

Fig.3 Test flow of method

图3 方法测试流程

### 3 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,则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,则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支持的操作进行模块抽象,使测试模式的产生更加适合与高效.

模块抽象需要考虑以下两个因素.

- (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.
- (2) 基于测试范围,需要保证浏览器的核心部分都能被测试,如 X-Fuzzer 仅关注 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,如 Pair-Fuzz<sup>[18]</sup>、cross\_fuzz<sup>[11]</sup>、chromefuzzer<sup>[19]</sup>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 个方面的操作:HTML 标签对象的操作、CSS 样式的操作以及 DOM 树的操作.

##### 1) 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 (use after free) 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.

具体描述见表 1.

**Table 1** Module abstraction of HTML tag object manipulation**表 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	<code>&lt;table height=100&gt;&lt;td&gt;Hello World&lt;/td&gt;&lt;/table&gt;</code> <code>&lt;div id="fuzzy"&gt;&lt;/div&gt;</code> <code>&lt;form action="/s"&gt;&lt;input type="hidden" value="1"&gt;&lt;/form&gt;</code>	static_html
HTML 对象的动态创建	<code>document.createElement("table");</code> <code>document.createElementNS("http://www.w3.org/1999/xhtml", "div");</code>	generate_node
HTML 对象的释放	<code>document.body.innerHTML="";</code> <code>document.documentElement.outerHTML="test";</code> <code>table.innerText="1";</code>	out_innerHTML
特殊标签元素的创建	<code>range, form, iframe, textbox</code> 等 <code>range=document.createRange();</code> <code>range.setStart(element0,0);</code> <code>&lt;iframe id="1"&gt;&lt;/iframe&gt;</code>	special_element

## 2) 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 样式的动态操作两个模块.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 样式所做的设定、修改和删除等操作.具体描述见表 2.

**Table 2** Module abstraction of CSS style operation**表 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	<code>h1 { color: red; font-size: 14px; }</code> <code>p { color: rgb(100%,0%,0%); }</code>	css_style
	<code>td, ul, ol, ul, li, dl, dt, dd {</code> <code>font-family: Verdana, sans-serif;</code> <code>}</code>	
CSS 样式动态操作	<code>document.body.style.backgroundColor="right";</code> <code>document.getElementById("p1").style.borderRight="thick solid #0000FF";</code> <code>document.getElementById("myTable").style.captionSide="right"</code>	style_operate

## 3) DOM(document object model)树的操作

DOM<sup>[20]</sup>,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,会将页面中的元素、属性、文本等解析为一个树形结构,即 DOM 树.页面中的元素、属性、文本等则作为 DOM 树中的树节点存在.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这 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.

具体描述见表 3。

**Table 3** Module abstraction of DOM tree operations  
**表 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	<code>document.getElementById("myList1").appendChild(node);</code> <code>list.removeChild(list.childNodes[0]);</code>	dom_operate
属性和属性值的设置	<code>document.getElementsByTagName("INPUT")[0].setAttribute("type","button");</code> <code>document.getElementsByTagName("H1")[0].removeAttribute("style");</code> <code>document.getElementById('myid').title="fuzzing";</code>	property_operate
方法的设置	<code>object.show()</code> <code>object.hide()</code> <code>document.getElementById('myAnchor').focus()</code>	method_set
事件的监听	<code>object.addEventListener(func_name,event)</code> <code>name1.attachEvent('onclick',function(){info.innerHTML+="red"+"&lt;br&gt;";});</code>	event_listener

### 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.** DOM 树操作的模块抽象。

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**

- 第 1 行~第 4 行定义了 3 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.
- 第 5 行~第 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 中 3 个类别中,其余成分不进行分析.
- 第 13 行~第 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*.
- 第 17 行~第 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*.
- 第 21 行~第 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见表 4.

**Table 4** Introduction to BNF symbols

表 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”
[]	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见表 5.

**Table 5** Abstract module and BNF description

表 5 抽象的模块与 BNF 描述

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

注(关键类型说明,特殊元素的描述依据具体需求,故此未给出相应的 BNF 描述):

- html\_tag 表示 html 的标签元素
- event 表示支持的事件类型
- selector 表示 CSS 的选择器
- arg 表示对应方法的参数.
- property 表示节点对象所对应的属性
- function\_name 表示注册事件的回调函数名称
- nodeX 表示节点对象
- value 表示属性所对应的属性值
- content 表示随机的内容
- method 表示所支持的方法类型



### 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 JavaScript 代码.经过规范后的样本,其 JavaScript 代码完全处于<script>标签内,因此,这将是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 4 所示.

