用 *func* 函数时,由于 *func* 函数中会调用 *read* 函数,此时 *scca*[0][0]增加 1 变为 8,但特征库中并没有符合 *scca*[0][0]=8 且 *procID*、*fID* 和 *NT* 分别为 128、5 和 1 的 RCP 存在,因此 BSV 失败.另外,测试过程中没有发现任何误报.通过上述测试,验证了 RCMon 方法检测代码复用攻击的可行性.同时,借助 RCMon 还可以对攻击的发生位置进行定位,这有助于攻击发生后的软件安全漏洞和攻击实现机制的分析.

Table 3 Results of attacking tests

表 3 攻击测试结果

复用代码	攻击是否成功	RCMon 报告信息	验证失败时的节点信息	是否存在误报
write	否	FSV 失败(WDC 超过临界值)	*procID*=128,*fID*=5,*NT*=3	无
read	否	FSV 失败(WDC 超过临界值)	*procID*=128,*fID*=5,*NT*=3	无
system	否	FSV 失败(WDC 超过临界值)	*procID*=128,*fID*=5,*NT*=3	无
func	否	BSV 失败(未找到匹配项)	*procID*=128,*fID*=5,*NT*=1	无

为了更好地验证 RCMon 的有效性,除上述测试外,我们还对 4 类真实的应用程序(nginx,proftpd,mcrypt 以及 TORQUE)进行了测试.测试中,分别采用来自 http://www.elis.ugent.be/□svolckae 的 4 种不同的 ROP 对上述程序实施攻击.

- 攻击 1 针对 Web 服务器 nginx,它首先读取栈中的 canary 值以及漏洞函数栈帧底部的返回地址,利用该地址计算出 nginx 的基地址,然后基于对 nginx 的已有知识构造 ROP 链,并利用 nginx 中的栈缓冲区溢出漏洞(CVE-2013-2028)使 ROP 链获得执行.
- 攻击 2 针对 ftp 服务器 proftpd,它首先通过扫描 proftpd 的可执行文件和 libc 库定位构造 ROP 链所需要的 gadget,然后从/proc/pid/maps 中读取 proftpd 和 libc 的加载地址来确定 gadget 的绝对地址,最后通过一个未授权的 FTP 链接将含有 ROP 链的 buffer 发送到 proftpd,并利用 proftpd 中的栈缓冲区溢出漏洞(CVE-2010-4221)使其得到执行.
- 攻击 3 针对 mcrypt(一个用于替换 crypt 的加密程序),它首先从/proc 接口获得 mcrypt 和 libc 库的加载地址来构造 ROP 链,然后通过一个 pipe 将 ROP 链发送给 mcrypt,并利用 mcrypt 中的栈缓冲区溢出漏洞(CVE-2012-4409)执行该 ROP 链.
- 攻击 4 针对 TORQUE 资源管理器服务器,它首先读取 pbs_server 进程的加载地址并构造 ROP 链,然后通过一个未授权的网络连接将该 ROP 链发送到 TORQUE,并利用 TORQUE 中的栈缓冲区溢出漏洞(CVE-2014-0749)使攻击得到执行.

测试前,首先分别处理各目标程序(植入监控代码),然后通过将处理后的程序在不同输入和操作条件下分别运行 20 次、40 次、60 次和 80 次(每次运行 30min)来构造程序运行特征库.然后,在关闭和开启 RCMon 特征监控两种情况下,分别使用上述 4 个 ROP 攻击对目标程序进行攻击.根据攻击是否成功,验证 RCMon 防御代码复用攻击的有效性.需要注意的是,由于植入监控代码的影响,原有的 4 个攻击在构造 ROP 链时所使用的地址都需要更新为代码植入后的新地址.最后,再在开启 RCMon 特征监控的情况下运行各目标程序 20 次(每次仍然运行 30min),以测试是否存在误报.最终的测试结果见表 4.

