**摘 要**

许多Android应用程序使用SSL / TLS安全的传输敏感信息。然而，开发人员通常会提供自己的实现标准的SSL / TLS证书的验证过程。但是，许多这样的自定义实现过程有细微的错误，都内置了对异常自签名证书，或盲目断言所有证书都是有效的，使很多android应用容易被中间人攻击。在本文中，我们提出了SMV- HUNTER 系统，结合了静态和动态分析，该系统能够大规模的自动识别此漏洞。静态组件，当检测到一个自定义的验证程序，从而确定潜在的易受攻击的应用程序，并提取信息用于指导动态分析，然后使用用户接口枚举和自动化技术来触发下一个潜在的中间人漏洞。我们利用SMV- HUNTER系统评估了从谷歌应用市场下载的23,418的应用程序，其中1,453应用程序被确定为潜在的易攻击，通过静态分析潜在的易受攻击的应用程序，同时运行16个进程，每个应用的平均开销约为4秒，通过动态分析，确定了726个漏洞应用。同时8个仿真器运行，每个应用耗时44秒

**关键词：中间人攻击 静态分析 动态分析 Android应用 SSL/TLS**

**Abstract**

Many Android apps use SSL/TLS to transmit sensitive information securely. However, developers often provide their own implementation of the standard SSL/TLS certificate validation process. Unfortunately, many such custom implementations have subtle bugs, have built-in exceptions for self-signed certificates, or blindly assert all certificates are valid, leaving many Android apps vulnerable to SSL/TLS Man-in-the-Middle attacks. In this paper, we present SMV-HUNTER, a system for the automatic, large-scale identification of such vulnerabilities that

combines both static and dynamic analysis. The static component detects when a custom validation procedure has been given, thereby identi-fying potentially vulnerable apps, and extracts informationused to guide the dynamic analysis, which then uses userinterface enumeration and automation techniques to trigger the potentially vulnerable code under an active Man-in-the-Middle attack. We have implemented SMV-HUNTER and evaluated it on 23,418 apps downloaded from the Google Play market, of which 1,453 apps were identified as being potentially vulnerable by static analysis, with an average overhead of approximately

4 seconds per app, running on 16 threads in parallel. Among these potentially vulnerable apps, 726 were confirmed vulnerable using our dynamic analysis, with an average overhead of about 44 seconds per app, running on 8 emulators in parallel

**Keyword:Man-in-the-Middle attcks static analysis dynamic analysis Application SSL/TSL**

目录

[**第一章 绪论** 4](#_Toc388948608)

[**第二章 背景介绍** 5](#_Toc388948609)

[2.1 SSL/TSL 5](#_Toc388948610)

[2.2 Android UI组件 5](#_Toc388948611)

[**第三章 系统概述** 6](#_Toc388948612)

[3.1 问题描述 6](#_Toc388948613)

[3.2 技术的难题和关键 6](#_Toc388948614)

[3.3 系统概述 8](#_Toc388948615)

[**第四章 静态分析** 8](#_Toc388948616)

[4.1 反编译应用 9](#_Toc388948617)

[4.3 易受攻击的入口点标识 9](#_Toc388948618)

[4.4 智能输入 10](#_Toc388948619)

[**第五章 动态分析** 12](#_Toc388948620)

[5.1 设备管理 12](#_Toc388948621)

[5.2 UI自动化 13](#_Toc388948622)

[5.3 MITM代理 15](#_Toc388948623)

[5.4 相关性分析 15](#_Toc388948624)

[**第六章 评价** 15](#_Toc388948625)

[6.1 数据集的特点 16](#_Toc388948626)

[6.2 静态分析 16](#_Toc388948627)

[6.3 动态分析 17](#_Toc388948628)

[6.4.有漏洞的应用程序 17](#_Toc388948629)

[6.5 总结有漏洞的应用程序 18](#_Toc388948630)

[**第七章 相关工作** 18](#_Toc388948631)

[**第八章 讨论和未来的工作** 18](#_Toc388948632)

[**第九章 总结** 19](#_Toc388948633)

[**致谢** 19](#_Toc388948634)

# 第一章 绪论

智能手机的普及，导致了许多软件厂商向移动领域扩展服务。目前在谷歌应用商店，有超过100万的应用，超过500亿的下载量[[1]](#endnote-2)[38],对于许多的应用，首要关注的问题是在网络上安全的传输数据，为了确保安全，许多应用使用HTTPS协议（基于SSL的HTTP/TLS），该协议用来确保安全，防止恶意的攻击者能够拦截和修改应用程序与服务器的网络通信。

但是，如果客户端没有正确的验证SSL/TLS（以下简称SSL）证书，这可能导致一个中间人攻击[[2]](#endnote-3)[27]，在中间人攻击中，攻击者可以截获并修改在客户端和服务器之间传输的数据，通常情况下，如果客户端正确的验证证书，攻击者无法解密传输的数据。然而，如果客户收证书后没有检查签名，或者如果客户接收自签名证书但不提示客户，攻击者可以通过提供虚假证书伪装成服务器。在这种情况下，攻击者可以用虚假证书将传输数据解密，并且可以随意读取和修改。

因此，为了自动，大规模鉴定SMV，我们需要一套新的技术和工具。这些技术的发展，是本文的重点。 具体来说，我们建议使用SMV-HUNTER技术，该技术能自动识别Android应用程序。关键的问题是，可靠地检测SMV，只使用静态分析是不够强大。智能的输入，运行时数据相关性和用户交互，。这种非确定性导致静态分析的不准确。因此，为了自动检测SMV， 我们必须包含动态分析，其中的目标应用 在执行过程中被实际观察到。在这种方法中， 在检测漏洞的关键步骤是通过模拟用户交互触发漏洞。

另一方面，一个纯粹的动态的方法将穷举所有可能的用户界面（UI）路径，这效率非常低。此外，在输入数据之前，应用程序经常验证或转换输入文本。纯粹动态的方法难以提供有效的输入文本， 因为它不能确定是什么验证或转换应用程序将适用于输入文本。为了解决这些问题，我们提出混合静态和动态的方法，其中静态分析使用修剪搜索空间，引导动态分析， 并提供更有效的文本。我们还开发了一种新的用户界面 自动化框架，它可以智能地分析运行 通过与Android的窗口管理器进行交互和应用程序 无需人工干预，提供智能的输入。

综上所述，本文提出了以下观点：

* 提出SMV-HUNTER，一个新的系统来自动分析SMV 在Android应用程序中。 SMV-HUNTER包含一个静态分析组件找出潜在的脆弱的应用程序，识别该应用程序的入口点，导致脆弱的行为， 并生成智能输入文本字段;和动态 分析组件，以确认该漏洞。此外， SMV-HUNTER是建立在模块化。它可以 很容易被重新用于其他任务，如软件 测试其他漏洞或检测。
* 开发一个完全自动化框架，通过多个仿真器运行android应用，从而收集重要信息，如应用程序日志，并可以进行网络和系统调用跟踪
* 2012年7月，从谷歌应用商店收集了23418个android应用，通过SMV-HUNTER分析

