

一种基于补丁比对和静态污点分析的漏洞定位技术研究

达小文, 毛俐旻, 吴明杰, 郭敏

(北京计算机技术及应用研究所, 北京 100854)

摘要: 目前对于开源软件的漏洞定位分析较为缺乏, 且缺少一种自动化的快速定位方法。针对这些不足, 文章提出一种基于补丁比对和静态污点分析的漏洞定位方法。该方法通过分析大量开源软件的缓冲区溢出错误的实例, 提取 6 种缓冲区错误的漏洞定位模型; 通过将补丁比对和污点传播结合, 生成污点传播路径图; 将补丁源码的污点传播路径图与定位模型匹配以定位某小块代码, 采用污点查找精确定位漏洞所在行。

关键词: 漏洞定位; 补丁比对; 污点分析; 缓冲区错误

中图分类号: TP393 **文献标识码:** A **文章编号:** 1671-1122 (2017) 09-0005-05

中文引用格式: 达小文, 毛俐旻, 吴明杰, 等. 一种基于补丁比对和静态污点分析的漏洞定位技术研究 [J]. 信息安全, 2017 (9): 5-9.

英文引用格式: DA Xiaowen, MAO Limin, WU Mingjie, et al. Research on a Vulnerability Location Technology Based on Patch Matching and Static Taint Analysis[J]. Netinfo Security, 2017(9):5-9.

Research on a Vulnerability Location Technology Based on Patch Matching and Static Taint Analysis

DA Xiaowen, MAO Limin, WU Mingjie, GUO Min

(Beijing Institute of Computer Technology and Applications, Beijing 100854, China)

Abstract: At present, there is a lack of the analysis for vulnerabilities location in open source software and a lack of an automatic method for fast locating the vulnerabilities. To address these issues, this paper proposes a vulnerabilities location method based on patch matching and the static taints analysis. By analyzing a large number of buffer errors instances of open source software, six vulnerabilities location models of buffer errors are extracted. By combining patch matching with taint propagation, a taint propagation path graph is generated. Match the taint propagation graph of patched source with the location models to locate a small block of code, which then locates vulnerabilities code rows accurately by searching for taints.

Key words: vulnerability location; patch matching; taint analysis; buffer errors

0 引言

近年来, 漏洞造成的危害和影响越来越大。2017 年 5 月爆发的 WannaCry 勒索病毒就是一个很好的例子。该病毒利用微软“永恒之蓝”漏洞 (MS17-010)^[1] 发起的攻击导致至少 150 个国家、30 万用户中招, 是继 MS-08-067 漏洞以来微软 SMB 服务的又一危害巨大的漏洞。目前 CVE 和 NVD 收录的漏洞数量达到 10 万余条, 但其中可进行验证的漏洞较少。对漏洞软件进行源码级漏洞定位, 可以为漏洞验证程序 (poc) 的编写提供重要依据。使用编写的 poc 可以验证漏洞, 并确认

收稿日期: 2017-8-1

作者简介: 达小文 (1989—), 男, 湖南, 助理工程师, 硕士, 主要研究方向为信息安全; 毛俐旻 (1977—), 女, 湖北, 高级工程师, 博士, 主要研究方向为信息安全; 吴明杰 (1987—), 男, 江苏, 工程师, 硕士, 主要研究方向为信息安全; 郭敏 (1991—), 女, 山东, 助理工程师, 硕士, 主要研究方向为信息安全。

通信作者: 达小文 drxiaowen@qq.com

漏洞的危害程度。因此,在出现针对高危漏洞的攻击之前,能够对这些漏洞做好正确的更新或防护。

关于漏洞定位技术,国内外对二进制程序的缓冲区溢出^[2,3]、整数溢出^[4,5]产生原因进行了分析和定位。文献[6]基于二进制平台对应用程序的缓冲区漏洞进行分析和定位,该方法具有较大的局限性,仅能分析在 Windows 上运行的程序。文献[7]比较补丁前后二进制文件的控制流图,根据它们的结构性差异定位漏洞,该方法对于不在差异性位置的漏洞不适用。文献[8]通过比对 EFLAGS 标志位差异,结合污点分析方法,定位整数溢出漏洞。文献[9]利用程序运行时外部输入数据着色的方法定位漏洞触发处,该方法对由于内部数据错误引起的整数溢出定位不适用。文献[10]提出用模型推理方法分析开源软件缓冲区溢出陷入点,但是需要分析的源码范围过大,影响定位速度和精度。

