现在 Android4.4 之前为 Dalvik 虚拟机, Android4.4 时 ndroid Runtime(ART) 虚拟机作为实验特性加入, 并喝 Dalvik 虚拟机共存, Android5.0 之后只保留了 ART 虚拟机。Android 框架层的许多服务程序和应用层的应用软件就运行在自身的虚拟机实例中。核心运行时库，可提供 Java API 框架使用的 Java 编程语言大部分功能。

其他系统运行库包括许多重要的功能的实现, 主要包括以下部分: AUDIO MANAGER 用于管理音频输入输出; LIBC 提供了 c 语言标准函数库; MEDIA FRAMEWORK 提供了对常见音频和视频处理的支持; OPENGL/ES 提供了 2D/3D 图形绘制功能; SQLITE 提供了访问 SQLite 数据库的函数; SSL 提供了常见的加密功能;

SURFACE MANAGER 提供对显示子系统的支持和管理; WEBKIT 提供了浏览器引擎的实现。

本地库层的各种功能函数除了提供给 Android 系统自身以实现系统服务功能, 还可以通过 Android Native Development Kit(NDK) 让应用程序通过 Java Native Interface(JNI) 调用, 因此对本层函数调用情况的监控可以获取到应用的行为。

### Android 框架层

Android 框架层包括了许多系统服务程序和组件以及提供给应用程序的访问系统资源的 Java API。这些服务程序和组件主要包括以下几部分:

1. 资源管理器，用于访问非代码资源，例如本地化的字符串、图形和布局文件

2. 通知管理器，可让所有应用在状态栏中显示自定义提醒

3. Activity 管理器，用于管理应用的生命周期，提供常见的导航返回栈

4. 内容提供程序，可让应用访问其他应用（例如“联系人”应用）中的数据或者共享其自己的数据

5. 丰富、可扩展的视图系统，可用以构建应用的 UI，包括列表、网格、文本框、按钮甚至可嵌入的网络浏览器

Android 框架层是与应用程序联系最紧密的层次, 也是应用程序最容易访问系统资源的层次, 因此对该层次提供的 API 的监控能够显示应用的主要行为。

### 应用层

应用层包括了所有用户直接使用的应用软件, 例如电话、短信、浏览器、微信、支付宝等。这些应用软件主要由 Java 语言开发, 通过调用 Android 框架层提供的 API 和 Java 语言的标准 API 实现功能, 每个应用运行于自己独立进程中的虚拟机实例中, 多个应用间借助框架层提供的 API 通信 (进程间通信最终由内核实现)。利用 NDK, 应用也可以实现自己的本地库, 访问 Android 系统本地库层的开放甚至隐藏的函数, 并通过 JNI 在应用的 Java 部分调用自身的本地库中的本地函数。由于应用能够直接执行本地代码, 增加了应用程序行为涉及的层次，需要同时在 Java 层次和本地层次监控应用的执行才能获取到应用的所有行为。

## 2.2 Android 应用结构

Android 应用程序主要以 Android Package(APK) 的文件形式分发和安装。APK 文件本质上是一种 zip 压缩文件, 由多个文件和文件夹组成其中包含了应用的代码文件、资源文件、证书文件和清单文件, 以 apk 作为文件后缀名。图2.2给出了 APK 文件的内部结构。