其中静态分析确定了1453个应用存在漏洞，然后通过动态分析确定了726个

本论文的其余部分安排：第二章，简述SMV-HUNTER的技术背景知识。第三章，概述SMV\_HUNTER系统。第四章，描述SMV-HUNTER中的静态分析。第五章，描述SMV-HUNTER中的动态分析。第六章，提出实验评估方案。第七章，回顾相关工作。第八章，讨论系统的局限性和未来工作。第九章，总结论文

# 第二章 背景介绍

## 2.1 SSL/TSL

根据RFC2818,2246和3280列举的规则来建立安全的SSL/TLS连接，客户端必须检查证书链是否有效，以及，该主机名是否有效。一个有效的证书链包含：

* 证书链中的每个证书都没有过期
* 根证书是由受信任的认证机构（CA）颁发给客户端的默认密钥
* 如果证书链中有多个证书，证书链应该通过检查来验证每一个由CA签署的证书链。如果只有一个证书，这就是自签名的，因此它是在根CA链，受上述验证。

Android操作系统提供内置的SSL证书验证和主机名验证，其中包括一个keystore，但允许开发人员通过提供自己的实现创建实现了这个X509TrustManager和类的HostnameVerifier接口

有许多的原因，开发者可能会选择重写SSL证书验证程序[[3]](#endnote-4)[9]

* 在Android平台的早期版本中，有SSL证书验证过程中的错误， 造成了一定的有效证件被拒绝[[4]](#endnote-5)9。
* 如果一个应用程序连接到服务器的证书的根CA是不存在的密钥库，应用程序不能建立使用内置证书的安全连接验证。
* 为避免采购发展有效证件的费用，测试和用户接受的环境中，

开发人员经常使用自签名或无效的证书。

* 一些流行的第三方库，如ACRA覆盖内置的SSL证书验证与弱势实现，使任何应用程序使用了这种库缺陷[[5]](#endnote-6)[32]。

## 2.2 Android UI组件

Android应用程序的可视化组件称为活动，它创建画面，这类似于在Windows典型的桌上型计算环境中[[6]](#endnote-7)[1]。这些活动创建并撰写UI组件通过使用声明在一个XML文件，或编程方式在运行。此外，活动可以使用碎片，这是可重复使用的UI组件允许开发人员定义和管理

屏幕在运行时，不切换活动[[7]](#endnote-8)1。那是在一屏可以由UI组件可分为三大类：可编辑的组件，可点击的部件和静态部件。可编辑的组件有一个内部的状态下，可以进行修改，例如文本框，复选框和单选按钮。可点击的组件是组件没有状态，导致某些行动时，抽头，如按钮或链接。静态组件不响应用户的交互。还有其他的方法就是行动能被触发，诸如菜单或主页按钮，分别是从不同设备的触摸屏。需要注意的是SMV - HUNTER只会在屏幕上的组件进行交互

通常情况下，每个Activity都与一个屏幕相关联，但它也可以使用一个单一的活动为整个应用程序[[8]](#endnote-9)[20]。因此，定义一个窗口屏幕的显示内容 在给定时间点。然后，一个典型的Android应用程序的UI会 由一个独立的窗口的集合。具体而言，我们 代表一个用户界面为一个有向图G，其节点对应到窗口，而其边缘对应的动作引起 通过点击组件。如果操作不改变UI显示，我们代表这是一个自我循环。这个抽象帮助我们描述复杂的用户界面简洁，并制定UI自动化算法在图形方面遍历。

# 第三章 系统概述

在本章A，定义了研究的问题，然后在B解决这个问题，最后在C得到系统的概述

## 3.1 问题描述

SMV-HUNTER的目标是识别SMV在Android中应用程序。解决方案必须是完全自动的，可扩展到非常大型数据集。为此，我们有四个主要的设计目标：

* **覆盖范围**：我们的目标是有一个可以应用到整个市场的工具，这意味着它需要能够运行尽可能多的应用程序。在android系统中，由于很多应用程序依赖专有的组件，必须解决这个难题
* **效率**：系统应该是有效的，足以同时运行大量的应用。应避免静态分析表明不容易受到攻击的代码路径。
* **健壮性**：Android模拟器[[9]](#endnote-10)[4]和Android 调试桥（ADB）[[10]](#endnote-11)[3]由谷歌提供有 长时间运行时具有不稳定的问题[[11]](#endnote-12)[8]，[[12]](#endnote-13)[10]。模拟器趋于切换到“脱机”状态，不响应。该系统应 能够避免这样的问题，或者检测到它们并采取 纠正措施。
* **精度**：系统应能够检测到缺陷应用程序没有误报。

系统如果满足这些要求可以在使用在市场层面（例如，在谷歌应用商店）强制实施更严格安全要求，或谁想要执行机构对员工的设备严格的安全要求：软件在设备上可以防止安装不审核的应用程序， 和员工请求的应用程序，它们可以被自动地分析并有效，而不需要手动维护白名单

## 3.2 技术的难题和关键

为实现这些目标，我们确定了以下挑战与技巧：

**模拟用户交互**  与模拟用户交互应用程序需要了解什么是被显示在屏幕和提供智能输入。作为一个例子，考虑一个在线银行应用程序的登录屏幕。一个典型的登录页面将包含用户名和密码文本框，还可能有“记住我”复选框，登录按钮点击后，将提交用户的凭据。该用户通常将输入提供给这些元件的顺序进行首先是用户名，并用自动点击登录按钮。一个有用的UI自动化组件应当能够模拟这种行为，无需人工干预或指导。

分析现有工具的UI自动化之后，我们得出的结论是，这些挑战需要新的技术。特别地，

谷歌的Monkey工具[[13]](#endnote-14)[17]不能准确地模拟的用户的控制行为，因为它提供的随机 UI事件。另一个现有的UI自动化框架是 Robotium[[14]](#endnote-15)[14]，这是Android的广泛使用的一种流行的工具 开发人员经常用它进行测试。这个框架是紧密结合 Android的仪表框架，从而导致Robotium 测试脚本与目标应用程序进行紧密耦合。每个目标应用程序需要一个独特的测试脚本，这使得它不适合作为一种通用的UI自动化解决方案

**管理应用程序状态** 在第二章B部分，将一个Android应用程序的UI作为一个有向图。然后，在一个应用程序的执行过程中，用户界面就像一个状态机，其中应用程序的“状态”是当前窗口的显示。跟踪这种状态下，以指导搜索程序的用户界面。被确定为有缺陷的“目标”节点入口点，并且希望探索该程序的代码路径 从这些入口点开始。为了防止过多的支出 太多的时间探索任何单一的入口点，希望限制搜索到目标节点。强制执行此限制，系统必须能够检测到的应用程序时转换到一个新的状态

现在存在的系统缺少关键的功能。而文献[[15]](#endnote-16)[41]提供了框架建议 通过削减导致所有代码路径类似的功能无用的活动，这是不一般的，并且需要定制 Android操作系统的。因此，系统利用所提供的FocusChange和WindowChange事件 Android的ViewServer

