软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(8):2229−2267 [doi: 10.13328/j.cnki.jos.005767]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

恶意代码演化与溯源技术研究*

宋文纳[1,2], 彭国军[1,2], 傅建明[1,2], 张焕国[1,2], 陈施旅[1,2]

[1](空天信息安全与可信计算教育部重点实验室(武汉大学),湖北 武汉 430072)

[2](武汉大学 国家网络安全学院,湖北 武汉 430072)

通讯作者: 彭国军, E-mail: guojpeng@whu.edu.cn

摘 要: 恶意代码溯源是指通过分析恶意代码生成、传播的规律以及恶意代码之间衍生的关联性,基于目标恶意代码的特性实现对恶意代码源头的追踪.通过溯源可快速定位攻击来源或者攻击者,对攻击者产生一定的震慑打击作用,具有遏制黑客攻击、完善网络安全保障体系的重要作用和价值.近年来,网络安全形势愈加严峻,归类总结了学术界和产业界在恶意代码溯源领域的研究工作,首先揭示了恶意代码的编码特性以及演化特性,并分析这些特性与溯源的关系;然后,分别从学术界和产业界对恶意代码的溯源技术和研究进行梳理,同时对每个溯源阶段的作用以及影响程度进行了讨论,并对目前恶意代码的溯源对抗手段进行分析;最后讨论了恶意代码溯源技术面对的挑战和未来的发展趋势.

关键词: 恶意代码溯源;演化;对抗;家族聚类;恶意代码检测

中图法分类号: TP311

中文引用格式: 宋文纳,彭国军,傅建明,张焕国,陈施旅.恶意代码演化与溯源技术研究.软件学报,2019,30(8):2229−2267. http://www. jos.org.cn/1000-9825/5767.htm

英文引用格式: Song WN, Peng GJ, Fu JM, Zhang HG, Chen SL. Research on malicious code evolution and traceability technology. Ruan Jian Xue Bao/Journal of Software, 2019,30(8):2229−2267 (in Chinese). http://www.jos.org.cn/1000-9825/5767.htm

Research on Malicious Code Evolution and Traceability Technology

SONG Wen-Na[1,2], PENG Guo-Jun[1,2], FU Jian-Ming[1,2], ZHANG Huan-Guo[1,2], CHEN Shi-Lü[1,2]

[1](Key Laboratory of Aerospace Information Security and Trust Computing (Wuhan University), Ministry of Education, Wuhan 430072, China)

[2](School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China)

Abstract: The traceability of malicious code refers to the tracking of the source of malicious code based on the characteristics of the target malicious code by analyzing the rules of the generation and propagation of malicious code and the correlation of derivation among malicious codes. The tracking technology can quickly locate the source of attacker or attacker, which would make a deterrent effect to the attacker. It has the important role and value in curbing deterring hacking attacks and improving the network security system. In recent years, the network security situation has become more and more severe. This study categorizes and summarizes research work in the academic and malicious code traceability field. First, the coding characteristics and evolutionary characteristics of malicious codes are revealled, and the relationship between these characteristics and traceability is analyzed. Then, the traceability techniques of malicious code are reviewed and the role and impact of each traceability phase are discussed in the industry and academia. Also analyzed is the level

* 基金项目: NSFC-通用技术基础研究联合基金(U1636107); 国家自然科学基金(61373168, 61332019)

Foundation item: Program of Joint Funds of NSFC-Universal Fundamental Research Fund (U1636107); National Natural Science Foundation of China (61373168, 61332019)

本文由“面向自主安全可控的可信计算”专题特约编辑贾春福教授推荐.

收稿时间: 2018-05-31; 修改时间: 2018-09-21; 采用时间: 2018-12-13; jos 在线出版时间: 2019-01-21

CNKI 网络优先出版: 2019-01-22 13:48:19, http://kns.cnki.net/kcms/detail/11.2560.TP.20190122.1348.003.html

of confrontation of traceability of current malicious code. Finally, the challenges and the future development trend faced by malicious code tracing technology are discussed.

Key words: malicious code traceability; evolution; confrontation; family clustering; malicious code detection

恶意代码溯源是指通过分析恶意代码生成、传播的规律以及恶意代码之间衍生的关联性,基于目标恶意代码的特性实现对恶意代码源头的追踪.随着互联网的蓬勃发展,恶意代码已经成为威胁互联网安全的关键因素之一.2017 年,AV-TEST 安全报告[1]指出,AV-TEST 系统检测恶意代码规模已经超过 6.4 亿,其中,Windows 和 Android 平台的恶意软件规模极为显著,与前一年相比,Android 设备端恶意软件数量翻了一番,恶意软件的数量变化如图 1[1]和图 2[1]所示.

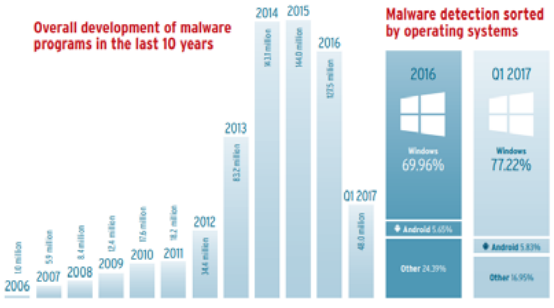


Fig.1 Over development of new malware programs in the last 10 years[1]

图 1 最近 10 年新恶意软件的发展[1]

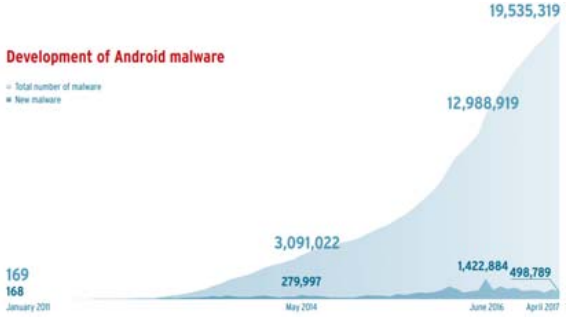


Fig.2 Development of Android malware[1]

图 2 Android 恶意软件的发展[1]

在与恶意样本的的对抗过程中,恶意软件分析和检测技术也在不断发展.基于静态分析的检测[2,3]、基于动态分析的检测[4−6]以及基于机器学习的检测[7−11]等技术不断涌现:基于静态分析的检测对非混淆样本更为准确;而基于动态分析的检测在检测混淆恶意软件方面表现更为出色;基于机器学习的检测[12−16]是通过对大规模恶意样本进行特征提取(如 API(application programming interface)、CFG(control flow graph)、关键字符串值等),然后采用机器学习算法(例如分类或聚类)训练样本,以构建模型判断软件的恶意特性.这为安全研究人员提供了良好的辅助功能,有效地提高了大规模恶意软件的检测速度.

虽然恶意软件检测技术的广泛应用对恶意代码攻击起到了一定抵抗作用,但是震慑力依然不足.

- 一方面,恶意软件检测技术有限,恶意作者可利用免杀技术构建变体绕过恶意软件检测.例如 2017 年,安天移动安全联合猎豹移动安全实验室捕获一例使用 MonoDroid 框架开发的移动端 C#病毒,该病毒将逻辑代码编译成 DLL 文件,进而逃避恶意代码的常规检测[17].2017 年 5 月份爆发的 WannaCry 样本与 2017 年 3 月份 Wcry 样本是同源样本,该变种利用微软 SMB 漏洞以及 DOUBLEPULSAR 后门实施攻击,绕过了包含 360 在内的多个安全检测工具,英国、法国、西班牙、韩国、俄罗斯及中国等多个国家遭受了严重的经济损失[18].
- 另一方面,恶意代码检测技术侧重于对恶意代码的发现和防范,尽管利用该技术可以检测到大多数恶意代码攻击,但是不能提供对恶意代码来源的有效追踪,因此不能从根源上遏制恶意代码的泛滥.

FireEye[19]面向多个组织进行了针对网络安全应急响应速度的调查,其报告指出,在复杂的新型恶意软件和高级先进的持续威胁(APT)环境下,只有 20%的组织认为其事件响应计划“非常有效”,而他们最大的安全差距在于是否能够检测和遏制 APT 类恶意软件.这说明安全组织对威胁事件的响应计划、人员和工具还不能跟上新的威胁.不过,现有恶意代码检测中的部分工作也可用于恶意代码的溯源研究.例如,恶意代码检测技术中的特征分析可为溯源特征的提取提供借鉴,因为不管是恶意代码的溯源还是检测,在特征提取阶段,均会考虑对代码中包含其典型恶意性的关键代码或数据片段进行分析.

为了进一步震慑黑客组织与网络犯罪活动,目前学术界和产业界均展开了恶意代码溯源分析与研究工作.其基本思路是:利用恶意样本间的同源关系发现溯源痕迹,并根据它们出现的前后关系判定变体来源.恶意代码同源性分析,其目的是判断不同的恶意代码是否源自同一套恶意代码或是否由同一个作者、团队编写,其是否具有内在关联性、相似性.从溯源目标上来看,可分为恶意代码家族溯源及作者溯源.家族变体是已有恶意代码在不断的对抗或功能进化中生成的新型恶意代码[20],针对变体的家族溯源是通过提取其特征数据及代码片段,分析它们与已知样本的同源关系,进而推测可疑恶意样本的家族.例如,Kinable 等人提取恶意代码的系统调用图,采用图匹配的方式比较恶意代码的相似性,识别出同源样本,进行家族分类[21].恶意代码作者溯源即通过分析和提取恶意代码的相关特征,定位出恶意代码作者特征,揭示出样本间的同源关系,进而溯源到已知的作者或组织.例如,文献[22]通过分析 Stuxnet 与 Duqu 所用的驱动文件在编译平台、时间、代码等方面的同源关系,实现了对它们作者的溯源.Kaspersky 实验室通过深入分析 Stuxnet 与 Flame 这两款恶意软件发现,2009 版的 Stuxnet 中的 Resource 207 模板与 Flame 中一个插件模块 mssecmgr.ocx 几乎完全相同,得出 Flame 与 Stuxnet 的开发人员有过早期合作等结论[23].2015 年,针对中国的某 APT 攻击采用了至少 4 种不同的程序形态、不同编码风格和不同攻击原理的木马程序,潜伏 3 年之久,最终,360 天眼利用多维度的“大数据”分析技术进行同源性分析,进而溯源到“海莲花”黑客组织[24].可见,发现样本间的同源关系对于恶意代码家族和作者的溯源,甚至对攻击组织的溯源以及攻击场景还原、攻击防范等均具有重要意义.

本文主要围绕恶意代码的家族、作者等溯源工作进行进展研究.首先介绍恶意代码的编码特征和演化特性,然后从学术界和产业界两个方面梳理现有的研究工作.归纳并基于学术界所共有的实现环节,分析各个环节面临的关键问题,以及解决这些问题的研究思路;然后对产业界的溯源机理,所能解决的关键问题进行分析,并对学术界和产业界在溯源分析方法方面的区别和联系进行了总结.最后,对已有的恶意代码溯源方法中存在的挑战进行分析,并对未来可进行的研究方向进行了展望.

1 恶意代码的生成过程与演化特性分析

恶意代码是指在一定环境下执行的对计算机系统或网络系统机密性、完整性、可用性产生威胁,具有恶意企图的代码序列[25].恶意代码的编写流程通常如图 3 所示.

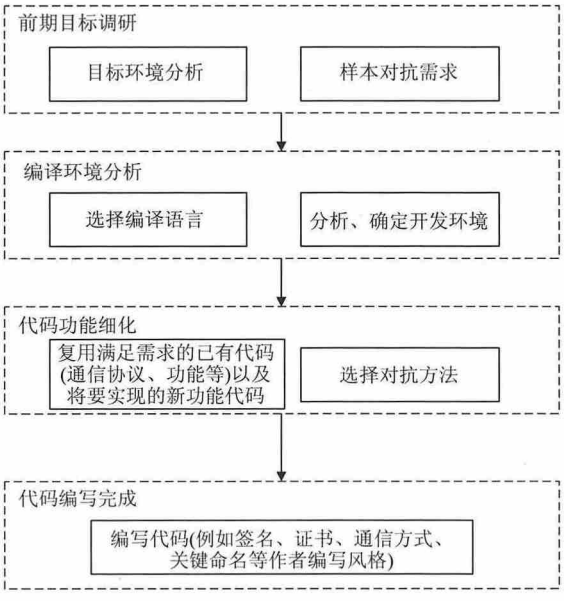


Fig.3 Malicious code writing process

图 3 恶意代码编写流程

恶意代码与正常应用程序的编写流程基本相似.

(1) 前期目标调研:制定自己代码功能需求,分析目标环境,主要获取目标环境信息.

(2) 编译环境选择:根据前期目标调研阶段的分析结果,确定恶意代码的开发语言与集成开发环境(通常包括编辑器、编译器、调试器和图形用户界面工具等)等,为代码的编写提供开发调试环境条件.

(3) 代码细化功能分析:结合第 1 阶段和第 2 阶段的代码功能的分析,进一步细化代码行为功能和对抗功能.全新编写代码或者从已有的样本源码库中选择满足自己需求功能的代码片段,并作为创建新程序的复用代码片段.选择合适混淆和对抗环境识别分析的方法,用于编写能够逃避静态和动态分析与检测的恶意代码.

(4) 代码编写:作者按照自己的代码编写风格与习惯进行编码实现,并选择合适的数字证书(APP 开发)进行签名,完成恶意代码的全部实现.

恶意代码的作者编写风格是作者在长期的编写过程中形成的不易改变的代码风格,利用代码编写风格的相似性可以实现对作者的溯源;另外,代码开发环境(如 IDE、特殊的代码路径、非默认编译参数等)也可能成为作者溯源特征.

恶意代码遵循正常软件开发流程,但实现的功能往往是破坏计算机系统或者窃取用户隐私等.为了使代码中敏感操作能够逃避检测,恶意作者往往会采用与正常程序不一样的编码方式,从而表现出作为软件与身俱来的软件特性(如复用性、固有缺陷性等)以及作为恶意软件所特有的多种恶意特性(如代码敏感性、对抗性等).此外,在进行恶意代码编写时,功能代码和特有对抗代码的复用可能导致前后衍生软件的相似性.因此,恶意代码的编码特性可为进行溯源分析提供有效线索.下面将具体从恶意代码个体的编码特性以及恶意代码作者和家族的相似性角度进行阐述.

1.1 恶意代码个体的编码特性

恶意代码个体的编码特性指作者在编写恶意代码过程中所呈现出来的代码编写特性,为恶意代码文件的溯源分析提供了良好理论和技术支撑,有助于溯源特征的提取.本文从代码复用性、代码对抗性、代码敏感性、代码固有缺陷性等方面分析恶意代码的编码特性.

1.1.1 代码复用性

复用性是指恶意代码作者在进行代码复用行为后形成的一种代码衍生特性,该特性推动了恶意代码的快速生成.复用行为是恶意代码作者采用的一种,将已有恶意代码中满足自己功能需求的代码片段提取出来,不修改或进行稍许修改并应用于创建新的恶意代码的行为.2014 年,Symantec 报告[26]指出,大部分恶意软件都是已存在恶意软件的变种而不是重新创建的新型恶意软件.2018 年,Symantec 发布的《2017 年度回顾:移动威胁情报报告》[27]指出,2017 年移动平台恶意软件变种的数量增长了 54%.可见,代码复用在移动恶意软件开发中具有普遍性,这在当前多个知名的 PC 端恶意软件中也得到充分体现.比如,CrySyS 实验室发现 Duqu 与 Stuxnet 的某些 DLL 文件具有多个相似的导出函数,driver 文件中也存在大量相似函数[28];Kaspersky 实验室发现 Flame 与 Gauss 的 C&C 服务功能函数、字符串的初始化函数、字符串解密函数等相似[29].另外,恶意软件 Cloud Atlas 与 RedOctober[30]的压缩算法实现函数相似.

1.1.2 代码固有缺陷性

固有缺陷性主要指恶意代码的编写缺陷.编写缺陷是指恶意代码作者因为个人水平或其他原因,在进行某些功能的编码时,有时候会产生一些编写或逻辑上的错误,而这种错误是在其编写类似代码时每次都会犯,这就形成了作者的固有缺陷.如果多个恶意软件均存在类似缺陷,则可能为同一作者所为.

1.1.3 代码对抗性

对抗性是指恶意代码具有的可以遏制逆向分析以及绕过杀软、穿透代理、防火墙以及对抗 IDS 等防护手段.根据对抗类别的不同,恶意代码的对抗分为基于静态分析与检测的对抗、基于动态分析与检测的对抗以及基于机器学习分析与检测的对抗等.恶意软件的对抗性使得恶意代码在系统设备中长期潜伏成为可能,这对系统资源和用户数据造成了严重的潜在威胁.恶意代码对抗类别及常见方法见表 1.

Table 1 Confrontation category and method of Android malware

表 1 Android 恶意软件的对抗类别及方法

对抗类别	对抗目的	常见对抗方法
基于静态分析与检测的对抗	防止被静态反汇编或者即使可以被反汇编但是不能对其提取有效特征、签名等,进而无法对恶意代码进行检测	1. 花指令;2. 简单加密;3. 多态与变形; 4. 代码混淆;5. 对抗杀软静态启发式检测
基于动态分析与检测的对抗	恶意代码在执行的时候所展现出来的能够逃避安全软件、调试器、虚拟机及模拟器等分析的特性,从而绕过动态检测	1. 检测调试环境;2. 检测调试行为;3. 检测沙箱环境; 4. 虚拟机环境检测;5. 对抗动态启发式查杀、主动防御; 6. 漏洞利用
对抗基于机器学习分析与检测方法	构建对抗样本绕过基于机器学习的检测系统	黑客利用机器学习的内部结构漏洞或者分析训练数据,训练出可以逃避分类模型的样本

表 1 中列出的常见方法描述如下.

1) 基于静态分析与检测的对抗

花指令指精心设计代码逻辑或在指令间插入定义设计的数据,干扰反汇编器给出错误反汇编结果;简单加密指对病毒主体代码采用不同密钥加密,导致不同个体文件数据差异大,从而导致特征值提取困难[31].多态和变形是在加密基础上,在保持代码等价功能的前提下,对解密逻辑及原始病毒主体代码进行变换.保持功能等价的代码变换技术[32]包括插入无关垃圾代码、指令扩展或收缩、改变无关指令执行顺序、插入条件跳转指令、寄存器重新分配、变量重命名等;代码混淆指通过采用字符串混淆[33,34]、控制流混淆[35]等混淆技术对代码自身做出变换,常见的代码变换工具包括 DexGuard[36]、ProGuard[37]、AMAD[38]、DroidChameleon[39]、代码隐藏(恶意组件隐藏在资源文件[40−43]中、Manifest 文件欺骗)、动态加载、加壳[44,45],如 UPX,ASPACK,ASPROTECT,VMProtect 等;静态启发式检测主要是通过静态启发式扫描分析文件代码的逻辑结构是否含有恶意程序特征[46].常见的对抗杀软静态启发式检测的技术包括多节病毒、加密宿主文件头的前置病毒、入口点隐藏技术、在代码中选择随机入口点、重新利用编译器对齐区域、不适用 API 字符串重命名已经存在的节等.

2) 基于动态分析与检测的对抗

(1) 检测调试环境:主要指在检测到调试环境的情况下终止敏感操作,包括从文件特征、进程名、进程数据特征、加载的特定模块、调试器窗口、调试器具有的特殊权限等方面进行检测.例如,利用调试器一般采用 DBGHELP 库来装载调试符号,因此根据进程是否加载了 DBGHELP.DLL,来判断是否存在调试器环境;通过 *FindWindow*(·)等函数查看是否包含调试器标题和类名的窗口:如果存在,则很可能有调试器存在;调试器进程对其他进程调试的时候,需要拥有 SeDebugPrivilege 权限,普通进程没有该权限,因此通过打开 CSRSS.EXE 进程验证自身进程是否具备 SeDebugPrivilege 权限,来确定进程是否被调试.

(2) 检测调试行为:主要指在调试行为发生的情况下,表现出对抗调试或终止敏感操作的行为.对抗调试行为的方法包括基于调试特征检测的调试对抗、基于调试特征隐藏关键代码的调试对抗.例如,利用 PEB 结构 BeingDebug 标志,在程序处于调试状态时为非 0,将其作为特性进行反调试;利用异常中断 int3 指令常被设置为软件断点的特性,在代码中置入 int3 指令,当程序未被调试时,将进入异常处理继续执行,但是程序处于调试阶段时,int3 被当成调试器自己的断点,而不会进入异常处理程序,通过将核心代码写入异常处理过程,能够避开调试器的执行调用.

