2. 遍历 EBS,计算原子动作 a 在 EBS 安全策略控制下的预期执行动作β=*wexpected*(a,w).根据 *wexpected*(·)的定义,最终有β=a(EBS 安全策略预期允许执行 a)或者β=Λ(EBS 安全策略预期不允许执行 a).
3. 如果β=a,则进程真实执行与预期执行一致(真实执行动作和预期执行动作都是 a,两者执行之后状态一致,故在 EBS 关注的 w 上等价),根据推论 1,u_p 执行 a 后可信,转步骤 1;如果β=Λ,则进程真实执行与预期执行不一致(真实执行动作 a,预期执行空动作Λ,两者执行之后导致状态空间不一致,故在 EBS 所关注的 w 上不等价),u_p 执行 a 后不可信,报警.

算法 1 结束.算法 1 只需要在每次系统调用 a 执行时,判定 a 是否被 EBS 安全策略所允许即可,因而其时间复杂度为 O(1).

4 实验评估

实验平台采用 TOSHIBA 笔记本电脑,CPU Intel i5 2.5G,内存 Samsung 8G.操作系统 Linux Fedora,内核版本 2.6.43.8-1.fc15.i686.PAE.实验测试了在 ROP 攻击以及传统代码多态、代码混淆、代码变形和代码编码等对抗形式下对攻击行为的可信度量功能.表 1 给出了实验结果.

Table 1 Attacks that can be measured and protected in real time

表 1 可被实时度量和防护的攻击方法

攻击方法	是否可以实时度量和防护
ROP(return-oriented programming)	√
代码多态(polymorphism)	√
代码混淆(obfuscation)	√
代码变形(mutation)	√
代码编码(encoding)	√

表 1 显示,本文的方法可以检测到 ROP(return-oriented programming)攻击.ROP 代表了现代攻击的发展趋势,其不寻求注入代码,而是从已有的“干净代码”中寻找合适的“指令片段”(gadget),通过“ret 或者与 ret 语义等价的指令”将前述指令片段链接形成完整的攻击代码,以达成攻击目的.ROP 可以高度自动化地构造图灵完全的攻击,危害巨大.为此,研究人员提出了多种对抗方法[49−53],取得了长足的进展.早期的 ROP 防御侧重于检测可疑的 ret 指令,但没有处理与 ret 语义等价的指令(如 jmp/pop 等)[49],对 ROP 的防御是不完全的.随后的工作实现了对各类 ROP 攻击的全覆盖,其基本思想在于消除 ROP 指令片段,或者检测并阻止(ROP 指令片段所导致)的非法间接跳转.但这些方法均需要额外的条件,如,需要被保护应用的非直接跳转指令和地址[50],或者被保护应用的指令片段 IG(instruction & gadget)信息[51];需要硬件的特殊功能[52];需要特殊的编译器[53];需要对受保护应用进行重写[50,53]等等.不同于这些工作,本文的方法基于如下观察:ROP 是一种代码重用方法,其本身并不能达成攻击,而仍然要通过调用敏感系统调用来实现恶意功能.因此,通过度量应用程序系统调用与 EBS 规范的偏离可以发现基于 ROP 技术所发起的攻击.严格来说,本文方法并不能检测 ROP 攻击本身,这是与文献[49−53]相比的不足之处.但从另外一个角度,在面向大量商业应用和通用度量平台的一般情况下——此时可能难以进行预处理(因而难以获得应用本身的信息[50,51]),不一定具备特殊硬件和软件(因而不一定满足特定功能的 CPU[52]和编译器[53]要求),且不一定能够进行代码重写(如开启了完整性度量架构 IMA),本文的方法可以作为上述方法的补充,结合应用.

对于表 1 中余下的 4 种经典攻击代码对抗方法,实验结果表明亦可以有效度量.从实验结果还可以发现,通过分析和抽取已有 shellcode 的特征,利用较少的 EBS 规范(实验中采用了 7 条主要 EBS 规范),就可以覆盖主要的攻击流程和大量的攻击代码,实现对软件可信性的实时度量,这表明方法是高效的.

