基于模式生成的浏览器模糊测试技术（软件安全漏洞检测）*

霍玮[1,2]，戴戈[1,2]，史记[1,2]，龚晓锐[1,2]，贾晓启[1,2]，宋振宇[1]，刘宝旭[1,2]，邹维[1,2]

[1](中国科学院信息工程研究所 中国科学院网络测评技术重点实验室 网络安全防护技术北京市重点实验室,北京 100195)

[2](中国科学院大学 网络空间安全学院,北京 100049)

通讯作者：史记,E-mail:shiji@iie.ac.cn

摘 要：模糊测试被广泛应用于浏览器的漏洞挖掘，其效果好坏的决定因素之一是测试者编写的测试模式。针对特定测试模式实现成本高、生存时间短等问题，本文提出了一种基于模式生成的浏览器模糊测试器自动构造方法，通过解析已知漏洞触发样本，自动提取测试模式，对模式中每个模块应用传统的变异策略，完成畸形样本的自动生成。实验表明，针对 5 款浏览器的 1089 个已知漏洞触发样本，平均仅用时 11.168 秒即可完成 1089 个不同模糊测试器的自动构建，远低于人为编写的时间消耗；随机选取其中 10 个模糊测试器分别对 IE 10、IE 11、Firefox 54.0 的全补丁版本进行测试，共产生 57 个不同的崩溃样本，发现 1 个高危未知漏洞，证明本方法具有较好的未知漏洞发现能力。

关键词：模糊测试；漏洞挖掘；浏览器；模式。

BROWSER FUZZING BASED ON PATTERN-GENERATION

HUO Wei[1,2], DAI Ge[1,2], SHI Ji[1,2], GONG Xiao-rui[1,2], JIA Xiao-qi[1,2], SONG Zhen-yu[1], LIU Bao-xu[1,2], ZOU Wei[1,2]

[1]（Institute of Information Engineering, Key Laboratory of Network Assessment Technology, Chinese Academy of Science, Beijing Key Laboratory of Network Security and Protection Technology, Beijing 100093, China)

[2]（School Of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Fuzzing is widely used for browser vulnerability mining, and one of the key factors determining its effectiveness is the test pattern written by the tester. While the test pattern is written with high cost and short survival time, in this article, a automatic construction of fuzzy tester based on Pattern-Generation is presented. By analyzing the known vulnerability samples, then extract the test pattern automatically and then the traditional mutation strategy will be applied to each module in the pattern to complete the automatic generation of the abnormal samples. Experiments show that averaging only 11.168 seconds to finish the automatic construction of 1089 different fuzzy testers based on 1089 known vulnerabilities for five browsers, which much lower than the time-cunsumption by testers themselves. Randomly selected 10 fuzzy testers and applies on IE 10, IE11 and Firefox 54.0 web browser, it

*本论文获得中国科学院网络测评技术重点实验室和网络安全防护技术北京市重点实验室资助.获得了北京市科委(D161100001216001、Z161100002616032), 国家自然基金(61572481、61602470)课题资助.

收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00

CNKI 在线出版时间: 0000-00-00

discovered a total of 57 different bugs, including a high-risk unknown vulnerability, proves that this method has better ability to find the unknown vulverabiliy.

Key Words: Fuzzing; Vulnerability Mining; browser; pattern

引言

随着互联网技术的发展与普及，已经越来越多的人加入互联网的行列。据《每日邮报》报道[5]，截至 2016 年底，全球网民数量已达到 35 亿，约占总人数的一半。而浏览器作为互联网的主要入口，几乎所有的网络行为都离不开浏览器。

正因为浏览器的广泛使用，使得浏览器所引发的安全问题危害巨大。据新华网 2010 年对 IE 浏览器极光漏洞的报道[6]，金山网盾共拦截数百万次通过 IE 极光零日漏洞进行的攻击，实际潜在受影响用户保守估计在千万级。据安全机构 Flexera Software 统计[1]，在 2016 年主流浏览器漏洞数量达到 713 个，这些漏洞以高危漏洞为主，意味着一旦被利用，将造成严重的损失。

浏览器漏洞挖掘是一个经久不衰的话题被广泛的关注。不同于其它软件漏洞挖掘，浏览器经过 20 余年的发展，其代码和结构已非常完善，目前浏览器的漏洞挖掘难度也越来越高。首先大量漏洞的修补，使得传统的漏洞类型已很少出现；其次浏览器厂商对其产品的大规模测试，使得一般的测试方法很难再产生较好的测试效果；再次漏洞奖励计划的实施，让大量的安全研究者加入到浏览器漏洞挖掘中，使得浏览器漏洞挖掘出现“僧多粥少”的局面。

如今对浏览器漏洞挖掘的主要方法是模糊测试(Fuzzing)[2]，其基本的思想是给目标软件提供大量特殊构造的数据作为输入，并同时监控目标软件运行过程中触发的异常，辅之以对异常信息的人为分析，从而确认目标软件中存在的漏洞。然而当前对浏览器进行模糊测试的效果好坏几乎都依赖于测试者编写的模糊测试器，而浏览器模糊测试器的“骨架”是对网页元素处理的流程，本文称之为测试模式。不同于其它软件的安全测试，浏览器的模糊测试需不断的修改测试模式来保证持续的测试效果，而由于浏览器组成的复杂性及安全机制的完善性，测试模式的设计与调整对测试者的要求很高且实现复杂，致使测试模式的产生效率很低。

针对上述问题，本文提出一种基于模式生成的浏览器模糊测试方法，通过对测试模式组成的分析与形式化定义，实现从已有的网页样本中测试模式的自动化提取，基于该测试模式自动产生模糊测试器，并完成测试数据构造。本文的主要贡献如下：