**测试效率** UI自动化通常是一个缓慢而 资源密集的过程。在我们的实验中，我们发现，用户界面 自动化平均需要约45秒 遍历所有可能的UI路径上的一个窗口。为了 保持可行性，我们使用静态分析技术，大幅 减少窗口，我们必须测试的数量。我们的系统 拆开每个应用程序，并检查自定义实现 该X509TrustManager的HostnameVerifier或接口。如果任何该等执行被发现，我们构造一个方法调用绘制和跟踪的潜在漏洞的调用代码返回到相应的窗口。然后，我们可以消除应用程序不提供这样的自定义实现，因为他们将使用（正确）内置的SSL证书验证过程。在那些不重写这些接口的应用程序，我们的UI自动化组件可以限制自动化那些入口点的窗口确定为调用重载的SSL证书验证界面。

即使在我们从静态分析实现的，可用的Android应用程序的数量之多使得连续测试应用程序慢得不得了的。以达到一个足够加速比，以使测试大市场是可行的，我们必须测试以并行方式多个应用程序

**大规模自动化** 另一个重大挑战是业务流程 多的Androi模拟器的并行执行 UI自动化。谷歌的MonkeyRunner[[16]](#endnote-17)[12]工具被设计为支持并行运行多个仿真器，但它 几个缺点，以防止它的使用在我们的大型自动化方案。的一个更严重的问题是 它缺乏误差传播。大部分所提供的方法 由MonkeyRunner API有没有返回值，异常也不例外。因此，API的用户不提供任何 反馈在预期的操作失败的情况下，使 API的不可预知。例如，按方法发送一个 指定按键事件模拟器。如果按键事件 调度失败，则用户将无视该故障。这种缺乏立即显式反馈

为了解决这些问题，我们已经建立了设备管理 基于Android ADB工具框架的组成部分[[17]](#endnote-18)[3]。

它管理的模拟器，并配合安装的过程 应用程序，执行UI自动化，并收集统计信息如应用程序日志和网络流量日志，而被宽容对仿真器的古怪行为

据我们所知，至今没有其他的出版工作 详细说明设备管理技术。在文献[[18]](#endnote-19)[37]，Rastogi医师等人。何况开发设备管理框架，但 包括具体细节。唯一可比公开工作就是MonkeyRunner[[19]](#endnote-20)[11]。相比MonkeyRunner，我们的系统中有许多优点，如表1所示

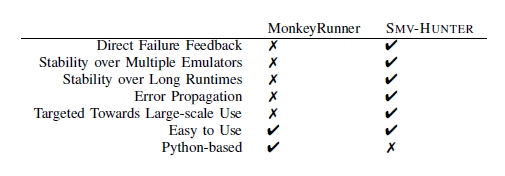


表1 与MonkeyRunner的对比

## 3.3 系统概述

SMV - HUNTER的概述示于图1 。鉴于一大套的Android应用程序， SMV- HUNTER首先执行

静态分析上的每个应用程序，确定他们是有缺陷的，当发现潜在的漏洞，进一步分析痕迹的漏洞代码的调用返回一个入口点窗口。潜在的易受攻击的应用程序，然后分析生成智能输入文本框。接着，把这些潜在的脆弱的应用程序和输入窗口是考虑到设备管理组件，将安装应用程序并执行UI自动化的窗口。而在UI自动化运行时，设备管理部分捕获记录信息以便进行后续处理。作为UI自动化触发器从应用程序的HTTPS流量，这个流量通过代理服务器，试图建立SSL中间人攻击，并记录成绩和失败以及识别功能和服务器是连接。最后，输出汇总和处理将数据从设备管理和代理相结合组件，产生最终名单确认有缺陷的应用程序。

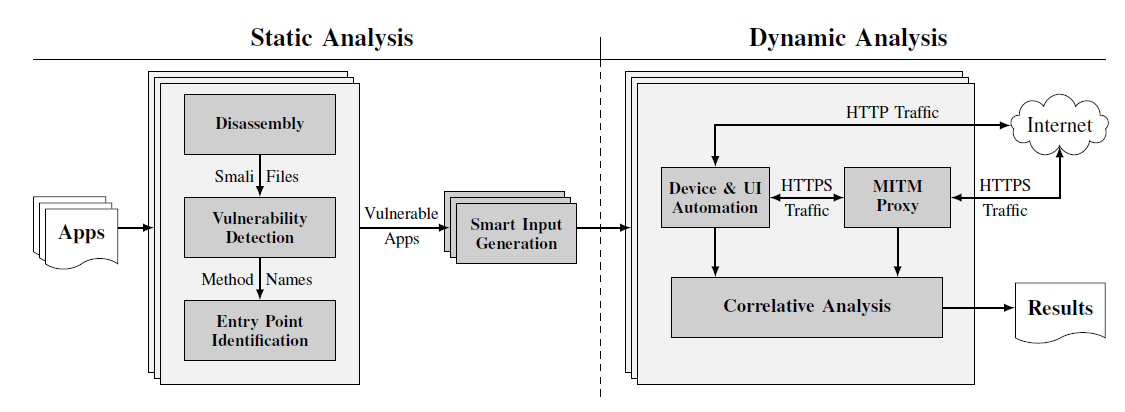


图1.系统概述

# 第四章 静态分析

在SMV-HUNTER，每个Android应用程序的第一阶段，反编译应用。这种静态分析组件的详细设计下面描述

## 4.1 反编译应用

Android应用程序是分布式的，包含各种包 编译的Android应用程序代码[[20]](#endnote-21)6。此代码可以被反编译，变成可读的格式Smali[[21]](#endnote-22)[23]。在SMV-HUNTER，使用apktool工具将字节码反编成到Smali[[22]](#endnote-23)[2]。我们选择Smali 反编译的原因有两个。首先，Smali 反编译比Java快得多。第二， 反编译过程可以通过模糊化受到阻碍产生的代码反编译技术不了解，而反编译没有这个缺点

4.2 静态SMV检测

一旦反编译的应用程序，确定潜在SMV很简单：只需检查应用程序是否重写

X509TrustManager的HostnameVerifier接口或如果应用程序的接口没有重写这些接口没有使用 SSL或使用未经修改原生的SSL支持，因此，可以认为是安全的。应用程序，重写这些接口要冒很大的风险，因此经常会出现漏洞

我们已经确定实现X509TrustManager的HostnameVerifier接口容易出现漏洞， 通过手动分析随机的1000个应用程序，252个应用的接口是有漏洞的

我们已经确定了导致SMV的四种常见模式：

基于X509TrustManager:

1. 无操作：最常见的实现X509TrustManager接口的方式是无操作，默认证书都是有效的，因此没有检查证书
2. 信任自签名证书：在这种模式中，如果证书链中只有一个证书，就实施检查，在这种情况下，使用checkValidity检查到证书尚未过期，却不会验证证书的签名或者询问用户是否信任自签名的证书。如在Apache HttpClient库的wiki部分[[23]](#endnote-24)[18],有一个EasySSLProtocalSocketFactory的例子，它遵循这种模式[[24]](#endnote-25)[32]，而且被应用到很多程序中（包括很多大型银行系统中）。
3. 只checkValidity:在这最后一中模式中，遍历证书链，用checkValidity检查每个证书是否过期，但是不做其它检查

基于HostNameVerifier:

1. 主机名验证 最常见的方法是使用Apache HttpClient库[[25]](#endnote-26)[22]提供的AllowAllHostnameVerifier类

