软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007476] [CSTR: 32375.14.jos.007476]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 JavaScript 代码分析的漏洞扫描器增强框架*

况博裕[1], 朱 焱[1], 杨善权[2], 苏 铓[1,2], 周永彬[1], 付安民[1,2]

[1](南京理工大学 网络空间安全学院, 江苏 南京 210094)

[2](南京理工大学 计算机科学与工程学院, 江苏 南京 210094)

通信作者: 苏铓, Email: sumang@njust.edu.cn

摘 要: 黑盒漏洞扫描器是用于 Web 应用漏洞检测的重要辅助工具, 能够在 Web 应用正式上线前有效识别潜在的安全威胁, 从而提升 Web 应用的整体安全性. 当前大多数黑盒扫描器主要通过模拟用户操作和正则匹配来收集攻击面. 然而, 模拟用户操作容易被输入验证机制拦截, 且难以处理复杂的事件操作, 而正则匹配方法无法有效处理动态内容. 这导致扫描器难以有效处理 JavaScript 代码中的隐藏攻击面和动态生成的攻击面, 使其在部分 Web 应用中漏洞检测效果不佳. 为解决上述问题, 提出一种基于 JavaScript 代码分析的漏洞扫描器增强框架 JSEScan. 该框架结合静态与动态代码分析技术, 绕过表单验证和事件触发的限制, 通过提取 JavaScript 代码中攻击面的特征, 实现 JavaScript 代码中攻击面的挖掘, 并且攻击面将被同步至多种扫描器, 从而增强其漏洞检测能力. 实验结果表明, JSEScan 能将单个扫描器的代码覆盖量提高 81.02%–242.15%, 并且相比于多扫描器同时工作的情况, 额外发现 239 个安全漏洞, 具备更强的攻击面收集能力和漏洞检测能力.

关键词: Web 安全; 黑盒扫描器; JavaScript; 漏洞检测; 攻击面挖掘

中图法分类号: TP311

中文引用格式: 况博裕, 朱焱, 杨善权, 苏铓, 周永彬, 付安民. 基于JavaScript代码分析的漏洞扫描器增强框架. 软件学报. http://www.jos.org.cn/1000-9825/7476.htm

英文引用格式: Kuang BY, Zhu Y, Yang SQ, Su M, Zhou YB, Fu AM. Vulnerability Scanner Enhancement Framework Based on JavaScript Code Analysis. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7476.htm

Vulnerability Scanner Enhancement Framework Based on JavaScript Code Analysis

KUANG Bo-Yu[1], ZHU Yan[1], YANG Shan-Quan[2], SU Mang[1,2], ZHOU Yong-Bin[1], FU An-Min[1,2]

[1](School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China)

[2](School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China)

Abstract: The black-box vulnerability scanner is an essential tool for Web application vulnerability detection, capable of identifying potential security threats effectively before a Web application is launched, thus enhancing the overall security of the application. However, most current black-box scanners primarily collect the attack surface through user operation simulation and regular expression matching. The simulation of user operations is vulnerable to interception by input validation mechanisms and struggles with handling complex event operations, while regular expression matching is ineffective in processing dynamic content. As a result, the scanner cannot effectively address hidden attack surfaces within JavaScript code or dynamically generated attack surfaces, leading to suboptimal vulnerability detection in some Web applications. To resolve these issues, this study proposes a JavaScript Exposure Scanner (JSEScan), a vulnerability scanner enhancement framework based on JavaScript code analysis. The framework integrates static and dynamic code analysis techniques, bypassing form validation and event-triggering restrictions. By extracting attack surface features from JavaScript code, JSEScan identifies attack surfaces and synchronizes them across multiple scanners, enhancing their vulnerability detection capabilities. The experimental results demonstrate that JSEScan increases coverage by 81.02% to 242.15% compared to using a single scanner and uncovers an additional

* 基金项目: 国家自然科学基金 (62372236, 62402223); 国家资助博士后研究人员计划 (GZB20240982); 江苏省青蓝工程; 江苏省卓越博士后计划

收稿时间: 2025-02-24; 修改时间: 2025-04-21; 采用时间: 2025-05-26; jos 在线出版时间: 2025-09-28

239 security vulnerabilities when compared to multiple scanners working concurrently, showing superior attack surface collection and vulnerability detection capabilities.

Key words: Web security; black-box scanner; JavaScript; vulnerability detection; attack surface mining

自 Web 应用程序诞生以来, 经过数十年的发展, 其安全防护日益复杂和完善, 但 Web 应用程序仍然是互联网攻击的主要目标[1]. 根据我国国家信息安全漏洞共享平台的统计数据, Web 应用程序漏洞占比从 2014 年的 15% 增长至 2024 年的 50% 左右[2]. 为提高 Web 应用程序的安全性, 安全工程师通常会定期对其进行漏洞检测, 以发现并修复存在的安全漏洞, 从而降低潜在的安全风险. 然而, 手动漏洞检测不仅耗时耗力, 还需要检测人员具备丰富的专业经验. 为此, 安全工程师通常会使用自动化工具来辅助漏洞检测, 提高检测效率[3].

Web 漏洞扫描器是一种旨在辅助安全工程师对 Web 应用程序进行漏洞检测的工具. 根据工作机制的不同, Web 漏洞扫描器可以分为白盒漏洞扫描器和黑盒漏洞扫描器两类, 如图 1 所示. 白盒漏洞扫描器通过分析 Web 应用程序的源代码进行漏洞检测[4]. 这类扫描器一般通过静态分析源代码, 为 Web 应用程序建立数据流模型[5,6]并使用污点分析技术来检测漏洞, 或者生成可执行的代码片段[7]并利用模糊测试进而检测漏洞. 然而, 白盒漏洞扫描器往往难以被没有源码访问权限的第三方安全工程师所使用. 此外, 其开发和运行成本通常较高, 且会忽略一些与 Web 应用程序运行环境密切相关的漏洞. 不同于白盒漏洞扫描器, 黑盒漏洞扫描器在 Web 应用程序的外部进行漏洞检测, 它模拟用户与 Web 应用程序的交互, 通过构造特定的输入实现漏洞检测. 因此, 黑盒漏洞扫描器不依赖于源代码, 仅需正常访问 Web 应用程序即可工作, 这使得其适用范围更加广泛[8].

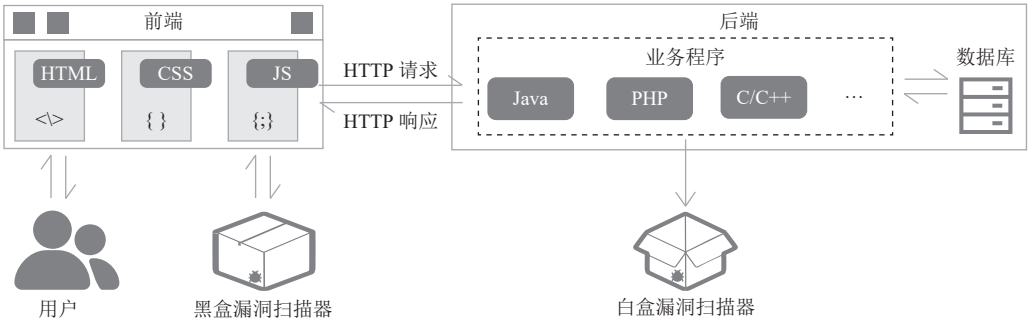

图 1 Web 应用程序结构与用户、黑盒漏洞扫描器和白盒漏洞扫描器的关系

典型的黑盒漏洞扫描器工作过程可分为 3 个阶段: 收集攻击面、发送攻击载荷和分析响应内容[9]. 黑盒漏洞扫描器首先模拟普通用户的操作遍历网页, 并解析页面中的链接和表单, 从而收集潜在的攻击面. 随后, 黑盒漏洞扫描器为每个攻击面构造多个不同类型的攻击载荷, 并通过 HTTP 数据包发送给 Web 应用程序. 最后, 黑盒漏洞扫描器分析 HTTP 响应内容, 以判断攻击载荷是否生效, 并检测是否存在漏洞. 目前, 一些商业化或者开源的黑盒漏洞扫描器, 如 BurpSuite、AWVS、Arachni、ZAP 等, 已经得到了广泛的认可与应用. 此外, 一些学者也在关注提升这些工具的漏洞检测能力. 例如, Zheng 等人[10]通过模拟用户行为并触发更多的事件, 从而扩展信息收集的范围, 以发现更多攻击面. Pellegrino 等人[11]通过捕获并拦截底层的请求发送函数, 从而发现更多攻击面, 提高扫描的覆盖率.

然而, 随着技术的不断进步, Web 应用程序已从传统的简单结构演变为功能丰富、特性复杂、接口多样化的系统[12,13]. 在实际使用过程中, 现有的黑盒扫描器在处理这类新型 Web 应用程序时仍面临多项挑战 (详细说明请见第 2 节).

(1) 动态生成的攻击面. 在动态网页, 尤其是在完全通过 JavaScript 动态加载内容的 Web 应用中, 一些黑盒漏洞扫描器难以从 Web 应用程序的 JavaScript 代码中识别攻击面, 从而无法进行有效的漏洞检测.

(2) 事件触发限制. 一些扫描器通过模拟用户操作来触发事件以发现攻击面, 但某些 Web 应用程序会将不同权限用户的接口交互操作代码统一存储在 JavaScript 中, 并仅为特定权限的用户配置相应的触发事件. 扫描器通常只能模拟单一用户的操作, 难以触发其他权限用户的事件.

(3) 表单验证问题. 许多新型 Web 应用程序增加了表单验证功能, 仅当输入的数据符合特定要求时才会执行相应操作. 然而, 扫描器通常按照预定的策略生成输入, 难以生成符合这类特性要求的输入, 从而无法收集到对应的攻击面信息.

为了增强现有黑盒漏洞扫描器对 JavaScript 中攻击面识别和漏洞检测能力, 本文提出一种基于 JavaScript 代码分析的漏洞扫描器增强框架 JSEScan (JavaScript exposure scanner). JSEScan 作为中间件介入到黑盒漏洞扫描器与 Web 应用程序之间的通信中, 捕获并分析 HTTP 数据包, 从中提取 JavaScript 代码. 通过静态与动态分析相结合的方式, JSEScan 挖掘出隐藏在 JavaScript 中的攻击面, 并将这些攻击面信息同步给黑盒漏洞扫描器, 从而提升其攻击面识别能力和整体漏洞检测效果. 本文的主要贡献具体如下.

