基于函数注入的沙箱拦截识别方法

赵 旭* 颜学雄 王清贤 魏 强
(解放军信息工程大学 郑州 450002)
(数学工程与先进计算国家重点实验室 郑州 450002)

摘 要：沙箱验证机制的测试需要首先识别沙箱拦截，即识别沙箱截获的系统函数集。已有的 Hook 识别方法大多仅关注钩子的存在性，识别沙箱拦截的能力不足。该文设计了一种基于函数注入的沙箱拦截识别方法，该方法分析系统函数的指令执行记录(Trace)来识别沙箱截获的系统函数。首先，向不可信进程注入并执行系统函数来获取函数的执行记录；其次，根据沙箱截获系统函数执行记录的特点，设计了地址空间有限状态自动机，并在自动机内分析获取的执行记录来判别沙箱截获的系统函数；最后，遍历测试函数集来识别目标沙箱截获的系统函数集。该文设计实现了原型系统 SIAnalyzer，并对 Chromium 和 Adobe Reader 进行了沙箱拦截识别测试，测试结果验证了方法的有效性和实用性。

关键词：沙箱拦截；系统函数集；钩子；函数注入；自动机
中图分类号：TP311.1 **文献标识码**：A **文章编号**：1009-5896(2016)07-1823-08
DOI: 10.11999/JEIT151074

Sandbox-interception Recognition Method Based on Function Injection

ZHAO Xu YAN Xuexiong WANG Qingxian WEI Qiang
(*The PLA Information Engineering University*, *Zhengzhou* 450002, *China*)
(*State Key Laboratory of Mathematical Engineering and Advanced Computing*, *Zhengzhou* 450002, *China*)

Abstract: Testing sandbox authentication mechanism needs to recognize the sandbox interception first, *i.e.* to recognize the intercepted system function sets by the sandbox. Existing Hook recognition methods and tools mainly focus on the existence of the hook, lacking the ability of recognizing sandbox interception. This study proposes a sandbox interception recognition method based on function injection. The method recognizes the sandbox intercepts testing functions by analyzing the trace of system functions. First, the method injects and executes the system functions in untrusted process to record the function trace. Then, according to the features of intercepted system function trace, the paper designs the address space finite state automata and identifies intercepted system functions by analyzing the trace. Next, the function sets are traversed to identify the intercepted system function sets by target sandbox. Finally, a prototype is implemented—SIAnalyzer, and tested with Chromium Sandbox and Adobe Reader Sandbox. Results show the method proposed is effective and practical.

Key words: Sandbox interception; System functionset; Hook; Function injection; Automata

1 引言

沙箱是隔离不可信程序(恶意程序或存在漏洞程序)的重要安全机制，其中，服务机制和验证机制是沙箱的主要实现机制[1,2]。沙箱的服务机制向不可信程序提供服务接口来满足其功能需要，该机制存在缓冲区溢出等缺陷会导致沙箱逃逸漏洞的产生，如 CVE-2014-0512, CVE-2014-0546, CVE-2015-2429[3−5]；沙箱的验证机制截获不可信程序调用的系统函数并验证其行为来保证系统安全，该验证过程实现复杂且容易出现逻辑缺陷，也可能导致沙箱逃逸，比如，CVE-2011-1353, CVE-2013-0641, CVE-2013-3186[6−8]。沙箱安全分析技术受到产业界和学术界重视，研究者分析了不同沙箱的实现机制并使用模糊测试技术[9,10]测试并发现沙箱的缺陷，比如，文献[11−14]分别研究了 Protect Mode IE, Chrome 等沙箱的实现机制，文献[15,16]设计了 COMEye 等模糊测试工具测试沙箱的缺陷。已有的沙箱测试研究多集中在沙箱的服务机制方面，而针对沙箱验证机制的测试方法方面的研究不多。

收稿日期：2015-09-21；改回日期：2016-03-03；网络出版：2016-04-07
通信作者：赵旭 zhx0117@sina.cn
基金项目：国家 863 计划项目(2012AA012902)
Foundation Item: The National 863 Program of China (2012AA012902)

截获不可信程序调用的系统函数是沙箱验证机制的首要步骤，因此，沙箱验证机制的测试需要首先识别沙箱拦截，即沙箱截获的系统函数集。已有的沙箱拦截识别方法分析特定沙箱的实现细节来识别沙箱拦截，比如，文献[17]使用 IDA 分析并识别了 Adobe Reader X 的沙箱拦截。但由于不同沙箱截获的系统函数不同，并且不同沙箱的沙箱拦截实现技术也存在差异，因此，已有的沙箱拦截识别方法无法满足高效、自动的沙箱拦截识别的需求。Hook 技术是沙箱拦截的主要实现技术，已有的 Hook 识别方法关注钩子的植入机制，缺乏对钩取的系统函数的识别，无法满足系统函数识别的要求，比如，文献[18,19]使用程序切片分析等技术识别 Rootkit 在系统中植入的钩子。Hook 识别工具只能识别部分类型的钩子及其截获的系统函数，无法完全识别沙箱截获的系统函数集，比如，文献[20,21]设计的工具只能识别系统服务描述表(SSDT)、输入/输出地址表(IAT&IDT)中存在的钩子，HookShark[22]能够识别 Inline 和 VTable 两种类型的钩子，但无法识别 Inline 类钩子截获的系统函数。

