北京航空航天大学计算机学院

硕士学位论文开题报告

论文题目：一种反爬虫系统的设计、实现以及评估

专　业：网络空间安全

研究方向：web安全

研 究 生：张浩凌

学　号：SY1706404

指导教师：李舟军、何跃鹰

2018年11月25日

1 论文选题的背景和意义 **4**
1.1 爬虫技术的复杂性与多样性 4
1.2 恶意爬虫的威胁 4
1.3 爬虫技术的更新迭代 5
1.4 攻击爬虫的可能性 6
1.5 反爬虫系统的意义 6

2 国内外研究现状及发展动态 **7**
2.1 爬虫识别技术研究现状 7
2.1.1 日志语法分析模式 7
2.1.2 流量分析模式 8
2.1.3 分析学习 8
2.1.4 图灵测试系统 9
2.2 浏览器指纹技术研究现状 9
2.2.1 浏览器特定指纹 9
2.2.2 canvas指纹 9
2.2.3 JavaScript 引擎指纹 10
2.2.4 跨浏览器指纹 10
2.2.5 webRTC技术 10

3 论文的研究内容及拟采取的技术方案 **10**
3.1 研究目标 10
3.2 研究内容 11
3.2.1 实时性的爬虫检测 11
3.2.2 基于session的爬虫检测 11
3.2.3 爬虫/浏览器的指纹生成与追踪 12
3.2.4 headless browser漏洞自动化挖掘 12
3.2.5 部署方案与评估方案 12
3.3 拟采取的技术方案 13
3.3.1系统构架 13
3.3.2 页面染色技术 14
3.3.3 链接染色技术 14
3.3.4 动态符号执行与模糊测试 15

4 关键技术 **16**
4.1 导向型自动随机化测试 16
4.2 EGT方法 16
4.3 动态污点分析 16
5 论文研究计划 **18**
6 主要参考文献 **19**

1 论文选题的背景和意义

1.1 爬虫技术的复杂性与多样性

在以数据为中心的信息时代，如何快速廉价地获取大量有效的数据，是维持高度数据依赖的组织的竞争力的关键。而在真实互联网环境，数据往往是不均匀地散落在以web为代表的互联网的各个角落中。为了更快更廉价的获取数据，很多互联网组织和个人开发了各式各样的网络爬虫（crawler）来抓取他们感兴趣的数据。爬虫在互联网数据传播过程中起到了至关重要的作用。

现有爬虫种类多，爬取的策略也具有很高的复杂性和多样性。爬虫一般可以按其特点分为四类：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。最常见的爬取策略是广度优先策略和深度优先策略。除此此外，比较常见的聚焦网络爬虫，甚至会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。

Paul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页面重要性的度量标准。Page-rank算法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[14]。Rennie和McCallum在他们的工作中[15]，将增强学习（reinforcement learning)引入聚焦爬虫，他们试图通过机器学习找到一组策略，使得对于该策略的激励达到最优解。Diligenti则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]，以此作为确定访问顺序的参考。

爬虫技术的复杂性和多样性也意味着反爬虫技术的复杂性。

1.2 恶意爬虫的威胁

网络中同样充斥着大量恶意的爬虫，他们要么多线程高并发地请求服务器[1]，要么抓取一些敏感的或者高价值的数据，转手将数据倒卖给商业竞争对手。因此，针对爬虫反制措施的重要性也凸显了出来。据资料显示，每年的三月份

，正处在在硕士需要撰写论文的时间节点，网络中充斥着大量应届毕业生写的劣质爬虫，导致很多数据业务相关的网站在三月份访问量大幅度增加。有一部分失控爬虫，不断地向相关网站发送请求，造成带宽的严重占用和流量资源的损耗。今年十月份爆出的马蜂窝评论数据涉嫌造假事件，其数据来源就是通过爬虫，从其竞争对手的网站上抓取的。在涉嫌欺诈的同时，也给其竞争对手带来大量的经济损失。

此外，一部分黑客在攻击网站之前，会通过通用爬虫爬取网站的所有可访问资源，并尝试对其中的参数进行模糊测试[6]。尽早发现带有恶意payload攻击的爬虫，能够在真实攻击发生的早期，进行有效的防范和加固。

1.3 爬虫技术的更新迭代

随着headless browser工具的兴起，以及以selenium为主的自动化web测试工具的出现，使得爬虫技术也从原始的静态网页处理方式，逐渐进化到能够与动态的javascript进行复杂交互的状态。这也使得大部分基于javascript脚本混淆（javascipt obfuscate）的反爬虫技术失去了原有的优势[5]。此外，大部分学术界提出的爬虫检测策略，往往基于流量日志，是非实时的（offline）检测机制，即不能对爬虫行为做出及时的反馈，也会在随后的离线分析中损耗大量的人力和时间成本。