(1) 设计一种结合静态和动态分析 JavaScript 代码攻击面的方法. 该方法使用抽象语法树解析代码, 从中提取攻击面相关特征代码片段, 绕过表单验证和事件触发的限制.

(2) 提出一个漏洞扫描增强框架 JSEScan, 能够在不同黑盒漏洞扫描器间同步包括 JavaScript 代码中隐藏攻击面在内的多种攻击面, 提高黑盒漏洞扫描器的覆盖范围, 进而增强黑盒漏洞扫描器的漏洞检测能力.

(3) 实验结果表明, 参与实验的 4 款扫描器在接入 JSEScan 后, 在所有测试目标中的代码覆盖量提高了 81.02%–242.15%, 有效地增强了黑盒漏洞扫描器的漏洞检测能力, 使其额外检测出 239 个漏洞.

本文第 1 节介绍 Web 应用中相关的扫描器. 第 2 节说明研究动机. 第 3 节介绍详细的设计. 第 4 节给出 JSEScan 对黑盒漏洞扫描器强化能力的验证实验与分析. 第 5 节总结全文.

1 相关工作

Web 应用程序可以分为前端和后端两个部分. 前端是用户在日常生活中接触到的网页界面, 提供可视化的交互功能; 后端则是服务器, 负责处理用户请求并提供所需的信息或内容. 前端主要由 HTML、CSS 和 JavaScript 组成, 其中 HTML 负责网页的内容结构, CSS 控制网页的样式布局, JavaScript 为网页添加交互性和动态效果. 后端通常使用 Java、PHP 等编程语言编写, 用于处理业务逻辑并与数据库进行交互. 因此, 从用户的角度, 只能看到 Web 应用的前端部分. 为了保障 Web 应用的安全性, 需要借助漏洞检测工具来识别和修复可能存在的漏洞. 一般来说, Web 应用程序的漏洞检测工具可分为白盒漏洞扫描器和黑盒漏洞扫描器, 白盒漏洞扫描器一般通过业务程序的源代码检测漏洞, 黑盒漏洞扫描器则是同用户一样通过前端与后端交互, 在 Web 应用的外部实现漏洞的检测.

1.1 白盒漏洞扫描器

白盒漏洞扫描器是一种通过分析 Web 应用程序源代码内部结构和逻辑, 从而实现漏洞检测的工具. 目前, 大部分白盒漏洞扫描器主要通过静态的数据流污点分析来实现漏洞检测, 并在不同编程语言上实现对应的白盒漏洞扫描器[14–16]. 为了提高分析的准确性, 一些研究[5,6]通过创建 Web 应用程序数据流的摘要信息, 利用上下文相关的字符串分析来细化污点分析, 进而更准确地检测出漏洞. 但这种方法需要对所有源代码都进行测试分析, 开销过于庞大. 为此, 一些研究以用户输入接口为起点进行测试. 如 Mai 等人[17]结合特定领域语言和自动化 Web 爬虫, 利用静态分析方法与动态数据收集相结合解决了 Web 安全测试中的预言机问题. Shi 等人[18]提出一种基于历史补丁分析的漏洞检测方法, 通过分析 Web 应用的历史补丁源代码, 评估修复方式的安全性, 从而识别潜在的漏洞. 王微微等人[19]则使用遗传算法生成测试用例, 更有效地完成 Web 应用的测试.

此外, 部分白盒漏洞扫描器的研究则专注于检测特定类型的漏洞. 例如, 针对文件上传漏洞, Huang 等人[7]使用静态程序分析从源代码中提取可执行代码模板, 并在本地对这些模板进行模糊测试以检测漏洞. 针对越权漏洞, Shen 等人[20]提出了一种基于自动突变配置方法, 通过探索多种可能的配置方向来解决访问权限问题. 针对对象注入漏洞, Park 等人[21]从源代码中提取函数和类的静态摘要, 得到可用的调用链, 然后收集 Web 应用程序的函数调用信息, 通过分析调用可能执行的函数检测对象注入漏洞. 针对 SQL 注入漏洞, Yuan 等人[22]将面向对象数据库扩展自动转换为语义等价的过程数据库扩展, 然后结合控制流图构建和污点分析技术检测漏洞. 针对 CRSF 漏洞, Pellegrino 等人[23]通过捕获 Web 应用的动态执行轨迹, 利用轨迹推断包含状态转换、数据流等信息的模型来识别

潜在的漏洞.

白盒漏洞扫描器虽然能够通过分析源代码来检测漏洞, 但其存在着一些显著的缺点. 首先, 白盒扫描器通常需要对整个源代码进行全面测试, 导致分析开销较高, 特别是在面对大型复杂的 Web 应用时, 这种开销通常难以被接受. 其次, 白盒扫描器对源代码的高度依赖性使其难以应用于无源码或第三方软件的漏洞检测. 此外, 由于编程语言和框架的多样性, 白盒扫描器需要针对不同应用编写特定的分析算法, 这显著增加了开发和维护成本. 最后, 一些与 Web 应用程序运行环境相关的漏洞, 往往难以被白盒漏洞扫描器发现.

1.2 黑盒漏洞扫描器

黑盒漏洞扫描器是一种无需访问应用程序源代码, 而是通过模拟外部攻击者的行为来检测 Web 应用程序漏洞的工具. 一般来说, 生成攻击载荷的质量决定了黑盒漏洞扫描器是否能发现漏洞, 而收集攻击面的数量则影响黑盒漏洞扫描器对扫描目标的代码覆盖量.

为了提高黑盒漏洞扫描器攻击载荷的质量, 研究者提出多种不同的方法. 例如, Eriksson 等人[24]通过字符串约束求解功能, 从 Web 应用的正则表达式中动态推断合适的输入, 以此提高攻击载荷通过数据校验的可能性. Rennhard 等人[25]通过修改扫描器的请求包字段, 为扫描器解决请求包因身份验证无法通过导致攻击载荷无效问题. Yin 等人[26]从扫描器发送和接收的数据包中提取攻击载荷, 将其同步给其他扫描器, 提高攻击载荷绕过 Web 应用防火墙检测的能力.

此外, Sigalov 等人[27]指出黑盒漏洞扫描器发现攻击面的能力与生成攻击载荷的能力同等重要, 如果无法发现攻击面, 即使生成的攻击载荷效果再强也无法检测漏洞. 为了提高黑盒漏洞扫描器收集攻击面的能力, Zheng 等人[10]利用采用好奇心驱动的强化学习来生成具有时间逻辑关系的高质量测试用例, 以增多收集的攻击面数量. Pellegrino 等人[11]提出编写 hook 函数捕获并替换 JavaScript 底层发送请求的函数, 以动态插桩的方法将其插入 Web 应用程序中, 以此捕获所有异步请求.

一些研究者也提出了多种针对特定类型漏洞设计的黑盒漏洞扫描器. 例如, 针对跨站脚本攻击漏洞, Eriksson 等人[28]提出通过构建页面导航模型表示页面之间的关系及用户输入相关的页面, 从而定位跨站脚本攻击漏洞可能出现的位置. Yao 等人[29]提出通过结合数据流分析、静态字符串分析和容错解析方法实现对跨站脚本攻击漏洞的检测. Lee 等人[30]利用强化学习来适应攻击有效载荷以发现漏洞, 在测试阶段根据当前环境状态选择动作来生成攻击有效载荷, 判断攻击载荷是否生效来检测跨站脚本漏洞. 针对逻辑漏洞, Kim 等人[31]提出通过代码分析确定与业务逻辑相关的候选函数, 并根据函数的多种特征进行排名, 依据排名和篡改策略生成篡改载荷, 定位篡改位置和操作来检测逻辑漏洞. Ali 等人[32]则记录和检查用户与目标网站的交互收集网站信息, 通过监控应用程序的执行来检测各种相关漏洞出现的可能性. 针对 SQL 注入漏洞, Aliero 等人[33]着重分析在同一页面中使用不同载荷访问时的区别, 以确定数据库查询数据的位置, 进而更精确地识别数据在前端的位置, 更好的识别漏洞.

因为重新开发一款黑盒漏洞扫描器的成本会随着 Web 应用程序中技术的进步越来越高, 因此提高现有扫描器的能力成为一种更具性价比的选择. 为了使扫描器能适应不断变化的技术环境并增强其漏洞检测能力, 研究人员提出了多种方法来增强现有扫描器. 例如, Drakonakis 等人[34]提出通过拦截并转发扫描器 HTTP 请求到浏览器中, 实现 Web 应用中动态内容的加载, 并记录攻击面生成导航图, 将其添加到响应中发送给扫描器, 以此提高扫描器的动态解析能力. Yin 等人[26]提出通过迭代执行多个扫描器的漏洞扫描任务, 利用前次扫描结果优化后续扫描过程, 从而提升所有扫描器的漏洞检测能力.

虽然黑盒漏洞扫描器能适用各种 Web 应用, 但是随着 Web 应用技术的发展, Web 应用程序已具备更多样化的页面生成方式, 事件触发方式日益复杂, 表单验证也更加严格. 现有的黑盒漏洞扫描器在处理动态网页时面临着多项挑战, 因此急需研究一款适用于动态网页的黑盒漏洞扫描增强器来弥补这些缺陷.

2 研究动机

提高黑盒漏洞扫描器的漏洞检测能力关键在于增强其攻击面发现能力和攻击载荷生成能力. 现有的研究通过

共享扫描器之间的信息或建立 Web 应用程序的状态转换关系来提高检测能力, 但是这些方法存在一个共同的问题: 忽略了黑盒漏洞扫描器在分析 JavaScript 能力上的不足, 尤其在处理动态网页时, 攻击面的覆盖率过低. 此外, 在漏洞检测的过程中, 大部分黑盒漏洞扫描器通过点击前端界面中的按钮、提交表单等简单操作触发事件, 或者通过使用正则匹配技术从 JavaScript 中寻找攻击面. 这些方法通常只能发现 JavaScript 代码中小部分简单直接且易于观测到的攻击面, 而大部分以各类方式隐藏于 JavaScript 代码中的攻击面则难以被检测. 代码 1 展示了包含多个 Web 应用攻击面的真实代码片段, 代码片段 1 为一个普通查询函数; 代码片段 2 为一个仅由管理员执行的订单查询函数; 代码片段 3 为一个由特定输入格式触发的查询按钮函数. 接下来, 我们以代码 1 为例, 介绍 3 类现有方案在攻击面发现方面的局限性.