通过对不可信进程(不可信程序的运行时进程)在沙箱内调用系统函数的流程的分析研究，本文发现：

(1)虽然沙箱使用不同的 Hook 技术截获不可信进程调用的系统函数，但都会将控制流重定向到沙箱的验证代码，而该过程会体现在该进程的指令执行记录(Trace)中；

(2)操作系统提供大量的系统函数，如果在沙箱内分别运行每个函数的测试程序来获取执行记录会耗费大量资源。而在沙箱内运行的不可信进程中注入并执行待识别的系统函数，则可以快速、准确地生成用于沙箱拦截识别的执行记录。

基于以上观察，本文设计了一种基于函数注入的沙箱拦截识别方法，方法包括两个子方法：(1)基于函数注入的执行记录生成方法，该方法选择不可信进程的 NOP，HLT 等指令填充的内存区域并注入待识别的函数，进一步控制不可信进程执行该函数来获取执行记录；(2)基于执行记录的沙箱截获系统函数分析方法，首先，方法建立描述指令地址空间转换的有限状态自动机模型，其次，以执行记录的指令作为自动机输入，通过自动机的状态和状态转换指令的位置来判断沙箱的系统函数截获行为及识别沙箱截获的系统函数。

论文的组织结构如下：第 2 节介绍基于函数注入的执行记录生成方法的工作原理；第 3 节介绍基于执行记录的沙箱截获函数的分析方法；第 4 节首先介绍了原型系统—SIAnalyzer 的结构及主要实现问题的解决方法，其次，介绍了方法性能的分析方法和实验结果；第 5 节总结了本文的主要工作并指出进一步研究方向。

2 基于函数注入的执行记录生成方法

基于函数注入的执行记录生成方法在不可信进程的内存中注入测试的系统函数，并控制该进程执行注入的函数来生成用于沙箱拦截识别的执行记录(图 1)。为了更好地描述基于函数注入的执行记录生成方法，下面给出 4 个定义：

定义 1 记地址空间表示不可信进程内存中来源不同的可执行代码所占用的内存区域。

为了便于沙箱拦截识别，本文将不可信进程的内存划分为：不可信进程地址空间(PC)、系统地址空间(OS)和沙箱地址空间(SC) 3 部分。

定义 2 记执行记录表示不可信进程在沙箱内调用系统函数所执行的指令。

根据上述定义，访问系统函数 A 的执行记录可以记为 $T(A)$，其中，$T(A)=\{i_1,i_2,\cdots,i_k,\cdots,i_{m-1},i_m\}$ 是执行的指令集合，$i_k=<$ no,op,saddr,taddr $>$ 表示 $T(A)$ 的第 k 个指令，包括指令在执行记录中的序号(no)，操作码(op)，指令所在地址(saddr)以及将要执行指令的地址(taddr)。

定义 3 记 ms 表示可用于函数注入的不可信进程的内存区域，ms 包括地址(addr)和长度(sz)两个属性。

Windows 系统内，由于 NOP, HLT 被大量用于填充程序的代码段来保证指令对齐，因此，为了不破坏不可信程序的功能代码，本文在 Windows 系统中选择 NOP, HLT 填充的区域为 ms 。

定义 4 记 s 表示测试的系统函数，包括函数参数信息(p)和所需内存大小(sz)两个属性。

$$\text{ms}_i.\text{sz} - \text{DJMP.sz} - \max(\text{Inst.sz}) \geq 0 \tag{1a}$$

$$\sum_{i=1}^{k} \text{ms}_i.\text{sz} - s.\text{sz} \geq 0\ ,\ \min(k) \tag{1b}$$

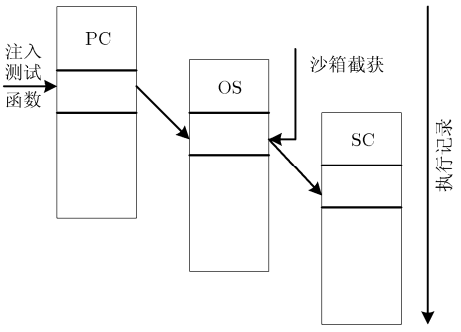


图 1 基于函数注入的执行记录生成示意图

基于函数注入的执行记录生成算法的主要步骤如表 1 所示，包括函数注入、函数执行和执行记录获取 3 个步骤。在函数注入步骤中，如果存在长度大于注入函数 s_i 所需的内存空间的 ms，直接在该 ms 注入 s_i，否则，算法选择多个 ms 注入 s_i。