此外，网络中充斥着匿名的网络代理节点，大量的爬虫会在爬取网站过程中不断更改访问的IP地址，使得原本针对IP封锁的方案往往达不到预期的效果，一旦误封正常用户的IP地址，可能造成普通用户群体的流失。即使是基于CAPTCHA验证码的访问验证，也可以通过打码平台的技术支持在一定程度上进行绕过，而且会影响到正常用户的访问体验。

随着爬虫技术的更新与迭代，以及反反爬虫技术的逐渐成熟，现有的反爬虫检测技术已经不能满足当下严峻形势的需求，而爬虫处理技术过于单一与被动，使得网站维护者们在反爬虫的战争中占尽劣势。

1.4 攻击爬虫的可能性

大部分的爬虫都是使用python进行编写的，其常用的数据存储方式一般采用传统数据库加文件持久化的方案[9]，在加载网页的过程中，还可能使用headless browser对爬取到的javascript进行解析。因此，基于以上的爬虫架构，针对于python语言特性，数据库乃至于javascript解析的传统攻击手段也有成功利用的可能性。基于这样的一种直觉性的判断，我们可以针对识别出爬虫的具体特征，如是否使用python编写，是否开启了javascript，可能使用的数据库类型（需要通过javascript进行内网扫描），这些特性信息均可以通过与爬虫之间的信息交互获得。进一步地，可以生成相应的攻击payload来尝试攻击爬虫系统以使之奔溃甚至获得爬虫宿主机的权限。

除了针对传统架构传统攻击方式之外，我通过观察发现，爬虫系统使用的headless browser，phantomjs占有很大的市场份额。而且，在phantomjs的github主页上，我们可以发现接近2000条issue，其中很多issue所展示出的问题，都是因为phantomjs在使用某些处理逻辑处理网页内容时，整个程序出现未知错误而奔溃，这些问题大部分没有被复现和修复。

因此，深入研究这些因为底层webdriver问题带来的收益也是巨大的。我们也许能够发现大量webdriver二进制程序的未知漏洞，这些漏洞能够时爬虫进程奔溃，但又不影响到正常用户的访问，这些漏洞甚至可以用于获取爬虫宿主机的权限。

1.5 反爬虫系统的意义

互联网时代，数据的价值无需置疑。爬虫技术与反反爬虫技术不断更新迭代，相应的反爬虫技术却一直停滞不前。很多优秀的反爬虫算法要么不具有实时性，要么需要大量的训练数据，要么还是停留在概念阶段，难以在实战中发挥作用。而传统反爬虫技术中爬虫处理和阻断部分过于单一与被动，不能对爬虫施加有效制裁和限制。因此，一个实时性，多样化，主动的反爬虫系统的实现将具有重大的意义，它能够扭转当前爬虫与反爬虫技术不对等的状况。

本文不仅包含完整反爬虫系统的设计和实现，还将提供相应的部署方案，并对反爬虫系统的爬虫识别漏检率，误检率，性能损失作出相应的评估。

2 国内外研究现状及发展动态

2.1 爬虫识别技术研究现状

目前主流的爬虫识别技术可以分为四大类：

日志语法分析模式（Syntactical log analysis），流量分析模式（Traffic pattern analysis），分析学习（Analytical learning techniques）以及图灵测试系统（Turing test systems）。

2.1.1 日志语法分析模式

Holloway以及keller提出了独立字段解析（individual field parsing）的方法[18]。他们通过比较HTTP请求中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。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。

Huntington等人则采用了多层次日志解析（multifaceted log analysis）的策略[19]：他们首先将请求robots.txt的访问者被标记为爬虫，这样的流量大概占总流量比例的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，通过从网络搜集的爬虫池的IP地址段，来标记来源于爬虫IP的访问。

2.1.2 流量分析模式

Geens等人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，并最终得出了相应的检测策略。

guo等人利用聚焦网络爬虫特性，提出了一种基于资源请求模式分析（resource request patterns）的爬虫检测方法[21]。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某种访问占比过多，则将访问者判定为爬虫。

在Duskin的文章中，他通过分析请求速率模式（query rate patterns）来区分正常访问和爬虫[22]。首先统计请求提交速率，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。然后再根据分析这些特性得到的结果来区分用户和爬虫。