（1）依据浏览器漏洞的类型与特征对浏览器模糊测试的测试模式进行模块抽象，并使用 BNF 范式对其进行规范化定义，作为不同模糊测试器的识别标准；

（2）提出了一种测试模式自动提取方法及测试器自动构建方法，使用不同的漏洞触发样本作为输入，自动生成不同的测试模式，并通过样本生成器产生大量的测试样本，形成一个新的模糊测试器；

（3）实现了一个浏览器模糊测试工具 autofuzzy，能够用于实际的浏览器模糊测试；通过实验验证了本文提出的方法能够快速产生有效的测试模式和在零日漏洞的挖掘方面的有效性。

本文将分以下六个部分进行介绍，第一节介绍模糊测试的相关研究进展，第二节通过对现有浏览器模糊测试方法的分析，提出了本文的测试方法；第三节阐述测试模式的提取方法；第四节描述样本生成器构造方法及运行原理；第五节实验与结果分析；第六节总结全文并展望未来的工作。

1 相关研究

浏览器作为一种组成复杂的软件，针对其不同结构的漏洞挖掘需采用不同的模糊测试方法[3,4]。

[5] http://www.dailymail.co.uk/sciencetech/article-3960180/Almost-half-world-online-end-2016-poorer-countries-lag-report-shows.html
[6] http://news.xinhuanet.com/tech/2010-03/10/content_13141841.htm

针对于浏览器中所依赖的第三方库，主要使用基于变异的模糊测试方法。该方法基于已有的输入样本（种子）使用随机数据变异的方式产生测试数据。Lcamtuf 提出了一个使用覆盖率指导数据变异的模糊测试工具 afl(american fuzzy lop)[13]，被认为是最有效的模糊测试器之一。通过在测试过程中实时统计所产生的覆盖路径来指导数据的变异策略，提高了代码覆盖率，发现了上百个以上浏览器所使用图形库和字体库的漏洞；LibFuzzer[14]关注于 Chrome 浏览器中使用的开源库的测试，该方法通过对接口函数中的参数进行变异以完成对给定函数的模糊测试，发现了如 Freetype，Ffmpeg 等开源库的漏洞。

针对浏览器的 ActiveX 控件以及 JavaScript 解释引擎的漏洞挖掘，也是研究的热点。Dormann 等[5]提出了一种检测浏览器 ActiveX 控件安全漏洞的方法，通过使用 Dranzer 自动提取并调用特定 ActiveX 接口函数完成测试；Hodován 等[6]通过构建一种图结构来描述 JavaScript 引擎中的类型信息，实现对 JavaScript 引擎 API 的测试，该方法相比于已有的方法在覆盖率上有了大规模的提升；Jesse Ruderman[7]实现了一个对 JavaScript 引擎测试的工具 jsfunfuzz，通过随机构造大量的 JavaScript 语句操作来进行测试，发现了 280 个 Firefox 浏览器 JavaScript 引擎[8]的漏洞。

针对于浏览器渲染引擎的测试是当前最热门的研究点之一，同时也是本文的关注点。X-Fuzzer[15]是由 Vinay Katoch 开发的一款轻量级的浏览器模糊测试工具，通过对设置好的 HTML 标签属性进行随机赋值，以实现对浏览器的测试；Google 安全研究人员 Michal Zalewski 开发的 cross_fuzz[17]，其思想对浏览器模糊测试产生了深远的影响，基于该方法以及其衍生的其他浏览器模糊测试方法发现了上百个以上的浏览器安全漏洞。该方法采用递归式的浏览器测试方式，首先生成浏览器 DOM 元素，然后随机产生冗长的 DOM 操作序列对 DOM 进行操作并以新生成的 DOM 元素作为下一轮操作的对象，从而完成递归测试；YD Lin[9]等通过对相关的浏览器模糊测试工具和框架的整合，提出一种新的测试数据生成方法，通过动态挑选种子样本文件以及调整基础数据的变异方式来生成样本，从而实现对浏览器渲染引擎[10]的测试；nduja[12]和 fileja[13]是 Valotta R 基于 grinder[11]实现的一个测试数据生成的文件，是近年来引领性的研究成果。其中 nduja 关注于对 DOM 的测试，它制定了一系列的新建、修改、删除 DOM 元素的规则，通过随机选择的方式产生不同的测试样本来对浏览器进行测试；而 fileja 则提出了两个新维度的测试，时间维度引入时间的依赖关系以发现条件竞争等漏洞；空间维度上考虑新版本的向下兼容所出现的错误，作者使用该方法在为期 5 个月的测试中发现 15 个浏览器安全漏洞。

对浏览器所依赖开源库使用的变异的模糊测试方法，尽管在对开源的图形库，字体库的测试中表现良好，但在对输入有严格语法和语义规范的目标来说，例如浏览器的渲染引擎，JavaScript 引擎，随机变异所产生的数据基本上在一开始就会被摒弃，从而无法达到预期的测试效果。而如今对浏览器渲染引擎的测试，需要测试者对目标本身以及目标所接受的输入数据规范有一定的了解，因此测试的成本和起点很高；并且单个测试工具（方法）的测试模式单一且有很强的时效性，在经过一段时间的测试之后基本不再具备漏洞发现的能力。经过对 nduja 和 fileja 的分析发现，其中 nduja 的核心代码有 1307 行，fileja 的核心代码有 2970 行，且完全由测试者手工构造；并且现在将其直接用于浏览器模糊测试也已不能产生好的测试效果[21]。

