面向资源泄漏的浏览器沙箱测试方法

赵 旭[1,2],颜学雄[1],王清贤[1],魏 强[1],李继中[3]
(1.解放军信息工程大学网络空间安全学院,河南郑州 450001;
2.63892部队,河南洛阳 471003;3.郑州战略投送基地,河南郑州 450002)

摘 要: 资源泄漏是导致浏览器沙箱逃逸的重要缺陷之一,已有浏览器沙箱测试方法不完全适用于发现资源泄漏缺陷.基于大多数导致沙箱逃逸的资源具有相同或相似属性取值的分析,本文提出了一种面向资源泄漏的浏览器沙箱测试方法.该方法首先分析并约简敏感资源的属性来生成资源筛选规则;其次,定义资源与资源筛选规则前件的最大加权语义相似度为逃逸指数,并使用逃逸指数阈值来筛选测试资源;再次,设计并实现了原型系统BSTS(Browser Sandbox Testing System),并在BSTS内分析了方法的性能.进一步,选择多个主流浏览器沙箱来测试本文方法的资源泄漏发现能力,实验结果显示本文方法具有良好的资源泄漏发现能力.

关键词: 浏览器沙箱; 资源泄漏; 粗糙集理论; 语义匹配
中图分类号: TP311.1 **文献标识码:** A **文章编号:** 0372-2112 (2017)07-1775-09
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.07.031

A Resource-Leakage Oriented Browser Sandbox Testing Method

ZHAO Xu[1,2], YAN Xue-xiong[1], WANG Qing-xian[1], WEI Qiang[1], LI Ji-zhong[3]
(1. *College of Cyberspace Security, PLA Information Engineering University, Zhengzhou, Henan* 450001, *China*;
2. 63892 *Unit, Luoyang, Henan* 471003, *China*; 3. *Strategic Delivery Base, Zhengzhou, Henan* 450002, *China*)

Abstract: Resource leakage is one of the important defects of sandbox escape. The existing browser sandbox testing methods are insufficient to discover leak resources. Based on the analysis that most leaking resources have same or similar attribute values, this paper designed a resource-leakage oriented browser testing method. The method firstly analyzes resources attributes and creates resource selecting rules; secondly, calculates the escape index of every resource of system and uses threshold to select testing resources; thirdly, designs and implements a prototype system-Browser Sandbox Testing System (BSTS) and analysis the capability of our method; then we select and test some browser sandboxes; in the end, we found an undisclosed resource leakage vulnerability.

Key words: browser sandbox; resource leakage; rough set theory; semantic match

1 引言

沙箱是一种基于隔离的安全机制,隔离可信/不可信软件模块来提高软件的安全性[1].大量厂商将沙箱植入浏览器来提高浏览器安全,比如,Chrome、Protect Mode IE(PMIE)、Firefox以及360安全浏览器、猎豹等国内的主流浏览器.引入沙箱的浏览器由可信的代理程序(代理进程)和不可信的渲染程序(渲染进程)组成,代理进程具有用户权限,接受渲染进程的资源访问请求,并根据安全策略满足/拒绝渲染进程的资源访问请求.

沙箱提高了攻击浏览器的门槛,但其自身也存在安全缺陷,如果攻击者利用沙箱缺陷实现逃逸,将会造成更大危害.操作系统包含文件、程序、图片等大量不同类型的资源,浏览器沙箱只允许渲染进程访问部分资源,一旦渲染进程能够访问敏感资源(禁止访问的资源),攻击者就可以进一步危害用户的信息安全.资源泄漏是沙箱特有的一类安全缺陷,本文将浏览器沙箱的资源泄漏分为三类:

(1)数据泄漏.渲染进程可以读取包含敏感信息的资源,导致用户信息泄漏,比如,Cookie文件、数据库文件等.

收稿日期:2016-01-27;修回日期:2016-06-12;责任编辑:孙瑶
基金项目:国家863高技术研究发展计划(No.2012AA012902)

(2)信息篡改. 渲染进程能够向高权限资源写入数据,影响高权限资源的行为,比如,CVE-2011-1353[2]、占桩攻击[3].

(3)任意代码执行. 渲染进程能够控制高权限资源执行,导致执行恶意行为,比如,CVE-2013-3186[4]、CVE-2013-4015[5].

研究者开展了浏览器沙箱实现机制分析和测试的大量安全研究. 比如,Tom Keetch 分析了 Protect Mode IE(PM IE)的实现机制和攻击面[6];Yason 分析了 Enhanced Protect Mode IE(EPM IE)的实现机制,并将 EPM IE 的攻击面总结为策略缺陷、编码缺陷等四类[7]. 在两者研究基础上,Forshaw 针对 EPM IE 的策略缺陷,设计了一种基于符号链接的沙箱逃逸技术[8-9];Li 则针对 EPM IE 的 COM 组件,设计并实现了模糊测试工具 COMEye[10]. 沙箱的进程间通信机制也被研究者广泛关注. 比如,Liu 手工分析了沙箱的拦截机制,并设计了针对代理程序的模糊测试方法[11];Vreugdenhil 进一步分析了进程间通信的 Cross Call,并提出了面向 Cross Call 的模糊测试方法. 浏览器沙箱的模糊测试技术可以测试发现导致浏览器沙箱崩溃的缺陷,而大多数泄漏的资源不会导致浏览器沙箱崩溃. 同时,模糊测试浏览器沙箱泄漏的资源,相当于在浏览器内随机访问系统的任一资源,时空消耗巨大,且效率低下[12-15].

