通过表 4 中的实验数据可知:相比传统的污点分析工具来看,在验证漏洞方面,本文提出的污点分析方法具备较强的漏洞检测能力,这是由于:

1) 本文提出的方法加入的黑名单机制使其在检测漏洞方面更加具有灵活性,使其可检测的漏洞模式包括内存任意读写、内存分配不当、调用危险函数,较 TEMU 更多;例如对 CVE-2010-0304 漏洞,需检测的危险函数为 *tvb_get_nstringz*,该函数第 3 个参数为读取的长度,由网络数据包中指定,为污点数据;第 4 个参数为栈变量.当第 3 个参数大于第 4 个参数时,造成栈溢出;
2) 本文提出的方法解决了 JIT(just in time)翻译执行导致的污点丢失问题,因此可处理使用了 JIT 机制的程序.例如 CVE-2012-0774 漏洞,若不解决 JIT 翻译问题,则会导致污点信息在传播过程中丢失,进而无法检测该类漏洞.

Table 4 Experiment of vulnerabilities detecting

表 **4** 漏洞检测实验

漏洞编号	TEMU 使用的污点分析方法		本文提出的污点分析方法		检测规则
	是否可验证	检测时间(s)	是否可验证	检测时间(s)	
CVE-2009-2347	Y	163	Y	36	*malloc*
CVE-2009-3459	Y	669	Y	127	*malloc*
CVE-2010-0188	Y	146	Y	27	*memcpy*
CVE-2010-0304	–	–	Y	243	*tvb_get_nstringz*
CVE-2010-3333	–	–	Y	155	内存任意读写
CVE-2011-0104	–	–	Y	149	*memcpy*
CVE-2011-0978	–	–	Y	127	内存任意读写
CVE-2012-0003	–	–	Y	350	内存任意读写
CVE-2012-0158	–	–	Y	171	*memcpy*
CVE-2012-4564	Y	210	Y	43	*malloc*
CVE-2012-0774	–	–	Y	156	内存任意读写
CVE-2012-1535	–	–	Y	78	*malloc*

在执行效率方面,本文提出的基于执行踪迹离线索引的污点分析方法更具有明显的优势,平均比传统的污点分析方法快 5 倍以上.

我们将本文所提的方法部署在漏洞挖掘系统平台中,已发现大量软件漏洞,其中发现了金山 WPS、Xnview 等软件中 10 个 0day 漏洞,已提交给国家信息安全漏洞库.出于安全性与厂家协作等考虑,故不在此批露这些漏洞的细节.

4.1 讨 论

二进制代码的污点分析方法是一种近似准确的数据流分析方法,可通过污点分析方法观察到敏感数据流向何处,如果其传入敏感位置时,则可能导致漏洞.然而,由于污点分析方法只关注数据流向而不关心数据的值,导致在有些情形下可能产生污点分析的不确定性,即污点分析方法的盲区.在实际测试中,发现污点分析算法中,盲区主要包含以下 3 种情况.

① 污点数据扩散

考虑以下这种情况:

$$\begin{cases} x = y + z \\ z = x - y \end{cases}.$$

假设 y 为污点数据,那经过第 1 条语句执行后,根据污点传播算法,x 也成为污点数据.同理,当执行完第 2 条语句后,z 也成为污点数据.然而从语义上看,z 的值并未发生变化,永远保持其原有的值,因此,将其认为是污点数据是错误的.类似这种情况,将会引发污点数据的扩散.

② 污点数据丢失

污点信息同样可能会由于语句中的副作用发生丢失,如 *strlen* 函数,当该函数的参数为污点数据时,理论上该函数的返回值的大小也应被认为是污点数据.但是按照传统的污点传播算法,这类污点信息必然会丢失.本文

提出函数黑名单的目的就是想解决这类型的问题,但如果用户内联实现 *strlen* 函数,则无法通过此方法解决.类似地,循环结构中,循环上界与循环次数之间也可被认为是相关的,尽管不少研究者[13,14]提出了对循环结构的识别,尽可能地弥补由循环引起的污点信息丢失,但仍无法适用于全部情况.

