Journal of Computer Applications
计算机应用,2015,35(1):72-76,82
ISSN 1001-9081
CODEN JYIIDU
2015-01-10
http://www.joca.cn

文章编号:1001-9081(2015)01-0072-05
doi:10.11772/j.issn.1001-9081.2015.01.0072

具有时间多样性的 JavaScript 代码保护方法

房鼎益[1,2],党舒凡[1,2*],王怀军[1,2],董　浩[1,2],张　凡[2,3]
(1.西北大学 信息科学与技术学院,西安 710127;　2.西北大学-爱迪德物联网信息安全联合实验室(西北大学),西安 710127;
3.爱迪德技术(北京)有限公司,北京 100125)
(*通信作者电子邮箱 dangshufan@163.com)

摘　要:Web 应用同本地应用一样面临恶意主机威胁。如何确保暴露于用户主机中的 Web 应用核心算法或关键业务流程等重要信息的安全成为亟待解决的问题。针对现有 JavaScript 代码保护方法难以抵御动态分析且抗累积攻击效果差的问题,提出了一种具有时间多样性的 JavaScript 代码保护(TDJSP)方法。首先,通过程序多样化处理和路径空间模糊化,使 JavaScript 程序在执行时具有多样性效果,以有效抵御累积攻击;其次,检测调试器、模拟器等非正常执行环境的特征,并根据检测结果进行响应,增加攻击者进行动态分析的难度。理论分析和实验结果表明,JavaScript 程序的抗逆向分析能力得到了提高,同时,其空间增长率约为 3.1(优于 JScrambler3),时间延迟为毫秒级。因此,该方法能够在不影响程序性能的前提下提升 Web 应用的安全性。

关键词:Web 应用;JavaScript 代码保护;累积攻击;时间多样性;代码混淆
中图分类号:TP309;TP311　　**文献标志码**:A

JavaScript code protection method based on temporal diversity

FANG Dingyi[1,2], DANG Shufan[1,2*], WANG Huaijun[1,2], DONG Hao[1,2], ZHANG Fan[2,3]
(1. *School of Information Science and Technology, Northwest University, Xi'an Shaanxi* 710127, *China*;
2. *NWU-Irdeto Internet of Things and Infomation Security Joint Laboratory (Northwest University), Xi'an Shaanxi* 710127, *China*;
3. *Irdeto Access Technology (Beijing) Company Limited, Beijing* 100125, *China*)

Abstract: Web applications are under the threat from malicious host problem just as native applications. How to ensure the core algorithm or main business process's security of Web applications in browser-side has become a serious problem needed to be solved. For the problem of low effectiveness to resist dynamic analysis and cumulative attack in present JavaScript code protection methods, a JavaScript code Protection based on Temporal Diversity (TDJSP) method was proposed. In order to resist cumulative attack, the method firstly made the JavaScript program obtain the diverse ability in runtime by building program's diversity set and obfuscating its branch space. And then, it detected features of abnormal execution environments such as debuggers and emulations to improve the difficulty of dynamic analysis. The theoretical analyses and experimental results show that the method improves the ability of JavaScript program against the converse analysis. And the space growth rate is 3.1 (superior to JScrambler3) while the delay time is in millisecond level. Hence, the proposed method can protect Web applications effectively without much overhead.

Key words: Web application; JavaScript code protection; cumulative attack; temporal diversity; code obfuscation

0　引言

Web 2.0 的提出使用户对 Web 应用交互能力的要求不断提高,Web 应用的逻辑已逐渐由服务器端向浏览器端转移。同时,HTML5 引入的许多本地特性加剧了这一趋势。作为 Web 应用的默认脚本语言,JavaScript 被越来越多地用于实现程序的核心功能。JavaScript 作为解释型语言,程序以源码形式发布且具有良好的可读性,攻击者可轻易对代码进行查看、分析和篡改。这使得由 JavaScript 承载核心算法或关键业务流程等重要信息的 Web 应用面临逆向分析的安全威胁:一方面,程序知识产权遭受侵犯,将影响互联网产业的健康发展;另一方面,程序机密性的破坏,使攻击者能够针对 Web 应用的逻辑漏洞展开进一步网络攻击[1]。如何有效对 JavaScript 代码进行保护成为亟待解决的问题。

目前,JavaScript 代码保护主要有混淆和加密两种方法[2],可在一定程度上抵御逆向分析,但存在以下缺陷:1)难以抵御动态分析;2)保护后的程序是静态对象(即每次运行时代码相同),且执行过程对攻击者完全可见[3]。这使得保护者在与攻击者博弈时处于劣势地位。对此,本文从抑制逆向分析过程中的经验累积出发,引入了一种具有时间多样性的 JavaScript 代码保护(JavaScript Code Protection based on Temporal Diversity, TDJSP)方法。该方法一方面以多样化的

