

基于有限状态机的漏洞基准测试集生成算法

谢元栋, 颜学雄, 赵光胜, 赵旭, 邱菡
(网络空间安全教育部重点实验室, 河南 郑州 450001)

摘要: 随着各类源代码漏洞挖掘技术的出现与发展, 如何评估各种工具的漏洞挖掘能力是一个重要问题。源代码漏洞基准测试集是用于评估面向源代码的漏洞挖掘工具能力的源代码程序集, 是开展评估工作的重要基础。已有的源代码漏洞基准测试集大多数对模拟真实程序代码的复杂性和多样性重视不足, 导致漏洞挖掘工具在基准测试集与真实程序代码中的表现差异很大。并且, 现有源代码基准测试集未提供对漏洞辨别率的测试支撑, 使得工具评估缺失一个重要维度。针对上述问题, 对漏洞抽象三元组及可利用漏洞链进行了研究, 提出了一种基于有限状态机的源代码漏洞基准测试集生成算法。首先建立源代码基准测试集有限状态机模型, 构建基本代码路径; 其次利用条件编辑距离选择辨别率指标专用路径; 最后使用代码圈复杂度包装器对路径元素进行包装, 以模拟现实代码的复杂结构。实验结果表明, 该算法生成的漏洞基准测试集与真实软件复杂性更加接近, 能够发现漏洞挖掘工具在代码路径上存在的偏向性和不足, 证明了其有效性和实用性。

关键词: 有限状态机; 漏洞挖掘工具; 漏洞基准测试集; 条件编辑距离

中图分类号: TP311

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2025036

Vulnerability benchmark generation algorithm based on finite state machines

XIE Yuandong, YAN Xuexiong, ZHAO Guangsheng, ZHAO Xu, QIU Han
Key Laboratory of Cyberspace Security, Ministry of Education, Zhengzhou 450001, China

Abstract: The emergence and development of various source code vulnerability detection techniques led to the evaluation of these tools' effectiveness becoming a critical issue. Source code vulnerability benchmarks were regarded as a basic basis for such evaluations. However, most of the existing benchmarks were found to inadequately simulate the complexity and diversity of real-world code, leading to significant discrepancies between benchmark testing and actual program analysis. Furthermore, current source code benchmarks were observed to lack support for testing vulnerability discriminability, resulting in an incomplete evaluation dimension. To address these limitations, vulnerability abstract triples and exploitable vulnerability chains were investigated, and a finite-state-

收稿日期: 2024-10-31; 修回日期: 2025-03-31

基金项目: 河南省自然科学基金 (242300421415)

Foundation Item: The Natural Science Foundation of Henan Province of China (242300421415)

引用格式: 谢元栋, 颜学雄, 赵光胜, 等. 基于有限状态机的漏洞基准测试集生成算法[J]. 网络与信息安全学报, 2025, 11(3): 150-162.

Citation Format: XIE Y D, YAN X X, ZHAO G S, et al. Vulnerability benchmark generation algorithm based on finite state machines[J]. Chinese Journal of Network and Information Security, 2025, 11(3): 150-162.

machine-based algorithm for generating source code vulnerability benchmarks was proposed. The approach consisted of three steps. First, a finite - state machine model was established to construct basic code paths. Second, conditional edit distance was employed to select discriminability-specific paths. Third, cyclomatic complexity wrappers were used to encapsulate path elements, thereby simulating the intricate structures of real-world code. Experimental results demonstrate that the generated benchmarks more closely resemble the complexity of real software, effectively exposing biases and deficiencies in vulnerability detection tools across different code paths. This study validates the algorithm's effectiveness and practical utility, confirming that it provides a more comprehensive and realistic benchmark for evaluating vulnerability detection tools.

Keywords: finite state machine, vulnerability discovery tools, vulnerability benchmark, conditional edit distance

0 引言

随着漏洞挖掘技术的持续演进, 除传统的人工审计与规则扫描方法外, 基于动态测试的智能挖掘工具也不断推陈出新。然而, 多项研究显示^[1-6], 当前大多数漏洞挖掘工具的实际有效性仍处于较低水平。通过对这些工具开展系统性的测试评估, 能够发现其存在的问题与缺陷, 进而提升检测的准确性与可靠性。一般而言, 漏洞挖掘工具的评估体系主要涵盖以下两个核心层面。

1) 构建评估指标。评估指标是衡量漏洞挖掘工具性能的关键标尺。其中, 常用的量化指标包括精确率 (precision)、召回率 (recall)、F1 值及辨别率 (discrimination rate, DR) 等。辨别率主要用于评估工具区分漏洞与非漏洞的能力, 其数值通过计算辨别率测试元组中正确辨别的比例得出。辨别率测试元组由两个代码结构相似但漏洞状态相反的测试用例构成。值得注意的是, 对于仅依靠随机猜测判断测试用例是否存在漏洞的工具, 其理论辨别率为 0%^[5]。辨别率的具体计算方式如下。

$$DR = \frac{\text{Discriminations}}{\text{Discrimination Tuples}} \quad (1)$$

2) 构建基准测试集。基准测试集是由软件代码、二进制程序或其他文件构成的数据集。按照是否包含漏洞, 基准测试集可进一步划分为漏洞基准集与非漏洞基准集。基准测试集是评估工作的基石, 理想的基准测试集应当具备统计显著性、标准答案可知、代表现实 3 个特征^[5]。统计显著性, 即基准测试集应达到一定规模, 确保评估结果具有统计学意义; 标准答案可知是指基准测

试集中测试用例是否真正含有漏洞是明确可知的; 代表现实是指测试用例必须充分反映真实软件代码的复杂性与多样性, 涵盖嵌套结构、条件循环、面向对象编程范式及代码执行路径等要素。

当前, 研究人员已提出多种漏洞基准测试集。大多数漏洞基准测试集较好地解决了统计显著性和标准答案可知问题, 但在反映真实代码的复杂性与多样性上仍存在明显不足。OWASP Benchmark^[7]在代码实现上采用极简的漏洞代码结构, 形式单一, 难以模拟实际场景的复杂性。Black 等^[8]虽通过嵌套条件判断、循环和方法调用等方式引入 Complexity 模块以提升复杂度, 但由于模板单一, 未能充分展现代码的多样性。Schuckert 等^[9]分别采用决策树和其他模式 (上下文、数据流) 构建测试集, 却未充分考虑生成的基准测试用例与真实代码在复杂度上的相似性。此外, 现有基准测试集普遍缺乏相似代码设计, 难以有效支持辨别率指标的评估。