③ 污点传播不当

污点传播不当,同样是由于污点分析方法只关注数据流而不关心其值所引起的问题,考虑如下的情况:

$$\begin{cases} x = y \times z \\ y = x \end{cases}.$$

假设 y 为 4 字节整数,其低两个字节为污点数据,当这两条语句执行完成后,很难确定 y 到底含有几个字节的污点数据,因为其结果与 z 的值相关.如果 z 的值为 1,则 y 的污点信息未发生变化;而如果 z 为 0,则 y 应该不再具有污点属性.*fscanf* 函数中也存在这样的问题,以下是其内部的几行代码:

$$\begin{cases} x = SHL(x,4) \\ \cdots \\ x = SHL(x,4) \end{cases}.$$

SHL 为左移指令,表示连续两次将 x 左移 4 位,两次左移达到 8 位,将发生对字节的污点传播.然而本文中实现的方法是以字节为粒度,只对左移 8 位以上才对污点数据进行移位.对这种情况无法感知,因此会发生污点信息传播不当的问题.

5 相关工作

近年来,国内外很多学者都围绕污点分析方法进行了较深入的研究[12−22].Yin 等人[13]提出了基于 QEMU 虚拟机的扩展平台 TEMU,可解决动态分析的困难,便于在其上进行程序分析.TEMU 使用全系统的视角,可分析内核中的活动与多进程间的交互;提供了很好的隔离性与透明性,使恶意程序难以检测到分析环境并影响分析结果,以细粒度的方式进行深度分析.

Schwartz 等人[15]描述了一种基于中间语言上的污点传播策略.

1) 污点引入:实现了一个简单的用户输入源——*get_input* 调用,表示从系统调用、库调用返回的结果,不同的输入源的污点引入规则也不同;
2) 污点检查:污点的状态值用以确定程序的异常行为,通过添加此类规则到操作语义的前提条件中进行检查;
3) 污点传播策略:内存操作包括内存的地址和内存单元两类值,污点传播策略分别追踪内存地址和内存单元的值;
4) 污点消除.如程序函数计算的结果恒为常量时,则需要消除污点.

Enck 等人[16]描述了 TaintDroid,一个基于动态污点传播方法的移动手机的扩展平台,可追踪私密的敏感信息流.其特性包括:(1) 借助虚拟机的解释器提供变量级的追踪,使用解释器提供的变量语义来避免 Intel x86 指令的污点信息扩散.对变量进行修改,并维护其污点标记;(2) 实现应用程序间的消息追踪,可减小进程间通信的成本;(3) 对于系统提供的库,使用函数级的追踪.直接运行库中的代码,并在其返回时修改污点传播策略,需事先可知函数的语义信息;(4) 使用文件级的追踪方法,确保持久信息保持其正确的污点标记.

Kang 等人[17]提出一种动态的污点传播方案 DTA++,包括两个阶段:首先通过诊断,产生不完全污点的分支生成规则,并确定离线分析所需额外的传播;其次,在以后动态污点分析中应用那些规则.其基本原理是:查找不完全污点的控制流的路径,由于确定输入需更多细粒度的信息,使用符号执行和基于路径谓词的方法.在符号执行中,指令执行的踪迹对应包括一系列分支判定的程序路径约束.查找分为两部分:检测谓词Φ确定一个路径子串中是否含有引发错误的隐式流;然后查找Φ中第 1 个路径前缀.

北京大学的王铁磊等人提出了 TaintScope[18]和 IntScope,其中,TaintScope[19]用于定位程序中的校验和,而 IntScope 的主要贡献是利用污点分析方法来检测整数溢出漏洞.其实现方法是:通过反编译方法,将二进制代码

翻译成为静态单赋值形式的中间表示,建立控制流图和调用图,使用深度优先遍历控制流图.对于分支,则使用求解器计算当前路径约束是否满足该分支的条件.

北京航空航天大学的忽朝俭等人[20]提出并深入分析了驱动程序中频繁出现的写污点值到污点地址漏洞模式,提出了一种针对该种漏洞模式的基于反编译、数据流分析和污点分析的检测方案,并实现一款既可分析本地二进制代码,也可分析 C 源代码的原型工具.