收稿日期:2014-07-25;**修回日期**:2014-09-26。　**基金项目**:国家自然科学基金资助项目(61202393);国家科技支撑计划项目(2013BAK01B02);陕西省教育厅产业化培育项目(2013JC07);陕西省自然科学基础研究计划项目(2012JQ8049)。

作者简介:房鼎益(1959-),男,陕西汉中人,教授,博士,主要研究方向:网络与信息安全、软件安全与保护、无线传感器网络;　党舒凡(1990-),女,陕西咸阳人,硕士研究生,主要研究方向:软件安全与保护、软件攻击;　王怀军(1981-),男,山东滕州人,博士,主要研究方向:软件安全与保护、软件攻击及软件保护有效性评测;　董浩(1989-),男,陕西西安人,硕士研究生,主要研究方向:软件安全与保护、软件攻击;张凡(1973-),男,陕西西安人,博士,主要研究方向:软件安全、数字内容保护。

攻击对象和路径空间模糊为基础,提高逆向分析的难度,延长被保护程序的安全周期;另一方面,通过在程序中插入环境监测代码,对非正常的执行环境进行感知及响应,以达到抵御动态分析的目的。

1　程序多样性与累积攻击

1.1　程序多样性

Cohen[4]提出代码变形可用于创建程序的多样性集以降低同构程序的易攻击性。在随后的发展中,程序多样性技术讨论了一系列不改变程序语义的代码变形技术,如等价指令变换、函数内联和外联、垃圾代码注入、模拟和运行时代码生成等[5-6]。目前,程序多样性技术被用于抵御多种形式的 MATE(Man-At-The-End)攻击,如非法逆向、篡改以及非法分发[7]203-209等,代码混淆也已成为构建程序多样性的主要方法。Collberg 等[8]通过在可信服务器端设置变异进程,把经过混淆的代码块派发给客户端,以抵御攻击者对客户端的篡改。JShadObf 混淆器则通过对 JavaScript 程序施加不同的变形算法序列产生代码种群,然后利用进化算法选取符合条件的个体以达到提升混淆质量的目的[9]。

2011 年,Collberg 等[10-11]提出程序在时间和空间上的多样性是持久保护数字资产的三个必要条件之一。时间多样性是指程序在不同时间执行时,其执行路径不同,但功能相同,主要用于抵御累积攻击;而空间多样性是指同一保护对象经多次保护所生成的不同版本具有不同的形态,通过降低程序的同构性来确保软件的安全,主要用于抵御共谋攻击[12-13]。

1.2　累积攻击

程序的逆向分析过程可分为若干阶段,并以黑盒阶段、动态分析阶段和静态分析阶段作为基础[7]68-72。逆向分析过程中,上述阶段交织进行,实际是一个不断对攻击目标进行学习并累积经验的过程。攻击者需要不断进行各种尝试,多次运行程序并重复上述过程以积累相应阈值的攻击知识。

在黑盒阶段,攻击者为程序给出不同的输入,并观察程序输出,继而推测程序行为与外部环境间的依赖关系;在动态分析阶段,攻击者运行程序并记录程序的某些运行时信息,对程序的内部逻辑展开分析;在静态分析阶段,攻击者则会按照局部分析、过程内分析及过程间分析三个步骤对代码直接展开阅读以更好地理解程序。

综上,攻击者的非法逆向过程可抽象为对攻击目标展开学习并进行知识累积的过程。当知识累积到一定阈值,便可顺利达到相应的攻击目的。

定义 1　累积攻击。对于逻辑复杂的程序,攻击者无法通过仅一次性执行程序获得足够的攻击知识,而必须反复运行目标程序,进行多次攻击以累积知识来达到相应的攻击目的。攻击者对应于上述过程所展开的攻击叫作累积攻击。

累积攻击的攻击场景可由函数 $p = Attack(n,k,\delta)$ 描述,如图 1 所示。

图 1 中:p 代表攻击进度,n 代表攻击次数,k 代表对于指定程序达到某一攻击目的所需积累的知识阈值,δ 代表程序的时间多样性能力。从图 1 中可以看出,当 δ 为 0 时,由于程序不具备时间多样性能力,随着攻击次数的增加,攻击者所累积的相关攻击知识持续增长并逐渐逼近阈值。而当 δ 大于 0 时,由于程序具有时间多样性能力,攻击者先前获取的攻击知识无法在后续新的攻击过程中得到有效利用。此时,攻击进度与攻击次数不再具有某种正相关的关系。