(3) 检测沙箱环境:主要指利用硬件序列号、固件版本和其他操作系统配置作为沙箱指纹,通过检测所处环境的沙箱系统和管理程序的每个工件来逃避沙箱系统的分析与检测[47,48].常见方法包括:访问特定注册表项(例如,HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Disk\Enum),然后解析子项的值,查看“vmware”“qemu”“xen”等子字符串的存在情况,以判定沙箱的存在;利用内核数量检测沙箱环境的存在,这是因为正常环境中内核处理器是多个,但是在沙箱环境中通常是单核处理器;通过检测设备信息,并将该信息与沙箱中已知的值进行比较检测沙箱环境;利用模块名称检测沙箱.例如,在模块名称上调用 *GetModuleHandleA*(·),如果返回 *Null*,则表示模块已加载,沙箱在进程中注入模

块以记录其执行活动,那么通过 *FreeLibrary*(·)来卸载模块,使得沙箱将不能记录任何执行痕迹[49].

(4) 虚拟机环境检测:基于虚拟机环境检测的对抗分析方法主要包括 3 类:语义攻击(CPU 语义攻击)、基于时间的攻击和字符串攻击[50].例如,文献[51,52]均使用CPU语义信息来检测虚拟机的存在;文献[53]利用基于时间的技术来确定管理程序自省操作的存在,来识别虚拟机的存在;Thanasis 等人提出的基于 QEMU 的模拟器的不完全仿真特点识别模拟器的存在[54];此外,枚举进程也可以识别虚拟机的存在,例如,通过使用 *Process*32*First*(·)/*Process*32*Next*(·)查找与虚拟机相关的 vmsrvc.exe 等进程名称:如果存在,执行非恶意操作,逃避安全检测.

(5) 对抗动态启发式查杀、主动防御:常见的方法包括:调用底层的未拦截 API 接口完成上层 API 功能;利用受信任进程完成对目标模块的加载,对主动防御拥有比较好的免杀效果;将多个行为在分离在多个进程中实现,将能成功绕过针对目标进程的行为进行综合分析的启发式查杀.例如,在进程 *A* 中完成文件释放,进程 *B* 中完成提权,进程 *C* 完成安装.

(6) 漏洞利用:漏洞利用[55]主要指利用程序中的某些漏洞(如缓冲区溢出漏洞[55]),得到计算机的控制权,进而逃避安全系统的分析与检测.例如,2017 年 8 月,FireEye 发布报告指出,APT28 使用 EternalBlue 漏洞利用工具和开源工具 Responder 进行横向传播[56].2017 年 4 月,白象组织利用 Office 漏洞,该漏洞利用 Office OLE 对象链接技术,将包含的恶意链接对象(HTA 文件)嵌在文档中,通过构造响应头中 content-type 的字段信息,最后调用 mshta.exe 将下载到的 HTA 文件执行[57].而目前漏洞利用的高级表现形式是组合漏洞的利用,突破了单一漏洞执行过程中被安全系统分析与检测到,而无法继续运行的问题.例如,2017 年的 FinSpy[58]利用 CVE-2017-0199,CVE-2017-8759,CVE-2017-11292 等多个漏洞来投递 FinSpy,该病毒的复杂性强,对其检测困难.

3) 对抗基于机器学习分析与检测方法

对抗基于机器学习分析与检测:主要是通过对抗性数据操纵恶意软件逃避模型检测[59].攻击者对训练数据进行变形,使其接近训练数据集中良性实例,从而逃避目标分类器分类.目前,学术界的对抗研究是在攻击者对机器学习模型内部(特征空间、分类算法等)[60]、训练数据集、分配给输入样本的分类分数[61]等至少一种了解的情况下,实施的逃避方案.其中,攻击者利用分配给输入样本的分类分数进行的方案,在实际的应用场景实现艰难,因为现实部署的机器学习模型只会暴漏给攻击者最终决策(拒绝、接受等).文献[62]在基于攻击者仅知道机器学习模型的最终决策的情况下,提出一种使用黑盒子变形器操作恶意样本逃避的方法 EvadeHC,使 PDF 恶意软件有效地逃避了检测器.这表明:基于机器学习模型的恶意代码分析与检测系统并非完全可靠,仍需兼并人工分析.

综上,恶意代码动态和静态分析与检测的对抗方法增强了代码被解读的困难性,加强了代码痕迹被捕获的难度,使得恶意代码具备了一定的自我保护能力.然而在实际操作中,为了平衡恶意代码的运行效率和功能,往往不会在恶意代码中加入非常全面的对抗技术.例如文献[63]中提出:卡巴斯基在 2015 年 7 月发现 Duqu 的新变种 Duqu2.0,利用了 0day 漏洞,该漏洞无法通过静态分析内容检测到,但是将该恶意软件放在沙箱环境中,利用强大的启发式引擎进行检测,可以拦截到行为.因此,在实际的恶意代码分析与检测中,动静结合使用,往往比采用单一的方法能发现更多的恶意代码.基于机器学习的分析与检测方法在检测大规模恶意代码方面效率大大提升,但是上述分析表明,该方法的自身缺陷可能为黑客提供对抗条件.由此可见,在面对大规模恶意软件的分析与检测方面,人工分析与基于机器学习的分析与检测结合使用,才能更加准确地检测到更多恶意代码.

1.1.4 代码敏感性

代码的敏感性是指恶意代码在进行行为隐蔽触发和敏感资源访问的所表现出来的特性,常见的恶意代码关键示例敏感点主要包括触发条件、系统(API)调用、代码结构、常量(关键字符串)等.图 4 为恶意敏感操作执行代码片段示例,图 5 为触发分支条件代码片段示例.

图 4 中第 1 行画线部分表示敏感函数入口点;第 5 行表示敏感函数触发分支条件;第 6 行和第 7 行表示正常操作代码片段;第 9 行和第 10 行表示恶意代码操作片段;第 7 行、第 10 行、第 15 行、第 16 行画线部分表

示调用的敏感 API.

```
1.  protected void onCreate(Bundle savedInstanceState){
2.    super.onCreate(saveInstanceState);
3.    setContentView(R.layout.activity_main);
4.    Date date=new Date(·);
5.    if (date.getHours(·)>3||data.getHours(·)<5){
6.    SmsManager smsManager=SmsManager.getDefault(·);
7.    smsManager.sendTextMessage("6667", null,getIMEI(this),null,null);
8.    }else{
9.    SmsManager smsManager=SmsManager.getDefault(·);
10.   smsManager.sendTextMessage("6666",null,this.getString(R.string.Greetings),null,null);
11.   }
12.
13.   public String getIMEI(Context mcontext){
14.   TelephonyManager TelephonyMgr=(TelephonyManager)
15.   mcontext.getApplicationContext(·).getSystemService(TELEPHONY SERVICE);
16.   String szImei=TelephonyMgr.getDeviceId(·);
17.   return szImei;
18.   }
19. }
```

Fig.4 Malicious sensitive operation execution code snippet example

图 4 恶意敏感操作执行代码片段示例

```
1.  public void hso(·){
2.    boolean containAV=false;
3.    List⟨PackageInfo⟩packagelist=getPackageManager(·).getInstalledPackages(0)
4.    for (PackageInfo package:packagelist){
5.    if (package.packagename.contains("com.antivirus")){
6.    containAV=true;
7.    };
8.    StringBuilder params=new StringBulider(·);
9.    if (!containAV){
10.   params.append("&no=").append(utils.getPhoneNumber(·));
11.   params.append("&lat=").append(utils.getLatitude(·));
12.   }else{
13.   params.append("&no=").append("00000000000");
14.   params.append("&lat=").append("0.00");
15.
16.   }
17. }
```

Fig.5 Example of a sensitive code snippet based on a triggering branch[64]

图 5 基于触发分支的敏感性代码片段示例[64]

图 5 中第 5 行~第 7 行用于判断杀软是否存在;第 9 行表示触发敏感操作的分支条件;第 10 行、第 11 行是正常操作代码片段;第 13 行、第 14 行是恶意操作代码片段.

下面结合图 4、图 5 中代码片段示例阐述代码敏感性.

1) 触发条件

触发条件指触发恶意敏感函数执行操作的条件,从触发条件的代码位置角度分为基于敏感函数入口点的触发和基于敏感操作分支的触发.

(1) 前者是指直接或间接触发恶意函数调用的程序入口点[65],从用户角度来看有两种入口点:用户界面和后台调用.根据文献[66,67]中实证研究,敏感 API 操作与函数入口点之间如果没有与 UI 相关的函数调用即采用后台回调,则所属样本通常具有恶意性.例如,当一个用户交互式 API 被悄悄调用时,恶意软件便成功实施该恶意行为[68].如图 4 所示,*SetTextMessage* 是基于回调方式将设备号发送出去.

(2) 触发分支条件是指能够触发恶意敏感函数执行的代码分支[69].该分支具有与正常程序分支不一样的特点,文献[64]对触发隐藏敏感操作的分支进行了详细的描述.该分支条件能够使恶意活动尽可能地逃避安全软件的检测,如图 5 所示,为了隐藏恶意活动,作者将恶意活动隐藏在第 13 行、第 14 行所表

示的分支路径中,而另一分支完全执行正常活动,且隐藏活动与另一分支及分支条件之间没有共享任何数据与资源,致使安全检测系统在语义层面的追踪也变得困难.

图 4 示例中,凌晨 3 点~5 点时间来确定何时执行恶意活动(例如窃取私人数据),然而正常软件却很少使用时间作为活动执行的条件输入.上述触发条件的触发形式比较多样化,比如还包括网络指令触发(如远控木马)、环境触发(如 GPS 位置变化)等.

2) 系统(API 等)调用

系统调用[70−76]是应用程序与系统交互的接口,实现了代码的功能,表达了关于应用程序行为的实质性语义.为了执行恶意行为,恶意软件需要调用敏感 API 函数来实现,但是仅靠单一的 API 调用无法判定所属代码的恶意性,因为 API 调用具有通用性,许多良性 App 也会调用这些 API(例如截屏、录音、定位等 API).然而,综合样本中多个 API 函数构成的序列[77−79],可以看出它存在哪些恶意行为甚至意图.例如,文献[80]从 vxheaven[81]中收集 271 092 个属于 137 个恶意软件家族的样本,相同恶意家族的样本因为共享相同的行为所以会调用更多相同的 API 集合,发现至少 90%样本中含有 15 个相同 API,因此,API 集合可以作为恶意性判定的一个特征.图 4 展示的恶意性操作代码片段中,API 序列为 *getDeviceId*(·),*sendTextMessage*(“6667”,*null*,*getIMEI*(*this*),*null*,*null*),设备号属于用户隐私信息,该序列表明这是将设备号发送出去的操作,存在泄密可能,展现了代码恶意性操作.

3) 代码结构

代码结构[82−84]主要指函数的逻辑结构,可以表达程序的语义信息,是一种细粒度的匹配特征.常见的代码结构表达方式包括如函数调用图 FCG(function call graph)、控制流图 CFG、程序依赖图 PDG (program dependance graph)等.以 API 为原子的代码结构表现为 FCG,该结构展现了函数的调用逻辑.在恶意代码的家族分类中,FCG 相比函数调用组成的序列集更能表达程序的原始信息,因此研究人员通常采用 FCG 作为家族特征,用于恶意代码家族的识别;CFG 是以指令为代码单元,能够从触发条件、API 调用、方法调用、结果返回等方面细粒度描述恶意程序的代码逻辑,较全面地覆盖代码所涉及的执行流程;PDG 是一种基于数据流依赖的代码结构,相比前两种语义信息更加丰富,能有效地发现恶意程序的污点传播路径,进一步准确定位恶意程序的执行范围.

4) 常量

常量通过揭示关键参数的值和细粒度的 API 语义来传达语义信息[85],例如在图 4 中,*sendTextMessage*(·)函数以一个名为 *PremiumRate* 的电话号码常量作为参数,比通过 *getText*(·)从用户输入接收电话号码的相同 API 的调用具有更可疑的行为.因此,恶意代码在操作敏感数据的时候关于常量的使用至关重要.

恶意代码在上述特征上通常具有典型敏感性,这是作者编写恶意代码内容的主要特点,也是安全研究人员进行溯源的主要依据之一.

1.2 恶意代码家族和作者的编码相似性

编码相似性是指恶意代码在编码环境及特征上具有相似性,基于恶意代码的编码特性,同一作者或者同一家族的恶意代码在内容和结构上往往存在相似之处[86],这为恶意代码的溯源提供了线索.

1) 同家族恶意代码的编码相似性(功能相似)

恶意代码以功能行为划分家族,同家族恶意软件的代码和行为具有相似甚至相同部分[87].作者为了快速构建恶意代码,复用已有恶意代码生成变体,使得同家族恶意代码中大部分代码及资源保持不变[88].而这些特性在编译后文件中表现为代码片段的相似性,其相似性主要集中在执行敏感操作的代码元素中,例如系统调用、关键字符串(如权限、重打包名字)和代码结构(逻辑结构)等.

- 系统调用:同家族恶意代码为了实施相同的功能行为,往往调用相同或相似的系统敏感函数 API.
- 关键字符串:主要指的是硬编码字符串,同家族恶意代码为实现特定元素相关的行为,会在代码中采用关键字符串.
- 代码结构:同家族恶意代码在实施相同的功能行为时,会执行相似代码流程,因此其代码结构相似.

利用上述特性识别恶意代码变体,并根据已知恶意代码家族揭示变体家族.例如,文献[87]通过多个签名条目(字符串、方法名、方法体等)评估不同样本间的相似性,进而归类同家族样本,使得检测变体成为可能;文献

[89]利用数据挖掘的方法分析 Android 恶意软件家族样本的代码结构,并运用静态分析提取与应用程序片段关联的代码结构 CFG,基于该代码结构计算家族指纹,使用该指纹与待检测的恶意软件进行相似度计算,可以识别出同家族恶意软件变体,验证了代码结构用于衡量家族变体间相似性的可行性;文献[90]为了识别 Android 恶意软件的多态变体,采用聚类算法和社区结构方法,从家族样本的敏感 API 调用子图中提取频繁子图作为恶意软件的家族行为特征,并利用该家族特征识别未知恶意软件,检测率达到 94.5%;钱雨村等人[91]提出采用函数调用、资源对象、控制流程图等元素构建五元组行为,并将其作为聚类特征,所提出的方法能够有效地对不同恶意代码及其变种进行同源性分析及判定.综上所述,同家族恶意代码中存在相似性的代码元素,这些元素可以作为家族识别的关键特征.

2) 同作者或团队的编码相似性(风格相似)

同作者或团队编写的恶意代码由于受拥有的相关领域知识、经验、编码工具等限制,使得编写的代码在内容和结构上具有相似性.这种相似性抽象刻画了作者的编码风格,即使通过混淆技术隐藏或不留下其身份信息,但是人的编码习惯不会因为创建非同恶意代码而产生强大的差异.基于人的编程习惯,研究人员可挖掘和利用代码风格追踪到相关的作者.

目前,源代码作者的溯源分析已经相对成熟,其主要用于软件取证和抄袭检测.Krsul 等人[92]将编程习惯主要分为编程布局、编程风格、编程结构这 3 类,见表 2.基于这些编程习惯,许多研究学者对程序的归属问题进行了广泛研究,提取的特征主要集中在字节码、*n*-gram 序列、结构化特征等方面.例如,MacDonell 等人[93]基于该编程习惯对恶意作者身份进行识别,识别率达到 81.1%;Lange 等人[94]采用布局、词汇以及遗传算法指标作为特征,对 20 个作者进行去匿名化,准确率达到 75%;Kothari 等人[95]将词汇标记与 *n*-gram 结合使用,对 12 个作者进行识别,准确率达到 76%;Burrows 等人[96]采用字节级的 *N*-gram 作为特征,对 10 个作者的程序进行区分,准确率达到 77%;Chen 等人提出通过比较程序数据流来识别作者身份的语义方法,实验结果表明,该方法具有健壮性,即使代码被有意修改,依然可以识别出来[97];Caliskan-Islam 等人[98]抽取抽象语法树作为特征,从 GCJ 数据集中识别出 1 600 个程序员,准确率达到 94%.

Table 2 Authors' programming habits[92]

表 2 作者编码习惯[92]

编码习惯	含义
编程布局	空格的使用、括号的放置等
编程风格	平均注释长度、平均变量长度
编程结构	平均函数长度、常用的数据结构等

目前,二进制代码作者的溯源相对源代码的溯源工作更加困难,这是因为代码编译导致源码中许多信息丢失,同时,编译器优化可能会改变程序的结构,进一步模糊作者编码风格.Alrabaee 等人[99]设计了 OBA2 实验,对 8 个不同作者创建的程序进行源码和二进制程序的溯源分析,准确率达到 77%,验证了源码中一些特征在编译之后仍存在于二进制代码形式中,可用于恶意代码作者身份的识别.目前,常见的二进制恶意代码的溯源作者的编码相似性主要从代码风格和代码结构相似性角度进行分析.

恶意软件的代码风格主要依赖于作者的编码习惯.乔延臣等人[100]通过分析恶意代码中模块的复用风格,构建代码模块的快速溯源机制,且具有较高的准确率和召回率.Rosenblumdent 等人[101]提取指令序列和控制流作为特征,去匿名化准确率达到 77%.Caliskan-Islam 等人[102]改善了该方法,分别从反汇编和反编译层面提取语义和语法特征来表示相关作者的编程风格,结合随机森林、SVM 等机器学习算法进行训练,针对 100 个作者的 900 个文件进行识别,准确率达到 96%.虽然该文的工作主要针对二进制文件而不是恶意文件的分析,但这依然能为二进制恶意代码的作者溯源提供借鉴.

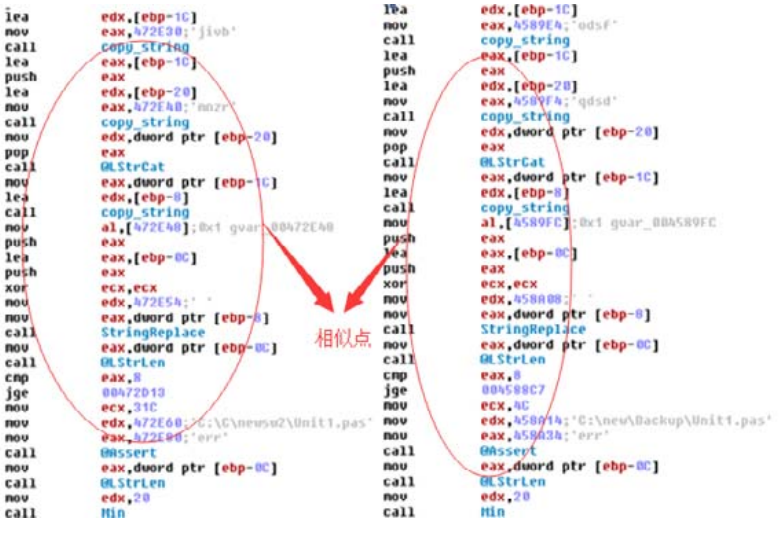
2018 年 2 月份,趋势科技[103]从代码结构着手分析,指出 Patchwork 和 Confucius 组织的 Delphi 恶意代码存在相似之处,如图 6~图 8 所示.

Fig.6 Decompiled Form structure of Confucius' sample[103]

图 6 Confucius 示例的反编译 Form 结构[103]

Fig.7 Decompiled Form structure of Patchwork's[103]

图 7 Patchwork 示例的反编译 Form 结构[103]



(a) Confucius 代码 (b) Patchwork 的代码

Fig.8 Confucius code and Patchwork's code[103]

图 8 Confucius 代码和 Patchwork 的代码[103]

图 6 的 Confucius 示例的反编译 Form 结构和图 7 的 Patchwork 示例的 Delphi 反编译 Form 结构中创建了相同的 TForm 对象,如圈内代码所示.图 8 展示了两个恶意代码中指令结构的相似性.此外,从代码中字符串域名的解析中发现,这两个团体主要攻击目标是东南亚,尤其是巴基斯坦,基于代码结构的相似性以及攻击方向的趋势可以判定它们来自同一组织.

由此可知,研究作者编写代码风格、代码结构相似性等对二进制恶意代码溯源具有重要意义.

1.3 恶意代码演化特性

恶意代码的个体编码特性、家族和作者的编码相似性,表明大部分恶意代码是对已有代码的修改,揭示了源码的演化趋势.源码的演化促使二进制恶意代码的变化,并为其提供逆向分析思路.移动端和 PC 端恶意软件在敏感行为执行上存在差异,例如移动端经常会悄悄发送短信等,但是 PC 端不会出现这样的操作,因此下面将分平台阐述二进制恶意代码的演化特性.

1.3.1 传统平台恶意代码的演化特性

了解恶意代码的演化,有助于更好地把握恶意代码的发展趋势,为发现新的恶意软件提供辅助信息,进而快速归属新恶意代码的家族或作者.表 3 从时间维度给出了恶意软件典型功能演变历程.

