北京航空航天大学计算机学院

硕士研究生文献综述

研究课题：一种反爬虫系统的设计、实现以及评估

研 究 生：张浩凌

指导教师：李舟军

学科专业：网络空间安全

学　　号：SY1706404

2018年11月25日

一种反爬虫系统的设计、实现以及评估

摘要：

随着各类基于大数据和AI的应用的兴起，能够快速廉价地获取大量有效数据的能力，成为互联网时代企业和个人竞争力的体现。因此，网络爬虫在数据收集收集方面的重要性逐渐凸显出来。但是，网络中充斥着大量恶意的爬虫，或者多线程高并发耗尽服务器的资源[1]，或者爬取敏感、高价值的数据用于不法用途。

因此，构建一个成熟有效的反爬虫系统，成为一个亟待解决的问题。但遗憾的是，传统的反爬虫方式过于保守和被动，在漏检率和误检率居高不下的情况下，还只能通过单一的反制手段来限制爬虫（如IP封锁，访问频率限制，虚假数据等），在反爬虫的战争中收效甚微。为此，本文提出了一种新型的反爬虫的系统，融合爬虫检测技术，爬虫行为分析和溯源技术，并通过动态符号执行、模糊测试以及污点分析等二进制漏洞挖掘方法，挖掘爬虫使用的框架、处理脚本，headless browser驱动程序的漏洞，并通过返回恶意的payload，对运行爬虫的主机进行反向渗透，最终诱使爬虫进程奔溃，甚至获取爬虫主机权限。

此外，本文将遵循上述的设计思路，实现该反爬虫系统的原型系统，并将其部署在真实的业务系统上，使用网络上常见的爬虫进行测试，从漏检率、误检率、系统性能损失等多方面的指标来评估反爬虫系统的性能。

关键词：网络爬虫，反爬虫机制，headless browser，爬虫检测，爬虫溯源，动态符号执行

Design, implement and evaluate an anti-crawling system

Abstract：

With the flourish of the Applications based on the Big Data and Artificial Intelligence, it has aroused our attention in how to collect numerous valid data rapidly at the least cost, which could be regarded as an aspect of competitive competence both for the individuals and the companies. And thus, the power of the crawlers in collecting data has been addressed. However, there are plenty of malicious crawlers filling in the cyberspace, try to deplete the servers' resource with endless requests concurrently, or theft the sensitive data for illegal use.

Therefore, we are supposed to build up a feasible, active and robust anti-crawling system to stop those malicious crawlers, while most of the contemporary anti-crawling systems are using passive strategies. Those anti-crawling systems could only harness single mechanism to block the crawlers(such as IP blocking, limit the frequency of requests, fake data), and that does not seem to be effective in most cases. In this paper, I propose an anti-crawling system, which combines the crawler detection with the analysis of crawler's behaviors, then utilize the dynamic symbolic execution, fuzzing and dynamic taint analysis, to excavate the vulnerabilities of the frameworks, the handling scripts and the headless web-driver binaries of the crawlers. Ultimately, malicious payloads would be generated to feed the crawler when it crawls our pages, leading it to crash down or even obtain the system privileges of the crawler's host. In addition, at the end of the paper, I would build up the prototype of the anti-crawling system according to the technologies and mechanisms mentioned before, with the deployment to the web application in real production environment. Furthermore, several tests would be conducted with the most common crawlers gathered from the Internet to testify its ability to block the crawlers and the performance once deployed.

Keyword：crawler， anti-crawling system, headless browser, crawler detection, crawler trace，dynamic symbolic execution

1 背景及概述

1.1 研究背景

1.1.1 网络爬虫定义

机器学习，数据挖掘等大量数据依赖型的技术在高速发展的同时，对于相关数据的质和量都提出了更高的要求。网络爬虫，又称网络蜘蛛或者网络机器人，在这样一个数据消费的时代，扮演着数据搬运者和传递者的角色。在其运行的生命周期中，往往会按照开发者预设的规则，爬取指定的URL地址或者URL地址列表，并将获取到的数据预处理成标准化的格式[2]。

1.1.2 恶意爬虫及其影响

普通的网络爬虫并没有危害，相反，搜索引擎存储数据的来源都是基于大量分布式网络爬虫爬取得到的结果，网络爬虫在数据传递过程中起到至关重要的重要。但是多线程高并发的失控爬虫，针对网站隐私数据的窃密爬虫，以及不遵守爬虫道德规范的恶意爬虫，都对网络空间健康的生态环境提出了巨大的挑战。

爬虫失控往往是具有多线程操作的通用型爬虫，未能控制爬行时间间隔，或者因为未添加地址环回检测的处理逻辑，在处理特殊的地址链接时陷入死循环中。失控的爬虫通常给网站的性能资源和带宽资源带来巨大的消耗，甚至会影响正常人类用户的体验，这样的爬虫对网站的影响相当于是DDOS攻击[1]。每年的三月份，是失控爬虫的高发期，原因正是因为大量的硕士在写论文时会爬取网站数据用于数据挖掘或者机器学习。

此外，部分互联网公司会爬取其他同行的优质数据用于商业用途，在给其竞争对手带来经济效益的损失的同时，还会促进产业内部恶性竞争的循环。例如，马蜂窝网站的用户评论数据涉嫌造假事件。

1.1.3 现有的反爬虫技术及其缺陷

为了防范恶意爬虫和失控爬虫，保护网站的重要数据以及网站的可用性，市场上已经出现较为成熟的反爬虫机制。但是，主流的反爬虫策略仍然停留在静态、单一、被动与非实时的状态，很容易被精心设计的爬虫欺骗[4]。主流反爬虫系统主要分为检测系统和处理系统两部分，检测系统往往通过User-agent检测，访问频率检测以及验证码方案来区分正常用户与爬虫访问的流量。处理系统则采用单一手段，通常过滤爬虫的IP地址，或者对爬虫的访问频率进行限制[4,5]。这样的反爬虫策略在日新月异的反反爬虫技术面前，往往效果不尽人意。