表 2 对实时度量的时间负载进行了分析.实时度量所引起的时间负载由两部分构成:一是捕获系统调用及其参数所花的时间 t_1;二是分析系统调用是否偏离了 EBS 安全策略所花的时间 t_2.t_2 主要是字符串比较等指针操作,时间相对较快,大致取 t_2≈t_1.再假设在未进行实时度量时(no real-time measurement)软件的运行时间为 T_{NRM}=

t_0,则进行实时度量时(real-time measurement)软件的运行时间为 $T_{RM}=t_0+t_1+t_2\approx T_{NRM}+2\times t_1$,此时,可求时间负载$\zeta$为如下等式(7),故只要得到 t_1/t_0 即可.

$$\zeta=T_{RM}/T_{NRM}=1+2\times t_1/t_0 \tag{7}$$

利用 time 命令和 strace 命令,可以立即求出 $T_0=t_0$ 以及 $T_1=t_0+t_1$,两者相比有:

$$t_1/t_0=T_1/T_0-1 \tag{8}$$

将等式(8)代入等式(7),即得ζ.

注意,time 命令会给出 3 种时间.

(1) 真实时间(real):软件从开始执行到结束执行的时间.

(2) 用户 CPU 时间(user):软件在用户态花费的 CPU 时间.

(3) 系统 CPU 时间(sys):软件在内核态花费的 CPU 时间.

那么对应可以定义 $T_{r0},T_{r1};T_{u0},T_{u1};T_{s0},T_{s1}$.由于真实时间包含了软件由于等待如 I/O 等资源的挂起时间,其意义不大,一般关注后两者(后两者之和 user+sys 给出了软件运行的真实 CPU 时间),则等式(7)和等式(8)可以对应改写如下.

$$\zeta=(\zeta_u+\zeta_s)/2,\zeta_u=1+2\times t_{u1}/t_{u0},\zeta_s=1+2\times t_{s1}/t_{s0} \tag{9}$$

$$t_{u1}/t_{u0}=T_{u1}/T_{u0}-1,t_{s1}/t_{s0}=T_{s1}/T_{s0}-1 \tag{10}$$

表 2 给出了 T_{u1}/T_{u0} 和 T_{s1}/T_{s0} 的实验结果.

Table 2 Time-load analysis of real-time measurement

表 2 实时度量时间负载分析

用户时间比(user time ratio):T_{u1}/T_{u0}			系统时间比(sys time ratio):T_{s1}/T_{s0}		
Max	Min	Average	Max	Min	Average
15.00	1.00	5.05	11.50	1.30	5.58

取平均时间比,将 $T_{u1}/T_{u0}=5.05$ 和 $T_{s1}/T_{s0}=5.58$ 代入等式(9),有 $t_{u1}/t_{u0}=4.05$ 以及 $t_{s1}/t_{s0}=4.58$;再代入等式(8),立即可得:$\zeta_u=9.1,\zeta_s=10.16,\zeta=9.63$.因此在实时度量时,一般情况下,用户时间负载$\zeta_u$约为 9.10 倍,系统时间负载$\zeta_s$约为 10.16 倍,总体时间负载$\zeta_s$约为 9.63 倍.

以上各类时间负载总体约为 10 倍,但这已经本方法所能达到的接近最优结果,且实际测试表明,该负载并不影响用户的实际使用体验.原因如下.

(1) 实验所选取的测试数据是接近最坏情况下的测试数据.本文方法对系统调用进行捕获和度量,对其他函数和指令则不做任何处理,显然,被测试代码中系统调用相关代码所占的比例越高,则对时间负载影响越大.注意到 shellcode 正好满足上述特征:考虑到存储空间,shellcode 会尽可能短小精干,去除冗余指令以调用系统调用完成攻击功能.换句话说,系统调用相关代码所占比例高.因此,本文直接选择 shellcode 代码为测试对象,所得的 10x 时间负载约为最坏情况下的结果.一般情况下,测试结果表明,时间负载会介于 1.6x~10x 之间.

(2) 前面已说明,时间负载由两部分构成:捕获系统调用和参数的 t_1 以及分析系统调用是否偏离 EBS 的 t_2,且取 $t_2\approx t_1$.实验中,为方便起见,使用 strace 命令获取系统调用和参数信息(strace 仍然基于 ptrace 实现).由于 strace 是 Linux 标准命令,因此 t_1 可以认为是优化后的最优结果,从而基于表 2 所计算到的时间负载ζ_u,ζ_s和ζ也可以认为是接近最优的结果.