Table 3 Evolution and impact of PC malware

表 3 PC 端恶意软件的演化及影响

时间段	代表样本	软件类型	目的或影响	样本功能技术特征
1971	Creeper	一般程序	实验	能够在计算机之间移动
1974	Wabbit	一般程序	致使系统奔溃	具有自我复制功能
1982	Elk cloner	病毒	克隆	具有传播、自我复制功能
1986	PC-Write trojan	病毒	测试公司软件漏洞	可以感染 MS-DOS 计算机
1991	Michelangelo virus	病毒	在 3 月 6 日 擦除硬盘中信息	感染,擦除硬盘信息
1999	Melissa virus	病毒	群发邮件	感染计算机,获取其 outlook 地址薄,群发邮件
2000	ILOVEYOU worm	蠕虫	损害大型企业 和政府机构	以良性主题发送电子邮件传播, 感染 5000 万台计算机,蔓延至全球.
2001	Annna Kournikova virus	病毒	传播恶意软件, 进行破坏	将恶意软件隐藏在吸引人的照片种, 通过电子邮件发送进行传播
2003	SQL slammer worm	蠕虫	感染计算机实施破坏	利用漏洞,传播速度快,感染范围广
2005	Koobface virus	病毒	针对社交网络进行攻击	感染 PC 然后传播到社交网站
2008	ConFicker worm	蠕虫	造成自 Slammer 出现 以来最严重的破坏	感染并实施破坏
2010	Stuxnet worm	蠕虫	攻击伊朗的核电站, 包括其硬件与软件功能.	具有 APT 团队开发的复杂性和先进性, 具有密集的资源信息.
2011	Zeus trojan	木马	窃取银行信息.	影响范围广,通过浏览器按键记录和 表单抓取来窃取银行信息.
2014	Backoff	后门	盗取信用卡数据	破坏 POS 系统以窃取信用卡数据
2017	Wannacry ransomware	勒索软件	获取支付赎金	利用漏洞,将用户数据锁定,致使感染 150 多个 国家超过 23 万台 Windows 计算机系统瘫痪

恶意软件的演化历程[104]主要分为 3 个阶段.

- 阶段 1. 1971 年~1999 年的恶意软件主要以原始程序的形式出现,恶意软件功能单一,破坏程度小,无对抗行为.
- 阶段 2. 2000 年~2008 年,恶意软件的破坏性增强,恶意软件及其工具包数量急剧增长,借助网络感染速率加快,电子邮件类蠕虫、受损网站、SQL 注入攻击成为主流.
- 阶段 3. 2010 年之后,经济利益和国家利益的驱使下的恶意软件存在团队协作紧密、功能日趋复杂、可持续性强及对抗性强等特点.

上述演变历程中,一方面将恶意软件扩展到了不同平台,例如从早期的 PC 计算机到工控行业,但更重要的是恶意代码的功能的进化:恶意代码在前一阶段的个别特性,会在后面阶段中持续增强.促进这种进化因素主要是恶意代码功能的不断改进以及攻防对抗技术的不断博弈.这主要涉及恶意代码攻击行为和生成方式两个过程的演化,具体演化描述如下.

1) 恶意软件的攻击行为演化特性

与传统恶意软件的攻击行为相比,新一代恶意软件[105−107]的攻击行为具备高级性,如图 9 所示.

高级恶意软件攻击行为的定向性、持续性、隐蔽性和技术先进性的具体行为表现如下.

(1) 定向性[107]:攻击者利用 Web 服务、浏览器和操作系统中的漏洞,或使用社会工程技术发现目标用户,并使用户运行恶意代码来散播恶意软件.

(2) 隐蔽性:一旦进入系统,恶意代码隐藏(加密流量等)并禁用主机保护[108],达到正常运行的目的.

(3) 持续性[108]:安装完成后,恶意代码会调用命令并控制服务器以获取进一步的指令(例如窃取数据、感染其他机器并允许侦察等指令),从而持续性地控制目标机器.

(4) 技术先进性[109]:恶意软件作者使用混淆技术,如无效的代码插入、寄存器重新分配、子程序重新排序、指令替换、代码转换和代码集成等,以逃避传统防御措施(如防火墙、防病毒和网关等).

基于上述行为描述,说明目前的高级恶意软件在功能和防御策略方面更为先进,其造成的影响和损失也更为严重.

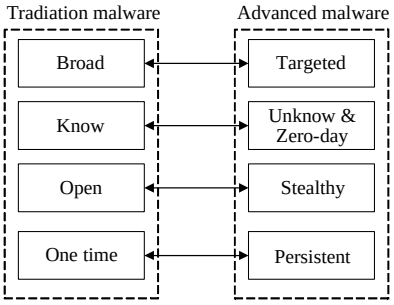


Fig.9 Attack behavior of traditional malware vs. advanced malware[107]

图 9 传统恶意软件 VS 高级恶意软件的攻击行为[107]

2) 恶意软件的生成演化特性

目前,新的恶意软件很少从头开始创建,而是采用自动生成工具、第三方库以及借用现有的恶意软件代码等[82]生成,许多恶意软件都是已存在恶意软件的变体[110].根据一项调查[111]指出,超过 98%的新恶意软件样本实际上是来自现有恶意软件家族的衍生产品(或变体).恶意代码经过修改或者自行演化等途径后,往往会形成数十种甚至更多的变种,使得变体数量迅速增长[112].

一个恶意代码变种与原来恶意代码形式上有所不同,但实现行为相似,那么这两个恶意代码称为同一个家族,新生成的恶意代码称为家族变体[113].当前的恶意代码变体在实现技术上大致可分为 2 大类:一类是共用基础技术(核心模块或理论),黑客通过重用基础模块实现恶意代码变种[114];另一类是针对现有防范和检测而研发的恶意代码混淆技术.恶意代码混淆技术按照其实现原理可分为 2 类[115]:一类是干扰反逆向(反汇编)的混淆,使反逆向不能够得到正确的分析结果,从而阻碍进一步机理分析;另一类是指令和控制流混淆,这类混淆技术通常采用加壳、垃圾代码插入、等价指令替换、寄存器重新分配及代码变换等方式改变恶意代码的语法特征,隐藏其内部调用逻辑关系,使得恶意代码逃避恶意软件的检测.

随着恶意代码家族功能和对抗策略的不断调整,新的恶意软件变种及其早期版本的运行时行为通常非常相似,但样本的演化可能会带来家族的进化,以适应新的计算机环境.例如,文献[116]中指出,蠕虫家族 Sobig 从版本 Sobig.A 到 Sobig.F,该蠕虫病毒只是增加或者减少一些特性,其行为近似相同;然而,蠕虫家族 Beagle worm 从版本 A 到版本 C 进行了不断的演化和升级,包括增加了后门、增加了阻碍本地安全机制的代码、增添了更好的在已有进程中隐藏蠕虫的机制等,其功能的复杂性提升,家族特性出现新的进化.目前,针对恶意软件生成演化特性的研究主要从家族进化角度进行分析.例如,文献[88]根据导出图的路径模式定义了一个恶意软件演化关系的框架,该方法的局限在于对恶意代码中源代码的定义不包括机器生成的代码,考虑到恶意代码通常是从现有恶意程序中自动生成的,具有限制性;文献[117]对恶意软件的系统进化模型进行研究,探索不同恶意样本数据集对演化模型分析结果的影响;研究人员[118]收集运行时执行指令、内存和寄存器修改等日志信息,构建家族演化树,研究变化样本的来源.

综上所述,传统平台恶意软件的行为演化描述了恶意软件在生成之后所表现的攻击特性.这种特性为研究恶意代码的安全防御系统提供技术支撑,进而加强安全防范,尤其是基于软件运行时行为的检测系统(如沙箱、云平台检测系统等),使之能够更可靠地检测大多数恶意软件及其变体.研究传统平台的恶意代码生成演化特性能够更加全面地把握恶意代码的发展趋势,进一步调整目前基于主机防御系统(如防病毒软件)的不足.恶意代码的攻击行为和生成特性作为恶意软件演化过程中的关键要素,对于构建恶意软件演化史具有重要意义:有助于促进对传统平台恶意软件的全面认识,明确未来恶意软件的发展方向,是恶意软件发展中一个需要不断跟进的领域.

1.3.2 移动平台恶意代码的演化特性

第一种智能手机病毒起源于 2004 年[119],被称为 Cabir,内容和功能均比较简单,使用传统的白名单即可以准

确地对其检测.随着时间的推移,由于恶意作者对目标的修改以及尝试逃避检测,使得构造出的恶意代码不断改变;同时,随着代码复用以及现成工具的广泛性,恶意代码的数量不断攀升.图 10 展现恶意家族样本自 2012 年到 2017 年每个季度的数量规模,表明恶意代码的家族数量整体呈上升趋势[120].

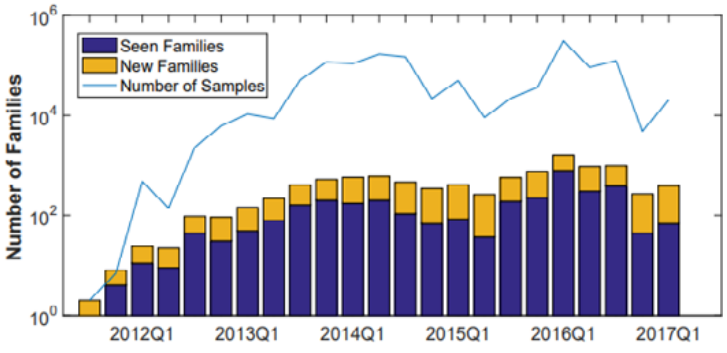


Fig.10 Quantitative scale of family samples in each quarter[120]

图 10 家族样本每个季度的数量规模[120]

下面具体分析移动平台恶意代码的演化特性,从行为演化和构成策略演化两个方面展开移动平台恶意代码的演化特性分析.

1) 恶意代码的行为演化

以文献[120–125]中对 Android 恶意代码演化分析为基础,描述移动端恶意代码的行为演化.恶意软件的行为主要指与特定攻击目标相关的行为[123].恶意代码的行为操作涉及的内容见表 4.

Table 4 Android malware behavior and goals

表 4 Android 恶意软件的行为及目标

行为名称	行为目标
隐私侵犯	对 Android Content Resolver 框架的查询、使用文件访问系统、访问用户位置等信息
数据窃取(exfiltration)	将网络的使用与隐私侵犯行为结合,致使个人信息的泄漏
欺诈	旨在从用户或用户使用的服务中获得直接利润,例如,恶意软件可能会通过 SMS 管理器发送额外费率消息,或者可能通过更改会员 ID 重定向收入来滥用广告网络
逃避	硬件序列号,固件版本和其他操作系统配置通常用于沙箱指纹,以避开动态分析
混淆	使用混淆和其他隐藏技术以逃避静态分析,Android 提供了在运行时动态加载代码的选项
漏洞利用(exploitation)	某些应用程序实现技术漏洞,并在安装后尝试获得 root 访问权限.大部分漏洞都是在 Native 代码中实现的,并使用与资源目录中的应用程序打包在一起的 bash 脚本触发

(1) 隐私侵犯:据文献[120]统计,2014 年之后,与隐私侵犯相关的 API 调用增加.例如 *HttpURLConnection.connect*(·)等,该调用结合 *ContentResolver.query*(·)可用于查询用户的隐私信息[124].这反映了移动平台恶意软件具备向隐私侵犯行为演化的趋势.

(2) 数据窃取:据文献[120]统计,在 2012 年到 2014 年,基于短信方式窃取数据的行为较多;而在 2014 年之后,主要以网络通道来窃取数据.窃取数据方式的改变表明:互联网技术的发展在为人们带来方便的同时,也给攻击者带来了更多的机会.由此表明:智能化时代的来临,数据保护愈发重要,这也为 APP 开发者提出了更高的安全需求.

(3) 欺诈:据 McAfee 指出,传统的基于高级短信的收费欺诈形式已经演变为僵尸网络广告欺诈、按次付费下载的分发诈骗以及勒索软件欺诈.目前的诈骗方式伪装手段更高级、迷惑性更强、可持续时间更久.2017 年,McAfee[122]发现目前绝大多数恶意软件都是 ad clicker Trojans,它在后台以欺诈手段操作移动广告,创建利润.

(4) 逃避:文献[120]指出,作用于更新攻击以及查询沙箱中某些硬件的相关 API 调用在 2017 年增加.此外,

文献[121]统计指出,自 2014 年之后,重命名、字符串加密是使用量增长最快的技术,动态加载、逃避动态分析等技术增加次之,而在 Native 中隐藏有效载荷关键代码的方法几乎处于平稳状态.由此可以得出,恶意软件行为具有向逃避动态分析发展的趋势.

(5) 漏洞利用:文献[120]中指出,*Process.killProcess*(·)、*File.Pid*(·)、*File.mkdir*(·)等与漏洞利用相关的 API 调用增加,表明该行为目前比较流行.文献[121]指出,目前,利用漏洞获取 root 权限已经不再流行;与此同时,利用漏洞获取 device-admin-privilege 权限成为主流.McAfee[122]指出,2017 年手机银行类木马软件(比如 Marcher malware)增加趋势明显,作为虚假更新或通过针对性的电子邮件或短信网络钓鱼提供.

结合上述演化行为,表明 Android 恶意代码向更复杂、更具有针对性、逃避性发展,这也对移动安全防御系统提出了更高的挑战.研究移动平台恶意代码的演化行为,可以帮助安全研究人员更好地研究移动可疑恶意软件,理解恶意代码的操作和意图,进而归属出家族变体,促进移动平台安全体系的稳固.

2) 恶意代码构成策略的演化

据赛门铁克统计:2012 年,每个移动恶意软件家族平均有 38 个变种;而在 2013 年,每个家族变种数量达到 58 个[125],展现了移动平台恶意软件变种规模逐年增加的趋势.与此同时,为了逃避移动平台防御系统的检测,自 Android 恶意软件出现以来,使用混淆技术构建恶意软件的数量也逐年增加.文献[126]指出,Google Play 中不足 25%合法应用程序被混淆[127],90%的恶意代码都使用了混淆技术.恶意软件在演化过程中继承了静态分析的局限性,根据衡量恶意软件行为演化过程中混淆 API 的演化趋势,发现加密、反射、动态加载、代码隐藏等混淆技术,在恶意样本的生成过程中被越来越多地采用[41].此外,在分析基于代码隐藏构建的恶意软件 Incognito 中,发现其 APK 和 Dex 的资源文件中,恶意代码所包含的常见方法并不多,这意味着在 Incognito 应用程序中隐藏的恶意代码要么不流行、要么被高度混淆而使代码看起来不明显,从而逃避自动化系统动态检测[40].

基于上述对移动平台恶意软件行为演化和对抗技术演化的分析,重建恶意软件系统演化史,将有助于为移动平台中当前恶意软件与之前恶意软件的相关性提供线索,同时明确目前移动平台恶意软件以及未来可能的发展趋势,达到更全面地分析及了解移动平台恶意软件的目的.目前已经提出了基于静态[128]和动态[129]的分析模型,用于识别恶意软件的进化关系.例如,文献[130]提出了基于模型检查的方法来构建 Android 恶意软件家族之间的进化关系.因为恶意软件作者通常会采用代码变换技术[131]隐藏派生关系,使得静态分析无效,文中采用动态分析[132]提取系统调用踪迹生成形式化模型,用于验证恶意软件行为的属性,具体为使用通信系统的微积(calculus of communicating systems,简称 CCS)来制定模型,结合选择性的 mu 演算逻辑表达行为属性,验证结果确定了恶意软件样本共有的常见恶意软件行为.基于该结果推断它们的祖先与后代之间的关系,从而获取家族进化树.建立恶意软件的进化模型有助于把握目前恶意代码的特性、快速地溯源同源样本,发现更多地未知移动平台恶意软件,进一步促进移动平台安全体系的完善.

2 恶意代码溯源机理

学术界和产业界的恶意代码溯源机理存在差异:在学术界,常见的恶意代码溯源依赖于代码的相似性分析,进而揭示恶意代码间的同源关系,且其溯源的目是定位变体家族或作者;而产业界,恶意代码溯源更倾向于恶意代码结构及其攻击链的关联性分析,且其溯源目的是为了挖掘攻击者或攻击背后的真正意图.基于这种差异性,本文将从学术界和产业界分别分析恶意代码的溯源机理.

2.1 学术界恶意代码溯源

学术界恶意代码溯源的基本思想是:采用静态或动态的方式获取恶意代码的特征信息,通过对恶意代码的特征学习,建立不同类别恶意代码的特征模型,通过计算待检测恶意代码针对不同特征类别的相似性度量,指导恶意代码的同源性判定[133].常见的恶意代码溯源主要包括 4 个阶段:特征提取、特征预处理、相似性计算、同源判定,各阶段间的流程关系如图 11 所示.

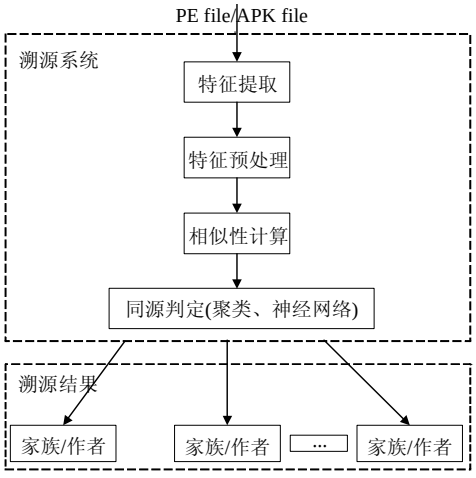


Fig.11 Malicious code traceability system model

图 11 恶意代码溯源系统模型

图 11 所示的溯源对象是 Windows 平台的 PE 文件/Android 平台的 APK 文件.将恶意文件输入溯源系统,经过同源分析,获取溯源结果.溯源系统模型的 4 个阶段中:实线框包围的特征提取和同源判定是溯源系统中不可缺少的阶段;虚线框包围的特征预处理则根据初始特征是否适合于下一阶段的输入来决定是否执行;虚线框包围的相似性计算是基于聚类的同源性分析中必须执行阶段,相似性计算值作为聚类度量值使用,然而该阶段在基于神经网络的同源分析中省去,这是因为基于神经网络的同源算法旨在通过调整神经网络参数,构建同源性分析模型,在调整好的模型中对样本进行同源处理,无需额外执行样本间相似性计算.本文将详细阐述 4 个阶段的实现机理,并评估其优势及缺陷.

2.1.1 特征提取

特征提取是溯源分析过程的基础,具有同源性的恶意代码是通过它们的共有特征与其他代码区分开来的.因此,所提取的特征既要反映出恶意代码的本质和具有同源性恶意代码之间的相似性,又要满足提取的有效性[134].依据溯源目的,溯源特征提取包括溯源家族的特征提取和溯源作者的特征提取.由于编译器的编译、优化以及恶意作者的混淆处理等,编译后的恶意代码中大部分源码信息丢失,特征提取的有效信息量变小.因此,有效地提取恶意代码特征需要做以下两点.

(1) 克服恶意代码的对抗手段.知道变体为避免检测而使用的技术,以便识别在其逃避策略下保持不变的变体的一部分[67].例如在代码级别的变体上,调用间接寻址、代码重排序、垃圾代码插入技术等,逃避使用字符串匹配与偏移或 API 调用匹配的简单签名方法.针对该问题已经提出对抗逃避策略的动态或静态分析技术,例如污点分析、代码语义分析、动态分析等.文献[87]提出采用统计性方法获取常见的多个字符串作为签名,这些签名对于重排序和硬编码都是健壮的,因此可以在一定程度上克服恶意代码的对抗手段.

(2) 相似性代码的片段提取.根据恶意代码家族和作者的编码相似性,恶意代码的家族溯源中,以相似性行为代码片段为导向,在编译后的代码中,提取同家族变体的相似性特征来识别同源样本;恶意代码的作者溯源分析中,在编译后的代码及其结构中,提取作者遗留的编码习惯作为特征,来识别同源样本.学术界常见的特征提取类型包括序列特征和代码结构特征,见表 5,在恶意代码家族溯源或作者溯源中均效果明显.然而因其溯源目标存在差异,所以两种类型的特征在不同的溯源应用中表现不同.此外,权限特征和资源使用特征在移动平台中广泛采用.下面将揭示序列特征、代码结构特征,权限特征和资源使用特征在溯源家族、溯源作者中的应用情况.