Table 4 Results of real applications tests

表 4 真实应用测试结果

目标程序	攻击是否成功		误报次数(20 次)	误报次数(40 次)	误报次数(60 次)	误报次数(80 次)
	关闭 RCMon	开启 RCMon				
nginx	√	×	3	2	2	1
proftpd	√	×	0	0	0	0
mcrypt	√	×	0	0	0	0
TORQUE	√	×	1	1	0	0

由表 4 可知,RCMon 能够有效防御针对 nginx 等应用程序的 4 种 ROP 攻击,无漏报;但 RCMon 存在误报,尤其是 nginx 误报情况较为明显.主要原因在于,该类程序运行特征的影响因素较多,不同服务状态和客户链接数量都会对服务器的执行过程产生影响.另外,从测试结果可以发现,在训练阶段的训练越充分,RCMon 产生误

报的可能性就越小.

另外,本文总结了当前已知的各类代码复用攻击技术,并根据其实现机制判断RCMon是否能够有效阻止这些攻击.分析过程中均假设这些攻击中含有对某些关键系统调用的执行过程或者这些攻击的实施改变了程序原有的关键系统调用执行情况,而这个假设对大多数攻击都是适用的[13,38].对于函数级复用攻击来说,不含任何关键系统调用的情况极少,另外,即使对于本身不含任何关键系统调用的指令片段复用攻击,在gadget跳转串联过程中也极有可能会干预原有的系统调用执行过程,如跳过某些执行关键系统调用的指令等.最终的分析结果见表 5.通过上述测试和分析可知,RCMon 可实现对已知各类代码复用攻击的有效防御.同时,还在表 5 中将RCMon与其他相关防御方法进行了防御能力的对比,对比结果进一步表明了RCMon的有效性.

Table 5 Analysis of RCMon's defensive ability

表 5 RCMond 的防御能力分析

攻击类型	攻击实质	ASLR[6−8]	Isomeron[11], Remix[12]	TASR[13]	CFI[14−17]	ROPecker[19] ROPdefender[21]	RCMon
return-into-libc[2]	函数复用	√	×	√	√	×	√
COOP[5]	虚函数复用	×	×	√	×	×	√
ROP[3]	gadget 复用 (以 ret 指令结尾)	√	√	√	√	√	√
JOP[4]	gadget 复用 (以 jump 指令结尾)	√	√	√	√	×	√
LOP[36]	函数复用	√	×	√	√	×	√
SROP[39]	gadget 复用且含 sigreturn gadget	√	√	√	√	×	√
CPROP[23]	gadget 复用(仅使用 call-preceded gadget)	√	√	√	√	×	√
JIT-ROP[10]	gadget 复用	×	√	×	√	×	√

3.2 性能测试

为了测试RCMon对整个系统性能的影响,本文采用SPEC CPU 2006的CFP 2006基准测试集中的部分测试程序对系统开销进行了测试.鉴于RCMon基于Hypervisor实现,本文分别测试了裸机、Hypervisor、Hypervisor下开启关键系统调用监控、测试程序部署未优化的RCMon以及部署优化后的RCMon这5种条件下的性能开销,测试结果如图9所示.

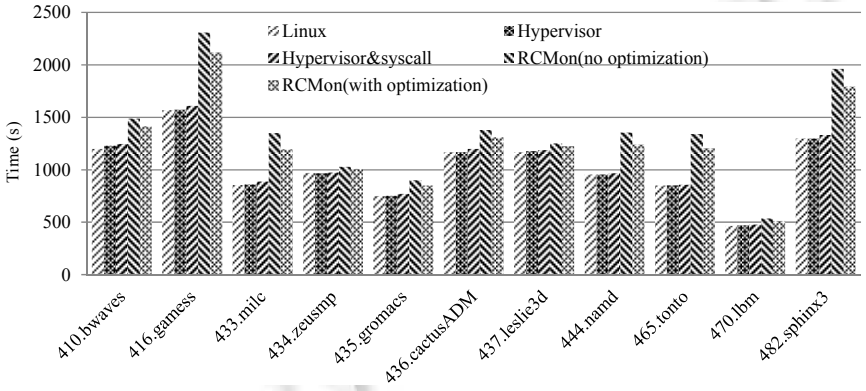

Fig.9 Results of performance overhead measurement

图 9 性能开销测试结果

由测试结果可知,本文所设计实现的Hypervisor由于采用了硬件辅助虚拟化技术Intel VT,并且尽可能地减少不必要的监控,使得该监控器在不监控内核函数时开销很小.当增加对关键系统调用的监控后,由于所有对关