(3) 以用户交互性强的应用为测试对象表明,本文方法所引起的时间负载并不影响用户使用感受.以 Firefox 为例,利用“time strace firefox”命令启动火狐浏览器并浏览文本、图片和视频等各类网页,统计时间信息并计算可得:若将 real time 视作 1,则 user time 约占 23%,sys time 约占 8%,即陷入内核 CPU 的运行时间(sys time)仅仅约占 8%,由于内核 CPU 时间(sys time)由系统调用和非系统调用时间两部分构成,因而实际捕获系统调用和参数信息的时间不到 8%.

根据测试结果,上述时间占比不影响用户使用感受.

5 结 论

运行时可信度量是可信计算有待解决的一个关键问题,本文提出了一种基于无干扰的软件实时可信度量方法,尝试为可信度量的理论构建和实践应用提供一种新的思路.选择无干扰模型是因为研究表明:无干扰模型可以以一种统一的形式描述“行为符合预期、完整性和机密性”等不同的主流“可信”定义,从而可以在统一的框架下对不同内涵的可信性进行研究.在此基础上,给出了行为可信性的实时判定算法(时间复杂度 $O(1)$),解决了无干扰模型难以应用于实时系统的难题.针对理论的实践应用问题,根据无干扰模型的定义,选择系统调用作为基本原子动作,将软件真实行为定义为原子动作的序列,并根据系统调用所属安全域之间的无干扰关系定义预期行为规范 EBS.最后,通过判定软件真实行为与预期行为之间是否存在偏差,实现对软件行为的实时可信度量.原型实验证明了本文方法的有效性.

后续的研究拟从如下几个方面入手:一是探索如何结合人工智能技术自动抽取恶意代码样本特征并形成预期行为规范库;二是研究适用于实时或者准实时环境的漏洞定位技术;三是研究移动智能终端、物联网等环境下的实时可信度量和防护问题.

References:

[1] Shen CX, Zhang HG, Wang HM, Wang J, Zhao B, Yan F, Yu FJ, Zhang LQ, Xu MD. Research on trusted computing and its development. Science in China: Information Science, 2010,53(3):405−433.

[2] Feng DG, Qin Y, Wang D, Chu XB. Research on trusted computing technology. Journal of Computer Research and Development, 2011,48(8):1332−1349 (in Chinese with English abstract).

[3] Trusted Computing Group. TCG architecture overview (Revision 1.4). 2007. https://trustedcomputinggroup.org/wp-content/uploads/TCG_1_4_Architecture_Overview.pdf

[4] Sailer R, Zhang XL, Jaeger T, Doorn VL. Design and implementation of a TCG-based integrity measurement architecture. In: Blaze M, ed. Proc. of the 2004 USENIX Security Symp. (Security 2004). Berkeley: USENIX Association Press, 2004. 223−238.

[5] Shi E, Perrig A, Doorn LV. BIND: A fine-grained attestation service for secure distributed systems. In: Martin DC, ed. Proc. of the 2005 IEEE Symp. on Security and Privacy (Oakland 2005). New York: IEEE Computer Society Press, 2005. 154−168.

[6] Jaeger T, Sailer R, Shankar U. PRIMA: Policy-reduced integrity measurement architecture. In: Ferraiolo D, ed. Proc. of the ACM Symp. on Access Control Models and Technologies (SACMAT 2006). New York: ACM Press, 2006. 19−28.

[7] Loscocco PA, Wilson PW, Pendergrass JA, McDonell CD. Linux kernel integrity measurement using contextual inspection. In: Ning P, ed. Proc. of the 2007 ACM Workshop on Scalable Trusted Computing (STC 2007). New York: ACM Press, 2007. 21−29.

[8] Carbone M, Cui WD, Lu L, Lee WK, Peinado M, Jiang XX. Mapping kernel objects to enable systematic integrity checking. In: Al-Shaer E, ed. Proc. of the 2009 ACM Conf. on Computer and Communications Security (CCS 2009). New York: ACM Press, 2009. 555−565.

