基于QEMU的Linux应用异常通信行为分析

敖 权[1] 陆慧梅[1] 向 勇[2] 曹睿东[2]
(北京理工大学计算机学院 北京 100081)[1] (清华大学计算机科学与技术系 北京 100084)[2]

摘 要 文中提出了一种基于QEMU的异常通信行为的半自动分析方法(Socket Analysis based on QEMU,SAQ),该方法能够及时发现Linux中elf格式应用程序的异常通信,预防信息泄露。通过改写QEMU,开发了一款动态跟踪工具QEMU-TRACER,SAQ可利用QEMU-TRACER定位应用程序中的可疑通信函数;通过二进制代码修改,逐一屏蔽可疑通信函数,并通过对比修改前后程序行为的变化来确定和清除异常的网络通信。针对OpenSSH和ProFTPD的测试表明,SAQ能够发现并成功屏蔽其中的异常通信行为。

关键词 隐蔽通信,动态跟踪,QEMU模拟器,函数调用,二进制修改
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.05.016

QEMU Based Abnormal Communication Analysis of Linux Applications

AO Quan[1] LU Hui-mei[1] XIANG Yong[2] CAO Rui-dong[2]
(School of Computer Science and Technology,Beijing Institute of Technology,Beijing 100081,China)[1]
(Department of Computer Science and Technology,Tsinghua University,Beijing 100084,China)[2]

Abstract This paper presented a semi-automatic analysis method based on QEMU emulator(Socket Analysis based on QEMU,SAQ),which can be used to detect covert communication of elf format program on Linux platform and prevent information leakage. By modifying QEMU,a dynamic tracing tools QEMU-TRACER was developed,which can locate the suspicious communication functions in the application using QEMU-TRACER. Utilizing binary rewriting,the suspicious functions were disabled one by one,and then the behaviors of program before and after modification were compared to determine and clear the abnormal communication. Experiments of OpenSSH and ProFTPD show that SAQ can detect the abnormal communication behaviors and succeed in disabling them.

Keywords Covert communication,Dynamic tracing,QEMU emulator,Function call,Binary rewriting

1 引言

随着计算机技术的快速发展,网络上出现了越来越多的恶意软件,包括木马、蠕虫、Rootkit、广告、后门、间谍程序等。恶意程序的泛滥使得计算机安全面临很大的威胁,仅2016年第二季度,Panda Security[1]就检测出了1800万款新的恶意软件样本。

恶意软件存在于各种操作系统中,并且随着互联网的飞速发展而广泛传播。Windows操作系统更加关注个人娱乐和办公,拥有庞大的用户群。Quick Heal报告[2]显示,2016年第一季度,恶意软件被检测3亿多次。对于Windows平台程序,早期的分析过程很难深入到程序的执行过程中;近年来,由于Pin[3]的广泛使用,研究人员能够通过动态插桩的方式来研究Windows平台上程序的行为[4]。Linux系统开源、免费且稳定,其中恶意软件相对较少,但该系统上运行着各种服务器软件,信息泄露会对大量用户的数据安全产生威胁。由于其开源的特性,分析Linux应用的工具有很多,如Strace[5]和SystemTap[6]等。Android占据了智能终端设备的大部分市场,市场份额从2009年的1.6%迅速增长到2016年的86.2%[7];同时,Android设备上的恶意软件数量也在快速增加,Intel在Mobile Threat Report 2016[8]中指出,全球大部分地区每小时都有100~1000个恶意软件被检出,恶意软件高发区——中国每小时有超过6000个恶意程序被检测出来。由于Android操作系统使用Linux内核,因此Android的安全性研究对于研究Linux系统具有很好的借鉴作用。

在种类繁多的恶意软件中,程序后门、间谍程序、广告等在未明确提示用户或未经用户许可的情况下恶意收集、发送用户的信息,与传统的病毒不同,它们通常是用户自愿使用的软件,并且攻击者事先在程序中植入了一些恶意代码来盗取用户的数据。

到稿日期:2017-04-29 返修日期:2017-07-12 本文受核高基项目(2012ZX01039-004-4,2012ZX01039-003)资助。
敖 权(1992-),男,硕士生,主要研究方向为操作系统和计算机网络;陆慧梅(1974-),女,博士,副教授,CCF高级会员,主要研究方向为操作系统和计算机网络;向 勇(1967-),男,博士,副教授,主要研究方向为操作系统和计算机网络,E-mail:xyong@csnet4.cs.tsinghua.edu.cn(通信作者);曹睿东(1992-),男,硕士生,主要研究方向为操作系统和计算机网络。

本文关注 Linux 平台上程序的网络通信行为，通过严格把控网络通信来降低用户隐私泄露的风险，并尽可能识别软件中的后门。Schuster[9] 给出了后门软件的定义："后门是一个隐藏的、没有文档说明的并且不被需要的程序，或者后门是一种程序修改或实现，其通过行使不被需要或没有文档说明的恶意动作来绕过安全机制。"

间谍程序隐藏在受害者的计算机中，并暗中向外发送一些用户的数据。整体来说，此类软件会在用户不知道的情况下进行收发数据等操作，而用户感受不到这些操作带来的效果，或者说这些操作的结果是用户不希望看到的，例如恶意 OpenSSH[10] 上传用户的用户名和密码。

目前，Linux 程序分析技术可以分为静态分析和动态分析两大类。

1）静态分析

静态分析一般不需要将程序放在真实的环境中运行。静态分析可以帮助人们理解程序的执行流程，例如 Source Insight[11] 和 Understand[12] 通过分析程序源码可以画出函数静态调用图，而 Egypt[13] 和 CG-RTL[14] 利用 GCC 的中间结果——寄存器传送语言 RTL，生成函数调用图；在静态程序分析中，常用抽象的符号来表示程序中变量的值，例如 PREfix[15] 利用符号执行来查找程序中的错误。静态分析方法相对容易，一部分静态分析工具依赖于程序源码。通过静态分析可以对程序进行比较全面的分析，分析范围更广，分析的路径也更多，易于找出隐蔽的执行路径，但可能会产生路径爆炸的问题。由于在静态分析过程中无法确定程序的各种输入以及运行环境，因此其分析的精度不高。

