Open Doors for Bob and Mallory: Open Port Usage in

Android Apps and Security Implications

Abstract—Open ports are typically used by server softwareto serve remote clients, and the usage historically leads toremote exploitation due to insufficient protection. Smartphoneoperating systems inherit the open port support, but since theyare significantly different from traditional server machines inperformance and availability guarantees, little is known abouthow smartphone applications use open ports and what thesecurity implications are. In this paper, we perform the firstsystematic study of open port usage on mobile platform andtheir security implications. To achieve this goal, we designand implement OPAnalyzer, a static analysis tool which caneffectively identify and characterize vulnerable open port usagein Android applications.Using OPAnalyzer, we perform extensive usage and vul-nerability analysis on a dataset with over 100K Androidapplications. OPAnalyzer successfully classifies 99% of themobile usage of open ports into 5 distinct families, and fromthe output, we are able to identify several mobile-specificusage scenarios such as data sharing in physical proximity.In our subsequent vulnerability analysis, we find that nearlyhalf of the usage is unprotected and can be directly exploitedremotely. From the identified vulnerable usage, we discover410 vulnerable applications with 956 potential exploits in total.We manually confirmed the vulnerabilities for 57 applications,including popular ones with 10 to 50 million downloads onthe official market, and also an app that is pre-installed onsome device models. These vulnerabilities can be exploited tocause highly-severe damage such as remotely stealing contacts,photos, and even security credentials, and also performingsensitive actions such as malware installation and maliciouscode execution. We have reported these vulnerabilities andalready got acknowledged by the application developers forsome of them. We also propose countermeasures and improvedpractices for each usage scenario.

An open port (or a listening port) is a communicationendpoint for accepting incoming connections in computernetworking model, typically used by server applications tohandle requests from remote clients. However, these portscan also be connected by malicious clients if not carefullyprotected, exposing potential vulnerability in the server soft-ware to remote exploitation. Such inherent weakness hasalways accompanied the usage of open ports throughout thehistory of network services, opening doors for large num-bers of severe Internet attacks such as TCP SYN floodingattacks [27], the Conficker worm [14], and more recentlythe Heartbleed bug [9]. To mitigate the problem in thesetraditional usage scenarios, firewalls and user authenticationmechanisms are usually adopted.In the recent evolution to the mobile era, smartphoneoperating systems inherit the support for open port. But forsmartphone applications (apps), traditional open port usecases such as hosting network services no longer apply.One major reason is that compared to stationary servermachines with wired network connectivity, the mobilitynature of smartphones makes it difficult to maintain a stableIP address. Moreover, the IPs assigned to mobile devicesare often behind a NAT (network address translation) pre-venting incoming network connections. Also, continuouslyreceiving network traffic can easily drain the battery of amobile device, leading to a form of denial-of-service (DoS)attack [49]. Due to these inherent differences, our currentunderstanding about smartphone usage of open ports arerather limited.With the immense popularity of smartphones, any po-tential smartphone open port usage may directly expose endusers to severe damage. Several such examples have alreadybeen reported recently, called “Wormhole” apps [12], whereopen ports in popular Android apps allow an attacker toremotely collect location data, insert contacts, and eveninstall app without authorization, and over 100M devicesare