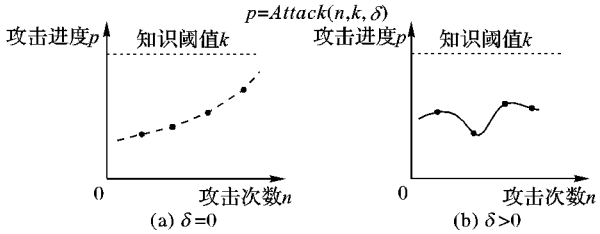

图 1　累积攻击场景示意图

综合上述分析可得出以下结论:一方面,程序的时间多样性可用于抵御累积攻击;另一方面,抑制累积攻击可以降低攻击者的攻击效率、增加攻击成本,继而对整个逆向分析过程产生消极影响,最终达到延长 Web 应用安全生命周期的目的。

2　具有时间多样性的 JavaScript 代码保护方法

2.1　方法概述

本文研究的 JavaScript 代码保护方法 TDJSP 通过有效抵御积累攻击,在增加逆向分析难度的同时延长 Web 应用的安全生命周期。主要体现在以下几个方面:

1)动态变化的攻击对象。利用程序多样性技术构建关键代码段的多样性集合,在程序执行时依据时间的不同触发关键代码段的动态选择,以此弱化攻击过程中知识累积的影响,有效抵御累积攻击。

2)降低程序对攻击者的可见度。对关键代码段进行路径空间模糊处理,即依据控制流图对代码进行分割,将其中某些分支条件隐藏于服务器端,缺失的基本块在程序运行时动态下载并执行。

3)对执行环境进行监测。对动态分析所依托的执行环境进行特征监测,并设置陷阱函数进行响应,以达到提升动态分析难度的目的。

TDJSP 对程序的保护过程如图 2 所示,主要由 4 个阶段构成:预处理、程序多样化处理、路径空间模糊、程序重构。

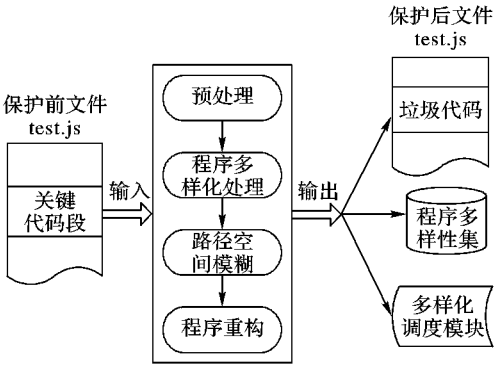

图 2　保护过程示意图

为便于描述,首先介绍相关概念。

定义 2　控制流图(Control Flow Graph, CFG)。控制流图是用于描述程序控制逻辑的有向图,表示为 CFG = (V,E)。其中:V 是节点集,E 是有向边的集合。图中一个节点对应于程序中的一个基本块,一条有向边对应程序中的一次控制流转移。

定义 3　谓词节点和语句节点。CFG 中的节点可分为谓词节点 pNode 和语句节点 sNode。其中,顺序流程的代码块构

成语句节点,其出度为 1;谓词节点则由含有判定谓词的语句构成,在程序运行时起到控制转移的作用。

下面分别对这 4 个阶段进行分析。

1)预处理。一般情况下,程序中待保护的关键代码段(以下称为 KeyCode)由用户指定。在预处理阶段,需对 KeyCode 中的标识符进行作用域识别和标记,在随后的多样化处理阶段中仅对作用域在 KeyCode 内部的标识符进行变形。

2)程序多样化处理。一方面,根据程序规模,在 JavaScript 程序体中随机插入多种执行环境监测代码及响应函数,生成初始多样性集合 D_0;另一方面,构建 JavaScript 混淆算法集 ObfLib,并依据混淆算法所作用的对象(如函数、字符串等)对混淆算法进行分类。在此基础上,根据混淆管理算法 ObfExe 生成多样性集合 D'。具体细节将在 3.1 节中进行介绍。

3)路径空间模糊处理。在多样性集 D' 中,首先,获取 KeyCode 的 CFG 并标记 pNode;然后,对 CFG 进行分析和优化,并根据优化结果将选取的 pNode 及其导出的基本块分割并部署在服务器端的运行时数据库中,以供 Web 应用在运行时根据当时的执行路径进行匹配和下载执行;剩余部分经通信代码填充后部署到运行时数据库,构成非关键代码部分 $eCode_i$ 及 KeyCode 的执行时框架 $rSkeleton_i$。

4)程序重构。KeyCode 所在位置以多样化模块及垃圾代码块填充,随后在 KeyCode 调用点前插入对多样化模块的调用语句。其中:多样化模块负责触发程序的时间多样性效果;垃圾代码块则是保留有 KeyCode 入口函数签名的陷阱函数,用来增加静态分析的难度。最后,为确保关键代码段的隐蔽性并减少网络通信负载,保护后的 JavaScript 程序需进行压缩处理。

多样化模块的伪代码如下:

```
If envFlag = true Then
  i = selectTrap();                          //选择某陷阱函数
  funTrap_i();                               //转入陷阱函数执行
Else
  blockCode = request(funCri, dumPar);       //包含伪参数
  eval(blockCode);                           //KeyCode 执行时定义及执行
End If
```

