doi:10.11887/j.cn.201505022　http://journal.nudt.edu.cn

软件漏洞分析中的脆弱点定位方法*

蔡　军[1],邹　鹏[1],杨尚飞[2],何　骏[1]
(1.装备学院 复杂电子系统仿真实验室,北京　101416;2.海军装备研究院,北京　100161)

摘　要:针对二进制程序漏洞成因复杂难以分析的问题,提出运用污点分析的软件脆弱点定位方法,并实现了一个工具原型SwordChecker。以动态污点追踪为基础,依据漏洞模式通过特征匹配来定位软件中的脆弱点,运用二分查找定位影响脆弱点的敏感字节。实验表明,使用SwordChecker能够精确快速识别定位软件中三种类型的脆弱点,已成功分析了多个已公开漏洞的成因,并已辅助挖掘出几个未公开漏洞。

关键词:污点分析;脆弱点定位;二分查找
中图分类号:TP309　**文献标志码**:A　**文章编号**:1001-2486(2015)05-141-08

Vulnerable spots localization methods for software vulnerability analysis

CAI Jun[1], *ZOU Peng*[1], *YANG Shangfei*[2], *HE Jun*[1]
(1. Science and Technology on Complex Electronic System Simulation Laboratory, Academy of Equipment, Beijing 101416, China;
2. Naval Academy of Armament, Beijing 100161, China)

Abstract: Aiming at the difficulty in analysis of binary program vulnerabilities, an approach for software vulnerable spots localization based on taint analysis was proposed, and a corresponding tool named SwordChecker was implemented. This method is based on dynamic taint tracing. Software vulnerable spots were localized by character matching according to vulnerability patterns, and sensitive bytes which affected the vulnerable spots were localized by binary-search. Experiment results show that SwordChecker can accurately identify and localize three types of software vulnerable spots fast, has successfully analyzed the causes of multiple open vulnerabilities, and has assisted mining several undisclosed vulnerabilities.

Key words: taint analysis; vulnerable spots localization; binary-search

当今时代是信息时代,社会信息化是当今世界发展的潮流,信息系统已经成为社会发展的重要战略资源,信息系统的安全性也越来越重要。软件安全是信息系统安全的重要组成部分,软件漏洞则是安全问题的根源之一,层出不穷的软件安全漏洞给国民经济和社会生活带来了负面的影响。软件漏洞检测、分析及利用成为信息安全领域的一个研究热点。

软件漏洞是软件生命周期中涉及安全的设计错误、编码缺陷和运行故障[1]。将对软件进行已公开漏洞的发现和未知漏洞的发掘的技术称为软件漏洞检测技术,将分析漏洞形成原因及利用价值的技术称为漏洞分析技术,将利用漏洞实施攻击的技术称为漏洞利用技术。当前研究较多的是自动化的软件漏洞检测技术,涌现出以各种模糊器[2-6]为代表的大量的自动化的漏洞检测工具,而对漏洞分析和漏洞利用的研究还处于起步阶段,仍以手工分析为主,缺乏良好的自动化工具,蔡军等的目标就是研究自动化的软件漏洞分析技术。

脆弱点定位是软件漏洞分析中的关键环节,所谓脆弱点是指直接导致软件漏洞产生的某条指令或某个函数。脆弱点受用户输入影响,一般情况下不影响软件正常运行,但在注入特定输入的情况下会导致软件产生异常,形成漏洞。脆弱点与漏洞类型相关,漏洞种类很多,每一种漏洞形成的具体原因都不同,其脆弱点类型也不同。蔡军等主要研究输入相关漏洞的脆弱点定位。

蔡军等提出一种基于污点分析的软件脆弱点定位方法,核心思想为:(1)以细粒度的动态在线

* **收稿日期**:2014-12-31
基金项目:国家863计划资助项目(2012AA012902);"核高基"国家科技重大专项基金资助项目(2013ZX01045-004)
作者简介:蔡军(1982—),男,湖北天门人,博士研究生,E-mail:cjgfkd@163.com;
邹鹏(通信作者),男,教授,硕士,博士生导师,E-mail:zpeng@nudt.edu.cn

污点追踪作为脆弱点定位的基础;(2)基于已公开漏洞总结漏洞模式,基于漏洞模式建立规则,查找定位脆弱点;(3)基于二分查找算法定位影响脆弱点的敏感输入字节。

1 相关工作

虽然软件开发人员尤其是通用软件制造商对软件安全越来越重视,也加大了软件发布前的安全性测试力度,但是软件漏洞依然或多或少地存在,几乎不可能完全避免。

以各种模糊器为代表的软件漏洞检测工具通常只能发现导致软件崩溃(或异常)的测试用例,但是崩溃的具体原因常常不得而知,需要进一步的分析。简而言之,发现软件崩溃只能算是找到了漏洞的半成品,只有分析清楚了崩溃的原因才算是真正发现了一个漏洞。目前的软件漏洞分析以借助于调试器手工分析为主,缺少自动化的分析工具。由于软件漏洞成因复杂,手工分析十分烦琐,且需要研究人员具备极强的专业知识和丰富的经验。因此,有必要研究自动化的软件漏洞分析技术来帮助研究人员分析漏洞成因。