本文选择多 ms 时基于以下两个原则：(1)减少不可信进程的指令对注入函数执行记录的影响，(2)保证注入函数的连贯执行及执行记录的生成效率。因此，本文使用式 (1a)和式(1b)来选择用于注入函数的多 ms，其中，式(1a)选择的 ms 能够容纳一个直接跳转指令和一个 s_i 的指令，式(1b)则选择尽可能少的 ms 注入测试的系统函数。

表 1 执行记录(trace)生成算法

算法：基于函数注入的执行记录生成算法
输入：$\mathrm{MS}=\{\mathrm{ms}_1,\mathrm{ms}_2,\cdots,\mathrm{ms}_n \mid n>0且n\in N\}$
　　测试函数集 $S=\{s_1,s_2,\cdots,s_m \mid m>0且m\in N\}$
输出：S 内元素的执行记录集合 T;
1　initialization: Untrust Process p, $T\leftarrow\phi$
2　while $i\le S.sz$:
3　select s_i
4　if exist $\mathrm{ms}_j.sz\ge s.sz$:
5　　inject s_i in ms_j ;
6　else use formula(1) to select multi-ms:
7　　inject s_i in multi-ms
8　execute s_i and get trace_{si}
9　$T\leftarrow \mathrm{trace}_{si}$
10　return T

以 32 位 Windows 系统中识别沙箱 S 是否截获 CreateFileW 为例，假设 S 的不可信进程不存在单个内存空间大于 CreateFileW 所需空间的内存，需要选择多个 ms 注入 CreateFileW。由于 32 位系统的直接跳转指令占用 5B 空间，并且该平台的最大指令长度为 14B，即 DJMP.sz=5，$\max(\mathrm{Inst.sz})=14$，因此，式(1a) 和式(1b)筛选的 ms 不小于 19B。

本文方法在注入函数时采用贪婪策略，即在 ms 中尽可能多地注入测试函数的相关指令，假设筛选的 ms 大小都是 19B，那么注入结果如图 2 所示，其

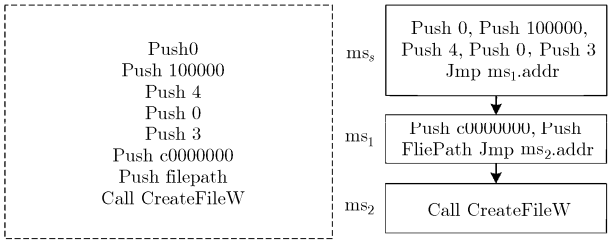


图 2 CreateFileW 注入示意图

中，ms_s，ms_1 和 ms_2 分别注入 CreateFileW 的 5 个，2 个和 1 个指令。

3　基于执行记录的沙箱截获函数分析方法

基于执行记录的沙箱截获函数分析方法将地址空间映射为有限状态自动机的不同状态，并将不同类型的指令作为自动机输入，通过在自动机内分析执行记录来识别沙箱截获的系统函数。该方法包括：(1)沙箱截获行为的识别，该识别过程分析系统函数执行记录的状态转换来判断是否存在截获系统函数的行为；(2)沙箱截获系统函数的识别，该识别过程根据实现地址空间状态转换指令的语义和位置信息来判定沙箱截获的系统函数。

定义 5　地址空间有限状态自动机 $M=<Q,q_0,\xi,q_0,\delta>$，Q 是地址空间状态集合，包括 q_0，q_1，q_2 3 个状态，其中，q_0，q_1，q_2 分别表示指令位于 PC, OS 和 SC; ξ 是 M 的输入表，由于 M 的状态转换是通过执行指令实现，因此，ξ 即为所在系统提供的所有指令；本文将 ξ 接受的指令分为 9 类，并设计了相应的地址空间转换语义，如表 2 所示；δ 是 M 的状态转换函数，且 $\delta\subset\{Q\times\xi\to Q\}$，如图 3 所示。

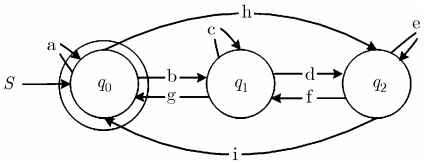


图 3 状态转换函数 δ 示意图

表 2 ξ 中元素的类型

指令类型	saddr	taddr	指令类型	saddr	taddr
a	PC	PC	f	SC	OS
b	PC	OS	g	OS	PC
c	OS	OS	h	PC	SC
d	OS	SC	i	SC	PC
e	SC	SC			