代码 1. 隐藏在 JavaScript 中的攻击面示例.

```
1. // 代码片段 1
2. function getCity(country, province) { //查询函数
3.       fetch("/v1/api/city", {
4.             method: "POST",
5.             body: JSON.stringify({country: country, province: province})
6.       })
7. }
8. // 代码片段 2
9. base_api = "/v1/api/"; //定义所有接口的前置路径
10. function getOrderList(t, e) { //管理员查询用户的订单信息
11.       (0, s.default.get)(base_api + "users/" + t + "/orders",
12.             {limit: 10, offset: e}
13.       )
14. }
15. // 代码片段 3
16. button.onclick = function() { //为按钮绑定输入格式检测函数
17.       country = document.getElementById("country").value();
18.       province = document.getElementById("province").value();
19.       if (country > 100 && province.length < 5) { //检查输入格式是否有误
20.             alert("输入格式错误!")
21.       }
22.       else {
23.             getCity(country, province); //只有输入满足要求才调用查询
24.       }
25. }
```

● 动态生成的攻击面: 新型 Web 应用程序通常使用 JavaScript 以异步方式与服务器的数据接口交互, 并动态加载内容以提升用户体验. 这一设计使得 Web 应用的攻击面从传统的 HTML 表单和链接形式扩展到了 JavaScript 代码中的请求逻辑. 而在传统的 Web 应用中, 攻击面通常固定于 HTTP 响应内容的 HTML 表单中, 扫描器可以通过简单的内容解析直接识别这些攻击面. 而新型 Web 应用中, 隐藏在 JavaScript 代码中的攻击面信息分布在不同的位置, 通常需要特定的事件触发才能生成完整的攻击面. 即使扫描器通过正则匹配技术从 JavaScript 中寻找攻击面, 可以发现第 3 行的"/v1/api/city"信息, 但面对如第 11 行这类动态生成的信息, 正则匹配则只能提取其中少部

分的信息, 如“users/”“/orders”. 即使扫描器通过正则匹配找到了请求路径信息, 也无法找到对应攻击面的载荷. 这是因为黑盒漏洞扫描器中的正则匹配技术侧重于从特定格式的字符串中提取信息, 因此难以识别动态生成的攻击面, 也难以捕获其他方式表示的载荷的信息, 如第 5 和 12 行.

● 事件触发限制: 许多 Web 应用的攻击面存在于特定的事件逻辑中, 但黑盒扫描器通常依赖模拟普通用户的交互操作进行攻击面收集, 因此难以触发这些逻辑. 如函数 *getOrderList* (第 10–14 行) 是 Web 应用中管理员查询订单操作, 普通用户没有触发相关操作的事件, 而扫描器往往是作为一般用户进行攻击面收集, 难以触发该这类特殊事件触发的操作, 降低了扫描器的攻击面探测覆盖范围.

● 表单验证问题: 在许多 Web 应用中, 表单在提交前会进行输入格式检查. 如果输入格式不正确, 表单提交会被阻止, 导致扫描器无法捕获相应的攻击面. 例如, 代码片段 3 中第 19–24 行对用户的输入进行格式检查, 要求 *country* 不大于 100 且 *province* 的长度不小于 5 时, 才能执行查询函数 *getCity*, 否则就不执行. 黑盒漏洞扫描器在处理表单时通常难以生成满足这类特定条件的输入, 导致表单无法提交, 就无法捕获该攻击面.

为了解决上述问题, 我们通过对大量 Web 应用程序中包含攻击面的代码片段进行分析, 总结出一个完整的攻击面通常由以下 3 类关键的特征字段组成.

● 请求路径: 通过字符串表示或参与拼接得到的请求路径, 如第 3 行的“/v1/api/city”和第 11 行的 base_api + “users/” + *t* + “/orders”.

● 请求方式: 通过函数名或参数值表示的请求方式, 如第 4 行中的“POST”和第 11 行中“s.default.get”的函数名“get”.

● 有效载荷: 通过键值对表示的有效载荷, 如第 5 行的“{*country*: *country*, *province*: *province*}”和第 12 行的“{limit: 10, offset: *e*}”.

因此, 如果能够从 JavaScript 代码中提取出包含攻击面的代码片段, 并通过执行结果确定动态生成的信息, 从中提取请求路径、请求方式和有效载荷这 3 个特征就能准确构建出攻击面. 这种方式可以在不提交表单的情况下, 绕过代码中存在的表单验证逻辑, 并且也不需要依赖触发事件, 直接从代码中分析得到攻击面. 最后将攻击面同步给扫描器, 就能提高扫描器的攻击面发现能力, 进而增强其漏洞检测能力. 基于这一关键思路, 本文设计了一个基于 JavaScript 代码分析的漏洞扫描器增强框架 JSEScan.

3 框架设计

针对第 2 节中提出的问题, 本文设计并实现了一种静态与动态分析相结合的 JavaScript 代码黑盒漏洞扫描器增强框架 JSEScan, 系统结构图如后文图 2 所示. JSEScan 增强框架主要分为两个阶段: 攻击面挖掘和攻击同步. 在攻击面挖掘阶段, JSEScan 捕获扫描器与 Web 应用程序之间通信的 HTTP 数据包, 通过字段提取, 收集多种扫描器发现的攻击面和流量中的 JavaScript 代码, 然后通过 JavaScript 代码分析, 挖掘隐藏在 JavaScript 中的攻击面, 并将其存储到信息同步库中. 而在攻击同步阶段, 框架通过 HTTP 数据包定位扫描位置, 从信息同步库中提取相关的修改规则对 HTTP 请求包进行修改, 提取相关攻击面生成索引页面, 并将其注入 HTTP 响应包中, 以实现攻击同步, 扩展扫描器的扫描范围, 增强其漏洞检测能力.

3.1 攻击面挖掘

为实现 JSEScan 对不同黑盒漏洞扫描器的兼容, JSEScan 利用代理服务捕获并解析 HTTP 数据包, 获取不同扫描器在工作过程中的信息, 整体流程如图 3 所示. JSEScan 首先从 HTTP 数据包中提取字段, 收集不同黑盒漏洞扫描器发现的攻击面并提取 JavaScript 代码, 接着通过静态和动态结合分析 JavaScript 代码挖掘隐藏的攻击面, 最后对收集到的攻击面进行筛选, 得到扫描目标的信息同步库.

3.1.1 扫描器发现的攻击面信息收集

攻击面挖掘模块会捕获扫描器与 Web 应用间通信的 HTTP 请求包和响应包. 由于黑盒漏洞扫描器在短时间内可能会发送和接收大量数据包, JSEScan 在捕获这些数据包后, 会将 HTTP 请求包与响应包进行配对, 随后, 从

配对的 HTTP 数据包中提取出主机名、请求路径、有效载荷、响应状态码、响应类型和响应内容这 7 个字段, 如图 4 所示. 在完成字段提取后, 通过响应状态码判断扫描器请求的内容是否存在, 对于存在的内容, 进一步结合主机名、请求方式、请求路径和有效载荷这 4 个字段生成攻击面, 从而收集不同扫描器在 Web 应用程序上发现的攻击面信息.

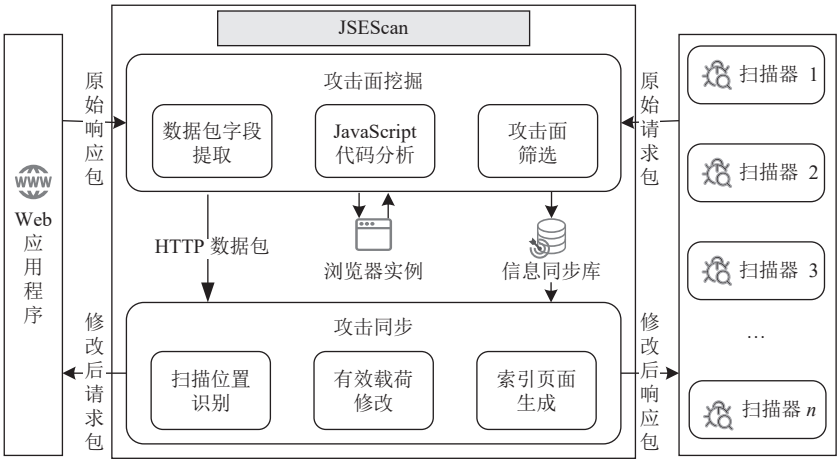

图 2　JSEScan 系统结构图

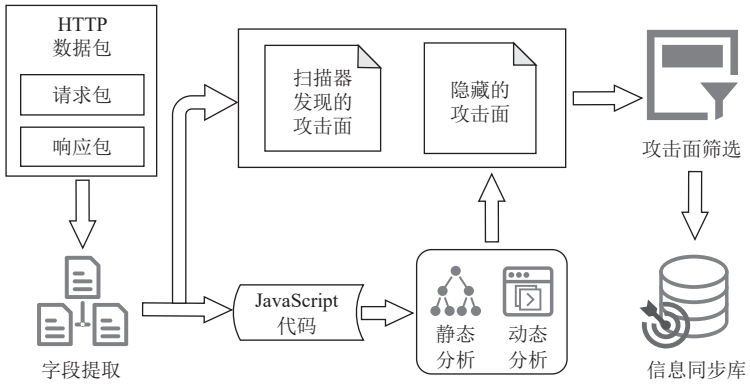

图 3　攻击面挖掘流程

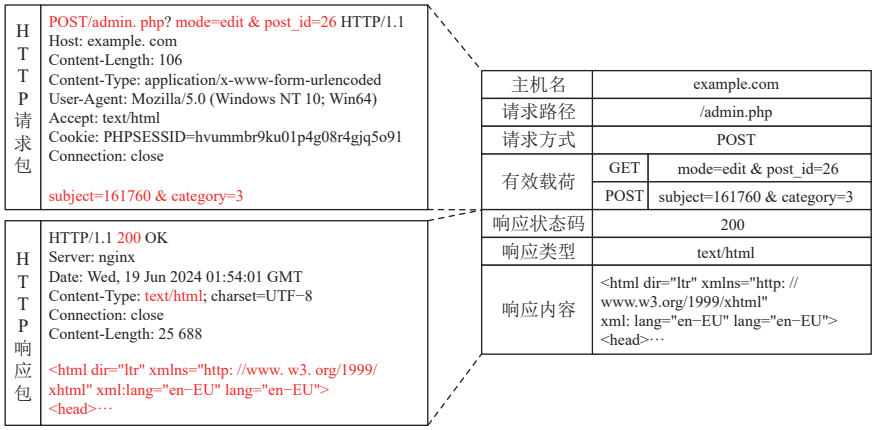

图 4　HTTP 数据包字段提取

3.1.2 JavaScript 代码提取

从上一步字段解析得到的响应内容中提取 JavaScript 代码. 由于 JavaScript 代码属于文本类型数据, 因此媒体、字体文件等非文本类型的响应内容不在分析范围内, 可通过响应类型排除. 对于 JavaScript 代码, 其主要包含两种形式: 一是 Web 应用程序中引入的 JavaScript 文件, 其响应内容都是代码, 因此直接将其添加到待分析列表中; 另一种是 HTML 文档, 该文档包含多个分布在不同位置的 JavaScript 代码片段, 这些代码之间可能存在调用关系. JSEScan 通过解析 HTML 标签提取出代码片段, 并将其拼接后添加到待分析列表中. 因为扫描器工作时会多次加载相同的 JavaScript 代码, 重复分析相同代码会浪费资源和时间. 因此, JSEScan 利用待分析列表存储为哈希表结构, 通过计算每段代码的散列值作为键值, 丢弃具有相同散列值的代码. 此外, Web 应用程序通常使用第三方 JavaScript 库来提供额外功能, 这些库的代码通常不涉及 Web 应用程序的核心数据逻辑处理. 然而, 对这些庞大库中代码的进行分析效率低下, 且难以挖掘出有效漏洞. 因此, JSEScan 通过识别常见标识符来辨别并排除这些第三方库[35].