[9] Schiffman J, Moyer T, Shal C, Jeger T, McDaniel P. Justifying integrity using a virtual machine verifier. In: Gates C, ed. Proc. of the 2009 IEEE Annual Computer Security Applications Conf. (ACSAC 2009). New York: IEEE Computer Society Press, 2009. 83−92.

[10] Kil C, Sezer EC, Azab AM, Ning P, Zhang X. Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. In: Verissimo P, ed. Proc. of the 2009 IEEE/IFIP Int'l Conf. on Dependable Systems & Networks (DSN 2009). New York: IEEE Computer Society Press, 2009. 115−124.

[11] Xu W, Ahn GJ, Hu H, Zhang X, Seifert JP. DR@FT: Efficient remote attestation framework for dynamic systems. In: Gritzalis D, Preneel B, Theoharidou M, eds. Proc. of the 2010 European Conf. on Research in Computer Security (ESORICS 2010). Berlin: Springer Press, 2010. 182−198.

[12] John B, Corey K, Xeno K, Amy H. BIOS chronomancy: Fixing the core root of trust for measurement. In: Sadeghi AR, ed. Proc. of the 2013 ACM Conf. on Computer and Communications Security (CCS 2013). New York: ACM Press, 2013. 35−36.

[13] Zhang ZK, Ding XH, Tsudik G, Cui JH, Li ZJ. Presence attestation: The missing link in dynamic trust boostrapping. In: Thuraisingham B, ed. Proc. of the 2017 ACM Conf. on Computer and Communications Security (CCS 2017). New York: ACM Press, 2017. 89−102.

[14] Abera T, Asokan N, Davi L, Ekberg, JE, Nyman T, Paverd A, Sadeghi AR, Tsudik G. C-flat: Control-flow attestation for embedded systems software. In: Weippl E, ed. Proc. of the 2016 ACM Conf. on Computer and Communications Security (CCS 2016). New York: ACM Press, 2016. 743−754.

[15] Zhang J, Yuan F, Xu Q. DeTrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans. In: Ahn GJ, ed. Proc. of the 2014 ACM Conf. on Computer and Communications Security (CCS 2014). New York: ACM Press, 2014. 153−166.

[16] Matetic S, Ahmed M, Kostiainen K, Dhar A, Sommer D, Gervais A, Juels A, Capkun S. ROTE: Rollback protection for trusted execution. In: Kirda E, Ristenpart T, eds. Proc. of the 2017 USENIX Security Symp. (Security 2017). Berkeley: USENIX Association Press, 2017. 1289−1306.

[17] Yu FJ, Chen L, Zhang HG. Virtual trusted platform module dynamic trust extension. Ruan Jian Xue Bao/Journal of software, 2017, 28(10):2782−2796 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5174.htm [doi: 10.13328/j.cnki.jos.005174]

[18] Deng L, Zeng QK. Inner TCB based application protection. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):1042−1058 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5016.htm [doi: 10.13328/j.cnki.jos.005016]

[19] Deng L, Zeng QK. Method to efficiently protect applications from untrusted OS kernel. Ruan Jian Xue Bao/Journal of Software, 2016,27(5):1309−1324 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5017.htm [doi: 10.13328/j.cnki.jos.005017]

[20] Lin J, Liu CY, Fang BX. IVirt: Runtime environment integrity measurement mechanism based on virtual machine introspection. Chinese Journal of Computers, 2015,38(1):191−203 (in Chinese with English abstract). http://cjc.ict.ac.cn/qwjs/view.asp?id=4414

[21] Chen ZF, Li QB, Zhang P, Wang W. Kernel integrity measurement method based on memory forensic. Ruan Jian Xue Bao/Journal of Software, 2016,27(9):2443−2458 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4875.htm [doi: 10.13328/j.cnki.jos.004875]

[22] Zhang X, Huang Q, Shen CX. A formal method based on noninterference for analyzing trust chain of trusted computing platform. Chinese Journal of Computers, 2010,33(1):74−81 (in Chinese with English abstract). http://cjc.ict.ac.cn/qwjs/view.asp?id=3015

[23] Zhang X, Chen YL, Shen CX. Non-interference trusted model based on processes. Journal on Communications, 2009,30(3):6−11 (in Chinese with English abstract).

