面向数据泄漏的Web沙箱测试方法

孙雅静 赵 旭 颜学雄 王清贤
(中国人民解放军信息工程大学 郑州 450001)

摘 要 数据泄漏是导致Web沙箱逃逸的重要原因,即在未授权情况下,程序可以访问系统的敏感数据。已有的Web应用安全分析方法不完全适用于发现Web沙箱的数据泄漏。设计一种面向数据泄漏的Web沙箱测试方法,在JavaScript对象建模的基础上,首先,采用深度优先的策略遍历浏览器的原生对象,获取程序可直接访问的对象集合;其次,设计敏感点导向的封装对象测试算法,获取程序间接访问的对象集合;再次,设计了多程序数据泄漏的测试算法,获取程序间可能的通信路径;最后,对比测试结果和Web沙箱的规格,以识别Web沙箱的数据泄漏。设计并实现了Web沙箱测试系统(WSTS),同时测试了不同版本的ADsafe沙箱,实验结果显示,所提方法具有良好的数据泄漏发现能力。

关键词 Web沙箱,JavaScript,数据泄漏,测试方法
中图法分类号 TP311.1 **文献标识码** A

Data Leakage Oriented Testing Method for Web Sandbox

SUN Ya-jing ZHAO Xu YAN Xue-xiong WANG Qing-xian
(PLA Information Engineering University, Zhengzhou 450001, China)

Abstract Data Leakage is an important cause of Web sandbox escape. Namely, unauthorized programs can access sensitive data of system. The existed security analysis methods of Web application are not applicable to detect data leakage of Web sandboxes. In this paper, a Web sandbox test method was proposed to detect Web sandbox data leakage. Based on the model of JavaScript object, first, the method uses depth-first strategy to traversal native objects of browser and gets the collection of directly access object. Then, the method designs sensitive-point oriented test algorithm of encapsulated objects and gets the collection of indirectly access object. Next, the method designs data leakage test algorithm of multiple applications and gets the possible communication paths of programs. Finally, the method compares the test results and the specification of tested web sandbox to detect data leakage. This paper designed and implemented a Web sandbox test system (WSTS), and tested the different versions of ADsafe. The experimental results show that the method has good ability to detect data leakage of Web sandbox.

Keywords Web sandbox, JavaScript, Data leakage, Testing method

1 引言

JavaScript是现代Web的核心,越来越多的网站通过整合第三方JavaScript程序(后文使用程序表示第三方JavaScript程序)来提供丰富的功能,比如,FaceBook允许第三方开发游戏,Yahoo和Google嵌入第三方应用等。同源策略使程序与可信网页工作在同一可信域并具有相同权限,如果程序存在恶意或缺陷,攻击者可以使用该程序盗取用户数据,如Cookie、Session信息以及用户访问习惯等[1-2]。为了保护用户的数据安全,研究者在网站中引入沙箱来限制程序,并将其称为Web沙箱[3-4]。Web沙箱限制程序访问部分敏感的JavaScript对象(DOM[5],BOM[6]以及ECMAscript[7]对象),同时提供封装部分原生对象的JavaScript对象,并通过在其中验证程序的权限来限制程序访问用户数据,如ADsafe[8],Caja[9]和FBJS[10]等。JavaScript语言具有灵活和事件驱动的特点,比如全局变量、作用域机制等,导致Web沙箱实现复杂且容易出现缺陷,例如,ADsafe的Bunch. ephemeral、FBJS的LiveMessage. prototype. setSendSuccessHandler以及String. prototype. htmlEncode都会泄漏页面的全局this指针,程序使用该指针访问并泄漏敏感数据。

为了检测Web沙箱的数据泄漏,研究者设计了不同的Web沙箱分析方法,比如,Taly等提出了JavaScript的对象能力模型SES_{light},并使用流分析方法发现ADsafe的单程序数据泄漏[11-12];Politz等设计了一种基于类型的Web沙箱分析方法,进一步将Web沙箱的数据泄漏分为单程序数据泄漏和多程序数据泄漏两类,并发现了ADsafe的多程序数据泄漏[13-14]。这些方法虽然能够发现Web沙箱的数据泄漏,但需要分析Web沙箱的源码,而一些Web沙箱并不提供源码,比如Caja。Web沙箱属于Web应用,模糊测试技术[15-16]被广泛应用于测试Web应用的缺陷,数据泄漏不一定会导致Web应用执行异常,同时由于模糊测试不理解程序逻辑,涉及应用逻辑本身的访问权限问题很难在模糊测试中被发现,例如,在测试中模糊测试器以普通用户的身份成功访问了只有管理员才能访问的功能,但是并不会被标识。因此,模糊测试技术不

孙静雅(1993-),硕士生,主要研究方向为网络安全,E-mail:qingxinyajing@foxmail. com。

完全适用于 Web 沙箱的数据泄漏测试。

基于对 Web 沙箱数据泄漏成因的分析，本文设计了一种面向数据泄漏的 Web 沙箱测试方法，使测试不再依赖于源码分析。方法包括 3 个步骤：1）原生对象遍历：采用深度优先的策略，构造测试代码并遍历访问浏览器中的原生对象，获取 Web 沙箱允许程序直接访问的原生对象集合。2）封装对象测试：Web 沙箱封装原生对象的敏感方法、属性（敏感点），并向程序提供替代的对象来满足程序的功能需求。本文将 Web 沙箱提供的对象称为封装对象。方法设计了敏感点导向的封装对象测试算法，算法首先识别封装对象的哪些方法包含敏感点，进一步通过变异相关方法的参数来获取 Web 沙箱允许程序间接访问的原生对象集合。3）多程序数据泄漏测试：在获取 Web 沙箱允许直接访问和间接访问对象的基础上，判断程序间是否存在共享对象，进一步测试程序间是否可以通过共享对象传递数据。