3.1.3 JavaScript 代码攻击面挖掘

从待分析列表中的 JavaScript 代码中挖掘隐藏的攻击面. 为了能解决攻击面内容动态生成的问题需要解决两个关键挑战: 1) 得到 JavaScript 代码执行结果; 2) 调用 Web 应用程序中定义的函数. 挑战 1) 需要通过执行 JavaScript 代码获得语句的执行结果, 来预测攻击面可能的信息. 挑战 2) 需要通过调用 Web 应用中定义的函数, 来获取这些自定义的处理函数中可能的攻击面信息. 为了解决这两个挑战, JSEScan 在本地维持一个浏览器实例, 并通过远程调试的方法获得 JavaScript 代码的动态分析能力. 具体而言, JSEScan 在浏览器中访问目标网页, 然后使用远程调试技术在浏览器控制台中执行 JavaScript 代码获得执行结果. 且由于是在目标 JavaScript 的环境中执行代码, 因此可以调用目标自定义的函数. 此外, 这种方法还可以避免引入第三方库, 因为这些库已经包含在目标环境中. 另外, 不同的 Web 应用程序中的 JavaScript 代码可能有不同的编写规范和风格, 为了避免这种代码无关因素对分析造成的影响, JSEScan 会将代码转换成为抽象语法树的形式进行分析, 减少真实语法中的代码无关细节, 实现对语法的精确解析.

算法 1 具体描述了静态和动态分析结合的 JavaScript 攻击面挖掘算法, 其主要包含两个步骤: 静态分析和动态分析. 静态分析 (第 3–12 行) 提取变量相关和包含攻击面的代码片段, 然后动态分析 (第 13–24 行) 从通过静态分析得到的代码片段中提取特征, 构造攻击面并添加到 JavaScript 信息同步库中. 具体而言, 在静态分析阶段, 先将待分析集合中的代码转换成抽象语法树. 然后遍历树中的每个节点, 通过节点的类型判断其是否与变量的定义、赋值操作有关 (第 6 行), 并通过节点及其子节点中是否包含疑似请求路径的字符串判断节点是否可能包含攻击面, 最后将可能的节点转换回 JavaScript 代码 (第 8 行), 并添加到变量相关代码片段集合 *VariableCode* (第 7 行) 或包含攻击面的代码片段集合 *AttackSurfaceCode* (第 9 行). 在动态分析阶段, 首先遍历 *AttackSurfaceCode*, 通过 *InferValue* 方法推断代码片段中的变量的值并替换 (第 14–19 行). 其中 *InferValue* 是上文提到的通过在本地维持浏览器实例, 基于收集到的 *VariableCode*, 利用远程调试执行代码获得执行结果. 此外, 代码片段中可能存在部分依赖用户输入的变量, 这类变量无法通过 *VariableCode* 中的代码片段得到具体的值, 为保证含攻击面的代码片段不包含变量, 将这类变量赋值为带有特殊标识字符串表示 (第 20 行). 接着, 通过 *Execute* 方法执行不含变量的语句, 将赋值后的值回代到代码片段中. 最后, 从结果中提取请求路径、请求方式和有效载荷这 3 个特征, 并生成攻击面添加到 JavaScript 信息同步库 *JSAttackInfoSet* 中.

算法 1. 静态和动态分析结合 JavaScript 攻击面挖掘算法.

输入: *JSCodeSet*[] //从 HTTP 响应中提取的待分析 JavaScript 代码集合;
输出: *JSAttackInfoSet*[] //JavaScript 信息同步库.

1. *VariableCode*← ∅, *AttackSurfaceCode*← ∅; //初始化参数
2. *JSAttackInfoSet*← ∅;

```
3. FOR JSCode in JSCodeSet[] DO //静态分析
4.     JSASTTree←GetAst(JSCode);
5.     FOR node in JSASTTree DO;
6.         IF IsVariableRelated(node) THEN;
7.             VariableCode←ASTToCode(node); //提取变量相关代码片段
8.         ELSE IF IsIncludeAttackSufFeatures(node); THEN
9.             AttackSurfaceCode←ASTToCode(node); //提取含攻击面的代码片段
10.        END IF
11.    END FOR
12. END FOR
13. FOR SurfaceCode in AttackSurfaceCode DO //动态分析
14.    FOR variable in VariableCode DO
15.        IF IncludeVar(variable, SurfaceCode) THEN
16.            value←InferValue(variable);
17.            tempCode←ReplaceVar(variable, value, SurfaceCode);
18.        END IF
19.    END FOR
20.    exactCode←ReplaceVarToString(SurfaceCode);
21.    result←Execute(exactCode);
22.    features←ExtractFeatures(result);
23.    JSAttackInfoSet←ConstructedAttackSurface(features, result);
24. END FOR
25. RETURN JSAttackInfoSet
```

以代码 1 为例, 在静态分析阶段先收集变量相关的代码片段, 这里只有 base_api = “/v1/api/”, 然后将代码片段 1 和代码片段 2 可视为包含攻击面的两个代码片段. 因为代码片段 1 可以直接从中提取出请求路径“/v1/api/city”, 请求方式“POST”和有效载荷“{*country*: *country*, *province*: *province*}”, 因此使用这些信息构造攻击面添加到信息同步库中. 代码片段 2 中存在字符串拼接语句 base_api + “users/” + *t* + “/orders”, 并且语句中存在两个变量“base_api”和“*t*”. 对于“base_api”, 因为在静态分析阶段收集到其定义相关的代码, 因此将代码通过远程调试的方式执行, 得到执行结果“/v1/api/”. 而变量“*t*”并没有相关的定义语句, 将其视为由用户输入决定的值并赋值为特殊标识“JSEScan_t”. 此后, 将变量替换为赋值得到新的语句“/v1/api/” + “users/” + “JSEScan_t” + “/orders”, 然后再将新语句通过远程调试的方式执行, 得到拼接结果“/v1/api/ users/JSEScan_t/orders”, 并将其视为攻击面中的请求路径. 最后从“s.default.get”中提取请求方式“get”, 提取有效载荷“{limit: 10, offset: *e*}”并构造攻击面添加到 JavaScript 信息同步库中.

3.1.4 攻击面筛选

信息同步库中的攻击面来源于两个步骤: 1) 通过捕获 HTTP 请求包和响应包, 收集各种扫描器发现的攻击面; 2) 从响应内容中的 JavaScript 代码中挖掘出隐藏的攻击面. 然而, 这之中可能包含许多重复或错误的攻击面, 如果直接将未经处理的信息同步库同步给扫描器, 则会显著降低扫描器的扫描效率. 为此, JSEScan 设计了一种筛选算法, 旨在从信息同步库中筛选出真正的攻击面, 如算法 2 所示. 针对从扫描器请求中提取的攻击面, 先计算每个攻击面对应响应内容的哈希值来判断它们之间的相似性, 去除哈希值相同的攻击面 (第 3–11 行). 值得注意的是, 由于页面的响应内容中常包含时间戳、日期等动态元素 (通常其长度较短), 会影响对相似页面的区分. 因此, JSEScan 会先移除响应内容中的 HTML 标签, 仅保留标签内的文本信息, 然后利用特定符号 (如句号) 将文本信息分割成多个片段, 按长度将片段排序, 选择长度在前 10% 的片段拼接成新的内容, 再计算其哈希值. 针对分析 JavaScript 得

到的攻击面, JSEScan 会计算每个攻击面的权重并保留权重较高的攻击面. 具体来说, 由于在同一个 Web 应用中, 请求的发送代码通常有相似的结构和相同的标识符, 所以先统计所有代码片段中各个标识符出现次数作为其初始权重 (第 12 行). 然后, 检查每个标识符的字面量是否包含如 ajax、fetch 这类与请求强相关的特征, 如果包含则将标识符的权重调高 (第 13–17 行). 最后累加代码片段中标识符的权重作为攻击面的权重, 并保留权重较高的攻击面到信息同步库中 (第 18–27 行).

算法 2. 攻击面筛选算法.

输入: *RawAttackInfo*[] //从扫描器请求中提取的和分析 JavaScript 得到的攻击面信息;
输出: *AttackfaceSet*[] //用于增强扫描的信息同步库.

```
1. RecordedSurfaces[], JSSurfaces[], Identifiers[]←RawAttackInfo; //初始化参数
2. Attackfaces← ∅, HashCount← ∅, jsSurWeight← ∅;
3. FOR recordInfo in RecordedSurfaces[] DO //筛选从扫描器请求中提取的攻击面
4.     hash←CaculateContentHash(recordInfo);
5.     IF ExistIdenticalHash(hash, HashCount) THEN
6.         CONTINUE;
7.     ELSE
8.         RecordHash(hash, HashCount);
9.         Attackfaces←recordInfo;
10.    END IF
11. END FOR
12. identifiersWeight←CountInCode(Identifiers, JSSurfaces); //计算标识符初始权重
13. FOR identifier in Identifiers DO //调整标识符权重
14.     IF IncludeFeatures(identifier) THEN
15.         IncreaseWeight(identifier, identifiersWeight);
16.     END IF
17. END FOR
18. FOR jsInfo in JSSurfaces DO //计算攻击面权重
19.     jsSurWeight←CaculateSurWeight(jsInfo, identifiersWeight);
20. END FOR
21. avgWeight←CaculateAvg(jsSurWeight)
22. FOR jsInfo in JSSurfaces DO //筛选分析 JavaScript 得到的攻击面
23.     IF AboveAvg(jsInfo, jsSurWeight, avgWeight) THEN
24.         Attackfaces←jsInfo;
25.     END IF
26. END FOR
27. AttackfaceSet[]=(Attackfaces, HashCount);
28. RETURN AttackfaceSet
```

3.2 攻击同步

攻击同步阶段将攻击面挖掘阶段收集到的信息同步库共享给多种现有的扫描器, 以判断攻击载荷是否生效以及是否存在漏洞. 由于现有扫描器一般通过分析响应内容来收集攻击面信息, 因此 JSEScan 通过修改 HTTP 数据包的方式, 利用信息同步库生成包含相关攻击面索引的页面, 并将其添加至原始响应中, 从而实现攻击信息

的同步, 如图 5 所示. 此外, JSEScan 还会修改原始请求中的载荷, 提高扫描器请求的有效性, 确保攻击信息的完整同步.