其中 *envFlag* 为执行环境监测结果。若运行环境正常,则多样化模块会向服务器请求 KeyCode 的执行时框架 $rSkeleton_i$,服务器解析请求参数后根据主机访问记录进行派发,浏览器端接收后通过 eval()调用在本地生效。

2.2 被保护应用的执行

Web 应用发布前,服务器端已部署好 JavaScript 代码的多样性集和负责调度的控制逻辑。被保护 Web 应用的执行过程如图 3 所示。

现描述如下:

1)用户请求应用 P 所包含的 JavaScript 程序,服务器端根据访问记录派发程序 $eCode_i$。

2)Web 应用执行 KeyCode 前调用多样化模块。若执行环境正常,服务器将根据访问记录从多样性集中派发 $rSkeleton_i$;否则程序转入预定义的陷阱函数中执行。

3)对获取到的 KeyCode 进行运行时定义及执行,此时垃圾入口函数将被填有功能代码的入口函数覆盖。

4)在 KeyCode 的执行过程中,$rSkeleton_i$ 根据当前执行路径请求隐藏于服务器端的代码块,此时将把真实参数与垃圾参数一起随请求告知服务器中的多样性管理模块。服务器收到请求后进行参数解析,并根据多样性管理模块的处理结果以数据形式派发缺失的代码块。

5)Web 应用接收并动态执行缺失基本块中的代码。

其中,运行时修改程序体利用了 JavaScript 语言的动态特性。JavaScript 中的 eval()函数为程序在运行时将字符串转化为代码执行提供了接口。如图 4 所示。

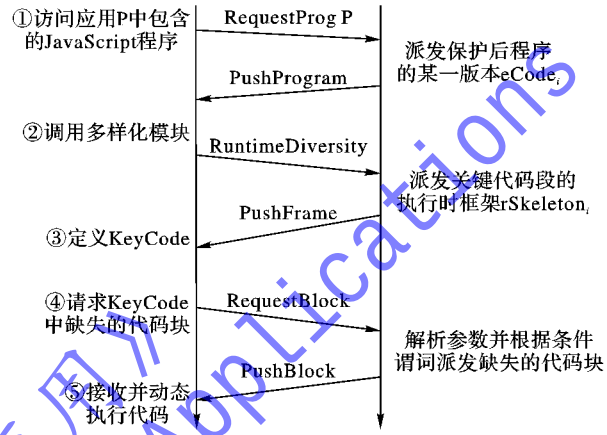

图 3 保护后 JavaScript 程序的执行过程

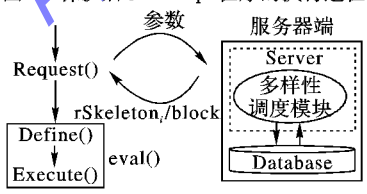

图 4 执行分片代码示意图

3 程序的时间多样性设计与实现

TDJSP 中时间多样性效果的设计主要包含两部分内容:一方面是程序的多样化处理,负责形成代码的多样性集合;另一方面是路径空间模糊化,旨在限制攻击者的单次学习收益,增强时间多样性的保护效果。

3.1 程序多样化处理

攻击者一般在调试器或模拟器中对 Web 应用进行动态分析。JavaScript 程序能够捕捉到这类非正常执行环境所暴露的某些特征。例如:调试窗口检测、调试服务检测、debugger 语句等可用于检测 JavaScript 程序是否处于调试环境;而执行时间校验、模拟执行限制校验等可对模拟执行环境进行检测。

因此,除代码混淆可用于构建程序的多样性集外,JavaScript 程序执行环境监测机制及监测点的不同均会使得程序的多样性集合更加丰富。

在对现有 JavaScript 保护工具,如 JScrambler、Jasob4、Stunnix JavaScript Obfuscator 等使用的混淆算法进行深入分析的基础上,参考应用于源代码混淆的方法[14],根据混淆算法所作用的对象对算法进行分类,构建 JavaScript 混淆算法集 *ObfLib*:

$$ObfLib = \begin{Bmatrix} T_1: f_{11}, f_{12}, \cdots, f_{1x} \\ T_2: f_{21}, f_{22}, \cdots, f_{2y} \\ \vdots \\ T_n: f_{n1}, f_{n2}, \cdots, f_{nz} \end{Bmatrix}; \quad x, y, z, n > 0$$

其中$\langle T_1,T_2,\cdots,T_n\rangle$为 *ObfLib* 中作用于不同程序元素上的混淆算法集。当出现新的 JavaScript 混淆算法时,可对 *ObfLib* 进行更新。现阶段构建的 JavaScript 混淆算法集如表 1 所示,其中包含文件对象模型(Document Object Model, DOM)以及浏览器对象模型(Browser Object Model,BOM)中的相关元素。