针对上述问题, 本文提出一种基于有限状态机 (finite state machine, FSM) 的源代码漏洞基准测试集 (简称基准测试集) 生成算法。本文的主要贡献如下。

1) 针对基准测试集多样性问题, 提出了漏洞抽象三元组和源代码基准测试集 FSM 模型, 并生成了基本代码路径, 有效模拟真实代码的多样性特征。

2) 提出一个基于条件编辑距离的路径相似度计算方法, 筛选适用于辨别率评估的代码路径, 构建专用测试集。

3) 针对基准测试集复杂性问题, 提出一种利用代码圈复杂度包装器增加基准测试集代码复杂度的方法, 生成满足复杂性要求的源代码基准测

试集。

4) 以 PHP 语言中的 SQL 注入和 XSS 漏洞为例, 对本文提出的方法进行实验验证与深度分析。

1 相关工作

1.1 漏洞基准测试集生成技术

漏洞基准测试集旨在模拟现实世界中包含真实漏洞的代码。根据基准测试集生成算法的差异, 可分为漏洞注入技术和代码生成技术两类。

漏洞注入技术是指通过分析抽象语法树、控制流和数据流等信息, 在现有源代码的特定位置插入漏洞代码。IARPA 在 STONESOUP 计划^[10-11]中向 16 个程序插入漏洞代码片段, 但这些代码与基本程序的控制流和数据流缺乏关联性, 无法体现真实漏洞的复杂性; Dolan-Gavitt 等^[12]提出的大规模自动化漏洞注入技术 (LAVA) 基于动态污点分析, 在 4 款程序中构建 LAVA-M 基准测试集, 但其注入的漏洞仅存在于特定路径, 导致代码路径多样性不足。EvilCoder^[13]结合抽象语法树、控制流和数据流组合分析, 通过修改或删除源代码中的安全机制在 libpng 等 4 款程序中引入漏洞, 但难以保证标准答案的正确性。Kashyap 等^[14]开发的 Bug-Injector 工具将 Bug 模板中的 Bug 注入程序代码, 但无法保证 Bug 模板的复杂性, 且尚未公开实现细节。Hazimeh 等^[15]通过人工分析二进制程序的漏洞与补丁, 然后在新版本程序中引入漏洞。其在 7 款真实软件中移植了 118 个真实漏洞, 构建了 Magma 测试集。但该方法依赖人工进行分析, 目前无法实现自动化, 移植漏洞数量规模受限, 不具备统计显著性。Zhang 等^[16]提出了一种基于修复撤销的自动化注入方法 FixReverter, 并据此构建了一个包含近 8 000 个漏洞的测试集。然而, 该方法在原始语料的选择上存在明显的局限性, 所选取的原始语料范围较狭窄。并且, 在引入漏洞方面, 该方法缺乏足够的可靠性, 难以确保所引入漏洞的准确性和有效性。Jiang 等^[17]提出了一种基于数据描述模型的漏洞分析基准集构建方法 BugAna-Bench。此方法选取了 7 款程序作为数据集, 但这些数据主要来源于特定的数据库, 这种数据来源的单一性可能会导致构建的基准集存在偏差,

使其难以全面、准确地反映真实世界中多样化的漏洞情况, 进而影响漏洞挖掘工具评估的准确性和可靠性。

代码生成技术是通过构建代码模板和代码片段, 利用计算机自动生成或人工手动编写完整的基准测试集。Stivalet 等^[18]基于代码片段嵌套对 OWASP 2013 Top10 中部分 CWE 自动生成 42 212 个测试用例。然而, 该方法生成基准测试集时采用的代码控制流与数据流模式单一, 难以满足真实代码场景的复杂性需求。Sestili 等^[19]运用包含控制语句、循环语句等基本要素的代码模板生成缺陷代码, 但由于代码模板结构的多样性不足, 代码路径过于简单的问题仍未得到有效解决。Lee 等^[20]提出的 Fuzzler 方法, 将合成测试样例等价于“迷宫生成”问题, 但其重点在于处理输入消耗, 无法保证生成代码路径的多样性。Felici 等^[21]从“种子”错误出发构建 C 程序, 但其所使用的种子错误均为简单的断言失败情况, 这种简化方式在模拟现实世界中复杂错误场景时有效性明显不足。

综上所述, 当前基准测试集构建技术主要存在以下两大问题。

1) 对真实代码场景的模拟覆盖不足。现有基准测试集大多仅模拟简单、典型的含有漏洞代码, 在代码执行路径上无法全面涵盖真实代码的复杂场景, 难以有效评估漏洞挖掘工具在复杂环境下的性能。

2) 测试功能存在局限。现有基准测试集主要适用于准确率、精确率、召回率等指标的测试评估, 而缺乏针对辨别率指标的测试集。

1.2 圈复杂度

圈复杂度 (circle complexity number, CCN) 是用于衡量程序中函数或方法复杂度的重要度量指标^[22]。研究表明, CCN 与漏洞密度显著相关^[23-25], 高 CCN 的代码更容易暴露分析工具的缺陷^[26-27]。现实世界的软件代码往往具有较高的 CCN, 因为它们需要处理复杂业务逻辑。漏洞挖掘工具通常依赖程序控制流分析来识别漏洞, 包含高 CCN 代码的基准测试集能够提供更复杂的控制流结构, 有助于全面评估工具的分析能力。

2 基于有限状态机的源代码基准测试集生成算法

本文提出的基于有限状态机的源代码基准测试集生成算法流程如图1所示，具体实施步骤如下。

1) 构建FSM模型。通过对漏洞三元组进行抽象化描述，构建适用于源代码基准测试集的FSM模型，为后续代码生成提供基础框架。

2) 生成与筛选代码路径。基于已构建的FSM模型生成基础代码路径，而后在此基础上运用条件编辑距离算法对生成的代码路径进行筛选，提取适用于判别率评估的代码路径。

3) 包装代码路径元素。针对代码路径集合中的各个元素，利用CCN包装器进行装饰，通过增加代码逻辑分支与控制结构，有效提升代码的整体复杂度，使其更贴近真实代码场景。

4) 生成基准测试集。采用人工编写或自动化生成方式，将筛选和优化后的代码路径中的各个要素，按照既定规则进行编码替换，最终生成完整的源代码基准测试集。

2.1 漏洞抽象三元组

通过深入剖析常见漏洞类型的作用原理与利用方式，可将漏洞抽象为图2所示的三元组结构。该三元组由可控变量（controllable variable, CV）、安全机制和有效触发点三大核心要素构成，其具体定义如下。