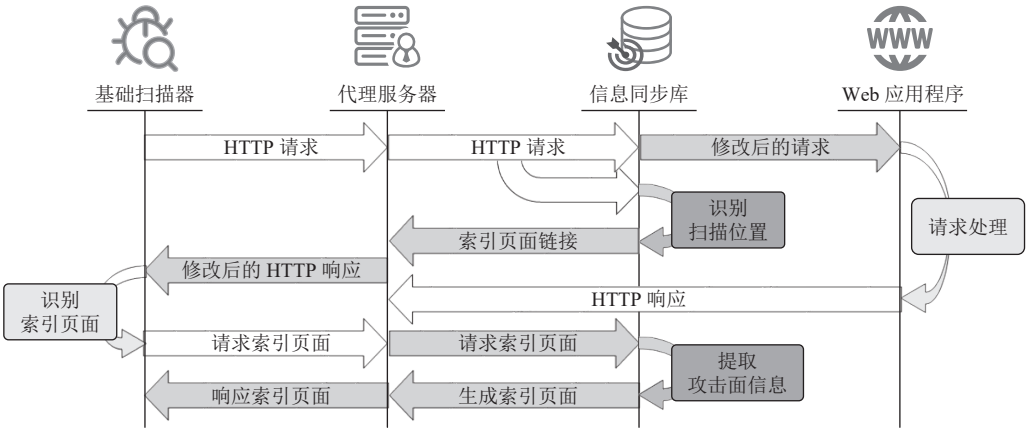

图 5 攻击同步流程图

具体而言, JSEScan 首先通过拦截和修改 Web 应用程序对 robots.txt 请求响应的方式, 与扫描器同步攻击面范围. 大多数扫描器在漏洞检测的初始阶段, 会尝试访问目标的 robots.txt 文件. 该文件是 Web 应用程序为爬虫等工具设定的访问指南, 明确哪些网页可以被访问, 包含 Web 应用的结构信息, 主流的扫描器一般也选择将其作为探索目标网站完整结构的起点. 因此, JSEScan 通过拦截 Web 应用程序对 robots.txt 请求的响应, 从信息同步库中提取所有攻击面的请求路径, 并将其转换成 robots.txt 文件中的数据格式, 附加到原始的 robots.txt 文件响应后, 进而实现将攻击范围同步给基础扫描器, 使扫描器能够快速地掌握目标结构.

然而, 依靠 robots.txt 文件只能将大致的扫描范围传达给扫描器, 而无法提供更为详尽的信息. 为同步更详细的信息, JSEScan 会进一步通过 HTTP 请求中的请求路径来确定扫描器当前的检测位置, 并将该位置相关的攻击面详细信息转换成特定格式, 添加到一个自定义的索引页面中. 接着, JSEScan 将指向索引页面的链接附加到响应内容中, 从而实现对攻击面的实时同步. 值得注意的是, 如果直接将攻击面信息添加到响应内容中可能会导致扫描器对同一个攻击面使用相同的有效载荷进行多次请求, 造成攻击面信息发生偏移, 影响扫描器的判断. 因此, JSEScan 使用了索引页面这种简洁的攻击面同步方式, 将攻击面信息的变化局限在索引页面中, 从而减少攻击同步对扫描器漏洞判断的干扰.

此外, 针对不同的攻击面类型, 具体包括 GET 类攻击面、POST 类攻击面和其他类攻击面, JSEScan 需要将其转换成不同的 HTML 元素并添加到索引页面中. 针对 GET 类攻击面, 其有效载荷直接体现在请求路径之后, 这类攻击面可以通过链接的形式表示. 且在 HTML 规范中, 链接一般会设置为具有 href 属性的标签. 因此, 针对每个 GET 类攻击面, JSEScan 会生成一个相应的含 href 属性的标签表示. 针对 POST 类攻击面, 在 HTML 中, 通常使用表单来发送 POST 请求, 而有效载荷中一般会嵌入到表单中以输入标签的方式传递, 因此 JSEScan 会将 POST 类的攻击面转换成表单来同步攻击面给扫描器.

其他类攻击面是指那些扫描器自身无法通过常规方式 (如链接、表单等) 构造正确请求的攻击面. 这些攻击面通常包含扫描器无法识别位置的载荷或无法构造正确有效的载荷格式. 例如, JSEScan 从代码 1 的片段 2 中提取的攻击面信息, 如图 6 所示, 真实的请求路径为“/v1/api/users/t/orders”, 其中“t”属于路由参数. 若直接将此类路径信息同步给扫描器, 扫描器会将其视为一个完整的 URL, 而无法将“t”识别为有效载荷. 针对这类其他攻击面, JSEScan 设计了特定的预处理机制进行处理, 其流程如图 6 所示. 首先, JSEScan 为其他类攻击面生成一个唯一标识 ID (如“JSEScanAttackFace0”), 并为需要处理的载荷分配一个标识关键字 (如“var1”). 随后, JSEScan 通过标识 ID 和载荷关键字生成请求路径 (如“/JSEScanAttackFace0?var=ZAP_183703”), 并将其以表单的形式同步至扫描

器. 扫描器解析表单后发送对应的 HTTP 请求, JSEScan 拦截会拦截这些请求, 并通过识别请求路径中的标识 ID, 判定其属于其他类攻击面. 最后 JSEScan 从攻击同步库中提取相关信息, 对请求进行修改, 将路径“/JSEScanAttackFace0”替换为真实路径“/v4/user/$var1$/orders”, 并对有效载荷“var1=ZAP_183703”进行替换, 得到正确的请求路径“/v1/api/users/ZAP_183 703/orders”. 通过上述步骤, 基础扫描器能够以传统表单的方式解析攻击面并进行漏洞检测, 从而解决基础扫描器无法生成正确格式请求包的问题.

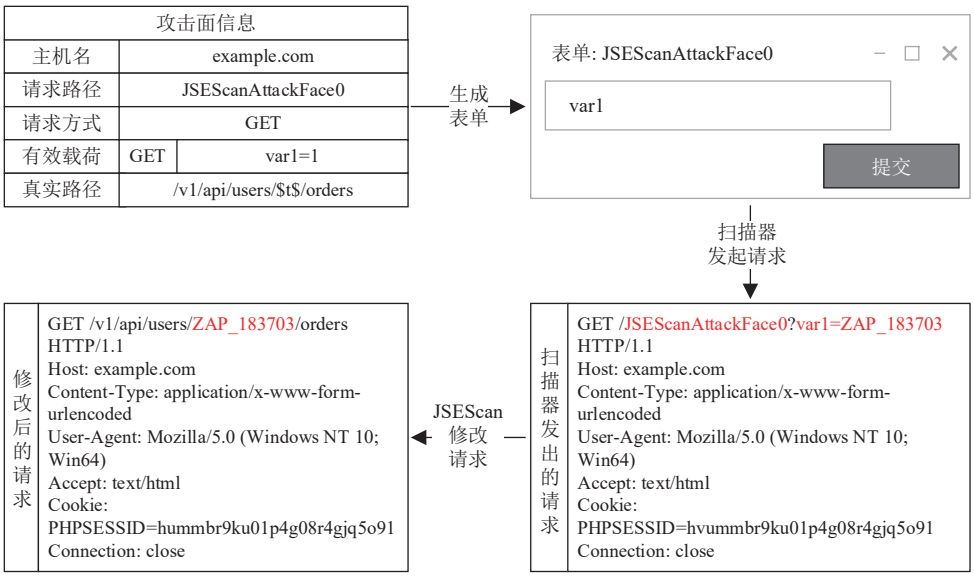

图 6 其他类攻击面的处理流程图

最后, 当请求路径对应的攻击面不在攻击范围内时, JSEScan 会将请求拦截, 并生成一个响应状态码为 404 的 HTTP 响应包发回给扫描器, 以此向扫描器传达当前请求路径对应文件不存在的信息, 从而让扫描器放弃对该攻击面进行扫描, 控制扫描器的扫描范围.

4 实验分析

JSEScan 是一个通过 JavaScript 代码分析挖掘攻击面来增强现有黑盒漏洞扫描器的方案, 本文的实验主要分为 5 个部分: 性能分析、代码覆盖量对比实验、漏洞检测能力对比实验、消融实验和真实应用漏洞挖掘实验, 以验证 JSEScan 从 JavaScript 代码中挖掘隐藏的攻击面的能力以及增强现有扫描器的能力.

4.1 实验设置

(1) 实验环境

本次实验选取了 10 个功能类型各异的网站作为测试环境[26], 如表 1 所示. 这些目标涵盖了软件应用网站 (SeaCMS)、学校管理网站 (SchoolMate)、博客网站 (myBloggie)、电子商务网站 (osCommerce)、在线公告网站 (Geccbblite)、个人网站 (Elemata)、扫描器能力测试网站 (Wackopicko)、在线象棋网站 (WebChess)、学术调查论坛 (SCARF) 和答疑网站 (FAQforge), 以验证 JSEScan 在不同场景下的适用性.