Table 5 Malicious code feature type

表 5 恶意代码特征类型

特征名称	提取特征的方法	代表文献
序列特征	主要指从指令、操作码、字节码、API 层次上提取的代码序列	2015 年,Faruki[135]在字节码级别提取统计性强的序列特征,文献[136–139]提取 *n*-gram 字节码序列作为特征,文献[140]以 WinAPI 调用模式作为编码习惯构建特征,文献[141]提出了捕获运行过程中的 API 序列作为特征,利用生物基因序列检测工具 ClustalX 对 API 序列进行相似性分析,最后得到恶意代码的同源性判定,文献[102]通过提取代码中语法信息,抽取代码风格特征
代码结构特征	主要指从指令级别、API 级别上提取的程序流,例如 API 调用图、CFG 图、PDG 图等	文献[133,142]采用静态方法提取 API(application programming interface)调用图作为特征,DroidClone[143]使用基于中间语言 MAIL[144,145]表示的控制流模式作为特征,DNADroid[146]使用 PDG 作为特征,DroidSim[147]是一种基于组件的 CFG 来表示相似性代码特征,与早期的方法相比,该系统检测代码重用更准确
权限特征	主要从 Manifest.xml 中提取〈user-permission〉标签对应的权限	文献[148]提出权限与 API 调用作为特征,文献[149]提出权限与描述文本的组合作为特征
资源特征	主要指 APK 解压文件夹下面的 resource.asrc 以及 res 文件夹下的图片,界面元素等资源	文献[150]提出包含资源的 APK 字节熵作为特征,文献[151]提出传感器等相关资源作为特征,文献[41]提出资源的机构不一致和逻辑不一致特征

1) 序列特征

常见的序列包括指令序列、字节码序列、API 调用序列、*n*-gram 序列等.序列提取是一种语法分析方法,通过分析语法上不容易被混淆的序列作为特征.文献[135]通过熵值评估和实证研究,提取统计性强的字节码序列作为特征,这种特征在同家族恶意代码中广泛存在,即使该样本被混淆,被混淆部分因发生变化很少出现在较多样本中.然而,如果文件被严重混淆,则无法通过静态分析方式获取特征.动态分析可以克服静态分析的此种缺点,文献[136–139]利用动态分析,从系统调用序列中提取 *n*-gram 序列作为特征.*n*-gram 序列相对于整个的 API 调用序列,在数据集中出现的次数更多,稳定性更强,因此该特征相较于 API 调用序列具有较好的统计特性,可以用于检测一般性混淆的变体.这些文献均是针对家族变体的分析,API 调用序列、统计性强的字节码序列反映了同家族的行为特征,可以在家族变体分析中使用.然而在基于作者的溯源分析中,不能单纯依据代码功能相似性提取特征,因为即使同一个作者或团队也会编写出不同家族的恶意代码,但是特定类型的编码习惯在编译过程后保留在了二进制文件里,利用该类信息,可以执行二进制作者的溯源.文献[102,140]均提取到了溯源到作者的特征,文献[140]提取了 7 类 WinAPI 调用模式作为作者的编码习惯特征,有效地溯源到了恶意代码的作者.然而该特征是依据作者已有的样本进行提取,依赖于已有的样本数量.Caliskan-Islam 等人[102]使用 Hex-Rays 反编译二进制文件为类似于 C 的伪代码,基于该代码构建模糊抽象语法树,然后从语法树中获取可以表示作者编码风格的语法特征,从而实现对恶意代码作者的溯源.在恶意代码的家族溯源和作者溯源中,由于恶意代码的特征大多反映功能、结构信息,反映作者与作者关系的特征较少,因此在溯源作者的特征提取中能够提取到的信息较少,且在学术界,针对作者的溯源文献低于针对家族的溯源文献.在序列特征提取中,统计性强的序列特征可以检测到密切相似的数据对象,但检测恶意软件变体的能力要低得多,因此在同源性分析中,序列特征并不用于准确性匹配.

2) 代码结构特征

代码结构提取是一种从代码逻辑结构提取特征的方法,该特征包含了程序的语义信息,属于细粒度特征.常见的代码结构主要包括 API 调用图结构、CFG 图结构、PDG 图结构等.代码结构包含程序的行为语义信息,即使样本的语法被混淆处理,只要语义相同,还是可以追踪到.例如,文献[146]采用 PDG 作为代码特征,通过计算两个样本间的相似性,可以有效地检测复用或克隆恶意代码,同时能够抵抗多种类型的检测逃避技术,例如语句重排序、插入以及删除代码等.虽然恶意代码的代码结构特征使得恶意代码对抗性在检测系统中作用变小甚至无效,但在基于代码结构特征的同源分析中,常用的子图同构算法是 NP 问题,该问题使得系统计算复杂度变大.例如,文献[142]提出加权的敏感 API 调用子图集,作为恶意代码的聚类特征,并对该图进行加权处理,对加权后的敏感 API 调用图执行同构计算,加大了系统分析的复杂度.为了消除该问题,文献[133]提出利用卷积神经网络对

恶意代码 API 调用图进行处理的方法,选择关键节点邻域构建感知野,使图结构数据转换为卷积神经网络能够处理的结构,消除了子图同构带来的问题.

相比代码结构信息,序列信息会丢失恶意代码执行过程中的某些空间特征.但是基于代码结构的相似性分析由于要分析程序执行流程,因此复杂度比较高,效率得不到保证;同时,序列信息比图结构信息更容易获得,因此大多数研究者仍然采用基于指令序列或 API 序列的特征进行研究[133].

3) 权限特征

权限是Android系统的三大保护机制之一,在Manifest文件中,声明的权限很容易捕获应用程序的数据使用意图,可用于识别应用程序的恶意行为[152].权限特征提取属于轻量级方法,避免了高成本的时间和计算,但是它只能捕获应用程序的粗粒度功能,通常需要结合序列特征、代码结构特征等使用,方能深刻地展示应用程序的行为信息.文献[148]提出采用 AXMLPrinter2 工具解析 AndroidManifest.xml 文件获取权限,并集合 API 调用作为特征集合,用于Android恶意软件的静态检测,使用权限与API结合的特征优于仅使用权限作为特征的分类结果,说明权限信息粒度较粗,API 信息有助于恶意软件的分类.文献[149]使用 Android 恶意软件的共同权限和描述文本的组合来估计权限的意图,该方法克服了良性应用有意、无意采用类似的权限所引起的错误检测问题,但是由于描述文本中的正面词语并非强制性的,可以伪造且不会丢失任何恶意功能,因此它仍然存在对错误检测和稳健性的挑战.

4) 资源使用特征

资源文件的使用特征在移动平台中应用广泛,该特征主要是从 APK 解压后的 resources.arsc 文件以及 res 文件中提取的信息.文献[150]提出在重打包的 APK 中,资源文件几乎相同,资源文件的使用可以作为检测恶意代码变体的特征之一,在采用统计性强的字节码熵作为特征的过程中,发现直接提取 APK 文件的字节码的检测率,高于仅使用 dex 字节码的检测.这主要是因为 APK 文件中包含有大量的资源信息,而恶意代码变种往往仅对原始恶意软件中 dex 中的代码进行修改,对资源改变很少甚至无改变,因此采用包含资源文件的字节码特征,更能够有效地检测到家族变体.文献[151]提出传感器类型和生成传感器数据传播路径作为特征,以提供传感器使用模式的概述,然后使用传感器使用模式识别应用程序是否具有危害性.文献[41]提出了结构不一致(structural inconsistencies)和逻辑不一致(logical inconsistencies)的资源特征,从而用于识别 Android 恶意软件的家族变体.但是文中的资源特征提取使用简单的启发式方法派生,没有涉及程序/资源分析,因此效率较高;但是针对混淆而一般样本的检测问题,该文提出的资源特征不足以分析.

2.1.2 特征预处理

特征预处理在溯源分析技术中起到承上启下作用:承上即对初始特征进行预处理;启下则是为相似性计算提供基础数据.提取的原始特征存在不具有代表性、不能量化等问题,特征预处理针对这一问题进行解决,以提取出适用于相似性计算的代表性特征.根据第 2.1.1 节所述,常见的特征类型包括序列特征和代码结构特征,其中,针对移动平台的权限特征和资源使用特征,在具体的表现形式上也以序列特征和代码结构特征为主,下面分析每类特征形式的预处理机理.

1) 序列特征预处理

常见的序列特征预处理方法包括信息熵评估、正则表达式转换、*N*-grams 序列、序列向量化、权重量化法等.序列特征预处理使得初始特征中冗余特征消除、特征语义表达增强、特征量化等以便于进行相似性计算,具体描述如下.

- 信息熵评估

熵是包含在数据中的信息量的非常广泛的量度,是提取稳定性特征的有用工具[153].根据熵值对特征的信息量进行评估,进而选择出稳定性的特征序列.2016 年,Cheng Wang 等人[154]为了优化操作码序列特征的精度和性能,提出了基于信息熵的特征提取方法来提取数量少但非常有用的信息,来表示恶意软件实例.关键步骤如下.

(1) 在汇编代码中选择 2-元组操作码序列,见表 6.

Table 6 Assembly code fragment example

表 6 汇编代码片段举例

回编代码片段	提取的操作码序列

mov eax, 0007BB01
mov ecx, 3E8D0001
call 058A2E97
cmp al, 24
je 000000A0
mov ah, 0E
	os1=⟨mov,mov⟩
	os2=⟨mov,call⟩
	os3=⟨call,cmp⟩
	os4=⟨cmp,je⟩
	os5=⟨je,mov⟩

(2) 利用变形信息熵计算操作码序列信息熵,信息熵公式如(1)所示.

$$ie(OS_k)=\sum_{t=1}^{n1}\left(\frac{\sum_{x_j\in y_i}p(os_k,x_j)}{n_j}\right)\Big/\left(\frac{\sum_{x_j\notin y_i}p(os_k,x_j)}{N-n_j}\right)\Big/n_1 \tag{1}$$

其中,$p(os_k,x_j)$表示恶意代码 x_j 中 os_k 操作码序列出现的概率,y_i 表示恶意软件家族的数目,N 为恶意代码总数.设置阈值α,并将 $ie(OS_k)>\alpha$的操作码序列提取出来用于相似性比较中的特征数据.选择具有高信息熵的操作码序列作为恶意软件实例的表示,结果表明:较少数量的具有高信息熵的操作码序列,比大量的操作码序列提供更好的准确性和性能,缓解了因操作码多样性导致恶意代码的特征维度大的问题.

- 正则表达式转换

正则表达式是由字符元素以及元素之间的组合组成规则序列,以用来表达对字符串的过滤.在序列的分析中,将序列转换为基于序列元素和序列间关系的正则表达式,从而实现对符合正则表达式的恶意软件识别.该特征是一种启发式特征,可以识别事前未出现的恶意软件.例如,L.Wu[155]通过分析恶意软件敏感 API 操作以及事件等,将 API 序列特征转换为正则表达式,并在发生类似的正则表达式模式时检测恶意代码,在最终的实验结果中,证明基于正则表达式的方法可以检测出部分新恶意软件.

- *N*-gram 序列

N-gram 指由较大字符串分割出的固定长度的子字符串[156],例如字符串序列 MALWARE,那么被分割出的 4-grams 是 MALW,ALWA,LWAR,WARE 等.IBM 研究小组最先将 *N*-gram 方法应用于恶意软件分析中[157],使用 *N*-gram 的统计属性可以预测给定序列中下个子序列.从恶意代码提取 *n*-gram 序列,一方面便于进行相似性计算;另一方面,基于 *n*-gram 序列相比原始序列统计性更强,有利于对未知恶意代码序列识别.文献[158]提出对 API 调用序列进行 *n*-gram 处理获取子序列,如果采用 API 序列作为特征,此时序列相对较长,其他代码(如附加功能)会添加到该部分;然而基于 *N*-gram 的序列子集比整个 API 序列的更稳定.采用 *n*-gram 方法将 API 调用序列转换为 *n*-gram 序列,具体过程如图 12 所示.

S_1,S_2 表示初始特征 API 序列,为了方便表示,将 API 名称用单个大写英文字符表示,并基于该字符序列提取对应的 *n*-gram 序列,该特征相比 API 序列特征更短,且序列种类多样化,在表达相似性上更具有代表性.但是 *n*-gram 很难同时捕获不同长度的序列,存在一个有意义的序列被拆开的可能,影响结果的准确性判定.

- 序列向量化

序列向量化是一种量化序列的方法,将具体的序列转换为可进行计算的数学模型.通过量化分析,有利于进行重要特征的分析,提取出具有代表性特征.此外,量化的特征便于进行相似性计算.例如,文献[158]以 API 调用序列作为神经网络分析的目标,经过分析,将一个恶意样本归属到某个恶意软件家族.API 调用序列是由具体名称序列组成的序列,不能进行数学计算,因此在将提取的 API 调用序列特征输入到神经网络之前会进行预处理,如果某个 API 调用序列中出现 2 次或 2 次以上相同的 API 调用子序列,则消除掉重复的子序列.然后采用 one-hot 编码对序列中每一个 API 构建唯一的二级制向量,该向量的长度等于不同 API 调用的总数,将 API 调用名称构成的序列转换为二进制向量序列.由于字典中一共包含 60 个不同的系统调用,因此不会遇到任何特征向量大小的挑战,实现了特征的量化表示,可以输入到神经网络进行运算.此外,基于神经网络的训练向量往往面临维度高问题,文献[159]初始产生基于字符串、3 元-API 调用组合、API-参数调用组合等 3 种类型的特征 0.5 亿个,

为了降低维度,作者提出采用随机映射将输入的特征维度降低了 45 倍,进而使得高纬度的特征数据能够高效地在神经网络模型中训练.序列向量化使得基于特征的数学计算变得可能,进而用于同源性分析的后续处理.

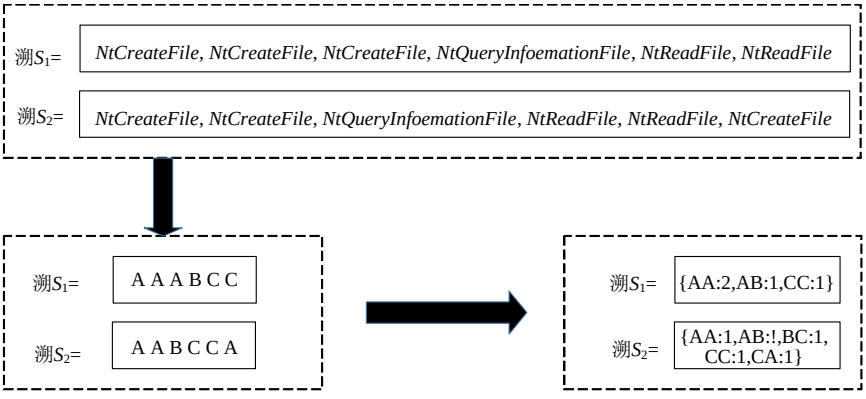


Fig.12 Conversion of API sequences to *n*-gram sequences

图 12 API 序列转化为 *n*-gram 序列

- 权重

权重是一种统计分析方法,权重定义了特征重要性.文献[160]提出计算 API 调用、返回值、模块名称等动态特征向量的权重,以选择重要的特征.由于权重特征本身的限制,该文构建了一种新的基于 TF-IDF 的权重计算方法,如公式(2)所示.

$$\omega(t,a)=\frac{TF(t,a)\times E[P(c\mid t)]}{\sqrt{\sum_{t\in a}[TF(t,a)\times E[P(c\mid t)]]^2}} \tag{2}$$

其中,a 表示恶意软件变体;t 表示恶意软件特征向量;$\omega(t,a)$表示样本 a 中特征 t 的权重;$TF(t,a)$表示变体 a 中包含该特征 t 的数目占该变体中所有特征数目的比例;$E[P(c|t)]$表示特征 t 在不同家族中出现的差异.ω值越大,说明该特征越能表达样本 a 的关键信息.根据该权重值可以选择具有强分辨力的特征,即,具有高权重值的靠前特征被选择,并采用 Min-Max 对选出的特征向量进行归一化处理.将该特征应用于集成学习算法的分类训练中,并与现有的恶意软件分析类方法,如卡方检验(chi-square test)[161]和主成分分析(principal component analysis,简称 PCA)[162]进行比较,发现恶意软件的分类工作比之前更加准确.

2) 代码结构特征预处理

一方面,基于代码结构的特征在相似度比较时存在边、节点等匹配问题即子图同构算法复杂性;另一方面,代码结构特征中存在冗余结构,因此除去冗余、保留与恶意操作相关的代码结构是预处理的主要目的.常见的结构图包括 API 调用图、CFG 调用图、PDG 图等.本文将代码结构特征中预处理机理阐述如下.

- API 调用图预处理

API 调用图是由图中敏感 API,根据前后调用关系建立的网络结构图.

为了消除图相似性计算的复杂性问题,文献[133]对调用图特征进行预处理,以选择 API 调用图中重要节点,具体为采用 PageRank 算法计算 API 调用节点的重要程度,但是在 API 调用图中,某个节点不需要用户随机访问,因此使用 PageRank 算法的时,需要考虑每个 API 对恶意代码行为的贡献程度,结合 PageRank 算法和贡献程度实现对 API 重要程度的计算;同时对 API 进行分类,为不同 API 设置不同的等级;然后计算各节点在图中重要程度,从而选择关键节点,构建出适合于神经网络算法的输入值.

2016 年,文献[90]为了对抗重打包中正常代码部分对恶意代码部分的影响,以提取出更能代表家族恶意代码部分的特征,在基于家族恶意代码 API 调用图的基础上,利用社区结构算法进一步提取 API 调用图的敏感频繁子图,并将其作为家族恶意代码特征,特征的进一步细化使得系统对应用程序的识别速率提升,识别一个软件的时间减少到了 4.4s.

- CFG 图预处理

CFG 图以单个代码语句为原子单位,细粒度地表示了程序的执行流程[163].面对其在图同构方面的复杂性问题,文献[124]采用 Hyperion[164]二进制静态分析工具,将恶意软件的 CFG 结构以正则表达式字符串的形式生成,正则表达式中包含汇编指令信息和程序的控制结构信息,使得基于图的相似性匹配转换为基于字符串的相似性匹配,解决了图匹配带来的复杂度.此外,文献[165]提出采用 Poulik 等人[166]设计的方法将 CFG 转化为基于一定语法的字符串,该方法抽象地表示了应用程序的信息,在抵抗代码变换方面,比基于 *n*-gram 特征表达的方法具有更好的抵抗性.文献[167]提出 CFG 与 DFG 结合,转化为矩阵形式用于 CNN 模型训练,在准确率方面取得了良好的效果,针对 Marvin 数据集的检测达到了 99.649%,但是在召回率方面低于文献[165].这可能是因为前者包含了数据流分析,匹配更加精准,使得漏报情况相对较重.

- PDG 图预处理

PDG 图是一种结合数据流和控制流分析的代码结构,展现了敏感操作的依赖性,包含程序语义信息.但是该特征结构在匹配时是一种准确性匹配,因此计算复杂度比较高.文献[168]对基于单个方法的 PDG 进行预处理,将 PDG 转换为一组特征向量,并根据特征向量找到相似性引用程序集群.该方法对应用程序中所有方法进行处理以提取一组特征向量,当一组新的应用程序加入时,它需要与以前的大量特征向量再次进行比较,因此效率低下.但是需要检测没有任何先验知识的未知恶意代码时,该方法有效.

基于上述描述表明:在图特征的预处理过程中,评估图中的关键节点、非图形化处理、排除图中非相关子图[169]均是对图进行预处理,以消除图相似性计算复杂度的有效方法,基于神经网络模型的应用使得消除图结构间的计算成为可能.在实际同源判定中,需要结合所构建的特征以及实际应用场景及目的,来选择合适的特征预处理方案.

2.1.3 相似性计算

溯源的目是通过分析样本的同源性定位到家族或作者,样本的同源性可以通过分析代码相似性来获取.相似性计算旨在衡量恶意代码间相似度,具体为采用一种相似性模型对恶意代码的特征进行运算.根据预处理特征类型的不同以及溯源需求(效率、准确性等)的差异,采用不同的相似性运算方法.目前比较流行的相似性计算方法主要集中在对集合、序列、向量、图等特征表现形式的处理.

1) 基于集合的相似性计算

集合是多个元素的组合,这是一种简单的特征形式,常见的相似性计算方法为 Jaccard 系数等.Qiao 等人[80]在不同恶意样本 API 集合的相似性比较中采用了 Jaccard 系数方法,将为 *A*,*B* 两个集合的交集在并集中所占的比例作为相似度,比例值越大,证明越相似,如公式(3)所示.

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|} \tag{3}$$

通过以上计算,将查询点与待比较空间中的集合点进行比较,返回满足条件的结果.该算法思想简单,操作方面,在基于集合形式的特征中广泛采用.

2) 序列形式特征的相似性计算

序列特征的相似性计算既要考虑序列元素的匹配,又要考虑序列的次序关系.2013 年,Faruki 等人[135]提出采用 SDhash 相似性散列技术构建样本的签名序列,并采用汉明距离法对序列进行相似性计算,从而识别同源性样本.恶意软件通常会将其恶意负载隐藏在 DEX 或托管为主应用 APK 的资源文件中,文献[163]从 DEX 或 APK 类型的应用程序中提取所有可用资源中的片段构建特征序列,结合模糊 Hash 和特征 Hash 的方法[164]计算样本之间的相似度.此外,也有研究者采用数据挖掘的方法进行相似性计算,例如,乔等人[140]利用频繁模式离群因子计算基于 WinAPI 调用序列的相似性;文献[143]将两个字符串形式的签名中公共的 LCS(largest common substring)长度占较大签名长度的比值作为相似性值,此种方法是一种简单的数学算法,计算复杂度低.