本文第 2 节分析了 Web 沙箱数据泄漏的成因，并描述方法的主要步骤；第 3 节介绍 3 种主要算法的细节；第 4 节介绍原型系统（Web 沙箱测试系统，WSTS）的设计和实现细节；第 5 节给出实验和结果分析，本文选择 ADsafe 作为测试的 Web 沙箱，WSTS 通过测试已知存在数据泄漏版本和最新版本的 ADsafe 来分析方法的性能；最后总结全文并展望下一步的研究方向。

2　方法概述

JavaScript 对象包含大量的属性，一部分指向其他对象或可读、写的字符串，另一部分指向可执行方法。比如，Document 的 cookie 和 domain 等是可读、写的属性，而 getElementById 和 getElementByName 是选取元素的方法。为了方便进一步的讨论，本节给出 JavaScript 对象等的概念。

定义 1　记 JavaScript 对象表示为 $o=\langle P,F\rangle$，其中，$P=\{p_1,p_2,p_3,\cdots,p_{|P|}\}$ 表示对象的读、写属性集合，$F=\{f_1,f_2,f_3,\cdots,f_{|F|}\}$ 表示对象的执行属性（方法）集合。

Array 和 Boolean 等 ECMAScript 对象，或 Window 和 Navigator 等 BOM 对象，以及 Domcument 和 Element 等 DOM 对象，是 Web 沙箱隔离的主要对象，因此在定义 JavaScript 对象的基础上将这 3 类对象定义为原生对象。

定义 2　记 $B=\{o_1,o_2,\cdots,o_{|B|}\}$ 表示原生对象集合，其中，o_i 表示第 i 个原生对象。

Web 沙箱限制程序访问 B 的元素，同时提供封装的 JavaScript 对象来满足程序的正常功能需求。比如，ADsafe 限制程序访问 DOM 对象，同时提供 dom 来满足程序的部分 DOM 子树访问需求。因此，本文定义 Web 沙箱如下。

定义 3　记 $W=\langle\beta_D,\beta_I\rangle$ 表示 Web 沙箱，其中，β_D 称为直接访问对象，由程序不需要经过 Web 沙箱代理可访问的原生对象组成，且 $\beta_D\subset B$；β_I 称为间接访问对象，由程序通过封装的 JavaScript 对象访问的原生对象组成，且 $\beta_I\subset B$。

Web 沙箱的数据泄漏包括 3 类：1）程序可直接访问的原生对象集合（β_D'）不等于 Web 沙箱的 β_D，即 Web 沙箱未按照沙箱规格限制程序访问原生对象，比如 2011 版的 ADsafe 允许程序访问 Bunch 对象的“__nodes__”属性，进而泄漏全局 DOM 对象；2）程序可间接访问的原生对象集合（β_I'）不等于 Web 沙箱的 β_I，即封装对象的方法存在缺陷，返回敏感对象，比如 2008 版 FBJS 的 LiveMessage. prototype. setSendSuccessHandler 和 String. prototype. htmlEncode 都返回全局的 this 指针；3）多程序间存在 Web 沙箱未允许的通信通道，即页面存在共享的可读写对象，多个程序间通过共享对象通信，比如 2008 版 FBJS 的（{}）. toString 和 2011 版 ADsafe 的[]. concat. channel。

基于对 Web 沙箱数据泄漏成因的分析，本文的 Web 沙箱数据泄漏测试方法包括 3 个步骤（见图 1）：1）深度优先的原生对象测试算法：算法基于 JavaScript 对象的特点，采用深度优先的策略来遍历 B 的元素并获取 β_D'，判断是否存在第一类数据泄漏；2）敏感点导向的沙箱封装对象测试算法：算法首先识别敏感点对应封装对象的方法，其次通过变异方法的参数来获取 β_I'，并判断是否存在第二类数据泄漏；3）多程序的数据泄漏测试算法：基于前两个算法获取的程序可访问对象，首先根据访问权限划分可读、写对象子集并计算两者的交集，进一步构造交集对象的读、写测试代码并测试程序间是否可以通过这些对象的通信来发现第三类数据泄漏。

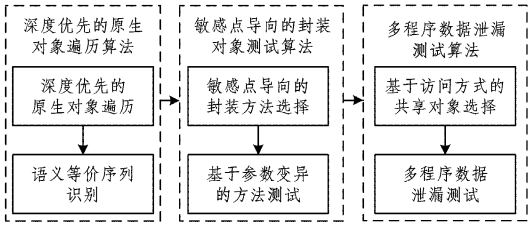


图 1　Web 沙箱数据泄漏测试方法

3　Web 沙箱数据泄漏的测试方法

本节介绍 Web 沙箱数据泄漏测试方法的细节，包括 3 部分：1）深度优先的原生对象遍历算法；2）敏感点导向的封装对象测试算法；3）多程序的数据泄漏测试算法。

3.1　深度优先的原生对象遍历算法

JavaScript 语言具有以下特点：1）基于 Prototype 的面向对象语言，且对象间通过 prototype 属性实现继承；2）Window 对象是浏览器内的 JavaScript 程序可操作的最高层对象；3）所有原型链底端都为 Object. prototype 指向的对象。因此，本节设计的原生对象遍历算法采用深度优先的遍历策略，将 Window 对象定义为起点对象，将 Object. prototype 指向的对象定义为终点对象，并通过对象的 prototype 属性实现原生对象的深度优先遍历。算法通过读、写对象的 P 集合的属性和调用 F 集合的方法来测试 Web 沙箱对具体对象的隔离效果，如果测试用例可访问的对象属性、方法集合大于 Web 沙箱允许访问的属性、方法集合，则说明 Web 沙箱未能有效隔离该对象，即存在数据泄漏。