同时, 本次实验选择了实际中常用的 4 款 Web 应用程序扫描器 BurpSuite、AWVS、Arachni 和 ZAP 作为基础扫描器, 以验证 JSEScan 对扫描器的增强效果. 其中 BurpSuite 是一款专为渗透测试设计的综合工具, 以代理拦截和流量操控为核心; AWVS 是一款企业级全自动化漏洞扫描器, 以高覆盖率和智能爬虫技术为核心; Arachni 是一款基于 Ruby 开发的开源漏洞扫描器, 以轻量高效和高度可定制化著称; ZAP 是由 OWASP 社区维护的开源 Web 安全工具, 具备极高的灵活扩展性. 由于 JSEScan 主要通过修改 HTTP 数据包来实现攻击面收集和同步, 因此

对于可利用的基础扫描器, 都可使用 JSEScan 增强框架的接口连接. 对于扫描器的选择, 我们认为应覆盖动态请求分析 (主动扫描)、被动流量监控、爬虫遍历、模糊测试、API 端点测试、认证与会话测试这 6 种主流的攻击面发现方法, 以挖掘更多潜在的漏洞, 提升扫描器的覆盖率. 此外, 我们还与一个先进的黑盒扫描器增强框架 Scanner++[26] 进行了对比实验, 以显示 JSEScan 框架的优越性. Scanner++通过收集扫描器流量中的请求路径和载荷作为攻击面信息, 并同步给其他扫描器, 从而增强了扫描器的漏洞检测能力.

表 1　单独扫描器 (A)、JSEScan (J) 的代码覆盖行数和两者共同覆盖的代码行数对比

网站	BurpSuite			AWVS			Arachni			ZAP		
	A	A∩J	J	A	A∩J	J	A	A∩J	J	A	A∩J	J
SeaCMS	633	631	**2003**	1547	1542	**2117**	599	599	**1976**	594	594	**1895**
SchoolMate	165	165	**1448**	637	623	**1502**	700	684	**1468**	345	345	**1474**
myBloggie	2702	2701	**3434**	3022	2998	**3260**	2890	2741	**3132**	3030	2977	**3545**
osCommerce	5178	4560	**6972**	6612	4220	**6779**	5331	5010	**5809**	5028	4612	**13994**
Geccbblite	105	105	**253**	243	240	**245**	122	117	**243**	257	252	**266**
Elemata	170	170	**852**	497	494	**979**	386	386	**979**	170	170	**960**
Wackopicko	870	843	**1292**	1119	1037	**1326**	962	941	**1264**	851	837	**1276**
WebChess	110	110	**846**	142	138	**853**	142	138	**817**	110	110	**782**
SCARF	373	373	**747**	743	694	**762**	577	553	**738**	384	384	**746**
FAQforge	691	691	**731**	643	640	**792**	583	542	**674**	719	710	**764**
平均提升率 (%)	—	—	+242.15	—	—	+81.02	—	—	+116	—	—	+197.12

(2) 评价指标

本次实验主要通过扫描器的代码覆盖量、漏洞检测数量和攻击面发现数量来评价 JSEScan 的效果.

代码覆盖量指的是黑盒扫描器在漏洞检测过程中对 Web 应用覆盖的代码行数. 对于 PHP 等支持动态代码生成的编程语言, 本文借鉴了文献 [34] 中的方法, 采用扫描器在检测过程中实际触发并执行的服务器端 Web 应用程序代码行数作为衡量基准, 而非简单地依据执行代码占总代码的比例, 这是因为实景场景中存在大量不会被执行的代码 (如初始化代码仅在首次启动时执行). 在本文实验中, 我们通过服务器端的调试模式追踪并记录代码的执行行数, 以此作为评估扫描器覆盖量的依据.

漏洞检测数量是衡量黑盒扫描器整体能力的重要指标. 这一数量不仅反映扫描器在探测攻击面广度上的能力, 更体现了其构造高效攻击载荷的有效性. 为确保评估的准确性和可靠性, 本文对扫描报告中的每一个漏洞进行了手动验证. 根据漏洞的具体类型和有效载荷的潜在位置, 通过构造相应的攻击载荷来验证漏洞的真实性和可利用性.

攻击面发现数量是指扫描器在审计过程中识别并记录的攻击面的数量, 这一指标直接反映了扫描器的检测范围和分析能力. 通过对比不同扫描器发现的攻击面发现数量, 可以验证 JSEScan 能否成功将挖掘到的攻击信息同步给基础扫描器, 同时确认基础扫描器是否能够在这些扩展范围中有效识别攻击面. 此外, 攻击面发现数量还为安全测试人员提供了评估扫描器覆盖范围的依据, 能够直观展现不同扫描器在识别潜在威胁面方面的优劣.

4.2　性能分析

时间开销: JSEScan 主要的时间开销取决于基础扫描器的运行时间, 而 JSEScan 漏洞扫描器增强框架的时间复杂度为 $O(n_1)$, 其中 n_1 为 Web 应用前端的 JavaScript 代码行数. 此外, 我们在第 4.5 节具体展示了 JSEScan 面向 10 个不同 Web 应用时的时间开销.

存储开销: 除基础扫描器所必需的存储开销外, JSEScan 漏洞扫描器增强框架仅为 2.21 MB, 这部分开销主要用于 JavaScript 代码分析和实现不同基础扫描器之间的信息交互. 而 JSEScan 漏洞扫描器增强框架的运行期间存储开销复杂度可以表示为 $O(\mu n_2)$, 其中 n_2 为扫描器发现攻击面的数量, 包括基础扫面发现的攻击面数量和 JavaScript 代码挖掘出的攻击面数量. μ 表示攻击面的复杂程度, 一些简单的攻击面 (如某些不包含有效载荷的 GET 类攻击面), 可能只需记录一个简单的请求路径, 对应的 μ 值也就较小; 而复杂的攻击面, 可能不仅需存储攻击路径, 还需

存储在不同位置的有效载荷及其处理规则, 占用的存储空间也就相对较多, 对应的 μ 值也就较大. 此外, 我们在第 4.5 节具体展示了 JSEScan 面向 10 个不同 Web 应用时的存储开销.

4.3 代码覆盖量对比实验

首先, 我们进行了 JSEScan 对单个黑盒扫描器的代码覆盖量增强实验. 实验对比了 4 款基础扫描器单独扫描和基础扫描器接入 JSEScan 后的代码覆盖量, 结果如表 1 所示 (加粗数据表示最优结果). 结果表明, 将单个扫描器接入 JSEScan 后, 所有测试目标的代码覆盖量均有显著提升, BurpSuite、AWVS、Arachni 和 ZAP 的平均覆盖量分别提高了 242.15%、81.02%、116% 和 197.12%.

增强后的扫描器不仅全面覆盖了原有扫描范围的大部分区域, 还进一步深入到更多关键代码段. 对统计数据进行手动分析后发现, 这些新覆盖的代码区域主要集中于一些更为核心和关键的功能模块, 例如在 osCommerce 中, JSEScan 从 JavaScript 代码中挖掘到后台管理页面的攻击面, 将 ZAP 的代码覆盖量从 5 028 行提高到 13 994 行. 在 WebChess 目标中, 因为大部分操作都与 JavaScript 代码有关, 现有扫描器仅能覆盖 100 多行代码, 而通过 JSEScan 增强后能覆盖 782–853 行代码, 说明 JSEScan 成功从 JavaScript 代码中挖掘到了更多的攻击面并同步给扫描器.

其次, 为进一步验证 JSEScan 框架的优势, 我们进行了 JSEScan 增强 4 款黑盒扫描器的代码覆盖量的对比实验. 实验统计了 4 款扫描器同时接入 Scanner++和 JSEScan 后代码覆盖量和提升率, 实验结果如表 2 所示 (加粗数据表示最优结果). 对比实验数据表明, Scanner++使代码覆盖量提升了 60.92%, 而 JSEScan 则提升了 100.38% 这一优势主要源于 JSEScan 框架不仅能够同步扫描器已发现的攻击面, 还具备对 Web 应用前端 JavaScript 代码进行分析的能力, 从而挖掘更多潜在攻击面并显著增强扫描器的代码覆盖量.

表 2　4 个扫描器总和、Scanner++和 JSEScan 的代码覆盖行数对比

网站	4款扫描总和覆盖量	Scanner++		JSEScan	
		覆盖量	提升率 (%)	覆盖量	提升率 (%)
SeaCMS	1 547	1 862	20.36	**2 236**	**44.54**
SchoolMate	759	1 421	87.22	**1 597**	**110.41**
myBloggie	3 562	3 458	−2.92	**3 719**	**4.41**
osCommerce	7 524	9 537	26.75	**14 316**	**90.27**
Geccbblite	257	**266**	**3.50**	**266**	**3.50**
Elemata	497	694	39.64	**988**	**98.79**
Wackopicko	1 124	1 516	34.88	**1 634**	**45.37**
WebChess	142	658	363.38	**931**	**555.63**
SCARF	758	937	23.61	**986**	**30.08**
FAQforge	783	883	12.77	**946**	**20.82**
合计	16 953	21 232	25.24	**27 619**	**62.92**
平均提升率	—	—	60.92	—	**100.38**

4.4 漏洞检测能力对比实验

为了验证 JSEScan 的漏洞检测能力, 我们进行了 JSEScan 对单个黑盒扫描器的漏洞检测能力增强对比实验, 相关结果如表 3 所示 (加粗数据表示最优结果). 结果表明, 在较大型的 Web 应用中如 SeaCMS 中, 经过 Scanner++增强的 BurpSuite 扫描器能额外发现 7 个漏洞, 而经过 JSEScan 增强后能额外发现 18 个漏洞, 这一趋势在其他扫描器中也得到了验证. 这种增长的原因在于, 较大型 Web 应用通常会将更多的攻击面信息嵌入在 JavaScript 代码中, JSEScan 通过挖掘 JavaScript 中的攻击面并将其同步至扫描器更具优势, 能更好提升扫描器的检测效果. 从整体数据来看, BurpSuite、AWVS、Arachni 和 ZAP 这 4 款扫描器接入 JSEScan 后, 分别能额外发现 112、86、113 和 226 个安全隐患, 这表明 JSEScan 能有效补充传统扫描器的漏洞检测盲区, 并一定程度弥补扫描器在攻击面发现和漏洞检测能力上的不足, 从而提升整体漏洞检测效果.