6 方法概述

针对浏览器渲染引擎的模糊测试器，核心在于针对不同的漏洞模式和漏洞可能存在的问题点，构造不同的测试样本，测试样本的准确程度和时效性对测试效果的好坏有决定性作用。一个测试样本通常由元素构建、事件监听、方法操作、属性操作等主要操作类组成，每一个操作类中随机选择具体的操作，从而生成测试样本。本文将测试样本中操作类的选择及组合顺序称之为测试模式，在测试模式中决定了该选取哪类操作以及该如何组合，构成了测试样本的“骨架”。

2.1 测试模式实例分析

下面以 X-Fuzzer[15]、Ndjua[12]这两款代表性的浏览器模糊测试器为例，分析其构建原理，总结出测试模式的概念与作用。

X-Fuzzer 主要针对于 HTML 标签和属性进行测试。通过对其源代码进行分析后，归纳出该工具的主要测试流程如图 2-1 所示。X-Fuzzer 的测试流程从整体上看是由创建元素节点对象、添加元素节点、Outer-innerHTML、删除对象等 4 类操作按照一定的组合顺序构成，最后生成一个 HTML 样本来对浏览器进行测试。

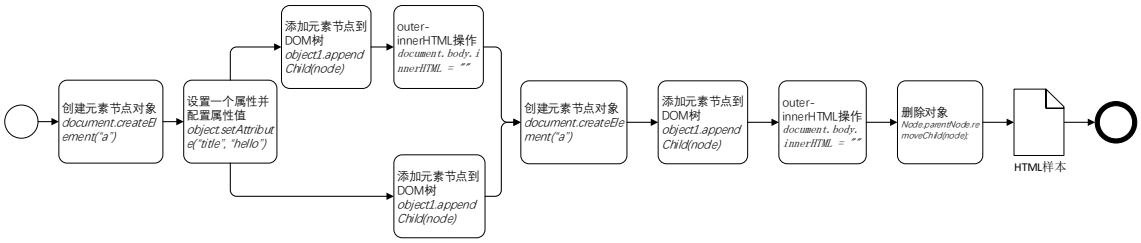

图 2-1 X-Fuzzer 测试流程

Ndjua 是一种针对于浏览器 DOM 结构进行模糊测试的方法，并基于 Grinder 实现。通过对其分析，总结其测试流程如图 2-2 所示。从整体上看 Ndjua 的测试模式是以创建元素节点、添加元素节点到 DOM 树中、添加事件监听、设置属性和属性值、特殊元素，对象的操作等 5 类操作按照一定的组合顺序构成，最后生成一个 HTML 样本完成对浏览器的测试。

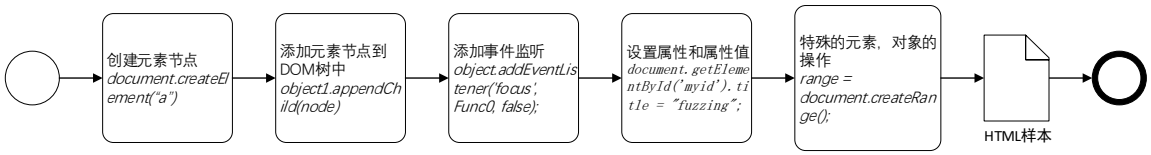

图 2-2 Ndjua 测试流程

2.2 基本概念

从 2.1 节的分析中可以得出，测试者在对浏览器进行模糊测试时，测试流程的组成和所涉及的操作存在相似之处，根据测试者对浏览器运行原理的分析和理解，从 HTML 网页的全部组成操作类中，如创建元素对象、添加事件监听、属性的设置等，挑选若干个类作为构建的基本组成，然后使用若干种排列方法进行组合，这些操作类和排列方法构成一个完整的测试流程。

为了下文描述方便，本文给出三个基础定义：

模块 HTML 网页构建中的 1 个操作类，称为 1 个模块。模块包括创建元素对象、方法设置、属性设置等类别，是测试模式中的最小组成结构。

逻辑 模块的排列顺序。例如给定了模块 M_1、M_2、M_3，其中一种排列组合，如 $M_1M_2M_3$ 或者 $M_3M_2M_1$ 即分别为一种逻辑。

测试模式 指定的模块按照特定的逻辑构成的测试流程。

2.3 方法介绍

为了能够低成本、高时效的构建一个测试模式，本文提出了一种自动生成测试模式的方法，并基于该模式产生测试样本。该方法首先分析已知漏洞触发样本，通过模块识别和逻辑确定两步，完成测试模式的自动提取，然后利用所设计的可以解析测试模式并变异基本操作的样本生成器，在基础数据集的支持下产生大量新的测试样本，完成模糊测试，主要流程如图 2-3 所示。下文针对测试模式提取及样本生成 2 个主要步骤进行具体介绍。

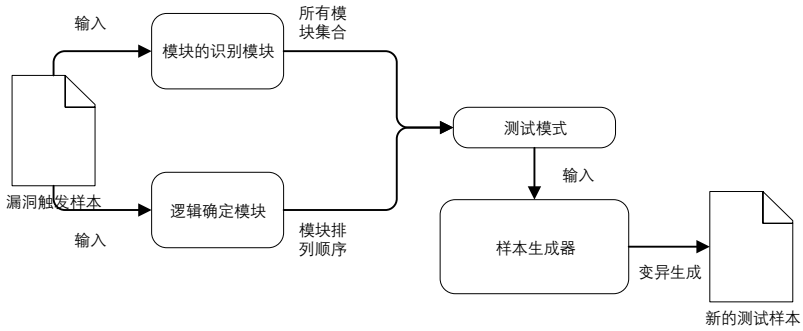

图 2-3 方法流程图

7 测试模式提取

从样本文件中提取测试模式的关键在于对模块的识别以及对逻辑的确定。本节将对模块进行划分与抽象后，对样本文件中的模块进行静态识别；并通过动态执行的方式获取各模块之间的依赖以确定逻辑。最后依据识别的模块以及模块之间的依赖关系形成测试模式。