可控变量是指在软件系统运行过程中，攻击者能够直接或间接操控的输入参数变量。这类变量广泛存在于用户输入信息、网络传输数据包、

系统配置文件内容等数据载体中。

安全机制是指用于验证数据信息的真实性和合法性，或对非法数据进行阻断和清除的机制，涵盖身份验证、数据过滤、加密保护等技术。完善的安全机制可以确保数据在进入系统关键部分之前是安全的；而不完善的安全机制则可被攻击者绕过或破坏，导致安全机制失效。

有效触发点是指系统中触发漏洞的关键操作点，攻击者可以利用这些点实现不正当操作。有效触发点包括错误的权限配置、缓冲区溢出漏洞、异常的数据操作逻辑等。

图2展示了漏洞链结构。在任意一条漏洞链中，只要其中某一要素不满足触发条件，就无法形成漏洞，对应的代码链将被判定为无漏洞代码链。无漏洞代码同样也是漏洞基准测试集的重要组成部分，可用于评估测试工具的误报率、判别率等。

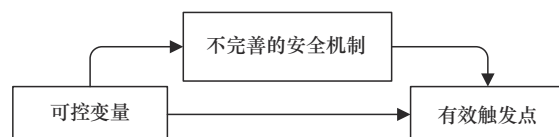


图2 漏洞抽象三元组

Figure 2 Vulnerability abstract triplet

2.2 源代码基准测试集有限状态机模型

FSM由一组状态以及在这些状态之间的转移和动作等行为组成。基于FSM的定义，在构建源代码基准测试集的FSM模型时，需首先明确其五元组 $(\Sigma, S, s_0, \delta, F)$ 中各要素的定义域，具体如下。

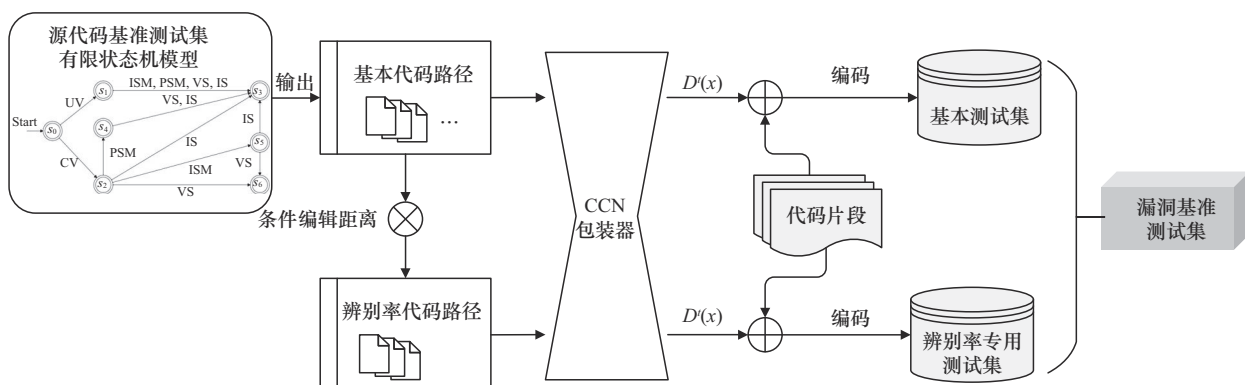


图1 基于有限状态机的源代码漏洞基准测试集生成算法流程

Figure 1 Algorithm flow for generating source code vulnerability benchmark based on FSM

1) 输入集合 Σ 中的元素。

可控变量：即变量值能够被用户直接或间接控制、修改的变量。以图3所示代码片段为例，`tainted` 变量直接接收用户输入数据，为可控变量。

```
1 | $tainted = $_GET['TaintedData'];
```

图3 CV 代码样例

Figure 3 Code sample of CV

不可控变量 (uncontrollable variable, UV)：与可控变量相对，即变量值无法直接或间接被用户控制或修改的变量。图4所示的代码片段是在图3所示代码的基础上新增了第2行代码，使得 `tainted` 变量不再可控。

```
1 | $tainted = $_GET['TaintedData'];
2 | $tainted = 'Constant';
```

图4 UV 代码样例

Figure 4 Code sample of UV

有效触发点 (valid sink, VS)：指程序中能够被潜在恶意输入到达的代码位置。在这些位置一旦恶意输入被执行，可能引发执行系统命令、非法访问敏感数据等高危操作。以图5中的代码为例，该段代码通过字符串拼接的方式构建 SQL 语句，其中 `mysqli_query` 函数便是典型的有效触发点。

```
1 | $query = "select * from users where id=$tainted";
2 | $result = mysqli_query($db, $query);
```

图5 VS 代码样例

Figure 5 Code sample of VS

无效触发点 (invalid sink, IS)：与有效触发点相对，即由于控制流或数据流的限制，潜在恶意输入无法到达的代码位置，或者即使到达该位置，由于使用了安全函数，恶意输入无法触发危险操作。图6所示代码使用了安全的参数化数据库查询方法。

```
1 | $query = "select * from users where id=?";
2 | $stmt = odbc_prepare($db, $query);
3 | $ret = odbc_execute($stmt, array($tainted));
```

图6 IS 代码样例

Figure 6 Code sample of IS

不完善的安全机制 (imperfect security mechanism, ISM)：指的是虽然系统设置了安全机制，

但该机制存在可被绕过的缺陷或已经失效。在图7所示的代码中，仅对字符进行转义处理，存在绕过可能，对于图5中的 VS 而言即为不完善的安全机制。

```
1 | $tainted = addslashes($tainted);
```

图7 ISM 代码样例

Figure 7 Code sample of ISM

完善的安全机制 (perfect security mechanism, PSM)：即存在安全机制，并且该机制能够有效阻断恶意攻击者的攻击，保护系统安全。图8所示代码可将变量转换为 `int` 类型，对于图5中的 VS 而言即为完善的安全机制。

```
1 | $tainted = filter_var($tainted,
2 | FILTER_SANITIZE_NUMBER_INT);
```