表 1　JavaScript 混淆算法集

混淆对象	混淆算法
DOM/BOM 元素	括号记法、对象枚举、自定义编码
字符串	ASCII、Unicode 及十六进制编码、字符串拆分
函数	函数内联、函数外联、函数签名合并
控制流	控制流扁平化、不透明谓词、冗余控制流
标识符	名称混淆
数据	ASCII、Unicode 及十六进制编码
数组	数组排序、数组重构

首先,根据程序规模决定环境监测代码的插入粒度,并由环境监测机制及监测位置作为随机化因子,生成程序的初始多样性集:

$$D_0=\{V_1,V_2,\cdots,V_k\}$$

然后,对 $\forall V_i\in D_0$ 在不改变 KeyCode 入口函数签名的基础上对 KeyCode 进行变形,生成多样性集:

$$V_i'=\{pv_1,pv_2,\cdots,pv_{w_i}\}$$

故多样性集 D' 的规模可经式(1)计算得出:

$$dSize=\sum_{i=1}^{k}w_i \tag{1}$$

算法 ObfExe 使用$\langle T_1,T_2,\cdots\rangle$中的混淆算法对 KeyCode 进行变形。其中:$\langle s_1,s_2,\cdots\rangle$表示待混淆的程序元素集;$T_i$ 表示对待混淆元素 s_i 可供使用的混淆算法集;$prio[s_i]$表示 s_i 的重要程度,其存储在优先级列表 Q 中;*acceptCost* 表示最大性能开销。s_i 的初始值和 *acceptCost* 可设定默认值或由用户直接给出。ObfExe 算法可描述如下:

1)构造算法所需的必要内部数据结构,如函数调用关系图、CFG 等。

2)根据 $prio[s_i]$ 将各待混淆元素 s_i 添加到优先级列表 Q 当中。

3)确定优先级最高的待混淆元素 s_i,然后从 *ObfLib* 中选择相应的混淆算法进行混淆,随后对 s_i 在 Q 中的优先级进行降级处理。反复进行这一过程,直到因混淆导致的性能开销大于预先设定的阈值。

多样性集合 D' 构建完成后,可根据用户需求,利用软件相似性分析技术对集合规模及个体差异度进行控制。

算法 ObfExe 的伪代码如下:

输入:混淆算法集$\langle T_1,T_2,\cdots\rangle$,KeyCode 中待混淆的程序元素$\langle s_1,s_2,\cdots\rangle$,存储 s_i 重要程度的优先级列表 Q,用户定义的最大空间开销 *thresholdSpace*。

输出:KeyCode 的多样性版本。

```
Do buildDescriber;
Do While totalSpace ≤ thresholdSpace
    S ← pObj;          // 选取 Q 中优先级最高的待混淆元素 pObj
    t ← SelecTrans(s,T);
    Apply(t,s);
    Update(Q,s);                  // 对 s 在 Q 中的优先级进行更新
End While
```

3.2　路径空间模糊化

路径空间模糊化以限制攻击者的单次学习收益为主要目的,通过隐藏 KeyCode 中的路径分支信息,对程序的内部逻辑进行保护。

由于控制逻辑隐藏直接导致 Web 应用与服务器端的通信开销,考虑到对 Web 应用运行时性能的影响,在对 KeyCode 进行路径空间模糊化时应尽量减少与服务器间的通信次数。

由谓词节点隐藏导致的时间开销可由式(2)进行描述:

$$t=\sum_{i=1}^{k}(n_i*\delta) \tag{2}$$

其中:k 代表分割的谓词节点数目,n 代表某谓词节点的执行次数,δ 代表一次通信所花费的平均时间。下面根据式(2)具体讨论如何减少由谓词节点隐藏而导致的时间开销。首先引入串联谓词节点的定义。

定义 4　串联谓词节点。指 CFG 中具有串联关系的谓词节点,其特征为某谓词节点是另一谓词节点的直接后继。

对指定关键代码段,隐藏的谓词节点数占谓词节点总数的比例决定了 k 的大小,在此称之为谓词节点的分割粒度。一方面,可通过控制分割粒度减少由谓词节点隐藏而导致的时间开销;另一方面,在进行控制流分割前通过对 CFG 中的串联谓词节点进行合并,可达到相同的目的。

另外,对于指定谓词节点,其执行次数 n 由所在循环深度和循环控制变量共同决定。而循环控制变量具有不确定性,与运行时环境相关。因此在选取谓词节点进行隐藏时可依据循环深度有效降低时间开销。

综上,路径空间模糊化的过程如图 5 所示,主要包含两个步骤。

1)对多样性集 D' 中各程序包含的 KeyCode 进行 CFG 处理。首先,依据基本块标识算法获取 CFG,并标记节点类型。然后按照下述规则化简 CFG 以降低对程序运行时的性能影响:合并环形深度大于 1 的节点,并将其标记为语句节点;对串联谓词节点执行合并操作。