上述传统的污点分析方法为本文的研究工作提供了坚实的理论基础和实现细节,但其中大多数污点分析方法不支持浮点指令,执行效率较低,且传播的精度也不够高.本文为解决这些问题,在前人的工作基础上进行更进一步的探索与研究.

6 结束语

本文实现了一种基于执行踪迹离线索引的污点分析方法,支持污点标签,并以字节为粒度进行污点传播.离线索引可跳过与污点数据无关的指令,极大地提高了污点分析的效率.本文首次描述并解决了由 JIT 翻译执行导致的污点丢失问题.利用污点标签,可标识污点来源于何种外部输入及其对应外部输入的哪些字节.以字节为粒度增加了污点传播的精度,保证污点信息以较合理、准确的方式从源操作数传递到目的操作数.

利用本文工具与 TEMU 在 12 个真实软件漏洞的对比检测结果表明:本文提出的污点分析方法更为完善,具备更强的漏洞检测能力和更高的分析效率,可检测和验证大量的已知漏洞.目前,本文工具已部署到检测 0day 漏洞的系统中.本文最后讨论了在实验过程中遇到的一些棘手的问题,这些问题可能导致污点信息的传播发生偏差,期望在以后的研究过程中逐步解决这些问题.

References:

[1] Mei H, Wang QX, Zhang L, Wang J. Software analysis: A road map. Chinese Journal of Computers, 2009,32(9):1697−1710 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2009.01697]

[2] Luk C, Cohn R, Muth R, Harish P, Klauser A, Lowney G, Wallace S, Reddi VJ, Hazelwood K. Pin: Building customized program analysis tools with dynamic instrumentation. In: Proc. of the ACM Conf. on Programming Language Design and Implementation. New York: ACM Press, 2005. 190−200. [doi: 10.1145/1065010.1065034]

[3] Lueck G, Patil H, Pereira C. PinADX: An interface for customizable debugging with dynamic instrumentation. In: Proc. of the IEEE/ACM Int'l Symp. on Code Generation and Optimization. New York: ACM Press, 2012. 114−123. [doi: 10.1145/2259016.2259032]

[4] Roy A, Hand S, Harris T. Hybrid binary rewriting for memory access instrumentation. In: Proc. of the 7th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. New York: ACM Press, 2011. 227−238. [doi: 10.1145/1952682.1952711]

[5] Skaletsky A, Devor T, Chachmon N, Cohn R, Hazelwood K, Vladimirov V, Bach M. Dynamic program analysis of microsoft windows applications. In: Proc. of the Int'l Symp. on Performance Analysis of Software and Systems. Timisoara: IEEE, 2010. 389−400. [doi: 10.1109/ISPASS.2010.5452079]

[6] Patil H, Pereira C, Stallcup M, Lueck G, Cownie J. PinPlay: A framework for deterministic replay and reproducible analysis of parallel programs. In: Proc. of the 8th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization. New York: ACM Press, 2010. 1020−1034. [doi: 10.1145/1772954.1772958]

[7] Song D, Brumley D, Yin H, Caballero J, Jager I, Kang MG, Liang ZK, Newsome J, Poosankam P, Saxena P. BitBlaze: A new approach to computer security via binary analysis. In: Proc. of Int'l Conf. on Information Systems Security. Hyderabad: Springer-Verlag, 2008. 1−25. [doi: 10.1007/978-3-540-89862-7_1]

[8] Brumley D, Poosankam P, Song D, Zheng J. Automatic patch-based exploit generation is possible: Techniques and implications. In: Proc. of the IEEE Symp. on Security and Privacy. California: IEEE, 2008. 78−102. [doi: 10.1109/SP.2008.17]

[9] Yin H, Liang Z, Song D. HookFinder: Identifying and understanding malware hooking behaviors. In: Proc. of the 15th Annual Network and Distributed System Security Symp. San Diego: Internet Society, 2008. 103−119.