图8 PSM 代码样例

Figure 8 Code sample of PSM

2) 状态集合 $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$ 。

3) s_0 是起始状态。

4) δ 是状态转移函数。

5) F 包括含漏洞和不含漏洞 2 种接受状态。

为解决 FSM 可能存在的状态爆炸问题，本文采用了路径剪枝策略，通过合并具有相似行为特征或属性冲突的路径，从而减少状态的数量。路径剪枝原则如下。

1) CV 与 UV 混合出现时，由最后出现的元素决定变量是否可控，应合并为最后出现的元素。

2) ISM 与 PSM 混合出现时，PSM 对变量影响更大，应记为 PSM。

3) VS 与 IS 混合出现时，VS 影响更大，应记为 VS。

4) 同一元素多次出现，表示该元素的多个实例，可抽象为 1 个元素。

根据上述原则，可以绘出源代码基准测试集的 FSM 图，如图9所示。

2.3 生成代码路径

(1) 基本代码路径

基本代码路径，即 FSM 从初始状态出发到达终止状态的迁移序列。图9中有 11 个状态转移序列，将这些状态转移序列转化为基本代码

路径后，可以得到19条基本代码路径，具体见表1。

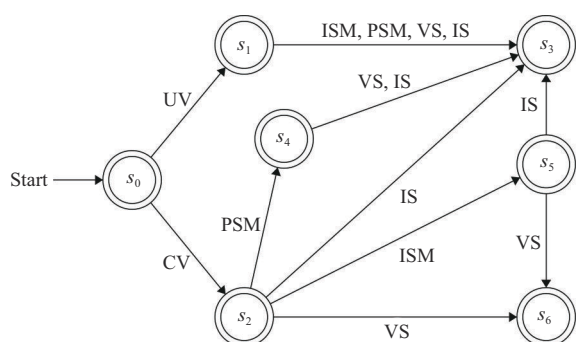


图9 源代码基准测试集FSM图

Figure 9 FSM diagram of source code benchmark

表1 漏洞基准测试集基本代码路径

Table 1 Basic code paths of vulnerability benchmark

序号	状态转移序列	基本代码路径	是否含有漏洞
1	s_0	$\emptyset$	否
2	s_0s_1	UV	否
3	$s_0s_1s_3$	UV.ISM	否
4	$s_0s_1s_3$	UV.PSM	否
5	$s_0s_1s_3$	UV.VS	否
6	$s_0s_1s_3$	UV.IS	否
7	$s_0s_1s_3$	UV.ISM.VS	否
8	$s_0s_1s_3$	UV.ISM.IS	否
9	$s_0s_1s_3$	UV.PSM.VS	否
10	$s_0s_1s_3$	UV.PSM.IS	否
11	s_0s_2	CV	否
12	$s_0s_2s_4$	CV.PSM	否
13	$s_0s_2s_4s_3$	CV.PSM.VS	否
14	$s_0s_2s_4s_3$	CV.PSM.IS	否
15	$s_0s_2s_3$	CV.IS	否
16	$s_0s_2s_5$	CV.ISM	否
17	$s_0s_2s_5s_3$	CV.ISM.IS	否
18	$s_0s_2s_5s_6$	CV.ISM.VS	是
19	$s_0s_2s_6$	CV.VS	是

(2) 基于条件编辑距离查找辨别率代码路径
通过在基本代码路径中寻找与含漏洞代码路径相似的不含漏洞代码路径，作为辨别率代码路径。在查找辨别率代码路径时，本文提出一种条件编辑距离（conditional edit distance, CED）作为相似度量。条件编辑距离是莱文斯坦距离

（Levenshtein distance）的一个变体，它通过引入替换约束条件来适应代码路径辨别率指标的测试要求。传统的编辑距离通过计算将一个序列转换为另一个序列所需的最小单字符编辑（插入、删除或替换）次数。条件编辑距离在此基础上增加了如下约束规则。

- 1) 仅允许替换操作。
- 2) 替换操作仅在允许的字符对之间进行。
- 3) 允许替换操作的距离为1，不允许替换操作的距离为无穷。

具体流程如算法1所示。根据元素相关性指定其操作允许的替换字符，具体见表2。计算不含漏洞路径与含漏洞路径的条件编辑距离，其中距离为1的路径对相似度最高，可作为辨别率代码路径。

表2 代码路径元素替换表

Table 2 Code path elements substitution table

元素	允许替换元素
$\emptyset$	$\emptyset$
UV	CV
CV	UV
ISM	PSM
PSM	ISM
VS	IS
IS	VS

算法1 条件编辑距离

输入 路径字符串 s_1, s_2

允许替换的字符表 T

输出 条件编辑距离 distance

- 1) **if** $\text{length}(s_1) \neq \text{length}(s_2)$ **then**
- 2) **return** ∞ ;
- // 长度不同，条件编辑距离为 ∞
- 3) **else if** $\text{length}(s_1) = 0$ **then**
- 4) **return** 0;
- // 空字符串，条件编辑距离为0
- 5) **else**
- 6) distance $\leftarrow 0$
- // 初始化条件编辑距离
- 7) **for** $i \leftarrow 0$ **to** $\text{length}(s_1)-1$ **do**
- 8) **if** $s_1[i] = s_1[i]$ **then**

```

9) distance  $\leftarrow$  distance + 0 ;
// 字符相同, 距离为 0
10) else if ( $s_1[i], s_2[i] \in T$ ) then
11) distance  $\leftarrow$  distance + 1 ; // 允许替换, 距离为 1
12) else
13) return  $\infty$  ; // 不允许替换, 距离为  $\infty$ 
14) nd
15) end
16) return distance ; // 返回条件编辑距离
17) end

```

表 3 为代码路径条件编辑距离计算结果, 根据计算结果可以得到以下 5 条辨别率代码路径。

- 1) CV.ISM.VS, UV.ISM.VS。
- 2) CV.ISM.VS, CV.PSM.VS。
- 3) CV.ISM.VS, CV.ISM.IS。
- 4) CV.VS, UV.VS。
- 5) CV.VS, CV.IS。

表 3 代码路径条件编辑距离计算结果
Table 3 The calculation results of condition edit distance between code paths

序号	代码路径	与 CV.ISM.VS 条件编辑距离	与 CV.VS 条件编辑距离
1	$\emptyset$	∞	∞
2	UV	∞	∞
3	UV.ISM	∞	∞
4	UV.PSM	∞	∞
5	UV.VS	∞	1
6	UV.IS	∞	2
7	UV.ISM.VS	1	∞
8	UV.ISM.IS	2	∞
9	UV.PSM.VS	2	∞
10	UV.PSM.IS	3	∞
11	CV	∞	∞
12	CV.PSM	∞	∞
13	CV.PSM.VS	1	∞
14	CV.PSM.IS	2	∞
15	CV.IS	∞	1
16	CV.ISM	∞	∞
17	CV.ISM.IS	1	∞