我们观察发现导致沙箱逃逸的资源具有相似性,比如,msdt. exe、mstsc. exe 等造成 PM IE 逃逸的资源是中等完整性、可执行的二进制文件等. 如果能够使用已知造成沙箱逃逸的资源的共同特征来筛选用于测试的系统资源;既可以减少测试资源的数量,提高测试效率,同时相比于随机选择,筛选的资源与已知造成沙箱逃逸的资源具有相似性,又可以提高发现浏览器沙箱的资源泄漏能力.

本文提出了一种面向资源泄漏的浏览器沙箱测试方法. 该方法首先使用属性来描述系统资源,在此基础上,分析已知造成沙箱逃逸资源(敏感资源)的属性向量,生成反映其共性的资源筛选规则,并根据资源筛选规则中不同属性对沙箱逃逸影响力的不同,为每个资源属性赋予不同权重. 其次,计算系统资源的属性向量与每个资源筛选规则前件的相似度,将其中的最大值作为资源的逃逸指数,进一步通过设定的逃逸指数阈值来筛选用于测试的系统资源. 最后,在测试资源选择的基础上,控制浏览器的渲染进程访问筛选的测试资源,并根据访问结果来判断浏览器沙箱是否存在资源泄漏.

2 测试方法概述

为了提高属性向量对系统资源的描述能力,同时,在测试资源筛选过程中,提高资源和规则的匹配准确度,本文使用数值和操作系统本体(OSO[16]、NIEO[17])的实例为资源属性赋值,比如,msdt. exe 对应的属性向量为 r_{msdt} = <binary,'c:\Windows\System32',1.02MB,'rx'> 其中,类型、路径以及访问方式都采用本体实例描述,而占用空间则由数值和单位描述. 在此基础上,本文方法包括三个主要步骤(如图 1 所示).

(1)分析敏感资源的属性向量,生成反映敏感资源共性的测试资源筛选规则

在分析敏感资源属性向量的过程中,不是所有的资源属性都会影响沙箱逃逸. 因此,首先使用粗糙集理论[18]的决策信息系统来描述已知敏感资源;其次,使用区分矩阵来约简与资源泄漏类型无关的资源属性,并生成测试资源筛选规则;再次,根据测试资源筛选规则中每个资源属性的重要性不同,为资源筛选规则中的每个属性赋予不同权重.

(2)匹配系统资源的属性向量和测试资源筛选规则的前件来筛选用于测试的系统资源

通过匹配系统资源的属性向量与规则前件来选择用于测试的资源,但由于生成规则的敏感资源数量有限,规则无法覆盖不同资源属性的所有取值,即存在规则和资源属性向量匹配失败的情况. 因此,使用语义相似度的计算方法来计算资源的属性向量和集中规则前件的相似度,并将语义相似度的最大值作为该系统资源的逃逸指数,进一步通过设定的逃逸指数阈值来筛选测试资源.

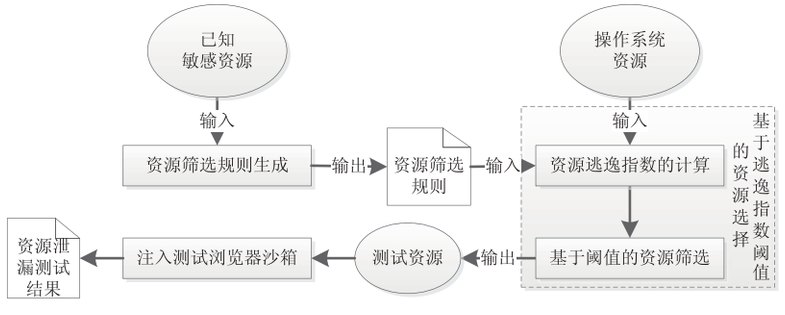


图1 面向资源泄漏的浏览器沙箱测试方法

(3)测试并判断浏览器沙箱是否存在资源泄漏

使用浏览器的渲染进程访问筛选的测试资源,并根据渲染进程的资源访问结果来判断测试的浏览器沙箱是否存在资源泄漏缺陷.

3　基于粗糙集理论的资源筛选规则生成方法

一方面,资源不同属性与资源泄漏的关联程度不同,即重要性不同;另一方面,任何一种基于属性的资源描述机制都包含大量属性,大量属性会增加筛选测试资源的计算负载. 因此,本节使用粗糙集理论的属性约简方法来约简与资源泄漏无关的属性,并生成资源筛选规则;使用属性重要性为资源筛选规则中的每个属性赋予权重来提高进一步资源筛选的准确度. 下面介绍本文使用的粗糙集理论相关概念及资源筛选规则生成算法.

3.1　基于粗糙集理论的资源筛选规则生成算法

为了使用粗糙集理论来生成资源筛选规则,本节首先介绍粗糙集理论的相关概念及资源筛选规则生成算法的主要思路.

定义 1　知识表达系统. 四元组 $I=(U,A,V,f)$ 是一个知识表达系统,其中,U 表示对象的非空有限集合,称为论域;A 表示属性的非空有限集合;$V=\bigcup_{a\in A}V_a$,V_a 是属性 a 的值域;$f:U\times A\to V$ 是一个信息函数,它为对象的属性进行赋值,即 $\forall a\in A,x\in U,f(x,a)\in V_a$.

本文使用知识表达系统来描述敏感资源,即 U 表示已知的敏感资源集合,A 表示非空的资源属性集合,V 表示资源属性的值域,f 表示资源属性到值域的映射,即敏感资源、资源属性及其取值和资源属性取值映射可以组成知识表达系统 I. 进一步,本文将资源泄漏类型 $D(A\cap D=\varnothing)$ 引入 I,得到敏感资源的决策信息系统 $I'=(U,A,D,V,f)$.