表 3　由单独扫描器 (Baseline)、扫描器接入 Scanner++ (S) 或 JSEScan (J) 后发现的漏洞数量

网站	BurpSuite			AWVS			Arachni			ZAP		
	Baseline	S	J	Baseline	S	J	Baseline	S	J	Baseline	S	J
SeaCMS	37	44	**55**	10	15	**18**	5	42	**44**	1	20	**32**
SchoolMate	4	18	**24**	2	20	**21**	2	11	**12**	2	11	**23**
myBloggie	4	15	**21**	11	17	**25**	2	6	**9**	1	25	**25**
osCommerce	4	8	**9**	0	3	**5**	0	2	**8**	0	40	**45**
Geccbblite	4	9	**10**	1	10	**12**	4	10	**9**	1	6	**13**
Elemata	1	5	**5**	8	8	**10**	1	8	**16**	1	29	**32**
Wackopicko	3	8	**9**	5	5	**6**	4	5	**8**	2	2	**6**
WebChess	4	10	**11**	10	13	**14**	11	12	**12**	6	8	**9**
SCARF	1	17	**18**	17	20	**22**	6	8	**14**	6	35	**38**
FAQforge	14	19	**26**	2	10	**19**	2	10	**18**	2	22	**25**
合计	76	153	**188**	66	121	**152**	37	114	**150**	22	198	**248**

类似的, 我们进行了 JSEScan 联合 4 款黑盒扫描器的漏洞检测能力增强对比实验. 实验统计了 4 款扫描器同时在不同测试目标中发现的漏洞数量, 并与 Scanner++和 JSEScan 联合的结果进行对比, 具体数据如表 4 所示 (加粗数据表示最优结果). 4 款扫描器在所有测试目标中共发现 140 个已知漏洞, 而 JSEScan 则发现 379 个, 相比 4 款扫描器多发现了 239 个已知漏洞, 证明了 JSEScan 能够显著增强现有扫描器的漏洞检测能力. 此处新发现的 239 个漏洞主要来源于两个方面: 1) JSEScan 通过对 Web 应用程序中 JavaScript 代码中的攻击面进行扫描, 进而发现更多漏洞; 2) JSEScan 漏洞扫描器增强框架通过对不同基础扫描器发现的攻击面进行同步, 进而挖掘出更多漏洞. 此外, 结合表 2 的结果, 可以看出不同扫描器在各种场景下的表现差异性较大, 而 JSEScan 能够有效实现扫描器间的信息同步, 平衡不同扫描器的检测能力.

表 4　4 个扫描器、Scanner++和 JSEScan 联合发现的漏洞数量

网站	4个扫描器漏洞检测数	Scanner++		JSEScan	
		漏洞检测数量	提升数	漏洞检测数量	提升数
SeaCMS	45	73	+28	**79**	**+34**
SchoolMate	6	32	+26	**36**	**+30**
myBloggie	15	40	+25	**45**	**+30**
osCommerce	4	47	+43	**53**	**+49**
Geccbblite	6	**16**	**+10**	**16**	**+10**
Elemata	8	32	+24	**35**	**+27**
Wackopicko	6	10	+4	**11**	**+5**
WebChess	13	16	+3	**20**	**+7**
SCARF	20	50	+30	**51**	**+31**
FAQforge	17	29	+12	**33**	**+16**
合计	140	345	+205	**379**	**+239**

4.5 消融实验

信息同步库是生成索引页面和确定扫描范围的关键. 如果攻击面过多且扫描范围过广, 会导致扫描器进行大量重复工作, 从而降低扫描效率. 攻击面筛选算法是提升 JSEScan 性能的核心, 因此本实验通过对比启用和未启用攻击面筛选算法时同步的攻击面发现数量、存储攻击面的开销和最终的扫描器时间, 来证明攻击面筛选算法的有效性, 结果如表 5 所示. 在所有测试目标中, JSEScan 共收集 7046 个攻击面, 其中包括重复和无效的攻击面. 使用攻击面筛选算法后, 攻击面发现数量减少到 2139, 减少幅度达 69.64%. 这表明 JSEScan 的攻击面筛选算法能显著减少攻击面的数量. 此外, 实验还对比了筛选前后扫描的存储开销和时间开销. 启用筛选算法后, 存储开销最大可减少 95.15%, 平均减少 54.00%, 这主要取决于攻击面减少的数量和冗余攻击面的复杂程度. 扫描时间最多可减少 84.45%, 平均减少 56.48%. 其中, 减少时间较多的目标如 Elemata 和 Geccbblite, 是因为 JSEScan 在筛选完攻击面

后, 将原本的单页面请求路径转换为不同的请求路径, 使扫描器能够像扫描多页面程序一样工作, 从而提高对不同攻击面的识别能力, 并提升扫描效率.

表 5 筛选前后攻击面发现数量和扫描器性能对比

网站	未筛选					筛选后					攻击面减少百分比 (%)	存储开销减少百分比 (%)	扫描时间减少百分比 (%)
	GET 类型	POST 类型	其他类型	存储开销 (MB)	扫描时间 (s)	GET 类型	POST 类型	其他类型	存储开销 (MB)	扫描时间 (s)			
SeaCMS	529	130	878	160.7	22 579	66	47	54	127.2	11 433	89.13	20.85	49.36
SchoolMate	454	193	134	45.4	28 189	68	62	13	27.7	10 106	81.69	38.99	64.15
myBloggie	125	33	62	66.8	28 574	71	30	31	35.1	14 388	40	47.46	49.65
osCommerce	952	77	159	223.0	40 710	751	73	86	95.8	24 718	23.40	57.04	39.28
Geccbblite	66	40	12	5.7	27 195	40	10	5	2.5	5 756	53.39	56.14	78.83
Elemata	821	68	180	109.2	35 036	160	55	36	5.3	5 448	76.52	95.15	84.45
Wackopicko	359	58	1 285	127.5	38 843	115	21	137	17.8	17 620	83.96	86.04	54.64
WebChess	43	15	5	18.5	11 429	19	9	2	11.3	9 262	52.38	38.92	18.96
SCARF	72	35	18	214.4	22 480	49	13	9	125.1	14 338	43.20	41.65	36.22
FAQforge	198	18	27	6.5	17 755	76	17	14	1.9	5 662	59.96	70.77	68.11
合计		7 046		977.7	272 790		2 139		449.7	118 731	69.64	54.00	56.48

4.6 真实应用漏洞挖掘实验

我们将 JSEScan 应用于开源企业网站开发管理系统 PbootCMS (https://www.pbootcms.com/) 等 10 多个真实 Web 应用的漏洞检测中. 具体来说, 我们利用 JSEScan 首先分析 JavaScript 代码挖掘隐藏的攻击面, 并将现有的漏洞扫描器接入 JSEScan 进行攻击面同步, 对测试目标进行全面扫描, 并由人工验证和确认扫描结果. 最终结果显示, JSEScan 不仅能够分析出 JavaScript 代码中潜在的攻击面, 还通过攻击面同步的方式, 进一步加强了基础扫描器漏洞检测的能力, 实现了对深层漏洞的挖掘. 特别的, 我们基于 JSEScan 成功挖掘出了多个未知漏洞, 其中已获得一个跨站请求伪造 CNVD 漏洞编号 (CNVD-2023-03852). 该漏洞发现于 PbootCMS (一款采用 MVC 架构模式开发的企业级内容管理系统, 因其高效、简洁、开放等特点, 被广泛应用于在企业官网、个人博客/自媒体、政府/学校网站和小型电商的网站建设中). 攻击者能够伪装成管理员, 获得系统信息的操作权限 (如删除普通用户、文章内容和日志记录, 或修改文章内容和置顶信息), 甚至还可以通过修改管理员的账户密码获得后台管理权限.

5 讨 论

5.1 手机应用程序漏洞扫描

JSEScan 是面向 Web 应用程序的漏洞扫描器增强框架, 通过提取 JavaScript 代码中的隐藏攻击面, 同时在现有的漏洞扫描器间实现攻击面信息同步, 进而提升漏洞挖掘检测的能力. 因此, 正如现有的 Web 漏洞挖掘工作[25–34]和商用工具 (如 BurpSuite、AWVS、Arachni 和 ZAP) 一样, JSEScan 难以直接实现面向手机应用程序的漏洞扫描工作, 仅能通过代理拦截 HTTP 流量, 检测后端 API 和 WebView 等相关的 Web 漏洞. 但若需全面覆盖移动端风险 (包括原生代码漏洞、移动端特有风险或非 HTTP 协议漏洞等), 未来可以考虑结合静态分析工具 (如 MobSF)、动态插桩工具 (如 Frida) 和组件扫描工具 (如 Dependency-Check) 等, 形成完整的手机应用程序漏洞扫描.

5.2 混淆 JavaScript 代码

JSEScan 提出的攻击面挖掘方法在处理动态加载和复杂逻辑方面表现优异, 但对混淆 JavaScript 代码的适应性仍有局限性. 混淆技术本身其实是一种代码防御的手段, 通过重命名、字符串编码等方式, 破坏代码的可读性, 进而干扰分析过程. 目前 JSEScan 仅能针对未被混淆的 JavaScript 代码进行分析, 未来可以考虑结合逆向解混淆工具, 或者通过 AST 还原等方法针对混淆代码设计专门的解析与还原技术, 从而提高 JSEScan 对混淆代码的适应

能力.

5.3 加密或编码数据分析

在现代 Web 应用中, 为了满足保护敏感数据、对抗中间人攻击或者数据压缩等需求, HTTP 请求包与响应包有时会进行加密或编码处理, 而 JSEScan 还无法解析此类数据. 且由于无法识别未经解密或解码的请求包, 导致扫描器发送的数据包无效. 未来可以考虑在 JSEScan 的基础上, 研究加密和编码数据的分析与处理方法, 结合现有的分析工具 (如 CyberChef), 进而增强 JSEScan 在面向加密或编码 HTTP 请求包与响应包时的漏洞扫描能力.

6 总 结

本文提出了一种基于 JavaScript 代码分析的漏洞扫描器增强框架 JSEScan. 针对现有黑盒漏洞扫描器难以绕过载荷数据检测和忽略 JavaScript 代码中的隐藏的、动态生成的攻击面问题, 提出了基于静态和动态结合分析 JavaScript 代码的方法, 通过抽象语法树和代码执行技术, 能够有效绕过载荷的数据检测, 并从 JavaScript 代码中提取出攻击面信息. 此外, JSEScan 将攻击面信息以索引页面的方式注入到 Web 应用发送给黑盒扫描器的 HTTP 响应包中, 显著提升了黑盒扫描器收集攻击面和检测漏洞的能力. 实验结果表明, 相比于多扫描器同时工作的情况, JSEScan 能够将代码覆盖量平均提高 100.38%, 且额外发现 239 个安全漏洞.

References

[1] Wang XX, Liu QX, Liu CG, Zhang FJ, Liu XY, Cui XX. Survey of Web tracking. Journal of Computer Research and Development, 2023, 60(4): 839–859 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.202110681]

[2] China National Vulnerability Database. Statistical data from China National Vulnerability Database. 2024 (in Chinese). https://www.cnvd.org.cn/flaw/statistic

[3] Kuang BY, Zhang ZB, Yang SQ, Su M, Fu AM. HMFuzzer: A human-machine collaboration-based firmware vulnerability mining scheme for IoT devices. Chinese Journal of Computers, 2024, 47(3): 703–716 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2024.00703]

[4] Güler E, Schumilo S, Schloegel M, Bars N, Görz P, Xu XY, Kaygusuz C, Holz T. Atropos: Effective fuzzing of Web applications for server-side vulnerabilities. In: Proc. of the 32nd USENIX Security Symp. Philadelphia: USENIX Association, 2024.