3.1 模块划分与抽象

一个浏览器模糊测试样本中一般存在上千个 HTML 元素构建、CSS 样式操作以及 JavaScript 语句操作，如果将每一项操作或者每一条语句作为模块的话，将会导致模块数量过多，造成模块描述困难，测试模式的产生效率过低；若对模块的区分过于松散，将会导致失去原始样本的漏洞触发特性，难以得到预期的测试效果。因此，需要在合理的粒度下对浏览器所能够支持的操作进行模块抽象，使测试模式产生更加适合与高效。

模块抽象需要考虑以下 2 个因素：

（1）基于软件缺陷产生，构造测试模式对目标进行测试的核心目标在于发现软件缺陷，因此模块抽象必须以此目标作为前提。需要考虑的包括软件缺陷的产生模式以及各个组成部分等。

（2）基于测试范围，需要保证浏览器的核心部分都能够被测试，如 X-Fuzzer 仅关注于 HTML 标签的测试，然而对于浏览器来说还有 CSS 样式的测试以及 DOM 树的测试等。

通过对典型浏览器模糊测试方法的分析，包括如 Pair-Fuzz[16]、cross_fuzz[17]、chromefuzzer[18]等，结合对浏览器的历史漏洞样本分析之后，总结出浏览器在处理过程中的核心操作、易出现安全问题的部分以及对出现软件缺陷存在影响的部分。主要分为 3 方面操作：HTML 标签对象的操作、CSS 样式的操作以及 DOM 树的操作。

4) HTML 标签对象的操作

HTML 标签对象的操作主要测试由于 HTML 标签的操作所导致的浏览器处理错误。分为 HTML 标签元素的静态创建、HTML 对象的动态创建、HTML 对象的释放以及特殊标签元素的创建，共计 4 个模块。

（1)HTML 标签元素的静态创建是基于 HTML 语言语法对元素进行创建。在测试样本加载之后，浏览器的 HTML 解析器将会对这部分创建进行解析，包括用于表格的标签 table，用于超链接的标签 a 等。

（2）HTML 对象的动态创建是基于 JavaScript 语法对 HTML 对象进行创建。基于 JavaScript 引擎对 JavaScript 代码进行解析后，将依据指定的类型动态分配一块内存区域存储创建好的 HTML 对象。

（3）HTML 对象的释放是基于 JavaScript 语法对指定的对象进行释放。该模块普遍出现于释放后重用漏洞（UAF）以及双重释放漏洞（Double Free）样本之中。

（4）特殊标签元素的创建。基于对历史漏洞的分析后发现，存在复杂的操作逻辑的特殊标签元素，也是漏洞出现的集中点，如 range、form、iframe 等，因此这部分需要重点考虑。

具体描述如表 3-1 所示：

表 3-1 HTML 标签对象操作的模块抽象

模块类别	实际表示举例	模块抽象表示
HTML 标签元素的静态创建	<table height=100><td>Hello World</td></table>	static_html
	<form action=“/s”> <input type=“hidden” value=“1”></form>	
HTML 对象的动态创建	document.createElement(“table”);	generate_node
	document.createElementNS("http://www.w3.org/1999/xhtml","div");	
HTML 对象的释放	document.body.innerHTML = "";	out_innerHTML
	document.documentElement.outerHTML = "test";	
	table.innerText = "1";	
特殊标签元素的创建	range, form, iframe, textbox 等	special_element
	range = document.createRange();	
	range.setStart(element0,0);	
	<iframe id=“1”> </iframe>	

5) CSS 样式的操作

CSS 样式的操作用于针对性发现浏览器处理样式时所存在的缺陷，分为 CSS 样式的初始化设置和对 CSS 样式的动态操作 2 个模块。CSS 样式的初始化设置是基于 CSS 语法来进行样式设定，主要关注于测试浏览器的 CSS 解析引擎;CSS 样式的动态操作是指依赖于 JavaScript 代码动态对 CSS 样式所做的的设定、修改和删除等操作。

具体描述如表 3-2 所示：

表 3-2 CSS 样式操作的模块抽象

模块类别	实际表示举例	模块抽象表示
CSS 样式初始化设置	h1 {color:red; font-size:14px;}	css_style
	p { color: rgb(100%,0%,0%); }	
	td, ul, ol, ul, li, dl, dt, dd { font-family: Verdana, sans-serif; }	
CSS 样式动态操作	document.body.style.backgroundPositionX="right";	style_operate
	document.getElementById("p1").style.borderRight="thick solid #0000FF";	
	document.getElementById('myTable').style.captionSide="right"	

6) DOM 树的操作

DOM（Document Object Model）[19]，称为文档对象模型，HTML DOM 定义了所有 HTML 元素的对象和属性以及通过 JavaScript 访问它们的方法。

浏览器在加载页面时会将页面中元素、属性、文本等解析为一个树形结构，即 DOM 树。而页面中的元素、属性、文本等则作为 DOM 树中树节点存在。JavaScript 对页面的操作则是基于 DOM 树结构，因此对 DOM 树的操作是浏览器的核心部分，也是漏洞频发的位置。