2）动态分析

动态分析需要将程序放在真实的环境中运行，在程序运行的过程中通过内部或外部的手段对程序进行跟踪，例如通过 SystemTap[6] 获取程序运行时的函数调用序列，以得到函数的调用序列图，用于程序的行为分析；另外，系统调用、参数以及返回值也常作为程序特征[16-17]。动态分析的优点在于可以准确获取系统以及程序的运行状态，Shahzad[18] 对进程控制块 PCB 进行读取和分析，以区分恶意软件；在动态分析过程中一般无法获取程序的源码和调试信息等，采用二进制插桩或者基于模拟器的方法可以直接对二进制程序进行深入分析，例如 Pin[3] 可以对程序进行指令级和函数级插桩；向勇[19] 利用 QEMU 动态跟踪内核函数的调用和返回；S2E[20] 对符号执行添加了约束，可以选择性地分析感兴趣的路径。动态分析一般只分析执行过的路径，因此在覆盖范围方面存在不足。

本文重点研究隐蔽的网络通信。网络后门、间谍程序为了达到更隐蔽的效果，通常会判断程序的网络连接或者网络发送数据是否成功，以便在网络无法连接或者发送数据失败时采取相应的措施。隐蔽的网络通信语句是否正确地返回结果不会影响程序的正常功能。例如，在 socket 编程中，执行 send 函数之前需要检查 connect 函数的返回值，如果返回 0，则执行 send 函数；如果 connect 失败并返回 −1，则不发送数据。基于异常通信的有无对程序的正常功能没有影响的假设条件，本文提出了一种半自动化的分析方法 SAQ，并将其用于对 Linux 平台上 elf 格式的应用程序进行动态分析。通过分析应用程序的网络行为，SAQ 可以从恶意程序的二进制代码中定位出隐蔽通信的位置，并且给予屏蔽；分析结束后，SAQ 将为用户提供一款屏蔽异常通信的程序。

本文的主要贡献如下：

1）基于 QEMU 模拟器，设计并开发了动态跟踪工具 QEMU-TRACER。通过修改 QEMU，在指令动态翻译过程中截获函数调用和返回，从而动态跟踪客户机操作系统的函数调用和返回信息，并获取函数调用参数和函数调用栈，进而回溯和定位异常函数。

2）基于工具 QEMU-TRACER，提出了一种 Linux 平台上针对应用软件异常通信行为进行分析的方法 SAQ。通过跟踪程序的运行并分析网络相关的函数调用栈和参数，SAQ 可以定位程序中异常通信的函数调用，并实现了对程序的半自动化分析。本方法不依赖程序的源码、调试信息甚至符号信息，使用者只需要提供可以运行的二进制程序即可，其通用性较好。

3）将分析过程与漏洞修复过程结合。由于 QEMU-TRACER 记录了函数调用的地址信息，因此通过 SAQ 分析方法不仅可以识别出程序中存在的异常通信，还能在二进制代码中定位异常函数调用的位置，从而在识别到异常通信之后可以将其封堵。

2 相关工作

在动态分析中，分析系统调用是 Linux 应用行为分析的常用方法。确定内核版本之后，系统调用的个数以及定义也随之确定，分析系统调用时不依赖于应用程序，因而该方法的适用范围较广。Maggi[16] 使用 OpenBSM 审计工具来获取系统调用和参数，并对系统调用序列和参数进行分析以实施入侵检测。Asmitha[17] 利用 Strace 获取系统调用，并动态筛选最少且最具区分度的系统调用，将其调用次数作为特征来对程序进行分类。分析系统调用的方法对程序依赖少，但恶意程序可能会通过无用的系统调用绕过检测。

通过动态分析能够获取操作系统内部的一些数据结构，可利用其值来判断某个程序的执行过程是否存在异常。Shahzad[18] 通过分析进程控制块 PCB 的信息来区分恶意程序和普通程序，并将进程拥有的页帧数以及自愿上下文切换的次数作为区分指标。

以上方法只能从统计学的角度对程序进行分类处理，无法进一步定位程序中的恶意语句。针对 Linux 平台上含客户端的服务器程序，Schuster[9] 提出了一种能够自动鉴别并移除二进制程序中后门的方法，通过对比控制流路径来得到某些特有的路径，从而找出可疑的区域，最后通过二值插桩或者二值重写来屏蔽某个功能。目前，该工作主要分析程序授权行为。

针对 Android 手机的异常行为检测，Saracino[21] 基于 Xposed 框架开发了一款 Data-Sluice 的框架。Xposed 具有钩住（hook）java 方法的能力，可以在 hook 的方法调用前定义一

些操作,通过 Data-Sluice 框架监听与网络相关的特定函数,用以动态地控制 Android 中 App 的网络连接行为,从而有效去除应用程序中的广告,并且防止数据泄漏。2015 年,Rubin[22]发现 Google Play 中排名前 500 的免费应用软件中 63%的网络通信是非必需的,这些额外的通信对用户能够观察到的程序功能没有影响。在其动态分析方法中,Android 的 Apk 被反编译成 java 代码后,逐个屏蔽掉这些触发网络连接的语句,同时加入相应的异常处理机制,通过对比网络通信语句被屏蔽前后程序运行的结果来判断该网络连接是否属于隐蔽通信。本文受其启发,采用类似的方法来分析 Linux 上的应用程序。

近年来,模拟器 QEMU 常被用于对程序进行动态分析,例如 S2E 利用 QEMU 运行目标程序。QEMU 采用动态二进制翻译技术,先将虚拟机中要执行的指令翻译成宿主机可以识别的指令之后再运行。文献[19]利用 QEMU 跟踪内核的运行,禁用了 QEMU 的块连接机制,并在块粒度上截获虚拟机中的函数调用和返回指令,其跟踪的数据包括虚拟机时间、进程标识符(CR3 寄存器)、栈顶指针、函数入口地址。基于文献[19]的研究,本文开发了 QEMU-TRACER,其不仅可以跟踪内核的运行,也可以跟踪用户态指定程序的运行,跟踪范围通过配置文件指定。由于 QEMU-TRACER 不依赖源码、调试信息和符号信息,因此 SAQ 方法具备分析二进制程序的能力。