[5] Luo CH, Li PH, Meng W. TChecker: Precise static inter-procedural analysis for detecting taint-style vulnerabilities in PHP applications. In: Proc. of the 2022 ACM SIGSAC Conf. on Computer and Communications Security. Los Angeles: ACM, 2022. 2175–2188. [doi: 10.1145/3548606.3559391]

[6] Huang YH, Shi CH, Lu J, Li HF, Meng HN, Li L. Detecting broken object-level authorization vulnerabilities in database-backed applications. In: Proc. of the 2024 ACM SIGSAC Conf. on Computer and Communications Security. Salt Lake: ACM, 2024. 2934–2948. [doi: 10.1145/3658644.3690227]

[7] Huang J, Zhang JJ, Liu JL, Li C, Dai R. UFuzzer: Lightweight detection of PHP-based unrestricted file upload vulnerabilities via static-fuzzing co-analysis. In: Proc. of the 24th Int'l Symp. on Research in Attacks, Intrusions and Defenses. San Sebastian: ACM, 2021. 78–90. [doi: 10.1145/3471621.3471859]

[8] Deng GL, Zhang ZY, Li YK, Liu Y, Zhang TW, Liu Y, Yu G, Wang DJ. NAUTILUS: Automated RESTful API vulnerability detection. In: Proc. of the 32nd USENIX Conf. on Security Symp. Anaheim: USENIX Association, 2023. 313.

[9] Xiao F, Su ZF, Yang GL, Lee W. Jasmine: Scale up JavaScript static security analysis with computation-based semantic explanation. In: Proc. of the 2024 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2024. 296–311. [doi: 10.1109/SP54263.2024.00183]

[10] Zheng Y, Liu Y, Xie XF, Liu YP, Ma L, Hao JY, Liu Y. Automatic Web testing using curiosity-driven reinforcement learning. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 423–435. [doi: 10.1109/ICSE43902.2021.00048]

[11] Pellegrino G, Tschürtz C, Bodden E, Rossow C. jÄk: Using dynamic analysis to crawl and test modern Web applications. In: Proc. of the 18th Int'l Symp. on Research in Attacks, Intrusions, and Defenses. Kyoto: Springer, 2015. 295–316. [doi: 10.1007/978-3-319-26362-5_14]

[12] Malik J, Pastore F. An empirical study of vulnerabilities in edge frameworks to support security testing improvement. Empirical Software Engineering, 2023, 28(4): 99. [doi: 10.1007/s10664-023-10330-x]

[13] Karami S, Ilia P, Polakis J. Awakening the Web's sleeper agents: Misusing service workers for privacy leakage. In: Proc. of the 28th

Network and Distributed System Security Symp. The Internet Society, 2021.

[14] Wang EZ, Chen JJ, Xie W, Wang CH, Gao YF, Wang ZH, Duan HX, Liu Y, Wang BS. Where URLs become weapons: Automated discovery of SSRF vulnerabilities in Web applications. In: Proc. of the 2024 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2024. 216–216. [doi: 10.1109/SP54263.2024.00198]

[15] Chen XC, Wang BZ, Jin Z, Feng Y, Li XL, Feng XC, Liu QX. Tabby: Automated gadget chain detection for Java deserialization vulnerabilities. In: Proc. of the 53rd Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks. Porto: IEEE, 2023. 179–192. [doi: 10.1109/DSN58367.2023.00028]

[16] Mao TY, Wang XY, Chang R, Shen WB, Ren K. Software supply chain analysis techniques for Java ecosystem. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2628–2640 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6852.htm [doi: 10.13328/j.cnki.jos.006852]

[17] Mai PX, Goknil A, Pastore F, Briand LC. SMRL: A metamorphic security testing tool for Web systems. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering: Companion Proc. Seoul: IEEE, 2020. 9–12.

[18] Shi YK, Zhang Y, Luo TH, Mao XY, Yang M. Precise (un)affected version analysis for Web vulnerabilities. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 76. [doi: 10.1145/3551349.3556933]

[19] Wang WW, Li YC, Zhao RL, Li Z. Parallel test case generation based on front and back end of Web applications with genetic algorithm. Ruan Jian Xue Bao/Journal of Software, 2020, 31(5): 1314–1331 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5955.htm [doi: 10.13328/j.cnki.jos.005955]

[20] Shen BY, Shan TY, Zhou YY. Multiview: Finding blind spots in access-deny issues diagnosis. In: Proc. of the 32nd USENIX Security Symp. Anaheim: USENIX Association, 2023. 7499–7516.

[21] Park S, Kim D, Jana S, Son S. FUGIO: Automatic exploit generation for PHP object injection vulnerabilities. In: Proc. of the 31st USENIX Security Symp. Boston: USENIX Association, 2022. 197–214.

[22] Yuan Y, Lu YL, Zhu KL, Huang H, Yu L, Zhao JZ. A static detection method for SQL injection vulnerability based on program transformation. Applied Sciences, 2023, 13(21): 11763. [doi: 10.3390/app132111763]

[23] Pellegrino G, Johns M, Koch S, Backes M, Rossow C. Deemon: Detecting CSRF with dynamic analysis and property graphs. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security. Dallas: ACM, 2017. 1757–1771. [doi: 10.1145/3133956.3133959]

[24] Eriksson B, Stjerna A, De Masellis R, Rüemmer P, Sabelfeld A. Black ostrich: Web application scanning with string solvers. In: Proc. of the 2023 ACM SIGSAC Conf. on Computer and Communications Security. Copenhagen: ACM, 2023. 549–563. [doi: 10.1145/3576915.3616582]

[25] Rennhard M, Esposito D, Ruf L, Wagner A. Improving the effectiveness of Web application vulnerability scanning. Int'l Journal on Advances in Internet Technology, 2019, 12(1–2): 12–27. [doi: 10.21256/zhaw-17956]

[26] Yin ZJ, Xu YW, Ma FC, Gao HH, Qiao L, Jiang Y. Scanner++: Enhanced vulnerability detection of Web applications with attack intent synchronization. ACM Trans. on Software Engineering and Methodology, 2023, 32(1): 7. [doi: 10.1145/3517036]

[27] Sigalov D, Gamayunov D. Dead or alive: Discovering server HTTP endpoints in both reachable and dead client-side code. Journal of Information Security and Applications, 2024, 82: 103746. [doi: 10.1016/j.jisa.2024.103746]

[28] Eriksson B, Pellegrino G, Sabelfeld A. Black widow: Blackbox data-driven Web scanning. In: Proc. of the 2021 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2021. 1125–1142. [doi: 10.1109/SP40001.2021.00022]

[29] Yao Y, He JJ, Li T, Wang YP, Lan XL, Li Y. An automatic XSS attack vector generation method based on the improved dueling DDQN algorithm. IEEE Trans. on Dependable and Secure Computing, 2024, 21(4): 2852–2868. [doi: 10.1109/TDSC.2023.3319352]

[30] Lee S, Wi S, Son S. Link: Black-box detection of cross-site scripting vulnerabilities using reinforcement learning. In: Proc. of the 2022 ACM Web Conf. Lyon: ACM, 2022. 743–754. [doi: 10.1145/3485447.3512234]

[31] Kim IL, Zheng YH, Park H, Wang WH, You W, Aafer Y, Zhang XY. Finding client-side business flow tampering vulnerabilities. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: IEEE, 2020. 222–233.

[32] Ali MM, Ghasemisharif M, Kanich C, Polakis J. Rise of inspectron: Automated black-box auditing of cross-platform electron APPs. In: Proc. of the 33rd USENIX Security Symp. Philadelphia: USENIX Association, 2024. 775–792.

[33] Aliero MS, Ghani I, Qureshi KN, Rohani MF. An algorithm for detecting SQL injection vulnerability using black-box testing. Journal of Ambient Intelligence and Humanized Computing, 2020, 11(1): 249–266. [doi: 10.1007/s12652-019-01235-z]

[34] Drakonakis K, Ioannidis S, Polakis J. ReScan: A middleware framework for realistic and robust black-box Web application scanning. In: Proc. of the 30th Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2023.

[35] Khodayari S, Pellegrino G. JAW: Studying client-side CSRF with hybrid property graphs and declarative traversals. In: Proc. of the 30th

USENIX Security Symp. USENIX Association, 2021. 2525–2542.

附中文参考文献

[1] 王晓茜, 刘奇旭, 刘潮歌, 张方娇, 刘心宇, 崔翔. Web 追踪技术综述. 计算机研究与发展, 2023, 60(4): 839–859. [doi: 10.7544/issn1000-1239.202110681]

[2] 国家信息安全漏洞共享平台. 国家信息安全漏洞共享平台统计数据. 2024. https://www.cnvd.org.cn/flaw/statistic

[3] 况博裕, 张兆博, 杨善权, 苏铠, 付安民. HMFuzzer: 一种基于人机协同的物联网设备固件漏洞挖掘方案. 计算机学报, 2024, 47(3): 703–716. [doi: 10.11897/SP.J.1016.2024.00703]

[16] 毛天宇, 王星宇, 常瑞, 申文博, 任奎. 面向 Java 语言生态的软件供应链安全分析技术. 软件学报, 2023, 34(6): 2628–2640. http://www.jos.org.cn/1000-9825/6852.htm [doi: 10.13328/j.cnki.jos.006852]

[19] 王微微, 李奕超, 赵瑞莲, 李征. Web 应用前后端融合的遗传算法并行化测试用例生成. 软件学报, 2020, 31(5): 1314–1331. http://www.jos.org.cn/1000-9825/5955.htm [doi: 10.13328/j.cnki.jos.005955]

作者简介

况博裕, 博士, 副教授, CCF 专业会员, 主要研究领域为漏洞挖掘, 物联网安全, 车联网安全.

朱焱, 硕士生, 主要研究领域为 Web 安全, 漏洞挖掘.

杨善权, 博士生, 主要研究领域为物联网安全, 漏洞检测.

苏铠, 博士, 副教授, CCF 专业会员, 主要研究领域为云计算, 访问控制.

周永彬, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为网络与信息安全理论及技术.

付安民, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为数据安全, 密码学, 隐私保护.