[24] Qin X, Chang CW, Shen CX, Gao L. Research on trusted terminal computer model tolerating untrusted components. Acta Electronica Sinica, 2011,39(4):1−6 (in Chinese with English abstract).

[25] Rushby J. Noninterference, transitivity, and channel-control security. Technical Report, CSL-92-02, Menlo Park: SRI Int'l, 1992. 1−47.

[26] Eggert S, Meyden RVD, Schnoor H, Wilke T. The complexity of intransitive noninterference. In: Fríncke D, ed. Proc. of the 2011 IEEE Symp. on Security and Privacy (Oakland 2011). New York: IEEE Computer Society Press, 2011. 196−211.

[27] Hadj-Alouane NB, Lafrance S, Lin F, Mullins J, Yeddes MM. On the verification of intransitive noninterference in multilevel security. IEEE Trans. on Systems, Man, and Cybernetics—Part B: Cybernetics, 2005,35(5):948−958.

[28] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In: Ning P, ed. Proc. of the 2007 ACM Conf. on Computer and Communications Security (CCS 2007). New York: ACM Press, 2007. 552−561.

[29] Goguen JA, Meseguer J. Security policy and security models. In: Schell R, ed. Proc of the 1982 IEEE Symp. on Security and Privacy (Oakland'82). New York: IEEE Computer Society Press, 1982. 11−20.

[30] Meyden RVD. What, indeed, is intransitive noninterference? In: Biskup J, López J, eds. Proc. of the 2007 European Symp. on Research in Computer Security (ESORICS 2007). Berlin: Springer-Verlag, 2007. 235−250.

[31] Ryan PYA. Mathematical models of computer security. In: Focardi R, Gorrieri R, eds. Proc of the 2001 Foundations of Security Analysis and Design. Berlin: Springer-Verlag, 2001. 1−62.

[32] Focardi R, Gorrieri R. Classification of security properties (Part I: Information flow). In: Focardi R, Gorrieri R, eds. Proc of the 2001 Foundations of Security Analysis and Design. Berlin: Springer-Verlag, 2001. 331−396.

[33] Zhang HG, Yan F, Fu JM, Xu MD, Yang Y, He F, Zhan J. Research on theory and key technology of trusted computing platform security testing and evaluation. Science in China: Information Science, 2010,40(2):167−188.

[34] Xu MD, Zhang HG, Zhao H, Li JL, Yan F. Security analysis on trust chain of trusted computing platform. Chinese Journal of Computers, 2010,33(7):1165−1176 (in Chinese with English abstract). http://cjc.ict.ac.cn/qwjs/view.asp?id=3120

[35] Toby M, Daniel M, Matthew B, Peter G, Timothy B, Sean S, Corey L, Xinn G, Gerwin K. SeL4: From general purpose to a proof of information flow enforcement. In: Sommer R, ed. Proc. of the 2013 IEEE Symp. on Security and Privacy (Oakland 2013). New York: IEEE Computer Society Press, 2013. 415−429.

[36] Mads D, Roberto G, Oliver S. Formal verification of information flow security for a simple ARM-based separation kernel. In: Sadeghi AR, ed. Proc. of the 2013 ACM Conf. on Computer and Communications Security (CCS 2013). New York: ACM Press, 2013. 223−234.

[37] Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA. A sense of self for Unix process. In: McHugh J, Dinolt G, eds. Proc. of the '96 IEEE Symp. on Security and Privacy (Oakland'96). New York: IEEE Computer Society Press, 1996. 120−128.

[38] Yang J, Sangho L, Evan D, Weiren W, Mattia F, Taesoo K, Alessandro O, Wenke L. RAIN: Refinable attack investigation with on-demand inter-process information flow tracking. In: Thuraisingham B, ed. Proc. of the 2017 ACM Conf. on Computer and Communications Security (CCS 2017). New York: ACM Press, 2017. 377−390.

[39] Enrico M, Lucky O, Panagiotis A, Emiliano DC, Gordon R, Gianluca S. MaMaDroid: Detecting Android malware malware by building markov chains of behavioral models. In: Bauer L, ed. Proc of 2017 ISOC Network and Distributed System Security Symp. (NDSS 2017). Reston: Internet Society Press, 2017. 1−15.