对自动化的软件漏洞分析的研究还不多。徐欣民[7]提出了一种基于静态分析结合动态跟踪调试的缓冲区溢出漏洞定位技术,以一种基于函数调用的缓冲区溢出漏洞模型为基础,通过动态跟踪调试捕获溢出并与静态反汇编结果比对来定位导致溢出的函数调用。杨滨诚等[8]提出了一种基于数据流分析和边界检查的缓冲区溢出定位技术,通过数据流和别名分析识别可能受外部输入影响的指针和数组,通过边界检查定位溢出。史胜利[9]提出了一种基于代码插桩的缓冲区溢出漏洞定位技术,通过代码插桩捕获信息,根据系统提供的异常信息,分析产生异常的指令,获取破坏的内存点,进而查找漏洞所在点。杨羡环[10]提出了一种基于函数调用序列的漏洞定位技术,通过多次程序执行和面向方面编程框架获取函数调用序列,通过分析函数调用序列定位漏洞。以上几种方法均只有一个笼统的思路,难以实现精确定位,只能分析缓冲区溢出漏洞,且未能分析真实应用软件的漏洞,自动化程度不高。

2 污点分析概述

污点分析[11-15]是一种新兴的程序分析技术,其主要思想是将用户输入作为污点源,在程序执行过程中追踪污点源的传播,所有直接或间接受污点源影响的变量都是被污染的变量,称为污点数据。用污点分析来分析软件漏洞的原理是:任何用户可控的变量都是不安全的,当一个被污染的变量被用来执行危险操作时就可能产生安全漏洞。

污点分析从是否需要运行软件的角度可以分为两种类型:静态污点分析和动态污点分析。静态污点分析工作在源代码级,通过分析程序源代码来推衍污点的传播。动态污点分析则是在程序实际运行过程中追踪污点的传播。静态污点分析的最大缺陷是依赖于源代码,然而大多数时候源代码是不可得到的,且静态的污点分析由于缺乏运行时信息,误报也很多。因此,动态的污点分析更加实用,但是其实现比较复杂,SwordChecker 使用的是动态污点分析。

动态污点分析的实现主要有两种方式,即通过硬件实现和通过软件实现。硬件实现方式虽然效率较高,但是需要对硬件进行扩展,实现难度很大且成本高,因而并不常用。软件实现方式一般通过动态二进制插桩来实现。动态二进制插桩是在程序执行过程中,在程序实际执行的某个基本执行单元(指令、基本块、函数等)前后插入分析代码来获取想要的运行时信息,通常借助于二进制插桩平台来实现。现有的二进制插桩平台主要有三个:DynamoRIO[16],Valgrind[17]和 Pin[18]。SwordChecker 使用 Pin 来实现。

3 运用污点分析的脆弱点定位

运用污点分析的脆弱点定位方法分为三个步骤:第一步是追踪用户输入即污点源在程序执行过程中的传播,第二步是定位程序中的脆弱点,第三步是定位输入中影响脆弱点的敏感字节。

3.1 Pin 的二进制程序在线污点追踪

Pin 是 Intel 公司研发的一个动态二进制插桩平台,它允许在可执行程序的任意位置插入任意代码,并且代码在程序运行时动态添加。可以把 Pin 比作一个超级即时编译器,这个超级编译器的输入不是字节码而是一个可执行程序。Pin 拦截可执行程序要执行的第一条指令,并将以这条指令开始的一个直线代码序列编译成新的代码在 Pin 虚拟机里执行,新的代码和原始代码基本一致。遇到跳转指令时,Pin 重新接管程序,再编译下一段要执行的代码。可执行程序要执行的所有代码都被编译成了新的代码在 Pin 虚拟机里执行,原始代码仅供参考,这样 Pin 就向用户提供了向目标程序注入它们自己代码的机会。

Pin 的结构如图 1 所示[19],可以看到 Pin 由虚拟机、代码缓存器和插桩 API 三个部分构成。

插桩 API 是 Pin 提供给用户的接口,用户可以利用它们来编写自己的插桩工具,称为“Pintool”;代码缓存器用来缓存由原始代码编译而来的新代码,这样可以避免重复编译,提高效率;虚拟机用来执行新代码。虚拟机又由即时编译器、仿真器和分派器构成,三者在 Pin 控制应用程序后通过互相协同来执行程序,即时编译器负责编译原始代码和插桩用户代码,分派器负责调度代码的执行,仿真器负责解释那些不能由虚拟机直接执行的指令,如系统调用。

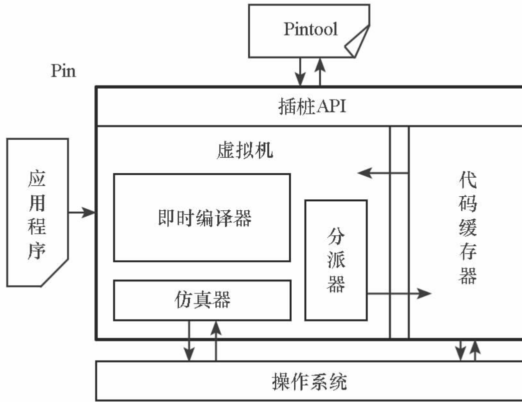

图 1　Pin 结构图
Fig. 1　Architecture of Pin

基于 Pin 的在线污点追踪是借助 Pin 在程序运行过程插入分析代码来实现污点追踪逻辑。污点追踪主要包含污点引入和污点传播两个过程。

3.1.1　污点引入

由于程序大部分时候都在执行与输入无关的指令,如加载动态链接库、共享库等,需要决定在程序执行的什么地方来引入污点源,这就是污点引入。污点引入的时刻实际上是程序刚开始操作污点源的时刻。污点源的种类主要有三种,即文件、网络数据包和命令行参数(常见于 Linux 系统),如使用 Microsoft Word 打开一个. doc 文件,这个. doc 文件就可以作为污点源。