深度优先的原生对象遍历算法如算法 1 所示，其中，$o.P$ 被划分为 $o.R$ 和 $o.W$ 两个子集，表示 Web 沙箱允许读、写的 o 对象的属性集合；$o.R'$ 和 $o.W'$ 表示测试用例实际可读、写的对象属性集合；$o.F'$ 和 $o.D'$ 分别表示测试用例实际可执行和禁止执行的 o 对象的方法集合。

算法 1　深度优先的原生对象遍历算法

输入：原生对象集合 $B=\{o_1,o_2,o_3,\cdots,o_{|B|}\}$，Web 沙箱允许直接访问的对象集合 $\beta_D=\{o_1,o_2,o_3,\cdots,o_{|\beta_D|}\}$，startObj

输出：Leak

1. 初始化：endObj←Object. prototype，TMP←∅，$o.R'\leftarrow\emptyset$，$o.W'\leftarrow\emptyset$，$o.F'\leftarrow\emptyset$，$o.D'\leftarrow\emptyset$，Leak=False
2. o←startObj

3. While o≠endObj:
4. for p in o.P:
5. if 测试程序读取 p 成功:p→o.R′
6. if 测试程序写入 p 成功:p→o.W′
7. for f in o.F:
8. if 测试程序调用 f 成功:f→o.F′
9. else f→o.D′
10. o.F′←FindSemanticEqual(o.F′,o.D′)
11. if o.R′⊃o.R ‖ o.W′⊃o.W ‖ o.F′⊃o.F
12. β_D'←o
13. o←o.prototype
14. if β_D'!=β_D:Leak=True
15. return Leak

FindSemanticEqual(S1,S2)原生对象的方法集合中存在相同或相似功能的方法(序列),如果 Web 沙箱对功能相似的方法(序列)不进行完全隔离,也容易导致程序具有危险功能并访问敏感数据,比如,String 的 split 和 match 都具有字符串分割能力,如果 Web 沙箱限制 split 而未限制 match,则导致程序仍然具有分割字符串的能力。因此,*FindSemanticEqual*(S1,S2)在算法原生对象方法遍历的基础上,在 Web 沙箱的 *o.F′* 和 *o.D′* 集合内寻找语义等价的方法(序列)。

FindSemanticEqual(S1,S2)包括两类语义等价规则来识别两个集合(S1,S2)之间的语义等价方法(序列):

1)$\{c_i \leftrightarrow y_j, c_i \in S1$ 且 $y_i \in S2\}$,即一对一等价规则,比如,String 对象实现字符串分割功能的 split 和 match 方法;

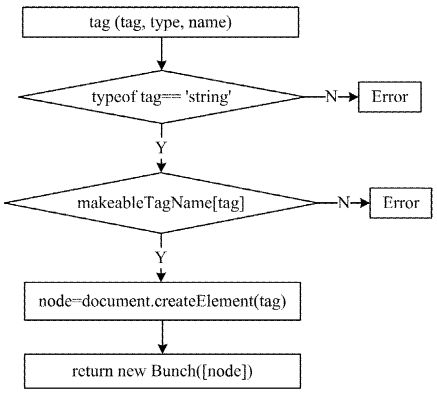
2)$\{(c_1 c_2 \cdots c_m c_n) \leftrightarrow y_i, c_i \in S1$ 且 $y_i \in S2\}$,即多对一等价,比如,String 对象的 indexOf 和 substring 序列具有与 split 相同的字符串分割功能。

3.2 敏感点导向的封装对象测试算法

程序通过 Web 沙箱的封装对象来间接访问原生对象,Web 沙箱在封装对象内验证程序的权限来保证数据安全,比如,ADsafe 的 dom.tag 方法封装 Document 对象,并在调用 document.createElement 方法前验证其参数类型、取值等,如果 tag 方法的参数验证存在缺陷,调用该方法的程序则可能在全局 DOM 树中创建节点,从而破坏 Web 沙箱的隔离,并造成数据泄漏,如图 2 所示。因此,本文设计了敏感点导向的封装对象测试算法来发现此类 Web 沙箱的数据泄漏。

```
tag:function (tag,type,name) {
        reject_global(this);
        var node;
        if (typeof tag !==‘string’) {
          error();
        }
        if (makeableTagName[tag] ! == true) {
          error(‘ADsafe: Bad tag:’ + tag);
        }
        node = document.createElement(tag);
        if (name) {
          node.autocomplete =‘off’;
          node.name = string_check(name);
        }
        if (type) {
          node.type = string_check(type);
        }
        return new Bunch([node]);
}
```