从以上分析可以看出,目前针对开源软件漏洞的定位范围较大,对开源软件的源码级漏洞定位研究较少。本文针对开源软件的已知漏洞进行分析,提出一种将补丁比对和静态污点分析结合的方法,精确定位开源软件的漏洞产生位置。

1 基于补丁比对和静态污点分析的漏洞定位

本文定位的对象为 C 语言类软件的缓冲区错误(CWE-119),缓冲区错误漏洞一直是行业中分析和关注的重点。针对缓冲区错误漏洞的基于补丁比对和静态污点分析的漏洞定位方法的基本思想如下:

1) 对大量缓冲区错误进行分析,抽象出缓冲区错误的漏洞定位模型。2) 通过补丁比对将定位范围缩小到一个或几个源文件中的某段代码。3) 从补丁中确定静态污点分析的污点,将污点传播路径图与事先提取的漏洞定位模型比对,进行漏洞的源码行号定位。

1.1 漏洞原理

缓冲区错误是指软件在对内存位置进行读或写操作时,超出了缓冲区的边界。缓冲区错误产生的原因^[11]主要包括函数未检查目标地址缓冲区长度、函数未检查源数据长度、缓冲索引变量未检查即使用、索引变量的数字上溢(下溢)、字符串尾部未指明空字符、经过运算后指针指向

无效位置、条件语句内指针的加减操作未正确检查缓冲区的结尾、使用动态的结束指针来结束循环等。由以上原因可知,缓冲区错误在代码中体现为存在一个与缓冲区操作有关的变量或函数^[12]。

1.2 技术原理

1) 补丁比对。补丁比对是指对软件补丁前和补丁后的源码进行相似性比对,得出其中的差异性部分,将该差异性部分定义为 Diff。补丁比对一般使用相似性算法匹配,查找补丁前后不同代码行,将删除的源码在行前用“-”标记,增加的源码在行前用“+”标记。补丁比对将定位的范围缩小到一段代码中,从一段代码或其上下文中寻找漏洞产生点,而不需要在整个源文件中定位漏洞。

2) 污点。污点是指代码中可能产生漏洞的可疑变量。由于污点特性与缓冲区错误产生原因相似,因此将其用于分析缓冲区错误。

3) 静态污点分析。静态污点分析^[13]是指不运行且不修改代码,根据程序变量间的数据依赖关系分析污点在源代码中的路径。静态污点分析与动态污点分析^[14,15]最大的不同是,前者不需要运行程序以及实时跟踪污点的传播情况。

4) 静态污点传播。静态污点传播是指污点在代码中扩散传播的过程。本文将污点在代码中的传播分为两种情况:

(1) 污点从变量 p 传递到 p' ,从 p 到 p' 的传递可能由一次赋值操作引起。

(2) 一种 p 对 p' 的约束关系,没有明显的传递关系。例如,for($p' < p$); p' 被限制在 p 的范围内,在此也看作是一种传播。

上述两种情况对于污点传播路径没有实质性影响。

为分析源码中污点传播的路径,将源代码看作由若干个分支体组成,一个分支体包含分支条件和基本块两个部分,其中分支条件包括 if-else 语句、switch-case 语句、断言语句、return 语句、goto 语句、三目运算符等,基本块包括变量的传递、函数的赋值、函数的调用语句等。

1.3 定位模型

对 100 余个 CVE 漏洞中缓冲区错误补丁实例进行分析,根据它们的修补方式提取出 6 种漏洞定位模型(如表 1~ 表 6)

所示)。这6种模型适用于定位大部分缓冲区错误漏洞。

模型中圆圈表示源码分支条件；椭圆表示一个完整的分支体；方框表示基本块；“+”表示源码的修改操作，“+”表示源码的增加，“-”表示源码的删除；“(p)”表示与缓冲区有关的变量；“(p’)”表示“(p)”经过污点传播后的变量，p和p’大部分情况下相同（即污点没有经过传播），p和p’中可能含有多个变量；func(p)和func(p’)表示函数中包含变量p和p’；“非(p’)”表示与变量无关的语句；“↓”表示两个基本块相连，且在一个分支体内；⌋表示两个基本块不在一个分支体内；“→”表示漏洞定位的位置。