使用 Pin 来实现污点引入的方法是通过 Pin 提供的插桩 API 来劫持程序执行过程中的打开和读取污点源的系统调用,插入污点引入代码。如使用文件作为污点源,则需劫持 open, read, readv 等读文件相关系统调用,如使用网络数据包作为污点源,则需劫持 socket, accept, recv 等 socket 相关系统调用。SwordChecker 当前主要支持文件作为污点源。

在使用文件作为污点源时,可以将整个文件设置为污点源,也可以将文件中的部分字节设置为污点源。SwordChecker 允许用户通过一系列参数来定制污点源。对于文件污点源,使用三个参数:“-tf”, “-to” 和 “-ts”。“-tf”参数指示了需要被设置为污点的文件,“-to”参数指示了污点文件中第一个需要被设置为污点源的字节的偏移量,“-ts”参数指示了以偏移量为开始需要被设置为污点的字节数目。

3.1.2　污点传播

污点传播是在污点引入后,根据一定的传播策略追踪污点源被程序指令进行的复制和更改。污点数据被指派一个污点标签,污点标签根据指令的语义从原操作数传播到目的操作数。污点传播策略直接影响整个污点传播过程的正确性。

在污点传播时有两个基本问题。一个问题是污点传播粒度,指最小的污点传播单元。选择一个合适的污点传播粒度很重要。如果粒度选择太大,如大到一片连续的内存块,容易造成污点数据丢失或是不精确的污点传播。相反,粒度选择太小,如小到一个位(bit),则容易导致污点传播过度扩散,且污点传播的实现逻辑也会变得相当复杂。SwordChecker 选取字节作为污点传播粒度,因为大部分指令都以字节为基本单位进行运算,且在大部分计算机体系结构中,最小的内存寻址单元也是字节。相对于传统的以四字节内存单元为最小污点传播单元的污点追踪,SwordChecker 实施的是更细粒度的污点追踪。

另一个问题是污点标签大小。每一个污点传播单元都将被指派一个污点传播标签。最小的污点标签大小可以是一个位。如,位为 1 代表这个污点单元被污染了,位为 0 代表未被污染。大的标签虽然功能更多,因为它允许不同类型的数据被唯一标记,但是大的标签需要更加复杂的传播逻辑和更大的存储空间。这一方面会显著增加内存消耗,另一方面会大幅降低污点追踪速度。SwordChecker 的目标是快速的在线污点追踪,在不影响污点传播精度的前提下越快越好,因此选择一个位大小的标签。

在二进制级别上,有两类对象可被污染:内存位置和寄存器。因此,SwordChecker 在两类数据结构中存储污点标签。一个数据结构是“mem_taint_map”,它为每个可寻址的内存单元维持一个位作为标签。另一个数据结构是“reg_taint_map”,它为每一个 32 位寄存器维持一个字节来存储标签,该字节的低四位存储了寄存器的四位污点标签。

关于污点传播策略,SwordChecker 主要考虑三类指令的污点传播:(1)数据复制指令(如

MOV，PUSH，POP 等）。针对这类指令，复制源操作数的标签到目的操作数。（2）数据运算指令（如 ADD，SUB，AND，SHL 等）。针对这类指令，将源操作数和目的操作数的标签一起运算来更新目的操作数的标签。（3）特殊指令。一些特殊指令可能会导致操作数的污点标签被清除，如 CPUID 指令，它是 intel IA32 架构下获得 CPU 信息的汇编指令，可以得到 CPU 类型、型号、制造商信息、商标信息、序列号、缓存等一系列与 CPU 相关的东西。它使用 eax 作为输入参数，eax，ebx，ecx，edx 作为输出参数，执行该指令后，这几个寄存器都将被立即数填充，而立即数不是污点数据。

3.2 漏洞模式的脆弱点定位

所谓漏洞模式是指不同类型漏洞的特征。漏洞类型不同，其脆弱点类型也不同。SwordChecker 适用的漏洞类型主要为输入相关漏洞，所谓输入相关漏洞是指那些能够由特定的输入触发的漏洞，这个输入是由用户提供的，可以是文件、网络数据包，也可以是键盘输入。输入相关漏洞主要包括除 0 漏洞、整数溢出漏洞、缓冲区溢出漏洞等。SwordChecker 根据漏洞模式建立规则，通过污点检查来实现脆弱点的定位，即检查是否有安全敏感操作操作了污点数据。