3　方法的提出

3.1　系统描述

SAQ 方法可以分为跟踪分析阶段、修改验证阶段和结果生成阶段;从功能上,可以分为 4 个模块:跟踪模块、分析模块、二进制修改模块和验证模块,如图 1 所示。

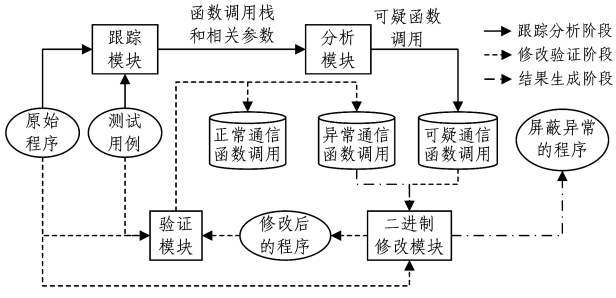

图 1　SAQ 方法框架

Fig. 1　Framework of SAQ

跟踪模块利用 QEMU-TRACER 动态跟踪指定应用的网络通信,并记录发生网络通信系统调用时的函数调用栈以及相关参数信息,将跟踪模块的结果作为分析模块的输入。分析模块通过分析函数调用栈信息得到应用进程中可疑通信函数的位置。跟踪和分析是 SAQ 方法的第一个阶段,该阶段被拆分成两个模块,使得跟踪模块的功能更加专一。由于在模拟器中跟踪程序运行的速度较慢,因此将分析过程提取出来以降低系统负载。在修改验证阶段,二进制修改模块读取分析模块中定位的可疑函数调用地址,通过二进制代码修改来逐一屏蔽应用中的可疑通信函数。该模块不依赖于前面两个模块,可以独立运行,在修改验证阶段和结果生成阶段都被用到。其作为一个独立的模块,降低了系统的耦合性。验证模块通过运行测试用例来对比程序修改前后行为的变化,从而确认异常通信函数。该模块需要分析人员根据不同的测试程序来编写测试用例。验证完毕之后,进入结果生成阶段,最终生成屏蔽了异常网络通信的程序。

SAQ 方法中的 4 个模块采用不同的编程语言。跟踪工具 QEMU-TRACER 使用 C 语言编写,每次对 QEMU-TRACER 进行修改后再运行时都要先编译,并且调试难度较大,划分模块简化了 QEMU-TRACER 的功能。分析模块和二进制模块采用 python 编写,实现简单,而且修改灵活。验证模块则采用 Linux Shell 编写,以便于使用 Linux 平台的工具。

3.2　跟踪模块

跟踪模块需要解决的问题是,在执行网络通信系统调用时,获取函数调用栈以及相关参数信息。与文献[19]相比,QEMU-TRACER 中记录了线程名和线程号,并将内核态跟踪扩展到了内核态和用户态,使其不仅能够跟踪内核态函数的调用和返回——包括全部的内核函数以及内核可加载模块,也能跟踪用户态程序。此外,在 QEMU-TRACER 中加入函数调用栈机制和参数获取机制,使其能够获取函数调用路径和指定函数的部分参数,以便进行异常行为的确认。QEMU-TRACER 的另一个优点是可通过配置文件动态指定跟踪范围,而不是每次都跟踪整个系统的函数调用和返回信息,加快了跟踪速度。

3.2.1　QEMU-TRACER 获取函数调用栈

对于获取函数调用栈,本文最初考虑通过分析 ebp 寄存器来进行回溯,但一些程序在编译过程中加入-fomit-frame-pointer 选项进行优化,程序去掉了 ebp 寄存器入栈以及将旧的 esp 寄存器的值赋给 ebp 寄存器的操作,这将导致无法通过 ebp 来回溯栈结构,因此需要另外的获取函数调用栈的机制。

QEMU-TRACER 获取函数调用栈时可以采用先记录后分析的方式,即先动态跟踪 Linux 内核中所有的函数调用和返回信息,然后从大量日志信息中匹配函数调用和返回。由于须跟踪全部的函数调用和返回,因此 QEMU-TRACER 的 I/O 负载会变得较大,例如跟踪 BusyBox 启动并马上关机的过程的内核函数调用与返回的数据量达 1.5 GB。为提高 QEMU-TRACER 的运行速度,本文采用实时分析的方法。

为了获得函数调用栈,在 QEMU-TRACER 中维护跟踪线程的调用栈,当创建一个线程时,就为该线程新建一个栈,若遇到 call 则入栈,若遇到 return 则退栈。当出现指定的系统调用时,将该线程的函数调用栈输出,以实时得到函数的调用栈。在函数调用和返回匹配的过程中,针对 fork 函数、带立即数的 ret 指令等情况做了特殊处理,以保证函数调用和返回相匹配。

为进一步提高跟踪运行速度,QEMU-TRACER 主要关注跟踪线程的用户态函数以及特定的系统调用,以过滤掉大部分内核态函数。基于内核地址空间的唯一性,文献[19]在

只跟踪内核时将栈顶指针 esp 的低 13 位屏蔽为 0 后的值作为线程标识，但在跟踪用户态以及内核态的函数调用和返回信息时，由于用户态地址空间不唯一，而且同一线程处于用户态和内核态时的栈顶指针 esp 也不一样，因此在 QEMU-TRACER 的实现中先读取 CPU 的任务寄存器 tr 以找到当前线程的任务状态段 TSS。TSS 的第二个字段 esp0 指向内核态的堆栈栈顶，把内核态栈栈顶的低 13 位屏蔽后就能访问到 task_struct，进而读取 tid 以及 processname。此方法将用户态和内核态跟踪统一化。