[40] Marko D, Simone A, Zvonimir R, Ivo U. Evaluation of Android malware detection based on system calls. In: Verma R, ed. Proc. of 2016 ACM Int'l Workshop on Security and Privacy Analytics (IWSPA 2016). New York: ACM Press, 2016. 1−8.

[41] Kimberly T, Salahuddin JK, Aristide F, Lorenzo C. CopperDroid: Automatic reconstruction of Android malware behaviors. In: Hutton T, ed. Proc of the 2015 ISOC Network and Distributed System Security Symp. (NDSS 2015). Reston: Internet Society Press, 2015. 1−15.

[42] Yang C, Xu ZY, Gu GF, Yegneswaran V, Porras P. DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in Android applications. In: Kutyłowski M, Vaidya J, eds. Proc. of the 2014 European Symp. on Research in Computer Security (ESORICS 2014). Berlin: Springer-Verlag, 2014. 163−182.

[43] Alessandro R, Aristide F, Lorenzo C. A system call-centric analysis and stimulation technique to automatically reconstruct Android malware behaviors. In: Stamatogiannakis M, ed. Proc. of the 2013 European Workshop on Systems Security. New York: ACM Press, 2013. 1−6.

[44] Calvin K, Timothy R. Noninterference and intrusion detection. In: Hinton H, ed. Proc. of 2002 IEEE Symp. on Security and Privacy (Oakland 2002). New York: IEEE Computer Society Press, 2002. 177−188.

[45] Zhang F, Zhang C, Chen W, Hu FN, Xu MD. Noninterference analysis of trust of behavior in cloud computing system. Chinese Journal of Computers, 2019,42(4):736−755 (in Chinese with English abstract). http://cjc.ict.ac.cn/online/onlinepaper/zf-20194179 2414.pdf

[46] George CN. Proof-carrying code. In: Lee P, ed. Proc. of the ACM '97 SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'97). New York: ACM Press, 1997. 1−14.

[47] George CN, Peter L. Safe kernel extensions without run-time checking. In: Peterson K, Zwaenepoel W, eds. Proc. of the '96 USENIX Symp. on OS Design and Implementation (OSDI'96). Berkeley: USENIX Association Press, 1996. 229−243.

[48] IBM DeveloperWorks. List of Linux system calls. 2002 (in Chinese). 2018. https://www.ibm.com/developerworks/cn/linux/kernel/ syscall/part1/appendix.html

[49] Li JK, Wang Z, Jiang XX, Grace M, Baharam S. Defeating return-oriented rootkits with "return-less" kernels. In: Morin C, ed. Proc. of the 2010 European Conf. on Computer Systems (EuroSys 2010). New York: ACM Press, 2010. 195−208.

[50] Zhang C, Wei T, Chen ZF, Duan L, Szekeres L, McCamant S, Song D, Zou W. Practical control flow integrity & randomization for binary executables. In: Sommer R, ed. Proc. of the 2013 IEEE Symp. on Security and Privacy (Oakland 2013). New York: IEEE Computer Society Press, 2013. 559−573.

[51] Cheng YQ, Zhou ZW, Yu M, Ding XH, Deng RH. Ropecker: A generic and practical approach for defending against ROP attacks. In: Hutton T. Proc. of the 2014 ISOC Network and Distributed System Security Symp. (NDSS 2014). Reston: Internet Society Press, 2014. 1−14.

[52] Vasilis P, Michalis P, Angelos DK. Transparent ROP exploit mitigation using indirect branch tracing. In: King S, ed. Proc. of the 2013 USENIX Security Symp. (Security 2013). Berkeley: USENIX Association Press, 2013. 447−462.

[53] Kaan O, Leyla B, Andrea L, Davide B, Engin K. G-free: Defending return-oriented programming through gadget-less binaries. In: Gates C, ed. Proc. of the 2010 IEEE Annual Computer Security Applications Conf. (ACSAC 2010). New York: ACM Press, 2010. 49−58.