(a)tag 方法



(b)tag 参数验证过程

图 2 ADsafe 的 dom.tag 方法

敏感点导向的封装对象测试算法包括两个步骤:1)扫描封装对象的方法集合,并选择封装敏感点的方法作为待测试的方法;2)变异每个待测试方法的参数并监控方法的输出来判断方法是否存在数据泄漏,算法细节如算法 2 所示。

算法 2 敏感点导向的封装对象测试算法

输入:Web 沙箱的封装对象集合 $S=\{o_1,o_2,o_3,\cdots,o_{|S|}\}$,敏感点集合 $CriticalSinks=\{s_1,s_2,\cdots,s_{|CriticalSinks|}\}$,$\beta_I$

输出:Leak

1. 初始化:TF←Ø,Leak=False
2. While o_i exist in S:
3. for f in o_i.F:
4. for s_k in CriticalSinks:
5. if f 包含 s_k;
6. TF←o_i.f
7. 从 S 内移除 o_i
8. for f in TF:
9. 变异 f 的参数并记录 f 执行的返回结果 r
10. if r⊂B:r←β_I'
11. if β_I'⇔β_I:Leak=True
12. return Leak

其中,*CriticalSinks* 表示敏感点集合,包括 this 等全局变量以及 eval 和 execScript 等原生对象的敏感方法。

3.3 多程序数据泄漏测试算法

基于前两种算法获取的单程序可访问对象,Web 沙箱的多程序数据泄漏测试算法首先划分 β_D' 和 β_I' 的对象,并选择程序同时具有读、写权限的对象作为测试对象;其次,在 Web 沙箱内,按照先写后读的顺序执行测试程序,并监控具有读权限的测试程序是否可以读取具有写权限的测试程序写入的数据来判断多程序的数据泄漏。

多程序数据泄漏测试算法如算法 3 所示,其中,*TS* 和 *TO* 分别表示对象的初始划分集合和测试对象集合;同时,为了方便区分用于插入读、写测试对象代码的程序模板,算法将相同的程序模板分别表示为 w_1 和 w_2。

算法 3 多程序数据泄漏测试算法

输入:β_D',β_I',程序模板 w_1 和 w_2

输出:多程序数据泄漏对象集合 LD

1. 初始化:TO←Ø,TS←Ø,LD←Ø
2. TS←$\beta_D' \cup \beta_I'$
3. While o in TS:

4.　for o. p in o. P：
5.　　if (WS 允许读 o. p)&(WS 允许写 o. p)：
6.　　TO←o. p
7.　从 TS 移除 o
8. for t in TO：
9.　w_1 内插入写 t 的代码，写入标志数据 d_t
10.　w_2 内插入读 t 的代码
11.　按 $w_1 \rightarrow w_2$ 顺序在 Web 沙箱内执行二者
12.　if w_2 读取到 d_t：LD←t
13. return LD

4　WSTS 的设计与实现

为了验证测试方法的性能，设计并实现了 Web 沙箱测试系统(WSTS)。WSTS 包括测试用例生成器、数据访问监控器和数据泄漏分析器(见图 3)。

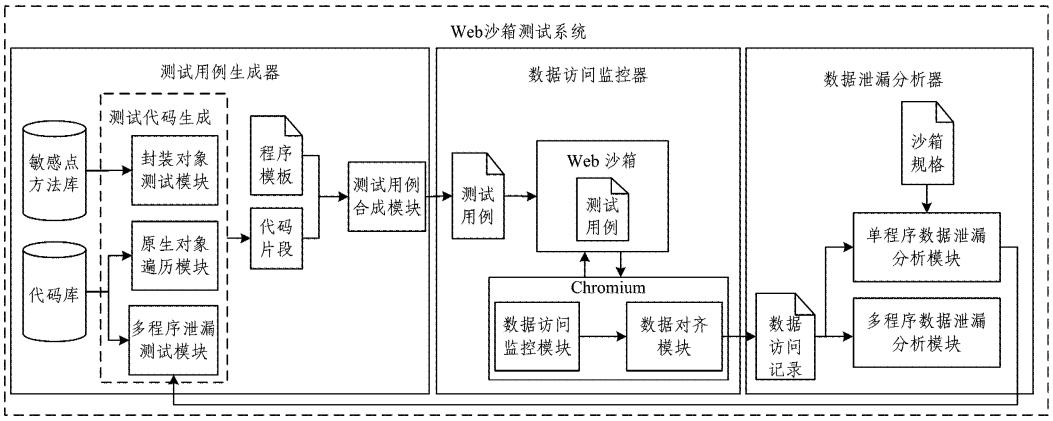


图 3　Web 沙箱测试系统

其中，测试用例生成器用于生成 Web 沙箱数据泄漏的测试代码片段，并将代码片段封装成测试用例；数据访问监控器基于 chromium 的 V8 引擎而实现，用于记录测试代码的执行情况和数据访问情况；数据泄漏分析器则根据测试代码的数据访问情况来判断 Web 沙箱是否存在数据泄漏。

(1)测试用例生成器

测试用例生成器由封装对象测试模块、原生对象遍历模块、多程序泄漏测试模块和测试用例合成模块组成。其中，封装对象测试模块从敏感点方法库中获取包含敏感点的封装对象的方法，并进一步变异方法对应的参数来生成用于测试的代码片段。比如，图 2 的 dom. tag 方法包含敏感点 document. createElement 方法，那么封装对象测试模块识别 dom. tag 方法的参数类型，并通过变异其参数来生成用于测试的代码片段。WSTS 测试的敏感点不仅包括已知易导致数据泄漏的属性和方法(如 cookie，eval 等)，还包括 Web 沙箱禁止访问对象的方法(如 ADsafe 禁止访问的 Document 的方法)。