爬虫常用的躲避检测的手段包括：

1. 降低访问频率，使访问频率接近于真实人类用户；
2. 修改HTTP头部特征（包括但不限于UA）；
3. 使用IP代理池（能够有效抵御反爬虫IP封锁的策略）；
4. 使用自动化测试工具（如Selenium）。

1.2 研究的意义

恶意的爬虫往往会给服务器的资源带来巨大的损耗，在盗用数据的同时，也会给针对访问的统计数据带来污染。甚至有少量的爬虫，在爬取某些CMS系统或者web中间件的版本信息后，会使用相应的攻击向量攻击脆弱主机[6]，给网站服务提供者及使用者造成巨大的损失。

网络空间一直是爬虫与反爬虫战斗的前线，随着反反爬虫技术的不断迭代更新，传统的静态、单一、被动与非实时的反爬虫技术难以与之对抗，或者又因为部署成本和部署带来的性能损失而被束之高阁。

本文提出的新型的反爬虫系统，将有效地缓解上述问题，能够增加爬虫扫描的资源成本，增加反反爬虫迭代更新的人力成本甚至在一定条件下，造成爬虫的crash，获取爬虫宿主机的系统权限。

1.3 研究内容概述

本文将提出一种新型反爬虫系统，并从该系统的设计、实现以及评估三个方面具体展开，对相关技术及原理进行深入的介绍。系统所使用的技术主要有：爬虫检测技术，爬虫溯源技术，以及爬虫漏洞挖掘与攻击载荷的自动化生成。

爬虫检测技术章节将重点介绍较为常见的爬虫检测技术以及新型爬虫检测技术，并简述这些技术的使用范围及缺陷和优点。

爬虫溯源技术将介绍如何通过浏览器指纹、服务端session、IP地址以及javascript来收集爬虫宿主机的一些信息。

爬虫漏洞挖掘技术将介绍如何自动化地挖掘爬虫使用的框架，处理脚本以及headless browser的漏洞。

2 爬虫检测技术

2.1 爬虫分类

网络爬虫大致可分为四种。通用网络爬虫，聚焦网络爬虫，增量式网络爬虫，深层网络爬虫[2,7]。

1. 通用爬虫：通用网络爬虫又称全网爬虫(Scalable Web Crawler)，爬行对象主要是门户站点搜索引擎和大型Web服务提供商采集数据。通常会从一个种子URL地址开始，递归地搜索遍历该种子地址下的所有URL地址。

2. 聚焦式网络爬虫：聚焦网络爬虫(Focused Crawler)，又称主题网络爬虫(Topical Crawler)[8]，是指选择性地爬行那些与预先定义好的主题相关页面的网络爬虫。通常只需要爬取与其感兴趣的主题相关的页面，保存的页面内容数量少，更新快。

3. 增量网络爬虫：增量式网络爬虫(Incremental Web Crawler)将参照已下载的网页内容，采用增量更新的方式只爬取没有下载或者已经内容发生变化的页面。以较高的访问频次来保证爬取的内容的实时性，但是因其只请求和持久化增量的静态内容[9]，所以请求的数据量较少。

4. 深层网络爬虫：web页面按其存在方式一般分为表层网和深层网。表层网通常可以由搜索引擎检索，以超链接的形式能够到达，Deep Web指的是不能直接检索，隐藏在表单之后的网页[3]。深层网络爬虫（Deep Web Crawler)主要用于爬取普通爬虫无法爬取到的深层网络，通常是提交某些参数（如登录操作）后才能到达的页面[10]。

2.2 爬虫调度策略与爬行策略

爬虫的目的是在短时间内尽可能获得最多的高质量数据。当前有五种评估页面质量的机制[11]：

- Similarity
- Backlink
- Pagerank
- Forwardlink
- Location

而为了爬虫的爬取速度，爬虫往往以并行地方式运行，单也引入了如下的问题[7]：

1. 重复性：线程增加的同时可能增加页面重复的几率
2. 通信带宽损耗：分布式爬虫之间的同步通信损耗大量的带宽和计算资源

并行运行时，爬虫通常采取如下三种方式[7]：

1. 独立运行：所有爬虫独立运行，互不影响，全部爬取完成后，再统一处理爬行资源。
2. 动态队列分配：使用统一的任务队列分配爬虫的爬取任务，每一次爬虫从任务队列中获取到的爬取任务，都是动态且随机的。
3. 静态分配：在爬虫任务开始前，事先为所有的爬虫分派爬取的任务，在爬虫爬取过程中不再做额外的修改。

对于通用爬虫而言，常用的策略主要有深度优先爬行和广度优先优先爬行两种[12]。

1. 深度优先：将链接深度从低到高排列，依次访问下一级网页链接直到不能深入为止。在完成一个分支网页的爬行任务后，返回上一个链接节点，进一步遍历其他链接。
2. 广度优先：按目录层次深浅来规划爬行的轨迹，从浅层的web目录开始爬行，低层次的目录爬行完毕后，再深入下一层继续爬行。可以有效的控制爬行的深度，避免无穷深层次分支的情况下无法走出陷阱。

聚焦爬虫策略的关键是对访问页面和链接的重要性进行评分，并以此作为页面爬取先后的顺序的参考依据，而不同算法计算出的页面重要性也有所不同，从而导致爬虫爬行轨迹的不同。

1. 基于内容的爬行策略：Bra在他的论文中提出了Fish-Search算法[13]，以页面内容与爬虫的聚焦主题的相关性作为度量标准。假设存在鱼群，鱼群按照深度优先的策略在爬行空间中巡游，相关度高的分支，鱼群的数量会增加；而相关度低的分支，鱼群的数量会降低。通过这样的机制来保证主要的爬虫资源和时间都能够聚焦在感兴趣的主题上。
2. 基于链接结构的爬行策略：Page-rank算法[14]主要用于对搜索引擎搜索到的内容进行结果排序，也可以用于评价链接的重要性。在爬虫选取爬行链接时会优先选取Page rank较大的页面中的链接作为优先爬取的对象。
3. 基于增强学习的爬行策略：Rennie 和 McCallum将增强学习（reinforcement learning)引入聚焦爬虫[15]，他们试图通过机器学习找到一组策略，使得在该策略的激励达到最优解。
4. 基于语境图的爬行策略：Diligenti提出一种通过建立语境图来描述不同网页之间相关度的算法[16]。用大量的数据训练出一个机器学习系统，利用该系统计算不同页面到某一主题页面的距离，以此作为确定访问顺序的参考.