3) 基于向量形式特征的相似性计算

向量是一种基于维度的几何表示方法,在相似性比较中,需要明确各维度的特征类别,使得不同样本的特征

向量中各维度具备一致性.文献[170]提出基于指纹的特征,但该指纹是位特征向量,因此不能直接采用 Jaccard 相似度进行计算.为了计算两个指纹之间的相似性,文中采用了位 Jaccard 相似度计算代表相同内容的位特征向量之间的相似性.具体过程为:1) 计算两个向量并集的基数,假设 A 和 B 是两个位特征向量,那么两个向量并集的基数是 $A+B$,按位求并的结果向量中“1”的数目;2) 计算两个向量交集中基数,仍然基于上述假设,两个向量交集的基数是两个向量按位交集结果中“1”的数目;3) 将交集的基数除以并集的基数即为位向量间的相似度值.Suarez-Tangil 等人[89]用数据挖掘算法中向量空间模型,展示家族的恶意代码特征形式,将同家族提取出来的具有代表性的 CFG 元素作为特征中维度,采用余弦算法对不同家族的向量空间模型进行相似度计算,根据余弦值来判断它们的相似性,从而识别出相似性样本,进而归属到对应的家族.用于比较向量的余弦相似度反映了恶意代码间的相似性,其具体公式如公式(4)所示.

$$Similarity=\cos(\theta)=\frac{A\cdot B}{\|A\|\|B\|}=\frac{\sum_{i=1}^{n}A_i\times B_i}{\sqrt{\sum_{i=1}^{n}A_i^2}\times\sqrt{\sum_{i=1}^{n}B_i^2}} \tag{4}$$

公式(4)中计算的相似性值在 0 和 1 之间,并且基于阈值确定相似性变体.该阈值与相似性分析结果的准确性密切相关,因此在现实世界中,如果要寻找恶意软件变体,则需要严格制定阈值.

4) 图特征的相似性计算

图特征的相似性计算涉及到图中边和节点的匹配,复杂性比较高.文献[171]提出采用的是最小距离匹配度量法,比较不同样本的 CFG 图特征的相似性.Kinable 等人通过静态分析恶意代码的系统调用图,采用图匹配的方式计算图相似性得分,该得分近似于图的编辑距离.利用该得分比较样本的相似性,采用聚类算法将样本进行聚类,实现家族分类[82].Fan 等人[90]将 API 调用图的敏感子图集作为特征,为了计算图之间的相似度,文中自定义加权敏感 API 子图匹配方法来计算同源恶意行为的图形之间的相似度,同时容忍较小的实现差异.文献[105]采用两个匹配——“Initial matching”,“second matching”和统计模型来计算样本特征的相似性.在“Initial matching”中,匹配具有完全相同 4 元组值的方法;在“second matching”中,递归地比较先前匹配方法的 Parent/Child 方法的 4 个元组值.Parent 方法是调用上一个匹配方法的方法,而 Child 方法是由以前匹配的方法调用的方法.

基于上述描述表明:针对不同的特征形式可以采用同一种算法,例如,Jaccard 系数既可以用于集合形式的相似性计算,也可以用于向量等其他特征.此外,在实际的特征相似性分析中,往往由于特征形式的多样性,会采用多种相似性计算方法结合使用.上述相似性比较方法中:基于集合的比较方法实现简单,但没有考虑变量间的相关性;而基于向量的相似性比较考虑了变量的位置关系,但是向量方法容易引起维度膨胀问题;基于序列的相似性比较考虑了变量的次序关系,但缺乏元素间的相关性分析;基于图特征的相似性比较则考虑了变量间的语义相关性,但计算复杂度大,该特征往往用于需要精确匹配的场景中.上述相似性比较的粒度依次变细,在实际溯源分析中,需要结合应用需求,选择合适的相似性计算方法.

2.1.4 同源判定

目前,学术界常见的同源判定方法主要包括基于聚类算法的同源判定、基于神经网络的同源判定等.利用聚类算法操作相似性值获取待测样本同源度,同源度越高,溯源结果越准确.

1) 目前,在恶意代码的聚类中,应用广泛的聚类算法是基于密度的聚类算法、划分聚类算法、层次聚类算法、近邻聚类算法,具体见表 7.

这里以 DBSCAN 聚类算法[172]来阐述其聚类原理.DBSCAN 聚类搜索数据空间中密集区域,并与低密度区域分隔,低密度区域样本被认为是“噪音”,因此被丢弃.DBSCAN 聚类一共包括 4 个阶段,步骤如下.

(1) 对于待测集合中尚未检查过的对象 p,如 p 未被处理过(归为某个簇或者标记为噪声),则检查其邻域:若包含的对象数大于阈值,则建立新簇 C,将其中的所有点加入候选集 N.

(2) 对候选集 N 中所有尚未被处理的对象 q,检查其邻域:若至少包含阈值个对象,则将这些对象加入 N;如果 q 未归入任何一个簇,则将 q 加入 C.

(3) 重复步骤 2),继续检查 N 中未处理的对象,直至当前候选集 N 为空.

(4) 重复步骤 1)~步骤 3),直到所有对象都归入了某个簇或标记为噪声.

Table 7 Application of clustering algorithm in homology judgment

表 7 聚类算法在同源判定中的应用

聚类算法名称	算法描述	代表文献
密度聚类算法	首先设置一个邻域半径,然后计算样本的邻近区域的密度(对象或数据点的数目),如果超过某个阈值,就继续聚类.擅于解决不规则形状的聚类问题	文献[172]采用 DBSCAN 算法对基于调用图聚类,发现类似的恶意软件
划分聚类算法	给定要构建的分区数 k,划分方法首先给出一个初始的分组.然后,它采用一种迭代的重定位技术,通过把对象从一个组移动到另一个组来进行划分	文献[173]采用 K-means 算法对恶意软件的流量特征进行聚类
层次聚类算法	又称树聚类算法,透过一种层次架构方式,反复将数据进行分裂或聚合	文献[89]提出采用层聚类算法,构建家族间演化模型,进而发掘家族功能的演化.文献[174]提出层次聚类和密度聚类算法结合的快速聚类算法对操作码序列特征进行聚类,以识别恶意软件变体,该方法在识别变体方面效率较高,时间复杂度减为 $O(N_{\text{dataset}}\times(N_{\text{cluster}}+N_{\text{neighbor}}))$
近邻聚类算法	设置阈值,从样本集合中任意选择一个作为聚类中心,然后从剩下的样本中挑选样本与该中心计算欧式距离,如果超过阈值则该样本为新的中心,否则,该样本属于前一个中心聚类中成员,以此下去,该方法是一种粗暴方法,且聚类结果受第一个中心影响严重	文献[175]通过基于 LSH 的近邻算法实现了接近实时的聚类功能,这使得利用该模型设计的系统可客观快速地对未知样本进行自动行为分析和家族判定,从而可以大幅提高未知样本的分析效率,减少需要人工分析样本的数量

图 13 描述了在训练时间内子簇及其代表性样本和对应的家族之间的关联.实线方形框表示聚出来的家族,椭圆形框表示簇,椭圆形里面的刺边圆圈表示簇中的代表样本即簇中心.新样本输入时,将新样本分别与已分类家族中的代表样本即椭圆中的刺边圆圈进行相似性计算,选择相似性值排名靠前的样本,通过同源决策(选择相似性样本的平均值,以最接近样本的家族作为归属家族或者将选出的靠前几个候选样本交给人工分析等决策方法),归属到对应的家族.

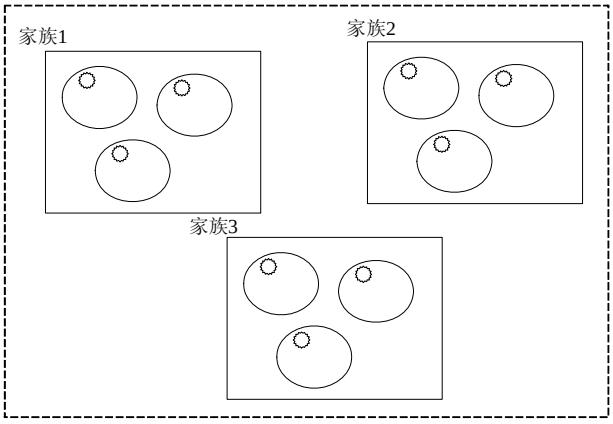


Fig.13 Abstracted cluster model

图 13 抽象聚类模型

DBSCAN 聚类算法的优势在于聚类时不需要事先指定聚类的簇数,可根据样本之间的差异性将相似样本聚为一类.此外,在该算法中,高密度区域可以是任意形状,样本不一定要围绕单个中心,这符合恶意代码演化、衍生的特点,适用于同源性恶意代码的聚类.

基于聚类算法的同源判定一般步骤如下.

- 设置聚类阈值.
- 利用聚类算法进行聚簇运算,使得同一簇内的数据对象的相似性尽可能大;同时,不在同一簇中的数据对象的差异性也尽可能地大.

- 同源判定,如果同簇中属于同一家族或同一作者的多;而同一家族或同一作者的样本几乎很少出现的不同簇中,则表明聚类出的模型同源性判定效果强.

2) 基于神经网络的同源判定.

神经网络是一种多层网络的机器学习算法,可以处理多特征以及复杂特征的同源判定.基本思想为:将样本特征作为输入层数据,然后不断调整神经网络参数,直到输出的样本与该样本是一种同源关系未为止.在测试过程中,将恶意代码特征送输入层,即可判断恶意代码的同源性.

赵炳麟等人[133]提出了基于神经网络的同源判定方法,其整体实现框架如图 14 所示.

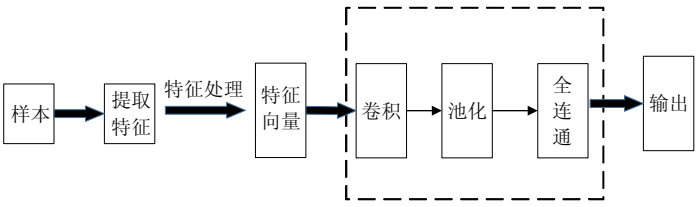


Fig.14 CNN-based homologous analysis model of malicious code

图 14 基于 CNN 的恶意代码同源分析模型

下面描述具体实现过程.

- 采用静态分析方法分析恶意代码,提取其 API 调用图作为恶意代码样本的特征,该阶段操作为图 14 中的“提取特征”.
- 选择调用图中的关键节点,使其能够与卷积神经网络的输入匹配,该阶段为图 14 中的“特征处理”.以关键节点邻域构建感知野,使图结构数据转换为卷积神经网络能够处理的结构,设置一个统一的规则,使图的邻域到感知野是一条单射函数,从而保证具有相似结构特征和属性的节点能够映射到感知野的相似区域.这个过程是对关键节点及其邻近节点的一个规范化的过程,至此,特征向量构建完毕.
- 将特征向量输入 CNN 模型进行训练,直到模型处于稳定状态.至此,基于 CNN 的恶意代码同源判定模型建立完毕.
- 对 8 个家族的恶意样本进行测试,实验结果表明,恶意代码同源性分析的准确率达到 93%,并且针对恶意代码检测的准确率达到 96%.

根据上述研究发现,基于聚类算法的同源判定过程为不断调整聚类阈值、测试其同源性强度的过程.在该过程中存在以下问题:1) 图特征的相似性计算是个 NP 问题,计算复杂度大;2) 当出现多类型特征共存时,需要考虑不同类型特征带来的影响,导致相似性判定模型变得复杂,影响同源判定结果.而基于神经网络的聚类算法主要的运算是对其调参过程,只要特征满足该神经网络模型的输入标准,即使包含多类型特征,依然可执行同源判定.

2.2 产业界恶意代码溯源

产业界除了采用与学术界类似的同源判定方法之外,还会通过关联的方法对恶意代码进行溯源.

2.2.1 溯源意图

产业界的溯源意图除了溯源出编写恶意代码作者、恶意代码家族之外,还要挖掘出攻击者及攻击者背后的真正意图,从而遏制攻击者的进一步行动.

2.2.2 溯源机理

产业界与学术界溯源方法的差异主要表现在特征提取和同源判定两个方面:在特征提取上,产业界更倾向于从代码结构、攻击链中提取相似性特征;在同源判定上,除了采用与已有的历史样本进行相似度聚类分析之外,产业界还会采用一些关联性分析方法.目前,产业界常见的溯源特征见表 8.

Table 8 Traceability characteristics of industry

表 8 产业界溯源特征

溯源特征	溯源文献
时间戳	文献[57,176]
数字证书	文献[176,177]
函数(导入、导出、加密等)	文献[178,179]
后门文件	文献[180]
攻击模式	文献[181]
漏洞利用	文献[57,182]
传播机制	文献[57]
编译环境(编译工具、语言、系统)	文献[22]
通信方式	文献[179]
功能(部分模块功能等)	文献[22]
背景来源(例如语言、域名、地理位置等)	文献[57,177]

表 8 展示了产业界溯源特征类别多,对溯源样本的描述全面详细,下面举例阐述这些特征.

FireEye[176]实验室于 2013 年对 APT 高级可持续攻击进行分析,在攻击所用的恶意代码中发现了相同的代码段、时间戳、数字证书等,收集恶意代码中这些信息作为特征,基于这些特征进行关联分析,认为攻击均是由同一个组织操纵.

文献[177]指出,将所有收集到的可执行文件的时区、时间戳作为溯源特征,根据特定标准分组统计,将统计出来的工作时间映射到攻击者的时区,再根据世界时区分布图,就可以定推断攻击者所在的区域或国家.

Mandiant 的研究人员通过文件的导入函数信息生成哈希作为特征,用于追踪 APT 攻击的恶意代码[178].

2017 年,McAfee 报告[180]指出,Appears 恶意软件与 Lazarus Cybercrime Group 有关,发现 Appears 中包含一个 ELF 格式的后门文件,该 ELF 文件与已报告属于 Lazarus 网络犯罪组织的多个可执行文件类似,因此判定该恶意软件的作者是 Lazarus Cybercrime Group.

攻击模式主要指某些攻击团队特有的攻击套路,并且长期专注于一个领域的攻击.例如 2016 年 10 月爆发的 Mirai 僵尸网络[181]主要利用物联网设备默认口令的问题感染大量设备,通过对其攻击模式进行相似性匹配,溯源到对应的作者.

在“白象”组织最新的攻击行动中[57],安天工程师对“白象组织”进行了追踪分析,通过分析样本中的系统账号、开发编译工具、样本编译时间等对黑客组织成员进行画像分析;继而对用户 ID 进行追踪,并将 ID 为“cr01nkzer0”的黑客关联到了 www.null.co.in 网站的一个发帖人,捕获其邮箱等个人信息;同时,在多个论坛进行关联,深入挖掘相关信息,最终确定其来源区域为东南亚某国.

启明星辰[182]揭露海德薇(Hedwig)组织在利用文件漏洞攻击的样本中,样本所利用的漏洞和内嵌的 Shellcode 高度同源.例如,2012 年至 2014 年拦截到的 CVE-2010-3333 漏洞的 shellcode 和 2015 年拦截到的 CVE-2012-0158 漏洞部分样本的 shellcode 功能、代码完全一致.这些可以作为关联分析的特征,进而溯源到黑客组织.

Gostev[22]花费 2 个月分析 Duqu 木马,发现 Stuxnet 与 Duqu 所用的驱动文件在编译平台、时间、代码等方面具有相似性,并通过关联分析得出 Duqu 是 Stuxnet 先驱的结论.文献[103]从代码模块着手分析,指出 Patchwork 和 Confucius 组织的 Delphi 恶意代码存在相似之处.

安天[57]对“女神”行动的溯源包括对 C&C 的分析,发现“女神”攻击样本涉及到 3 个域名,通过 whois 查询溯源到注册人均来自同一个人,且注册地为东南亚;此外,进行时间戳的分析关联当时活跃的黑客组织,最终确定该攻击来自于东南亚某国.此外,安天对“苦酒(BITTER)”行动进行 C&C 分析,获取攻击邮件域名、伪装样本 App、邮件和数据中“非常好的”英文等,并对这些信息进行关联分析,推断攻击者的区域及特点.

2013 年曝光的“阿克斯(Arx)”组织利用 0day 漏洞(CVE-2013-3906 等传播 Trojan[Spy]/Win32.Zbot 和 Trojan/Win32.Dapato 恶意软件.安天对从攻击者样本的时间节点、攻击目标、传播细节、相关的 C&C 基础设施的注册信息等提取特征进行关联分析,发现“阿克斯”组织与“白象”组织并不存在明显联系,但是也来自东南亚某

国[57],且攻击大部分目标均为中国,这些组织具有相似的背景线索和特点,推测它们可能来自某国的不同组织.

安天对“白象二代”组织[57]的一名开发人员ID为“Kanishk”进行关联分析,发现中文翻译为“迦腻色迦”,迦腻色伽是贵霜帝国(Kushan Empire)的君主,贵霜帝国主要控制范围在印度河流域,此背景信息提供攻击者潜在的来源信息.

上述案例主要从样本、C&C 信息、背景来源等作为溯源特征关联到攻击者.相比学术界溯源特征,产业界溯源特征更加详细全面,信息复杂度大.因此,学术界的同源判定方法并不能完全用于产业界各类特征的相似性分析中.通过对 360 安全威胁中心[183,184]、火眼[185]、安天 CERT[57]、绿盟[181]等公开的溯源素材进行分析,将产业界溯源方法分类见表 9.

Table 9 Common malicious code traceability method in industry

表 9 产业界常见恶意代码溯源方法

溯源方法名	溯源目标
域名/IP[57]	对攻击者使用的域名和 IP 地址进行分析,挖掘攻击源头
全流量分析[181]	某些攻击者或者组织的反跟踪意识非常强,在达成入侵目的之后(窃取数据),会完全清除入侵痕迹,或者干脆销毁主机硬盘,使得在受害者机器上找不到任何痕迹,通过全流量分析溯源就相当有效
入侵日志[57,185]	对攻击者入侵到主机后留下的大量行为操作日志进行分析,可以提取相关攻击者的信息
攻击模型[57,185]	这种溯源方法主要见于某些专业化程度比较高的个人或者组织,他们有自己的攻击常规套路,并且长期专注于一个领域的攻击
样本分析[57,183,184]	通过静态或动态的方法提取样本特征,然后分析攻击者相关信息

溯源方法已经在产业界广泛使用,在实际溯源分析中,一般会结合关联分析技术、画像分析等溯源到攻击者及真实意图.例如,文献[183]获取 C&C 服务器域名后,查询其 whois 信息,然后与贴吧信息关联定位到攻击者.文献[184]中在获得下载恶意代码的域名信息后,利用 360 威胁情报中心分析平台扩展样本信息,获取历史样本信息以及与其相关的溯源报告,通过信息关联定位出 APT 攻击者.绿盟[181]指出,2015 年,乌克兰电厂遭受攻击之后,攻击者利用 killdisk 组件销毁了全部数据,在主机上没有留下任何操作痕迹,最终通过全流量分析找到了溯源线索.FireEye 通过多重调查、针对端点和网络侦测持续监控,并对 APT28 组织的攻击活动进行追踪和画像,最终确定 APT28 是俄罗斯政府幕后支持的黑客组织[185].绿盟[181]在溯源关于攻击者使用加密的 SMTP 服务器窃取敏感信息的案例中,分析入侵日志,获取到邮箱的用户名与密码,登陆攻击者邮箱,在登陆邮箱后发现攻击者真实邮箱,通过进一步的关联分析,定位到了攻击者.文献[181]对样本进行分析获取时间戳,统计出相应的时区,推断出攻击者所在的国家,最后,基于互联网上公开的信息进行关联分析和画像分析,确定攻击组织.这些案例均是通过手动分析关联恶意代码攻击者,虽然定位结果信息全面,但是效率较低,无法做到完全自动化地溯源.此外,从“白象”系列的攻击中发现,中国大量基础的信息安全环节和产品能力还不到位.“白象一代”以免杀 PE 辅助有限的社会工程技巧,成功实施了攻击,这是一种轻量级 APT 攻击;而“白象二代”虽然采用高级的对抗方法,但也未见其使用 0day 漏洞,且所使用的漏洞是已经被修补过的,其中 2 个漏洞也未经过免杀处理,但依然成功实施了攻击.由此可见,当前,补丁、系统加固等基础安全环节不到位、产品能力仍然不足.

3 恶意代码溯源对抗

自从“GhostNet”幽灵网、APT1 等事件之后,近年来,全球范围内各类网络攻击事件不断被曝光,恶意代码溯源手段和技术也在不断发展,“白象”“海莲花(APT32)”“摩诃草”等各类溯源分析报告陆续被公布.2017 年,美国有 24 个研究机构[186]展开了 APT 的相关溯源研究,发布相关研究报告多达 47 篇;我国有 4 个机构发布了 8 篇报告,居全球第二.

恶意样本、攻击行为(攻击者)、攻击手段和渠道(攻击者与攻击事件的关联)是溯源分析的关键要素,黑客组织为了尽可能消除溯源线索,目前从代码、攻击行为等多个层面采取了对抗措施.

(1) 代码角度的溯源对抗:恶意开发者尽可能在代码编写生成阶段消除(或伪造)溯源痕迹,或采用技术手段混淆甚至隐藏自身关键特征信息,避免特征信息的暴漏.

(2) 攻击行为的溯源对抗:黑客组织采用精准定位、伪装、隐藏(持续免杀或无文件运行)、自毁等方式,使得攻击行为不被发现;在攻击渠道方面,黑客组织采用代理、匿名网络技术阻止对实际攻击来源、渠道的追踪.