在决策信息系统中,决策属性对条件属性的依赖程度不同,有些条件属性甚至是冗余的;相应的,敏感资源的逃逸类型对不同的资源属性的依赖程度也存在差异. 粗糙集理论的一项重要研究内容就是找出并去掉冗余的条件属性. 因此,在 I' 内,本文使用粗糙集的属性约简和规则生成能力来计算获取反映敏感资源属性特征的资源筛选规则,下面分别介绍属性约简和资源筛选规则相关概念.

定义 2　正区域. 设 P,Q 是论域 U 上的等价关系,则集合 $\mathrm{POS}(Q)=\bigcup_{X\in U/Q}\underline{P}X$ 称为正区域,其中,集合 $\underline{P}X=\{x\mid[x]_P\subseteq X\}$ 称为 X 的下近似集.

定义 3　约简. 在 I' 内,设有 $P\subseteq A,P\neq\varnothing$,若 P 满足条件 $\mathrm{POS}_P(D)=\mathrm{POS}_A(D)$,则称 P 是 A 的一个约简. 记 $\mathrm{RED}(A)$ 为 A 的所有约简.

区分矩阵是计算获取决策信息系统约简属性的主要方法,本文使用区分矩阵来获取 I' 的约简属性集合. 在得到 I' 的资源属性约简结果之后可以构造规则. 将约简结果作为规则的前件,同时将对应的决策属性作为规则的后件,获得的规则就反映了敏感资源在哪些属性取值下会导致何种类型的资源泄漏,本文将获取的规则称为资源筛选规则.

定义 4　区分矩阵. 在 I' 内,设 $|U|=n$,那么,I' 的区分矩阵(Discernibility Matrix,DM)是一个 $n\times n$ 矩阵,其任一元素为

$$\mathbf{DM}[i][j]=\begin{cases}\{a\in C\mid f(u_i,a)\neq f(u_j,a)\}, & D(u_i)\neq D(u_j)\\ 0, & D(u_i)=D(u_j)\end{cases}$$
$$i,j=1,2,\cdots,n$$

定义 5　资源筛选规则. 记 $\mathrm{ru}:\mathrm{des}(U/(\mathrm{RED}(C)))\to\mathrm{des}(U/D)$ 表示资源筛选规则,其中,$\mathrm{des}(U/(\mathrm{RED}(A)))$ 表示等价类 $U/(\mathrm{RED}(A))$ 对资源各属性的特定取值,$\mathrm{des}(U/D)$ 表示等价类 U/D 对于各逃逸属性的特定取值,且 $(U/(\mathrm{RED}(C)))\cap(U/D)\neq\varnothing$.

为了能够精确地筛选用于测试的系统资源,本文引入粗糙集理论的属性重要性来为规则前件的不同资源属性赋予权重.

定义 6　资源属性的权重. 记 w_i 表示规则前件中属性 a_i 的权重,则 $w_i=\gamma_{\mathrm{RED}(C)}(D)-\gamma_{\mathrm{RED}(C)-\{a_i\}}(D)$,其中,$\gamma_{\mathrm{RED}(C)}(D)=|pos_{\mathrm{RED}(C)}(D)|/|U|$ 表示属性依赖度,且 $0\leqslant\gamma_{\mathrm{RED}(C)}(D)\leqslant 1$.

在上述定义基础上,本节设计了基于粗糙集理论的资源筛选规则生成算法. 算法首先使用区分矩阵约简计算 I' 内敏感资源的资源属性,并将约简的属性和资源泄漏类型组织得到资源筛选规则;进一步,使用定义 6 来获取规则前件中每个资源属性的权重,细节如算法 1 所示.

算法 1　基于粗糙集理论的资源筛选规则生成算法

输入:敏感资源的决策信息系统 I';系统资源描述本体 OSO;
输出:资源筛选规则集合 RU;属性权重集合 W;
(1)　初始化:RU = ∅,RE = ∅,CORE = ∅,$i=j=0$,区分矩阵 **DM** = {0};
(2)　While($i<|U|$){
(3)　　$j=i+1$;
(4)　　While($j<|U|$){
(5)　　if($D[i]==D[j]$): **DM**$[i][j]=0$;
(6)　　else if (($a(u_i)\neq a(u_j)$) ∨ (class($a(u_i)$) ≠ class($a(u_j)$))):
(7)　　$a\to$**DM**$[i][j]$;
(8)　　if ||**DM**$[i][j]$| == 1|:**DM**$[i][j]\to$CORE; }
(9)　$i++$; }
(10)　使用定义 3 计算 DM 的约简;

(11) 生成规则 ru_i,并将 ru_i 置入 RU;
(12) $\forall a \in RE$,使用定义 6 计算每个资源属性的权重,并将 w 置入 W;
(13) Return RU,W

算法在获取区分矩阵过程中两次遍历敏感资源集合,因此,算法的时间复杂度与敏感资源数量$|U|$和资源属性的个数$|A|$正相关,即时间复杂度 $O(|A||U|^2)$.

3.2 资源筛选规则生成示例

本节将以一个实例来说明算法 1 的原理. 已知 msdt. exe、mstsc. exe等十个敏感资源,并用路径(a_1)、访问方式(a_2)、属主(a_3)、规模(a_4)和类型(a_5)五个属性描述资源. 为了讨论方便,本文对路径、规模属性进行了处理,其中,路径属性取值包括系统程序目录(SPD)、系统数据目录(SDD)、用户程序目录(UPD)和用户数据目录(UDD)四类,规模属性取值包括大(B)、中(M)、小(S)三个级别,那么,这十个资源如表 1 所示.