2)控制逻辑隐藏。首先,根据分割粒度隐藏 CFG 中的部分谓词节点 pn_{ij},并以通信代码块填充,得到 KeyCode 的执行框架 $rSkeleton_i$ 以及运行时动态下载的基本块 $B_i=\{b_{i1},b_{i2},\cdots,b_{in}\}$;然后,将它们部署在服务器端的运行时数据库当中;最后,根据裁剪结果部署多样性调度模块,以负责程序运行时的代码分派。

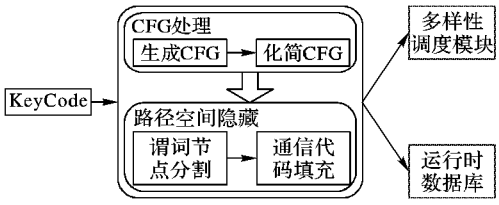

图 5　路径空间模糊化过程

4　分析及实验

4.1　安全分析

首先对保护前后 JavaScript 代码的差异进行分析,如图 6 所示。从静态代码来看,保护后程序的 KeyCode 以垃圾代码填充。从程序的执行流程来看,多样化模块的插入使得每次运行 KeyCode 之前均会触发环境监测代码,若执行环境异常,程序转入陷阱函数中执行;否则将对 KeyCode 进行运行时多样化定义,然后动态执行从服务器端动态下载的代码块。

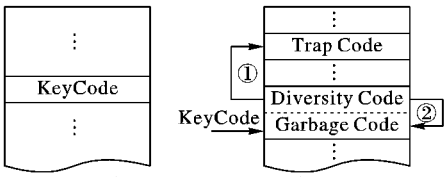

图 6 保护前后程序代码的差异示意

基于上述分析,并围绕逆向分析过程中反复进行的两类知识积累手段:静态分析和动态分析,TDJSP 的抗攻击性主要体现在以下方面:

1)针对静态分析。首先,关键代码段在保留函数签名的条件下以垃圾代码填充,使得静态分析时无法得到真正的功能代码,攻击者难以获取有效信息;其次,路径空间模糊化后,关键代码段的控制流信息被部分隐藏于服务器端,攻击者无法获得对于攻击对象的完整控制权,攻击对象的运行时信息对其不再完全可见。

2)针对动态分析。一方面,程序具备时间多样性的能力,并以路径空间模糊化进一步增强效果,这有效降低了攻击者经累积攻击所得到的学习收益,增加了对原始程序进行还原的难度;另一方面,JavaScript 程序体中散布有多种环境监测代码及陷阱函数以抵御动态分析,同时也确保了时间多样性机制自身的安全。

4.2 性能实验与评估

实验环境 Windows 7 操作系统,3.30 GHz 处理器,4 GB 内存,Chrome 浏览器(版本号 31),Tomcat 服务器(版本号 7.0.52),MySQL 数据库(版本号 5.6.16),网络接入速度 8 Mb/s。

4.2.1 加载时性能分析

本方法对 Web 应用加载时的性能影响主要由保护后 JavaScript 文件大小的增长率决定。

实验过程 在 Chrome 应用商店中选取 Calculator、5InARow、RGB 三款应用进行测试,分别选择其中的四则运算功能块、连线判断功能块以及 RGB 与 Hex 转换功能块作为 KeyCode;然后使用 JScrambler3、Jasob4 以及 TDJSP 保护程序,并记录保护前后 JavaScript 文件的大小。

表 2 保护前后 JavaScript 文件对比

实验对象	初始文件/KB	JScrambler3		Jasob4		TDJSP	
		保护后文件/KB	增长因子	保护后文件/KB	增长因子	保护后文件/KB	增长因子
Calculator	12.0	42.36	3.53	3.72	0.31	39.12	3.26
5InARow	38.0	116.42	3.06	13.30	0.35	93.97	2.47
RGB	1.1	3.58	3.25	0.33	0.30	3.72	3.38

由表 2 可以看出,与商用 JavaScript 混淆器 JScrambler3 相比,TDJSP 并未增加空间开销。但与 Jasob4 相比空间开销略大,这是由于 Jasob4 中混淆算法单一,并以压缩为主要目的所致。

4.2.2 执行时性能分析

该方法对 Web 应用执行时造成的性能影响主要由三方面因素构成,分别是服务器与数据库的交互时间、网络传输时间以及由动态执行 JavaScript 代码所导致的性能损耗。其中,由网络传输引起的执行延迟占绝大部分。

实验 1 由通信导致的性能影响。

为评估实验环境下由单次通信导致的时间延迟,实验中选取具有简单控制结构的测试用例 TestCase 作为实验对象,选取算法 isLeapYear 作为关键代码段。

由于 JavaScript 提供的时间精读为毫秒级,难以获取单次运行 isLeapYear 所引起的时间开销,故实验中采取多次运行得累加值,然后求平均值的方法进行单次数据的采集。具体实验过程如下。