2.3 爬虫检测方法

2.3.1 日志语法分析模式

2.3.1.1 独立字段解析（individual field parsing）

通过比较HTTP请求中的独立字段与数据库中的爬虫常使用关键字和banner信息，来判定请求是否来源于爬虫[18]。常用的字段主要是User-Agent。基于User-Agent和其他类似的头部内容检测方法，是检测爬虫最为简单且常用的方法。部分爬虫不会修改其HTTP头部的User-Agent字段，而直接使用其原始的UA对网页进行爬取。而浏览器的UA与爬虫的UA有着明显的区别，甚至可以通过User-Agent字段中的特殊关键词判断访问的浏览器/爬虫的类型。但是，一部分未知爬虫的UA特征可能并未收集在数据库中，还有一些爬虫会修饰其的User-Agent字段，使得通过UA进行单一爬虫检测的漏检率很高。

2.3.1.2 多层次日志分析（multifaceted log analysis）

首先，请求robots.txt的访问者被标记为爬虫，这样的流量大概占总web应用访问流量的0.5%。然后，再对日志中的访问IP地址做反向DNS查询，如果dns解析得到的域名中包含诸如robot, bot, search, spider以及crawl这样的关键字，则可以将访问者标记为爬虫，使用该方法大概能检测到16.1%的流量来源于爬虫。最后，通过从网络搜集爬虫池的IP地址段，将来源于爬虫IP的访问标记为爬虫[19]。

2.3.2 流量模式分析

2.3.2.1 语法与模式分析（syntactic and pattern analysis）

Geens最早提出了将语法分析与流量分析模式结合在一起的检测方法，并给定了两个评估参数[20]:

r 覆盖率 r = 检测出爬虫数量/实际爬虫数量

p 准确率 p = 检测出爬虫数量/被检测器标记为爬虫的数量

通过人工标记数据的方法，分别计算每种分析模式的r和p，并最终得出了以下的检测策略（以伪代码描述）：

```
if ("/robots.txt" in session.url):
   session.role = robots
if (session.ip in robots.ip):
   session.role = robots
if (session.ua in robots.ip):
   session.role = robots
if (HEAD in session.methods):
   session.role = robots
if (session.refer == NULL && ! "*.[png|jpg|gif]" in session.url):
   session.role = robots
```

2.3.2.2 资源请求模式分析（resource request patterns）

爬虫在爬取网站时，往往会聚焦于一些特殊的网络资源。guo等人利用这一特性，提出了一种基于资源请求模式分析的爬虫检测方法[21]。首先在第一种策略中，他们将请求的url资源分为网页，文档，脚本，图片，视频，下载等多种类型。然后他们将依据IP，UA这些信息，区分不同的请求客户端，并统计每一个客户端针对每种资源的访问次数。如果访问次数过多或者访问占比过多，则可以认为请求来自于爬虫；在第二种策略中，他们统计了同一个session对同一个网页的所有资源的请求次数，如果同一个网页中的资源请求数有较大的差异，则可以认为当前的session来自于爬虫。

2.3.2.3 请求速率模式分析 （query rate patterns）

为了从速率模式区分正常访问与爬虫访问，需要采集如下的信息：请求提交速率，请求平均时间间隔，session的活动持续时间，请求时间与作息时间的相关性。

在Duskin的文章中，他们提出了两种不同的爬虫检测方法[22]：首先，基于每个session的请求总次数来划分人类和爬虫；然后，基于每个session的最短请

求间隔时间来区分人类和爬虫。

2.3.3 分析学习

2.3.3.1 决策树（decision tree）

Tan 和 Kumar 利用爬虫的session中的一些特征向量来区分人类用户与爬虫[23]。首先，他们从日志数据中，将所有的请求按照session进行分类。然后他们从每个session中抽取出25个特征向量，并使用其中显著的三个特征为原始数据集打上标签。并在随后的处理中，考虑全部25个特征向量，以之前已经标记好的数据为学习数据使用 C4.5 决策树算法构造决策树。最后的分类效果显示该算法具有较好的覆盖率和准确率。

2.3.3.2 隐马尔科夫模型（Hidden Markov model）

Lu 和 Yu 在2006年提出一个基于请求到达模式的用户与爬虫识别程序[24]。他们把人类访问的特征概括为访问所有内嵌页面资源的突发性访问HTTP请求，在整个session过程中，会因为阅读和处理页面内容而存在不活跃时间。而爬虫则会以一个稳定均匀的速率爬行特定类型的资源。在他们提出的算法中，他们先将流量按等时间段切片，在每个时间片中会有一到多个请求到达。通过UA来判定是否为爬虫，并以此来标记数据训练HMM。在最后，以未来的请求序列作为已训练的HMM模型的输入，使用forward-backward procedure来计算其可能是爬虫请求的概率。

2.3.4 图灵测试系统（turing test system)

2.3.4.1 CAPTCHA

Completely Automated Public Turing test to tell Computers and Humans Apart（CAPTCHA）最早在2003年由Ahn等人提出[25]。CAPTCHA通常都是以验证码形式存在，设计的目标是让真实使用者更容易通过测试而爬虫更难通过测试。

2.3.4.2 隐性的人类浏览行为(implicit human browsing behavior)