表 1 已知逃逸资源表 S

序号	路径(a_1)	访问方式(a_2)	属主(a_3)	规模(a_4)	类型(a_5)	资源泄漏类型(d)
1	SPD	W	System	M	Document	信息篡改(IM)
2	SDD	X	User	S	Document	代码执行(CX)
3	UPD	W	User	S	Program	信息篡改(IM)
4	SDD	W	System	M	Document	代码执行(CX)
5	SDD	W	Guest	B	Data	信息篡改(IM)
6	UPD	X	User	S	Program	代码执行(CX)
7	SDD	R	User	S	Data	数据泄漏(DL)
8	UDD	R	User	B	Document	数据泄漏(DL)
9	SDD	W	Guest	B	Program	代码执行(CX)
10	SDD	R	System	M	Data	数据泄漏(DL)

算法 1 首先计算 S 的区分矩阵,并得到该区分矩阵的待约简布尔表达式:

$$\begin{aligned}\Delta &= \prod_{(u_i,u_j)\in U\times U}\sum a^*(e_i,e_j)\\ &=(a_1\vee a_2\vee a_3\vee a_4)\wedge(a_1)\wedge\\ &\quad(a_1\vee a_3\vee a_4\vee a_5)\wedge\cdots\wedge\\ &\quad(a_1\vee a_2\vee a_3\vee a_5)\wedge\\ &\quad(a_2\vee a_3\vee a_4\vee a_5)\end{aligned}$$

进一步,约简 Δ 得到相对于 d 的约简属性为$\{a_1,a_2,a_5\}$,即资源的属主和规模不影响资源泄漏类型. 在 Δ 约简的基础上,得到 10 条资源筛选规则,即:

$rule_1:(a_1,SPD)\&(a_2,W)\&(a_5,Document)\rightarrow(d,IM)$
$rule_2:(a_1,SDD)\&(a_2,E)\&(a_5,Document)\rightarrow(d,CX)$
$rule_3:(a_1,UPD)\&(a_2,X)\&(a_5,Program)\rightarrow(d,IM)$
$rule_4:(a_1,SDD)\&(a_2,W)\&(a_5,Document)\rightarrow(d,CX)$
$rule_5:(a_1,SDD)\&(a_2,W)\&(a_5,Data)\rightarrow(d,IM)$
$rule_6:(a_1,UPD)\&(a_2,E)\&(a_5,Program)\rightarrow(d,CX)$
$rule_7:(a_1,SDD)\&(a_2,R)\&(a_5,Data)\rightarrow(d,DL)$
$rule_8:(a_1,UDD)\&(a_2,R)\&(a_5,Document)\rightarrow(d,DL)$
$rule_9:(a_1,SDD)\&(a_2,W)\&(a_5,Program)\rightarrow(d,CX)$
$rule_{10}:(a_1,SDD)\&(a_2,R)\&(a_5,Data)\rightarrow(d,DL)$

再进一步,算法 1 使用定义 6 计算 $C=\{a_1,a_2,a_5\}$ 的每个属性权重:

$$w_d(a_1)=\gamma_C(d)-\gamma_{C-\{a_1\}}(d)=1-8/10=2/10$$
$$w_d(a_2)=\gamma_C(d)-\gamma_{C-\{a_2\}}(d)=1-6/10=4/10$$
$$w_d(a_5)=\gamma_C(d)-\gamma_{C-\{a_5\}}(d)=1-7/10=3/10$$

即对于资源泄漏类型来说,资源的访问方式更为重要,类型次之,而资源的路径重要性最小.

4 基于逃逸指数阈值的资源选择方法

虽然算法 1 会生成大量的资源筛选规则,但由于用于生成资源筛选规则的敏感资源无法覆盖资源属性的所有取值,因此,在资源筛选规则和系统资源匹配过程中,存在规则和资源属性向量匹配失败的情况. 所以,方法使用语义相似度来计算语义相似度来计算资源的属性向量和每个资源筛选规则前件的相似度,并将相似度的最大值作为该系统资源的逃逸指数;进一步,为了使测试结果更为准确,本节定义了逃逸指数的阈值,并通过该阈值来筛选测试的系统资源.

4.1 逃逸指数的计算方法

本文的资源属性取值类型包括数值和本体实例两类,那么,在计算资源属性向量和资源筛选规则前件的语义相似度时,需要分别计算两种属性取值类型的语义相似度.

$$\mathrm{match}(v_i,v_j)=\frac{|v_i-v_j|}{\mathrm{range}} \tag{1a}$$

$$\mathrm{match}(v_i,v_j)=\begin{cases}1, & v_i\equiv v_j\\ \dfrac{1}{2}+\dfrac{1}{e^{(\|v_i,v_j\|-1)}}, & v_i\subseteq v_j,\ \|v_i,v_j\|\geqslant 2\\ \dfrac{1}{2\times e^{(\|v_i,v_j\|-1)}}, & v_i\supseteq v_j,\ \|v_i,v_j\|\geqslant 1\\ 0, & v_i\cap v_j=\varnothing\end{cases} \tag{1b}$$

式(1)是应用广泛的数值相似度和本体实例相似度计算方法,因此,本文使用二者来分别计算属性取值为数值、本体实例的属性相似度. 其中,式(1a)的 range 表示该属性的取值范围;式(1b)将本体实例间的匹配关系分为四个不同级别,并使用语义距离计算不同本体实例间的相似度,$\|v_i,v_j\|$表示两个本体实例间的语义距离.