3.1 沙箱截获行为的识别

识别沙箱截获的系统函数需要首先判断是否存在沙箱截获行为，因此，本文分析了不可信进程调用系统函数时，执行记录地址空间自动机状态的转换情况。如图 4 所示，其中，图 4(a)表示函数未被沙箱截获，执行记录的指令只属于程序地址空间和系统地址空间；图 4(b)和图 4 (c)表示沙箱在系统地址空间实现系统函数截获的自动机状态转换情况；图 4(d)和图 4(e) 表示沙箱在不可信进程地址空间实现系统函数截获的自动机状态转换情况。

对比 M 状态转换的 5 种不同情况，本文发现 q_2 状态是判断是否存在沙箱截获行为的标志，因此，方法分析不可信进程调用特定系统函数的执行记录地址空间状态是否包含 q_2 状态来识别沙箱的截获行为。沙箱截获行为的识别包括两个步骤：(1)将执行记录转换为可用于 M 分析的中间表示；(2)在 M 内分析执行记录的中间表示并判断是否存在 q_2 状态。

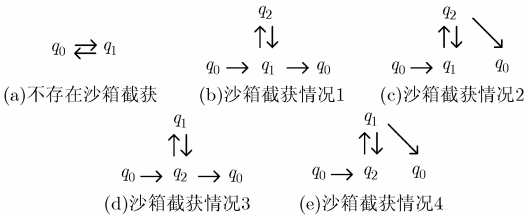


图 4 执行记录地址空间转换示意图

3.2 沙箱截获的系统函数识别

分析系统函数的执行记录地址空间状态是否包含 q_2 状态能够判断是否存在沙箱的截获行为，但仅通过 q_2 状态来判断沙箱截获的系统函数则会出现误报。以图 5 的执行记录为例，假设函数 A 是测试的系统函数，函数 B 是函数 A 调用的系统函数，图 5(a)的执行记录未执行沙箱的代码，即沙箱未截获系统函数，图 5(b)和图 5 (c)的执行记录都包含沙箱的指令，相应的地址空间状态都包含 q_2 状态，即存在沙箱截获，但图 5(b)截获的是 A 函数，图 5(c)截获的却是 B 函数。

$$\left.\begin{aligned} &m.\mathrm{no} - c_i.\mathrm{no} \geq 0, && 0 < i \leq k \\ &r_i.\mathrm{no} - m.\mathrm{no} \geq 0, && 0 < i \leq k \\ &m.\mathrm{no} - c_i.\mathrm{no} \leq \min(m.\mathrm{no} - c_j.\mathrm{no}), && 0 < i, j \leq k \end{aligned}\right\} \quad (2)$$

d，h 类指令改变不可信进程调用系统函数的控制流并执行沙箱的指令，即存在 $q_0 \xrightarrow{\mathrm{h}} q_2$ 或 $q_1 \xrightarrow{\mathrm{d}} q_2$ 的地址空间状态转移，而 d，h 类指令的位置就是沙箱截获行为的实现位置，其对应的系统函数即为沙箱截获的系统函数，因此，本文通过分析 d，h 类指令的地址关系来识别沙箱截获的系统函数。

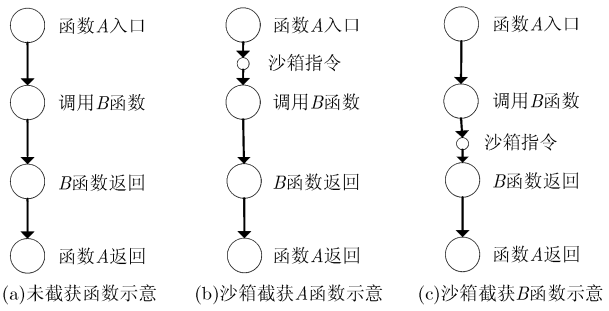


图 5 沙箱截获函数识别误报成因示意图

沙箱截获系统函数的识别包括两个步骤：(1)识别系统函数的执行记录中所有调用指令(c)和返回指令(r)，并根据 c.saddr = r.taddr 关系将其组织成 < c,r > ; (2)选择距离 d，h 类指令最近的 < c,r > 对应的系统函数为识别的沙箱截获的系统函数。本文方法使用式(2)选择包含 d，h 类指令的最小 < c,r >，其中，式(2)的前两个子式保证 < c,r > 包含 d，h 类指令，式(2)的第 3 个子式选择距离 d，h 指令最近的 < c,r > 对应的系统函数为沙箱截获的系统函数。

4 原型系统和实验验证

为了测试方法的有效性和性能，本文在全系统模拟器 QEMU[23]上构建了原型系统 SIAnalyzer。首先，本文选择开源的 Chromium 沙箱作为测试对象，验证 SIAnalyzer 的沙箱拦截识别能力，即验证方法的有效性；其次，本文记录了 SIAnalyzer 生成执行记录的数量、时间，并分析了 SIAnalyzer 的效率；最后，本文选择 Adobe Reader X(10.0.1)来测试 SIAnalyzer 对闭源沙箱的沙箱拦截识别能力，即验证本文方法的实用性。