affected. While these exploits are alarming, it is stillunclear whether these vulnerabilities are exposed by popularuse cases of open ports in the smartphone ecosystem, or justby poor implementation practices.In this work, we perform the first systematic study ofopen port usage and the security implications on mobileplatform. To achieve this goal, we design and implementa tool called OPAnalyzer, which can effectively identifyand characterize vulnerable open port usage in Androidapps. To use OPAnalyzer, we first formalize open port appdesign pattern in the language of program analysis, whichin high level specifies what sensitive functions are triggeredfrom open ports, and how they are triggered. With thesedefinitions, OPAnalyzer first uses static taint analysis totrack the information flow from the remote input entry point,and identifies the sensitive functionalities that can potentiallybe triggered. After this step, a set of usage paths of theopen port are generated, which will lead to remote exploitsif not well protected. To help prioritize human inspection,OPAnalyzer examines the security checks along the usagepaths guarding the sensitive functionality. If the executionof a given path is found to have no constraints or containsonly weak checks, a potential remote exploit is directlyrevealed ( § 4.4). OPAnalyzer also dynamically tests whetherthe vulnerable port is open by default, and labels the weakpaths as highly insecure if the corresponding port opensautomatically at app launching time. For high precision,our design leverages the Amandroid approach [46], whichsupports flow-, context-sensitive data-flow analysis.To ensure high effectiveness, we overcome several en-gineering challenges in the tool implementation. First, ouranalysis needs accurate identification of the permission-protected APIs, but the API to permission mappings pro-vided by the most recent work, PScout [21], are incompletefor our purpose since it does not consider the prerequisitesof the API usage. To address this limitation, we improvePScout to automatically fix some common missing cases( § 4.3). Second, we find that Java reflection is commonlyused to handle remote input from open ports, which isnot resolved by many static taint analysis tools such asAmandroid. To ensure the call graph completeness, weadd an extra analysis to locate the target class or method,which successfully resolves over 86% Java reflection usecases in our app dataset ( § 4.4). Third, we find that manyapps actually implement open port usage in native code,which cannot be captured by Java-layer static analysis alone.Therefore, our tool also includes native code support basedon binary analysis techniques, which is commonly excludedin nearly all existing static analysis tools on Android appsdue to high engineering efforts [20], [33], [34], [46]( § 4.2).Using OPAnalyzer, we perform an open port usageanalysis on 24K popular Android apps from Google Play,and successfully classify 99% of the usage paths into 5categories: data sharing, proxy, remote execution, VoIP call,and PhoneGap ( § 5.2). We also find that significantly dif-ferent from traditional usage, ports in some categories weremostly intended only for clients in physical proximity of thesmartphone, or even on the same device.Among these open port usage families, many are foundto directly enable a number of serious remote exploits ifnot well protected. More specifically, we use OPAnalyzerto examine the security checks along the identified usagepaths, and find that they generally lack sufficient protection:for the most popular usage, data sharing, over half of thepaths can be easily triggered by any remote attacker, and insome usage categories such as proxy, over 80% of the pathsare not protected. From OPAnalyzer output, we uncover410 vulnerable applications with 956 potential exploits intotal, and manually confirm 57 vulnerable apps that havenot been previously reported, including popular ones onthe market and even a pre-installed app on some devicemodels. These newly-discovered exploits