3.2.2 QEMU-TRACER 获取函数参数

在分析过程中，只依据函数名并不能精确地确定该操作的行为，而被跟踪的函数的参数有时能够准确地反映程序的行为。在 QEMU-TRACER 中加入参数获取机制，这样通过直接内存读取便可以获取指定的参数，这些参数将作为分析模块的输入。例如在分析网络行为时，发起网络连接的系统调用为 sys_connect，其定义为：SYSCALL_DEFINE3(connect, int, *fd*, struct sockaddr__user*, uservaddr, int, addrlen)。该系统调用连接到一个 socket，为发送数据做准备。sys_connect 系统调用的第一个参数 *fd* 表示网络地址族，当 *fd*=AF_LOCAL|PF_LOCAL 时，sys_connect 只是进行本地的通信连接，目前 SAQ 认为本地网络通信是安全的。通过参数的获取，SAQ 还能够获取通信对方的 IP 和端口信息，例如 sys_connect 系统调用的第二个参数指向一个套接字结构体，通过读取该指针指向的内存地址，即可获取通信对方的 IP 地址和端口信息。

3.2.3 跟踪过程

跟踪应用程序的运行时，需要在配置文件中指定需要跟踪的程序名、跟踪的函数以及参数信息，其中函数通过其内核地址来指明，配置文件的格式如表 1 所列。

表 1 跟踪模块的配置文件格式

Table 1 Configuration file format of tracing module

配置项	描述	举例
第一行	跟踪的程序名	proftpd
第二行	跟踪的内核态的范围	7fffffff000－ffffffffffffffff
第三行	跟踪的用户态的范围	0－7fffffff000
第四行	程序 got 表的地址，用于打印动态链接库信息	linkmap:68f000
第五行	是否跟踪函数调用栈，取值为 0 或 1	funcstack:1
第六行及以后	被跟踪函数的地址以及需要记录的参数的位置和类型，目前支持 int，string，socket	ffffffff816c8530,1,socket ffffffff816c8550,1,socket …

从配置文件中可以看出，指定一个程序进行跟踪时，需要指明程序名，然而 QEMU-TRACER 跟踪该程序的子进程时，由于子进程的名字可能与父进程不同，因此仅利用程序名可能无法进行跟踪。为了解决该问题，QEMU-TRACER 在运行过程中维护了一个跟踪线程的列表，最开始利用程序名进行匹配，一旦跟踪到匹配的线程后，就将该线程的线程号加入到该列表，接下来的跟踪便开始利用线程号进行匹配。若某个线程不在该列表中，但其父进程在该列表中，则表明该线程是跟踪程序的子线程，将该线程加入到列表中，开始跟踪该线程。

本文在获取函数调用栈信息时，参照文献[19]中的格式输出函数调用信息，具体如表 2 所列。在输出函数调用栈之前，SAQ 先输出跟踪的 IP 地址和端口等参数信息，参数信息的格式如表 3 所列。实验跟踪了 sys_sendto，sys_accept，sys_connect 和 sys_recvfrom 4 个系统调用。

表 2 QEMU-TRACER 输出函数调用栈的格式

Table 2 Format of function call stack outputted by QEMU-TRACER

名称	描述
类型	函数调用，使用 C 表示
线程名	被跟踪线程的名字
调用地址	该地址处调用了另外一个函数
被调地址	被调用函数的入口地址
进程标识	进程的唯一标识，cr3 寄存器
线程标识	线程的线程号
栈顶指针	栈顶指针 esp 的值

表 3 QEMU-TRACER 输出参数的格式

Table 3 Format of function parameters outputted by QEMU-TRACER

名称	描述
类型	用 S 表示，用以区分调用栈
地址族	值为 AF_INET 或 PF_INET 时表示与外界通信
端口号	通信的端口
IP 地址	通信的对方 IP 地址

为了让 SAQ 自动化地运行，需要人工编写测试用例，但是对于图形界面 GUI 程序，还不能很好地实现自动化，目前仍采用手动运行的办法。对于字符界面程序，将待分析程序的一些常用的命令写入测试用例，在开机后自动化地运行这些命令，并且捕获程序的输出和运行结果。编写测试用例时，需要对被分析程序的约定功能有一定了解。例如在分析 ProFTPD 时，使用了 get 命令和 put 命令，测试程序如图 2 所示。

```
1. #! /bin/bash
2. qemuIP= $ 1
3. ftp -i -n  $ qemuIP<<EOF
4. user usernamepassword
5. binary
6. get a. txt
7. put b. txt
8. bye
9. EOF
10. ssh root@ $ qemuIP "poweroff"
```

图 2 ProFTPD 测试用例

Fig. 2 Testcase of ProFTPD

3.3 分析模块

分析模块用于分析跟踪模块输出的函数调用栈以及相应的参数信息，并得到可疑网络通信函数调用的地址。分析模块首先通过参数信息判断该网络通信是否安全，若判断网络通信可疑，则回溯函数调用栈，定位可疑函数调用。

分析模块的流程如图 3 所示。分析模块依据 IP 地址和网络通信地址族来判断某通信过程是否属于本地通信。网络通信地址族表示该网络通信的类型，通过判断其是否为 AF_INET 或 PF_INET 能排除一部分本地通信，如果该条件成

立,再判断 IP 是否属于本地 IP,SAQ 的分析模块默认本地的通信是安全的。由于 sys_accept 系统调用的 *sockaddr* 套接字参数在传入之前没有被赋值,其值在连接之后被写入,QEMU-TRACER目前不能获取到该参数,在实验中读取出的 sys_accept 系统调用参数没有意义,无法判断该连接是否属于本地连接,因此对于 sys_accept 系统调用,不管判断其属于本地通信还是远程通信,全部都需要进行分析,这样才能确保不会遗漏异常网络通信。