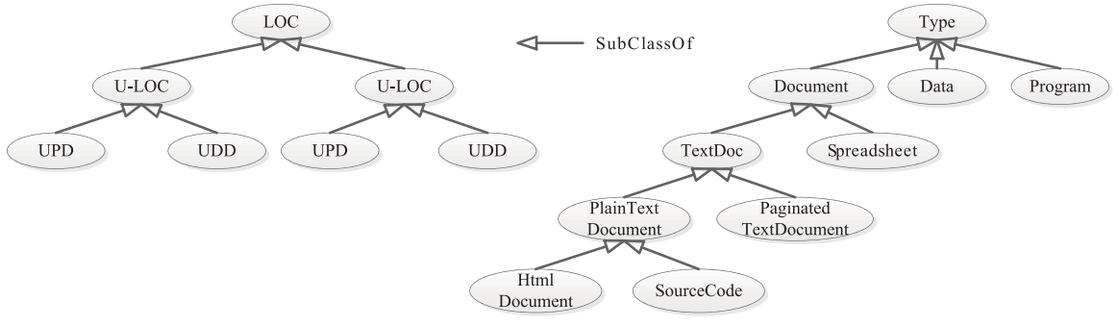
由于不同资源属性权重存在差别,因此,在属性取

值相似度计算的基础上,本节使用加权相似度来计算资源与规则的相似度,如式(2)所示,其中,$\mathrm{Sim}(u_i,\mathrm{ru})$表示系统资源 u_i 和资源筛选规则 ru 前件的相似度.

$$\mathrm{Sim}(u_i,\mathrm{ru}) = \frac{\sum_{k=1}^{m} w(a_k)\,\mathrm{match}(f(u_i,a_k),f(\mathrm{ru},a_k))}{m} \tag{2}$$

以 3.2 节生成的资源筛选规则 rule_1 和系统资源 u = <SPD,W,System,B,HTMLDocument> 为例计算 $\mathrm{Sim}(u,\mathrm{rule}_1)$. 在计算资源的路径、类型属性和资源筛选规则相应前件的相似度时,本例使用图 2 的本体片段,其中,图 2(*a*)是资源的路径属性本体片段,图 2(*b*)是资源类型属性的本体片段,那么,两者的语义相似度即为

$$\mathrm{Sim}(r,\mathrm{rule}_1) = \frac{\sum_{k=1}^{3} w(a_k)\,\mathrm{match}(f(u,a_k),f(\mathrm{rule}_1,a_k))}{3} = \frac{\frac{2}{10}+\frac{4}{10}+\frac{3}{20e^2}}{3} = \frac{12e^2+3}{60e^2}$$



(*a*) 资源的路径属性本体片段　　(*b*) 资源的类型属性本体片段

图2　资源本体片段

由于在规则生成过程中会生成大量资源筛选规则,计算候选系统资源与单个资源筛选规则的相似度无法完全反映该资源导致沙箱逃逸的能力. 因此,记资源属性向量与资源筛选规则集内规则前件的最大加权语义相似度为逃逸指数,即 $\mathrm{Escape}(u,\mathrm{RU}) = \max(\mathrm{Sim}(u,\mathrm{ru}))$.

4.2　基于逃逸指数阈值的资源选择算法

在逃逸指数计算方法的基础上,算法首先计算系统内任意资源在 RU 规则集内的逃逸指数,其次使用设定的逃逸指数阈值来筛选测试的系统资源. 细节如算法 2 所示.

算法 2　基于逃逸指数阈值的测试资源选择算法

输入:系统资源集合 Res;沙箱逃逸规则集合 RU;本体 OSO;属性权重集合 W;逃逸指数阈值 threshold
输出:测试资源集合 TestRes
(1)　初始化:EscapeRes = ∅
(2)　for (i = 0; i < Res. size, i + +)
(3)　　for (j = 0; j < RU. size, j + +)
(4)　　　计算 $\mathrm{Sim}(r_i,\mathrm{ru}_j)$
(5)　　计算 $\mathrm{Escape}(r_i,\mathrm{RU})$
(6)　　if $\mathrm{Escape}(r_i,\mathrm{RU}) \geqslant$ threshold:
(7)　　　将 r_i 加入测试资源集合 TestRes
(8)　Return TestRes

其中,为了保证筛选的系统资源与规则的相似度,又满足测试的普遍性要求,本文默认设定阈值 threshold 为 0.8. 算法包含资源数量和规则数量两个循环,即算法时间复杂度与资源数量 Res. size,规则数量 RU. size 相关,记 RU. size、Res. size 为 m、n,算法的时间复杂度为 $O(mn)$.

5　原型系统及实验分析

为了验证本文方法的有效性,本节首先构建了原型系统 BSTS(Browser Sandbox Testing System);其次,分析了方法的有效性,包括基于属性的资源筛选规则生成算法的有效性和资源筛选能力分析两部分;再次,使用 BSTS 测试了不同的浏览器沙箱来验证方法发现资源泄漏的能力.

5.1　BSTS 设计与实现

BSTS 由规则生成器、资源筛选器、资源测试器和资源访问记录器四部分组成,如图 3 所示.

(1)资源筛选规则生成器

规则生成器分析逃逸资源库中资源的属性及其取值,并使用算法 1 生成测试资源筛选规则. 敏感资源库包括两部分资源,一部分是已知的导致浏览器沙箱逃逸的资源泄漏漏洞,比如,CVE-2013-3186 的 msdt. exe,CVE-2013-4015 的 mstsc. exe;另一部分是人工分析的部分可导致沙箱逃逸的资源,比如,cmd. exe 等.

(2)资源筛选器

收集测试系统内浏览器相关资源,比如,文档、程

序、浏览器对应内存中的堆、栈对象等,计算每个资源的逃逸指数并使用逃逸指数阈值来筛选资源,并将筛选的资源置入测试资源集.