键系统调用的执行都会产生陷入,并在陷入后执行对当前关键系统调用执行状态的更新和对函数入口原有的两条指令的模拟,因此开销略有增加.在对测试程序部署 RCMon 之后,由于不仅要监控系统中的所有关键系统调用,还要监控程序执行到每个关键节点处的陷入,并在陷入后根据当前节点信息到运行特征库中查找是否存在匹配项,当存在匹配项时,还要根据运行特征设置 WDC 对关键系统调用进行实时监控,因此,RCMon 引入后,测试程序的运行开销增加较多.与裸机(Linux 纯净系统)相比,当采用未优化的 RCMon 时,平均开销约为 31%(其中,434.zeusmp 开销最小,为 6%;而 465.tonto 开销最大,为 58%);当采用优化后的 RCMon 时,平均开销降至 22%,这也进一步验证了第 2.3 节中优化方法的有效性.

RCMon 的开销小于传统的内存安全保护方法(例如,SoftBound[24]的平均开销为 67%,Baggy Bounds Checking[25]的平均开销为 60%)、数据流完整性保护方法 DFI[30](平均开销高达 104%)以及控制流完整性方法 CCFI[15](平均开销 52%)等多类防御方法,略高于 CFI[14](平均开销 16%)以及 ASLR[6](的平均开销为 10%)等方法,但 RCMon 除了能够有效防御代码复用攻击以外,对所有含有系统调用恶意执行的攻击都具有一定的防御能力,防御的攻击类型更广,能够比这些防御方法提供更高的安全性.

4 RCMon 的局限性

由于 RCMon 不针对某个或某些具有特定特征的代码复用攻击类型进行防御,而是基于程序的系统调用运行特征防御代码复用攻击,是基于攻击的某种共性(即攻击者在实施恶意操作时难免要通过关键系统调用和操作系统进行交互)实施的检测和防御,因此无论是已知的还是未知的代码复用攻击,只要其实现过程包含对关键系统调用的使用或者其实现过程对原程序的关键系统调用执行过程造成了影响(即破坏了原程序的运行特征),RCMon 就能检测和防御.但 RCMon 仍然存在一些局限性.

(1) 如果一个代码复用攻击不使用任何关键系统调用且攻击实施过程不对原程序的关键系统调用执行过程产生任何影响,则可以绕过 RCMon 的检测,从而产生漏报.但这种情况下,代码复用攻击的发挥空间是非常有限的.首先,不借助任何关键系统调用的攻击所能实施的操作非常有限;另外,代码复用攻击的核心仍是对程序控制流的改变,而控制流改变的情况下要保证原程序关键系统调用执行过程不发生任何改变,这也使攻击变得更加难以实现.可见,虽然 RCMon 存在漏报某些攻击的可能,但该类攻击的危害性和实现的可能性都非常小,对 RCMon 的整体安全性的影响很小.而在第 3.1 节的测试中未发现任何漏报,这也在一定层度上验证了上述分析.

(2) 当在训练阶段构造的目标程序运行特征不够完备时,RCMon 可能会产生误报.由表 4 的测试结果可知,通过不断加大训练阶段的训练量,能够在较大程度上降低 RCMon 误报的可能.但这种方法效率低,而且存在很多不确定性.要在最大程度上消除误报,必须保证在训练阶段能够遍历目标程序所有的执行路径来构造完备的程序运行特征.拟在下一步工作中,利用动态符号执行技术来解决这一问题.动态符号执行能实现对程序的高路径覆盖,它以某个具体的输入来执行目标程序,并获取当前路径中的所有路径约束条件;然后通过对这些约束条件的修改,生成一条新的路径的约束条件,并用约束求解器获得一个新的输入;然后再以该输入执行目标程序.通过反复执行上述过程,即可实现对被测试对象所有路径的动态遍历.随着训练过程中的路径覆盖率的提高,就可构建更加完备的程序运行特征,从而消除误报.

5 总 结

针对当前代码复用攻击防御方法中存在的防御类型单一等问题,本文提出一种基于程序运行特征监控的代码复用攻击防御方法 RCMon.该方法首先定义并构造了程序的运行特征模型 RCMod(该模型包含了在程序中的函数调用位置前后和函数出入口这 4 类关键节点处的关键系统调用的运行特征信息),并基于该模型向目标程序的关键节点处植入监控代码,以实现基于 Hypervisor 的对目标程序运行过程的监控;然后,通过训练过程得到目标程序的运行特征库;最后,在实际运行过程中监控程序运行特征是否与事先得到的运行特征库中对应

特征一致,来判断程序是否遭到代码复用攻击,从而实现对代码复用攻击的有效防御.