原生对象遍历模块和多程序泄漏测试模块都从代码库中获取原生对象的访问代码，两者的差别在于原生对象遍历模块根据深度优先的原生对象遍历算法从代码库中选择测试代码，而多程序泄漏测试模块则根据数据泄漏分析器反馈的结果从代码库中选择测试代码。本文使用 Scrapy[17] 爬取了 10000 个不同网页的 JavaScript 代码(包含使用被测试 Web 沙箱的网页)，并根据访问对象的不同分类和筛选了 JavaScript 代码，同时为了尽可能多地测试 Web 沙箱的对象访问限制，WSTS 保留了相同对象的不同访问方式的代码，比如 obj. x 和 obj['x']都可以访问对象 obj 的 x 属性，而方法调用代码不仅包括直接调用代码，还包括使用 apply 和 call 调用代码。

测试用例合成模块将测试代码生成阶段选择的代码片段和程序模板组合成 Web 沙箱的测试用例。以 ADsafe 为例，图 4(a)为 widget 的程序模板，测试代码分别为读取 input_text 节点数据和该节点赋值这两个操作的代码；测试用例合成模块将 widget 模板与测试代码合成测试用例，如图 4(b)和图 4(c)所示。其中，图 4(b)、图 4(c)分别是读、写 input_text 属性的 widget。

```
〈div id="TESTTEMPLET"〉
〈script〉ADSAFE. id('TESTTEMPLET')〈/script〉
〈script〉
  ADSAFE. go('TESTTEMPLET', function (dom, arg1, arg2, …, argn) {
    "use strict";
    代码插入位置
    });
  〈/script〉
〈/div〉
```

(a)widget 模板

```
〈div id="TESTTEMPLET"〉
〈script〉ADSAFE. id('TESTTEMPLET')〈/script〉
〈script〉
  ADSAFE. go('TESTTEMPLET', function (dom, lib) {
    "use strict";
    var a=dom. q("input_text");
});
〈/script〉
〈/div〉
```

(b) 获取"input_text"节点取值的测试 widget

```
〈div id="TESTTEMPLET"〉
〈script〉ADSAFE. id('TESTTEMPLET')〈/script〉
〈script〉
  ADSAFE. go('TESTTEMPLET', function (dom, lib) {
    "use strict";
    dom. q("input_text"). value("hello world");
});
〈/script〉
〈/div〉
```

(c) "input_text"节点赋值的测试 widget

图 4　ADsafe 的测试用例

(2)数据访问监控器

数据访问监控器由数据访问监控模块和数据对齐模块组成，部分内容如下所示。其中，数据访问监控模块监控测试用例的数据访问情况，为了能够减少 JavaScript 对象别名问题

对数据泄漏分析的影响，数据对齐模块采用语义的方法对齐的方式标识相同的对象，即通过引入基于地址的对象读、写操作扩展的 JASIL 来表示程序的操作语义，同时用于记录程序访问的数据。

$$
\begin{aligned}
&\tau ::= \eta \mid \beta[\eta] \mid \text{Bool} \mid \text{Null} \mid \text{Undef} \mid \text{PC} \\
&\eta ::= \text{Int} \mid \beta \\
&\beta ::= ref(\tau) \mid \text{String}(\kappa) \mid \text{Char}(\kappa) \\
&\kappa ::= UTF8 \mid UTF7 \mid \cdots \\
&x:\tau :: v:\tau \quad \text{(Assignment, Type Conversion)} \\
&x:\tau :: *(v:ref(\tau)) \quad \text{(Dereference)} \\
&x:\text{Int} :: v1:\text{Int op } v1:\text{Int} \quad \text{(Arithmetic)} \\
&x:\text{Bool} :: v1:\text{Bool op } v1:\text{Bool} \quad \text{(Logical)} \\
&x:\text{Bool} :: v1:\tau \text{ op } v1:\tau \quad \text{(Relational)} \\
&x:PC :: \text{if}(testvar:\text{Bool})\text{then}(c:\text{Int})\text{else}(c:\text{Int}) \quad \text{(Control Flow)} \\
&x:\text{String} :: substring(s:\text{String}, startpos:\text{Int}, len:\text{Int}) \quad \text{(String Ops)} \\
&x:\text{String} :: concat(s1:\text{String}, s2:\text{String}) \quad \text{(String Ops)} \\
&x:\text{String} :: fromArray(s1:ref(\tau)) \quad \text{(String Ops)} \\
&x:\text{String} :: convert(s1:\text{String}) \quad \text{(String Ops)} \\
&x:\text{Char}*\kappa :: convert(i:\text{Int}) \quad \text{(Character Ops)} \\
&x:\text{Int} :: convert(i:\text{Char}*\kappa) \quad \text{(Character Ops)} \\
&x:\tau :: \mathbf{F}(i_1:\tau, \cdots, i_n:\tau) \quad \text{(Uninterpreted Function Call)} \\
&x:obj :: read(x:obj, addr) \quad \text{(read Object)} \\
&x:obj :: write(x:obj, addr) \quad \text{(write Object)}
\end{aligned}
$$