3.1 代码角度的溯源对抗

代码的溯源对抗主要指攻击者在恶意代码发布之前,对代码文件进行痕迹(如生成时间、时区、特殊字符串、语言、C&C 域名、团队代码特征、开发环境等)消除、伪造或混淆处理,使得安全研究人员从中无法或者尽可能少地提取到有效信息.例如 2017 年,安天移动安全分析团队对 Dvmap 病毒分析报告[187]显示:该黑客组织故意隐匿 APK 文件生成时间,在生成 APK 时修改系统本地时间,导致解包 APK 文件获取到的生成时间为 1979 年.同样在 2017 年,维基解密公布 Vault7 系列数百份文件中泄露,CIA 在恶意程序源码 Marble 中插入外语,嫁祸中国、俄罗斯等国.比如,将恶意程序中使用的语言伪装为汉语而非美式英语,然后假装掩饰使用汉语的痕迹,用于阻碍取证调查人员和反病毒公司将病毒、木马和黑客攻击行为溯源到 CIA 身上.另外,大量使用加壳和代码加固方案,也提升了代码痕迹被提取分析的难度.

第 1.1.3 节描述了编写恶意软件的对抗性方法,这些方法同样使得从恶意代码中提取同源特征信息变得困难.

以上这些措施不仅会妨碍产业界对单个样本的人工溯源进度以及与其他样本的关联难度,同时也将给需要大量标注样本的学术领域溯源研究带来障碍.

然而,随着主机行为监控[188]、恶意代码 API[91]、栈异常的 shellcode 检测[189]、敏感信息测量[190]、内核虚拟机防护[191]、解析壳[192]等各种安全防护机制的不断出现,恶意开发者也开始对代码进行了自我保护处理,使得代码的解读变得更加困难.

3.2 攻击行为的溯源对抗

恶意软件攻击行为的溯源对抗主要指软件运行后,攻击团队或恶意软件自身进行的对抗行为,主要包括自身伪装、持续免杀策略、攻击定向性提升、控守网络隐匿、攻击载体隐匿等.

(1) 增强自身伪装,提高人工分析溯源难度.

目前常见的伪装程序分为重打包程序和区域性伪装.重打包程序伪装将恶意程序逻辑隐藏在合法程序的有用功能之后,且恶意程序只占重打包程序的小部分,致使系统调用[193]和敏感路径[194]等特征在合法组件和恶意组件中无法区分,因此在恶意代码检测及恶意代码家族变体识别时容易逃离.陈凯等人[195]为了识别出重打包恶意软件,构建基于 UI 界面结构和代码方法层结构的 MassVet,该方法在识别重打包恶意代码方面取得效果,但仍然有限,特别是识别具有防御策略的恶意软件.

区域性伪装则往往与特定区域相关,主要是通过将特定区域的语言、域名、时区、组织等嵌入到伪装程序中,从而逃离区域溯源.例如,2017 年 6 月 8 日,卡巴斯基[196]发布报告《Dvmap: The first Android malware with code injection》指出,该恶意软件所包含的“.root_sh”脚本文件中存在中文注释,推测其开发人员可能是中国人,但通过对其恶意样本载荷向位于亚马逊云的页面接口回传的数据进行分析,发现该页面域名中包含“d3pritf0m3bku5”字样,经分析为印度尼西亚方言;同时,对恶意样本 colourblock 的 Google Play 市场缓存及全球其他分发来源的页面留存信息关联分析,得到 Retgumhoap Kanumep 为该恶意样本声明的作者姓名,其姓“Kanumep”各字母从右至左逆序排列则为 Pemunak,即印度尼西亚语“软件”之意,融合多方证据证明,该黑客来源于印度尼西亚.

(2) 增强免杀手段,提高抗分析能力,提高安全软件检测难度.

恶意代码的复杂度显著增强,主要指恶意代码开发的复杂性提升,以逃避查杀.2017 年,在高级攻击领域[58],FinSpy 的代码经过多层虚拟机保护,并且还有反调试和反虚拟机等功能;在 2017 年 4 月,Seduploader 的新版本增加了一些新功能,例如截图功能或从 C2 服务器直接加载到内存中执行,从而逃避恶意软件的检测.安天在《白象的舞步——HangOver 攻击事件回顾及部分样本分析》中指出,“白象一代”使用了超过 500 个 C&C 域名样本,同时采用多种环境开发编译,例如 VC,VB,.net,Autoit 等,结合 PE 免杀处理等手段,且在该次攻击后,具有相关基

因特点的攻击载荷变少,说明黑客组织已经开始着手对抗检测方法,并已经拥有对抗溯源的意识.而在 2015 年的“白象二代”中,黑客组织使用了具有极高社工构造技巧的鱼叉钓鱼邮件进行定向投放、在传播方式上不再单纯采用附件而转为下载链接、部分漏洞利用采用了反检测技术对抗手段,初步具备了更为清晰的远程控制的指令体系.“白象二代”相比“白象一代”的技术手段更为高级,其攻击行动在整体性和技术能力上的提升,可能带来攻击成功率上的提升.

(3) 提升目标定位精度,增强攻击定向性,降低样本被捕获几率.

传统的恶意代码攻击未分析目标特点,实行随机性的最大传播策略,这种机制容易暴漏自己.因此,为了避免被发现,攻击者会尽可能地多了解攻击目标,实施小范围的攻击活动,这样可有效降低样本被捕获分析的几率.例如海莲花团伙的攻击活动中,鱼叉邮件的社工特性突出,体现为对攻击目标的深度了解,例如样本中的附件名:invitation letter-zhejiang ***** working group.doc,星号是非常具体的目标所在组织的简称,意味着这是完全对目标定制的攻击木马.2018 年,Confucius 黑客组织执行客户端和服务器端的 IP 过滤,仅对指定 IP 地址进行破坏,如果受害者来自攻击目标以外的国家,该程序将自行删除并退出.这与 2017 年底,来自该组织的 C&C 不仅可以从任何 IP 地址访问,而且可以在不进行身份验证的情况下,浏览服务器目录树,形成鲜明对比[58].

(4) 提升通信控制方式的隐匿性,提高攻击者被溯源难度.

目前,恶意代码通信方式更加隐蔽,体现在域名隐藏、IP 地址动态变化等.例如,海莲花组织为了抵抗溯源开启 Whois 域名隐藏和并且不断更换服务器的 IP 地址,并使用 DGA 算法生成动态域名,增加了安全分析人员定位有效服务器难度.海莲花组织在最新攻击中,攻击者对采用的网络基础设施也做了更彻底的隔离,使之更不容易做关联溯源分析[58].

(5) 通过无痕迹运行技术,隐匿攻击代码,提升攻击代码被定位难度.

近年来,无文件恶意代码攻击又开始逐渐引起注意.无文件恶意代码没有文件载体仅在内存中运行,该类软件运行后不会在磁盘上留下痕迹,溯源检测困难.比如,安天分析“女神”(Shakti)行动发现,其样本运行时会在内存中解密一个 dll 模块并注入到浏览器进程中,这些 dll 模块都被直接注入到内存中运行,在磁盘中并无实体文件[58].趋势科技[197]发现,木马软件 JS-POWMET.DE 通过完全无文件的感染,完成整个攻击过程,这种极其隐蔽的操作使得沙箱难以分析,甚至专门的安全分析师也难以察觉.

这种恶意软件的出现已经表明,网络犯罪分子将会使用一切手段来躲避安全软件的检测和分析.这在一定程度上说明那些不常见的无文件恶意软件的感染方法也在不断发展,即使安全研究人员能够从内存中获取代码,调查工作仍然很难开展.

4 恶意代码溯源面临的挑战和发展趋势

严峻的网络安全对抗和博弈形势,使得对恶意代码的演化与溯源技术的研究价值凸显,学术界、产业界近年来分别从攻击和防护[198,199]两个方面展开了深入的研究.前文基于已有的研究总结了恶意代码的生成过程和编码特征,并对来自产业界、学术界恶意代码的溯源机理和溯源对抗方法进行了详细描述.目前,学术界和产业界在恶意代码溯源技术方面取得了较大的进步,在追踪恶意代码组织、黑客组织(攻击者)、发现未知恶意代码方面取得了部分研究成果,例如海莲花、白象、方程式组织等典型 APT 攻击计划和黑客团队的不断曝光.但依然存在不足和挑战.具体描述如下:

(1) 产业界恶意代码溯源特征提取与分析的自动化程度严重不足.

目前,产业界的溯源分析过程主要基于人工分析手段,例如,Sasser 与 Netsky、Duqu 与 Stuxnet、Flame 与 Gauss 等的识别工作均由相关安全专家人工分析完成,虽然分析结果详细全面、可信度高,但受专家经验影响较大,其效率较低.

这主要是因为溯源特征碎片化,特征提取位置不稳定,与此同时,恶意代码作者采用大量的对抗技术应对自动化检测与分析,使得特征自动化提取困难.这些困难都是目前自动化溯源分析面临的重要挑战.

(2) 编译器及开源代码复用给溯源工作带来干扰.

一方面,大量攻击程序源代码被公开,开源项目及第三方库如雨后春笋,这也导致目前很多恶意代码在编码阶段开始进行大量的代码复用;另外一方面,即便处理不同的复用代码,编译器也将自动插入大量相似代码,这给基于代码相似性的溯源分析带来很大干扰.比如,文献[200]使用 VC6.0 和 GCC-4.7.2 编译了一个只包含用 c 语言编写的主函数的源代码文件,使用 IDA pro 逆向二进制可执行文件,发现 VC6.0 插入了 103 个函数.而 GCC-4.7.2 插入了 18 个函数,不同编译器插入的函数与函数插入的位置均不同,需要大量的经验和技巧才能识别这些函数,这类函数对恶意代码分析与同源判定工作造成干扰.如何有效排除公共复用代码干扰,是后续需要解决的一个问题.

除此之外,即便是对于相同的病毒代码,当攻击者采用不同的编译器、不同的编译参数进行编译时,最终生成的代码特征也可能发生较大变化,从而导致原本同源的代码被漏判.

(3) 溯源对抗和伪造手段将对溯源工作带来严重挑战.

目前,有效的恶意代码溯源案例中,大多都依赖了类似特殊字符串、语言、控守地址、作者编码心理、代码风格、时间戳等极具个性化且易被伪造的低维特征.为了防止被溯源追踪,目前部分恶意软件作者和团队已经具备溯源对抗意识并开始采用溯源对抗甚至伪造手段,这将对恶意软件的溯源工作带来极大挑战.

(4) 恶意代码溯源基础库缺乏,溯源分析效率低下.

提升溯源效率的第一要素,需要构建恶意代码、代码编写者、攻击团队的代码或行为基础库,否则,即便获得以往攻击样本的变种,也很难快速获得溯源数据支撑.

目前,在溯源基础库的构建方面还存在较多问题.

- 首先,在恶意代码溯源特征库构建方面,目前已有的恶意性判定特征和积累难以直接适用.目前,恶意代码捕获样本数量多,但以往主要是以“恶意性判定”为目标来构建特征库,但恶意性判定特征与恶意代码家族聚类特征存在很大差异.
- 其次,在恶意代码编写者和攻击者基础库方面,由于之前被定位的恶意代码编写者与攻击者基础数据少,这一工作还难以有效开展.

(5) 溯源攻击者(代码使用者)困难.

目前,针对攻击者的追踪主要发生在产业界,恶意代码攻击者的追踪需要对攻击者手段、攻击者行为进行刻画,同时需要结合攻击意图等关联到攻击者.攻击意图的分析需要结合多个攻击样本集分析其功能,同时了解黑客背景,例如黑客的活跃区域、擅长的攻击手段等,构建攻击行为演化趋势,揭示出攻击背后的真正意图.然而在实际的攻击中,黑客活动异常隐蔽,所暴露的活动只是很少部分,此外,嫁祸类攻击日益增多,因此识别真正来自于某个组织的攻击特点是溯源攻击者的另一挑战.

除此之外,攻击者信息还依赖于跨境、跨国第三方平台(论坛、Github、网盘资源)信息等,但是第三方平台人员信息复杂、数量庞大,同时存在伪造信息的可能,因此,即使能够追溯到攻击者的部分信息,准确定位到个人还是需要进一步针对个人进行审查.

(6) 新型 APT 攻击溯源难度大.

目前,学术界和产业界在恶意代码家族聚类[91]及溯源分析[189]方面主要依赖已知的恶意代码,已知样本量越大,溯源结果越准确.已有的恶意样本为溯源变体提供了先验知识,但如果仅根据已有的样本来识别变体,将可能导致溯源工作低效甚至无效,特别是在应对 APT 环境下的零日恶意样本时.随着 APT 功能的复杂以及逃避技术的增强,已有的历史数据无法检测出高级的 APT 攻击.

(7) 产学研合作需要增强.

目前,学术界恶意代码溯源分析方法主要采用学术界已有算法和相关理论成果,并辅以产业界公开的安全分析网站及报告等信息,样本提取与标注数据严重不足,难以构建满足现实溯源需求、切实有效的溯源分析方法.而产业界获取的特征信息虽然详细全面,但是缺乏良好的自动化特征提取模式与溯源关联分析机制,更多的是借助于人工分析平台辅以有限的信息自动化关联,效率低下.产业界与学术界如何进一步加强合作,构建更加有效的溯源技术与体系,是今后发展的一个必然选择.

在应对相关挑战的同时,恶意软件溯源研究与分析工作将得到系统化推动,溯源对抗技术也将在对抗博弈中得到不断发展.对于后续发展,相关展望如下.

(1) 政府部门将在溯源基础信息库的建设工作起到主导作用,多方协同建设机制将被构建.

恶意代码的演化衍生特性涉及软件静态特征、行为规律、家族衍生特性、作者编码习惯、恶意软件团队特性等,这些是做好恶意代码溯源分析工作的基础与重要依据.

选择合适的特征及特征粒度构建基础库特征模板,充分利用现有的恶意软件信息库以及程序自动化分析技术,构建融合恶意软件静态特征、行为规律、家族衍生特性、作者及团队风格等的强大溯源基础信息库,是今后不可回避的一个研究和建设重点.

在溯源基础信息库的建设过程中,基于恶意代码溯源工作的重要性与特殊性,政府部门必将起到主导作用,各大安全厂商、重要互联网企业、各类重要信息系统所在单位等将作为重要支撑单位进行协同建设.比如,当前的各大反病毒厂商的样本以及归属判定知识,将为基础信息库奠定重要基础,同时由于其强大的样本捕获与分析体系,反病毒厂商也必将成为溯源基础信息库后续不断完善的重要渠道.

(2) 网络威胁情报库建设与共享将进一步增强.

准确、及时的网络威胁情报库,是恶意软件发现和溯源的另外一个重要基础.目前,多个安全公司都构建了自己的网络威胁情报分析平台,如何有效地加强合作建设与资源共享,将成为国家网络威胁溯源的重要基础.

从建设内容上说,除了目前已有的威胁数据之外,来自暗网和攻击第一现场的最新威胁情报、最新恶意软件样本、最新漏洞攻击样本以及代码静态和行为特征,都将成为网络威胁情报库建设的重要内容;与此同时,开源软件平台、典型网络攻击平台和框架的公开资源也将成为网络威胁情报库构建时的重要来源.

(3) 溯源对抗措施将日益成熟和体系化,溯源质量和可信度将受到影响

目前,逃避溯源已经成为恶意软件攻击的一个重要考量因素.与此同时,产业界中基于低维经验特征的溯源取证分析工作还将持续,人工分析在相当一段时间内依然是溯源分析的主要手段.

在此背景下,恶意软件作者和团队将投入更多精力在溯源对抗技术研发中,基于现有溯源机制的溯源对抗方法、框架和体系将会得到不断成熟,相应的溯源伪造、干扰工具和体系将被构建.这不仅可以去除、混淆甚至能够伪造溯源特征,这将对今后恶意软件溯源分析报告的质量与可信度形成极大影响.

(4) 溯源痕迹伪造识别技术将成为恶意代码溯源工作的重要需求.

随着溯源痕迹伪造技术的不断应用,现有的基于低维简单溯源特征的溯源分析手段将难以发挥应有的作用.如何有效识别溯源伪造样本,排除溯源伪造样本形成的干扰,成为产业界溯源工作不可回避的重要任务,这也将成为学术界溯源分析研究的重要需求之一.

(5) 难以伪造的基于高层语义的溯源特征将被提出和构建.

细粒度的恶意软件行为刻画、代表攻击本质的恶意软件攻击意图重塑等,将可能成为对抗伪造溯源痕迹的重要依据与突破点.

(6) 多维度的溯源特征智能化解析机制将被构建.

随着样本特征库信息量以及样本分析粒度的不断丰富,对恶意代码的家族和来源认识将更加全面,在样本细粒度分析与提取技术的不断提升下,家族恶意代码特征智能化解析将成为发展趋势,各类恶意软件溯源特征自动化分析平台将被构建,进而可脱离对人工分析的过度依赖.而自动化解析机制的形成,也将进一步推动溯源基础信息库的构建.

(7) 基于智能化的恶意软件自动化溯源定位机制将逐渐构建.

随着恶意软件家族聚类研究和溯源分析技术的不断推进,以及恶意样本的溯源知识库的不断完善,恶意代码、团队、作者溯源基因库将逐步构建和细化,产业界与学术界将合作日益紧密.结合网络态势感知、数字画像技术、大数据分析、机器学习、深度学习等技术实现的攻击者自动化溯源定位机制将被构建和不断推动.

References:

[1] AV-TEST researchers. Security report 2016/17. 2017. https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf

[2] Feng Y, Anand S, Dillig I, Aiken A. Apposcopy: Semantics-based detection of android malware through static analysis. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2014. 576–587.

[3] Sharif M, Yegneswaran V, Saidi H, Porras P, Lee W. Eureka: A framework for enabling static malware analysis. European Symp. on Research in Computer Security. Berlin, Heidelberg: Springer-Verlag, 2008. 481–500.

[4] Christodorescu M, Jha S, Kruegel C. Mining specifications of malicious behavior. In: Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC-FSE). 2007. 5–14.

[5] Xue L, Zhou Y, Chen T, Luo X. Malton: Towards on-device non-invasive mobile malware analysis for ART. In: Proc. of the 26th USENIX Security Symp. (USENIX Security 2017). 2017. 289–306.

[6] Willems C, Holz T, Freiling F. Toward automated dynamic malware analysis using cwsandbox. IEEE Security & Privacy, 2007, 5(2):32–39.

[7] Cui Z, Xue F, Cai X, Cao Y, Wang G, Chen J. Detection of malicious code variants based on deep learning. IEEE Trans. on Industrial Informatics, 2018,14(7):3187–3196.

[8] Sundarkumar GG, Ravi V, Nwogu I, Govindaraju V. Malware detection via API calls, topic models and machine learning. In: Proc. of the 2015 IEEE Int'l Conf. on Automation Science and Engineering (CASE). 2015. 1212–1217.

[9] Canfora G, Medvet E, Mercaldo F, Visaggio CA. Detecting android malware using sequences of system calls. In: Proc. of the 3rd Int'l Workshop on Software Development Lifecycle for Mobile. ACM Press, 2015. 13–20.

[10] Li W, Ge J, Dai G. Detecting malware for android platform: An SVM-based approach. In: Proc. of the 2015 IEEE 2nd Int'l Conf. on Cyber Security and Cloud Computing (CSCloud). IEEE, 2015. 464–469.

[11] Zhao K, Zhang D, Su X, Li W. Fest: A feature extraction and selection tool for Android malware detection. In: Proc. of the 2015 IEEE Symp. on Computers and Communication (ISCC). IEEE, 2015. 714–720.

[12] Narudin FA, Feizollah A, Anuar NB, Gani A. Evaluation of machine learning classifiers for mobile malware detection. Soft Computing, 2016,20(1):343–357.

[13] Wang C, Qin Z, Zhang J, Yin H. A malware variants detection methodology with an opcode based feature method and a fast density based clustering algorithm. In: Proc. of the 2016 12th Int'l Conf. on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD). IEEE, 2016. 481–487.

[14] Kolosnjaji B, Zarras A, Webster G, Eckert C. Deep learning for classification of malware system call sequences. In: Proc. of the Australasian Joint Conf. on Artificial Intelligence. Cham: Springer-Verlag, 2016. 137–149.

[15] Avasarala BR, Day JC, Steiner D. System and method for automated machine-learning, zero-day malware detection. U.S. Patent 9,292,688, 2016.

[16] Wu WC, Hung SH. DroidDolphin: A dynamic Android malware detection framework using big data and machine learning. In: Proc. of the 2014 Conf. on Research in Adaptive and Convergent Systems. Towson: ACM Press, 2014. 247–252.