SwordChecker 当前支持以下几种类型的脆弱点定位。

3.2.1 除法指令

除法指令是除 0 漏洞的脆弱点。除 0 漏洞是指在做除法运算时除数为 0 导致软件出现除 0 异常，这种漏洞经常发生在一个非预期的值被提供给程序，然后用这个值来计算物理维度，如大小、长度、宽度和高度时。图 2 是含有除 0 漏洞的一段简单代码，变量 a 和 b 的值从一个输入文件中读取，如果 $a>b$ 且 $a-b=100$，就会在代码段第 2 行产生除 0 异常，因此这段代码的第 2 行就是一个脆弱点（注意不是所有的除法操作都是脆弱点，只有受输入影响的除法操作才是）。

```
1. if (a > b) {
2.        return (a + b)/(a - b - 100);
3. }
4. else
5.    return 0;
```

图 2　含有除 0 漏洞的代码片段

Fig. 2　Code fragment including a division by zero

要定位这一类脆弱点，只需借助于 Pin 的 API “INS_AddInstrumentFunction”来实施指令级插桩，插入分析代码，如果是 div 和 idiv 指令就检查它们的除数是否被污染。如果除数存放在内存中，则从 mem_taint_map 查找对应的污点标签看是否为 1，如果除数存放在寄存器中，则检查 reg_taint_map。

3.2.2 内存操作函数

内存操作函数（如 malloc，calloc，memcpy 等）是整数溢出漏洞的脆弱点。整数溢出是指算术运算的结果超出了整型变量能够表示的值的范围（或大于最大值，或小于最小值）。并不是所有的整数溢出都会产生安全问题，只有当溢出值被用来作为安全敏感函数的参数时，才会产生漏洞。例如，将一个溢出值用来作为内存分配函数 malloc 代表分配内存大小的参数时，就会导致所分配的内存小于实际所需的内存，最终可能导致缓冲区溢出。

图 3 是含有整数溢出漏洞的一段代码，整数变量 a 和 b 的值也是由用户输入决定，而整数变量 e 的值则由 a 和 b 决定。显然，如果没有溢出检查，变量 e 很可能会溢出。更进一步，如果一个溢出的 e 被用作内存分配函数“malloc”的参数（第 3 行），就很有可能导致一个漏洞，因此这段代码的第 3 行就是一个脆弱点。

```
1. if (a > 0 && b > 0) {
2.        e = a * b * 4;
3.        p = malloc (e);
4.        if (p == NULL) return 0;
5. ……
```

图 3　含有整数溢出漏洞的代码片段

Fig. 3　Code fragment including an integer overflow

要定位这一类脆弱点，只需借助于 Pin 的 API “RTN_AddInstrumentFunction”来实施函数级插桩，插入分析代码，检查函数的有关参数是否为污点数据。以 calloc 函数为例，其原型为“void * calloc(size_t n, size_t $size$)”，其功能是在内存的动态存储区中分配 n 个长度为 $size$ 的连续空间，函数返回一个指向分配起始地址的指针；如果分配不成功，返回 NULL。由于 n 过大和 $size$ 过大都可导致溢出，所以在分析代码中需要检查 n 和 $size$ 是否被污染。

3.2.3 字符串操作函数

字符串操作函数（如 strcpy，strcat，sprintf，vsprintf，gets 等）是缓冲区溢出漏洞的脆弱点。缓冲区溢出可能是最众所周知的一种漏洞。当一个

程序试图将比给定内存块更大的数据放入这块内存时就会发生缓冲区溢出,溢出会导致在分配内存之外写数据,进而导致程序崩溃或执行恶意代码。虽然大部分软件开发人员都知道缓冲区溢出,但这种漏洞并不好检测。

造成缓冲区溢出的主要原因是不完整或不恰当的对用户输入的检查。图 4 是含有缓冲区溢出漏洞的一段代码,字符串 str 的值是输入文件的一行文本,在示例代码的第 5 行,函数 strcpy 直接复制"str"的内容到"buffer",只要"str"的长度大于 50,就会导致"buffer"溢出,因此这段代码的第 5 行就是一个脆弱点。

```
1.  ifstream in("input. txt");
2.  char buffer[50];
3.  string str;
4.  getline(in, str);
5.  strcpy(buffer, str);
```

图 4　含有缓冲区溢出漏洞的代码片段

Fig. 4　Code fragment including a buffer overflow

要定位这一类脆弱点,也是借助于 Pin 的 API "RTN_AddInstrumentFunction"来实施函数级插桩,插入分析代码,检查函数的有关参数是否为污点数据。以 strcpy 函数为例,其原型为"extern char * strcpy(char * dest, const char * src)",其功能是把从 src 地址开始且含有 NULL 结束符的字符串复制到以 dest 开始的地址空间。在插入的分析代码中需要检查 dest 和 size 的地址以及它们指向的内存地址是否被污染。

3.3　二分查找的敏感字节定位

上一步的目的是检查程序是否存在脆弱点,实际上是将整个输入文件都作为污点源来进行污点追踪,然后检查是否有安全敏感操作来操作污点数据。如果上一步检测出有脆弱点存在,则接下来进一步定位输入文件中影响脆弱点的敏感字节。例如,图 2 中变量 a 和 b 的值影响第 2 行代码中除法操作的除数值,则输入文件中影响 a 和 b 的值的字节就是要定位的敏感字节。

SwordChecker 使用二分查找算法来定位敏感字节,算法通过 shell 脚本编程实现。算法流程如图 5 所示,具体步骤如下:

Step1:获取输入文件大小 *Size*,初始化文件偏移量 $Low = 0$, $High = Size$, *Low* 代表下限, *High* 代表上限,最终定位的敏感字节位置在 *Low* 与 *High* 之间(一般为 4 个字节)。

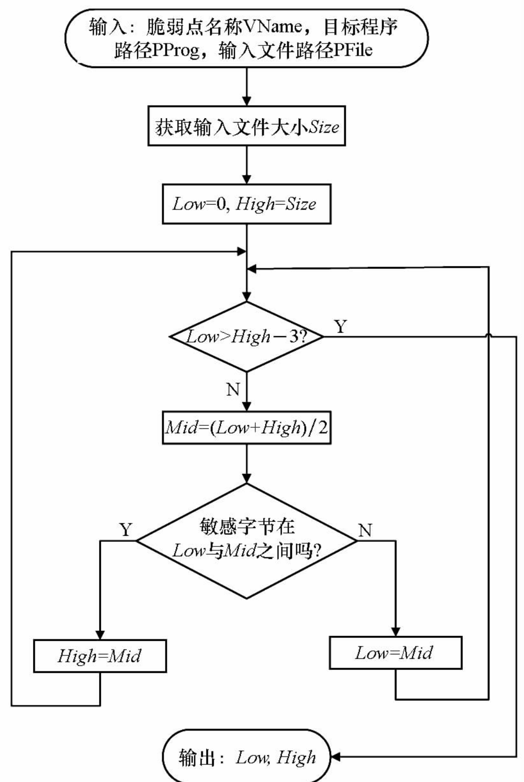

图 5　敏感字节的二分查找定位算法

Fig. 5　Binary search algorithm for sensitive bytes localization

Step2:检查是否满足终止条件 $Low > High - 3$,如果满足则输出 *Low*, *High*。

Step3:计算 *Low*, *High* 之间的中间位置 $Mid = (Low + High)/2$,检查敏感字节是在 $[Low, Mid]$ 之间还是在 $[Mid, High]$ 之间(分别将输入文件的这两个区间设为污点源,看是否有指定的脆弱点 VName 出现),如果敏感字节在 $[Low, Mid]$ 之间,则 $High = Mid$,如果敏感字节在 $[Mid, High]$ 之间,则 $Low = Mid$。

Step4:重复 Step2, Step3。

4　实验分析

实验分为两类,一是用 SwordChecker 来分析定位已公开漏洞的脆弱点,主要从国际上知名的公开漏洞库中选取一些真实漏洞作为测试对象;二是用 SwordChecker 来辅助挖掘未公开漏洞(即 0day 漏洞),主要选取一些运行在 Linux 上的常用真实应用软件作为测试对象。所有实验均在 Ubuntu 12.04 32 位系统上进行。

4.1　已公开漏洞脆弱点定位

首先,从国际上著名的公开漏洞库 Common Vulnerabilities and Exposures (CVE)[20], Open Source Vulnerability Database(OSVDB)[21], Exploit

Database(EDB)[22]里选取了一些真实的软件漏洞来对 SwordChecker 进行测试,实验结果见表 1。选取的每个漏洞都在 EDB 里有一个 POC (Proof of Concept)文件可供下载,即选取的每个漏洞都可以由对应的 POC 文件触发。表 1 的第一列是漏洞在漏洞库中的编号,如 CVE-2014-1684 指 CVE 中编号为 2014-1684 的漏洞,OSVDB-ID-88610 指 OSVDB 中编号为 88610 的漏洞,EDB-ID-7812 指 EBD 中编号为 7812 的漏洞。

表 1 公开漏洞脆弱点定位

Tab. 1 Vulnerable spots localization of several open vulnerabilities

漏洞编号	软件名称版本	漏洞类型	脆弱点	POC 文件类型大小	敏感字节
CVE-2014-1684	VLC 2.12	除 0	div	ASF 9 900 966B	123 ~ 126
OSVDB-ID-88610	Totem 3.4.3	除 0	div, idiv	AVI 1 149 900B	17 ~ 20, 4423 ~ 4424
CVE-2012-5470	VLC 2.0.3	缓冲区溢出	memcpy, malloc	PNG 11 077B	1 ~ 4, 84 ~ 87
CVE-2007-4938	MPlayer 1.0	整数溢出	calloc	AVI 46 956 236B	169 ~ 172, 225 ~ 228
EDB-ID-7812	MPlayer 1.0rc2	缓冲区溢出	memcpy	VQF 120 132B	13 ~ 16

以 OSVDB-ID-88610 漏洞为例进行分析。该漏洞是多媒体播放器 Totem Movie Player(一套在类 Unix 操作系统上运行的多媒体播放器,也是 Ubuntu 系统的预设影片播放器)上的一个除 0 漏洞,软件版本号为 3.4.3,漏洞现象如图 6 所示,用该软件(程序名为 totem)打开 POC 文件 poc1. avi时发生浮点数例外崩溃退出。

```
cj@phoenix: ~/SwordChecker/tools
cj@phoenix:~/SwordChecker/tools$ /home/cj/totem/bin/totem /home/cj/poc1.avi

(totem:4102): Clutter-WARNING **: Whoever translated default:LTR did so wrongly.
浮点数例外 (核心已转储)
cj@phoenix:~/SwordChecker/tools$
```