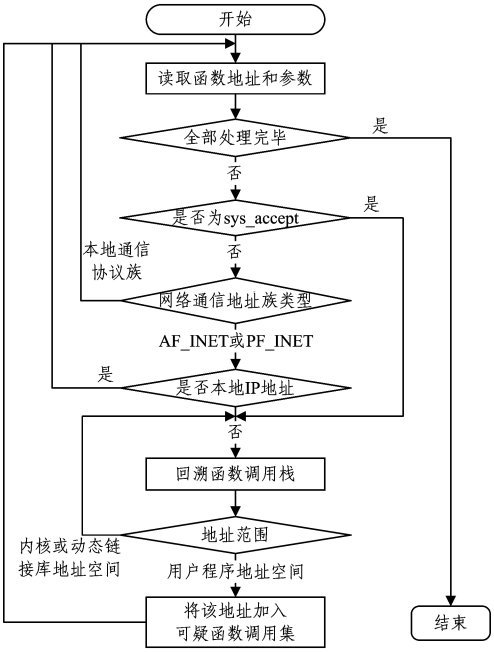

图 3　分析模块流程图

Fig. 3　Flow chart of analysis module

接下来,需要定位可疑函数调用的地址,以便于后续屏蔽该函数调用。屏蔽与网络通信相关的函数,以观察该网络通信语句对程序的影响。直观思路是对动态链接库中的系统调用语句进行屏蔽,但对动态链接库的改写会影响到系统中的其他程序,目前的解决方案是屏蔽位于动态链接库上一层并且处于应用程序中的函数调用,因此分析模块需要定位出该函数调用。根据程序虚拟地址分布空间的不同,可以对函数调用栈中的地址进行区分。以 x86_64 系统为例,若用十六进制表示地址,则内核态函数的地址空间为 0x7fffffff000－0xffffffffffffffff,动态链接库的地址一般大于 0x7ffff0000000,而用户态程序的地址空间一般都小于 1 G。根据以上规则逐级回溯函数调用栈,找到动态链接库的上一级函数调用,并将其调用地址存入可疑函数调用集中。

3.4　二进制修改模块

通过分析模块得到可疑函数调用地址之后,二进制修改模块读入这些地址并逐个屏蔽,以达到屏蔽特定网络通信的目的。二进制修改模块使用 python 脚本编写,接收两个参数,第一个参数是被修改的程序路径,第二个参数是一个字符串数组,其中保存了需要屏蔽的函数调用的地址。在修改验证阶段,根据分析模块得到的可疑函数列表来对程序进行逐一修改和屏蔽。在该阶段,每次调用二进制修改模块时,字符串数组中只包含一处可疑函数调用的地址;而在结果生成阶段,该字符串数组中可能会存在多个函数调用的地址,用于一次性屏蔽掉所有触发异常通信的函数调用。二进制修改模块实现了完全自动化的修改,不需要人工参与,实现过程如图 4 所示。

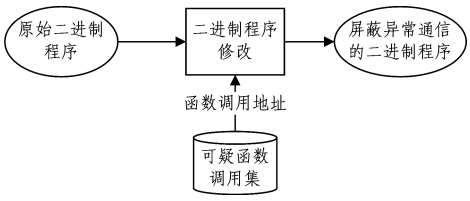

图 4　二进制修改过程

Fig. 4　Binary rewriting process

在修改过程中,二进制程序不会被反编译成汇编语言,二进制修改模块读取程序后,直接对其进行修改,修改的过程可被看作是内容替换。本实验跟踪的 4 个网络相关的函数调用失败时都返回－1。二进制修改的目的是使用 mov $ffffffff,%eax 指令替代 call 指令,即使用二进制的 0xb8ffffffff 来替换程序 call 指令,在屏蔽函数调用的同时,将 eax 寄存器赋值为－1。模拟函数调用返回－1,表示该函数调用失败。

3.5　验证模块

验证模块的目的是找出异常通信函数调用,其判断条件是“可疑通信函数调用集合中不影响测试用例中约定程序功能的函数调用”。在二进制修改模块中,程序中可疑的网络通信函数调用被逐一屏蔽,并生成一系列的修改程序。为了判断程序在被屏蔽了某个网络通信函数后的功能和行为是否正常,验证模块逐一将这些程序放在相同的环境下,并运行与跟踪阶段相同的测试用例脚本,对比程序修改前后的运行结果,从而判断异常通信。以此为依据,验证模块对可疑的网络通信函数调用进行分类,生成异常网络函数调用地址的集合。

验证模块的功能是对比程序修改前后的运行差异。由于被分析程序的功能不同,其运行结果也不同,因此对程序运行结果的比较需要分析人员参与。对于字符界面程序,判断运行结果相对容易。以本文中列举的 ProFTPD 为例,通过验证 a. txt 是否被下载以及 b. txt 是否被上传来对比程序修改前后的行为。对于 GUI 程序,在相同操作的情况下,不同的程序界面反映了程序的差异,Rubin[22] 使用 ImageMagick 来识别两台机器中用户界面的变化,但最终仍需人工确认。为了简化,对于 GUI 程序,直接采用人工确认的方法,GUI 界面的自动化对比可能是未来工作的一部分。

4　实验与结果

选取 GUI 和控制台两类程序来进行实验。对于 GUI 程序的分析,以 Sublime Text 为例来说明本方法的可行性;对于控制台程序,选取了两款常用的 Linux 服务器程序——OpenSSH 和 ProFTPD。由于一些程序的运行结果很难通过机器自动化验证,而且隐蔽通信的触发条件比较复杂,目前本文所给的例子较为简单,后续将完善自动化验证,使得本方法适用于更多场景的恶意软件分析。实验中,客户机使用 64 位的 Lubuntu15. 10,实验结果如表 4 所列。

表 4 程序网络通信的分析结果

Table 4 Analysis results of network communication of programs

应用程序	系统调用	可疑函数调用地址	屏蔽后对程序正常功能的影响	其他行为	通信类型
Sublime Text	sys_connect	0x5bc9e6	不再提示版本更新		正常
OpenSSH	sys_connect	0x43c069	无影响		异常
OpenSSH	sys_accept	0x408247	无法连接 ssh 服务器		正常
ProFTPD	sys_sendto	0x42990c	无影响	SocketReceiver 未收到登录凭证	异常
ProFTPD	sys_sendto	0x467504	无影响	SocketReceiver 收到登录凭证	异常
ProFTPD	sys_connect	0x418844	连接服务器后命令操作失败	SocketReceiver 收到登录凭证	正常
ProFTPD	sys_accept	0x418535	无法连接 ProFTPD 服务器	SocketReceiver 未收到登录凭证	正常