数据访问监控模块记录程序访问的数据，通过两种方式判别数据的禁止访问：1）如果测试用例未执行，且 Web 沙箱提示程序越权访问对象属性，则记录该属性为禁止访问属性；2）如果测试用例可执行，但未能操作目标属性，则认为 Web 沙箱禁止访问对象属性，WSTS 将对象的该属性也标记为禁止访问属性。

（3）数据泄漏分析器

数据泄漏分析器通过对比 Web 沙箱的规格和获取的程序访问记录来识别 Web 沙箱的数据泄漏。数据泄漏分析器包括单程序数据泄漏分析模块和多程序数据泄漏分析模块，其中单程序数据泄漏分析模块不仅分析 Web 沙箱的单程序数据泄漏，而且将单程序的数据泄漏访问结果返回给多程序泄漏测试模块；多程序数据泄漏分析模块在单程序数据泄漏分析模块向多程序泄漏测试模块反馈结果后执行。

5 实验验证与结果分析

本文在处理器为 Intel(R) Core(TM) i7-2600 3.4GHz、内存为 8GB、硬盘容量为 1TB、操作系统为 Windows 8 的实验机上部署 WSTS，并选择 ADsafe 作为测试对象来测试方法的性能。WSTS 首先测试了已知存在数据泄漏的 ADsafe 的早期版本（2011），并根据 WSTS 是否可以发现该版本 ADsafe 的已知数据泄漏来验证方法的有效性；其次，测试 2015 版的 ADsafe 来分析方法对未知数据泄漏的发现能力；在 WSTS 测试 ADsafe 的过程中，本文统计了 WSTS 测试 ADsafe 的代码覆盖率，以进一步分析的效率。

5.1 方法有效性验证

ADsafe 提供多个选项以满足数据库、测试以及网页的沙箱化需求，其中，Browser 和 ES6 选项用于限制不可信 JavaScript程序，因此 WSTS 的测试基于 ADsafe 来选择 Browser 和 ES6 两个选项。WSTS 测试的步骤如下：

（1）遍历原生对象。被遍历的主要原生对象如表 1 所列。以 WSTS 遍历到 History 为例，该对象包括 length 和 state 两个读、写属性以及 go，back 和 forward 3 个方法，WSTS 分别构造读和写 length、state 属性以及执行 3 个方法的测试用例；进一步，WSTS 记录测试用例访问属性和执行方法的结果，并与 ADsafe 的规格进行对比来判断是否泄漏原生对象的属性。

表 1 原生对象遍历表

序号	对象	属性数量	方法数量
1	Array	3	14
2	Boolean	2	3
3	Date	2	46
4	Math	8	20
5	Number	7	6
6	String	3	34
7	RegExp	5	3
8	Window	22	19
9	Navigator	12	2
10	Screen	13	0
11	Document	7	7
12	Location	8	3
13	History	2	5

（2）敏感点导向的封装对象测试。ADsafe 包括 5 个封装对象，如表 2 所列，WSTS 首先扫描包含敏感点的封装对象的方法，比如，“__”包裹的属性、全局变量以及 adsafe.js 封装的 DOM 对象的属性和方法；其次，变异测试包含敏感点方法的参数。在测试 Bunch.prototype.getStyle 时，当参数变异为“__proto__”时，方法返回全局 this 指针，Bunch.ephemeral 不接受参数，WSTS 直接调用该方法并返回全局 this 指针，这样便验证了 ADsafe 存在的两个数据泄漏。

表 2 ADsafe 封装对象列表

序号	对象	属性数量	方法数量
1	ADSAFE	0	12
2	dom	0	11
3	hunter	0	6
4	pecker	0	24
5	Bunch	2	52

（3）多程序的数据泄漏测试。WSTS 在前两步工作的基础上分析 ADsafe 允许程序读、写的原生对象、封装对象的属性，发现 Array 实例[]的 concat 方法可以被多个程序共享，即程序间可以通过[].concat 实现通信，这样便验证了 ADsafe 存在的多程序数据泄漏。

实验结果表明，本文的数据泄漏测试方法能够发现 ADsafe 中已存在的数据泄漏，即方法是有效的。

5.2 未知数据泄漏的发现能力

WSTS 在多程序数据泄漏测试过程中除验证[].concat 导

致的 ADsafe 多程序数据泄漏外，还发现了 ADsafe 的另一个多程序数据泄漏——lib. toString。

WSTS 的测试过程如下：ADSAFE. lib(name，f)允许执行 f 指向的方法来完成名字为 name 的 lib 对象的初始化，即该方法包含敏感点；WSTS 发现 lib 对象可以被不同的测试程序(widget)读、写，即 lib 对象可能存在多程序的数据泄漏，因此 WSTS 通过从代码库中选择读、写 lib 对象各属性的代码来构造测试用例，以测试 lib 对象是否存在多程序的数据泄漏。实验结果显示，lib. toString 可以在 widget 之间传递数据，即 ADsafe 未能有效隔离 lib。同时，本文在 2015 版的 ADsafe 中也验证了该多程序的数据泄漏。