DOM 树的操作是一个很复杂的过程，本文将对 DOM 树的操作分为 DOM 树的修改、属性和属性值的设置、方法的设置以及事件的监听 4 个模块。

（1）DOM 树的修改，基于 HTML DOM 提供给 JavaScript 的接口方法，能够在 DOM 树中添加节点、删除节点、替换节点等。旨在通过对 DOM 树结构的改变来触发浏览器对 DOM 树处理的错误。

（2）属性和属性值的设置，通过 DOM 树支持的属性节点操作，设置属性来添加属性节点，修改属性值来改变属性节点中的内容等来进行测试。

（3）方法的设置，针对不同的元素对象和节点对象，HTML DOM 提供了上百个不同的操作方法，如对于 MediaRecorder 对象，HTML DOM 提供 start 方法用于开始记录，提供 stop 方法用于终止记录等。这些方法在执行过程中也会间接或直接对 DOM 树进行操作，从而有可能引发处理的错误。

（4）事件的监听，通过对 HTML DOM 支持的事件接口注册事件，实现对浏览器事件机制的测试。事件响应的顺序对 DOM 树的操作产生影响。

具体描述如表 3-3 所示：

表 3-3 DOM 树操作的模块抽象

模块类别	实际表示举例	模块抽象表示
DOM 树的修改	document.getElementById("myList1").appendChild(node);	dom_operate
	list.removeChild(list.childNodes[0]);	
属性和属性值的设置	document.getElementsByTagName("INPUT")[0].setAttribute("type","button");	property_operate
	document.getElementsByTagName("H1")[0].removeAttribute("style");	
	document.getElementById('myid').title = "fuzzing";	
方法的设置	object.show()	method_set
	object.hide()	
	document.getElementById('myAnchor').focus()	
事件的监听	object.addEventListener(func_name, event)	event_listener
	name1.attachEvent('onclick',function () { info.innerHTML += "red" + " "; });	

3.2 模块识别

在给定一个 HTML 网页形式的漏洞触发样本后，模块识别算法将逐行处理，输出其中存在的模块，具体处理流程如算法 1 所示。

算法 1 模块识别算法

Algorithm 1
Input: sample: Sample After Normalize
Output: ModelList: Model List

```
1          HTMLLines := NULL
```

```
2          CSSLines := NULL
3          JSLines := NULL
4          Lines := getLinesFromSample(sample)
5          for each line in Lines:
6             if isLineBelongHTML(line) then:
7                  HTMLLines.append(line)
8             else if isLineBelongCSS(line) then:
9                  CSSLines.append(line)
10            else if isLineBelongJS(line) then:
11                 JSLines.append(line)
12         ModelList := NULL
13         for each line in HTMLLines:
14            tags, property, event := getTagsProEvent(line)
15            if tags != NULL then:
16                 ModelList.append(Mapping(tags, property, event))
17         for each line in CSSLines:
18            selector, property := getSelectorProperty(line)
19            if selector != NULL then:
20                 ModelList.append(Mapping(selector, property))
21         for each line in JSLines:
22            for function ModelFeature in ModelFeatureList:
23                 model := ModelFeature(line)
24                 if model != NULL then:
25                      ModelList.append(model)
26                      break
27         return ModelList
```

1-4 行，定义了三个变量，分别用于记录输入样本中的 HTML、CSS、JavaScript 部分；并从输入样本中按照每行为单位读入。

5-11 行，依次访问每一行代码，依据相应的特征将其分配到 HTML、CSS、JavaScript 三个类别中，其余成分不进行分析。

13-16 行，针对 HTML 类别中的每一行代码，识别静态 HTML 代码中标签名、属性、以及注册事件等模块，识别函数为 getTagsProEvent。

17-20 行，针对 CSS 类别中的每一行代码，识别出 CSS 样式的选择器，属性类型等模块，识别函数为 getSelectorProperty。

21-26 行，针对 JavaScript 类别中的每一行代码，根据识别函数确定其所属模块，识别函数为 ModelFeatureList。

本文使用 BNF 格式[7]对模块特征进行规范化定义，并作为 getTagsProEvent，getSelectorProperty 和 ModelFeatureList 等核心识别函数实现的依据。BNF 规范是推导规则的集合，表示为 <符号> := <表达式>，符号是非终结符，表达式则是一个符号序列用于对左侧符号的描述。本文所使用的基本语法符号如表 3-4 所示：

表 3-4 BNF 符号介绍

符号	含义	符号	含义
<>	包含的内容表示必选项	::=	表示“被定义为”

[7] BNF(Backus-Naur Form)[20]，巴克斯范式，是一种用于表示上下文无关文法的语言，被广泛使用于程序设计语言、指令集等的语法表示中。

[]	里面的内容表示可选项	"…"	双引号，表示特定的术语（关键字）
{}	里面内容表示重复项，0 次或者多次	()	分组，里面内容表示一组
|	并列选项，仅能选择一项		