RCMon 的验证过程包含基于特征库匹配的后向安全验证 BSV 和基于 WDC 的前向安全验证 FSV,分别实现了阶段性的安全验证和一个阶段内的实时安全验证,能有效防御函数级和指令级代码复用攻击,加强了目标程序的安全性.测试结果表明,RCMon 能够有效防御各类含关键系统调用执行过程的代码复用攻击,且实现过程不需要源码,实用性较好.另外,方法的平均性能开销也在可接受的范围内,约为 22%.

致谢 在此,由衷地感谢对本文研究工作提供基金支持的单位和参与本文评阅的审稿专家,同时向 Dyninst 研发团队以及其他为本文提供研究基础的前辈致敬.

References:

[1] Andersen, S, Abella, V. Changes to functionality in Microsoft Windows XP service pack 2, part 3: Memory protection technologies, data execution prevention. 2015. https://technet.microsoft.com/en-us/library/bb457155.aspx

[2] Nergal. The advanced return-into-lib(c) exploits: PaX case study. 2001. http://phrack.org/issues/58/4.html

[3] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In: Proc. of the 14th ACM Conf. on Computer and Communications Security. Alexandria: ACM Press, 2007. 552−561. [doi: 10.1145/1315245.1315313]

[4] Bletsch T, Jiang X, Freeh VW, Liang Z. Jump-oriented programming: A new class of code-reuse attack. In: Proc. of the 6th ACM Symp. on Information, Computer and Communications Security. Hong Kong: ACM Press, 2011. 303−307. [doi: 10.1145/1966913.1966919]

[5] Schuster F, Tendyck T, Liebchen C, Davi L, Sadeghi AR, Holz T. Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications. In: Proc. of the 36th IEEE Symp. on Security and Privacy. San Jose: IEEE CS Press, 2015. 745−762. [doi: 10.1109/SP.2015.51]

[6] PaX Team. PaX ASLR. 2003. http://pax.grsecurity.net/docs/aslr.txt

[7] Koo H, Polychronakis M. Juggling the gadgets: Binary-level code randomization using instruction displacement. In: Proc. of the 11th ACM on Asia Conf. on Computer and Communications Security. Xi'an: ACM Press, 2016. 23−34. [doi: 10.1145/2897845.2897863]

[8] Gupta A, Habibi J, Kirkpatrick MS, Bertino E. Marlin: Mitigating code reuse attacks using code randomization. IEEE Trans. on Dependable and Secure Computing, 2014,12(3):326−337. [doi: 10.1109/TDSC.2014.2345384]

[9] Fu JM, Liu XW, Tang Y, Li PW. Survey of memory address leakage and its defense. Journal of Computer Research and Development, 2016,53(8):1829−1849 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2016.20150526]

[10] Snow KZ, Monrose F, Davi L, Dmitrienko A. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization. In: Proc. of the 34th IEEE Symp. on Security and Privacy. Berkeley: IEEE CS Press, 2013. 574−588. [doi: 10.1109/SP.2013.45]

[11] Davi L, Liebchen C, Sadeghi AR, Snow KZ, Monrose F. Isomeron: Code randomization resilient to (just-in-time) return-oriented programming. In: Proc. of the 22nd Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2015. [doi: 10.14722/ndss.2015.23262]

[12] Chen Y, Wang Z, Whalley D, Lu L. Remix: On-demand live randomization. In: Proc. of the 6th ACM Conf. on Data and Application Security and Privacy. New Orleans: ACM Press, 2016. 50−61. [doi: 10.1145/2857705.2857726]

[13] Bigelow D, Hobson T, Rudd R, Streilein W, Okhravi H. Timely rerandomization for mitigating memory disclosures. In: Proc. of 22nd ACM Conf. on Computer and Communications Security. Denver: ACM Press, 2015. 268−279. [doi: 10.1145/2810103.2813691]