1) 模型 1

当修复方式为修改与缓冲区变量有关的条件语句时，其补丁意图是通过修改判定条件以改变后续的缓冲区内容，则所用漏洞定位模型是定位到该分支体的基本块中的缓冲区有关语句。

该模型如表1左侧所示，右侧为典型实例漏洞 CVE-2014-3158 的补丁代码。用模型1表示 CVE-2014-3158 时，p、p’都为 len 变量，可知应定位 word[len]=value 语句。

表 1 模型 1 及漏洞补丁实例

模型 1	CVE-2014-3158 的补丁代码
	<pre> @@@ -1289,9 +1289,10 @@ getword(f, word, newlinep, filename) - if (len < MAXWORDLEN-1) + if (len < MAXWORDLEN) { + word[len] = value; + ++len; + ++len; + } + if (!got) + c = getc(f); </pre>

2) 模型 2

当修复方式为修改某个分支体中的基本块语句，并增加一个完整分支体时，其补丁意图是改变原来不合理的函数调用，并通过增加分支体来控制缓冲区相关变量，则所用漏洞定位模型是定位到被修改的基本块部分。

该模型如表2左侧所示，其中椭圆表示增加的完整的分支体；右侧为实例漏洞 CVE-2013-1929 的补丁代码。用模型2表示 CVE-2013-1929 时，p和p’都为 len 变量，定位源码位置为表2右侧代码中加下划线的部分。

3) 模型 3

当修复方式为增加一个分支体或删除原有分支体后增加新的分支体时，其补丁意图是改变代码的原有路径，则所用漏洞定位模型是在分支体后的基本块中查找与函数

表 2 模型 2 及漏洞补丁实例

模型 2	CVE-2013-1929 的补丁代码
	<pre> @@@ -14604,8 +14604,11 @@ static void tg3_read_vpd(struct tg3 *tp) + if (j + len > block_end) + goto partno; - memcpy(tp->fw_ver, &vpd_data[j], len); - strcat(tp->fw_ver, "bc ", vpdlen - len - 1); + if (len >= sizeof(tp->fw_ver)) + len = sizeof(tp->fw_ver) - 1; + memset(tp->fw_ver, 0, sizeof(tp->fw_ver)); + snprintf(tp->fw_ver, sizeof(tp->fw_ver), "%s bc ", len, + &vpd_data[j]); } partno: </pre>

调用有关的语句，如与内存操作有关的函数调用语句或其他调用语句。

该模型如表3左侧所示，其中椭圆表示增加的分支体。模型3适用于 CVE-2013-1796、CVE-2014-3181、CVE-2016-2385、CVE-2016-5829、CVE-2013-6381、CVE-2013-7023 等漏洞。表3右侧为 CVE-2014-3181 的补丁代码。用模型3表示 CVE-2014-3181 时，p为“npoints”，p经过传播后可知p’为“ii”，func(p’)为 magicmouse_emit_touch(ii,·)，因此应定位至 magicmouse_emit_touch(msc, ii, data + ii * 9 + 4) 语句。

表 3 模型 3 及漏洞补丁实例

模型 3	CVE-2014-3181 的补丁代码
	<pre> @@@ -290,6 +290,11 @@ static int magicmouse_raw_event(struct hid_device *hdev, + if (size < 4 ((size - 4) % 9) != 0) + return 0; + npoints = (size - 4) / 9; + if (npoints > 15) { + hid_warn(hdev, "invalid size value (%d) for TRACKPAD_REPORT_ID", + size); + return 0; + } + msc->mtouches = 0; + for (ii = 0; ii < npoints; ii++) + magicmouse_emit_touch(msc, ii, data + ii * 9 + 4); </pre>

4) 模型 4

当修复方式为修改基本块中单一函数调用语句，且该函数调用语句中含有与缓冲区操作有关的变量时，则所用漏洞定位模型是直接定位该修改语句。

该模型如表4左侧所示，适用于 CVE-2009-0023、CVE-2010-1623、CVE-2016-4478 等漏洞。表4右侧为 CVE-2016-4478 的补丁代码。用模型4表示 CVE-2016-4478 时，可知其中p为“s”，应定位 memcpy(outbuffer, s->str, XMLRPC_BUFSIZE) 语句。