通过一些隐性的人类浏览网页时特有的行为来帮助区分真实用户和爬虫。比如在服务端生成一个秘钥k，并编码在混淆后的javascript中，当javascript被执行后，或者监听到某些人类用户特有的行为后（如点击事件或者键盘键入），使用该k向服务端请求响应的url地址，则该请求来源可以被标记为人类用户。针对真实用户与爬虫访问的资源类型不同，可以使用特殊标记的CSS样式表文件或者透明像素的图片文件来识别爬虫。正常加载这些标记文件有很大的可能性是正常用户，而未加载这些资源的用户，则很可能是爬虫。

3 爬虫溯源技术

3.1 浏览器指纹

不同用户的浏览器具有不同的特征信息和丰富的数据，网站可以通过在用户访问时通过不同的 API 和技术手段获取浏览器特征信息构建独特的浏览器指纹。浏览器指纹技术能够在无cookie的情况下提供一种追踪用户的鲁棒性方案。

3.1.1 浏览器指纹算法

令指纹生成算法为 $F(\cdot)$。当出现一个新的浏览器信息 x 时，生成一个浏览器指纹 F(x)。该算法遵循离散概率密度函数 $P(f_n)$。其中，fn 为某个特征信息的指纹生成结果 ；n∈[0,1,…,N]，N 为特征信息的个数。对于某单个特征信息的指纹生成结果 fn，使用自信息量 I 表示该浏览器指纹所包含信息的比特数，即：

$$I(F(x)=f_n)=-log_2(P(f_n)) \qquad (1)$$

当指纹由多个不同的特征信息组合而成时，假设不同特征信息对应的指纹

生成算法为 Fs()，s∈[0,1,…,S]（S 为指纹生成算法的个数），则根据公式（2）单独计算每个特征信息的指纹生成结果 fn,s 的自信息量 Is(fn,s)，并根据公式（3）定义指纹组件的信息熵 Hs(Fs)。

$$I_s(f_{n,s}) = -log_2(P(f_{n,s})) \quad (2)$$

$$H_s(F_s) = -\sum_{n=0}^{N} P(f_s, n) log_2(P(f_s, n)) \quad (3)$$

信息熵表征浏览器所有自信息量的期望值。对于两个相互独立的特征组件，自信息量可直接线性相加。根据自信息量 I 可以确认指纹归属对象的身份。I 包含的若干比特信息中的每一比特信息都能将该浏览器指纹可能的归属集合减半。特征信息进行组合生成指纹，每一个特征信息具有若干比特的信息熵，熵值越大，则越能准确地区分不同的浏览器实体。因此要选取恰当且包含足够比特信息的特征信息集合，通过该特征信息集合生成的指纹能够唯一确认指纹归属对象的身份。本文选取的特征信息集合为浏览器类型及其版本、屏幕分辨率、设备像素比、Firefox 特殊信息和字体列表。

3.2 浏览器特定指纹

ECKERSLEY根据浏览器的环境，通过收集一些网站能够获取的浏览器常用或者不常用的特性，用以生成浏览器的指纹[26]。这些特性中有一部分可以通过简单静态的HTTP请求中推断出来，也可以经由AJAX接口收集。特性收集过程中，有一部分特性是简单明了的，但有一部分特性是来源于一些浏览器细节。有些浏览器禁用了javascript，那么他会使用video，plugins，fonts和supercookie的默认值。因此可以通过这一细节来推断出浏览器是否禁用了javascript。以下是一些常用的，易于采集的常见浏览器特性：

Variable	Source	Remarks
User Agent	Transmitted by HTTP, logged by server	Contains Browser micro-version, OS version, language, toolbars and sometimes other info.
HTTP ACCEPT headers	Transmitted by HTTP, logged by server	
Cookies enabled?	Inferred in HTTP, logged by server	
Screen resolution	JavaScript AJAX post	
Timezone	JavaScript AJAX post	
Browser plugins, plugin versions and MIME types	JavaScript AJAX post	Sorted before collection. Microsoft Internet Explorer offers no way to enumerate plugins; we used the PluginDetect JavaScript library to check for 8 common plugins on that platform, plus extra code to estimate the Adobe Acrobat Reader version.
System fonts	Flash applet or Java applet, collected by JavaScript/AJAX	Not sorted; see Section 6.4.
Partial supercookie test	JavaScript AJAX post	We did not implement tests for Flash LSO cookies, Silverlight cookies, HTML 5 databases, or DOM globalStorage.

表 3.2 浏览器特性

3.3 Canvas 指纹

Mowery和Shacham提出了通过HTML 5 的 Canvas API以及WebGL渲染文本得到像素差异，作为一种具有高信息熵的浏览器指纹生成方式[27]。

canvas元素是html5规范中新增的一类元素，能够在屏幕上可编程地绘制图像，并且被大部分主流的浏览器支持。

绘制一个基本的canvas图形的方法相当简单：需要浏览器提供给用户一个图形渲染的环境，使用环境中的API来渲染你对canvas图像的操作和改变。在目前使用的html5规范中，广泛使用的2d环境能提供诸如fillRect， lineTo以及arc这样的一些基本绘图功能。也有一些复杂的功能可以支持贝塞尔曲线，颜色梯度的这样一些功能。

- Canvas Text渲染：给定字体，字体颜色以及位置参数，2d context能够在canvas绘制出任意的文本，如下是一个调用canvas的案例：

  <script type="text/javascript">

```
var canvas = document.getElementById("drawing");
var context = canvas.getContext("2d");
context.font = "18pt Arial";
context.textBaseline = "top";
context.fillText("Hello, user.", 2, 2);
</script>
```

- Canvas 像素抽取： 2D context提供了一个getImageData方法，通过该方法能够获取一个给定区域范围内的图片对象，该对象是以图片中的每一个元素的RGBA值组成的。其次，canvas对象本身提供toDataURL方法，当提供一个图片作为该方法的输入后，该方法能够将完整的图片内容以base64编码的形式返回。以上两个方法均严格遵循浏览器同源策略。