can lead to a largenumber of severe security and privacy breaches. for exampleremotely stealing sensitive data such contacts, photos, andeven security credentials and performing malicious actionssuch as executing arbitrary code and installing malware re-motely ( § 6). To get an initial estimate on the impact of thesevulnerabilities in the wild, we performed a port scanning inour campus network, and immediately found a number ofmobile devices in 2 minutes which were potentially usingthese vulnerable apps. we have reported these vulnerabilitiesto the relevant parties through vulnerability tracking systemsincluding CVE [5] and CERT [16], and some of them havebeen acknowledged (e.g., CVE-2016-5227, VR-176). Weencourage readers to view several short attack video demosat https://sites.google.com/site/openportsec/ [11].Leveraging the insights from these analysis, we furthercategorize the vulnerable apps based on their intentions ofopen port, and discuss defense strategies depending on theunique characteristics in each category ( § 7). Specifically,for the physical proximity usage, which does not have anyeffective and usable protection yet, we propose a transparentsocket-level solution that allows users to conveniently verifya connection from a device nearby and can be easily adoptedby app developers.We summarize the key contributions of this paper:• We formalize open port app design pattern, anddevelop OPAnalyzer to systematically characterize open portusage in Android apps and detect exposed vulnerability. Toensure high accuracy, we tackle several challenges, e.g.,improving the API to permission mapping completeness,resolving Java reflection, and enabling native code analysis.• Using our tool, we perform the first systematicstudy of open port usage and their security implicationson mobile platform. We are able to classify 99% of theidentified usage into 5 distinct usage families, and discoversome mobile-specific scenarios. We find that nearly half ofthese usage paths have no protection implemented, whichcan directly be triggered by remote attackers to leak sensitiveinformation and perform high-privileged actions.• We perform an in-depth analysis on the vulnerableopen port usage, and construct real exploits to validate thethreats. From the results, we manually confirmed 57 newvulnerable apps containing popular ones on the market andalso a pre-installed app on some device models, which canbe used to remotely steal sensitive user data such as photos,security credentials, and perform malicious actions such asexecuting arbitrary code and installing malware. We alsosuggest countermeasures and improved practices to mitigatethese problems in each intended open port usage scenario.2. Background and Threat ModelIn this paper, we broadly define mobile apps with openTCP or UDP ports as open port apps. And two types ofopen port apps are covered by our study. (1) Mobile serviceapp provides useful functionality such as sharing files on thehandset by opening a file server to be connected by user’sdesktop. (2) Malicious open-port apps intentionally openports to carry out malicious activities such as receiving com-mands from remote attackers for data theft or device control.Our study does not focus on malware detection, since it’s2very hard to distinguish malicious and legitimate open portusage without having a comprehensive understanding ofthe designed functionality of each app. Instead, we focuson identifying problematic usage (including both maliciousand legitimate) that exposes vulnerabilities to attacker andaffects the well-being of the user.Threat model. The threat to an app with open portscomes from the attackers with the ability to reach theseports. In the design of popular smartphone operating systemssuch as Android, ports are reachable from both the samedevice, e.g., another app or a script on the web page, and an-other host in the same network with the victim device. Thus,compared to the majority of previously-reported smartphoneapp vulnerabilities that only consider the threat from on-device malware [20], [28], [30], [50], [51], open port appsadditionally face threats from network attackers, e.g., localnetwork attacks, and web attackers, e.g., malicious scripts,which is much more diverse and also of wider range. Morespecifically, in this paper we consider the following threeadversary types:(1) Malware on the same device. A maliciousapp, or malware, installed by the smartphoneuser can use netstat command or proc file/proc/<pid>/net/tcp to find the listening portson the same device and send exploitation traffic.