[10] Caballero J, Yin H, Liang Z, Song D. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In: Proc. of the 14th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2007. 567−581. [doi: 10.1145/1315245.1315286]

[11] Yin H, Song D, Egele M, Kruegel C, Kirda E. Panorama: Capturing system-wide information flow for malware detection and analysis. In: Proc. of the ACM Conf. on Computer and Communication Security. New York: ACM Press, 2007. 103−119. [doi: 10.1145/1315245.1315261]

[12] Kang M, Poosankam P, Yin H. Renovo: A hidden code extractor for packed executables. In: Proc. of the 5th ACM Workshop on Recurring Malcode. New York: ACM Press, 2007. 327−341. [doi: 10.1145/1314389.1314399]

[13] Ma JX, Li ZJ, Hu CJ, Zhang JX, Guo T. Research of array type abstraction reconstruction in binary code. Journal of Tsinghua University: Science and Technology, 2012,10(1):1329−1334 (in Chinese with English abstract). [doi: 10.16511/j.cnki.qhdxxb.2012.10.003]

[14] Ma JX, Li ZJ, Hu CJ, Zhang JX, Guo T. A reconstruction method of type abstraction in binary code. Journal of Computer Research and Development, 2013,50(11):2418−2428 (in Chinese with English abstract).

[15] Schwartz J, Avgerinos T, Brumley D. All you ever wanted to know about dynamic taint analysis and forward symbolic execution. In: Proc. of the IEEE Security and Privacy Symp. 2010. 723−734. [doi: 10.1109/SP.2010.26]

[16] Enck W, Gilbert P, Chun BG, Cox LP, Jung J, McDaniel P, Sheth AN. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In: Proc. of the USENIX Symp. on Operating Systems Design and Implementation. Vancouver: USENIX, 2010. 817−822. [doi: 10.1145/2619091]

[17] Kang M, McCamant S, Poosankam P, Song D. DTA++: Dynamic taint analysis with targeted control-flow propagation. In: Proc. of the 18th Annual Network and Distributed System Security Symp. San Diego: Internet Society, 2011. 913−926.

[18] Wang TL, Wei T, Gu GF, Zou W. TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability. In: Proc. of the 2010 IEEE Symp. on Security and Privacy. California: IEEE, 2010. 497−512. [doi: 10.1109/SP.2010.37]

[19] Wang TL, Wei T, Lin ZQ, Zou W. IntScope: Automatically detecting integer overflow vulnerability in x86 binary using symbolic execution. In: Proc. of the 16th Network and Distributed System Security Symp. San Diego: Internet Society, 2010. 333−347.

[20] Hu CJ, Li ZJ, Guo T, Shi ZW. Detecting the vulnerability pattern of writing tainted value to tainted address. Journal of Computer Research and Development, 2011,48(8):1455−1463 (in Chinese with English abstract).

[21] Babak Y, Saumya D. Symbolic execution of obfuscated code. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security. New York: ACM Press, 2015. 732−744. [doi: 10.1145/2810103.2813663]

[22] Michelle W, David L. IntelliDroid: A targeted input generator for the dynamic analysis of android malware. In: Proc. of the 22nd Network and Distributed System Security Symp. San Diego: Internet Society, 2016. 481−496.

附中文参考文献:

[1] 梅宏,王千祥,张路,王戟.软件分析技术进展.计算机学报,2009,32(9):1697−1710. [doi: 10.3724/SP.J.1016.2009.01697]

[13] 马金鑫,忽朝俭,李舟军,张俊贤,郭涛.二进制代码中数组类型抽象的重构方法.清华大学学报:自然科学版,2012,10(1):1329−1334. [doi: 10.16511/j.cnki.qhdxxb.2012.10.003]

[14] 马金鑫,李舟军,忽朝俭,张俊贤,郭涛.一种重构二进制代码中类型抽象的方法.计算机研究与发展,2013,50(11):2418−2428.

[20] 忽朝俭,李舟军,郭涛,时志伟.写污点值到污点地址漏洞模式检测.计算机研究与发展,2011,48(8):1455−1463.

马金鑫(1986－),男,山西吕梁人,博士,副研究员,主要研究领域为软件安全,程序分析.

李舟军(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网络与信息安全,数据挖掘与智能信息处理.

张涛(1977－),男,博士,研究员,主要研究领域为软件安全,漏洞分析.

沈东(1989－),男,博士生,CCF 学生会员,主要研究领域为移动安全,虚拟化安全.

章张锴(1990－),男,博士生,主要研究领域为信息安全.