本文选择 lib. toString. channel 作为示例来描述该多程序数据泄漏的利用过程，如图 5 所示。

```
〈div id="SETMESSAGE_"〉
〈p〉trust me and give me some secret〈/p〉
〈input type=text autocomplete=off〉
〈p id="SETMESSAGE_A"〉〈/p〉
〈script〉ADSAFE. id('SETMESSAGE_')〈/script〉
〈script src="comm. js"〉〈/script〉
  〈script〉
  ADSAFE. go('SETMESSAGE_',function (dom,lib) {
      "use strict";
      var AfromLib;
      var input = dom. q("input_text");
      AfromLib = dom. tag('input','button')
      . value('setMessage')
      . on('click',function (e) {
      lib. comm(). toString. channel=input. getValue();});
      dom. append(AfromLib) });
  〈/script〉
〈/div〉
```

(a) 数据写入 widget

```
〈div id="GETMESSAGE_"〉
〈p〉get secret from the trust widget. 〈/p〉
〈p id="GETMESSAGE_RESULT"〉〈/p〉
〈script〉ADSAFE. id('GETMESSAGE_')〈/script〉
〈script src="comm. js"〉〈/script〉
〈script〉
ADSAFE. go("GETMESSAGE_",function (dom,lib) {
  "use strict";
  var getMessage;
      var show=dom. q('#GETMESSAGE_RESULT');
      getMessage = dom. tag('input','button')
      . value('getMessage')
      . on('click',function (e) {
      show. value(lib. comm(). toString. channel); });
      dom. append(getMessage)});
〈/script〉
〈/div〉
```

(b) 数据读取 widget

```
ADSAFE. lib("comm",function () {
  "use strict";
  return function () {
      return {
      getA :function(){},
      setA: function(a){},};};});
```

(c) 测试 lib

图 5　ADsafe 数据泄漏利用代码

该数据泄漏利用代码模拟存在 widget 从而获取用户的敏感输入，并将该敏感数据传递给恶意 widget 的过程。其中，图 5(a)的 widget 具有写 lib. toString. channel 的权限，并通过 input. getValue 获取用户的敏感输入；图 5(b)的 widget 通过读取 lib. comm(). toString. channel 属性来获取图 5(a)的 widget 传递的数据；图 5(c)是实现数据传递的示例 lib 对象(comm)。

5.3　数据泄漏测试方法的效率分析

代码覆盖率是用于分析及测试方法性能的一项重要指标，2015 版 ADsafe 的 JSLint. js 和 adsafe. js 分别包含 4425 行和 1931 行代码。本文统计并计算了测试时间为 5000s 的 ADsafe 代码覆盖率，统计结果如图 6 所示。ADsafe 的代码覆盖率随着测试时间的增加而增加，但增速逐渐变慢。这是由于：1)ADsafe 的 JSLint. js 存在部分无法被 WSTS 测试的沙箱初始化代码；2)WSTS 只选择部分包含敏感点的封装方法测试 adsafe. js，并未测试 adsafe. js 的所有方法，同时，WSTS 通过随机变异方法的参数来测试包含敏感点的封装方法，无法完全覆盖封装方法的所有路径，因此 ADsafe 的代码覆盖率无法达到 100%。但在观察时间内，WSTS 分别覆盖了 JSLint. js 和 adsafe. js 的 3477 行和 1661 行代码，即方法的代码覆盖率分别达到了 78.5%和 86%，实验结果表明方法的代码覆盖率是可被接受的。

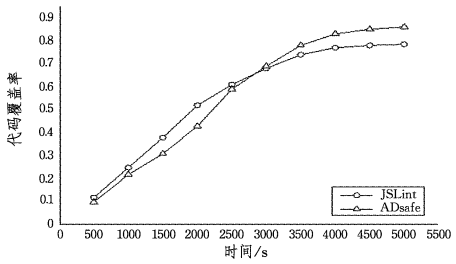


图 6　ADsafe 代码覆盖率

结束语　本文设计了一种 Web 沙箱的数据泄漏测试方法。该方法首先分析了 Web 沙箱数据泄漏的成因；其次，设计了深度优先的原生对象遍历算法、敏感点导向的封装对象测试算法和多程序数据泄漏测试算法；再次，设计了原型系统 WSTS 并测试了 2011 和 2015 两个版本的 ADsafe，不仅验证了 ADsafe 已有的数据泄漏，而且发现了一个 ADsafe 的未公开多程序数据泄漏。实验结果显示，本文方法具有良好的数据泄漏发现能力，并且其代码覆盖能力在可接受范围。

在敏感点导向的封装对象测试的基础上，下一步研究将引入符号执行、污点分析等精确测试技术来提高测试的代码覆盖率，并发现更多的 Web 沙箱数据泄漏。

参 考 文 献

[1] BHARGAVAN K，DELIGNAT L A，MAFFEIS S. Defensive JavaScript[M]// Foundations of Security Analysis and Design VII. Springer International Publishing，2014：88-123.

[2] TRIPP O，FERRARA P，PISTOIA M. Hybrid security analysis of web javascript code via dynamic partial evaluation[C]// Proceedings of the 2014 International Symposium on Software Testing and Analysis. ACM，2014：49-59.

[3] POLITZ J G，ARJUN G，SHRIRAM K. Typed-based verifica-

tion of Web sandboxes[J]. Journal of Computer Security, 2014, 22(4): 511-565.

[4] MAASS M, SALES A, CHUNG B, et al. A systematic analysis of the science of sandboxing[J]. BMC Evolutionary Biology, 2016, 2(3): e43.

[5] JavaScript and the Document Object Model[OL]. http://www.ibm.com/developerworks/web/library/wa-jsdom.

[6] The Browser Object Model[OL]. http://msdn.microsoft.com/en-us/library/ms952643.aspx.