## 4.3 易受攻击的入口点标识

一个典型的Android应用程序将有多个入口点（例如， Activities和Service[[26]](#endnote-27)[15]），使得动态分析详尽地执行它们变得很难。然而，许多（有时是大多数）这些入口点的代码路径不涉及HTTPS连接。因此，SMV-HUNTER 通过静态分析识别识别易受攻击的入口点。为了达到这个目的，对每个应用程序，构造一个方法调用图（MCG），并追踪易受攻击的入口点，并执行。因为我们只构造包含在编译的应用程序的方法的图表，我们使用修改后的MCG遍历程序算法1确定易受攻击的切入点

为了寻找易受攻击的入口点，我们从其中一个节点出发，遍历父节点，然后在父节点中遍历它的父节点，直到我们到达根节点。在一个典型的MCG的遍历程序，这将是遍历的结束。然而，这是经常的情况是，一 调用系统代码，所以当到达根节点的时候，跳转到构造函数的类，然后继续遍历。当开发人员实例化一个对象并传递给操作系统。直到遍历到一个构造函数没有被任何代码调用就停止。这个构造函数就是应用的入口点。

入口点要么是对应Activities要么对应Sevices。Services是非UI组件，大多与长时间运行的后台进程相关联，不太可能触发SSL连接，所以只需要在应用的manifest文件[[27]](#endnote-28)[6]中寻找要触发的Activities

|  |
| --- |
| Algorithm 1: traverse: Vulnerable Entry Point Identification  input : seed, the constructor of a vulnerable class node, the current node in the MCG  constructors, the set of traversed constructors  output : the set of vulnerable entry point methods  1 begin  2 if parents(node) != /0 then  3 for parent " parents(node) do  4 traverse (seed,parent,constructors)  5 else  6 for method " methods(class(node)) do  /\* method is never called. Continue traversing from its class’ constructor.  \*/  7 if method is the class’ constructor # method !" constructors then  8 constructors$constructors%method  9 traverse(seed,method,constructors)  /\* method is a constructor that is never called; report it as an entry  point. \*/  10 else if method " constructors then  11 output(method, seed)  Apps |

## 4.4 智能输入

Android应用程序通常在文本输入时进行验证，或者将文本转化成其他数据类型（如整型或浮点数），如果动态分析工具不能提供有效的输入，应用程序将不会执行所需的操作，甚至可能导致应用程序崩溃。以前的动态分析工作中，一直是提供随机的输入或手工制作[[28]](#endnote-29)[33]，以实现有效输入。SMV-HUNTER，充分利用现有信息中的应用程序“元数据 和代码，以确定有效的输入形式[[29]](#endnote-30)[37]。特别地SMV-HUNTER使用两个来源的信息：开发人员使用的输入类型注释和类型转换的代码。该输入类型注释主要是为了控制当用户选择输入字段出现的键盘，并限制了用户能够输入的字符

如图算法2，SMV-HUNTER通过尝试来智能生成每个文本字段的输入。一旦一个类型已被确定时，系统可以使用一个简单的表，以提供该类型的典型的输入。类型赋值过程首先在目标应用程序中的每个Activity中循环,每个Activity将调用函数setContentView布局。该系统从这个代码调用中提取并加载相关的布局XML文件（第4行）。系统从布局文件中提取的UI元素，特别是元素该类型的EditText，并遍历这些元素，提取元素的ID和输入类型注释（5 – 7行）。如果没有类型标注，系统使用和移动上到下的UI元素（8-10行）。

如果没有标注，SMV-HUNTER试图从Activity的反汇编代码的类型中提取信息。要做到这一点，首先找到引用的元素变量ID（第11行）。接下来，系统收集代码的所有部分访问这些变量（第12行），并为每个调用元素的getText函数（第13行），它跟踪的是使用重视通过任何类型转换操作（行14-15），和用途任何这种类型的强制转换类型的标签（第17行）。最后，将这些类型注释输入字符串（第21行）。SMV-HUNTER是第一个可以提供智能输入到UI元素的系统

图2显示了一些典型的示例代码从 SMV-HUNTER可以提取类型信息。特别地，有两个的EditText字段：其中一个预期的整数，但不提供输入类型标注，和一个预期的电话号码，并使用适当的输入类型标注。要提取系统中的这些类型 首先要在AndroidManifest.xml中找到活动名称（MainActivity）。下一步在MainActivity.smali中，寻找调用的函数setContentView，和被传递的参数ID（在这种情况0x7f03）。然后，该系统在R $ layout.xml找到与之相关的名称ID（activity\_main）。最后，SMV-HUNTER打开 相关文件（activity\_main.xml），并搜索EDITTEXT字段，并提取他们的变量，和任何输入类型标注。在命名phone\_field字段的情况下，现在有足够的信息与该字段关联：手机号类型。

其他的字段，命名INTEGER\_FIELD，不提供任何类型的标注，因此系统必须依赖于代码

分析，以确定其类型。 SMV-HUNTER首先查找的命名在文件R.id.smali找到其相关数字ID（0x7f080000）。接着，会在反编译代码中，特别是在MainActivity.smali中，为了得到与ID关联的变量名。在MainActivity.smali的1-3行，SMV-HUNTER通过调用函数findViewById，它返回一个对象，它关联一个整数。在后面的代码，系统然后使用提供的数据流分析 Androguard[[30]](#endnote-31)[28]寻找到这名被访问的地方（第5行），然后搜索的getText方法调用（6号线），并跟踪结果（在寄存器V3），以调用parseInt函数（第10行）。然后，系统可以与相关联的整数类型名称INTEGER\_FIELD。

|  |
| --- |
| Algorithm 2: Smart Input Generation  input : app, the app to be analyzed  output : input text for text boxes  1 begin  2 map$ /0  3 for act " getActivities(app) do  4 layout $getLayoutForActivity(act)  // loop over EditText elements in the layout  5 for elem " getUIElemsByType(layout,"EditText") do  6 elem id $getUIElemID(elem.name) // translate the element’s name to an id  7 input type$getUIElemInputType(elem id) // get the input type if there is one  8 if input type found then  9 map$map%{elem id &' input type}  10 continue to the next element  // if there is no type annotation,  // trace the access of the element through the activity’s code  11 instance var$getInstanceVariableForID(act, elem id)  12 reads$getAccessLocations(instance var)  13 for {read " reads | "getText" " read} do // when the getText function is called  // search for and record type casts  14 text var$ variable containing the result of the getText call  15 type cast $getTypeCastOperation(text var)  16 if type cast found then  17 map$map%{elem id &' type cast}  18 continue to the next element  19 else  20 continue to the next read  21 return translateTypeToInput (map) |

# 第五章 动态分析

在SMV-HUNTER系统的第二阶段，设备管理组件组件在模拟器上运行每个应用程序，触发了UI自动化组件和收集日志信息。同时，代理监控所有HTTPS流量，并尝试发动中间人攻击，登录成功和失败。该组件的每次输出被分析组件收集和聚集

## 5.1 设备管理

设备管理组件是动态分析得核心。它负责管理模拟器，监控其状态，安装的应用程序，并运行用户界面的自动化。为了确保完整性和有效性，它必须有如下特点：

* 管理并联的多个仿真器，
* 可理解检测每个仿真器的内部状态（“在线”与“离线”），通过重新启动 “脱机”的模拟器，来纠正状态
* 处理模拟器死机等错误，
* 动态管理模拟器池
* 计划和分发应用程序测试，以及
* 收集andmanage日志数据，包括安装与卸载