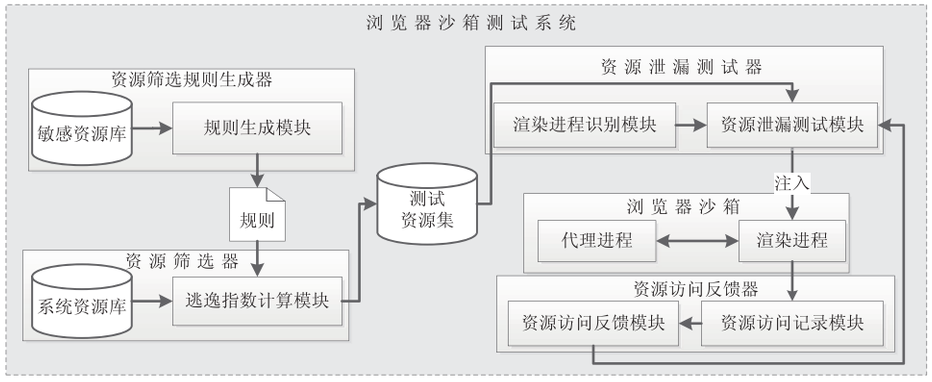


图3 浏览器沙箱测试系统

(3)资源泄漏测试器

资源泄漏测试器包括渲染进程识别模块、资源泄漏测试模块两部分. 渲染进程识别模块遍历目标浏览器的同名进程,根据进程权限、命令行参数等特征来选择渲染进程作为资源泄漏测试的载体,比如,基于 Chromium 的沙箱,使用-renderer 标识渲染进程;IE 沙箱渲染进程的路径与代理进程的路径存在差别. 资源泄漏测试模块从测试资源库中选择测试资源,并将访问该资源的代码注入选择的渲染进程.

(4)资源访问反馈器

资源访问反馈器包括资源访问记录模块和资源访问反馈模块两部分. 资源访问记录模块记录渲染进程访问的资源. 不同类型的资源泄漏判断方式存在差异,比如,代码执行需要检查相应的资源是否执行,而通过观察标记数据是否存在就可以判断读、写资源是否泄漏,因此,资源访问记录模块根据不同资源的访问方式和结果来判断是否存在资源泄漏.

5.2 方法的性能分析

本节在处理器 Intel(R) Core(TM) i7-2600 3.4GHz、内存4GB、硬盘500G,且操作系统为 Windows 7 SP1 的主机上部署 BSTS 来测试本文方法的性能. 本节首先分析了测试资源筛选规则生成算法的有效性,其次,选择不同的浏览器沙箱来测试方法的资源泄漏发现能力.

5.2.1 规则生成算法有效性分析

本节使用如下方法验证规则生成算法的有效性:定义若干条资源筛选规则,并随机生成一定量的资源属性向量(比照数据集),将手工定义的规则应用于这些资源属性向量得到资源泄漏类型属性,这些资源属性向量和资源泄漏类型属性共同构成测试数据集.

从测试数据集中任取一部分数据作为算法 1 的输入,生成资源筛选规则,并将这些资源筛选规则应用于比照数据集中的资源属性向量,并将得到的资源泄漏类型属性与比照数据集进行比较,得出算法 1 生成规则的有效性(effective).

本文将规则有效性(effective)定义为:

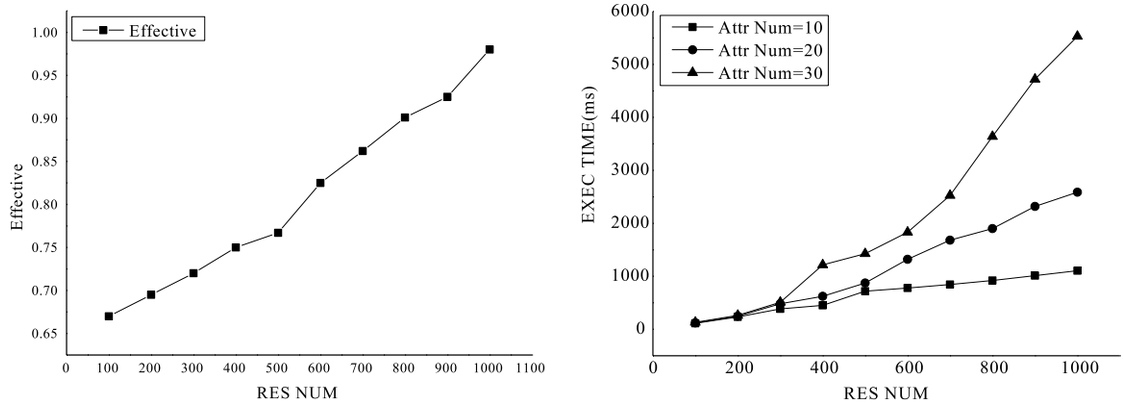
$$\mathrm{effective}(\mathrm{RG})=\frac{\|\{(r,\mathrm{dec}(\mathrm{RD},r))\}\cap\{(r,\mathrm{dec}(\mathrm{RG},r))\}\|}{\|\{(r,\mathrm{decision}(\mathrm{RD},r))\}\|}$$

其中,RD、RG 分别表示手工定义规则集合和算法 1 生成的规则;r 表示比照数据集中的资源属性向量;$\mathrm{dec}(R,r)$表示将规则集 R 应用到资源属性向量 r 上得到的资源泄漏类型属性.

在具体实验过程中,本节使用3.2 节的前三条规则作为手工定义的规则. 在这些规则的基础上,随机生成包含 5000 条资源属性向量的 U,使用包括访问方式、路径、规模、属主、机密性、参数可执行性等30 个属性描述资源. 选取其中4000 条资源属性向量作为比照数据集,并从另外 1000 条资源属性向量中选择 n 条资源属性向量作为规则生成算法的输入.