附中文参考文献:

[2] 冯登国,秦宇,汪丹,初晓博.可信计算技术研究.计算机研究与发展,2011,48(8):1332−1349. http://crad.ict.ac.cn/CN/abstract/abstract2333.shtml

[17] 余发江,陈列,张焕国.虚拟可信平台模块动态信任扩展方法.软件学报,2017,28(10):2782−2796. http://www.jos.org.cn/1000-9825/5174.htm [doi: 10.13328/j.cnki.jos.005174]

[18] 邓良,曾庆凯.引入内可信基的应用程序保护方法.软件学报,2016,27(4):1042−1058. http://www.jos.org.cn/1000-9825/5016.htm [doi: 10.13328/j.cnki.jos.005016]

[19] 邓良,曾庆凯.一种在不可信操作系统内核中高效保护应用程序的方法.软件学报,2016,27(5):1309−1324. http://www.jos.org.cn/1000-9825/5017.htm [doi: 10.13328/j.cnki.jos.005017]

[20] 林杰,刘川意,方滨兴.IVirt:基于虚拟机自省的运行环境完整性度量机制.计算机学报,2015,38(1):191−203. http://cjc.ict.ac.cn/qwjs/view.asp?id=4414

[21] 陈志锋,李清宝,张平,王炜.基于内存取证的内核完整性度量方法.软件学报,2016,27(9):2443−2458. http://www.jos.org.cn/1000-9825/4875.htm [doi: 10.13328/j.cnki.jos.004875]

[22] 张兴,黄强,沈昌祥.一种基于无干扰模型的信任链传递分析方法.计算机学报,2010,33(1):74−81. http://cjc.ict.ac.cn/qwjs/view.asp?id=3015

[23] 张兴,陈幼雷,沈昌祥.基于进程的无干扰可信模型.通信学报,2009,30(3):6−11.

[24] 秦晰,常朝稳,沈昌祥,高丽.容忍非信任组件的可信终端模型研究.电子学报,2011,39(4):934−939.

[34] 徐明迪,张焕国,赵恒,李峻林,严飞.可信计算平台信任链安全性分析.计算机学报,2010,33(7):1165−1176.

[45] 张帆,张聪,陈伟,胡方宁,徐明迪.基于无干扰的云计算环境行为可信性分析.计算机学报,2019,42(4):736−755.

[48] IBM 开发工厂.Linux 系统调用.2002. https://www.ibm.com/developerworks/cn/linux/kernel/syscall/part1/appendix.html

附录. EBS 编写示例

攻击者在定位漏洞之后,往往会执行一些通用攻击操作,以达成进一步攻击目的,这些通用攻击操作包括(但不限于)生成 shell、开启反向连接、清空防火墙规则、关闭地址空间布局随机化 ASLR 等等.针对这些攻击,这里给出部分 EBS 示例如下:

1. EBS 规范 1:预期不能生成 shell.

$u_p \nrightarrow u_{[0][0]} == \{2\}$

FOR INDEX==2:{1==“*bin*/*bash*”∨1==“*bin*/*sh*”∨1==“*bin*/tc*bash*”∨1==“*bin*/*csh*”∨1==“*bin*/*dash*”∨}

$u_{[i][j]}$的下标分别是 i,j,其表明:i 是安全域 u_i 的编号,j 是 u_i 中子安全域编号.因此,这里 $u_{[i][j]}=u_{[0][0]}$的含义是 u_0 安全域中的第 1 个子安全域.类似的,如果是 u_1 安全域中的第 3 个子安全域,则表示为 $u_{[1][2]}$.

回到 $u_{[0][0]}=\{2\}$,查阅[48]可知,u_0 安全域为进程控制类系统调用,其中,编号 2 对应 execve,故 $u_{[0][0]}=\{2\}=\{execve\}$.再结合 **FOR INDEX==2**:语句的参数安全策略,由于 execve 的第 1 个参数是字符串类型,指明了要运行的二进制文件名,因此规范 1 的含义是:进程 u_p 预期不能运行/bin 目录下的 bash,sh,tcsh,csh 和 dash 文件以开启 shell.

解释:攻击者突破系统之后,一般需要开启 shell 以进一步执行命令.规范 1 度量并禁止了这种行为.需要说明的是:Linux 的 shell 并不只有上述 5 种类型(规范 1 是根据实验者机器的配置情况编写的),使用者可以根据实际安全需求容易地更新规范 1.