为了解决这些问题，装置管理组件需要两个主要线程：仿真器管理线程，一个应用程序的线程调度

模拟器管理 模拟器管理线程管理两个模拟器池（每个模拟器有自己的线程）：运行池和空闲池

当启动系统时，它注册一个DeviceChange-侦听器回调ADB。当模拟器启动或动态地添加到系统中，它进入“在线”状态，管理线将该模拟器程增加到运行池和空闲池中。如果仿真器进入“离线”状态或崩溃，管理线程移除两组线程，停止仿真器，并启动新的模拟器。因此，运行库包含的所有模拟器是在处在“在线”状态。空闲池中包含的模拟器是随时可以使用;这些模拟器是“在线”，而不是目前正在测试一个应用程序。当调度请求一个模拟器，它是从空闲池中取出，返回到调度器。当调度程序完成工作，它返回仿真器仿真器管理线程，这将其添加回空闲池。

应用程序调度 应用程序线程调度管理一个将要进行测试的应用列表，并通过算法3详细的处理该列表，如图所示的算法，遍历每个应用进行测试（第2行），首先得到设备管理线程处得到一个模拟器，然后安装应用，运行UI组件，通过静态分析易受攻击的Activity(4-7行)。接着将卸载应用，并将模拟器返回给设备管理线程（8,9行）。这个算法被简化，然而，调度器监视运行池的大小，并为每个模拟器创建一个线程，以使模拟器并行执行。此外，调度器还能处理模拟器报告的任何错误，如果安装失败，重新安装一次，如果依然失败，则放弃该应用。如果在UI自动化的过程中发生故障，将移动到下一个入口点

|  |
| --- |
| Algorithm 3: schedule: Application Scheduling  input : apps, a list of apps to be tested  1 begin  2 for app " apps do  /\* get an emulator from the management thread; this is a blocking call \*/  3 emulator$getEmulator ()  4 install (emulator,app)  /\* for each vulnerable entry point from static analysis \*/  5 for activity " getEntryPoints(app) do  6 startActivity (emulator,activity)  7 automateUI (emulator,activity)  8 uninstall (emulator,app)  9 releaseEmulator (emulator) |

## 5.2 UI自动化

**UI自动化**是SMV-HUNTER的重要组成部分。它模拟用户与Android应用程序的交互，驱动应用程序以有可能导致产生漏洞的方式执行。系统分析了代码路径的起源于在静态标识，将其作为切入点分析。用户界面自动化组件示于图3。UI自动化组件有三个目标：理解它的显示方式，提供智能输入接口，并理解和管理应用程序的状态。

**了解接口** 自动化的第一步是将UI分解成基本组成元素。对于基本元素，系统提取诸如属性，边界，坐标，输入（例如文本），有了这些信息，系统构造相应的输入事件发送给应用程序。例如，如果用户界面是一个按钮button或者复选框checkbox，系统就会生成click事件并结合了准确的坐标，如果界面是文本框textbox，系统就会生成输入事件

为了识别窗口的元素，并提取它们的属性，

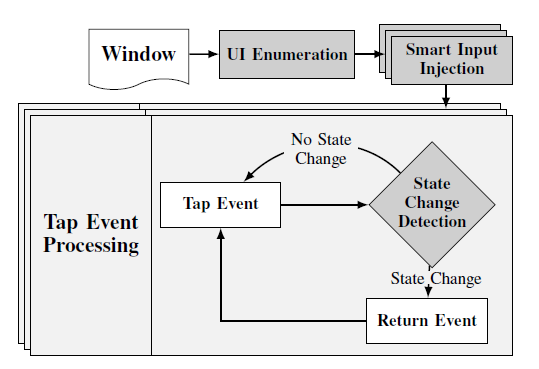


图3 UI自动化组件

该系统采用了Android ViewServer中，Android应用程序的工具框架的内部组件。该ViewServer提供了一组可命令用于查询的Android窗口管理，它处理UI元素的显示和输入事件的调度到相应的元件。具体来说，我们的系统会查询该ViewServer[[31]](#endnote-32)[39]检索整个视图层次，这包含的所有UI元素和它们的属性，如坐标和可编辑性。

生成输入事件 一旦系统识别用户界面中的元素和输入类型，他必须生成输入事件发生到应用程序。对于文本字段，使用智能输入组件生成文本。一旦生成输入事件，系统通过ADB使用命令行，该命令同时支持文本输入事件和在特定坐标的点击事件

总体而言，自动化的窗口的过程中有两个阶段。首先，系统通过迭代填充所有可编辑的文本字段，通过它们，产生点击事件，并点击它们，然后通过静态分析产生的智能输入。接着，

系统遍历所有可点击元素，在适当的坐标生成点击事件。每次点击的间隙，用户界面自动化组件等待来自设备管理状态的响应（下面描述）。当它接收到一个响应，系统就进行到下一个元素

应用状态管理 利用Android的ViewServer组件提供的API来获取应用程序的当前状态，并检测状态的变化。当应用状态发生变化时，ViewServer提供的WindowChange和FocusChange事件被触发，通过为这些事件注册,当状态发生变化时，系统就会发出通知。直到每个事件被被处理完，UI自动化停止运行，使得处理器有时间反应应用状态的变化，当应用程序转换到一个新的 状态，在UI自动化组件生成一个“后退按钮”事件，这将导致Android的从弹出当前窗口它的堆栈，返回到目标窗口，

Android系统允许“不可撤销”对话框和类似UI组件，具有暂时禁用返回键的效果。在这些情况下，用户界面自动化组件所生成回退按钮的事件没有任何效果，因此，系统检查 为恢复正常操作前的状态变化。如果 状态保持不变，会产生额外的点击事件; 这些应该点击任何“确定”或“取消”按钮，关闭该对话框并返回到目标窗口。如果三次这样的点击事件，状态依然无法改变，系统将终止应用程序。放弃当前的activity,移动到下一个入口点

## 5.3 MITM代理

要执行一个SSL中间人攻击，必须拦截模拟器和Internet之间所有的HTTPS流量。当运行很多的模拟器，由于大量的连接，会是一个标准的代理软件过载。一个广泛使用的MITM攻击代理是马洛里[[32]](#endnote-33)[19],但是在实验中，不能处理太多的并发连接，当运行时间过长，有崩溃的趋势。由于这个问题，我们设计的系统使用Burp Suite proxy[[33]](#endnote-34)[21]，它生成一个自签名的证书用于攻击，Burp允许用户修改和记录流量的脚本。我们使用此功能向数据库记录成功的HTTPS连接（即成功攻击），Burp Suite proxy比马洛里更加稳定，它能处理一个接一个的连接，如果有多个模拟器，这导致许多应用程序级别的连接超时。为了减轻这一点，因此使用iptables[[34]](#endnote-35)[25]，以绕过为代理所有非HTTPS通信，如图1 ，减少了负载对代理，并允许我们的系统扩展到所需的水平。