2.4 CCN 包装器

CCN 包装器结构如图 1 所示。图中 x 为原始元素, $D^t(x)$ 表示 CCN 包装器对原始元素的包装, t 为包装深度。CCN 包装器主要采用条件语言、嵌套循环、函数调用和类包装进行包装。

CCN 包装器仅对路径元素进行包装, 不改变元素属性。代码复杂度包装器实现流程如算法 2 所示。图 10 所示为使用 while 循环的 $D^1(x)$ 的 CCN 包装器代码样例, 其中第 4 行 placeholder 为被包装代码插入位置。需要注意的是, 在处理辨别率代码路径时, 应使用相同的 CCN 包装器, 以保证代码相似度不变。

```

1  $a = 10;
2  while($a <= 10) {
3      // x 元素代码开始
4      placeholder
5      // x 元素代码结束
6      $a++;
7  }

```

图 10 $D^1(x)$ 的 CCN 包装器代码样例

Figure 10 Code sample of $D^1(x)$ CCN wrapper

算法 2 代码复杂度包装器

输入 代码路径 path

复杂代码片段 ccn

最大包装层数 t

输出 含代码复杂度包装的代码路径

ccn_path

1) ccn_path $\leftarrow \emptyset$;

2) **foreach** $x \in \text{path}$ **do**

3) **for** $i \leftarrow 1$ **to** t **do**

4) $c \leftarrow \text{choices}(\text{ccn})$;

// 随机选择复杂代码片段

5) $x \leftarrow c.\text{replace}(\text{'placeholder'}, x)$; // 包装

路径元素

6) **end**

7) ccn_path.add(x) ;

// 将包装后的路径添加到结果

8) **end**

9) **return** ccn_path ;

// 返回包装后的代码路径

2.5 时间复杂度分析

算法 3 为基于 FSM 漏洞基准测试集生成

算法。

算法3 基于FSM漏洞基准测试集生成

输入 路径元素代码片段 *elements*

复杂代码片段 *ccn*

最大包装层数 *t*

基准集生成次数 *n*

输出 漏洞基准集 *B*

辨别率基准集 *D*

1) **Function** CodeGenerator(*path*, *elements*):

2) *code* $\leftarrow \emptyset$;

3) **foreach** *p* $\in$ *path* **do**

4) *replace*(*p*, *elements*[*p.type*]);

//元素替换为代码片段

5) *code.append*(*p*);

6) **end**

7) **return** *code*;

8) **end**

9) *B* $\leftarrow \emptyset$;

10) *D* $\leftarrow \emptyset$;

11) *base_paths* $\leftarrow$ FSM-Model ; // FSM 模型
基本代码路径

12) *disc_paths* $\leftarrow$ 条件编辑距离 = 1 ; //辨别
率代码路径

13) **for** *i* $\leftarrow$ 1 **to** *n* **do**

14) **foreach** *base_path* $\in$ *base_paths* **do** // 基本
路径

15) 代码复杂度包装器 (*base_path*, *ccn*, *t*);

16) *code* $\leftarrow$ CodeGenerator(*base_path*, *ele-*
ments);

17) *B.add*(*code*);

18) **end**

19) **foreach** (*v_path*, *n_path*) $\in$ *disc_paths* **do** //
辨别率路径

20) 代码复杂度包装器 ((*v_path*, *n_path*),
ccn, *t*);

21) *code* $\leftarrow$ CodeGenerator((*v_path*, *n_path*),
elements);

22) *D.add*(*code*);

23) **end**

24) **end**

25) **return** *B*, *D*

根据算法3, 通过分析代码的每个部分确定其对时间复杂度的贡献。

1) 代码1)~8)行为CodeGenerator函数, 将代码路径元素替换为代码片段, 替换次数为路径中元素个数, 记为 $O(1)$ 。

2) 代码9)~10)行为初始化, 通常记为 $O(1)$ 。

3) 代码11)~12)行为基本代码路径和辨别率代码路径初始化。由于FSM模型状态确定, 因此代码路径为固定值, 记为 $O(1)$ 。

4) 代码15)行和20)行为代码复杂度包装器函数, 复杂度取决于包装深度 t , 记为 $O(t)$ 。

5) 代码13)~24)行, 外循环运行 n 次。

6) 代码14)~18)行和19)~23)行为2个独立内循环。由于代码路径长度已知且有界, 复杂度受代码复杂度包装器和CodeGenerator函数影响, 可记为 $O(t+1)$, 即 $O(t)$ 。

因此, 算法的整体时间复杂度主要受包装深度 t 和生成测试用例的次数 n 影响, 可记为 $O(n \cdot t)$ 。

3 实验与分析

本文以PHP语言SQL注入漏洞和XSS漏洞为例进行实验验证与分析, 主要解决以下3个问题。

1) 与现有方法相比, 本文方法生成的基准测试集是否更接近真实软件。

2) 本文方法生成的多种代码路径的基本测试集在漏洞挖掘工具测试中是否具有优势。

3) 本文方法生成的辨别率专用测试集在评估工具漏洞挖掘性能上是否有效。

本文实验硬件配置为处理器Intel i9-10900, 内存64 GB。

近年PHP语言漏洞基准测试集生成方法相关研究成果汇总在表4中, 这些基准测试集生成方法具有一定的典型性和代表性, 已被纳入美国国家标准技术研究所软件保证参考数据集。本次实验将以这些已纳入SARD的方法为参考, 展开全面的对比分析。

3.1 生成源代码基准测试集

本文基于FSM的源代码基准测试集生成算法(简称FSM-TSG)代码已开源, 可在github(FSMTSG)下载查看。实验中使用的代码片段主要来源于公开披露的CVE代码、公开漏洞库

表 4 PHP 语言漏洞基准测试集生成方法研究

Table 4 Research on the generation of PHP vulnerability benchmark

名称	年份	简称
Large scale generation of complex and faulty PHP test cases ^[18]	2016	L-TSG
Systematic generation of XSS and SQLi vulnerabilities in PHP as test cases for static code analysis ^[9]	2022	S-TSG
Vulnerability test suite generator (VTSG) version 3 ^[8]	2023	V-TSG

和专家经验。实验中生成的 PHP 语言漏洞基准测试集汇总在表 5 中, 其中, L-TSG、S-TSG、V-TSG 均未提供辨别率测试用例, V-TSG 暂不支持 XSS 漏洞, 因此相关数据为 0。