2. 规范 2:预期不能开启非法监听端口连接或者不能进行反向连接.

$u_p \nrightarrow u_{[5][0]} == \{2,3\}$

FOR INDEX==2:{2→*sin_port*!=*PORT_LIST*$_1$}

FOR INDEX==3:{2→*sa_family*==AF_INET,2→*sin_port*==*PORT_LIST*$_2$,2→*sin_addr*==*IP_LIST*}

规范 2 的含义是:进程 u_p 预期对子安全域 $u_{[5][0]}$无干扰.查阅文献[48]可知,u_5 中 2 和 3 号系统调用分别为 bind 和 connect.进一步查阅 bind 和 connect 参数信息,两者的第 2 个参数均是常量结构体指针 const struct sockaddr*,该结构体的内部元素 sa_familiy 指明了协议族;sin_port 指明了端口号;sin_addr 指明了远程连接的 IP 地址.结合 **FOR INDEX==2**:语句的参数安全策略可知,进程 u_p 预期对所有不在端口列表 *PORT_LIST*$_1$ 当中的端口都不能开放监听.类似地,结合 **FOR INDEX==3**:语句的参数安全策略可知,进程 u_p 预期不能对端口列表 *PORT_LIST*$_2$ 和地址列表 *IP_LIST* 中的机器发起远程连接.

解释:攻击者突破系统之后,往往会开启受害机器监听端口监听外部连接,以方便进一步入侵.为此,规范 2 的安全策略预期进程 u_p 不会开启必要端口(即 *PORT_LIST*$_1$ 中列出的端口)之外的任何端口,从而能够度量并禁止攻击者非法开启监听端口的行为.另一方面,在内网中,由于防火墙的存在,外网机向内网机的连接会触发警报.此时,攻击者往往会从受害的内网机向外网特定机器发起反向连接.规范 2 度量并禁止了这种反向连接(反弹端口)行为.

3. 规范 3:预期不能清空防火墙规则.

$u_p \nrightarrow u_{[0][1]} == \{2\}$

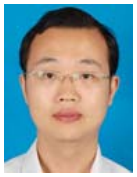
FOR INDEX==2:{1==/*sbin*/*iptables*}∧{2==[“/*sbin*/*iptables*”,“*–F*”]}

规范 3 的含义是:进程 u_p 预期对子安全域 $u_{[0][1]}$无干扰.由于规则 1 中 $u_{[0][0]}$和这里 $u_{[0][1]}$均是来自于 u_0 的子安全域,因此两者赋予不同的编号 j=0 和 j=1.类似地,查阅文献[48]可知:规范 3 表明,进程 u_p 预期不能带参数“*–F*”执行/*sbin*/*iptables* 文件以清空所有防火墙规则.

解释:规范 3 度量并禁止了清空防火墙规则行为.

限于篇幅,这里给出 3 个 EBS 示例.

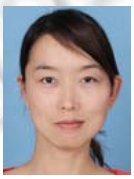
根据我们的分析结果,大多 shellcode 的代码有很强的功能重合性.换句话说,绝大多数 shellcode 的恶意功能都局限于生成 shell、开启反向连接、清空防火墙规则、关闭地址空间布局随机化 ASLR、修改/etc/passwd 文件、提升权限和重启等,只是其机器码形式或者对抗手段(多态、混淆、变形和编码)不一致.在我们的实验中,通过提炼这些重合功能,只需要不到 10 条通用 EBS,即可检测到大多数 shellcode,表明了方法的有效性.



张帆(1977－),男,湖北当阳人,博士,副教授,CCF 专业会员,主要研究领域为信息系统安全,软件安全,机器学习在网络空间安全中的应用.

张聪(1968－),男,博士,教授,主要研究领域为多媒体信号处理,模式识别,多媒体安全.

徐明迪(1980－),男,博士,研究员,CCF 专业会员,主要研究领域为网络空间安全,信息系统安全.



刘小丽(1981－),女,博士,讲师,主要研究领域为信息安全,移动计算.

赵涵捷(1963－),男,博士,教授,博士生导师,主要研究领域为移动计算,云计算,物联网,量子计算,网络及信息安全.

胡方宁(1976－),女,博士,讲师,主要研究领域为通信,嵌入式系统.