## 5.4 相关性分析

MITM代理可以通过成功的攻击应用程序检查漏洞，但是不能反向映射漏洞，它只是看到网络流量。因此，用一个相关性分析组件映射成功被攻击的漏洞，在使用生成的日志的应用程序由设备管理组件和MITM代理。因为在并行测试多个应用程序，可能有多个成功的攻击在大约相同的时间。此外，网络延迟会造成时间戳略有不同设备管理组件和中间人之间 代理。因此，我们不能用简单的时间戳匹配攻击的应用程序。因为模拟器都在运行同一台机器共享相同的网络接口，他们这样做不能得到唯一的MAC地址或IP地址，所以不能使用的地址来匹配攻击的应用程序。然而，因为MITM代理日志模拟域名，可以使用DNS查找来加强模糊时间匹配

相关分析组件的工作原理如下：首先，从设备管理组件安装时间戳用于每个应用程序映射到的时间。第二，网络从设备管理日志部件中搜索的DNS查询，这是用来时间段映射到Internet网络域。最后，MITM代理日志被用来产生时间第二映射块到Internet域。当从中间人的时间块代理从设备管理重叠的时间块与同域组件，相关的应用程序被标记为有漏洞

# 第六章 评价

在本章中展示了实验结果，通过两台配备了Inter Xeon E5620 8核处理器和24G内存的DELL T7500机器的静态分析，为了简化任务调度程序，避免MITM代理，在其中一台机器中使用了动态分析。因此16核用于静态分析，8个模拟器用于动态分析

在第五章A部分，讨论了数据集。在第五章B和C部分，展示了静态和动态分析的结果。在第五章D部分，分析了易受攻击的应用程序。最后在第五章E部分，分析这些程序的更新状态

## 6.1 数据集的特点

在试验中使用了两个互相排斥的数据集。第一个数据集（用DS1表示）收集的是谷歌应用商店中与金融有关的应用。因此，该数据集是偏向金融方面的应用，所以这些应用程序的漏洞，通常会有破坏性的后果。此数据集包含3165个应用程序。第二个数据集（用DS2表示）收集无任何特点的应用程序。此数据集包含20316个应用程序。该数据集的分布

示于图4（a）所示。 DS1的大多是与金融有关的应用程序

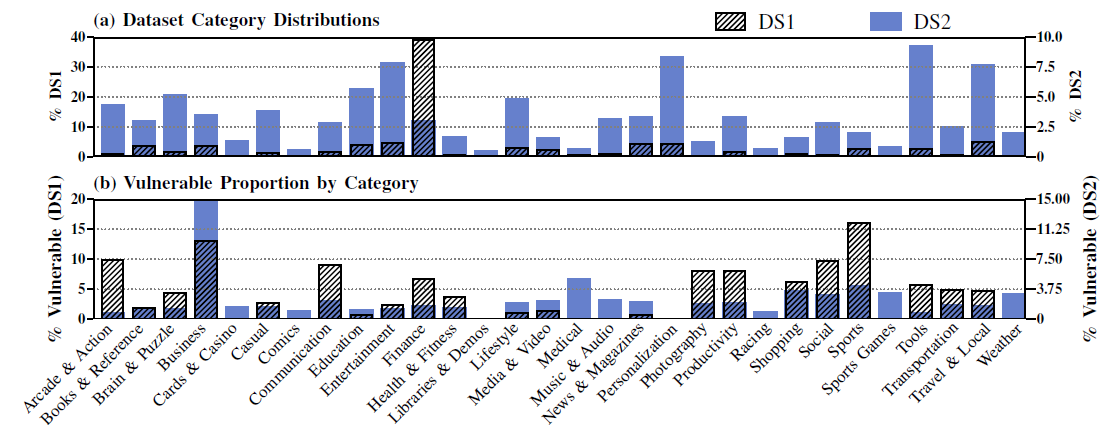


图4.应用种类分布

往往比一般简单的应用程序接口。例如，DS2可能包含具有高度自定义游戏应用程序

复杂的界面。 DS1的分离提供基线绩效指标

## 6.2 静态分析

进行静态分析的两个数据集，识别有漏洞的应用，并确定导致这些漏洞的切入点。静态分析的结果在表2中给出

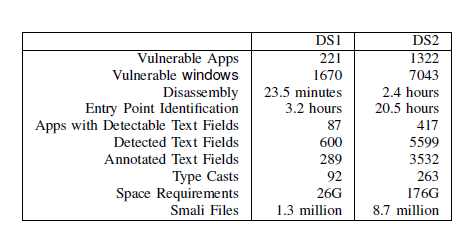


表2 静态分析结果

**时间要求** 平均而言，静态分析花4s分析一个应用程序，它可以分成3个部分：

- 反汇编：正如第四章所讨论的，使用apktool[[35]](#endnote-36)[2]应用程序的反编译成中间格式称为Smali[[36]](#endnote-37)[23]，这非常类似JVM字节码指令，但更具有可读性和结构。这个过程每个应用程序平均需要0.42秒。这比反编译成Java更快（使用DED反编译器[[37]](#endnote-38)[35]），这取每个应用程序276秒的平均值，并很多多较不可靠。

- 漏洞的入口点标识：静态分析过程检测漏洞的入口点花了24小时来分析两个数据集（平均而言每个应用程序3.63秒）。

- 智能输入生成：生成智能输入文本字段，SMV-HUNTER大约需要一个半小时来处理DS1和DS2（每个应用程序1.2秒）。

**空间要求** 存放下载的应用程序和它们的反汇编和分析结果，需要大量的磁盘空间，如在表2中所详述。从这些结果可以推断，在分析了超过100万的应用程序谷歌应用市场[38]将需要超过8 TB的存储容量含有超过420万个文件。扩展到这个水平，建议使用分布式计算框架，如 Hadoop的[7]，这将提供大型计算机集群分析数据，增加了任务的可行性。

**260395个界面的统计** 静态分析确定8,713 作为潜在的易受攻击的入口点。其中，607个应用程序默认的窗口（即中所示的第一窗口当应用程序启动）。智能输入生成统计信息 在表2中示出

## 6.3 动态分析

对于动态分析过程中，我们使用8个模拟器运行Android OS 4.1测试并行应用程序。这表明，该框架可以一次管理和自动化多个仿真器。总体而言，这一过程花了18.81小时来分析两个数据集（2.91小时，DS1和15.90小时DS2）。在运行时，我们录得12次模拟器崩溃，并发现每个仿真器崩溃（或去一个“离线”状态）至少一次，显示长时间运行时Android模拟器的不稳定性

通常情况下，Activity依赖先前窗口的输入数据。例如，一个登录窗口后的任何窗口依赖于登录结果。由于直接启动Activity导致信息的丢失，这个Activity有两种状态：要么不能正常重定向的跳过窗口，要么就崩溃。总的来说，动态分析得8712个切入点，其中1705个在启动时就崩溃

在动态测试时收集详细的统计数据，其中显示给定：成功攻击的应用程序，

安装失败的应用程序，应用程序的所有活动失败 发动，平均时间来处理每个窗口，以及

窗户启动失败，每个数据集和元数应用程序类别。有趣的是，我们可以看到在企业的应用

类别有显著更可能是脆弱的（这也很明显在图4（b））。另一个值得注意的数据是 在金融类的应用程序是显著更有可能有失败的启动窗口。这可能是由于应用程序需要登录凭据：如果你启动一个窗口，通常自带的登录界面后，这些应用程序经常崩溃