2.1.3 分析学习

Tan和 Kumar利用爬虫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，并选择特征向量以及对数据进行标记。以该标记数据为学习数据使用 C4.5 决策树算法构造决策树（decision tree）。

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HTTP访问请求，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。

2.1.4 图灵测试系统

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。

2.2 浏览器指纹技术研究现状

通过访问者浏览器生成相应的指纹，可以在访问者在使用分布式代理的时候追踪访问者的会话。从而为爬虫检测提供准确的参考数据，也可以用于爬虫溯源以及爬虫行为分析。

2.2.1 浏览器特定指纹

ECKERSLEY根据浏览器的环境，通过一些网站收集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从简单静态的HTTP请求中得出，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直接的，但也有一部分特性是来源于浏览器细节。例如，有些浏览器禁用了javascript，那么他会使用video，plugins，fonts和supercookie的默认属性值。因此可以通过这一细节来推断出浏览器是否禁用了javascript。

2.2.2 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。

2.2.3 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，从而匹配识别出一个特定的浏览器类型及其版本。

2.2.4 跨浏览器指纹

Cao等人提出可以通过利用OS以及硬件层的特性[29]，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。

2.2.5 webRTC技术

webRTC全称为（web-based realtime 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，通过这些接口，可以获取浏览器或者爬虫的所有网络接口的IP，并以此生成具有较高信息熵的指纹。

3 论文的研究内容及拟采取的技术方案

3.1 研究目标

本文将从爬虫检测机制、浏览器指纹入手，设计爬虫识别模块和追踪模

块。通过针对爬虫使用的headless broswer二进制程序的漏洞挖掘，结合传统的攻击载荷，生成爬虫攻击模块，拼接实现完整的反爬虫系统。进一步地，将该爬虫系统部署到真实的业务系统，并测试其在挂载反爬虫系统后的识别率，误检率以及相应的性能变化，最终完成该反爬虫系统的设计，实现以及评估。

3.2 研究内容

3.2.1 实时性的爬虫检测

结合目前的实时反爬虫机制，并尝试实现一些论文中提出的，但仍没有实际应用的实时反爬虫机制。实时的爬虫检测机制，要求能够在爬虫爬取网页的过程中，实时地将其识别出来。正因为实时性的需要，所有的爬虫检测都是在用户访问网页的时候同步进行，所以实时性的爬虫检测可能会对服务器性能产生一定的影响。如何在尽量控制性能损失的情况下，尽可能提高爬虫的检测率，将是本文重点关注的对象。

3.2.2 基于session的爬虫检测

通过服务器端的session以及客户端的cookie，将一次浏览会话中请求的序列信息记录下来，包括请求的时间以及请求的页面内容等相关信息。在随后爬虫检测中，通过分析这些信息来判断某个会话是否来自于爬虫。

基于session的爬虫检测难点在于如何设立合适的阈值，既保证识别率又能降低误检率。此外，因为要保留会话的请求数据，在会话量较大，并发较高的场景下，如果使用服务端存储的策略，需要考虑到会话序列存储的长度，存储清空的时机等问题。而如果采用客户端存储的情况，需要设计相应的cookie防篡改机制。

3.2.3 爬虫/浏览器的指纹生成与追踪

指纹生成的算法比较多，但是为了指纹标记结果的准确性，需要通过多种方式计算指纹，那么这些指纹之间的主次关系该如何规定。此外，指纹算法可以用于分类会话，那么是否可以采用某些机制将指纹标记与基于session的爬虫检测机制耦合起来。进一步地，如何保证从客户端接受到的指纹是可信的，而不是任意伪造的。这些问题都在本文研究和探讨的范围之内。

3.2.4 headless browser漏洞自动化挖掘

使用动态符号执行处理较大的程序时，容易出现路径爆炸的问题。但是因为headless browser大部分的程序都是开源的，因此可以针对headless browser的一些关键部分的代码进行修改后重编译，生成轻量的二进制文件，再使用动态符号执行以及污点分析的方法。但是，如何选择合适的可能存在漏洞的代码片段，以及进一步对漏洞产生原因的分析，都需要较大的工作量。

3.2.5 部署方案与评估方案

设计过程中各个模块可能是分立，一旦这些模块组装成一个完整的系统，为了系统能够正常运作，还需要很多细微的调整。一个灵活性的部署方案也是值得深思的，部署系统尽量秉持不干扰到原先业务系统的原则，也不需要在原先的业务系统上做出太多的改动，以控制人力和时间成本。