[17] AVLTeam. Mobile-side C# virus "resurrection", using well-known application communication to achieve remote control privacy theft. 2017. 12–28 (in Chinese). https://bbs.pediy.com/thread-223596.htm

[18] Kroustek J. Avast reports on WanaCrypt0r 2.0 ransomware that infected NHS and telefonica. Avast Security News. Avast Software Inc. 2017. https://blog.avast.com/ransomware-that-infected-telefonica-and-nhs-hospitals-isspreading-aggressively-with-over-50000-attacks-so-far-today

[19] FireEye. The Need for Speed: 2013 Incident Response Survey. Information Security Media Group, 2013. 6–10.

[20] Ahmadi M, Ulyanov D, Semenov S, Trofimov M, Giacinto G. Novel feature extraction, selection and fusion for effective malware family classification. In: Proc. of the 6th ACM Conf. on Data and Application Security and Privacy. Louisiana: ACM Press, 2016. 183–194.

[21] Kinable J, Kostakis O. Malware classification based on call graph clustering. Journal of Computer Virology and Hacking Techniques, 2011,7(4):233–245.

[22] Gostev A, Soumenkov I. Stuxnet/Duqu: The evolution of drivers. 2011. http://www.securelist.com/en/analysis/204792208/Stuxnet Duqu The Evolution of Drivers

[23] Kaspersky Lab. Resource 207: Kaspersky Lab research proves that stuxnet and flame developers are connected. 2017. http://www.kaspersky.com/about/news/virus/2017/Resource_207_Kaspersky_Lab_Research_Proves_that_Stuxnet_and_Flame_Developers_are_Connected

[24] 360 SkyEye Labs. Ocean Lotus APT report abstract. 2015 (in Chinese). http://blogs.360.cn/blog/oceanlotus-apt/

[25] Kong DG, Tan XB, Xi HS, Gong T, Shuai JM. Obfuscated malware detection based on boosting multilevel features. Ruan Jian Xue Bao/Journal of Software, 2011,22(3):522−533 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3727.htm [doi: 10.3724/SP.J.1001.2011.03727]

[26] Wood P, Nahorney B, Chandrasekar K, Wallace S, Haley K. Internet security threat report. Report, Symantec Corporation, 2014.

[27] Symantec. 2018 Internet Security Threat Report, 2017. https://www.symantec.com/content/dam/symantec/docs/reports/mobile-threat-intelligence-report-2017-en.pdf

[28] Bencsáth B, Pék G, Buttyán L, Félegyházi M. Duqu: A stuxnet-like malware found in the wild. CrySyS Lab Technical Report, 2011. 10−14.

[29] GReAT. Gauss: Abnormal distribution 2012. 2012. https://securelist.com/analysis/publications/36620/gauss-abnormal-distribution/

[30] Manku GS, Jain A, Sarma AD. Detecting near-duplicates for Web crawling. In: Proc. of the 16th Int'l Conf. on World Wide Web. Banff, 2007. 141−149.

[31] Wang R, Feng DG, Yang Y, Su PR. Semantics-based malware behavior signature extraction and detection method. Ruan Jian Xue Bao/Journal of Software, 2012,23(2):378−393 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3953.htm [doi: 10.3724/SP.J.1001.2012.03953]

[32] Thompson GR, Flynn LA. Polymorphic malware detection and identification via context-free grammar homomorphism. Bell Labs Technical Journal, 2007,12(3):139−147.

[33] Cai Z, Yap RH. Inferring the detection logic and evaluating the effectiveness of android anti-virus apps. In: Proc. of the ACM Conf. on Data and Application Security and Privacy (CODASPY). 2016. 172−182.

[34] Maiorca D, Ariu D, Corona I, Aresu M, Giacinto G. Stealth attacks: An extended insight into the obfuscation effects on Android malware. Computers & Security, 2015,51:16−31.

[35] Alam S, Qu Z, Riley R, Ryan R, Chen Y. DroidNative: Automating and optimizing detection of Android native code malware variants. Computers & Security, 2017,65:230−246.

[36] SQUARE. G. Dexguard. 2015. https://www.saikoa.com/dexguard

[37] SQUARE. G. Proguard. 2015. https://www.saikoa.com/proguard

[38] Zheng M, Lee PP, Lui JC. Adam: An automatic and extensible platform to stress test android anti-virus systems. In: Proc. of the Detection of Intrusions and Malware, and Vulnerability Assessment. 2013. 82−101.

[39] Rastogi V, Chen Y, Jiang X. Catch me if you can: Evaluating android anti-malware against transformation attacks. IEEE Trans. on Information Forensics and Security, 2014,9(1):99−108.

[40] Suarez-Tangil G, Tapiador JE, Peris-Lopez P. Stegomalware: Playing hide and seek with malicious components in smartphone apps. In: Proc. of the 10th Int'l Conf. on Information Security and Cryptology (Inscrypt). Springer-Verlag, 2014. 496−515.

[41] Suarez-Tangil G, Dash SK, Ahmadi M, Kinder J, Giacinto G, Cavallaro L. DroidSieve: Fast and accurate classification of obfuscated Android malware. In: Proc. of the 7th ACM Conf. on Data and Application Security and Privacy. ACM Press, 2017. 309−320.

[42] Suarez-Tangil G, Tapiador J, Lombardi F, Di Pietro R. Alterdroid: Differential fault analysis of obfuscated smartphone malware. IEEE Trans. on Mobile Computing, 2016,15(4):789−802.

[43] Suarez-Tangil G, Tapiador JE, Peris-Lopez P. Stegomalware: Playing hide and seek with malicious components in smartphone apps. In: Proc. of the 10th Int'l Conf. on Information Security and Cryptology (Inscrypt). Springer-Verlag, 2014. 496−515.

[44] Cheng BL, Ming J, Fu JM, Peng GJ, Chen T, Zhang XS, Marion JY. Towards paving the way for large-scale windows malware analysis: Generic binary unpacking with orders-of-magnitude performance boost. In: Proc. of the 25th ACM Conf. on Computer and Communications Security. Toronto, 2018.

[45] KanXue.Tool of KanXue. 2018. https://tools.pediy.com/

[46] Peng GJ, Fu JM, Liang Y. Software Security. Wuhan: Wuhan University Press, 2015. 262−264 (in Chinese).

[47] Garcia DR. AntiCuckoo. 2017. https://github.com/David-Reguera-Garcia-Dreg/anticuckoo

[48] Chubachi Y, Aiko K. SLIME: Automated anti-sandboxing disarmament system. Black Hat Asia, 2015. https://www.blackhat.com/docs/asia-15/materials/asia-15-Chubachi-Slime-Automated-Anti-Sandboxing-Disarmament-System.pdf

[49] Sudeep S. Breaking the Sandbox. 2015. https://www.exploitdb.com/docs/34591.pdf

[50] Kang MG, Yin H, Hanna S, McCamant S, Dawn S. Emulating emulation-resistant malware. In: Proc. of the 1st ACM Workshop on Virtual Machine Security (VMSec). 2009.

[51] Pék G, Bencsáth B, Buttyán L. nEther: In-guest detection of out-of-the-guest malware analyzers. In: Proc. of the 4th European Workshop on System Security. ACM Press, 2011.

[52] Shi H, Alwabel A, Mirkovic J. Cardinal pill testing of system virtual machines. In: Proc. of the 23rd USENIX Security Symp. 2014. 271−285.

[53] Wang G, Estrada ZJ, Pham C, Kalbarczyk Z, Iyer RK. Hypervisor introspection: A technique for evading passive virtual machine monitoring. In: Proc. of the 9th USENIX Workshop on Offensive Technologies. 2015.

[54] Thanasis P, Giannis V, Elias A. Rage against the virtual machine: Hindering dynamic analysis of android malware. In: Proc. of the 7th European Workshop on System Security. ACM Press, 2014.

[55] Shao SH, Gao Q, Ma S, Duan FY, Ma X, Zhang SK, Hu JH. Process in research on buffer overflow vulnerability analysis technologies. Ruan Jian Xue Bao/Journal of Software,2018,29(5):1179−1198 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5504.htm [doi: 10.13328/j.cnki.jos.005504]

[56] Smith L, Read B. APT28 targets hospitality sector, presents threat to travelers. 2018. https://www.fireeye.com/blog/threat-research/2017/08/apt28-targets-hospitality-sector.html

[57] Antiy CERT. Lurking elephants—Attacks that cross the roof of the world. 2017 (in Chinese). https://mp.weixin.qq.com/s/nnrDVgH-mEzZ8cytaUEUVg

[58] Helio Team, 360CERT, SkyEye Labs. 2017 Chinese APT research. 2018 (in Chinese). https://mp.weixin.qq.com/s/Su3IFORhc55Py7oXOn32ng

[59] Brückner M, Kanzow C, Scheffer T. Static prediction games for adversarial learning problems. Journal of Machine Learning Research, 2012,13:2617−2654.

[60] Laskov P. Practical evasion of a learning-based classifier: A case study. In: Proc. of the 2014 IEEE Symp. on Security and Privacy (SP). IEEE, 2014. 197−211.

[61] Xu W, Qi Y, Evans D. Automatically evading classifiers. In: Proc. of the 2016 Network and Distributed Systems Symp. 2016. 1−15.

[62] Dang H, Huang Y, Chang EC. Evading classifiers by morphing in the dark. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security. ACM Press, 2017. 119−133.

[63] Kaspersky Lab. Alisa Shevchenko. An overview of mobile device security. 2005. https://securelist.com/an-overview-of-mobile-device-security/36059/

[64] Pan X, Wang X, Duan Y, Wang X, Yin H. Dark hazard: Learning-based, large-scale discovery of hidden sensitive operations in Android apps. In: Proc. of the ISOC Network and Distributed System Security Symp. (NDSS). 2017. 1−15.

[65] Zhang M, Duan Y, Yin H, Zhao Z. Semantics-aware Android malware classification using weighted contextual API dependency graphs. In: Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security. ACM Press, 2014. 1105−1116.

[66] Elish KO, Shu X, Yao DD, Ryder BG, Jiang X. Profiling user-trigger dependence for Android malware detection. Computers & Security, 2015,49:255−273.

[67] Yang W, Xiao X, Andow B, Li S, Xie T, Enck W. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Software Engineering, Vol.1. IEEE, 2015. 303−313.

[68] Chen KZ, Johnson N, D'Silva V, Dai S, MacNamara K, Magrino T, Wu EX, Rinard M, Song D. Contextual policy enforcement in Android applications with permission event graphs. In: Proc. of the 20th Annual Network and Distributed System Security Symp. (NDSS 2013). 2013.

[69] Fratantonio Y, Bianchi A, Robertson W, Kirda E, Kruegel C, Vigna G. Triggerscope: Towards detecting logic bombs in Android applications. In: Proc. of the IEEE Symp. on Security and Privacy. 2016. 377–396.

[70] Wang H, Guo Y, Ma Z, Chen X. Wukong: A scalable and accurate two-phase approach to Android app clone detection. In: Proc. of the 2015 Int'l Symp. on Software Testing and Analysis ACM Press, 2015. 71–82.

[71] Reina A, Fattori A, Cavallaro L. A system call-centric analysis and stimulation technique to automatically reconstruct Android malware behaviors. In: Proc. of the EuroSec. 2013.

[72] Octeau D, McDaniel P, Jha S, Bartel A, Bodden E, Klein J, Le Traon Y. Effective inter-component communication mapping in Android: An essential step towards holistic security analysis. In: Proc. of the Presented as Part of the 22nd USENIX Security Symp. (USENIX Security 2013). 2013. 543–558.

[73] Kirat D, Vigna G. Malgene: Automatic extraction of malware analysis evasion signature. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security. ACM Press, 2015. 769–780.

[74] Lindorfer M, Kolbitsch C, Comparetti PM. Detecting environment sensitive malware. In: Proc. of the Int'l Workshop on Recent Advances in Intrusion Detection. Springer-Verlag, 2011. 338–357.

[75] Fredrikson M, Jha S, Christodorescu M, Sailer R, Yan X. Synthesizing near-optimal malware specifications from suspicious behaviors. In: Proc. of the 2010 IEEE Symp. on Security and Privacy (SP). IEEE, 2010. 45–60.

[76] Kwon I, Im EG. Extracting the representative API call patterns of malware families using recurrent neural network. In: Proc. of the Int'l Conf. on Research in Adaptive and Convergent Systems. ACM Press, 2017. 202–207.

[77] Kolbitsch C, Comparetti PM, Kruegel C, Kirda E, Zhou X, Wang X. Effective and efficient malware detection at the end host. In: Proc. of the 18th Conf. on USENIX Security Symp. 2009. 351–366.

[78] Ding Y, Xia X, Chen S, Li Y. A malware detection method based on family behavior graph. Computers & Security, 2018,73: 73–86.

[79] Cho I, Kim T, Shim Y, Park H, Choi B, Im E. Malware similarity analysis using API sequence alignments. Journal of Internet Services and Information Security, 2014,4(4):103–114.

[80] Qiao Y, Yun X, Zhang Y. How to automatically identify the homology of different malware. In: Proc. of the 2016 IEEE Trustcom/ BigDataSE/I SPA. IEEE, 2016. 929–936.

[81] Vx Heaven Windows virus collection. 2016. https://archive.org/details/vxheaven-windows-virus-collection

[82] Fredrikson M, Jha S, Christodorescu M, Sailer R, Yan X. Synthesizing near-optimal malware specifications from suspicious behaviors. In: Proc. of the 2010 IEEE Symp. on Security and Privacy (Oakland 2010). 2010. 45–60.

[83] Nguyen MH, Nguyen DL, Nguyen XM, Quan TT. Auto-detection of sophisticated malware using lazy-blinding control flow graph and deep learning. Computer & Security, 2018,76:128–155.

[84] Fan M, Liu J, Luo X, Chen K, Tian Z, Zheng Q, Liu T. Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Trans. on Information Forensics and Security, 2018,13(8):1890–1905.

[85] Ojah L, Pathak A, Medhi S. SMS monitoring system for detecting premium SMS malware in smart phone. Int'l Journal, 2016,5(5): 56–59.

[86] Marastoni N, Continella A, Quarta D, Zanero S, Preda MD. GroupDroid: Automatically grouping mobile malware by extracting code similarities. In: Proc. of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop. ACM Press, 2017.

[87] Lee J, Lee S, Lee H. Screening smartphone applications using malware family signatures. Computers & Security, 2015,52: 234–249.

[88] Walenstein A, Lakhotia A. A transformation-based model of malware derivation. In: Proc. of the 2012 7th Int'l Conf. on Malicious and Unwanted Software (MALWARE). 2012. 17–25.

[89] Suarez-Tangil G, Tapiador JE, Peris-Lopez P, Blasco J. Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families. Expert Systems with Applications, 2014,41(4):1104–1117.

[90] Fan M, Liu J, Luo X, Chen K, Tian Z, Zhang X, Zheng Q, Liu T. Frequent subgraph based familial classification of android malware. In: Proc. of the 2016 IEEE 27th Int'l Symp. on Software Reliability Engineering (ISSRE). IEEE, 2016. 24−35.

[91] Qian YC, Peng GJ, Wang Y, Liang Y. Homology analysis of malicious code and family clustering. Computer Engineering and Applications, 2015,51(18):76−81 (in Chinese with English abstract).

[92] Krsul I, Spafford EH. Authorship analysis: Identitying the author of a program. Computer & Security, 1997,16(3):233−257.

[93] MacDonell SG, Gray AR, MacLennan G, MacLennan G, Sallis PJ. Software forensics for discriminating between program authors using case-based reasoning, feedforward neural networks and multiple discriminant analysis. In: Proc. of the 6th Int'l Conf. on Neural Information Processing (ICONIP'99). IEEE, 1999. 66−71.

[94] Lange RC, Mancoridis S. Using code metric histograms and genetic algorithms to perform author identification for software forensics. In: Proc. of the Annual Conf. on Genetic and Evolutionary Computation. ACM Press, 2007.

[95] Kothari J, Shevertalov M, Stehle E, Mancoridis S. A probabilistic approach to source code authorship identification. In: Proc. of the Information Technology (ITNG 2007). IEEE, 2007.

[96] Burrows S, Uitdenbogerd AL, Turpin A. Application of information retrieval techniques for source code authorship attribution. In: Proc. of the Int'l Conf. on Database Systems for Advanced Applications. Berlin, Heidelberg: Springer-Verlag, 2009. 699−713.

[97] Chen R, Hong L, Lu C, Deng W. Author identification of software source code with program dependence graphs. In: Proc. of the 2010 34th IEEE Annual Computer Software and Applications Conf. Workshops (COMPSACW). IEEE, 2010. 281−286.

[98] Caliskan-Islam A, Harang R, Liu A, Voss C, Narayanan A, Greenstadt R. De-anonymizing programmers via code stylometry. In: Proc. of the 24th USENIX Security Symp. (USENIX Security). Washington, 2015. 254−270.

[99] Alrabaee S, Shirani P, Debbabi M, Wang L. On the feasibility of malware authorship attribution. In: Proc. of the Int'l Symp. on Foundations and Practice of Security. Cham: Springer-Verlag, 2016. 256−272.

[100] Qiao YC, Yun XC, TUO YP, Zhang YZ. Fast reused code tracing method based on simhash and inverted index. Journal on Communications, 2016,37(11):104−113 (in Chinese with English abstract).

[101] Rosenblum N, Zhu X, Miller BP. Who wrote this code? Identifying the authors of program binaries. In: Proc. of the European Symp. on Research in Computer Security. Berlin, Heidelberg: Springer-Verlag, 2011. 172−189.

[102] Caliskan A, Yamaguchi F, Dauber E, Harang R, Rieck K, Greenstadt K, Narayanan A. When coding style survives compilation: De-anonymizing programmers from executables binaries. In: Proc. of the Network and Distributed Systems Security (NDSS) Symp. 2018. 1−15.

[103] Trend Micro Cyber Safety Solutions Tream. Confucius update: New tools and techniques, further connections with patchwork. 2018. https://blog.trendmicro.com/trendlabs-security-intelligence/confucius-update-new-tools-and-techniques-further-connections-with-patchwork/

[104] Rankin B. A brief history of malware—Its evolution and impact. 2018. https://www.lastline.com/blog/history-of-malware-its-evolution-and-impact/

[105] Fruhlinger J. Petya ransomware and NotPetya malware: What you need to know now. 2017. https://www.csoonline.com/article/3233210/ransomware/petya-ransomware-and-notpetya-malware-what-you-need-to-know-now.html

[106] Kortepeter D. Cerber ransomware: How it works and how to handle it. 2017. http://techgenix.com/cerber-ransomware/

[107] Gandotra E, Bansal D, Sofat S. Malware analysis and classification: A survey. Journal of Information Security, 2014,5(2):56−64.

[108] Next Generation Threats. 2013. https://www.threatprotectworks.com/Next-Generation-Threats.asp

[109] You I, Yim K. Malware obfuscation techniques: A brief survey. In: Proc. of the Int'l Conf. on Broadband, Wireless Computing, Communication and Applications. Fukuoka, 2010. 297−300.

[110] Rafique MZ, Chen P, Huygens C, Joosen W. Evolutionary algorithms for classification of malware families through different network behaviors. In: Proc. of the 2014 Annual Conf. on Genetic and Evolutionary Computation. ACM Press, 2014. 1167−1174.

[111] Symantec. The future of mobile malware. 2014. http://www.symantec.com/connect/blogs/future-mobile-malware

[112] Zuo LM, Liu EG, Xu BG, Tang PZ. The malicious code tribal grouping tribal group collects especially with analysis technology. Journal of Huazhong University of Science and Technology (Natural Science Edition), 2010,38(4):46−49 (in Chinese with English abstract).

[113] Prowell SJ, Sayre KD, Awad RL. Automatic clustering of malware variants based on structured control flow: U.S. Patent Application 15/172,884, 2016.

[114] Han XG, Qu W, Yao XX, Guo CY, Zhou F. Studies based on the texture fingerprint malicious code variety examination (method). Journal on Communications, 2017,35(8):125−136 (in Chinese with English abstract).

[115] Li Y, Zuo ZH. An overview of object-code obfuscation technologies. Journal of Computer Technology and Development, 2007, 17(4):125−127.

[116] LURHQ Threat Intelligence Group. Sobig.a and the spam you received today. Technical Report, LURHQ, 2003. http://www.lurhq.com/sobig.html/

[117] Hayes M, Walenstein A Lakhotia A. Evaluation of malware phylogeny modelling systems using automated variant generation. Journal in Computer Virology, 2009,5(4):335−343.

[118] Khoo WM, Lio P. Unity in diversity: Phylogenetic-inspired techniques for reverse engineering and detection of malware families. In: Proc. of the 2011 1st SysSec Workshop (SysSec). IEEE, 2011. 3−10.

[119] Shevchenko A. An overview of mobile device security. https://securelist.com/analysis/36059/an-overview-of-mobiledevice-security/

[120] Suarez-Tangil G, Stringhini G. Eight years of rider measurement in the Android malware ecosystem: Evolution and lessons learned. arXiv preprint arXiv:1801.08115, 2018.

[121] Argus Cyber Security Lab. Android malware evolution (2010-2016). 2018.

[122] McaFee. Mobile threat report. 2018. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf

[123] Suarez-Tangil G, Tapiador JE, Peris-Lopez P, Ribagorda A. Evolution, detection and analysis of malware for smart devices. IEEE Communications Surveys & Tutorials, 2014,16(2):961−987.