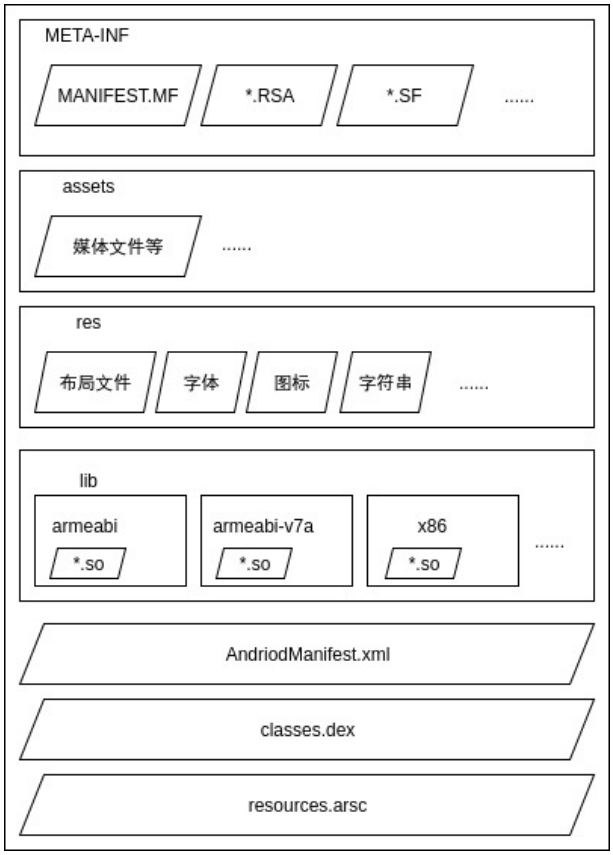


图 2.2 APK 文件结构

META-INF 该文件夹包含了与 APK 文件的签名和校验相关的文件, 一般应该包括至少 3 个文件:MANIFEST.MF、*.RSA、*.SF(“*”表示文件名不确定)。其中 MANIFEST.MF 记录了 APK 中的所有文件名和经过 base64 编码的 SHA256 校验值(不包括自身); *.SF 记录了 MANIFEST.MF 文件的经过 base64 编码的 SHA256 校验值以及 MANIFEST.MF 中每一项记录的经过 base64 编码的 SHA256 校验值; *.RSA 文件中保存了公钥、所采用的加密算法以及对 CERT.SF 中的内容的用私钥进行加密之后的值。

assets　该文件夹包括了 res 中定义类型之外的其他类型的资源文件，例如音频文件，视频文件等媒体文件。应用还可以把本地库文件放在这个文件里在运行时根据需要动态的加载。

res　该文件夹包括了没有编译到 resource.arsc 中的常用的预定义类型的资源文件。

lib　该文件夹包括了应用的本地库文件,根据适用的 Application Binary Interface(ABI)不同，这些本地库文件会被放在不同的子文件夹里。当 APK 被安装时, 系统会选择合适的本地库文件进行安装, 在启动应用时系统会自动加载对应的本地库文件。

AndroidManifest.xml　该文件包含了应用的配置信息, 具体来说包含以下几个方面:

1. 描述了应用包名;

2. 描述了应用的所有组件, 包括构成应用的 Activity、服务、广播接收器和内容提供程序, 以及这些组件可以处理的 Intent 消息;

3. 描述了应用需要使用的权限

4. 声明了应用所需的最低 Android API 级别 (与 Android 版本相对应)

5. 列出应用必须链接到的本地库

通过 AndroidManifest.xml 我们可以得到应用的基本信息。

classes.dex　该文件为应用的 Java 代码编译后的运行于 ART 或者 Dalvik 虚拟机的可执行文件。该文件包含了应用自定义的类的实现代码, 通过反编译该文件可以得到应用的 Java 源代码, 因此通常会使用加壳和混淆的手段隐藏该文件真正的内容。对于大型应用, 可能会有多个 dex 文件。

resources.arsc　该文件为编译后的资源文件, 包括 xml 布局文件, 字符串资源文件等等。

## 2.3 Android 动态分析技术

## 2.4 Android 加壳和混淆技术

## 2.5 Android 运行时环境

### 2.5.1 简介

### 2.5.2 应用的启动

### 2.5.3 应用的加载

### 2.5.4 方法的执行

## 2.6 Frida

Frida 是一个著名的开源 hook 框架。

# 3 系统设计实现

## 3.1 概览

## 3.2 启动监控

## 3.3 Java 方法执行监控

## 3.4 native 函数调用监控

## 3.5 log 系统

# 4　实验与结果分析

## 4.1　监控数据

## 4.2　性能

# 5　总结与展望

本系统还不够成熟

# 参考文献


[1] droidbox - apimonitor.wiki. https://code.google.com/archive/p/droidbox/wikis/APIMonitor.wiki.

[2] A. Desnos and P. Lantz. Droidbox: An android application sandbox for dynamic analysis. Lund Univ., Lund, Sweden, Tech. Rep, 2011.

[3] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones [c]. In Proceedings of the 9th USENIX conference on Operating systems design and implementation. USENIX, pages 1–6, 2010.

[4] Kantar. Smartphone os sales market share evolution. https://www.kantarworldpanel.com/global/smartphone-os-market-share/, 2019.

[5] newzoo. Top 50 countries/markets by smartphone users and penetration. https://newzoo.com/insights/rankings/top-50-countries-by-smartphone-penetration-and-users/, 2018.

[6] statcounter. Mobile operating system market share worldwide. http://gs.statcounter.com/os-market-share/mobile/worldwide.

[7] statista. App stores - statistics & facts. https://www.statista.com/topics/1729/app-stores/.

[8] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro. Copperdroid: Automatic reconstruction of android malware behaviors. In Ndss, 2015.

[9] L. Xue, Y. Zhou, T. Chen, X. Luo, and G. Gu. Malton: Towards on-device non-invasive mobile malware analysis for {ART}. In 26th {USENIX} Security Symposium ({USENIX} Security 17), pages 289–306, 2017.

[10] L. K. Yan and H. Yin. Droidscope: Seamlessly reconstructing the {OS} and dalvik semantic views for dynamic android malware analysis. In Presented as part of the

21st {USENIX} Security Symposium ({USENIX} Security 12), pages 569–584, 2012.

[11] 腾讯移动安全实验室. 腾讯移动安全实验室 2018 年手机安全报告. https://m.qq.com/security_lab/news_detail_489.html, 2018.

# 致　　谢