图 4(a)显示了资源属性向量规模 n 与生成规则之间的有效性关系,算法 1 生成规则的有效性随着资源属性向量基数的增加而增大,并逐步逼近最大值 1.0,即随着资源属性向量基数的增加,算法 1 生成的规则逐渐逼近手工定义的规则,当数据量充分大时,生成规则将与手工定义的规则等价.

图 4(b)显示了规则生成时间与资源属性个数以及资源属性向量个数之间的关系. 实验结果显示:①在资源属性向量个数一定的情况下,规则生成时间和资源属性的数量正相关,即随着资源属性个数的增加,规则生成时间增长加快;②在资源属性个数一定的情况下,规则生成时间和资源属性向量的数量正相关,且随着资源属性向量数量的增加,规则生成时间增长加快.



(*a*) Effective和资源数量的相关性　　(*b*) 资源生成时间和属性数量的相关性

图4 规则生成算法的有效性和规则生成时间

5.2.2 方法的资源筛选能力分析

本节测试方法的测试资源筛选能力. 分别选择 10、20 个属性来生成资源筛选规则,并使用默认阈值(0.8)来筛选测试的资源. 实验分析了敏感资源数量、资源筛选规则数量和筛选资源数量之间的关系,资源筛选结果如表 2 所示.

表 2 Windows 7 Sp1 环境的 BSTS 资源筛选效果

敏感资源数量	属性数量	规则数量	阈值	筛选资源数量	敏感资源数量	属性数量	规则数量	阈值	筛选资源数量
800	10	110	0.8	1605	800	20	190	0.8	1416
900	10	80	0.8	1302	900	20	120	0.8	1025
1000	10	50	0.8	943	1000	20	70	0.8	750

实验结果表明:

(1)在敏感资源数量不同,属性数量相同的情况下,敏感资源数量越多,约简获得的属性的区分能力越强,并且筛选获得的资源与敏感资源也更为相似;因此,生成的资源筛选规则数量随敏感资源数量增加而减少.

(2)在敏感资源数量相同,属性数量不同的情况下,生成的资源筛选规则随着属性数量的增加而增加,但筛选的资源数量却变少了. 这是由于描述资源的属性越多,约简得到属性个数也随之增加,而计算系统资源的逃逸指数需要匹配的资源属性变多. 因此,生成的资源筛选规则数量增加而筛选的资源数量减小.

5.2.3 浏览器沙箱资源泄漏的发现能力分析

在方法性能分析的基础上,本节测试方法的浏览器沙箱的资源泄漏发现能力. 本节首先使用表 2 中 10 个资源属性,800 个敏感资源生成的资源筛选规则来测试不同的浏览器沙箱,并判断是否存在资源泄漏.

在默认逃逸指数阈值(0.8)的基础上,为了充分分析方法的资源泄漏发现能力,本节通过设置逃逸指数阈值为 0.7 来提高资源泄漏的测试范围,设置逃逸指数阈值为 0.9 来提高筛选资源的相似性.

实验结果如表 3 所示,IE 和 Chrome 沙箱只有在阈值较低(0.7)的情况下泄漏个别系统资源,且人工分析发现泄漏的系统资源难以导致沙箱逃逸. 而猎豹浏览器(版本号为 v5.3.108.10728)的渲染进程具有与代理进程相同的资源访问能力,即该进程具有用户相同的资源访问能力. 攻击者可以控制渲染进程访问用户可访问的资源,即绕过猎豹浏览器的沙箱,实现沙箱逃逸.

表 3 浏览器沙箱资源泄漏测试结果

浏览器名称	版本号	逃逸指数阈值	筛选的资源数量	泄漏资源数量
		0.7	1902	1
IE	11.0.9600.18124	0.8	1605	0
		0.9	1100	0
		0.7	1902	1
Chrome	46.0.2490.86	0.8	1605	1
		0.9	1100	0
		0.7	1902	1500
猎豹浏览器	5.3.108.10728	0.8	1605	1200
		0.9	1100	1000

为了判断猎豹浏览器是否存在资源泄漏缺陷,本节使用已知造成浏览器沙箱逃逸,并用于生成资源筛选规则的 800 个敏感资源来测试这三个浏览器沙箱. 实验结果如表 4 所示,IE 和 Chrome 沙箱对已知的敏感资源实现了良好隔离,而猎豹浏览器沙箱并未有效隔离这些敏感资源,进一步验证了该版本猎豹浏览器沙箱

确实存在资源泄漏重大安全缺陷.

表 4 用于生成资源筛选规则的敏感资源的泄漏测试结果

浏览器名称	版本号	资源数量	泄漏资源数量
IE	11.0.9600.18124	800	0
Chrome	46.0.2490.86	800	0
猎豹浏览器	5.3.108.10728	800	800

进一步,在默认逃逸指数阈值(0.8)情况下,本文选择测试了 2015 年猎豹浏览器的四个发行版本. 实验结果如表 5 所示,本文分析发现猎豹浏览器的 4.5 版本及其后续版本沙箱的特定渲染进程初始化过程存在缺陷,渲染进程初始化后未进行降权操作,导致该渲染进程具有与代理进程相同的资源访问权限,能够直接访问敏感的系统资源. 而 4.3 版本的猎豹浏览器由于未引入该特定渲染进程,因而不存在该缺陷. 我们将该测试结果及缺陷的分析情况提交给中国国家信息安全漏洞库,被认定为猎豹浏览器的安全漏洞(CNNVD-201512-407).(注:该安全漏洞已在猎豹浏览器的后续版本(v 5.3.108.11895)中得到修复).