- WebFront：webFront是定义在CSS3中的规范，允许用户按需加载远程字体，而不是只能依赖于已经安装在本地的字体。在使用这个特性时，需要加入@front-face这样一条规则，并使用src属性指定远程字体资源的url地址。为了使用WebFront，需要开启WebFront Loader函数库。通过使用该函数库，WebFront仅仅通过JavaScript就能够加载。

- WebGL：WebGL提供了一个javascript的API用于在canvas上绘制图像，目前的主流浏览器都支持WebGL的硬件加速选项，可以使用图形硬件来渲染每一帧，目前WebGL也通过各自的canvas环境暴露出相应的函数接口，类似于OpenGL API,可以使用GLSL（OpenGL Shading Language）编程，在编译后，可以直接运行在硬件显卡上。

3.4 JavaScript 引擎指纹

MULAZZANI等人提出了通过javascript的一致性测试的结果来区分不同浏览器，从而生成指纹的方法[28]。首先对于某一浏览器执行一组一致性测试，然后

将失败的测试集与已知的浏览器的失败测试集作比较，从而匹识别出一个特定的浏览器类型及其版本。

3.5 跨浏览器指纹

BODA等人提出可以通过利用OS以及硬件层的特性，如显卡，CPU和已经安装的脚本插件等，可以用于区分不同的浏览器，对于同一宿主机上的不同浏览器，依然有着显著的效果[29]。

3.6 WebRTC技术

webRTC全称为web-based real-time communication。它在浏览器与浏览器或者设备与设备之间直接建立通信的信道，而不需要经由第三方服务器的转发[30]。最初的设计目的是提供直接地点到点的视频和音频传输手段，因此webRTC协议对外提供了一系列的realtime transport protocol的API接口，其系统的系统架构图如下图所示：

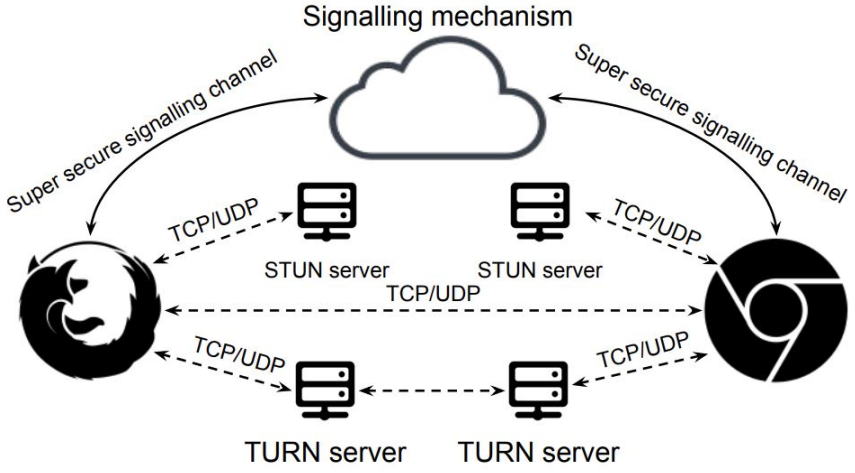

图 3.6 系统架构图

webRTC的通信信道建立过程如下:

1. 初始节点创建一个包含初始端点的offer；

2. 该offer通过信号通道（signalling channel)传递到另一个节点；
3. 另一个节点基于offer创建相应的回答并将其返回到初始节点；
4. 两个通信节点将通过ICE协议连接STUN服务器来解析公共地址信息，并检查连通性；
5. 当每个节点都有对方的信息时，连接完成，数据传输开始。

WebRTC规范中不允许明文发送数据，双方为每次连接生成一个新的自签名的证书，signal channel作为一个可信信道来传输证书指纹，从而实现认证。尽管webRTC使用了具有高度高度安全性的传输协议，但是也引入了新的需要额外保护的敏感资源：

1. 两个端点的外网IP（C-S模型中，server能获得client的ip，在webRTC中，两个建立通信的端点可以获得对方的外网IP）；
2. 私有IP和内部网络（在连接过程中，client的私有IP被暴露）
3. 节点身份（c-s模型中只需要验证服务器的身份，在webRTC模型中需要验证双方的身份）

而这些敏感信息，可以通过WebRTC接口被javascript获取。因为为了构建最高效的连接链路，比如在NAT的内网中，浏览器所在宿主机使用私有IP，在NAT之后需要进行NAT穿透，所以所有的网络接口IP地址都是暴露给Javascript。利用这个特性，可以获得使用网络代理后的爬虫的真实IP地址，也可以使用javascript进行内网扫描，发现内网中宿主机的弱点。

进一步，利用javascript获得的各个IP地址，从而对每一个浏览器建立唯一的识别标记，可以用于跟踪浏览器或者爬虫。

3.7 selenium与headless browser

越来越多的网站使用混淆的javascript来进行动态的页面渲染，以增加爬虫爬取网站的难度。但是，开源的web自动化测试工具集selenium为运行页面中内嵌的javascript提供了成熟的解决方案，因此，常被各类爬虫用于对抗使用了js混淆的网站。在selenium升级到2.0之后[31]，它重点发展了selenium webdriver，由以前的使用selenium server与浏览器通信到使用浏览器自身的webdriver与浏览器进行通信，绕过了javascript沙箱，提升了测试效率和对各个浏览器的兼容性。