表 5 PHP 语言漏洞基准测试集

Table 5 Vulnerability benchmarks of PHP

名称	漏洞类型	漏洞用例数	非漏洞用例数	辨别率用例数	总数
L-TSG	SQLi	912	8 640	0	9 552
	XSS	4 352	5 728	0	10 080
S-TSG	SQLi	9 982	40 628	0	50 610
	XSS	24 171	130 809	0	154 980
V-TSG	SQLi	109 824	175 296	0	285 120
	XSS	0	0	0	0
FSM-TSG	SQLi	800	6 800	4 000	11 600
	XSS	800	6 800	4 000	11 600

3.2 漏洞基准测试集生成方法对比分析

本节从代码圈复杂度、控制流图 (control flow graph, CFG)、耦合度和平均运行时间 4 个维度, 将本文提出的基准测试集生成方法与现有方法进行系统性对比分析, 具体指标见表 6。实验数据通过代码质量分析工具 `phploc` 采集获得, 详细的对比结果见表 7~表 9 及图 11。

漏洞基准测试集与真实软件代码圈复杂度对比结果汇总在表 7 中。为直观地展示实验结果, 采用 K-Means 算法对表 7 数据进行了聚类, 并将结果进行三维可视化, 如图 11 所示。图中数据分成 3 个簇: 簇 0 包括 FSM-TSG、MediaWiki、Joomla、ThinkPHP、phpBB, 簇内样本数量最多, 反映出相似的代码圈复杂度特征; 簇 1 为 phpLiteAdmin 和 WordPress, 两者复杂度较高, 尤其 phpLiteAdmin 为单文件应用, 复杂度远超平均水平; 簇 2 包括 L-TSG、S-TSG 和 V-TSG, 3 者代码圈复杂度较低, 与真实软件相比差异较大。

表 6 漏洞基准集生成方法对比指标

Table 6 Comparison indicators of vulnerability benchmark generation method

类别	指标	简称
代码圈复杂度	逻辑代码行平均圈复杂度	AvgCcn.LLOC
	类平均圈复杂度	AvgCcn.Class
	方法平均圈复杂度	AvgCcn.Method
控制流图	CFG 平均节点数	AvgNode.CFG
	CFG 平均边数	AvgEdge.CFG
耦合度	对象平均耦合度	AvgCBO
平均运行时间	每 1 000 个测试用例平均耗时	AvgTime.1 000

表 7 漏洞基准测试集与真实软件代码圈复杂度对比

Table 7 Comparison of CCN between vulnerability benchmarks and real software

类别	名称	AvgCcn.LLOC	AvgCcn.Class	AvgCcn.Method
漏洞基准测试集	L-TSG	0.12	1.0	1.0
	S-TSG	0.07	0	0
	V-TSG	0.17	1.16	1.08
	FSM-TSG	0.37	10.30	3.73
真实软件	WordPress 6.6.2	0.46	48.28	5.55
	Joomla 5.1.4	0.40	12.62	3.42
	ThinkPHP 6.1.4	0.42	14.43	2.73
	phpBB 3.3.13	0.45	11.74	3.25
	phpLiteAdmin 1.9.8.2	0.42	78.00	6.42
	MediaWiki 1.42.1	0.32	15.44	2.97

表 8 中, S-TSG 的测试用例代码中仅包含了漏洞的简单实现, 因此其 AvgNode.CFG 较低。L-TSG 与 V-TSG 的 AvgNode.CFG 和 AvgEdge.CFG 仍然偏低, 均未达到表中真实软件中的最低值。以上 3 者的 AvgCBO 均为 0, 其测试用例代码中未引入 PHP 语言面向对象特性或对象间耦合度太低。FSM-TSG 通过构建包含函数调用和类

间引用的代码片段，使得表中3项指标均达到真实软件水平。

表8 漏洞基准测试集与真实软件CFG和耦合度对比

Table 8 Comparison CFG and CBO of between vulnerability benchmarks

类别	名称	AvgNode. CFG	AvgEdge. CFG	AvgCBO
漏洞 基准 测试集	L-TSG	8.30	8.49	0
	S-TSG	5.37	2.83	0
	V-TSG	8.97	10.22	0
	FSM-TSG	89.38	103.15	0.82
真实 软件	WordPress 6.6.2	141.82	182.7	1.91
	phpBB 3.3.13	120.11	164.32	2.66
	phpLiteAdmin 1.9.8.2	1 424	1 915.5	0.8
	ThinkPHP 6.1.41.42.1	36.04	36.04	3.20

为对比不同基准测试集生成方法性能，本实验在同一环境下运行代码10次，计算每生成1 000个测试用例的平均耗时，实验结果见表9。从结果来看，V-TSG的平均耗时最长（16.53 s），其余3款工具的平均耗时差异较小，FSM-TSG耗时最短（仅为0.72 s）。由于不同编程语言可能带来性能差异，在均采用Python语言的3种方法中，FSM-TSG的平均耗时具有明显优势。

表9 漏洞基准集生成平均耗时

Table 9 Average time-consuming of vulnerability benchmark generation

名称	L-TSG	S-TSG	V-TSG	FSM-TSG
编程语言 及版本	Python 3.12.7	Java 21.0.5	Python 3.12.7	Python 3.12.7
AvgTime.1000	1.00 s	0.89 s	16.53 s	0.72 s

表11 PHP语言漏洞挖掘工具测试结果

Table 11 Evaluation result of PHP language vulnerability discovery tools

指标	RIPS				WAP				Kunlun-M			
	L-TSG	S-TSG	V-TSG	FSM-TSG	L-TSG	S-TSG	V-TSG	FSM-TSG	L-TSG	S-TSG	V-TSG	FSM-TSG
精确率	0.154	0.441	0.503	0.261	0.122	0.230	0.459	0.333	0.140	0.313	0.470	0.417
召回率	0.224	0.050	0.206	0.363	0.109	0.010	0.134	0.001	0.171	0.269	0.181	0.063
F1值	0.183	0.090	0.292	0.304	0.115	0.018	0.208	0.002	0.154	0.290	0.261	0.109
辨别率	—	—	—	0.093	—	—	—	0	—	—	—	0.034