4.1 Sublime Text

Sublime Text 是一款文本编辑器，由于功能强大且界面简洁而深受很多用户的喜爱。将 Sublime Text 作为 GUI 程序分析的一个例子，来验证 SAQ 方法的有效性。实验选取的 Sublime Text(Build 3114 版本)在每次启动过程中都会提示用户更新，检测版本更新并不是一个恶意的网络行为，仅以此为例说明 SAQ 分析方法适用于 GUI 程序。测试过程中只打开了程序，然后关闭，没有别的任何操作，我们尝试分析该操作中是否存在一些通信行为以及这些通信是否是非必须的。通过分析实验数据可以看出，SAQ 跟踪到了 sys_connect 系统调用，实验跟踪出的部分数据如表 5 所列，其格式参考 3.2.3 节。

表 5 QEMU-TRACER 跟踪数据

Table 5 Log data of QEMU-TRACER

应用程序	跟踪数据
Sublime Text	S,2,80,209.20.75.76 C,update_check,ffffffff817ef9f0,ffffffff816c8550,000000007c359000,1023,ffff88007c3a3f50 C,update_check,00000000005bc9e6,0000000000405d40,000000007c359000,1023,00007fffd2dc88c8 C,update_check,00000000005505cb,00000000005bc89a,000000007c359000,1023,00007fffd2dc9b58 C,update_check,000000000047bcf5,00000000005504de,000000007c359000,1023,00007fffd2dc9c78 C,update_check,0000000000073897e,000000000047c1cc,000000007c359000,1023,00007fffd2dc9ef8 C,update_check,00007ffff74586a8,0000000000738960,000000007c359000,1023,00007fffd2dc9f18 C,update_check,00007ffff6947e9b,00007ffff74585e0,000000007c359000,1023,00007fffd2dc9fb8
OpenSSH	S,2,80,190.123.47.106 C,sshd,ffffffff817ef9f0,ffffffff816c8550,00000000778e0000,626,ffff8800778ebf50 C,sshd,000000000043c069,0000000004060a8,00000000778e0000,626,00007fffffff9db8 C,sshd,0000000004072a5,000000000043bf50,00000000778e0000,626,00007fffffff9eb8 C,sshd,0000000000409e8c,0000000000407090,00000000778e0000,626,00007ffffffe4f8 S,0,0,0.0.0.0 C,sshd,ffffffff817ef9f0,ffffffff816c8530,00000000778e0000,626,ffff8800778ebf50 C,sshd,0000000000408247,0000000000406748,00000000778e0000,626,00007fffffff8168 C,sshd,0000000000409efb,0000000000407a50,00000000778e0000,626,00007ffffffe4f8
ProFTPD	S,2,53,10.108.20.208 C,proftpd,ffffffff817ef9f0,ffffffff816c8580,000000007c8af000,1086,ffff88007c333f50 C,proftpd,000000000042990c,0000000000403c80,000000007c8af000,1086,00007ffffffe398 … S,2,53,10.108.20.208 C,proftpd,ffffffff817ef9f0,ffffffff816c8580,000000007c8af000,1086,ffff88007c333f50 C,proftpd,0000000000467504,0000000000403ae0,000000007c8af000,1086,00007ffffffc028 … S,2,51982,10.108.20.208 C,proftpd,ffffffff817ef9f0,ffffffff816c8550,000000007c8af000,1086,ffff88007c333f50 C,proftpd,0000000000418844,0000000000404360,000000007c8af000,1086,00007ffffffe3e8 … S,1,0,0.0.0.0 C,proftpd,ffffffff817ef9f0,ffffffff816c8530,000000007c1d1000,635,ffff88007c1f3f50 C,proftpd,0000000000418535,00000000004042b0,000000007c1d1000,635,00007ffffffe8f8 …

从表 5 中可以看到，线程号 pid 为 1023、线程名为update_check的线程连接了 209.20.75.76 这个 IP 地址的 80 端口。参照 3.3 节描述的规则，SAQ 认为这个行为有可能是异常的；接下来，SAQ 自动分析可疑函数的调用栈，找到 Sublime Text 程序中触发该行为的函数调用地址，分析模块认为程序中地址为 0x5bc9e6 的函数调用导致了 sys_connect 系统调用的产生，从而二进制修改模块屏蔽 0x5bc9e6 处的函数调用语句。值得注意的是，以上栈结构中没有出现动态链接库中的函数调用，程序似乎没有经过动态链接库就直接进入了内核。这是由于程序在调用动态链接库中的函数时先调用了一个以@plt 结尾的函数，此时函数调用栈第二行中的 0x405d40 为 connect@plt 的入口地址；接下来程序通过 jmp 指令跳转到动态链接库中，在没有调用其他函数的情况下直接执行了 syscall 指令，而 QEMU-TRACER 只跟踪函数调用和返回，所以出现了以上的栈结构。

屏蔽该网络通信前，每次启动 Sublime Text 时都会弹出

一个提示用户下载新版的窗口,提示的界面截图如图 5 所示。在屏蔽上述网络通信后,Sublime Text 启动时不再出现更新的提示。

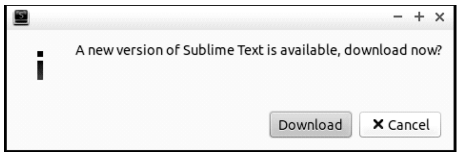

图 5　Sublime Text 软件更新提示

Fig. 5　Software update reminder of Sublime Text

为了进一步验证实验结果,采用 Strace 来跟踪修改前后程序的运行,并确认修改之后的程序没有与 209.20.75.76 的 80 端口通信。观察到的现象以及进程名 update_check 表明,Sublime Text 的该网络连接操作是在启动时检测有无版本更新,查询得到 www.sublimetext.com 域名的 IP 地址是 209.20.75.76,从而证实了 SAQ 跟踪的 IP 和端口的正确性。