图 6 OSVDB-ID-88610 漏洞现象

Fig. 6 Phenomenon of OSVDB-ID-88610

用 SwordChecker 对该漏洞进行分析的结果如图 7 所示。该软件有两个脆弱点,一个是“div dword ptr [esp + 0x1c]”指令(地址为 0x952c8372),一个是“idiv ecx”指令(地址为 0x8c594e12),这两条指令的操作数都是污点数据,直接导致漏洞产生的是 idiv 指令。SwordChecker 进一步分析出影响这两个脆弱点的敏感字节分别为 POC 文件的第 17 ~ 20 字节和第 4423 ~ 4424 字节。

从表 1 可以看出,CVE-2014-1684 的脆弱点为 div 指令,敏感字节为第 123 ~ 126 字节;CVE-2012-5470的脆弱点为 memcpy 和 malloc 函数,敏感字节分别为第 1 ~ 4 和第 84 ~ 87 字节;CVE-2007-4938 的脆弱点为 calloc,敏感字节为 169 ~ 172,225 ~ 228;EDB-ID-7812 的脆弱点为 memcp,敏感字节为第 13 ~ 16 字节。另外,在实验中还发现一个规律,敏感字节只占输入文件

```
fdset insert: /home/cj/poc1.avi, 0x14
Taint introduce: 0x8a37088, 0xc
Taint introduce: 0x8a37088, 0x8
Taint introduce: 0x8f195d8, 0x2268

The divisor is tainted
0x952c8372: div dword ptr [esp+0x1c]

Integer Division By zero
0x8c594e12: idiv ecx

The divisor is tainted
0x8c594e12: idiv ecx
cj@phoenix:~/SwordChecker/tools$
```

图 7 OSVDB-ID-88610 漏洞的分析结果

Fig. 7 Analysis result of OSVDB-ID-88610

的很小一部分(如 CVE-2007-4938 的 POC 文件大小为 46 956 236B,但敏感字节只有 8 个),且一般位于文件的头部,大体在前 1000 个字节以内。可在文件的前 1000 个字节里运用二分查找算法定位。

4.2 脆弱点定位辅助 0day 漏洞挖掘

通过脆弱点定位不但能帮助分析已公开漏洞的成因,还能辅助挖掘 0day 漏洞,已经用 SwordChecker 结合开源模糊测试工具 zzuf[6] 挖出了几个 0day 漏洞,以 swftools-0.9.2 png2swf 整数溢出漏洞为例进行分析。种子输入为一个正常的 PNG 文件 input. png,其十六进制格式如图 8 所示。

```
phoenix@Lenovo-E4430:~/workspace/SwordChecker/tools$ hd input.png
00000000  89 50 4e 47 0d 0a 1a 0a  00 00 00 0d 49 48 44 52  |.PNG........IHDR|
00000010  00 00 00 01 00 00 00 01  08 02 00 00 00 90 77 53  |..............wS|
00000020  de 00 00 00 1b 74 45 58  74 54 69 74 6c 65 00 54  |.....tEXtTitle.T|
00000030  68 69 73 20 69 73 20 6d  79 20 74 65 73 74 20 69  |his is my test i|
00000040  6d 61 67 65 72 d5 4e 17  00 00 00 0c 49 44 41 54  |mager.N.....IDAT|
00000050  08 99 63 60 60 60 00 00  00 04 00 01 a3 0a 15 e3  |..c```..........|
00000060  00 00 00 00 49 45 4e 44  ae 42 60 82              |....IEND.B`.|
0000006c
phoenix@Lenovo-E4430:~/workspace/SwordChecker/tools$
```

图 8 Input. png 的十六进制格式

Fig. 8 Hexadecimal format of input. png

```
phoenix@Lenovo-E4430:~/workspace/SwordChecker/tools$ zzuf -s 0:1000 -r 1 -b 11 /
home/phoenix/swftools-0.9.2/src/png2swf input.png
input.png is not a PNG file!
Error opening input file: input.png
No png files found in arguments
input.png is not a PNG file!
Error opening input file: input.png
No png files found in arguments
input.png is not a PNG file!
Error opening input file: input.png
No png files found in arguments
input.png is not a PNG file!
Error opening input file: input.png
No png files found in arguments
zzuf[s=4,r=1]: signal 9 (memory exceeded?)
phoenix@Lenovo-E4430:~/workspace/SwordChecker/tools$
```

图 9　用 zzuf 对 png2swf 进行模糊测试

Fig. 9　Using zzuf fuzzing png2swf

```
phoenix@Lenovo-E4430:~/workspace/SwordChecker/tools$ hd new.png
00000000  89 50 4e 47 0d 0a 1a 0a  00 00 00 2d 49 48 44 52  |.PNG.......-IHDR|
00000010  00 00 00 01 00 00 00 01  08 02 00 00 00 90 77 53  |..............wS|
00000020  de 00 00 00 1b 74 45 58  74 54 69 74 6c 65 00 54  |.....tEXtTitle.T|
00000030  68 69 73 20 69 73 20 6d  79 20 74 65 73 74 20 69  |his is my test i|
00000040  6d 61 67 65 72 d5 4e 17  00 00 00 0c 49 44 41 54  |mager.N.....IDAT|
00000050  08 99 63 60 60 60 00 00  00 04 00 01 a3 0a 15 e3  |..c```..........|
00000060  00 00 00 00 49 45 4e 44  ae 42 60 82              |....IEND.B`.|
0000006c
phoenix@Lenovo-E4430:~/workspace/SwordChecker/tools$
```