[14] Abadi M, Budiu M, Erlingsson U, Ligatti J. Control-flow integrity. In: Proc. of the 12th ACM Conf. on Computer and Communications Security. Alexandria: ACM Press, 2005. 315−326. [doi: 10.1145/1102120.1102165]

[15] Mashtizadeh AJ, Bittau A, Boneh D, Mazieres D. CCFI: Cryptographically enforced control flow integrity. In: Proc. of the 22nd ACM Conf. on Computer and Communications Security. Denver: ACM Press, 2015. 315−326. [doi: 10.1145/2810103. 2813676]

[16] Victor VDV, Andriesse D, Goktas E, Gras B. Practical context-sensitive CFI. In: Proc. of the 22nd ACM Conf. on Computer and Communications Security. Denver: ACM Press, 2015. 927−940. [doi: 10.1145/2810103.2813673]

[17] Chen ZF, Li QB, Zhang P, Wang Y. A kernel code reuse attack detection technique for Linux. Ruan Jian Xue Bao/Journal of Software, 2017,28(7):1732−1745 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5058.htm [doi: 10.13328/j.cnki.jos.005058]

[18] Carlini N, Barresi A, Payer M, Wagner D. Control-flow bending: On the effectiveness of control-flow integrity. In: Proc. of the 24th USENIX Conf. on Security Symp. Washington: USENIX Association, 2015. 161−176. https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-carlini.pdf

[19] Cheng YQ, Zhou ZW, Yu M, Ding XH, Robert HD. ROPecker: A generic and practical approach for defending against ROP attacks. In: Proc. of the 21st Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2014. [doi: 10.14722/ndss.2014.23156]

[20] Göktas E, Athanasopoulos E, Polychronakis M, Bos H, Portokalidis G. Size does matter: Why using gadget-chain length to prevent code-reuse attacks is hard. In: Proc. of the 23rd USENIX Conf. on Security Symp. San Diego: USENIX Association, 2014. 417−432. https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-goktas.pdf

[21] Davi L, Sadeghi AR, Winandy M. ROPdefender: A detection tool to defend against return-oriented programming attacks. In: Proc. of the 6th ACM Symp. on Information, Computer and Communications Security. Hong Kong: ACM Press, 2011. 40−51. [doi: 10.1145/1966913.1966920]

[22] Wicherski G. Taming ROP on Sandy Bridge. SyScan, 2013. https://infocon.org/cons/SyScan/SyScan%202013%20Singapore/SyScan%202013%20Singapore%20presentations/SyScan2013_DAY1_SPEAKER05_Georg_WIcherski_Taming_ROP_ON_SANDY_BRIDGE_syscan.pdf

[23] Carlini N, Wagner D. ROP is still dangerous: breaking modern defenses. In: Proc. of the 23rd USENIX Conf. on Security Symp. San Diego: USENIX Association, 2014. 385−399. https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-carlini.pdf

[24] Nagarakatte S, Zhao J, Martin MMK, Zdancewic S. SoftBound: Highly compatible and complete spatial memory safety for C. ACM SIGPLAN Notices, 2009,44(6):245−258. [doi: 10.1145/1543135.1542504]

[25] Akritidis P, Costa M, Castro M, Hand S. Baggy bounds checking: An efficient and backwards-compatible defense against out-of-bounds errors. In: Proc. of the 18th USENIX Security Symp. Montreal: USENIX Association, 2009. 51−66. https://www.usenix.org/legacy/event/sec09/tech/full_papers/akritidis.pdf

[26] Onarlioglu K, Bilge L, Lanzi A, Balzarotti D, Kirda E. G-free: Defeating return-oriented programming through gadget-less binaries. In: Proc. of the 26th Annual Computer Security Applications Conf. Austin: ACM Press, 2010. 49−58. [doi: 10.1145/1920261.1920269]

[27] Backes M, Holz T, Kollenda B, Koppe P, Nurnberger S, Pewny J. You can run but you can't read: Preventing disclosure exploits in executable code. In: Proc. of the 21st ACM Conf. on Computer and Communications Security. Scottsdale: ACM Press, 2014. 1342−1353. [doi: 10.1145/2660267.2660378]

[28] Kuznetsov V, Szekeres L, Payer M, Candea G, Sekar R, Dawn S. Code-pointer integrity. In: Proc. of the 11th USENIX Symp. on Operating Systems Design and Implementation. Broomfield: USENIX Association, 2014. 147−163. https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-kuznetsov.pdf