4.2　OpenSSH

OpenSSH 是 OpenBSD Secure Shell 的简称,是 ssh 的开源实现,用于对远程控制或文件传输中的数据进行加密,以保证数据的安全性。2013 年,一款被称为 Linux/SSHDoor.A 的恶意软件被发现[10],该 sshd 守护程序会盗取登录用户的用户名和密码。Schuster[9] 以 OpenSSH 为例重点分析了授权行为,而我们重点关注程序的网络通信行为,在此仅分析 OpenSSH 中守护程序 sshd 利用网络通信盗取用户数据的行为。

在测试用例中让 sshd 服务开机启动,利用 ssh 远程登录到 sshd 服务器并执行一些操作(例如 ls 和 cd 等),最后关机。SAQ 跟踪出的部分数据如表 5 所列。

跟踪发现,sshd 在运行过程中与 IP 为 190.123.47.106 的 80 端口进行了通信。第一处跟踪出来的数据显示在 0x43c069 处触发了 sys_connect 系统调用,SAQ 屏蔽了该处的 connect 调用后,sshd 运行的各项功能正常,而且不管是使用 QEMU-TRACER 跟踪还是使用 Strace 跟踪,都没有发现连接 190.123.47.106 的通信行为。在第二处函数调用栈中,位于 0x408247 地址处的函数调用最终导致了 sys_accept 系统调用的发生,在屏蔽了 0x408247 处的函数调用语句后,ssh 无法连接到服务器,该修改影响到了 sshd 的正常运行,这表明触发 sys_accept 系统调用通信语句是程序正常运行所必需的,即属于正常通信。

从 SSHDoor[10] 的分析可以知道,sshd 会与 openssh.info 和 Linuxrepository.org 这两个域名的服务器进行通信,我们验证了这两个域名的 IP 地址都是 190.123.47.106,与分析结果吻合。文献[10]中提及 openssh.info 和 Linuxrepository.org 两个域名的 IP 地址都是 82.221.99.69,而不是 190.123.47.106,原因在于两个域名更换了 IP 地址。

4.3　ProFTPD

ProFTPD 是一款 ftp 服务器。Schuster[9] 为了验证 WEASEL 的效果,该研究小组的其他成员在没有被监督的情况下在 ProFTPD 中植入了 11 个后门,其中有一个会盗取用户的用户名和密码,并将盗取的用户名和密码通过 53 号端口发往某 IP 地址,由于 53 号端口一般是 DNS 服务器的端口,这使得该后门更加隐蔽。该恶意软件使用明文传输用户名和密码,但是为了不引起注意,在字符串的开头和结尾加上了一些其他字符串。攻击者监听该 IP 地址的 53 号端口,就可以获取到用户的登录凭证。攻击流程如图 6 所示。

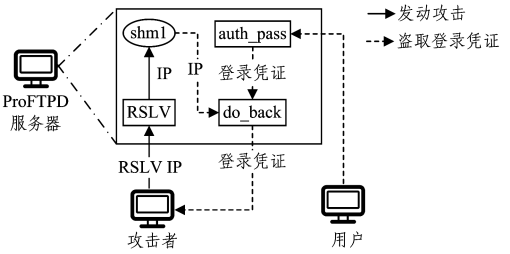

图 6　ProFTPD 后门的实现过程

Fig. 6　Implementation process of ProFTPD backdoor

为了触发该漏洞,攻击者需要向 ProFTPD 服务器指定一个 IP 地址,用于接收发送回来的用户名和密码。攻击者首先使用 telnet 登录到存在该漏洞的 ProFTPD 服务器的 21 号端口,通过运行 RSLV 命令来指定接收登录凭证的机器的 IP 地址,RSLV 模块会创建共享内存 shm1 并将该 IP 地址保存在其中,运行 RSLV 命令时不需要登录。当用户登录 ProFTPD 服务器时,授权模块 auth_pass 调用 do_back 模块,将成功登录的用户凭证发送到 shm1 指定的 IP 地址上。

在编写测试用例的过程中,攻击者和用户使用同一台机器,ProFTPD 服务器则运行在模拟器中的 Lubuntu 上,将 ProFTPD 服务器设置为开机启动,并在 Lubuntu 中安装 ssh 服务器用于远程控制。攻击者先运行 RSLV 命令,指定 IP 地址为 10.108.20.208,然后用户登录并运行 get 和 put 文件操作。为了验证该漏洞的确在偷窃用户登录凭证,用 C 语言设计了一个小程序 SocketReceiver 来监听攻击者本机的 53 号端口。在验证过程中,只需要验证 get 和 put 操作是否成功并对 SocketReceiver 的监听结果进行分析,即可区分正常与异常的网络通信。

在实验过程中,检测到 4 处与外界的通信,其中 3 处与 10.108.20.208 的 53 号端口进行了通信,通信语句为 sys_sendto,sys_connect;另外,ProFTPD 也产生了 sys_accept 系统调用。部分跟踪信息如表 5 所列,受篇幅限制,ProFTPD 跟踪的栈数据只列出了靠近栈顶的前两层。

前两处触发 sys_sendto 系统调用的函数地址分别是 0x42990c 和 0x467504,第三处在地址 0x418844 触发了 sys_connect 系统调用,第四处显示 ProFTPD 在地址 0x418535 触发了 sys_accept 系统调用,实验结果如表 4 所列。将 0x42990c 处调用 sys_sendto 的语句屏蔽之后,ProFTPD 不再向外界发送用户的登录凭证,而且正常使用 get 和 put 命令;屏蔽在 0x467504 处调用 sys_sendto 的语句后,攻击者依然能够在 10.108.20.208 的 53 号端口上收到用户登录的凭证,使用 get 和 put 命令也能够正常下载文件;当在 0x418844 处调用 sys_connect 的语句被屏蔽之后,ProFTPD 依旧向外界发送用户的登录凭证,而且命令运行不正常;对 0x418535 处的函数调用的屏蔽导致了 ftp 客户端无法连接到服务器。由此证明 0x42990c 和 0x467504 很有可能是恶意网

络通信，至少不是程序运行所必须的，而 0x418844 处的语句产生的sys_connect系统调用以及 0x418535 处产生的sys_accept系统调用是 ProFTPD正常工作所必需的。通过分析 Socket-Receiver 接收的数据，可以确定在 0x42990c 处的网络通信行为是盗取用户的登录凭证。