在最终的评估中，需要设计评估的方案，即如何去模拟真实的网络环境中的正常访问与来自爬虫的访问。此外，设计用什么样的指标去无偏见地做出评价也是极为重要的。

3.3 拟采取的技术方案

3.3.1系统构架

反爬虫系统的架构图如下所示：

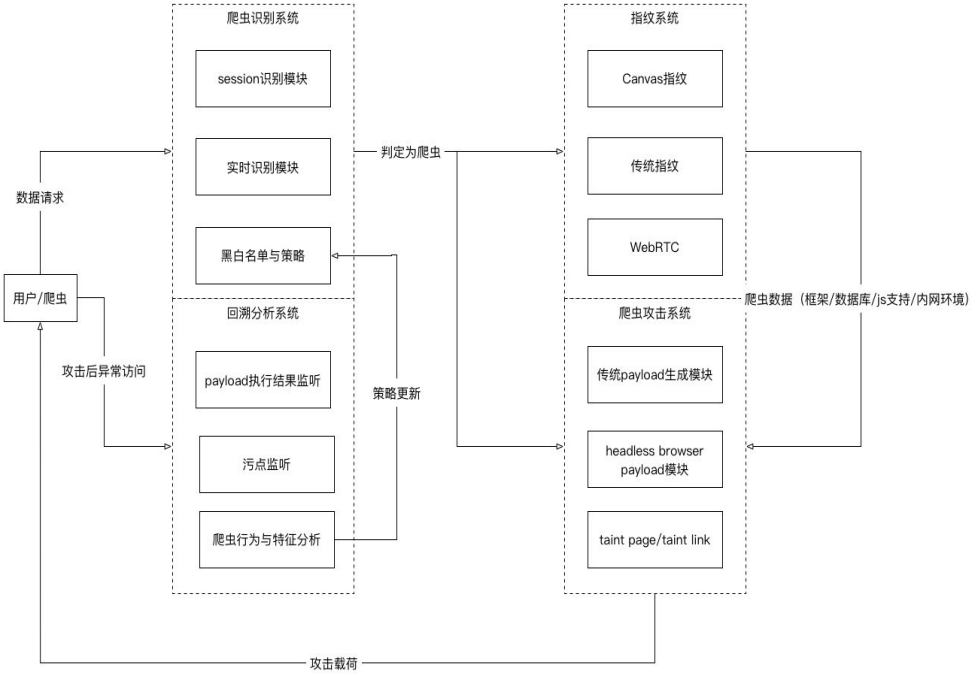

图 3.1 反爬虫系统的架构图

在该系统中，获得请求后，通过实时识别模块与session识别模块判断是否为爬虫请求。一旦被标记为爬虫请求，则使用指纹系统探测爬虫的相关信息，包括其使用的框架，后端数据库，js支持以及内网环境等信息。将这些信息提供爬虫攻击系统，用以生成合适的攻击载荷，加入到页面内容的返回中。一旦爬虫执行了恶意的攻击载荷，会主动访问回溯分析系统，并通过此系统收集爬虫的行为和特征，这些特征将用于更新识别系统中的黑白名单以及其他策略的配置。

3.3.2 页面染色技术

正常用户以及爬虫在访问一个页面时，他们处理页面内容的方式不一样，他们获取到的下一级链接信息也不一样。例如页面A中有页面B和页面C的链接，其中页面C通过javascript或者CSS的处理方式，使得其对用户不可见，但是爬虫仍然能够成功爬取并解析。我们将这种页面称之为染色页面（taint page），而所有访问了染色页面的访问者都可以被标记为爬虫。

3.3.3 链接染色技术

链接染色技术（taint link）可以用于对于某些敏感数据流向的溯源，目前主要的想法是一旦某个访问者或者某个会话被识别为爬虫，则使用染色链接污染数据。而相应的，该链接的路由会指向回溯和分析系统的相关功能模块。如果该敏感数据被爬虫爬取并使用在某商业竞争对手的页面上，一旦用户访问该页面，加载页面中的链接资源，会在HTTP header带上包含来源信息的referer字段，该字段将会被回溯和分析系统所记录。由此，我们可以回溯到敏感数据是被哪一个商业对手所恶意窃取，并被使用在什么业务上，这些信息将为法务部门的维权提供坚实的支持持。链接染色的原理图如下图所示：