表 4 模型 4 及漏洞补丁实例

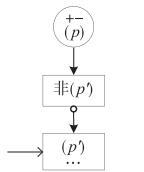
模型 4	CVE-2016-4478 的补丁代码
	<pre> @@@ -777,8 +777,9 @@ void xmlrpc_char_encode(char *outbuffer, const char *s1) + s->append_char(s, c); } } + s->append_char(s, 0); - memcpy(outbuffer, s->str, XMLRPC_BUFSIZE); + strncpy(outbuffer, s->str, XMLRPC_BUFSIZE); } } </pre>

5) 模型 5

当修复方式为修改分支条件语句, 且分支体中的基本块为与条件变量不相关的语句时, 则所用漏洞定位模型是定位到分支体外下一条语句。

该模型如表 5 左侧所示。其中, “非(p')”属于分支体中的基本块, 且该基本块与变量 p' 无关; “非(p')”与 “(p')”用带空心圆的箭头连接, 表示 “非(p')”所在基本块位于相邻分支体外。模型 5 适用于 CVE-2011-1180、CVE-2016-8658、CVE-2013-7016 等漏洞。表 5 右侧为 CVE-2016-8658 的补丁代码。用模型 5 表示 CVE-2016-8658 时, 可知 p 为 “ssid_ie”, 该变量没有经过传播, p' 也为 “ssid_ie”, 因此定位 memcpy(ssid_ie.SSID, ssid_ie->data, ssid_ie->len) 语句。

表 5 模型 5 及漏洞补丁实例

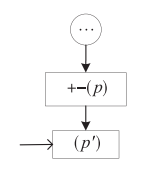
模型 5	CVE-2016-8658 的补丁代码
	<pre>@@ -4527,7 +4527,7 @@ brcmf_cfg80211_start_ap(struct wiphy *wiphy, struct net_device *ndev, (u8 *)&settings->beacon.head[ie_offset], settings->beacon.head_len - ie_offset, WLAN_EID_SSID); - + if (!ssid_ie) + if (ssid_ie ssid_ie->len > IEEE80211_MAX_SSID_LEN) return -EINVAL; memcpy(ssid_ie.SSID, ssid_ie->data, ssid_ie->len);</pre>

6) 模型 6

当修复方式为修改基本块中与缓冲区变量有关的赋值语句时, 则所用漏洞定位模型是定位至该赋值语句后含有这些缓冲区变量的有关函数调用语句。

该模型如表 6 左侧所示, 其中两个基本块在一个条件分支体内, 表 6 右侧为 CVE-2013-7022 的补丁代码。用模型 6 表示 CVE-2013-7022 时, 可知 p 为 “ $c->framebuf_stride$ ”、“ $aligned_height$ ”两个变量, 经过传播后 p' 与 p 相同, 因此定位 $c->framebuf = av_mallocz(c->framebuf_stride * aligned_height)$ 语句。

表 6 模型 6 及漏洞补丁实例

模型 6	CVE-2013-7022 的补丁代码
	<pre>@@ -443,8 +443,8 @@ static int g2m_init_buffers(G2MContext *c) int aligned_height; if (!c->framebuf c->old_width < c->width c->old_height < c->height) { - c->framebuf_stride = FFALIGN(c->width * 3, 16); + aligned_height = FFALIGN(c->height, 16); + c->framebuf_stride = FFALIGN(c->width + 15, 16) * 3; + aligned_height = c->height + 15; av_free(c->framebuf); c->framebuf = av_mallocz(c->framebuf_stride * aligned_height); if (!c->framebuf)</pre>

1.4 定位过程

由以上分析可知, 本文提出的漏洞定位步骤如下:

1) 对某个版本软件补丁前和补丁后源文件进行对比, 生成 Diff 文件。

2) 对 Diff 文件中涉及的变量 $\{para_1, para_2, \dots, para_n\}$ 进行分析, 选取污点 (污点数 $N \leq 5$)。污点的选取有 3 个原则: (1) 选取删除和增加的行中共有的变量; (2) 选取赋值左侧的变量; (3) 选取 if 条件语句中被约束的变量。依据这 3 个原则即可选取合适的污点。