## 6.4.有漏洞的应用程序

显示了漏洞的应用程序的分布类别。这也示于图4（b），其中每个类的长代表一个应用程序有漏洞的可能性。显然，该业务类是最脆弱的：从图4（b）中，我们看到在商业类应用程序中大约有12-15％ 是脆弱的在这两个数据集。需要注意的是少数中的DS1非金融应用程序引入了大量的噪音的相对似然的计算。

## 6.5 总结有漏洞的应用程序

该项目进行了一年时间，现在重新审视脆弱的应用程序，并检查他们是否有被修补。因此，最近试图重新下载 从两个数据集和所有726确认脆弱的应用程序分析更新后的版本。显示的可用性的更新版本，以及最近的分析结果。在“不可用”列显示有多少应用程序能不重新下载。在“Still Vulnerable”列显示脆弱的应用程序的比例是被重新下载并发现仍然是脆弱的。

总体而言，应用程序的14.6％是重新下载不可用，和76.17％，仍然是脆弱的，这表明SMV仍然非常普遍

# 第七章 相关工作

**数据分析** 专注于使用镜头分析来检测恶意软件，隐私泄露，和山寨应用程序是一个相当大的工作。其中包括[[38]](#endnote-39)[26],[[39]](#endnote-40)[31] ,[[40]](#endnote-41)[34] ,[[41]](#endnote-42)[35] ,[[42]](#endnote-43)[42] ,[[43]](#endnote-44)[43].Octeau等人[[44]](#endnote-45)[35]开发了一种Dalvik的反编译工具，ded,使用自动化的组合测试和人工检查来分析应用程序。他们研究了广告库和敏感信息的泄露。虽然ded是准确的，但是太慢了不适合大数据分析，在,他们从1100免费应用程序中分析了2100万行代码，花了近500小时

Zhou等研究人员。[[45]](#endnote-46)[43]进行了系统的，大规模的研究 使用静态分析的Android恶意软件。他们DroidRanger 系统中使用的各种试探法来进行更多有效的静态分析。他们分析了约20万的应用程序 从不同的Android市场，并报告211恶意应用程序，包括两个“0day”攻击

FAHL等研究人员[[46]](#endnote-47)[30]利用静态分析13,500个受欢迎 Android应用程序，产生1,074应用程序可能会遭受 SMV。然后，他们进行人工分析的样本 100潜在的易受攻击的应用程序，产生41易受攻击的应用程序

**动态组件** 在文献[[47]](#endnote-48)[33]，Hu和Neamtiu使用自动随机 （使Monkey[17]），测试生成测试用例和 检测程序错误。最近，SmartDroid系统[41]发现 一些揭示基于UI的触发条件的问题 在Android应用程序，但它需要的Android定制操作系统。具体来说，它是基于Android的定制

开源项目[[48]](#endnote-49)[5]，而这还不包括专有 中所包含的商业版本的库。中 在726 APPS我们确定为明确易受系统 SSL的MITM攻击，370（51％）所需的专有谷歌 映射库。因此，这些应用程序将无法安装在该SmartDroid系统

# 第八章 讨论和未来的工作

在本节中，我们将讨论我们的方法的局限性并提出今后的工作中加以解决。

静态分析 我们的静态分析组件认为任何 应用程序，覆盖X509TrustManager的HostnameVerifier或接口可能存在漏洞（见第四章）。这样的保守分析不会导致任何漏报率。但是， 它可能会引入误报的情况下一个应用程序的 开发人员重写这些接口，但不存在安全漏洞

动态分析 由静态分析引入的任何误报分析组件是由动态分析排除组成部分，我们将只申报一个应用程序容易受到攻击如果我们真正利用该漏洞。然而，目前的实现UI自动化组件将推出，由于以下限制漏报

* 多窗口输入：在某些情况下，一个应用程序的提示建立前几页输入的用户安全连接。因为我们只遍历每个入口点的直接子窗口，我们的系统将无法触发在这些情况下的连接。解决的办法是提供推荐的输入，在一个深度优先搜索情况下，直到应用程序建立SSL连接。这种方法可能很慢或导致无限循环，所以它需要某种启发式的，以确定何时移动到一个新的切入点。例如，它可有可能使用静态分析来构造码路径从入口点的漏洞，并检测执行从偏离这条道路。
* 高级UI操作：有些应用程序需要更多的复杂的UI交互，如刷卡或长时间触摸事件。当需要这样形式的输入的检测，和如何生成输入事件将是未来的工作。此外，该系统被限制为简单的UI元素具有一个可点击的区域，不改变形状 或位置的基础上输入的，但它可以被扩展到支持复杂的UI元素如Spinner[[49]](#endnote-50)[16]Pickers[[50]](#endnote-51)[13]或的customUI元素的requiremore 复杂的输入。扩展SMV-HUNTER到支持 这些是今后的研究方向之一
* WebViews：有些应用程序使用的WebView UI元素，这是本质上的嵌入式浏览器组件，其中ViewServer不能检查。未来工作任务就是检测到这些元素并生生适当的UI事件，以测试它们。这样的解决方案可以充分利用现有的工作，在浏览自动化[[51]](#endnote-52)[24]。

# 第九章 总 结

在本文中，我们已经开发了SMV-HUNTER系统，它结合了静态和动态分析技术， 自动为Android 应用程序执行大规模SMV检测。首先采用静态分析组件检测可能的漏洞，确定用户界面的目标来触发 ，并生成智能输入引导动态分析组件，它执行自动的UI 尝试MITM攻击的探索。实验研究结果表明，系统是可行的和有效的

# 致 谢

研究资料是由The Air Force Office of Scientific Research提供编号FA-9550-12-1-0077文档，非常感谢匿名审稿人的意见

1. # 参考文献

   [↑](#endnote-ref-2)
2. [1] Activities. http://developer.android.com/guide/components/activities.html.

   [2] android-apktool. http://code.google.com/p/android-apktool/.

   [3] Android debug bridge. https://developer.android.com/tools/help/adb.html.

   [4] Android emulator. https://developer.android.com/tools/help/emulator.html.

   [5] Android open source project. http://source.android.com/.

   [6] Glossary. http://developer.android.com/guide/appendix/glossary.html.

   [7] Hadoop. https://hadoop.apache.org/.

   [8] Issue 10255: Adb hangs intermittently. http://code.google.com/p/

   android/issues/detail?id=10255.

   [9] Issue 1946: javax.net.ssl.sslexception: Not trusted server certificate.

   http://code.google.com/p/android/issues/detail?id=1946.

   [10] Issue 38315: Devices are going in offline state in “adb devices” after

   random time. http://code.google.com/p/android/issues/detail?id=38315.

   [11] Monkeydevice. http://developer.android.com/tools/help/MonkeyDevice.html.

   [12] Monkeyrunner. https://developer.android.com/tools/help/MonkeyRunner.html.

   [13] Pickers. https://developer.android.com/design/building-blocks/pickers.html.