selenium对外分开放了大量API接口，可以使用各类编程语言控制测试过程。selenium架构图如下图所示：

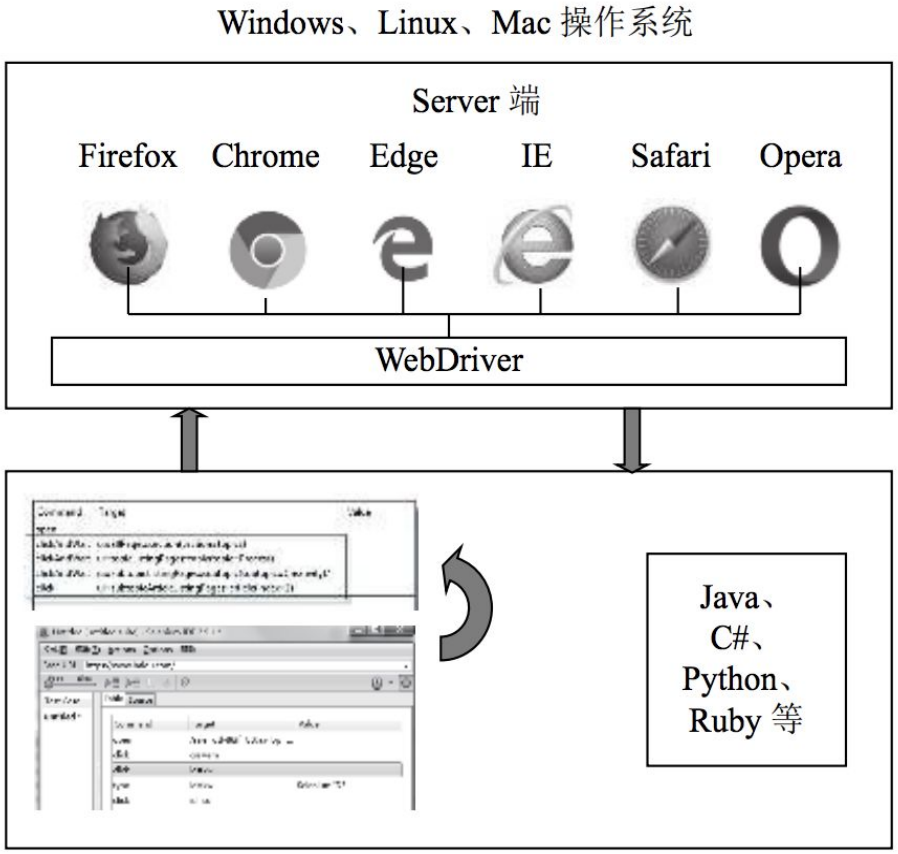

图 3.7 selenium架构图

Headless browser 是一种没有UI界面的模拟浏览器，这些浏览器能够运行网页中的代码，但是不会直接显示其渲染效果。比较著名的headless broswser有chrome headless以及phantomjs。phantomjs可以运行大部分主流的操作系统上，使用QtWebKit作为后端，可以支持各种各样的web标准，包括但不限于：DOM树解析，CSS选择器，JSON数据解析，CANCAS渲染等。

4 爬虫漏洞自动化挖掘技术

4.1 符号执行

4.1.1 符号执行定义

符号执行是二进制动态分析中一种至关重要的技术。它的关键思想是使用符号值而不是固定值，用作程序或者函数的输入。并将程序中的变量表示成与符号输入相关的符号表达式。最终，程序输出的计算结果以符号输入的函数表示[32]。

在软件测试中，符号执行常用于生成能够达到每一个执行路径的测试输入集。简而言之，就是一个程序执行的路径通常是true和false条件的序列，这些条件是在分支语句处产生的。在序列的i^{th}　　位置如果值是true，那么意味着i^{th}条件语句走的是then这个分支；反之如果是false就意味着程序执行走的是else分支。一个程序中的所有路径可以构成一株执行树。

符号执行在执行过程中维护一个状态a，用于将变量映射到符号表达式，以及一个符号路径约束pc，这是一个无量词的一阶公式。在函数的初始化阶段，a被初始化成空的映射，pc为True。a和PC在符号执行过程中不断被更新。当符号执行到达一条执行路径的尽头时，可以通过约束求解器和给定的PC，计算出能够到达该路径的实际值。

对于包含循环或者递归的代码的符号执行，如果循环或者递归的次数是符号化的，那么可能回得到一个无穷的路径。例如在如下的代码中：

```
void testme inf () {
int sum = 0;
int N = sym input();
while (N > 0) {
sum = sum + N;
N = sym input();
}
}
```

一个包含n个True，并紧接着一个False的符号路径约束表达式是：

$$\left(\bigwedge_{i\in[1,n]} N_i > 0\right) \wedge (N_{n+1} \leq 0)$$

因此，在实际应用中，通常需要限制符号执行的超时时间或者路径的数量。传统的符号执行还有一个致命的缺陷，就是当符号路径中存在不能被约束求解器高效求解的公式时，符号执行的输入就难以被计算出来。

4.1.2 现代符号执行技术

4.1.2.1 导向型自动随机化测试

一般称具体执行，或者导向型自动随机化测试[34]。在动态符号执行的基础上，程序会接受一些固定的输入值。具体执行维护了维护固定状态（concrete state）以及符号状态（symbolic state)两种状态。符号状态仅仅保留没有具体值的变量的映射，而固定状态保留所有变量到他们具体值的映射。和传统的符号执行不一样，因为具体执行技术保存了在符号执行过程中保留了完整的固定状态，它需要一些初始的输入值。

具体执行在程序执行开始前接受给定的或者随机的输入，并在遇到条件申明时收集符号约束，然后使用约束求解器推测之前输入的变化以此来调整下一次符号执行向其他方向执行。这样的策略会不断执行指导该程序的所有路径被遍历。

4.1.2.2 EGT方法

又称为EGT方法，被EXE工具和KLEE工具[35]所采纳，通过区分一个程序的固定状态和符号状态来生效。并在这一过程结束后，如果相关的值是静态的，那么EGT将会在每一次操作前，通过动态检测混合固定执行和符号执行。在这种情况下，所有的执行操作就像在原始程序中进行一样。否则，至少有一个值是符号化的，则这个操作也是符号化的。

4.1.3 符号执行的问题和解决方案

4.1.3.1 路径爆炸

符号技术的主要挑战就是，即使在最简单程序中，也可能会有庞大数目的执行路径。而且路径的增长速度通常都是指数级的。因此在给定时间内，应当优先去探索最相关的路径。此外，符号执行隐式地过滤掉和符号不相关的路径，而且也很难给出这些路径的约束。目前针对路径爆炸问题有两种主要的解决方案：