4.1 SIAnalyzer 的设计与实现

SIAnalyzer 包括内存监控器、函数注入器、执行记录记录器和拦截分析器 4 个主要部分(图 6)。内存监控器监控和记录不可信进程的内存信息，同时，搜索不可信进程的代码空间并标记函数可注入的内存地址；函数注入器从测试函数库中选择用于测试的系统函数，进一步在内存监控器提供的可注入地址注入测试的系统函数；执行记录记录器获取系统函数的调用执行记录，将执行记录转换为便于沙箱拦截识别的中间表示并存入执行记录库；拦截分析器获取执行记录库中的执行记录中间表示，并在地址空间的有限状态自动机内分析识别沙箱截获的系统函数。

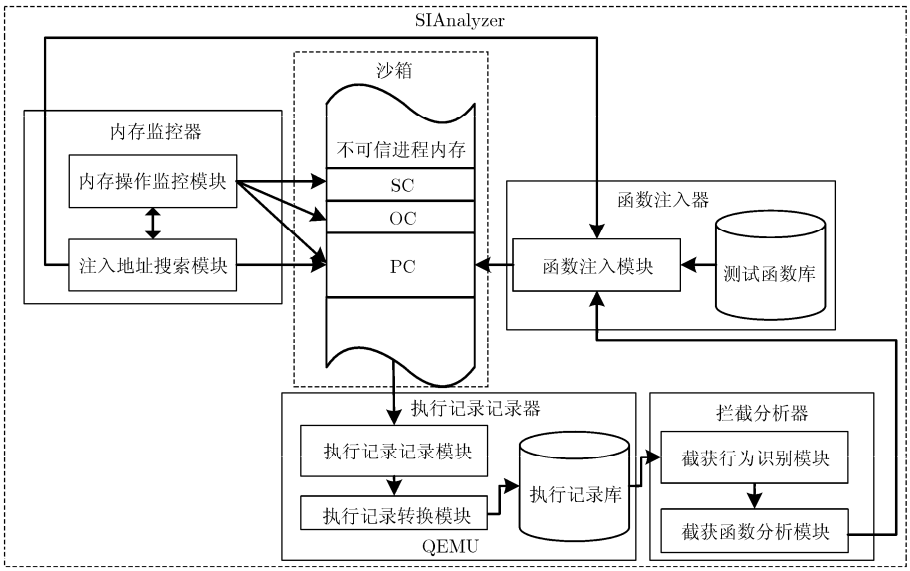


图 6　SIAnalyzer 结构示意图

本文在实现 SIAnalyzer 时，处理了测试系统函数集构造、函数注入时机选择等 4 个关键问题：

(1)测试系统函数集的构造：　测试系统函数集由基础函数集和增量函数集两部分组成，其中，基础函数集包括操作系统提供的进程、文件、注册表、网络以及安全相关的系统函数；增量函数集包括沙箱功能对应的系统函数。

(2)函数注入时机选择：　注入测试系统函数必须在沙箱拦截部署完成后，才能获取适用于分析的执行记录，因此，SIAnalyzer 需要识别沙箱拦截部署是否完成，而该识别过程需要监控沙箱行为，耗费资源巨大并且不同沙箱的拦截部署存在差异，存在大量误报。本文分析了不可信进程的初始化和执行步骤，发现一旦不可信进程执行，则说明该沙箱已经结束拦截部署，因此，SIAnalyzer 从不可信进程的角度选择函数注入时机，首先，SIAnalyzer 标记不可信进程的入口函数，其次，监视该入口函数的指令是否执行，如果该函数的指令被执行，则说明沙箱拦截部署完成，此时，SIAnalyzer 控制不可信进程并注入测试的系统函数。

(3)内存监控：为了能够准确标记各类内存空间和搜索可用于测试函数注入的内存地址，内存监控器监控系统提供的内存申请、释放函数，并通过这些函数的参数标记各类内存空间，比如，memcpy, free, MapViewOfFile 和 UnmapViewOfFile 等函数。

(4)执行记录转换规则：为了提高沙箱拦截识别的效率，SIAnalyzer 在执行记录获取的基础上，将执行记录的指令转换为便于拦截分析的中间表示。指令的中间表示转换规则将系统指令划分为转移指令(TI)和非转移指令(NTI)两类，系统指令的具体转换规则如表 3 所示。

4.2 实验验证

本文将 SIAnalyzer 部署在 3.4 GHz i3 处理器和 4 GB RAM 的 Linux 机器上，并选择 32 位的 Windows XP Sp3 系统作为测试系统，即在本文的实验验证过程中，式(1a)和式(1b)的 DJMP.sz, max(Inst.sz) 取值分别为 5 和 14。