[7] ECMAScript Language Specification 2015[OL]. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

[8] DOUGLAS C. ADsafe[OL]. http://www.adsafe.org.

[9] MARK S, MILLER S, BEN L, et al. Caja: Safe active content in sanitized JavaScript[OL]. http://google-caja, googlecode.com/files/caja-spec-2008-06-07.pdf.

[10] Facebook. FBJS[OL]. http://developers.facebook.com/docs/fbjs.

[11] SERGIO M, MITCHELL J C, TALY A. Isolating JavaScript with filters, rewriting, and wrapper [C]// European Symposium on Research in Computer Security. Springer Berlin Heidelberg, 2009: 505-522.

[12] TALY A, ERLINGSSON Ù, MITCHELL J C, et al. Automated analysis of security-critical javascript apis[C]// 2011 IEEE Symposium on Security and Privacy (SP). IEEE, 2011: 363-378.

[13] POLITZ J G, SPIRIDON A E, ARJUN G, et al. ADsafety: Typed-based verification of JavaScript sandboxing[C]// Proceedings of the 20th USENIX Conference on Security. 2011.

[14] POLITZ J G, ARJUN G, SHRIRAM K, Semantics and Types for Objects with First-Class Member Names[M]// Workshop on Foundations of Object-Oriented Languages (FOOL). 2012: 15-22.

[15] SAXENA P, AKHAWE D, HANNA S, et al. A symbolic execution framework for javascript[C]// 2010 IEEE Symposium on Security and Privacy (SP). IEEE, 2010: 513-528.

[16] LI Y F, PARAMJIT K D, DAVID L D. Two decades of Web application testing-A Survey of recent advances[J]. Information System, 2014, 43: 20-54.

[17] DANIEL M, JAMES W. Choosing Scrapy[J]. Journal of Computing Sciences in Colleges, 2015, 31(1): 83-89.

(上接第 292 页)

[7] MENZEL M, RANJAN R, WANG L, et al. CloudGenius: a hybrid decision support method for automating the migration of web application clusters to public clouds[J]. IEEE Transactions on Computers, 2015, 64(5): 1336-1348.

[8] ZHU H M, WU L F, HUANG K Y, et al. Research on Methods for Discovering and Selecting Cloud Infrastructure Services Based on Feature Modeling[OL]. http://downloads.hirdawi.com/journals/mpe/2016/8194832.pdf.

[9] CROSBY S, DOYLE R, GERING M, et al. Open virtualization format specification[R]. Distributed Management Task Force, 2010.

[10] DEB K, PRATAP A, AGARWAL S, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II[J]. IEEE transactions on evolutionary computation, 2002, 6(2): 182-197.

[11] COELLO C A C, PULIDO G T, LECHUGA M S. Handling multiple objectives with particle swarm optimization[J]. IEEE Transactions on Evolutionary Computation, 2004, 8(3): 256-279.

[12] PAWLUK P, SIMMONS B, SMIT M, et al. Introducing STRATOS: A cloud broker service[C]// 2012 IEEE 5th International Conference on Cloud Computing (CLOUD). IEEE, 2012.

[13] PETCU D, DI MARTINO B, VeENTICINQUE S, et al. Experiences in building a mOSAIC of clouds[J]. Journal of Cloud Computing: Advances, Systems and Applications, 2013, 2(1): 12.

[14] KESSACI Y, MELAB N, TALBI E G. A pareto-based genetic algorithm for optimized assignment of vm requests on a cloud brokering environment [C]// 2013 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2013.

[15] SIEGEL J, PERDUE J. Cloud services measures for global use: the Service Measurement Index (SMI)[C]// 2012 Annual SRII Global Conference (SRII). IEEE, 2012.

[16] LI Z, O'BRIEN L, ZHANG H, et al. On a catalogue of metrics for evaluating commercial cloud services[C]// Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing. IEEE Computer Society, 2012.

[17] GARG S K, VERSTEEG S, BUYYA R. A framework for ranking of cloud computing services[J]. Future Generation Computer Systems, 2013, 29(4): 1012-1023.

[18] SPEC[OL]. http://www.spec.org.

[19] Cloudharmony[OL]. http://www.cloudharmony.com.

[20] 朱华旻, 吴礼发, 康红凯. 基于 SecLA 的云服务商选择方法研究[J]. 计算机科学, 2016, 43(5): 100-107.

[21] 公茂果, 焦李成, 杨咚咚, 等. 进化多目标优化算法研究[J]. 软件学报, 2009, 20(2): 271-289.

[22] 胡旺, YEN G, 张鑫. 基于 Pareto 熵的多目标粒子群优化算法[J]. 软件学报, 2014(5): 1025-1050.

[23] Rackspace[OL]. http://www.rackspace.com.

[24] Digitalocean[OL]. http://www.digitalocean.com.

[25] Linode[OL]. http://www.linode.com.

[26] ZITZLER E, DEB K, THIELE L. Comparison of multiobjective evolutionary algorithms: Empirical results [J]. Evolutionary computation, 2000, 8(2): 173-195.

[27] SCHOTT J R. Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization[J]. Cellular Immunology, 1995, 37(1): 1-13.

[28] SBALZARINI I F, MÜLLER S, KOUMOUTSAKOS P. Multiobjective optimization using evolutionary algorithms[C]// Proceedings of the Summer Program. CiteSeer, 2000.

[29] Minitab[OL]. http://www.minitab.com/en-us.