依据 BNF 的基础符号规范，对抽象后的模块描述如表 3-5 所示：

表 3-5 抽象的模块与 BNF 描述

抽象的模块名称	BNF 描述	备注
static_html	static_html ::= "<" <html_tag> {" " ((<property> "=" <value>) | (<event> "=" <function_name>))} ">" [<content>] ["</" <html_tag> ">"]	静态 HTML 标签的构建，依据 HTML 语法构建
css_style	css_style ::= <selector> "{" <property> ":" <value> ";" { <property> ":" <value> ";" } "}"	CSS 样式构建
generate_node	generate_node ::= [variable_declaration "="] "document" "." ("createElement" "(" <html_tag> ")") | (("createElementNS" "(" <namespace_url> ", " html_tag ")"))	元素对象的创建
dom_operate	dom_operate ::= [variable_declaration "="] <node1> "." ((("appendChild" | "applyElement") "(" <node2> ")")| (("insertBefore" | "replaceChild") "(" <node2> "," <node3> ")") | (("cloneNode" | "removeChild") "(" [<boolean>] ")"))	DOM 树修改操作的设置
property_operate	property_operate ::= <node1> "." (("setAttribute" "(" <property> "," <value> ")") | ("removeAttribute" "(" <property> ")") | ("setAttributeNode" "(" <attributenode> ")"))	对象属性及属性值的设置
method_set	method_set ::= <node1> "." <method> "(" [(<arg> {"," <arg>})] ")"	对象方法的设置
event_listener	event_listener ::= <node1> "." (("addEventListener" "(" <event>"," <function_name>"," <boolean> ")") | ("attachEvent" "(" <event>"," <function_name> ")") | (("attachEvent" | "setTimeout") "(" <function_name>, <time> ")"))	事件的监听
style_operate	style_operate ::= <node1> "." "style." <property> "=" <value>	样式的操作
out_innerHTML	out_innerHTML ::= (("eval" "(" <content> ")") | ("document" "." ("write" | "writeln") "(" <content> ")") | (<node1> "." ("innerHTML" | "outerHTML" | "innerText" | "outerText") "=" <content>))	元素对象的删除操作

注：关键类型说明，html_tag 表示 html 的标签元素；property 表示节点对象所对应的属性；value 表示属性所对应的属性值；event 表示支持的事件类型；function_name 表示注册事件的回调函数名称；content 表示随机的内容；selector 表示 CSS 的选择器；nodeX 表示节点对象；method 表示所支持的方法类型；arg 表示对应方法的参数。特殊元素的描述依据具体需求，故此处未给出相应的 BNF 描述。

3.3 逻辑确定

逻辑的确定关键在于识别出各模块之间的依赖关系。对于一个 HTML 样本来说，CSS 和 HTML 代码属于一开始加载的，而 JavaScript 的执行才包含执行的顺序，因此逻辑的确定主要关注于 JavaScript 代码。经过规范后的样本[8]，其 JavaScript 代码完全处于<script>标签内，因此这将是我们进行动态逻辑识别的前提。识别依据代码插桩的思想，在关键位置设置标识，本文中设置的为<script>标签位置以及 function 函数位置；然后实际运行修改后的样本，通过标识位置从而确认执行的顺序。具体流程图如图 3-1 所示：

[8] https://github.com/icepng/html-normalize

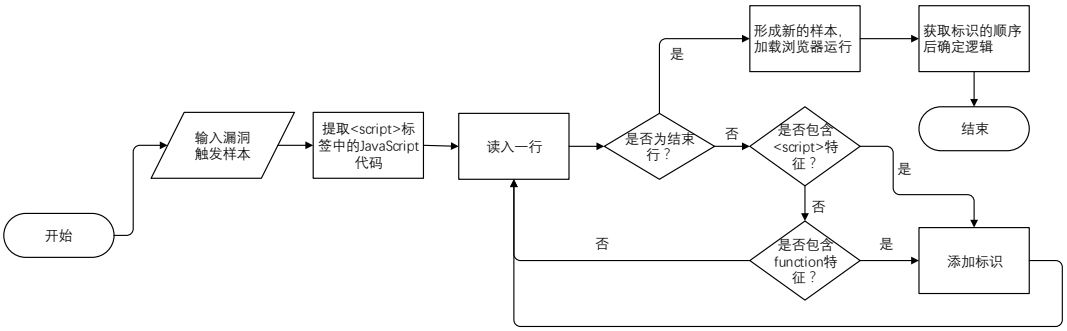

图 3-1 逻辑获取流程

8 样本生成器

样本生成器用于依据获取到的测试模式来构造大量的测试样本，主要包含两个部分，第一部分是人工构造的一个基础的数据集，第二部分是构造出测试样本。

4.1 数据集

针对于每一个模块，人工构造了一个与之对应的数据集，包含该模块所对应的操作集合。这将是样本生成的基础数据来源。

4.2 测试样本生成

测试样本生成以数据集为基础，当输入一个测试模式之后，将按照测试模式的模块组成和逻辑顺序，产生新的测试样本。流程如图 4-1 所示：

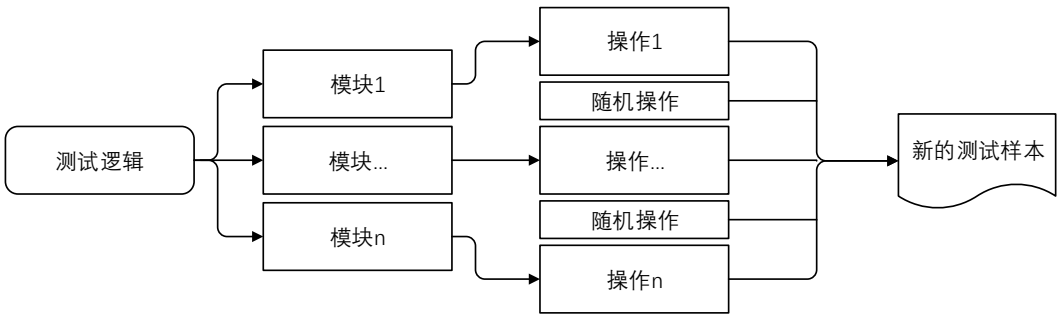

图 4-1 测试样本生成流程

当输入一个测试模式之后，样本生成器将把测试模式依据模块之间的依赖将其划分为模块序列，每一个模块将依据相应的模块数据集随机获取相应的操作，并在各个模块之间填充随机操作来优化测试效果。最终组合所有操作生成新的测试样本。