(2) Local network attacker. For victims behind NATor using private WiFi networks, attackers sharing the samelocal network can use ARP scanning [4] to find reachablesmartphone IP addresses at first, and then launch targetedport scanning to discover vulnerable open ports.(3) Malicious scripts on the web. When a victim uservisits an attacker-controlled website using their mobile de-vice, malicious scripts running in the handset’s browser canexploit the vulnerable open ports on the device by sendingnetwork requests, which doesn’t require any permission.For each of these three threat models, we have preparedshort attack video demos on our website [11] to help readersmore concretely understand their practicality.Scope and assumptions. Our study focuses on TCPports, which are most commonly used. We did not studyUDP ports, but we argue that our methodology can be easilyadapted for it. Our tool is expected to handle obfuscatedAndroid apps as long as they can be disassembled. In thecurrent implementation, our tool only fails to analyze veryfew samples (0.6% of apps in our dataset); for them, eventhe disassembling process cannot succeed.

Bob和Mallory打开门：Android应用程序中的开放端口使用和安全隐患

摘要 - 开放端口通常由服务器软件用于为远程客户端提供服务，并且由于保护不足，历史上会导致远程使用。智能手机操作系统继承了开放端口支持，但由于它们与传统服务器机器的性能和可用性保证有很大不同，因此对智能手机应用程序如何使用开放端口以及安全隐患的了解甚少。在本文中，我们首先对移动平台上的开放端口使用情况及其安全影响进行了系统的研究。为了实现这一目标，我们设计并实现了OPAnalyzer，这是一种静态分析工具，可以有效识别和表征Android应用程序中易受攻击的开放端口使用情况。使用OPAnalyzer，我们对具有超过10万个Android应用程序的数据集进行广泛的使用和易用性分析。 OPAnalyzer成功地将99％的开放端口的移动使用分类为5个不同的系列，并且从输出结果中，我们能够识别一些移动特定的使用场景，例如物理接近的数据共享。在我们随后的漏洞分析中，我们发现几乎一半使用情况不受保护，可直接利用远程方式。从已识别出的易受攻击的应用情况来看，我们发现410个易受攻击的应用程序共有956个潜在漏洞。我们手动确认了57个应用程序的漏洞，其中包括在官方市场上下载量在1000万到5000万之间的受欢迎应用程序以及预装在某些设备上的应用程序楷模。这些漏洞可能会被用来造成严重的损害，例如远程窃取联系人，照片甚至是安全凭证，以及执行敏感操作，如恶意软件安装和恶意代码执行。我们已经报告了这些漏洞，并且已经被其中一些应用程序开发人员所认可。我们还针对每种使用场景提出了对策和改进措施。

开放端口（或监听端口）是用于在计算机网络模型中接受传入连接的通信端点，通常由服务器应用程序用于处理来自远程客户端的请求。但是，如果没有仔细保护，这些端口也可能被恶意客户端连接，从而将服务器软件中的潜在漏洞暴露给远程利用。这种固有的弱点一直伴随着在整个网络服务历史中开放端口的使用，为大量严重的互联网攻击（TCP SYN floodingattacks [27]，Conficker蠕虫[14]以及最近的Heartbleed bug [9 ]。为了缓解这些传统使用场景中的问题，通常采用防火墙和用户认证机制。在最近向移动时代进化的过程中，智能手机操作系统继承了对开放端口的支持。但是，智能手机应用程序（应用程序），托管网络服务等传统开放式端口使用情况不再适用。一个主要原因是与具有有线网络连接的固定式服务器机器相比，智能手机的移动性使其难以维护稳定的IP地址。此外，分配给移动设备的IP通常位于NAT（网络地址转换）之后，以防止进入的网络连接。此外，不断接收网络流量可能会轻易地消耗移动设备电池，导致拒绝服务（DoS）攻击[49]。由于这些固有的差异，我们目前对智能手机使用开放端口的理解有限。随着智能手机的大量普及，任何潜在的智能手机开放端口使用可能会直接使最终用户受到严重损害。最近有几个这样的例子被称为“Wormhole”应用程序[12]，其中流行的Android应用程序的开放端口允许攻击者远程收集位置数据，插入联系人，甚至未经授权甚至安装应用程序，超过100M设备受到影响。虽然这些漏洞是令人震惊的，但是这些漏洞是否暴露在智能手机生态系统中的开放端口的普遍使用案例中，还是仅仅是由于糟糕的实施做法。在这项工作中，我们首次系统地研究了开放端口使用和移动平台。为了实现这一目标，我们设计并实现了一个名为OPAnalyzer的工具，它可以有效识别和描述Androidapps中易受攻击的开放端口使用情况。为了使用OPAnalyzer，我们首先使用程序分析语言对开放端口appdesign模式进行形式化，然后在高级别中指定开放端口触发哪些敏感功能以及如何触发它们。通过这些定义，OPAnalyzer首先使用静态污点分析来整合来自远程输入入口点的信息流，并识别可能触发的敏感功能。在这一步之后，会产生一组开放端口的使用路径，如果没有很好的保护，将导致远程攻击。为了帮助确定人员检查的优先级，OPAnalyzer会检查保护敏感功能的使用路径中的安全检查。如果发现给定路径的执行没有约束或仅包含弱检查，则直接揭示潜在的远程利用（第4.4节）。 OPAnalyzer还可以动态测试脆弱端口是否默认打开，如果相应端口在应用启动时自动打开，则将弱路径标记为高度不安全。为了实现高精度，我们的设计采用了Amandroid方法[46]，该方法支持流量，上下文敏感的数据流分析。为确保高效率，我们克服了工具实施中的几个工程挑战。首先，分析需要准确识别受权限保护的API，但最近的工作PScout [21]提供的权限映射API对于我们的目的来说是不完整的，因为它没有考虑API使用的先决条件。为了解决这个限制，我们改进PScout以自动修复一些常见的遗漏案例（第4.3节）。其次，我们发现Java反射通常用于处理来自开放端口的远程输入，而许多静态污点分析工具（如Amandroid）不能解决这个问题。为了确保调用图的完整性，我们增加了一个额外的分析来定位目标类或方法，它成功地解决了我们的应用数据集（§4.4）中超过86％的Java反射用例。第三，我们发现许多应用程序实际上在本地代码中实现了开放端口使用，而这仅仅是Java层静态分析所无法捕获的。因此，我们的工具还包括基于二进制分析技术的本机代码支持，几乎所有现有的静态分析由于采用了Android应用的工具，因此需要高度的工程设计[20,33,34,46]（第4.2节）。使用OPAnalyzer，我们对来自Google Play的24K热门Android应用进行开放端口使用分析，并成功对99 ％使用路径分成5个类别：数据沙