3) 对补丁源码及其上下文使用静态污点分析生成污点传播路径集 $\{cfg_1, cfg_2, \dots, cfg_m\}$, 由路径集组合成污点传播路径图。

4) 查找污点传播路径图中是否有与某个缓冲区错误的定位模型相匹配的一段路径。

5) 若查找到与定位模型相同或相似的某段路径, 则根据污点所在位置精确定位漏洞所在代码行, 标记出定位结果。

图 1 为由某缓冲区错误漏洞的补丁源码及其上下文生成的部分污点传播路径图, 图中带标号的圆圈为分支条件, 带箭头的直线为基本块, 灰色小圆圈表示污点。若由步骤 1) 可知分支条件 4 处为 Diff 文件中修改的部分, 根据步骤 4), 发现其与模型 1 相匹配 (修改的部分都为条件语句), 且污点传输至相邻的基本块中, 则漏洞定位为基本块 4 → 7。确定漏洞具体所在行需要查找污点所在行, 定位基本块 4 → 7 中含有该污点的语句。

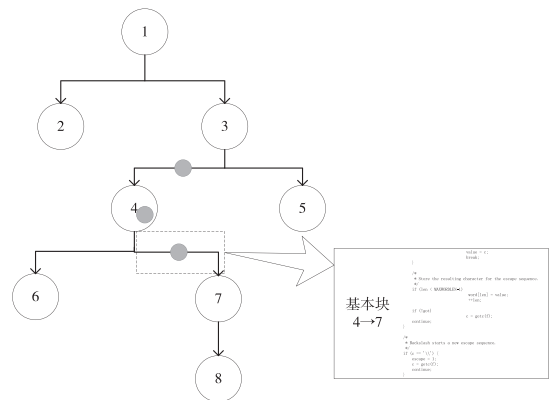


图 1 某补丁源码的部分污点传播路径图

2 系统设计

本文漏洞定位系统主要包括补丁比对、静态污点分析、定位模型匹配和污点查找 4 个模块, 如图 2 所示。系统输入为补丁前后的软件源码, 输出为定位的源码行号, 主要

接口设计为调用现有比对工具或静态污点分析工具的过程。定位模型采用树图的形式($G=(V,E)$)进行描述和存储,后续如果定位其他类别的漏洞,只需将该定位模型添加至定位模型库中即可,系统扩展性良好。

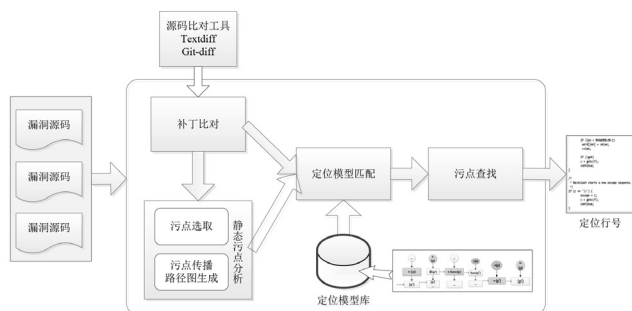


图2 基于补丁比对和静态污点分析的漏洞定位系统

1) 补丁比对

补丁比对模块功能是将存在漏洞的某软件源码和补丁后的源码对比,得到带有差异性标记的 Diff 文件,并将结果输入给静态污点分析模块和定位模型匹配模块。目前现有的源码比对工具有开源工具 DiffMerge、Textdiff、Meld、Git diff 等。本文系统调用 Textdiff 或 Git diff 进行辅助设计。

2) 静态污点分析

静态污点分析模块功能是结合 Diff 文件对源码做词法分析和语法分析,生成污点传播路径图,并将污点传播路径图作为定位模型匹配模块的输入。静态污点分析模块需要完成的工作有:(1)根据 Diff 文件及 1.4 节中的污点选取原则确定污点;(2)根据污点源码生成污点传播路径图。

3) 定位模型匹配

定位模型匹配模块的功能是根据 1.3 节中的定位模型,查找与由静态污点分析模块生成的污点传播路径图匹配的模型,若找到匹配的模型,则根据模型定位源码的某一个块,并将结果输入给污点查找模块。定位模型匹配依赖于定位模型,每一类漏洞有多个定位模型,定位模型的完善与否是定位漏洞的关键。