9 实验与结果分析

针对于上述提出的模糊测试方法，本文设计并实现了一个全新的浏览器模糊测试器——autofuzzy，并通过相关实验验证本文提出方法的有效性。

5.1 实验环境

本文中测试的机器均为 Windows 7 旗舰版，内存 4G，处理器为 Intel Xeon E312xx(Sandy Bridge) 2.40 GHz (2 处理器)。开发语言为 Python 2.7。

5.2 实验样本集

本文通过网络爬虫模块从 packetstorm，guanting 中依据浏览器相关的关键字（internet explorer，IE，firefox，opera，chrome，edge，browser）一共爬取了 1545 个样本，经过筛选（去除非 HTML 样本），一共得到 1089 个已知漏洞触发样本作为种子输入，用于验证测试模式的产生效率以及实际的测试实验。

5.3 实验结果分析

为了验证本文中提出的方法的有效性，本文提出了三个问题并通过实验来验证：

1） 本文的测试模式产生效率如何，能否优于人为的测试模式编写？

3） 本文提出的模式生成的方法是否有发现新的浏览器漏洞的能力？

4） 将本文提到的模糊测试方法应用于实际的浏览器模糊测试中，测试效果如何？

针对于上述提到的三个问题，本文一共设计了三个实验来进行验证与分析：

（4） 问题一

测试模式的产生为从样本文件作为输入，经过测试模式提取器后输出测试模式这一个过程，通过对 1089 有效的种子文件进行测试模式提取，得到如表 5-1 所示结果：

表 5-1 测试模式提取时间消耗

所属浏览器	IE	Firefox	Chrome	Edge	Opera	Other	总计
文件数量	326	189	126	59	122	267	1089
总用时(秒)	3598.65	2086.08	1390.36	650.5	1347.42	3088.9	12161.91
均用时(秒)	11.039	11.037	11.035	11.025	11.044	11.569	11.168

并且通过对 1089 个种子文件测试模式提取时间进行统计，平均每个测试模式获取用时为 11.168 秒，远快于人为构造的数千行代码的时间消耗。

（5） 问题二

通过设计实验对 IE 浏览器 UAF 漏洞 CVE-2012-4792 进行模式提取后形成的模糊测试器对 IE8 8.0.7601.17514 进行为期了 2 天的测试，最终发现了 33 个相同模式的软件异常，其中包括用于提取模式的漏洞样本。除此之外并经过确认包含在该漏洞之后所出现的 2 个已知的安全漏洞，CVE-2013-1347 和 CVE-2013-3189。具体信息如表 5-2 所示：

表 5-2 漏洞详细信息

漏洞编号	公布时间	更新的补丁信息
CVE-2012-4792	2012/11/29	Internet Explorer 8 (KB2799329)
CVE-2013-1347	2013/05/03	Internet Explorer 8 (KB2847204)
CVE-2013-3189	2013/08/13	Internet Explorer 8 (KB2862772)

下面将对 CVE-2013-1347 的发现进行分析以验证本文所提出的方法在理论上也具备发现其他漏洞的能力。

```
<!doctype html>
<html>
<head>
<script>
function helloWorld() {
try {
    e0 = document.getElementById("a");
    e1 = document.getElementById("b");
    e2 = document.createElement("q");
    e1.applyElement(e2);
    e3 = document.createElement('button');
    e1.appendChild(e3);
    e1.applyElement(e0);
    e2.outerText = "";
    e4 = document.createElement('body');
    e2.appendChild(e4);
} catch(e) { }
    CollectGarbage();
}
</script>
</head>
<body onload="eval(helloWorld())">
<form id="a">
</form>
<dfn id="b">
</dfn>
</body>
</html>
```

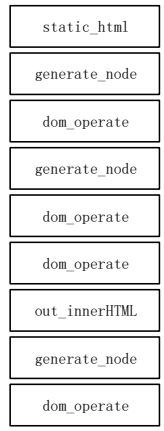

图 5-1 CVE-2012-4792 样本及测试模式

如图 5-1 所示，当输入样本 CVE-2012-4792 之后，通过我们的测试模式提取器将得到右侧所示的测试模式，其表示方式对应于模块的 BNF 描述。我们的样本生成器将会按照右侧的测试模式构造大量的数据样本来对浏览器进行测试。

图 5-2 为我们精简过后的异常样本 CVE-2013-1347，将我们的异常样本的测试模式与原始种子样本的测试模式进行对比之后发现，两者的测试模式基本一致，其中异常样本中多的一次 style_operate 操作为样本生成器中所产生的随机操作，而原始样本中多的模块也在精简过程中被去除了。因此从理论上我们也是能够依据原始样本的测试模式得到我们的漏洞样本，因此本文的测试方法是具备发现新的漏洞的能力的。

```
<!doctype html>
<html>
<head>
<script type = 'text/javascript'>
function logicFuzz(){
    aa = document.createElement("address") ;
    document.getElementById("sss").appendChild(aa);
    bb = document.createElement("area");
    aa.appendChild(bb);
    document.all(8).innerHTML = '<img isMap="false"/>';
    document.all(4).outerHTML = '<th
clientTop="1073741825">ECWzWwhPUqp3YQgh</th>';
    document.all(7).style.display = "none";
    CollectGarbage();
}
function fuzz(){
logicFuzz();
}
</script>
</head>
<body onload = "fuzz();" contentEditable="true" >
<ol><u><span><ins><var><td><ins id="sss">
</ins></td></var></ins></span></u></ol></body>
</html>
```