表 3 指令转换规则

指令类型	跳转类型	指令转换规则
TI	Unconditional jump	saddr op taddr -> <no op saddr taddr>
	Conditional jump with branch hints	saddr op taddr -> <no op saddr saddr+sz(ins)>
		saddr op taddr-> <no op saddr taddr>
	Conditional jumps without branch hints	saddr op Label-> <no op saddr [Label]>
	Return	saddr op-> <no,op,saddr,esp>
NTI		saddr op taddr-> <No,op saddr,saddr+sz(ins)>

4.2.1 方法的有效性分析 Chromium 沙箱是主流的开源沙箱产品，Chromium 沙箱默认截获的系统函数共 20 个，主要包括操作文件、命名管道等 7 类函数。本文选择 Chromium 沙箱作为测试沙箱，同时，选择 HookShark 来对比分析本文方法的性能，即沙箱拦截识别能力。

测试的主要步骤如下：(1)修改 Chromium 的 AddRule 函数的参数来控制沙箱截获不同的系统函数；(2)使用 SIAnalyzer 和 HookShark 分别识别沙箱截获的系统函数，实验结果如表 4 所示。

表 4 SIAnalyzer 与 HookShark 的沙箱拦截识别能力对比

序号	函数名称	操作资源	Hook 方式	SIAnalyzer	HookShark
1	NtCreateFile	文件	代码 Hook	Y	N
2	NtOpenFile			Y	N
3	NtQueryAttributesFile			Y	N
4	NtQueryFullAttributesFile			Y	N
5	NtSetInformationFile			Y	N
6	CreateNamedPipeW	命名管道	数据 Hook	Y	Y
7	DuplicateHandle	句柄	数据 Hook	Y	Y
8	NtOpenThread	进、线程	代码 Hook	Y	N
9	NtOpenProcess			Y	N
10	NtOpenProcessToken			Y	N
11	NtOpenProcessTokenEx			Y	N
12	CreateProcessW		数据 Hook	Y	Y
13	CreateProcessA			Y	Y
14	NtCreateKey	注册表	代码 Hook	Y	N
15	NtOpenKey			Y	N
16	NtOpenKeyEx			Y	N
17	CreateEventW	事件	数据 Hook	Y	Y
18	OpenEventW			Y	Y
19	NtMapViewOfSection	内存	代码 Hook	Y	N
20	NtUnmapViewOfSection			Y	N

实验结果显示，SIAnalyzer 能够识别 Chromium 拦截的所有函数，而 HookShark 无法识别代码 Hook 方式截获的函数。这是因为 Chromium 的代码 Hook 修改函数指令来截获系统函数，HookShark 可以识别函数被修改的指令，但无法判定沙箱截获的系统函数，而 SIAnalyzer 以执行记录作为拦截分析的载体，不论沙箱采用何种 Hook 技术，执行的指令都会记录在执行记录中，因此，SIAnalyzer 的拦截识别能力不受沙箱使用的具体 Hook 技术影响。

4.2.2 方法的效率分析 执行记录是拦截分析的基础，因此，本文方法的效率分析主要关注生成执行记录的规模，生成执行记录的时间以及执行记录的利用效率。不失一般性，本文选择 Windows Xp Sp3 的两个核心库 NTDLL 和 KERNEL32 的导出函数作为测试函数集，其中，NTDLL 和 KERNEL32 分别导出 1317 个和 955 个系统函数。

方法效率分析的步骤如下：(1)修改 Chromium 沙箱源码，在 Chromium 沙箱内分别增加实现测试函数截获相关的 dispatcher.cc, interception.cc 和 policy.cc；(2)在 Chromium 默认拦截的基础上，使用 AddRule 函数增加 Chromium 沙箱截获的系统函数，并重新编译 Chromium 沙箱源码；(3)使用 SIAnalyzer 测试重新编译的 Chromium 沙箱，并在 Chromium 沙箱截获系统函数数量不同的情况下，统计 SIAnalyzer 生成执行记录的数量、规模和时间。

本文方法效率分析的结果如图 7 所示，其中，图 7(a)是执行记录的规模、时间与测试函数数量之间的关系图，实验结果显示，方法生成执行记录的规模和耗费的时间都随沙箱截获系统函数数量的增加而增加，但由于存在系统函数间的相互调用，随着沙箱截获函数的增加，方法能够从单条执行记录