4) 污点查找

污点查找模块功能是在补丁源码及其上下文中查找污点所在行,根据定位模型匹配结果,定位具体源码行号。

3 结束语

本文提出一种开源软件的缓冲区错误漏洞定位方法,

并针对该方法提出了一种扩展性良好的漏洞定位系统框架。该方法的难点在于漏洞定位模型的提取,定位模型能够使人工分析过程用机器来实现。目前该软件处于开发阶段,未来将对其进行测试和验证,并根据测试结果修正定位模型,完善系统。●(责编 马珂)

参考文献:

- [1] 微软安全技术中心. Microsoft 安全公告 MS17-010 - 严重 [EB/OL]. <https://technet.microsoft.com/zh-cn/library/security/MS17-010>, 2016-5-10.
- [2] GAO Fenguan, WANG Linzhang, LI Xuandong. BovInspector: Automatic Inspection and Repair of Buffer Overflow Vulnerabilities[C]//ACM.Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, September 3-7, 2016. Singapore. New York: ACM, 2016: 786-791.
- [3] PADMANABHUNI B M, TAN H B K. Light-weight Rule-based Test Case Generation for Detecting Buffer Overflow Vulnerabilities[C]//IEEE. Automation of Software Test (AST), 2015 IEEE/ACM 10th International Workshop on, May 23-24, 2015. Florence, Italy. NJ: IEEE, 2015: 48-52.
- [4] DIETZ W, LI Peng, REGEHR J, et al. Understanding Integer Overflow in C/C++[C]//IEEE. Software Engineering (ICSE), 2012 34th International Conference on, June 2-9, 2012. Zurich, Switzerland. NJ: IEEE, 2015: 2.
- [5] LAGUNA I, SCHULZ M. Pinpointing Scale-dependent Integer Overflow Bugs in Large-scale Parallel Applications[C]//IEEE.Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, November 13-18, 2016. Salt Lake City, Utah. NJ: IEEE, 2016: 216-227.
- [6] 董鹏程, 舒辉, 康维, 等. 基于动态二进制平台的缓冲区溢出过程分析 [J]. 计算机工程, 2012, 38(6): 66-68.
- [7] SONG Yang, ZHANG Yuqing, SUN Yingfei. Automatic Vulnerability Locating in Binary Patches[C]// IEEE.International Conference on Computational Intelligence and Security, December 11-14, 2009. Beijing, China. NJ: IEEE, 2009:474-477.
- [8] 黄克振, 连一峰, 陈恺, 等. 基于标志位差异分析的整数溢出漏洞溢出点定位方法 [J]. 计算机科学, 2014, 41(12): 19-23.
- [9] 肖海, 陈平, 茅兵, 等. 基于运行时类型分析的整形漏洞二进制检测和定位系统 [J]. 计算机科学, 2011, 38(1): 140-144.
- [10] YAMAGUCHI F, MAIER A, GASCON H, et al. Automatic Inference of Search Patterns for Taint-style Vulnerabilities[C]//IEEE.Security and Privacy (SP), 2015 IEEE Symposium on. May 17-21, 2015. San Jose, CA, USA. NJ: IEEE, 2015: 797-812.
- [11] SHAHRIAR H, HADDAD H M, VAIDYA I. Buffer Overflow Patching for C and C++ Programs: Rule-based Approach[J]. ACM SIGAPP Applied Computing Review, 2013, 13(2): 8-19.
- [12] The MITRE Corporation. CWE[EB/OL]. <http://cwe.mitre.org/data/definitions/119.html>, 2017-5-19.
- [13] 王蕾, 李丰, 李炼, 等. 污点分析技术的原理和实践应用 [J]. 软件学报, 2017, 28(4):860-882.
- [14] WANG Xuefei, MA Hengtai, JING Lisha. A Dynamic Marking Method for Implicit Information Flow in Dynamic Taint Analysis[C]//ACM.Proceedings of the 8th International Conference on Security of Information and Networks, September 8-10, 2015. Sochi, Russia. New York: ACM, 2015: 275-282.
- [15] 宋铮, 王永剑, 金波, 等. 二进制程序动态污点分析技术研究综述 [J]. 信息安全, 2016(3):77-83.