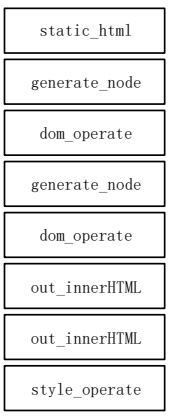

图 5-2 异常样本及测试模式

（6） 问题三

为了验证本文的测试方法在实际的浏览器测试过程中也能产生一定的效果。本文通过从爬取的测试样本随机选取了 10 个作为种子文件作为模式提取并产生 10 个模糊测试器用于全补丁的浏览器模糊测试。通过对

Internet Explorer，Firefox 浏览器进行为期 10 天的测试，为提高测试效率，采用并行方式开展测试。测试过程中，发现了 57 个软件异常，其中包含 1 个高危漏洞（Firefox UAF），漏洞样本简单分析。测试结果如表 5-3 所示：

表 5-3 测试结果

浏览器类型	并行度	测试时间（天）	异常数量	不同异常数量
Internet Explorer 10	10	10	15522	23
Internet Explorer 11	10	10	68972	30
Firefox	10	10	7845	4

通过对全补丁的浏览器进行测试，能够发现一定数量的软件异常，说明本文中提到的方法在对实际的浏览器模糊测试中也是有效的。

6 结论与展望

本文中，提出了一种新的浏览器模糊测试方法。立足于浏览器模糊测试的核心点在于如何构造合适的测试样本，而测试样本的构造核心依靠于构造测试样本的逻辑算法，即测试模式。本文通过对样本文件中测试模式提取从而达到了测试模式快速产生的目的，弥补了现今浏览器模糊测试过程中测试模式产生对人的过度依赖，时间消耗过大等缺陷。最后通过三个实验来验证本文所提出的测试方法的确能够快速产生有效测试模式；并且还能够在对已有漏洞样本测试模式提取基础上，使用产生的模糊测试器测试发现新的漏洞；最后在应用于实际的浏览器测试也能产生一定的测试效果。

然而，本文中提到的测试方法仍具有以下不足之处。一是在对获取的测试模式并未有一个很好的管理和反馈机制；二是本文主要关注于浏览器渲染引擎的测试，而缺少对 JavaScript 解析引擎的支持；三是本文实现的模糊测试器-autofuzzy 中的样本生成器仅为一个雏形，仍需添加更多的基础数据来满足样本的生成。随着浏览器的功能逐渐复杂化以及浏览器各类安全机制的出现，如何能够有效地快速地对浏览器进行测试将会是未来工作的重点，机器学习，深度学习等智能化技术也将会在未来逐渐引入到浏览器模糊测试中。

参考文献

[1]Flexera Software. VULNERABILITY REVIEW 2017. March 13, 2017.

[2]Sutton M, Greene A, Amini P. Fuzzing: brute force vulnerability discovery[M]. Pearson Education, 2007.

[3]吴世忠. 软件漏洞分析技术[M]. 科学出版社, 2014.

[4]Miller C, Peterson Z N J. Analysis of mutation and generation-based fuzzing[J]. Independent Security Evaluators, Tech. Rep, 2007.

[5]Dormann W, Plakosh D. Vulnerability detection in activex controls through automated fuzz testing[J]. Unpublished working paper, 2008.

[6]Hodován R, Ákos Kiss. Fuzzing JavaScript Engine APIs[C]// International Conference on Integrated Formal Methods. Springer International Publishing, 2016:425-438.

[7]“Jesse Ruderman, Introducing jsfunfuzz.” [Online]. Available:http://www.squarefree.com/2007/08/02/introducing-jsfunfuzz/.[Accessed: 14-Jun-2013].
[8]MDN. SpiderMonkey 1.8.8. https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Releases/1.8.8.
[9]Lin Y D, Liao F Z, Huang S K, et al. Browser fuzzing by scheduled mutation and generation of document object models[C]// International Carnahan Conference on Security Technology. IEEE, 2016:1-6.
[10]朱永盛. WebKit 技术内幕[M]. 电子工业出版社, 2014.
[11]Stephen Fewer, “Grinder.” [Online]. Available:https://github.com/stephenfewer/grinder. [Accessed: 18-May-2014].
[12]Valotta R. Taking Browsers Fuzzing To The Next (DOM) Level[J]. 2012.
[13]Valotta R. Fuzzing browsers in 2014[J]. 2012.
[13]M. Zalewski, “american fuzzy lop,” http://lcamtuf.coredump.cx/afl/.
[14]“libFuzzer - a library for coverage-guided fuzz testing - LLVM 3.9 documentation,” http://llvm.org/docs/LibFuzzer.html.
[15]Vinay Katoch. X-Fuzzer. https://code.google.com/archive/p/x-fuzzer/.
[16]Bo Qu, Royce Lu. POWER IN PAIRS:How one fuzzing template revealed over 100 IE UAF vulnerabilities. blackhat EUROPE2014.
[17]Michal Zalewski. cross_fuzz. http://lcamtuf.coredump.cx/cross_fuzz/.
[18]demi6od. ChromeFuzzer. https://github.com/demi6od/ChromeFuzzer.
[19]MDN.Introduction to the DOM.https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
[20]"Panini biography". School of Mathematics and Statistics, University of St Andrews, Scotland. Retrieved 2014-03-22.
[21] 钱文祥. 白帽子讲浏览器安全[M]. 电子工业出版社, 2016.