图 10　New. png 的十六进制格式

Fig. 10　Hexadecimal format of new. png

```
phoenix@Lenovo-E4430:~/workspace/SwordChecker/tools$ /home/phoenix/swftools-0.9.
2/src/png2swf new.png
段错误 (核心已转储)
phoenix@Lenovo-E4430:~/workspace/SwordChecker/tools$
```

图 11　Swftools-0. 9. 2 png2swf 漏洞现象

Fig. 11　Phenomenon of swftools-0. 9. 2 png2swf vulnerability

首先用 SwordChecker 定位到 input. png 的第十二个字节“0x0d”影响内存分配函数 malloc,接下来用 zzuf 通过集中对第十二个字节进行变异来实施模糊测试就发现了一个漏洞。如图 9 所示,zzuf 有三个参数:s 是随机数种子,s 为 0 : 1000 是指种子从 0 到 1000 变化;r 是变异率,r 为 1 表示百分之百的变异;b 是变异范围,b 为 11 表示第十二个字节(从 0 开始起算)。从图 9 可以看到当 $s=4$,$r=1$ 时检测到了“memory exceeded”,用 $s=4$,$r=1$ 作为参数生成一个新的文件 new. png(zzuf 使用相同的参数生成的文件永远相同)。如图 10 所示,可以看到第十二个字节变成了“0x2d”。将 new. png 作为 png2swf 的输入就触发了漏洞,漏洞现象如图 11 所示,可以看到发生了段错误。进一步用 SwordChecker 分析原因,得到的分析结果如图 12 所示,可以确定该漏洞是由 malloc 参数过大($n=0x61676572$,与 new. png 的第 66 ~ 69 个字节对应)引起的整数溢出漏洞。

```
fdset insert: new.png, 0x6
Taint introduce: 0x958f2000, 0x6c
Malloc: n= 0x2d, 0xbfee1ed0
Taint introduce: 0x958f2000, 0x2b
Malloc: n= 0x61676572, 0xbfee1ed0
phoenix@Lenovo-E4430:~/workspace/SwordCheck/tools$
```

图 12　Swftools-0. 9. 2 png2swf 漏洞分析结果

Fig. 12　Analysis result of swftools-0. 9. 2 png2swf vulnerability

5　结论

本文提出了一种基于污点分析的软件脆弱点定位方法,该方法以动态污点追踪为基础,基于漏洞模式建立检查规则,通过污点检查来定位脆弱点,通过二分查找来进一步定位输入文件中影响脆弱点的敏感字节。实现了一个相应的原型系统 SwordChecker。实验表明,SwordChecker 不但能快速精确定位除 0 漏洞、整数溢出漏洞和缓冲区溢出漏洞的脆弱点,还能精确定位输入文件中影响脆弱点的敏感字节;不但能帮助分析漏洞的成因,还能辅助 0day 漏洞挖掘,已经分析了多个公开漏洞的成因,并辅助挖掘了几个 0day 漏洞。

SwordChecker 当前适用于 Linux 下的 32 位二进制程序,目前主要支持文件处理程序,下一步计划将其扩展到支持 Windows 系统和网络处理程序。

参考文献(References)

[1]　吴世忠, 郭涛, 董国伟, 等. 软件漏洞分析技术进展[J]. 清华大学学报(自然科学版), 2012, 52(10): 1309 – 1319.
WU Shizhong, GUO Tao, DONG Guowei, et al. Software vulnerability analyses: a road map[J]. Journal of Tsinghua University (Science and Technology), 2012, 52(10): 1309 – 1319. (in Chinese)

[2]　Caca labs. Zzuf-Multi-purpose fuzzer[EB/OL]. http://caca.zoy.org/wiki/zzuf.

[3]　A pure-python fully automated and unattended fuzzing framework[EB/OL]. https://github.com/OpenRCE/sulley.

[4]　Eddington M. Peach fuzzer[EB/OL]. http://peachfuzzer.com/.

[5]　Sogeti ESEC Lab. Fuzzgrind[EB/OL]. http://eseclab.sogeti.com/pages/Fuzzgrind.

[6]　FuzzBALL: Vine-based binary symbolic executi-on[EB/OL]. https://github.com/bitblaze-fuzzball/fuzzball.

[7]　徐欣民. 一种缓冲区溢出漏洞自动挖掘及漏洞定位技术[D]. 武汉:华中科技大学, 2008.
XU Xinmin. An automatic mining and positioning technology for buffer overflow vulnerabilities[D]. Wuhan: Huazhong University of Science and Technology, 2008. (in Chinese)

[8]　杨滨诚, 茅兵. 输入相关的缓冲区溢出检测和定位[C]//2010(第三届)全国网络与信息安全学术会议, 2010: 103 – 108.
YANG Bincheng, MAO Bing. Detection and location of input-related buffer overflows[C]//Proceedings of 2010 (3rd) National Conference on Computer Networks and Information Security, 2010: 103 – 108. (in Chinese)

[9]　史胜利. 基于代码插装的缓冲区溢出漏洞定位技术[J]. 计算机工程,2012, 38(9): 138 – 140.
SHI Shengli. Buffer overflow vulnerability location technology based on code instrumentation[J]. Computer Engineering, 2012, 38(9): 138 – 140. (in Chinese)

[10]　杨羡环. 基于函数调用序列的漏洞定位方法研究[D]. 武汉:华中科技大学, 2013.