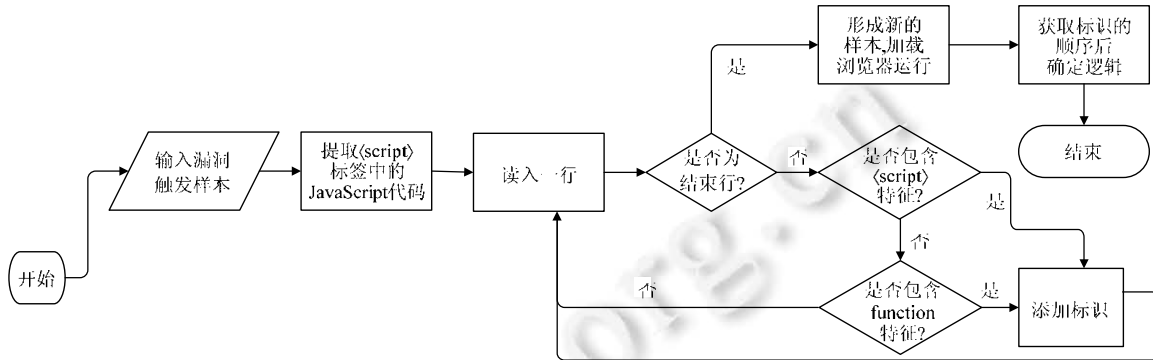


Fig.4 Logical acquisition process

图 4 逻辑获取流程

## 4 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 1 部分是人工构造的一个基础的数据集,第 2 部分是构造出测试样本.

### 4.1 数据集

针对每一个模块,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 5 所示.

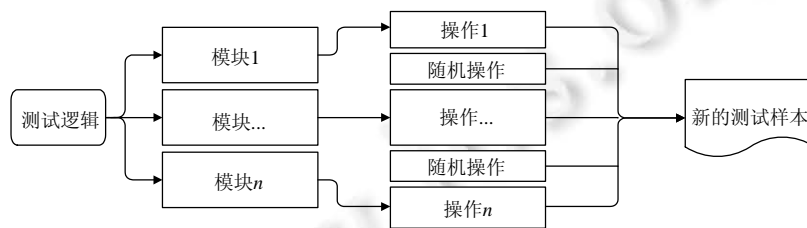


Fig.5 Test cases generation process

图 5 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.

## 5 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,并通过相关实验来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) 2.40GHz(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 1 545 个样本,经过筛选(去除非 HTML 样本),一共得到 1 089 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 3 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?
- 2) 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?
- 3) 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?

针对上述 3 个问题,本文一共设计了 3 个实验来进行验证与分析.

#### (1) 问题 1

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过程.通过对 1 089 个有效的种子文件进行测试模式提取,得到如表 6 所示的结果.

**Table 6** Test pattern extraction time consumption

表 6 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(s)	3 598.65	2 086.08	1 390.36	650.5	1 347.42	3 088.9	12 161.91
均用时(s)	11.039	11.037	11.035	11.025	11.044	11.569	11.168

另外,通过对 1 089 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 11.168s,远快于人为构造的数千行代码的时间消耗.

#### (2) 问题 2

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 进行了为期 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.并经过确认其中包含 CVE-2012-4792 之后出现的两个已知的安全漏洞,CVE-2013-1347 和 CVE-2013-3189.具体信息见表 7.

**Table 7** Vulnerabilities details

表 7 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,以验证本文所提方法在理论上 also 具备发现其他漏洞的能力.

如图 6 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.

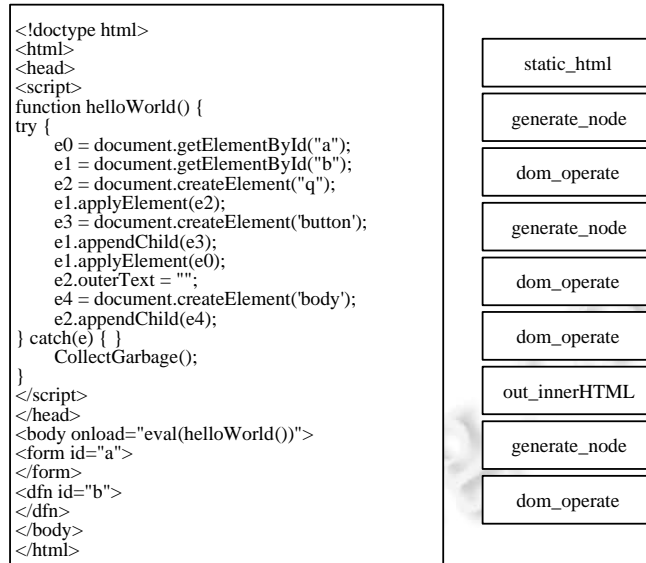


Fig.6 Crash sample and test patterns of CVE-2012-4792

图 6 CVE-2012-4792 异常样本及测试模式

图 7 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,我们也能依据原始样本的测试模式得到相应的漏洞样本,因此,本文的测试方法具备发现新漏洞的能力.



Fig.7 Crash sample and test patterns

图 7 异常样本及测试模式

(3) 问题 3

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,本文从爬取的测试样本中随机选取 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,为了提高测试效率,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.测试结果见表 8.

Table 8 Test results

表 8 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	15 522	23
Internet Explorer 11	10	10	68 972	30
Firefox	10	10	7 845	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

本文提出了一种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的提取,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 3 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的测试模式;并且还能在对已有漏洞样本测试模式提取的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,还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、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.