1)生成保护后的程序 Pro_TestCase。其中,算法 isLeapYear 中的串联谓词节点在优化后进行隐藏。

2)随机选取 100 个数据,测试 5 次,分别记录保护前后关键代码段的执行时间 *totalCost* 及 *Pro_totalCost*。

3)由式(3)计算单次通信导致的时间延迟:

$$comCost = \left(\sum_{i=1}^{5} Pro\_totalCost - \sum_{i=1}^{5} totalCost \right) / (5 \times 100) \tag{3}$$

保护前后关键代码段的执行时间如表 3 所示。

表 3 保护前后关键代码段的执行时间 ms

实验对象	序列 1	序列 2	序列 3	序列 4	序列 5
TestCase	38	47	26	38	49
Pro_TestCase	325	330	332	329	326

根据式(3),由表 3 中数据计算得到 *comCost* 为2.89 ms,这对于即时性要求不高的 Web 应用来说其导致的时间延迟是可以接受的。

实验 2 对 Web 应用执行效率的影响。

以 JScrambler 官网提供的 html5_demo 作为实验对象。这是一款基于 canvas 的绘图应用,帧速率设置为 60,运行时间为 10 s,执行结束后显示已绘制的图像数目。实验中选取绘图控制函数作为关键代码段。它负责图像的重绘及运动方向的控制,并对所绘制图像的累计数目进行标记。

实验过程如下:

1)生成经 TDJSP 保护后的程序 Pro_demo;

2)将保护前后的 Web 应用先后运行 5 次并分别记录绘制的图像数目,如表 4 所示。

表 4 保护前后 Web 应用所绘制的图像数

实验对象	测试 1	测试 2	测试 3	测试 4	测试 5
html5_demo	596	599	598	598	597
Pro_demo	521	526	523	525	520

由表 4 中的实验数据可以看出,TDJSP 未对 Web 应用的执行效率造成较大影响,验证了该保护方法的可行性。

5 结语

本文提出了一种具有时间多样性的 JavaScript 代码保护方法以抵御对于 Web 应用的累积攻击。与传统保护方法相比具有以下优势:一方面,JavaScript 程序的时间多样性能力延长了 Web 应用的安全生命周期;另一方面,环境监测代码的插入使 JavaScript 程序具备环境感知的能力,可达到抵御动态分析的目的。通过安全分析和实验评估,该保护方法的有效性和可行性得到验证。下一步工作主要是研究如何量化软件保护的有效性,平衡保护强度和性能影响的关系。

(下转第 82 页)

hu. com/20100318/n270924241. shtml. (搜狐 IT. MySpace 出售用户信息,威胁用户隐私权益[EB/OL]. [2014-03-18]. http://it. sohu. com/20100318/n270924241. shtml.)

[6] InfoWorld. Deleted cloud files can be recovered from smartphones, researchers find [EB/OL]. [2013-03-19]. http://www. infoworld. com/t/mobile-security/deleted-cloud-files-can-be-recovered-smart-phones-researchers-find-214779.

[7] GEAMBASU R, KOHNO T, LEVY A, *et al*. Vanish: increasing data privacy with self-destructing data [C]// SSYM'09: Proceedings of the 18th USENIX Security Symposium. Berkeley: USENIX Association, 2009: 299 – 315.

[8] WANG G, YUE F, LIU Q. A secure self-destructing scheme for electronic data [J]. Journal of Computer and System Sciences, 2013, 79(2): 279 – 290.

[9] XIONG J, YAO Z, MA J, *et al*. A secure self-destruction scheme with IBE for the Internet content privacy [J]. Chinese Journal of Computers, 2014, 37(1): 139 – 150. (熊金波, 姚志强, 马建峰, 等. 面向网络内容隐私的基于身份加密的安全自毁方案[J]. 计算机学报, 2014, 37(1): 139 – 150.)

[10] TANG Y, AMES P, BHAMIDIPATI S, *et al*. CleanOS: limiting mobile data exposure with idle eviction [C]// OSDI'12: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation. Berkeley: USENIX Association: 2012: 77 – 91.

[11] DAI J, ZHANG C, LI Z. A cryptographic and time-constraint access control scheme for mobile terminals [M]// Network Computing and Information Security. Berlin: Springer, 2012: 352 – 357.

[12] TUNG T, LIN L, LEE D. Pandora messaging: an enhanced self-message-destructing secure instant messaging architecture for mobile devices [C]// WAINA 2012: Proceedings of the 26th International Conference on Advanced Information Networking and Applications Workshops. Piscataway: IEEE, 2012: 720 – 725.

[13] SHAMIR A. How to share a secret [J]. Communications of the ACM, 1979, 22(11): 612 – 613.