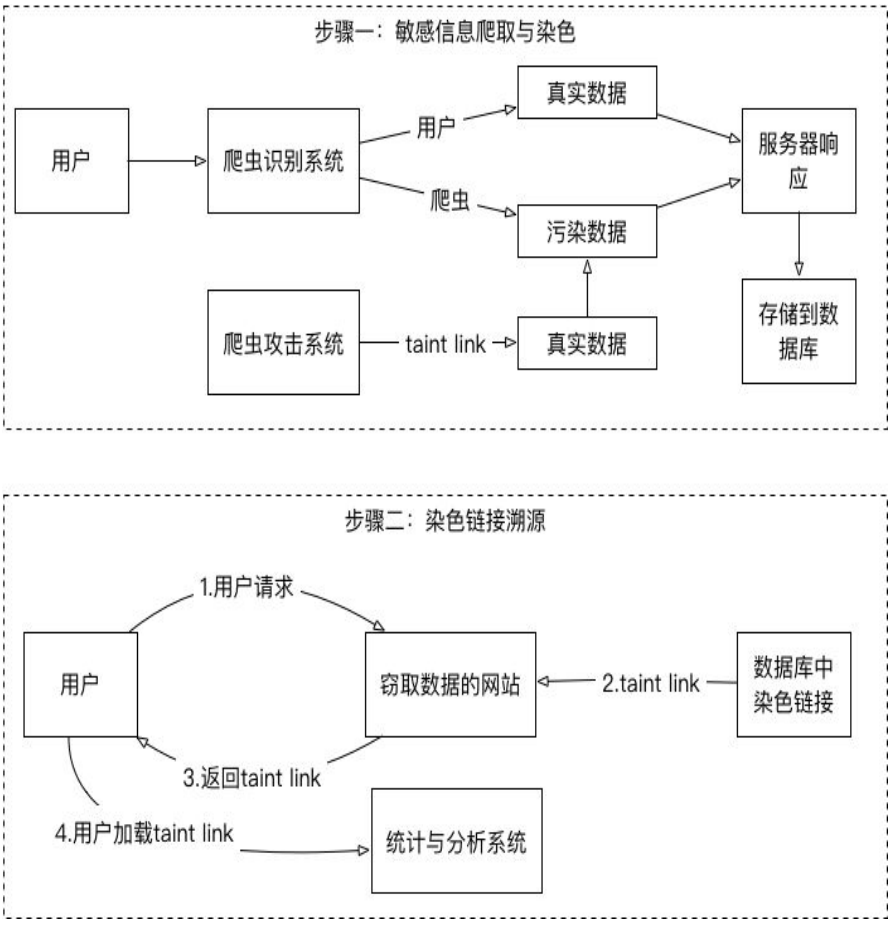

图 3.2 链接染色的原理图

3.3.4 动态符号执行与模糊测试

在漏洞挖掘部分，考虑到很多crash的触发点均处于较深的代码层次，仅仅做动态符号执行无法保证路径的覆盖率，因此，在动态符号执行的过程中引入模糊测试（fuzzing）。首先在给定输入的基础上，通过模糊测试变异输入，来发现能够到达的路径。当模糊测试无法发现更多的路径时，使用动态符号执行来约束求解以到达模糊测试未能覆盖的路径的输入。以此循环往复，直到程序退出。

4 关键技术

4.1 导向型自动随机化测试

一般称具体执行（concolic execution），或者导向型自动随机化测试。由Godefroid等人提出[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。

4.2 EGT方法

全称是execution-generated testing，被EXE工具和KLEE工具所采纳[35]，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。

4.3 动态污点分析

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]。动态污点分析最关键的部分是污染策略，而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。

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[38]。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。污点策略在不同的输入源类型中也会有所不同。

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。

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[98]中，我们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。

5 论文研究计划

2019.07.01-2019.08.31

设计和测试多种爬虫识别算法，包括实时的爬虫识别以及基于session的爬虫识别。

2019.09.01-2019.10.31

设计和测试浏览器指纹系统，并收集不同浏览器，不同爬虫的指纹。

2019.11.01-2019.11.30

尝试对常用的headless browser的二进制程序进行自动化漏洞挖掘。

2019.12.01-2020.1.31

根据已经发掘出的漏洞以及爬虫框架的传统漏洞，设计和实现爬虫攻击模块。

2020.02.01-2020.02.28

根据现有的模块，整合成完成的反爬虫系统，并部署到真实业务中。

2020.03.01-2020.03.31

设计对反爬虫系统的评估方案，并收集不同类型的爬虫代码。

2020.04.01-2020.04.30

利用不同的爬虫测试反爬虫系统，并在公网环境中测试效果，收集评估指标参数。对反爬虫系统进一步地优化。

2020.05.01-2020.05.31

撰写论文，准备答辩。

6 主要参考文献

[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.