YANG Xianhuan. Research of software-defect localization based on function calling sequence mining[D]. Wuhan: Huazhong University of Science and Technology, 2013. (in Chinese)

[11] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[C]//Proceedings of the Network and Distributed System Security Symposium,2005.

[12] Clause J, Li W C, Orso A. Dytan: a generic dynamic taint analysis framework[C]//Proceedings of the 2007 international symposium on Software testing and analysis. ACM, 2007: 196 – 206.

[13] Kang M G, McCamant S, Poosankam P,et al. DTA++: dynamic taint analysis with targeted control-flow propagation [C]// Proceedings of the 18th Network and Distributed System Security Symposium, 2011.

[14] Bosman E, Slowinska A, Bos H. Minemu: the world's fastest taint tracker[C]//Proceedings of Recent Advances in Intrusion Detection. Springer Berlin Heidelberg, 2011,6961: 1 – 20.

[15] Kemerlis V P, Portokalidis G, Jee K, et al. libdft: practical dynamic data flow tracking for commodity systems[C]// Proceedings of ACM SIGPLAN Notices, 2012, 47(7): 121 – 132.

[16] DynamoRIO[EB/OL]. http://www.dynamorio.org/.

[17] Valgrind[EB/OL]. http://valgrind.org/.

[18] Intel. Pin-a dynamic binary instrumentation tool[EB/OL]. https://software. intel. com/en-us/articles/pin-a-dynamic-binaryinstrumentation-tool.

[19] Hazelwood K, Klauser A. A dynamic binary instrumentation engine for the ARM architecture[C]//Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems, 2006: 261 – 270.

[20] Common vulnerabilities and exposures[EB/OL]. https://cve.mitre.org/.

[21] Wikipedia. Open source vulnerability database[EB/OL]. http://en.wikipedia.org/wiki/Open_Source_Vulnerability_Database.

[22] Offensive Security[EB/OL]. The Exploit Database. http://www.exploit-db.com/.

(上接第 39 页)

[7] Cheng T P. Accelerating universal Kriging interpolation algorithm using CUDA-enabled GPU [J]. Computers & Geosciences, 2013, 54: 178 – 183. [8] Shi X, Ye F. Kriging interpolation over heterogeneous computer architectures and systems[J]. GIScience & Remote Sensing, 2013,50(2):196 – 211.

[9] 卢风顺,宋君强,银福康,等. CPU/GPU 协同并行计算研究综述[J]. 计算机科学, 2011, 38(3): 5 – 9.
LU Fengshun, SONG Junqiang, YIN Fukang, et al. Survey of CPU/GPU synergetic parallel computing [J]. Computer Science, 2011,38(3): 5 – 9. (in Chinese)

[10] 方留杨,王密,李德仁,等. 负载分配的 CPU/GPU 高分辨率卫星影像调制传递补偿方法[J]. 测绘学报, 2014, 43(6): 598 – 606.
FANG Liuyang, WANG Mi, LI Deren, et al. A workload-distribution based CPU/GPU MTF compensation approach for high resolution satellite images [J]. Acta Geodaetica et Cartographica Sinica, 2014, 43(6): 598 – 606. (in Chinese)

[11] Wang S, Armstrong M P. A theoretical approach to the use of cyberinfrastructure in geographical analysis [J]. International Journal of Geographical Information Science, 2009, 23(2): 169 – 193.

[12] Dong B, Li X, Wu Q M, et al. A dynamic and adaptive load balancing strategy for parallel file system with large-scale I/O servers [J]. Journal of Parallel and Distributed Computing, 2012, 72(10):1254 – 1268.

[13] 夏飞,朱强华,金国庆. 基于 CPU-GPU 混合计算平台的 RNA 二级结构预测算法并行化研究[J]. 国防科技大学学报, 2013, 35(6): 138 – 146.
XIA Fei, ZHU Qianghua, JIN Guoqing. Accelerating RNA secondary structure prediction applications based on CPU-GPU hybrid platforms [J]. Journal of National University of Defense Technology, 2013,35(6):138 – 146. (in Chinese)

[14] 赵斯思,周成虎. GPU 加速的多边形叠加分析[J]. 地理科学进展, 2013, 32(1): 114 – 120.
ZHAO Sisi, ZHOU Chenghu. Accelerating polygon overlay analysis by GPU[J]. Progress in Geography, 2013, 32(1): 114 – 120. (in Chinese)

[15] 马安国,成玉,唐遇星,等. GPU 异构系统中的存储层次和负载均衡策略研究[J]. 国防科技大学学报, 2009, 31(5): 38 – 43.
MA Anguo, CHENG Yu, TANG Yuxing, et al. Research on memory hierarchy and load balance strategy in heterogeneous system based on GPU [J]. Journal of National University of Defense Technology, 2009, 31(5): 38 – 43. (in Chinese)