## References:

- [1] Flexera Software. Vulnerability review 2017. Research Report, 2017. 20–22.
- [2] Sutton M, Greene A, Amini P, Wrote; Huang L, Yu LL, Li H, Trans. Fuzzing: Brute Force Vulnerability Discovery. Beijing: China Machine Press, 2009 (in Chinese).
- [3] Wu SZ, Guo T, Dong GW, Zhang P. Software Vulnerability Analysis Technology. Beijing: Science Press, 2014. 215–246 (in Chinese).
- [4] Miller C, Peterson ZNJ. Analysis of mutation and generation-based fuzzing. Technical Report, Independent Security Evaluators, 2007. <https://www.defcon.org/html/links/dc-archives/dc-15-archive.html#Miller>
- [5] Zalewski M. American fuzzy lop. 2017. <http://lcamtuf.coredump.cx/afl/>
- [6] libFuzzer—A library for coverage-guided fuzz testing—LLVM 3.9 documentation. 2017. <http://llvm.org/docs/LibFuzzer.html>
- [7] Hodován R, Kiss Á. Fuzzing JavaScript engine APIs. In: Proc. of the Int'l Conf. on Integrated Formal Methods. Springer Int'l Publishing, 2016. 425–438. [doi: 10.1007/978-3-319-33693-0\_27]
- [8] Ruderman J. Introducing jsfunfuzz. 2015. <http://www.squarefree.com/2007/08/02/introducing-jsfunfuzz/>
- [9] MDN. SpiderMonkey 1.8.8. 2014. <https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Releases/1.8.8>
- [10] Katoch V. X-Fuzzer. 2012. <https://code.google.com/archive/p/x-fuzzer/>
- [11] Zalewski M. cross\_fuzz. 2011. [http://lcamtuf.coredump.cx/cross\\_fuzz/](http://lcamtuf.coredump.cx/cross_fuzz/)
- [12] Lin YD, Liao FZ, Huang SK, Lai YC. Browser fuzzing by scheduled mutation and generation of document object models. In: Proc. of the Int'l Carnahan Conf. on Security Technology. IEEE, 2016. 1–6. [doi: 10.1109/CCST.2015.7389677]
- [13] Zhu YS. WebKit Technology Insider. Beijing: Electronic Industry Press, 2014. 15–63 (in Chinese).
- [14] Valotta R. Taking browsers fuzzing to the next (DOM) level. 2012. 1–40.

- [15] Valotta R. Fuzzing browsers in 2014. SyScan360. 2014. 1–43.
- [16] Fewer S. Grinder. 2014. <https://github.com/stephenfewer/grinder>
- [17] Qian WX. White Hat Talks about Browser Security. Beijing: Electronics Industry Press, 2016. 229–231 (in Chinese).
- [18] Qu B, Lu R. Power in Pairs: How one fuzzing template revealed over 100 IE UAF vulnerabilities. In: Proc. of the Blackhat EUROPE 2014. 2014. 1–28.
- [19] demi6od. ChromeFuzzer. 2015. <https://github.com/demi6od/ChromeFuzzer>
- [20] MDN. Introduction to the DOM. 2017. [https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model/Introduction](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction)

## 附中文参考文献:

- [2] Sutton M, Greene A, Amini P, 著;黄陇,于莉莉,李虎,译.模糊测试:强制性安全漏洞发掘.北京:机械工业出版社,2009.
- [3] 吴世忠.软件漏洞分析技术.北京:科学出版社,2014.215–246.
- [13] 朱永盛.WebKit 技术内幕.北京:电子工业出版社,2014.15–63.
- [17] 钱文祥.白帽子讲浏览器安全.北京:电子工业出版社,2016.229–231.



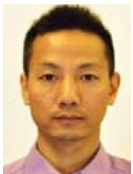
霍玮(1981—),男,河北秦皇岛人,博士,副研究员,博士生导师,CCF 专业会员,主要研究领域为软件漏洞挖掘,安全评测,基于大数据的软件安全分析,智能终端系统及应用安全分析.



戴戈(1992—),男,硕士,主要研究领域为软件安全,软件漏洞挖掘.



史记(1990—),男,博士,主要研究领域为软件安全,软件漏洞挖掘.



龚晓锐(1973—),男,高级工程师,主要研究领域为网络攻防,软件逆向分析,Web 安全,移动互联网安全.



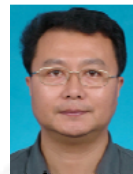
贾晓启(1982—),男,博士,研究员,博士生导师,主要研究领域为网络攻防技术,操作系统安全,云计算安全.



宋振宇(1985—),男,博士,助理研究员,主要研究领域为网络空间安全,网络技术.



刘宝旭(1972—),男,博士,研究员,博士生导师,CCF 专业会员,主要研究领域为网络攻防技术,安全态势感知技术.



邹维(1964—),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件安全分析理论与技术,网络安全评测.