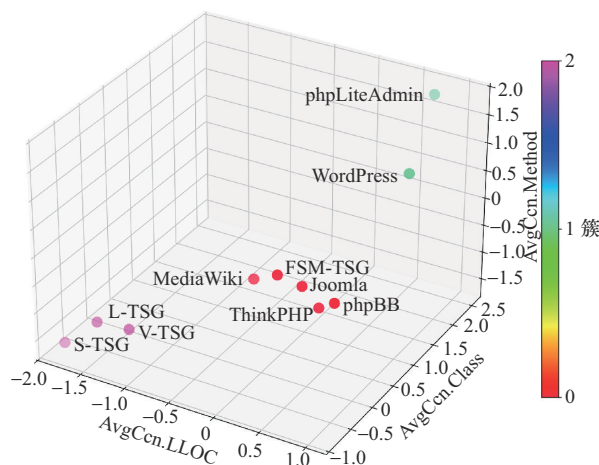


图11 聚类结果三维可视化

Figure 11 3D visualization of clustering results

3.3 漏洞挖掘工具测试评估

本文共选用3款开源PHP语言漏洞挖掘工具，软件名称及版本号见表10。评价指标选用精确率、召回率、F1值和辨别率。

表10 待测漏洞挖掘工具版本

Table 10 Version of vulnerability discovery tools

漏洞挖掘工具	版本
RIPS	0.55
WAP	2.1
Kunlun-M	2.6.5

在SQL注入漏洞测试集进行测试，结果见表11。对于RIPS，FSM-TSG在精确率方面，相比S-TSG和V-TSG下降显著，召回率和F1值变化较小，说明代码复杂度和路径多样性的变化导致其误报增多。Kunlun-M在FSM-TSG上相较其他测试集在召回率和F1值上的表现较差，原因可能是其内置静态分析规则对多路径适应性不足，导致漏报增多。WAP在4种基准测试集上的表现均较差，尤其是在FSM-TSG上，其召回率

和 $F1$ 值均低于 0.3%，说明代码复杂度的增加对 WAP 漏洞挖掘能力影响较大。

(1) 基本路径测试集结果分析

3 款漏洞挖掘工具在源代码基准测试集基本代码路径测试集中报告的漏洞数量见表 12。对于含有“IS”和“VS”的路径，3 款工具均报告了漏洞，在不含 Sink 元素的路径中没有报告任何漏洞，说明 3 款漏洞挖掘工具均采取了首先检测 Sink 点的策略。与含“IS”代码路径相比，含“VS”的代码路径所报告的漏洞数量更多，说明 3 款工具在区分 Sink 点有效性上具备一定的能力。其中，RIPS 分别在包含“ISM”和“PSM”的路径中报告了 371 个、339 个“漏洞”，数量相近，说明 RIPS 在判断安全机制对 Sink 点的影响时存在不足。Kunlun-M 未报告含“UV”代码路径漏洞，可以猜测其在检测漏洞时，将变量是否可控作为一项重要因素。WAP 仅报告了 3 个漏洞，结果统计性不强，无法对其多路径支持能力进行分析判断。

使用基本路径测试集对漏洞挖掘工具进行测试，不仅可以给出常用评估指标的结果，还能够在代码要素方面呈现出工具的漏洞挖掘能力缺陷，为工具后续改进提供参考。

(2) 辨别率专用测试集结果分析

在辨别率测试中，各漏洞挖掘工具正确辨别的漏洞数量见表 13。在辨别率指标中，由于其他基准测试集生成方法未提供专用测试用例，结果仅基于 FSM-TSG 进行分析。在表 11 中，RIPS 辨别率数值较高，Kunlun-M 次之，这 2 款漏洞挖掘工具的辨别率均低于 0.1。导致这一情况的原因可能是漏洞挖掘工具仅对 Sink 点进行了判断，而对变量可控性、安全机制完善性和触发点有效性判断不足。WAP 辨别率为 0，说明其缺乏辨别相似代码中的漏洞的能力。

表 12 漏洞挖掘工具在基本代码路径测试集中报告的漏洞数量

Table 12 The number of vulnerabilities reported by tools in the basic code path benchmark

基准测试集类型	代码路径	测试用例数量	RIPS	WAP	Kunlun-M
不含漏洞	Ø	400	0	0	0
	UV	400	0	0	0
	UV.ISM	400	0	0	0
	UV.PSM	400	0	0	0
	UV.VS	400	110	0	0
	UV.IS	400	54	0	0
	UV.ISM.VS	400	106	0	0
	UV.ISM.IS	400	63	0	0
	UV.PSM.VS	400	112	0	0
	UV.PSM.IS	400	51	0	0
	CV	400	0	0	0
	CV.PSM	400	0	0	0
	CV.PSM.VS	400	119	0	25
	CV.PSM.IS	400	57	0	18
	CV.IS	400	81	2	12
	CV.ISM	400	0	0	0
	CV.ISM.IS	400	67	0	15
	CV.ISM.VS	400	135	0	26
	CV.VS	400	155	1	24
	汇总	7 600	1 110	3	120

相比表 11，表 13 中 3 款工具在辨别率专用测试集的精确率、召回率和 $F1$ 值方面的表现差异较大，其中 RIPS 召回率达到 0.909，Kunlun-M 精确率达到 0.687。原因在于，辨别率专用测试集中正样本和负样本各占 50%，与基本路径测试集中分布情况不同，而这 3 个指标受样本分布影响

表 13 漏洞挖掘工具在辨别率专用基准测试集上的评估结果

Table 13 Evaluation result of vulnerability discovery tools in the discrimination benchmark

漏洞挖掘工具	CV.ISM.VS			CV.VS		精确率	召回率	$F1$ 值	辨别率
	CV.ISM.IS	CV.PSM.VS	UV.ISM.VS	CV.IS	UV.VS				
RIPS	52	16	24	66	28	0.572	0.909	0.702	0.093
WAP	0	0	0	0	0	0	0	0	0
Kunlun-M	9	4	27	11	16	0.687	0.153	0.251	0.034

较大。辨别率指标反映工具区分相似的含漏洞代码与不含漏洞代码的能力, 即边界模糊领域的二分类能力, 是对工具识别真实漏洞能力的较极端考验。由于辨别率指标限定了测试集中正、负样本成对存在, 计算过程也限定在相似的正、负样本之间, 避免了指标结果受样本分布而产生的错误结论。

4 结束语

本文提出一种基于FSM的源代码漏洞基准测试集生成算法, 依据漏洞构成三元组建立了漏洞基准测试集FSM模型, 生成基本代码路径, 以此为基础采用条件编辑距离生成辨别率代码路径, 并利用CCN包装器增加代码复杂度。本文基于PHP语言SQL注入漏洞和XSS漏洞进行实验, 并使用3款漏洞挖掘工具进行测试。结果表明, 该算法生成的基准测试集能够模拟现实代码的真实性和复杂性, 代码路径能够发现工具具体的能力缺陷, 辨别率专用测试集对判断工具辨别漏洞和非漏洞代码能力具有显著优势。未来可考虑使用并行FSM或Petri网模型, 结合更多代码复杂特性(如上下文敏感性、信息流等), 进一步提高生成基准测试集的真实性和多样性。