从跟踪方法上进行对比，文献[19]记录了所有的函数调用和返回，用于事后分析，而 QEMU-TRACER 能灵活指定跟踪程序的范围，并记录函数的部分参数。文献[22]通过反编译 Apk 变相得到了程序的源代码，Systemtap 进行跟踪时需要程序的调试信息；而 SAQ 方法不依赖程序的源码、调试信息和符号表，能够直接分析二进制程序中的恶意网络通信。

结束语 本文基于 QEMU 开发了 QEMU-TRACER 跟踪工具，并增加了实时函数调用栈获取功能，降低了 QEMU-TRACER 的 I/O 负载；在此基础上，提出了一种半自动化的分析方法 SAQ，用于动态分析 Linux 平台应用程序的网络通信行为。本方法的优势在于：1）能够有效找出 Linux 应用中的异常网络通信，并对不必要或者恶意的网络通信语句进行屏蔽，从而降低被攻击的可能性；2）不依赖程序的源码、调试信息以及符号等信息，适用范围广；3）具有比较好的扩展性，对网络行为的分析可以方便地扩展到对文件的读写以及其他敏感事件上。接下来，我们将会对网络通信之外的其他通信行为进行分析。

在后续工作中，本方法仍然可以从系统跟踪能力、异常分析能力和分析自动 3 个方面进一步完善，并且尝试将其用于分析更多、更复杂的软件。目前，该 SAQ 对于 GUI 应用还不能实现较好的自动化，主要受限于 GUI 程序的自动化执行和比较；对于命令行程序，SAQ 能够实现比较好的自动化，但是测试脚本的编写需要人工参与。目前，QEMU-TRACER 对 64 位系统有更好的支持效果，未来我们可能会增强 QEMU-TRACER 对 32 位系统的支持。对于一些触发条件比较隐蔽的异常通信程序，SAQ 的分析能力不足，如果某些网络通信行为不在测试过程中表现出来，则无法进行有效的分析，后期可以尝试结合静态分析的方法，使得分析更加完备。

参 考 文 献

[1] Pandalabs report q2 2016[EB/OL]. http://resources.pandasecurity.com/newhome2016/micrositeAD/resources/Pandalabs/Pandalabs-2016-Q2-en.pdf.

[2] Quick Heal[EB/OL]. http://dlupdate.quickheal.com/seqrite/documents/en/threat-reports/quarterly_threat_report_q1_2016.pdf.

[3] LUK C K,COHN R,MUTH R,et al. Pin:building customized program analysis tools with dynamic instrumentation[J]. Acm Sigplan Notices,2005,40(6):190-200.

[4] SKALETSKY A,DEVOR T,CHACHMON N,et al. Dynamic program analysis of microsoft windows applications[C]// 2010 IEEE International Symposium on Performance Analysis of Systems & Software(ISPASS). 2010:2-12.

[5] Strace[EB/OL]. http://linux.die.net/man/1/strace.

[6] JACOB B,LARSON P,LEITAO B,et al. SystemTap:instrumenting the Linux kernel for analyzing performance and functional problems[M]// IBM Redbook. 2008.

[7] Global market share of mobile operating system[EB/OL]. https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems.

[8] Mcafee mobile thread report 2016[EB/OL]. http://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf.

[9] SCHUSTER F,HOLZ T. Towards reducing the attack surface of software backdoors[C]// 2013 ACM SIGSAC Conference on Computer & Communications Security. 2013:851-862.

[10] Linux/sshdoor.abackdooredssh daemon that steals passwords [EB/OL]. http://www.welivesecurity.com/2013/01/24/linux-sshdoor-a-backdoored-ssh-daemon-that-steals-passwords.

[11] Source Insight[EB/OL]. http://www.sourceinsight.com/.

[12] Understand[EB/OL]. http://scitools.com.

[13] Egypt-create call graph from gccrtldump[EB/OL]. http://www.gson.org/egypt/egypt.html.

[14] SUN W Z,DU X Y,XIANG Y,et al. CG-RTL:a RTL-based Function Call Graph Generator[J]. Journal of Chinese Computer Systems,2014,35(3):555-559.(in Chinese)
孙卫真，杜香燕，向勇，等. 基于 RTL 的函数调用图生成工具 CG-RTL[J]. 小型微型计算机系统，2014，35(3)：555-559.

[15] BUSH W R,PINCUS J D,SIELAFF D J. A static analyzer for finding dynamic programming errors[J]. Software-Practice and Experience,2000,30(7):775-802.

[16] MAGGI F,MATTEUCCI M,ZANERO S. Detecting intrusions through system call sequence and argument analysis[J]. IEEE Transactions on Dependable and Secure Computing,2010,7(4):381-395.

[17] ASMITHA K,VINOD P. Linux malware detection using non-parametric statistical methods[C]// 2014 International Conference on Advances in Computing,Communications and Informa-tics(ICACCI). 2014:356-361.

[18] SHAHZAD F,SHAHZAD M,FAROOQ M. In-execution dynamic malware analysis and detection by mining information in process control blocks of Linux OS[J]. Information Sciences,2013,231:45-63.

[19] XIANG Y,CAO R D,MAO Y H. QEMU-based Dynamic Function Call Tracing[J]. Journal of Computer Research and Development,2017,54(7):1569-1576.(in Chinese)
向勇，曹睿东，毛英明. 基于 QEMU 的动态函数调用跟踪[J]. 计算机研究与发展，2017，54(7)：1569-1576.

[20] CHIPOUNOV V,KUZNETSOV V,CANDEA G. S2E:A platform for in-vivo multi-path analysis of software systems[J]. Acm Sigplan Notices,2011,46(3):265-278.

[21] SARACINO A,MARTINELLI F,ALBORETO G,et al. DataSluice:Fine-grained traffic control for Android application[C]// 2016 IEEE Symposium on Computers and Communication (ISCC). 2016:702-709.

[22] RUBIN J,GORDON M I,NGUYEN N,et al. Covert communication in mobile applications(t)[C]// 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2015:647-657.