[124] Fan M, Liu J, Wang W, Li H, Tian Z, Liu T. Dapasa: Detecting Android piggybacked apps through sensitive subgraph analysis. IEEE Trans. on Information Forensics and Security, 2017,12(8):1772−1785.

[125] Wood P, Nahorney B, Chandrasekar K, Wallace S, Haley K. Internet security threat report. Report, Volume 19, Symantec Corporation, 2014.

[126] Wermke D, Huaman N, Acar Y, Reaves B, Traynor P, Fahl S. A large scale investigation of obfuscation use in google play. arXiv preprint arXiv:1801.02742, 2018.

[127] Faruki P, Fereidooni H, Laxmi V, Conti M, Gaur M. Android code protection via obfuscation techniques: Past, present and future directions. arXiv preprint arXiv:1611.10231, 2016.

[128] Karim ME, Walenstein A, Lakhotia A, Parida L. Malware phylogeny generation using permutations of code. 2005,1(1-2):13−23.

[129] Fang Y, Yu B, Tang Y, Liu L, Lu Z, Wang Y. A new malware classification approach based on malware dynamic analysis. In: Proc. of the Australasian Conf. on Information Security and Privacy. Cham: Springer-Verlag, 2017. 173−189.

[130] Cimitile A, Mercaldo F, Martinelli F, Nardone V, Santone A, Vaglini G. Model checking for mobile android malware evolution. In: Proc. of the 5th Int'l FME Workshop on Formal Methods in Software Engineering. IEEE Press, 2017. 24−30.

[131] Mercaldo F, Nardone V, Santone A, Visaggio CA. Download malware? No, thanks. How formal methods can block update attacks. In: Proc. of the 2016 IEEE/ACM 4th FME Workshop on Formal Methods in Software Engineering (FormaliSE). IEEE, 2016. 22−28.

[132] Canfora G, Medvet E, Mercaldo F, Visaggio CA. Detecting Android malware using sequences of system calls. In: Proc. of the 3rd Int'l Workshop on Software Development Lifecycle for Mobile (DeMobile 2015). New York: ACM Press, 2015. 13−20.

[133] Zhao BL, Meng X, Han J, Wang J, Liu FD. Homology analysis of malware based on graph. Journal on Communications, 2017, 38(Z2):86−93 (in Chinese with English abstract).

[134] Ge YW, Kang F, Peng XX. Homology analysis of malicious code based on dynamic BP neural network. Journal of Chinese Computer System, 2016,37(11):2527−2531 (in Chinese with English abstract).

[135] Faruki P, Laxmi V, Bharmal A, Gaur M, Ganmoor V. Androsimilar: Robust signature for detecting variants of android malware. Joural of Information Security and Applications, 2015,22:66−80.

[136] Kolter JZ, Maloof MA. Learning to detect and classify malicious executables in the wild. Journal of Machine Learning Research, 2006,7:2721−2744.

[137] Perdisci R, Lanzi A, Lee W. Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In: Proc. of the ACSAC. 2008. 301−310.

[138] Marko D, Atzeni S, Ugrina L, Rakamaric Z. Evaluation of Android malware detection based on system calls. In: Proc. of the ACM on Int'l Workshop on Security and Privacy Analytics (IWSPA). New Orlean, 2016. 1−8.

[139] Xiao X, Wang Z, Li Q, Xia S, Jiang Y. Back-propagation neural network on Markov chains from system call sequences: A new approach for detecting android malware with system call sequences. In: Proc. of the IET Information Security. 2017. 8−15.

[140] Qiao YC, Yun XC, Zhang YZ, Li SH. Based on transfer custom malicious code automation homology determination method. Acta Electronica Sinica, 2016,44(10):2410−2414 (in Chinese with English abstract).

[141] Ki Y, Kim E, Kim HK. A novel approach to detect malware based on API call sequence analysis. Int'l Journal of Distributed Sensor Networks, 2015,11(6):No.659101.

[142] Zhou H, Zhang W, Wei F, Chen Y. Analysis of Android malware family characteristic based on isomorphism of sensitive API call graph. In: Proc. of the 2017 2nd IEEE Int'l Conf. on Data Science in Cyberspace (DSC). IEEE, 2017. 319−327.

[143] Alam S, Riley R, Sogukpinar I, Carkaci N. Droidclone: Detecting android malware variants by exposing code clones. In: Proc. of the 2016 6th Int'l Conf. on Digital Information and Communication Technology and its Applications (DICTAP). IEEE, 2016. 79−84.

[144] Alam S, Horspool RN, Traore I. MAIL: Malware analysis intermediate language—A step towards automating and optimizing malware detection. In: Proc. of the SIN. ACM SIGSAC, 2013. 233−240.

[145] Aho AV, Lam MS, Sethi R, Ullman JD. Compilers: Principles, Techniques, and Tools. 2nd ed., Boston: Addison-Wesley Longman Publishing Co., Inc., 2006.

[146] Crussell J, Gibler C, Chen H. Attack of the clones: Detecting cloned applications on android markets. In: Proc. of the European Symp. on Research in Computer Security. Berlin, Heidelberg: Springer-Verlag, 2012. 37−54.

[147] Sun X, Zhongyang YB, Xin Z, Mao B, Xie L. Detecting code reuse in Android applications using component-based control flow graph. In: Proc. of the ICT. Springer-Verlag, 2014. 142−155.

[148] Chan PPK, Song WK. Static detection of Android malware by using permissions and API calls. In: Proc. of the 2014 Int'l Conf. on Machine Learning and Cybernetics (ICMLC). IEEE, 2014. 82−87.

[149] Zhu J, Guan Z, Yang Y, Yu L, Sun H, Chen Z. Permission-Based abnormal application detection for Android. In: Proc. of the Information and Communications Security. Springer-Verlag, 2012. 228−239.

[150] Faruki P, Laxmi V, Bharmal A, Gaur MS, Ganmoor V. AndroSimilar: Robust signature for detecting variants of Android malware. Journal of Information Security and Applications, 2015,22:66−80.

[151] Liu X, Liu J, Wang W. Exploring sensor usage behaviors of Android applications based on data flow analysis. In: Proc. of the 2015 34th IEEE Int'l Performance Computing and Communications Conf. (IPCCC). IEEE, 2015. 1−8.

[152] Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K. DREBIN: Effective and explainable detection of Android malware in your pocket. In: Proc. of the 21th Annual Network and Distributed System Security Symp. (NDSS). 2014. 23−26.

[153] Roussev V. Building a better similarity trap with statistically improbable features. In: Proc. of the 2009 42nd Hawaii Int'l Conf. on System Sciences (HICSS 2009). IEEE, 2009. 1−10.

[154] Wang C, Qin Z, Zhang J, Yin H. A malware variants detection methodology with an opcode based feature method and a fast density based clustering algorithm. In: Proc. of the 2016 12th Int'l Conf. on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD). IEEE, 2016. 481−487.

[155] Wu L, Ping R, Ke L, Hai D. Behavior-Based malware analysis and detection. In: Proc. of the 2011 1st Int'l Workshop on Complexity and Data Mining (IWCDM). New York: IEEE, 2011. 39−42.

[156] Santos I, Penya YK, Devesa J, Bingas PG. *N*-grams-based file signatures for malware detection. In: Proc. of the 2009 Int'l Conf. on Enterprise Information Systems (ICEIS), Volume AIDSS. 2009. 317−320.

[157] A biologically inspired immune system for computers. In: Proc. of the Artificial Life IV: 4th Int'l Workshop on the Synthesis and Simulation of Living Systems. MIT Press, 2011. 130−139.

[158] Kolosnjaji B, Zarras A, Webster G, Eckert C. Deep learning for classification of malware system call sequences. In: Proc. of the Australasian Joint Conf. on Artificial Intelligence. Cham: Springer-Verlag, 2016. 137−149.

[159] Dahl GE, Stokes JW, Deng L, Yu D. Large-scale malware classification using random projections and neural networks. In: Proc. of the 2013 IEEE Int'l Conf. on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2013. 3422−3426.

[160] Fang Y, Yu B, Tang Y, Liu L, Lu Z, Wang Y. A new malware classification approach based on malware dynamic analysis. In: Proc. of the Australasian Conf. on Information Security and Privacy. Cham: Springer-Verlag, 2017. 173−189.

[161] Aoyama H. On the chi-square test for weighted samples. Annals of the Institute of Statistical Mathematics, 1953,5(1):25−28.

[162] Cesare S, Xiang Y, Member S. Control flow-based malware variant detection. IEEE Trans. on Dependable Secure Computing, 2014,11(4):304−317.

[163] Awad RA, Sayre KD. Automatic clustering of malware variants. In: Proc. of the 2016 IEEE Conf. on Intelligence and Security Informatics (ISI). IEEE, 2016. 298−303.

[164] Hanna S, Huang L, Wu E, Li S, Chen C, Song D. Juxtapp: A scalable system for detecting code reuse among Android applications. In: Proc. of the 9th Conf. on Detection of Intrusions and Malware & Vulnerability Assessment. 2012.

[165] Allix KFT, Jerome BQ, Klein J, State R, Traon YL. Empirical assessment of machine learning-based malware detectors for Android. Empir Software Engineering, 2016,21:183−211.

[166] The Phrack Staff. "Phrack" 14(68). 2012. http://phrack.org/issues/68/1.html

[167] Xu Z, Ren K, Qin S, Craciun F. CDGDroid: Android malware detection based on deep learning using CFG and DFG. In: Proc. of the Int'l Conf. on Formal Engineering Methods. Cham: Springer-Verlag, 2018. 177−193.

[168] Crussell J, Gibler C, Chen H. Andarwin: Scalable detection of semantically similar Android applications. In: Proc. of the European Symp. on Research in Computer Security. Springer, Berlin, Heidelberg, 2013. 182−199.

[169] Yang W, Li J, Zhang Y, Li Y, Shu J, Gu D. APKLancet: Tumor payload diagnosis and purification for Android applications. In: Proc. of the 9th ACM Symp. on Information, Computer and Communications Security. ACM Press, 2014. 483−494.

[170] Karbab EMB, Debbabi M, Mouheb D. Fingerprinting Android packaging: Generating DNAs for malware detection. Digital Investigation, 2016,18:S33−S45.

[171] Cesare S, Xiang Y, Zhou W. Control flow-based malware variant detection. IEEE Trans. on Dependable and Secure Computing, 2014,11(4):307−317.

[172] Kim J, Kim TG, Im EG. Structural information based malicious app similarity calculation and clustering. In: Proc. of the 2015 Conf. on Research in Adaptive and Convergent Systems. ACM Press, 2015. 314−318.

[173] Feizollah A, Anuar NB, Salleh R, Amalina F. Comparative study of *k*-means and mini batch *k*-means clustering algorithms in Android malware detection using network traffic analysis. In: Proc. of the 2014 Int'l Symp. on Biometrics and Security Technologies (ISBAST). IEEE, 2014. 193−197.

[174] Niu Z, Qin Z, Zhang J, Yin H. Malware variants detection using density based spatial clustering with global opcode matrix. In: Proc. of the Int'l Conf. on Security, Privacy and Anonymity in Computation, Communication and Storage. Cham: Springer-Verlag, 2017. 757−766.

[175] Xu XL, Yun XC, Zhou YL, Kang XB. Online analytical model of massive malware based on feature clusting. Journal on Communications, 2013,34(8):146−153 (in Chinese with English abstract).

[176] Moran N, Bennett J. Supply chain analysis: From quarter master to sunshop. Technical Report, Fire Eye Labs, 2013. https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-malware-supply-chain.pdf

[177] Antiy CERT. White elephant dance—Cyber attacks from the south Asian subcontinent. 2016 (in Chinese). https://mp.weixin.qq.com/s/XGZGaylS1B84v-NWaevLEw

[178] Mandiant. Tracking malware import hashing. 2014. https://www.mandiant.com/blog/tracking-malware-import-hashing

[179] Bencsáth B, Pék G, Buttyán L, *et al*. Duqu: A stuxnet-like malware found in the wild. CrySyS Lab Technical Report, Vol.14, 2011. 1−60. https://www.crysys.hu/publications/files/bencsathPBF11duqu.pdf

[180] MCAFEE. Android malware appears linked to Lazarus cybercrime group. 2017. https://securingtomorrow.mcafee.com/mcafee-labs/android-malware-appears-linked-to-lazarus-cybercrime-group/

[181] Li DH. Malicious sample analysis handbook—Traceability. 2018 (in Chinese). http://blog.nsfocus.net/trace-source/

[182] Venustech. Hedwig (Haiwei) Organization analysis report: A game that cannot be ended. 2016 (in Chinese). http://www.freebuf.com/news/92945.html

[183] 360CERT. QQKEY hacking Trojan new variant traceability analysis. 2018 (in Chinese). https://www.anquanke.com/post/id/147591

[184] 360CERT. Take advantage of a variety of office OLE features for sample analysis and traceability. 2018 (in Chinese). https://www.anquanke.com/post/id/101722

[185] FireEye iSIGHT Intelligence. APT28: At the center of the storm. 2017. https://www.fireeye.com/blog/threat-research/2017/01/apt28_at_the_center.html

[186] 360CERT. 2017 China advanced persistent threat (APT). 2018. http://www.360doc.com/content/18/0227/06/43535834_732761511.shtml

[187] AVLTeam. Antiy mobile security's "Dvmap" Android malware analysis report. 2017. http://www.freebuf.com/articles/terminal/137015.html

[188] Peng GJ, Wang TG, *et al*. Unknown Trojan Detection System Based on Host Behavior Perception. [referred to as: XDetector]v1.0. [Registration number:2014SR113893], China, 2014 (in Chinese).

[189] Liang Y, Fu JM, Peng GJ, Peng BC. S-tracker: Shellcode detection method based on stack anomaly. Journal of Huazhong University of Science and Technology (Natural Science Edition), 2014,42(11):39−46 (in Chinese with English abstract).

[190] Sha LT, Fu JM, Chen J, Huang SY. A sensitivity measurement for sensitive information processing. Journal of Computer Research and Development, 2014,51(5):1050−1060 (in Chinese with English abstract).

[191] Fu JM, Sha LT, Li PW, Peng GJ. Kernel data active protection method using hardware virtualization. Journal of Sichuan University (Enigineering Science Edition), 2014,46(1):8−13 (in Chinese with English abstract).

[192] Cheng BL, Ming J, Fu JM, Peng GJ, Chen T, Zhang XS, Marion JY. Towards paving the way for large-scale windows malware analysis: Generic binary unpacking with orders-of-magnitude performance boost. In: Proc. of the 25th ACM Conf. on Computer and Communications Security. Toronto, 2018.

[193] Lin YD, Lai YC, Chen CH, Tsai HC. Identifying Android malicious repackaged applications by thread-grained system call sequences. Computers and Security, 2013,39:340−350.

[194] Yang C, Xu Z, Gu G, Yegneswaran V, Porras P. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in Android applications. In: Proc. of the Computer Security (ESORICS 2014). Springer-Verlag, 2014. 163−182.

[195] Chen K, Wang P, Lee Y, *et al*. Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google-play scale. In: Proc. of the USENIX Security Symp. 2015. 658−674.

[196] Roman Unuchek. Dvmap: The first Android malware with code injection. 2017. https://securelist.com/dvmap-the-first-android-malware-with-code-injection/78648/2017.8

[197] Villanueva MJ. JS_POWMET.DE. 2017. https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/js_powmet.de

[198] Peng GJ, Tao F. Malicious Code Forensics. Beijing: Science Press, 2009 (in Chinese).

[199] Fu JM, Peng GJ, Zhang HG. Computer Virus Analysis and Confrontation. 2nd ed., Wuhan: Wuhan University Press, 2009 (in Chinese).

[200] Qiao YC, Yun XC, Zhang YZ. Fast reused function retrieval method based on simhash and inverted index. In: Proc. of the 2016 IEEE Trustcom/BigDataSE/I SPA. IEEE, 2016. 937−944.

附中文参考文献:

[17] AVLTeam.移动端动端 C#病毒"东山再起",利用知名应用通信实现远控隐私窃取.2017.12−28. https://bbs.pediy.com/thread-223596.htm

[24] 360 天眼实验室.OceanLotus(海莲花)APT 报告摘要.2015. http://blogs.360.cn/blog/oceanlotus-apt/

[25] 孔德光,谭小彬,奚宏生,宫涛,帅建梅.提升多维特征检测迷惑恶意代码.软件学报,2010,22(3):522−533. http://www.jos.org.cn/1000-9825/3727.htm [doi: 10.3724/SP.J.1001.2011.03727]

[31] 王蕊,冯登国,杨轶,苏璞睿.基于语义的恶意代码行为特征提取及检测方法.软件学报,2012,23(2):378−393. http://www.jos.org.cn/1000-9825/3953.htm [doi: 10.3724/SP.J.1001.2012.03953]

[46] 彭国军,傅建明,梁玉.软件安全.武汉:武汉大学出版社,2015.262−264.

[55] 邵思豪,高庆,马森,段富尧,马骁,张世琨,胡津华.缓冲区溢出漏洞分析技术研究进展.软件学报,2018,29(5):1179−1198. http://www.jos.org.cn/1000-9825/5504.htm [doi: 10.13328/j.cnki.jos.005504]
[57] 安天发布:潜伏的象群——越过世界屋脊的攻击.2017. https://mp.weixin.qq.com/s/nnrDVgH-mEzZ8cytaUEUVg
[58] 360 追日团队,360CERT,3602 天眼实验室.2017 中国高级持续性威胁(APT)研究报告.2018. https://mp.weixin.qq.com/s/Su3IFORhc55Py7oXOn32ng
[91] 钱雨村,彭国军,王滢,梁玉.恶意代码同源性分析及家族聚类.计算机工程与应用,2015,51(18):76−81.
[100] 乔延臣,云晓春,庹宇鹏,张永铮.基于 simhash 与倒排索引的复用代码快速溯源方法.通信学报,2016,37(11):104−113.
[112] 左黎明,刘二根,徐保根,汤鹏志.恶意代码族群特征提取与分析技术.华中科技大学学报(自然科学版),2010,38(4):46−49.
[114] 韩晓光,曲武,姚宣霞,郭长友,周芳.基于纹理指纹的恶意代码变种检测方法研究.通信学报,2017,35(8):125−136.
[133] 赵炳麟,孟曦,韩金,王婧,刘福东.基于图结构的恶意代码同源性分析.通信学报,2017,38(Z2):86−93.
[134] 葛雨玮,康绯,彭小详.基于动态 BP 神经网络的恶意代码同源性分析.小型微型计算机系统,2016,37(11):2527−2531.
[140] 乔延臣,云晓春,张永铮,李书豪.基于调用习惯的恶意代码自动化同源判定方法.电子学报,2016,44(10):2410−2414.
[175] 徐小琳,云晓春,周勇林,康学斌.基于特征聚类的海量恶意代码在线自动分析模型.通信学报,2013,34(8):146−153.
[177] 安天实验室安全研究与应急处理中心.白象的舞步——来自南亚次大陆的网络攻击.2016. https://mp.weixin.qq.com/s/XGZGaylS1B 84v-NWaevLEw
[181] 李东宏.恶意样本分析手册——溯源篇.2018. http://blog.nsfocus.net/trace-source/
[182] 启明星辰.海德薇(Hedwig)组织分析报告:一场无法结束的博弈.2016. http://www.freebuf.com/news/92945.html
[183] 360CERT.QQKEY 盗号木马新型变种溯源分析.2018. https://www.anquanke.com/post/id/147591
[184] 360CERT.利用了多种 Office OLE 特性的免杀样本分析及溯源.2018. https://www.anquanke.com/post/id/101722
[188] 彭国军,王泰格,等.基于主机行为感知的未知木马检测系统[简称:XDetector]v1.0[登记号:2014SR113893],中国,2014.
[189] 梁玉,傅建明,彭国军,彭碧琛.S-Tracker:基于栈异常的 shellcode 检测方法.华中科技大学学报(自然科学版),2014,42(11):39−46.
[190] 沙乐天,傅建明,陈晶,黄诗勇.一种面向敏感信息处理的敏感度度量方法.计算机研究与发展,2014,51(5):1050−1060.
[191] 傅建明,沙乐天,李鹏伟,彭国军.一种采用硬件虚拟化的内核数据主动保护方法.四川大学学报(工程科学版),2014,46(1):8−13.
[198] 彭国军,陶芬.恶意代码取证.北京:科学出版社,2009.
[199] 傅建明,彭国军,张焕国.计算机病毒分析与对抗.第 2 版,武汉:武汉大学出版社,2009.

宋文纳(1989－),女,河南平顶山人,博士生,主要研究领域为信息安全.

张焕国(1945－),男,教授,博士生导师,CCF 高级会员,主要研究领域为信息安全,可信计算,密码学.

彭国军(1979－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为恶意代码检测,可信软件.

陈施旅(1994－),男,硕士,主要研究领域为信息安全.

傅建明(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为系统安全,网络安全.