中识别多个不同的被截获系统函数，因此，随着沙箱截获函数的增加，本文方法生成执行记录的规模和时间开销的增长速度变缓。

进一步，本文分析了方法生成执行记录的数量与沙箱截获函数数量的关系。本文定义了沙箱拦截的识别率来分析执行记录数量与沙箱截获函数数量之间的关系：

$$\text{IRatio}=\frac{\text{FuncNum}}{\text{TraceNum}}\times 100\%$$

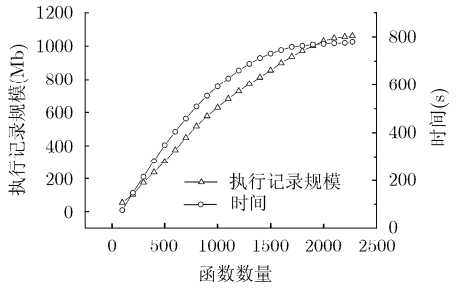
其中，FuncNum 表示沙箱截获的系统函数数量，TraceNum 表示用于识别沙箱截获函数的执行记录数量。

图 7(b)是执行记录数量，IRatio 与函数数量之间的关系图，分析结果显示，执行记录数量随着沙箱截获系统函数数量的增加而增加；而 IRatio 却是先增长后减小，这是由于在沙箱截获的系统函数数量属于 (0,1200] 时，单条执行记录能够识别的系统函数数量随之增加，即 IRatio 增长，当沙箱截获的系统函数数量属于 (1200,2272] 时，部分截获的函数已被 SIAnalyzer 在前期实验中识别，即 IRatio 下降。

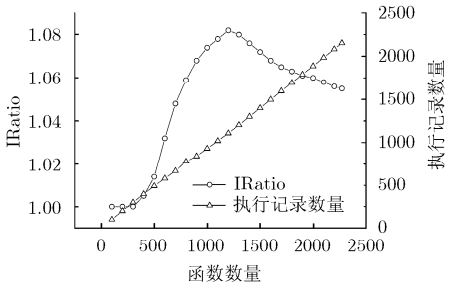
4.2.3Adobe Reader 沙箱的拦截分析　本文选择文献[17]测试的版本号为 10.0.1 的 Adobe Reader X 闭源沙箱来验证方法的实用性。Adobe Reader X 的 pdf 解析进程是该沙箱的不可信进程，因此，SIAnalyzer 选择该进程注入测试函数。实验结果显示，SIAnalyzer 具有与文献[17]方法相同的沙箱拦截识别能力，即能够识别该版本 Adobe Reader X 沙箱截获的系统函数，共 193 个函数，如表 5 所示；同时，SIAnalyzer 自动化程度更高，且不需要研究人员具有沙箱的先验知识。

表 5　SIAnalyzer 识别 Adobe Reader 沙箱拦截统计表

序号	函数库	截获函数数量
1	GDI32	53
2	CRYPT32	40
3	WINNET	31
4	USER32	22
5	NTDLL	21
6	WINSPOOL	15
7	KERNEL32	4
8	MPR	3
9	OLE32	2
10	KERNELBASE	2



(a)执行记录规模、时间与函数数量之间关系图



(b)执行记录数量、IRatio与函数数量之间关系图

图 7　SIAnalyzer 效率分析图

5　结束语

本文设计了一种基于函数注入的沙箱拦截识别方法，该方法首先在不可信进程中注入并执行系统函数来获取用于分析的执行记录，其次，引入地址空间的有限状态自动机，通过自动机的状态转换来识别沙箱的截获行为，并分析状态转换指令的信息来识别沙箱拦截的系统函数；再次，设计实现了原型系统 SIAnalyzer；最后，以 Chromium 沙箱和 Adobe Reader X 沙箱作为测试沙箱，测试了本文方法的有效性和实用性，同时，分析了本文方法的效率，实验结果显示，与已有的沙箱拦截识别方法相比，本文方法具有相同沙箱拦截识别能力的同时，自动化程度和执行效率更高。

在沙箱拦截识别的基础上，研究沙箱验证机制的测试技术并发现沙箱验证机制存在的缺陷是进一步研究的主要方向。

参 考 文 献

[1] YEE B, SEHR D, DARDYK G, *et al*. Native client: A sandbox for portable, untrusted x86 native code[C]. 2009 IEEE Symposium on Security and Privacy, Oakland, USA, 2009: 79–93.

[2] MAASS M, SALES A, CHUNG B, *et al*. A systematic analysis of the science of sandboxing[J]. *PeerJ Computer Science*, 2016, 2: e43. doi:10.7717/peerj-cs.43.

[3] CVE-2014-0512[OL]. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0512, 2014.

[4] CVE-2014-0546[OL]. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0546, 2014.

[5] CVE-2015-2429[OL]. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-2429, 2015.

[6] CVE-2011-1353[OL], https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-1353, 2011.

[7] CVE-2013-0641[OL]. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0641, 2013.

[8] CVE-2013-3186[OL]. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-3186, 2013.

[9] 崔宝江, 梁晓兵, 王禹, 等. 基于回溯和引导的关键代码区域覆盖的二进制程序测试技术研究[J].电子与信息学报, 2012, 34(1): 108–114. doi: 10.3724/SP.J.1146.2011.00532.
CUI B J, LIANG X B, WANG Y, *et al*. The study of binary program test techniques based on backtracking and leading for covering key code area[J]. *Journal of Electronics & Information Technology*, 2012, 34(1): 108–114. doi: 10.3724/SP.J.1146.2011.00532.