[29] Evans I, Fingeret S, Gonzalez J, Otgonbaatar U, Tang T, Shrobe H, Sidiroglou-Douskos S, Rinard M, Okhravi H. Missing the point (er): On the effectiveness of code pointer integrity. In: Proc. of the 36th IEEE Symp. on Security and Privacy. San Jose: IEEE CS Press, 2015. 781−796. [doi: 10.1109/SP.2015.53]

[30] Castro M, Costa M, Harris T. Securing software by enforcing data-flow integrity. In: Proc. of 7th Symp. on Operating Systems Design and Implementation. Seattle: USENIX Association, 2006. 147−160. https://www.usenix.org/legacy/event/osdi06/tech/full_papers/castro/castro.pdf

[31] Crane S, Liebchen C, Homescu A, Davi L, Larsen P, Sadeghi AR, Brunthaler S, Franz M. Readactor: Practical code randomization resilient to memory disclosure. In: Proc. of the 36th IEEE Symp. on Security and Privacy. San Jose: IEEE CS Press, 2015. 763−780. [doi: 10.1109/SP.2015.52]

[32] Seibert J, Okhravi H. Information leaks without memory disclosures: Remote side channel attacks on diversified code. In: Proc. of the 21st ACM Conf. on Computer and Communications Security. Scottsdale: ACM Press, 2014. 54−65. [doi: 10.1145/2660267.2660309]

[33] King M, Dave N. Automatic generation of hardware/software interfaces. In: Proc. of the 17th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. London: ACM Press, 2012. 325−336. [doi: 10.1145/ 2150976.2151011]

[34] Intel. Intel 64 and IA-32 architectures software developer's manual. 2016. http://www.intel.cn/content/dam/www/public/us/en/documents/ manuals/64-ia-32-architectures-software-developer-manual-325462.pdf

[35] Wang XL, Wang ZL, Sun YF, Liu Y, Zhang BB, Luo YW. Detecting memory leak via VMM. Chinese Journal of Computers, 2010, 33(3):463−472 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2010.00463]

[36] Lan B, Li Y, Sun H, Su C, Liu Y, Zeng QK. Loop-oriented programming: A new code reuse attack to bypass modern defenses. In: Proc. of the 14th IEEE Trustcom/BigDataSE/ISPA. Helsinki: IEEE CS Press, 2015. 190−197. [doi: 10.1109/Trustcom.2015.374]

[37] Dyninst9.1.0. 2016. https://github.com/dyninst/dyninst/releases

[38] Das S, Liu Y, Zhang W, Chandramohan M. Semantics-based online malware detection: Towards efficient real-time protection against malware. IEEE Trans. on Information Forensics and Security, 2017,11(2):289−302. [doi: 10.1109/TIFS.2015. 2491300]

[39] Bosman E, Bos H. Framing signals—A return to portable shellcode. In: Proc. of the 35th IEEE Symp. on Security and Privacy. Berkeley: IEEE CS Press, 2014. 243−258. [doi: 10.1109/SP.2014.23]

附中文参考文献:

[9] 傅建明,刘秀文,汤毅,李鹏伟.内存地址泄漏分析与防御.计算机研究与发展,2016,53(8):1829−1849. [doi: 10.7544/issn1000-1239.2016.20150526]

[17] 陈志锋,李清宝,张平,王烨.面向 Linux 的内核级代码复用攻击检测技术.软件学报,2017,28(7):1732−1745. http://www.jos.org.cn/1000-9825/5058.htm [doi: 10.13328/j.cnki.jos.005058]

[35] 汪小林,王振林,孙逸峰,刘毅,张彬彬,罗英伟.利用虚拟化平台进行内存泄露探测.计算机学报,2010,33(3):463−472. [doi: 10.3724/SP.J.1016.2010.00463]

张贵民(1987－),男,山东济南人,博士,讲师,主要研究领域为信息安全,可信计算.

张平(1969－),女,博士,副教授,主要研究领域为并行识别,并行编译,信息安全.

李清宝(1967－),男,博士,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.

程三军(1966－),男,高级工程师,CCF 专业会员,主要研究领域为计算机图形图像,网络安全.