使用启发式的算法探索那些最有可能的路径，或者使用一些程序分析的方法来降低路径的复杂度。

4.1.3.2 约束求解问题

尽管近几年在约束求解问题上有了很大的进步，约束求解问题任然是符号执行技术的一个瓶颈，它会占用主要的运行时间。实际上，最主要的原因是一部分程序产生了使求解器奔溃的表达式。因此，在符号执行过程通过约束条件的类型来约束求解就很有必要。目前符号执行的工具采用以下两种优化方法：

1. 消除不相关的约束：通过观察发现，一个程序的分支往往只决定与该程序中的少量变量，这也意味着分支也只依赖于路径条件中的一小部分约束[36]。因此，移除与当前路径不相关的约束条件后，符号执行的性能将得以大幅度提升；
2. 增量求解（Incremental solving)：符号执行过程中产生的约束有一个重要的特点，他们在源代码中由一组固定的静态分支表示。因此，许多路径有着相同的约束，进一步地，也会有类似的解。这种情况可以用于提高约束求解的速度，也可以重用之前相似的约束的解。这一特性已经在CUTE和KLEE中被使用[37]。

4.1.3.3 内存建模

程序申明转换成符号约束的准确度将对符号执行的路径的覆盖率有着重大

的影响。例如，使用一个固定长度整数的内存模型也许更加高效，但也有可能导致整数溢出问题，而这也将导致符号执行遗漏关键路径，或者被引导到不相关的路径上去。在准确度和可扩展性的折中上，应参考具体被分析代码的类型，并且考虑不同约束求解算法的性能差异。

4.2 动态污点分析技术

4.2.1 基本定义

动态污点分析的目的是在源和目的之间追踪信息流。程序中任何从污点数据计算得出的变量都被认为已污染（tainted），其他变量被认为未污染（untainted）。污染策略P决定污点在程序执行的过程中如何流动，什么样的操作会产生新的污点，以及针对已污染的数据会有什么样的检查。尽管污染策略会随着被分析的程序的不同而改变，但是基本的概念是一致的[33]。

在污点分析的过程往往会有两种错误，一是污点分析可能会将一个不是从污点数据导出的数据标记成已污染，这样现象被称为过污染（overtained）；此外，动态污点分析可能在分析过程中损失一些信息流，被称之为欠污染（undertainting）。如果一个动态污点分析，既不是欠污染，又不是过污染，那么它可以被认为是准确的。

4.2.2 动态污点分析语义

因为动态污点分析是在代码运行过程中进行的，我们自然而然地会使用到操作语义来表达动态污点分析。污点是否进行传播，引入或者检查这样的策略会被加入到操作语义规则中。为了追踪每一个变量的污点状态，我们把变量重新定义为元组<v,t>的形式，v是一个变量的初始值，t是一个变量的污点状态。

4.2.3 动态污点分析策略

一个完整的动态污点策略包含三个属性：如何引入污点，如何传播污点以及如

何检查污点。

1. 污点引入：一种典型的引入方法是，把所有变量，内存单元全部初始化为污染状态，我们只有一个单一的输入源get_input函数。在真实场景中，get_input获取到的用户输入来自于系统调用或者函数库调用的返回值[38]。污点策略在不同的输入源类型中也会有所不同。
2. 污点传播：污点传播规定了从污点数据导出的变量的污点状态。因为污点状态可以用1bit来表示，因此，通常用命题逻辑来表示污点传播策略。例如，t1 v t2表示如果t1和t2均被污染，则当前结果为被污染。
3. 污点检测：污点状态值通常被用于决定程序运行时的行为。例如，攻击监测器会在跳转地址被污染的情况下停止程序。在SIMPIL[39]中，通过把策略加入到操作语义的前置条件中来实现污点检测。T-GOTO规则使用的是Pgotocheck(t)策略，如果跳转地址有一个被污染的值t，且跳转是安全的时候，Pgotocheck(t)会返回T，否则返回F。如果F被返回，规则的前提条件不能满足，最终导致程序的不正常退出。

4.2.4 动态污点分析的挑战

目前动态污点分析面临如下挑战：

1. 污染地址：很难区分存储地址和存储单元；
2. 欠污染：不能够正确处理某些形式的信息流；
3. 过污染：去除污点比引入污点更加难；
4. 检测时间以及攻击时间：当动态污点分析用于攻击检测的时候，往往不能得到及时的结果反馈。

5 总结

本文从分析恶意爬虫可能造成的危害入手，介绍了当前反爬虫技术的缺陷以及这些技术在实际环境中使用时遇到的问题，从而说明一个具有实时性，健壮性，主动型的反爬虫系统的重要性。

进一步地，本文介绍了整个反爬虫系统设计时可能使用到的技术手段。从爬虫的识别技术，再到通过指纹技术对爬虫的行为进行追踪，最终利用现代爬虫对headless browser的依赖的特点，使用主流的二进制漏洞自动化挖掘技术，发掘headless browser中可能存在的漏洞，并根据爬虫识别的结果生成并发送相应的攻击payload。

最终本文将爬虫识别，追踪和攻击技术综合起来，实现一个具有极高实用价值的反爬虫系统，并将其部署到真实的业务系统上，对其相应的指标进行评估。

6 参考文献

[1] 刘书林, 赵运弢. 基于SOA架构的恶意爬虫DDoS攻击检测技术研究[J]. 数字技术与应用, 2016(10):202-202.

[2] 孙立伟, 何国辉, 吴礼发. 网络爬虫技术的研究[J]. 电脑知识与技术, 2010(15).

[3] M K. Bergman. The Deep Web: Surfaceing Hidden Value [EB/OL]. http://www.completeplanet.com/Tutorials/DeepWeb, 2000.