表 5 多版本猎豹浏览器沙箱资源泄漏测试结果

浏览器名称	版本号	测试的资源数量	泄漏的资源数量
猎豹浏览器	4.3.28.5985	1605	0
猎豹浏览器	4.5.35.6563	1605	1200
猎豹浏览器	4.7.53.7840	1605	1200
猎豹浏览器	5.1.73.9168	1605	1200

6 总结

本文针对浏览器沙箱的资源泄漏问题提出了一种测试方法,该方法在资源属性分析基础上,首先使用粗糙集理论生成测试资源筛选规则;其次,使用语义相似度计算方法来计算不同资源的逃逸指数,并使用阈值筛选用于测试的资源并测试浏览器沙箱. 本文实现了原型系统 BSTS,分析了方法的规则生成的有效性和资源筛选能力,进一步,本文选择测试了部分主流浏览器沙箱. 实验结果显示,本文方法具有良好的规则生成能力和资源筛选能力,且方法筛选的资源能够发现未知的资源泄漏漏洞.

参考文献

[1] VREUGDENHIL P. Adobe Sandbox When The Broker is Broken [OL]. https://cansecwest.com/ csw13archive.html,2016-01-27.

[2] NIST. CVE-2011-1353 [OL]. https://web.nvd.nist.gov/view/vuln/detail? vulnId = CVE-2011-1353, 2016 - 01-03.

[3] YASON M V. Diving into IE 10's Enhanced Protected Mode Sandbox [OL]. https://www.blackhat.com/html/bh-media-archives/bh-archives-2013.html,2016-01-07.

[4] NIST. CVE-2013-3186 [OL]. https://web.nvd.nist.gov/view/vuln/detail? vulnId = CVE-2013-3186, 2015 - 01-22.

[5] NIST. CVE-2013-4015 [OL]. https://web.nvd.nist.gov/view/vuln/detail? vulnId = CVE-2013-4015, 2016 - 01-03.

[6] KEETCH T. Escaping from Protected Mode Internet Explorer [OL]. http://archive.hack.lu/2010/ Keetch-Escaping-from-Protected-Mode-Internet-Explorer-slides. ppt, 2015-12-13.

[7] YASON M V. Understanding the Attack Surface and Attack Resilience of Project Spartans New EdgeHtml Rendering Engine [OL]. https://www.blackhat.com/html/ bh-media-archives/bh-archives-2015.html,2015-03-05.

[8] FORSHAW J. Digging for Sandbox Escapes Finding sandbox breakouts in Internet Explorer [OL]. https://www.blackhat. com/html/bh-media-archives/bh-archives-2014.html,2015-12-15.

[9] FORSHAW J. The Windows Sandbox Paradox [OL]. http://nullcon. net/website/archives/ppt/goa-15/the-windows-sandbox-paradox.pdf,2015-12-11.

[10] LI Xiao-ning, LI Hai-fei. Smart COM Fuzzing-Auditing IE Sandbox Bypass in COM Objects [OL]. https://cansecwest.com/slides/2015/Smart_COM_Fuzzing_Auditing_IE_Sandbox_Bypass_in_COM_Objects-Xiaoning_li.pdf,2015-11-13.

[11] LIU Zhen-hua, LOVET G. Breeding Sandworms: How to fuzz your way out of Adobe Reader's Sandbox [OL]. http://media.blackhat.com/bh-eu-12/Liu_Lovet/bh-eu-12-Liu_Lovet-Sandworms-WP.pdf,2016-01-05.

[12] CUI Bao-jiang, JI Yu-peng, WANG Jian-xin. An instruction-level symbolic checksum system for Windows X86 program [J]. Chinese Journal of Electronics, 2012, 21(1): 22-26.

[13] CUI Bao-jiang, LIANG Xiao-bing, ZHAO Bing, et al. Detecting integer overflow vulnerabilities in binary executables based on target filtering and dynamic taint tracing [J]. Chinese Journal of Electronics, 2014, 23(2): 348-352.

[14] 王颖,谷利泽,杨义先,等. EWFT: 基于程序执行过程的白盒测试工具[J]. 电子学报,2014,42(10): 2016-2023. DOI: 10.3969/j.issn.0372-2112.2014.10.023.
WANG Ying, GU Li-ze, YANG Yi-xian, et al. EWFT: execution-based whitebox fuzzing for executables [J]. Acta Electronica Sinia, 2014, 42(10): 2016-2023. DOI: 10.3969/j.issn.0372-2112.2014.10.023. (in Chinese)

[15] 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. (in Chinese)
[16] MA Yong-bin. Ontology of Operating System [OL]. http://medianet.kent.edu/techreports/TR2006-09-01-OS-ontology/index.html,2015 - 04 - 08.
[17] ONKI. Operating system Ontology [OL]. https://onki.fi/en/browser/overview/operating-system,2015 - 03 - 09.
[18] 张文修,吴伟志,梁吉业,等. 粗糙集理论与方法[M]. 北京: 科学出版社,2001. 19 - 32.

作者简介

赵　旭　男,1986 年 1 月出生,河南三门峡人. 2016 年在解放军信息工程大学网络空间安全学院获博士学位,研究方向为网络安全、二进制程序分析、漏洞挖掘等.
E-mail: zhx0117@sina.cn

颜学雄　男,1975 年 12 月出生,湖南耒阳人. 2008 年在解放军信息工程大学信息工程学院获博士学位,现为解放军信息工程大学网络空间安全学院副教授,研究方向为网络信息安全、Web 应用系统漏洞分析等.
E-mail: yanxuexiong@sina.com