[10] 欧阳永基, 魏强, 王清贤, 等. 基于异常分布导向的智能 Fuzzing 方法[J].电子与信息学报, 2015, 37(1): 143–149. doi: 10.11999/JEIT140262.
OUYANG Y J, WEI Q, WANG Q X, *et al*. Intelligent fuzzing based on exception distribution steering[J]. *Journal of Electronics & Information Technology*, 2015, 37(1): 143–149. doi: 10.11999/JEIT140262.

[11] SABABAL P and MARK V Y. Playing in the reader X sandbox[C]. Black Hat USA 2011, Las Vegas, USA 2011. https://media.blackhat.com/bh-us-11/Sabanal/BH_US_11_SabanalYason_Readerx_WP.pdf.

[12] MARK V Y. Understanding the attack surface and attack resilience of project spartans new edgeHtml rendering engine[C]. Black Hat USA 2015, Las Vegas, USA, 2015. https://www. blackhat. com/ docs/ us-15/materials/us-15-Yason-Understanding-The-Attack-Surface-And-Attack-Resilience-Of-Project-Spartans-New-EdgeHTML-Rendering-Engine-wp.pdf.

[13] JAMES F. Digging for sandbox escapes-finding sandbox breakouts in Internet explorer[C]. Black Hat USA 2014, Las Vegas, USA, 2014. https://www.blackhat.com/docs/ us-14/materials/us-14-Forshaw-Digging-For_IE11-Sandbox-Escapes.pdf.

[14] KOH Y C. Understanding the microsoft office 2013 protected-view sandbox[C]. Recon, Montreal, Canada, 2015. http://recon.cx/2015/slides/recon 2015-16-yong-chuan-koh-Understaning-the-Microsoft-Office-Protected-View-Sandbox.pdf.

[15] LI X N and LI H F. Smart COM fuzzing-auditing IE sandbox bypass in COM objects[C]. CanSecWest Vancouver 2015, Vancouver, Canada, 2015. https://cansecwest.com/ slides/2015/Smart_COM_Fuzzing_Auditing_IE_Sandbox_Bypass_in_COM_Objects-Xiaoning_li.pdf.

[16] BRIAN G and JASIEL S. Thinking outside the sandbox: Violating trust boundaries in uncommon ways[C]. Black Hat USA 2014, Las Vegas, USA, 2014. https: //www. blackhat. com/docs/us-14/materials/us-14-Gorenc-Thinking-Outside-The-Sandbox-Violating-Trust-Boundaries-In-Uncommon-Ways-WP.pdf.

[17] LIU Z H and GUILAUME L. Breeding Sandworms: How to fuzz your way out of Adobe Reader's Sandbox[C]. Black Hat EUROPE 2012, Amsterdam, Netherlands, 2012. https://media.blackhat.com/bh-eu-12/Liu_Lovet/bh-eu-12-Liu_Lovet-Sandworms-Slides.pdf.

[18] Wang Z, JIANG X, CUI W, *et al*. Countering persistent kernel rootkits through systematic hook discovery[C]. Recent Advances in Intrusion Detection 2008, Cambridge, England, 2008: 21–38.

[19] YIN H, POOSANKAM P, HANNA S, *et al*. HookScout: proactive binary-centric hook detection[C]. 7th Detection of Intrusions and Malware, and Vulnerability Assessment, Bonn, Germany, 2010: 1–20.

[20] BUTLER J and GREG H. VICE–catch the hookers[C]. Black Hat USA 2004, Las Vegas, USA, 2004. http:// 120.52.72.44/www.blackhat.com/c3pr90ntcsf0/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf.

[21] JOANNA R. System Virginity verifier-defining the roadmap for malware detection on windows system[C]. Hack In The Box 2005, Kuala, 2005. http://www.cs.dartmouth.edu/~sergey/cs258/rootkits/hitb05_virginity_verifier.ppt.

[22] HookShark[OL]. http://www.gamedeception.net/threads/20596-HookShark-Beta-0-9-highlight=hookshark, 2010.

[23] BELLARD F. QEMU, a fast and portable dynamic translator[C]. Proc. USENIX Annual Technical Conference, Marroitt Anaheim, USA, 2005: 41–46.

赵 旭： 男，1986 年生，博士生，研究方向为二进制程序漏洞挖掘、Web 安全.

颜学雄： 男，1975 年生，副教授，研究方向为二进制程序分析、Web 安全.

王清贤： 男，1960 年生，教授，研究方向为网络空间安全.

魏 强： 男，1979 年生，副教授，研究方向为程序分析、漏洞挖掘.