[4] Alireza Aghamohammadi and Ali Eydgahi. A novel defense mechanism against web crawlers intrusion. In Electronics, Computer and Computation (ICECCO), 2013 International Conference on, pages 269–272. IEEE, 2013.

[5] Doran D , Gokhale S S . Web robot detection techniques: overview and limitations[J]. Data Mining and Knowledge Discovery, 2011, 22(1-2):183-210.

[6] 彭赓, 范明钰. 基于改进网络爬虫技术的SQL注入漏洞检测[J]. 计算机应用研究, 2010, 27(7):2605-2607.

[7] 李盛韬,余智华,程学旗.Web 信息采集研究进展[J].计算机科学,2003.

[8] 刘洁清.网站聚焦爬虫的研究[D].南昌:江西财经大学,2006.

[9] Yan HF, Wang JY, Li XM, et al. Architectual design and evaluation of an efficient Web-crawling system [J].Journal of Systems and
Software, 2002.

[10] S. Raghavan, M. Garcia. Crawling the Hidden Web [C]. In Proceedings of the 27th International Conference on Very Large Database,
2001.

[11] J. Cho. Crawling the web: Discovery and Maintenance of Large-scale Web Data [D]. L.A.: Stanford University, 2001.

[12] 王学松. Lucene + Nutch 搜索引擎开发[M].北京:人民邮电出版社,2008.

[13] Bra D, P. Houben, Kornatzky M. Information retrieval in distributed hypertexts [C]. In Proceedings of the 4th RIAO Conference, New
York, United Staes, 1994.

[14] Menczer F. Complementing search engines with online web mining agents [J].

Decision Support System, 2003.

[15] Rennie J, McCallum A. Using reinforcement learning to spider the web efficiently [C]. In Proceedings of the International Conference on Machine Learning, Slovenia, 1999.

[16] M. Diligenti., F. Coetzee, S. Lawrence, et al. Focused crawling using context graphs [C]. In Proceedings of 26th International Confer-

ence on Very Large Database, Cairo, Egypt, 2000.

[17] Doran D , Gokhale S S . Web robot detection techniques: overview and limitations[J]. Data Mining and Knowledge Discovery, 2011, 22(1-2):183-210.

[18] Ramachandran A, Feamster N. Understanding the network-level behavior of spammers[C]. ACM SIGCOMM Computer Communication Review. ACM, 2006, 36(4): 291-302.

[19] Huntington P, Nicholas D, Jamali H R. Web robot detection in the scholarly information environment[J]. Journal of Information Science, 2008, 34(5): 726-741.

[20] Geens N, Huysmans J, Vanthienen J. Evaluation of Web Robot Discovery Techniques: A Benchmarking Study[C]. Industrial Conference on Data Mining. Springer Berlin Heidelberg, 2006:121-130.

[21] Guo W , Ju S , Gu Y . Web robot detection techniques based on statistics of their requested URL resources[C]. International Conference on Computer Supported Cooperative Work in Design. IEEE, 2005.

[22] Duskin O, Feitelson D G. Distinguishing humans from robots in web search logs: preliminary results using query rates and intervals[C]. The Workshop on Web Search Click Data. ACM, 2009:15-19.

[23] Tan P N, Kumar V. Discovery of web robot sessions based on their navigational patterns[M]. Intelligent Technologies for Information Analysis. Springer, Berlin, Heidelberg, 2004: 193-222.

[24] Lu W Z , Yu S Z . Web Robot Detection Based on Hidden Markov Model[C]. International Conference on Communications. IEEE, 2006.

[25] Ahn L V, Blum M, Hopper N J, et al. CAPTCHA: Using Hard AI Problems for Security.[M]. Advances in Cryptology — EUROCRYPT 2003. Springer Berlin Heidelberg, 2003:294-311.

[26] Eckersley P. How Unique Is Your Web Browser?[C]. Privacy Enhancing

Technologies, International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings. DBLP, 2010:1-18.

[27] Mowery K, Shacham H. Pixel perfect: Fingerprinting canvas in HTML5[J]. Proceedings of W2SP, 2012: 1-12.

[28] Mulazzani M, Reschl P, Huber M, et al. Fast and reliable browser identification with javascript engine fingerprinting[C]. Web 2.0 Workshop on Security and Privacy (W2SP). 2013, 5.

[29] BODA K,FOLDES A M,GULYAS G G,et al.User Tracking on the Web via Cross-browser Fingerprinting[M]. Information Security Technology for Applications. Heidelberg:Springer, Berlin, Heidelberg, 2012:31-46.

[30] Reiter A , Marsalek A . WebRTC: your privacy is at risk[C]. Symposium. 2017.

[31] 张添. 基于Selenium的Web自动化测试[D]. 北京交通大学, 2014.

[32] Cadar C , Sen K . Symbolic execution for software testing: three decades later[M]. ACM, 2013.

[33] All You Ever Wanted to Know About
Dynamic Taint Analysis and Forward Symbolic Execution

[34]Godefroid P, Klarlund N, Sen K. DART: directed automated random testing[C]. ACM Sigplan Notices. ACM, 2005, 40(6): 213-223.

[35] Cadar C , Dunbar D , Engler D . KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs[C]. Usenix Conference on Operating Systems Design & Implementation. USENIX Association, 2009.

[36] Sen K, Marinov D, Agha G. CUTE:a concolic unit testing engine for C[J]. Acm Sigsoft Software Engineering Notes, 2005, 30(5):263-272.

[37] Cadar C , Dunbar D , Engler D . KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs[C]. Usenix Conference on Operating Systems Design & Implementation. USENIX Association, 2009.

[38] Manuel Egele, Christopher Kruegel, Engin Kirda, Heng Yin, and Dawn Song. Dynamic spyware analysis. In Proceedings of the USENIX Annual Technical Conference, June 2007.

[39] Flanagan C, Saxe J B. Avoiding exponential explosion: generating compact verification conditions[J]. ACM SIGPLAN Notices, 2001, 36(3):193-205.