   [14] Robotium. https://code.google.com/p/robotium [↑](#endnote-ref-3)
3. [15] Services. https://developer.android.com/guide/components/services.html. [↑](#endnote-ref-4)
4. [16] Spinners. https://developer.android.com/design/building-blocks/spinners.html.

   [17] Ui/application exerciser monkey. https://developer.android.com/tools/

   help/monkey.html.

   [18] Httpclient - httpclient ssl guide. http://hc.apache.org/httpclient-3.x/

   sslguide.html, 2008.

   [19] Mallory: Transparent tcp and udp proxy. http://intrepidusgroup.com/

   insight/mallory/, 2010 [↑](#endnote-ref-5)
5. [20] android: Single activity, multiple views. http://stackoverflow.com/

   questions/10862052/android-single-activity-multiple-views, 2012.

   [21] Burp suite. http://www.portswigger.net/burp/, 2012.

   [22] Class allowallhostnameverifier. http://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/conn/ssl/AllowAllHostnameVerifier.html, 2012.

   [23] smali. http://code.google.com/p/smali/, 2012.

   [24] ARTZI, S., DOLBY, J., JENSEN, S. H., MØLLER, A., AND TIP, F. A

   framework for automated testing of javascript web applications. In Proceedings

   of the 33rd International Conference on Software Engineering(2011).

   [25] AYUSO, P. N. netfilter/iptables project homepage - the netfilter.org“iptables” project. http://www.netfilter.org/projects/iptables/.

   [26] CHRISTODORESCU, M., AND JHA, S. Static analysis of executables to detect malicious patterns. In Proceedings of the 12th conference on USENIX Security Symposium - Volume 12 (2003).

   [27] CLARK, J., AND VAN OORSCHOT, P. C. Sok: Ssl and https: Revisiting past challenges and evaluating certificate trust model enhancements. 2013 IEEE Symposium on Security and Privacy 0 (2013), 511–525 [↑](#endnote-ref-6)
6. [28] DESNOS, A. androguard. https://code.google.com/p/androguard/.

   [29] ENCK, W., GILBERT, P., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL,P., AND SHETH, A. N. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In Proceedings of the 9th USENIX conference on Operating systems design and implementation (2010).

   [30] FAHL, S., HARBACH, M., MUDERS, T., SMITH, M., BAUMG ¨ARTNER, L., AND FREISLEBEN, B. Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security. In 19th ACM Conference on Computer and Communications Security (2012).

   [31] FELT, A. P., FINIFTER, M., CHIN, E., HANNA, S., AND WAGNER, D.A survey of mobile malware in the wild. Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices- SPSM ’11 (2011).

   [32] GEORGIEV, M., IYENGAR, S., JANA, S., ANUBHAI, R., BONEH, D.,AND SHMATIKOV, V. The Most Dangerous Code in the World:Validating SSL Certificates in Non-Browser Software. In 19th ACM Conference on Computer and CommunicationsSecurity(2012).

   [33] HU, C., AND NEAMTIU, I. Automating GUI testing for Android pplications. In Proceedings of the 6th International Workshop on utomation of Software Test(2011).

   [34] KOLTER, J. Z., AND MALOOF, M. A. Learning to detect and classify alicious executables in the wild. J. Mach. Learn. Res. 7 (2006).

   [35] OCTEAU, D., ENCK, W., AND MCDANIEL, P. The ded Decompiler.Tech. Rep. NAS-TR-0140-2010, Network and Security Research Center,epartment of Computer Science and Engineering, Pennsylvania State niversity, 2010.

   [36] PORTOKALIDIS, G.,HOMBURG, P.,ANAGNOSTAKIS, K., AND BOS, H.Paranoid Android: Versatile Protection For Smartphones. In Annual omputer Security pplications Conference (2010).

   [37] RASTOGI, V., CHEN, Y., AND ENCK, W. AppsPlayground: Automatic ecurity Analysis of Smartphone Applications. In Third ACM Conference n Data and Application Security and Privacy (2013).

   [38] TAM, D. Google forecasts 70 million android tablet activations by year’s nd. http://news.cnet.com/8301-1023 3-57595262-93/google-forecasts- 0-million-android-tablet-activations-by-years-end/, 2013.

   [39] TEITELBAUM, D. Posts tagged ‘viewserver’. http://blog.apkudo.com/

   tag/viewserver/, 2012.

   [40] YAN, L. K., AND YIN, H. Droidscope: seamlessly reconstructing the s and dalvik semantic views for dynamic android malware analysis. I Proceedings of the 21st USENIX conference on Security symposium(2012).

   [41] ZHENG, C., ZHU, S., DAI, S., GU, G., GONG, X., HAN, X., AND ZOU,W. Smartdroid: an automatic system for revealing ui-based trigger onditions in android applications. In Proceedings of the second ACM orkshop on Security and privacy in smartphones and mobile devices(2012).

   [42] ZHOU, W., ZHOU, Y., JIANG, X., AND NING, P. Detecting repackaged martphone applications in third-party android marketplaces. In roceedings of the second ACM conference on Data and Application ecurity and Privacy (2012).

   [43] ZHOU, Y., WANG, Z., ZHOU, W., AND JIANG, X. Hey, You, Get Off My Market: Detecting Malicious Apps in Official and Alternative ndroid Markets. In 19th Network and Distributed System Security ymposium (2012). [↑](#endnote-ref-7)
7. [↑](#endnote-ref-8)
8. [↑](#endnote-ref-9)
9. [↑](#endnote-ref-10)
10. [↑](#endnote-ref-11)
11. [↑](#endnote-ref-12)
12. [↑](#endnote-ref-13)
13. [↑](#endnote-ref-14)
14. [↑](#endnote-ref-15)
15. [↑](#endnote-ref-16)
16. [↑](#endnote-ref-17)
17. [↑](#endnote-ref-18)
18. [↑](#endnote-ref-19)
19. [↑](#endnote-ref-20)
20. [↑](#endnote-ref-21)
21. [↑](#endnote-ref-22)
22. [↑](#endnote-ref-23)
23. [↑](#endnote-ref-24)
24. [↑](#endnote-ref-25)
25. [↑](#endnote-ref-26)
26. [↑](#endnote-ref-27)
27. [↑](#endnote-ref-28)
28. [↑](#endnote-ref-29)
29. [↑](#endnote-ref-30)
30. [↑](#endnote-ref-31)
31. [↑](#endnote-ref-32)
32. [↑](#endnote-ref-33)
33. [↑](#endnote-ref-34)
34. [↑](#endnote-ref-35)
35. [↑](#endnote-ref-36)
36. [↑](#endnote-ref-37)
37. [↑](#endnote-ref-38)
38. [↑](#endnote-ref-39)
39. [↑](#endnote-ref-40)
40. [↑](#endnote-ref-41)
41. [↑](#endnote-ref-42)
42. [↑](#endnote-ref-43)
43. [↑](#endnote-ref-44)
44. [↑](#endnote-ref-45)
45. [↑](#endnote-ref-46)
46. [↑](#endnote-ref-47)
47. [↑](#endnote-ref-48)
48. [↑](#endnote-ref-49)
49. [↑](#endnote-ref-50)
50. [↑](#endnote-ref-51)
51. [↑](#endnote-ref-52)