[14] WELCH T A. A technique for high-performance data compression [J]. IEEE Computer, 1984, 17(6): 8 – 18.

[15] WANG L, REN Z, YU R, *et al*. A data assured deletion approach adapted for cloud storage [J]. Acta Electronica Sinica, 2012, 40(2): 266 – 272. (王丽娜, 任正伟, 余荣威, 等. 一种适于云存储的数据确定性删除方法[J]. 电子学报, 2012, 40(2): 266 – 272.)

[16] LIU J, LI W, SUN Y. Security issues and solutions on social networks [J]. Journal of University of Science and Technology of China, 2011, 41(7): 565 – 575. (刘建伟, 李为宇, 孙钰. 社交网络安全问题及其解决方案[J]. 中国科学技术大学学报, 2011, 41(7): 565 – 575.)

[17] BWACH A, GARTRELL M, HAN R. Social-K: real-time K-anonymity guarantees for social network applications [C]// PerCom 2010: Proceedings of the 8th Annual IEEE International Conference on Pervasive Computing and Communications. Piscataway: IEEE, 2010: 600 – 606.

[18] BWACH A, GARTRELL M, HAN R. Solutions to security and privacy issues in mobile social networking [C]// CSE'09: Proceedings of the 2009 International Conference on Computational Science and Engineering. Piscataway: IEEE, 2009: 1036 – 1042.

[19] ZHANG T, CHEN T. An energy-efficient cloud-based offloading decision algorithm for mobile devices [J]. SCIENTIA SINICA: Informationis, 2012, 42(3): 333 – 342. (章铁飞, 陈天洲. 基于移动设备云迁移的节能决策算法[J]. 中国科学: 信息科学, 2012, 42(3): 333 – 342.)

[20] XIONG J, YAO Z, MA J, *et al*. A secure self-destruction scheme for composite documents with attribute based encryption [J]. Acta Electronica Sinica, 2014, 42(2): 366 – 376. (熊金波, 姚志强, 马建峰, 等. 基于属性加密的组合文档安全自毁方案[J]. 电子学报, 2014, 42(2): 366 – 376.)

(上接第 76 页)

参考文献:

[1] FELMETSGER V, CAVEDON L, KRUEGEL C, *et al*. Toward automated detection of logic vulnerabilities in Web applications [C]// Proceedings of the 19th USENIX Security Symposium. Berkeley: USENIX Association, 2010: 143 – 160.

[2] XU W, ZHANG F, ZHU S. JSTill: mostly static detection of obfuscated malicious JavaScript code [C]// Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy. New York: ACM, 2013: 117 – 128.

[3] XU W, ZHANG F, ZHU S. The power of obfuscation techniques in malicious JavaScript code: a measurement study [C]// Proceedings of the 7th International Conference on Malicious and Unwanted Software. Piscataway: IEEE, 2012: 9 – 16.

[4] COHEN F B. Operating system protection through program evolution [J]. Computers & Security, 1993, 12(6): 565 – 584.

[5] CHEN P-Y, KATARIA G, KRISHNAN R. Software diversity for information security [EB/OL]. [2014-06-06]. http://infosecon.net/workshop/pdf/47. pdf.

[6] XIN Z. Study on program diversity for software security [D]. Nanjing: Nanjing University, 2013. (辛知. 程序多样性技术研究[D]. 南京: 南京大学, 2013.)

[7] COLLBERG C, NAGRA J. Surreptitious software: obfuscation, watermarking, and tamperproofing for software protection [M]. Boston: Addison-Wesley, 2009: 203 – 209.

[8] COLLBERG C, MARTIN S, MYERS J, *et al*. Distributed application tamper detection via continuous software updates [C]// Proceedings of the 28th Annual Computer Security Applications Conference. New York: ACM, 2012: 319 – 328.

[9] BERTHOLON B, VARRETTE S, BOUVRY P. JShadObf: a JavaScript obfuscator based on multi-objective optimization algorithms [M]. Berlin: Springer, 2013: 336 – 349.

[10] COLLBERG C. The case for dynamic digital asset protection techniques [EB/OL]. [2014-06-15]. http://www. irdeto-prod. com/documents/AC_Dynamic_vs_Static_Security_Prof_Collberg. pdf.

[11] COLLBERG C, DAVIDSON J W, GIACOBAZZI R, *et al*. Toward digital asset protection [J]. IEEE Intelligent Systems, 2011, 26(6): 8 – 13.

[12] ANCKAERT. Diversity for software protection [D]. Ghent: Ghent University, 2008.

[13] O'DONNELL A J, SETHU H. Software diversity as a defense against viral propagation: models and simulations [C]// Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation. Washington, DC: IEEE Computer Society, 2005: 247 – 253.

[14] COLLBERG C, THOMBORSON C, LOW D. A taxonomy of obfuscating transformations [R]. Auckland: University of Auckland, 1997.