参考文献:

- [1] OKUN V, GAUCHER R, BLACK P E. The static analysis tool exposition (SATE) 2008: NIST SP 500-279[R]. Gaithersburg, MD: National Institute of Standards and Technology, 2009.
- [2] OKUN V, BLACK P E, DELAITRE A M. The second static analysis tool exposition (SATE) 2009: NIST SP 500-287[R]. Gaithersburg, MD: National Institute of Standards and Technology, 2010.
- [3] OKUN V, BLACK P E, DELAITRE A M. Report on the third static analysis tool exposition (SATE 2010): NIST SP 500-283[R]. Gaithersburg, MD: National Institute of Standards and Technology, 2011.
- [4] OKUN V, DELAITRE A M, BLACK P E. Report on the static analysis tool exposition (SATE) IV: NIST SP 500-297[R]. Gaithersburg, MD: National Institute of Standards and Technology, 2013.
- [5] DELAITRE A M, STIVALET B C, BLACK P E, et al. SATE V report: ten years of static analysis tool expositions: NIST SP 500-326 [R]. Gaithersburg, MD: National Institute of Standards and Technology, 2018.
- [6] DELAITRE A, BLACK P E, CUPIF D, et al. SATE VI report: bug injection and collection: NIST SP 500-341[R]. Gaithersburg, MD: National Institute of Standards and Technology, 2023.
- [7] MBURANO B, SI W. Evaluation of web vulnerability scanners based on OWASP benchmark[C]//2018 26th International Conference on Systems Engineering. Piscataway:IEEE Press, 2018: 1-6.
- [8] BLACK P E, MENTZER W, FONG E, et al. Vulnerability test suite generator (VTSG) version 3: NIST IR 8493[R]. Gaithersburg, MD: National Institute of Standards and Technology, 2023.
- [9] SCHUCKERT F, LANGWEG H, KATT B. Systematic generation of XSS and SQLi vulnerabilities in PHP as test cases for static code analysis[C]//Proceedings of 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). 2022: 261-268.
- [10] OLIVEIRA C. Real world software assurance test suite: STONESOUP (presentation) [C]//Proceedings of IEEE 27th Software Technology Conference (STC'2015). 2015.
- [11] DE OLIVEIRA C, FONG E, BLACK P. Impact of code complexity on software analysis: NIST internal or interagency report (NISTIR) 8165[R]. Gaithersburg, MD: National Institute of Standards and Technology, 2017.
- [12] DOLAN-GAVITT B, HULIN P, KIRDA E, et al. LAVA: large-scale automated vulnerability addition[C]//Proceedings of 2016 IEEE Symposium on Security and Privacy (SP). Piscataway:IEEE Press, 2016: 110-121.
- [13] PEWNY J, HOLZ T. EvilCoder: automated bug insertion[C]//Proceedings of the 32nd Annual Conference on Computer Security Applications. New York: ACM Press, 2016: 214-225.
- [14] KASHYAP V, RUCHTI J, KOT L, et al. Automated customized bug-benchmark generation[C]//Proceedings of the 19th International Working Conference on Source Code Analysis and Manipulation (SCAM). Piscataway:IEEE Press, 2019: 103-114.
- [15] HAZIMEH A, HERRERA A, PAYER M. Magma: a ground-truth fuzzing benchmark[J]. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2020, 4(3): 1-29.
- [16] ZHANG Z, PATTERSON Z, HICKS M, et al. FixReverter: a realistic bug injection methodology for benchmarking fuzz testing[C]//Proceedings of 31st USENIX Security Symposium (USENIX Security 22). Berkeley: USENIX Association, 2022: 3699-3715.
- [17] JIANG Z, LI R, TANG C. BugAnaBench: benchmark for software vulnerability analysis and its construction method[C]//Proceedings of International Symposium on Computer Technology and Information Science (ISCTIS 2022). Guilin: SPIE, 2022.
- [18] STIVALET B, FONG E. Large scale generation of complex and faulty PHP test cases[C]//Proceedings of 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST). Piscataway:IEEE Press, 2016: 409-415.
- [19] SESTILI C D, SNAVELY W S, VANHOUDNOS N M. Towards security defect prediction with AI[J]. arXiv Preprint, arXiv: 1808.09897, 2018.
- [20] LEE H, KIM S, CHA S K. Fuzzle: making a puzzle for fuzzers[C]//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. New York: ACM Press, 2022.
- [21] FELICI R, POZZI L, FURIA C A. HyperPUT: generating synthetic faulty programs to challenge bug-finding tools[J]. Empirical Software Engineering, 2024, 29(2): 38.

- [22] MCCABE T J. A complexity measure[J]. IEEE Transactions on Software Engineering, 1976, SE-2(4): 308-320.
- [23] PAGE M J, MCKENZIE J E, BOSSUYT P M, et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews[J]. Systematic Reviews, 2021, 10(1): 89.
- [24] CHOWDHURY I, ZULKERNINE M. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities[J]. Journal of Systems Architecture, 2011, 57(3): 294-313.
- [25] MEDEIROS N, IVAKI N, COSTA P, et al. Software metrics as indicators of security vulnerabilities[J]. IEEE International Symposium on Software Reliability Engineering, 2017.
- [26] 马总帅, 武泽慧, 燕宸毓, 等. 漏洞基准测试集构建技术综述[J]. 计算机科学, 2024, 51(1): 316-326.
- MA Z S, WU Z H, YAN C Y, et al. Survey of vulnerability benchmark construction technique[J]. Computer Science, 2024, 51(1): 316-326.
- [27] 黄诚, 孙明旭, 段仁语, 等. 面向项目版本差异性的漏洞识别技术研究[J]. 网络与信息安全学报, 2022, 8(1): 52-62.
- HUANG C, SUN M X, DUAN R Y, et al. Vulnerability identification technology research based on project version difference[J]. Chinese Journal of Network and Information Security, 2022, 8(1): 52-62.

[作者简介]



谢元栋 (1989-), 男, 山东泰安人, 网络空间安全教育部重点实验室硕士生, 主要研究方向为 Web 攻防和漏洞检测。



颜学雄 (1975-), 男, 湖南耒阳人, 网络空间安全教育部重点实验室副教授, 主要研究方向为 Web 安全和漏洞挖掘。



赵光胜 (1984-), 男, 河南濮阳人, 网络空间安全教育部重点实验室讲师, 主要研究方向为嵌入式设备安全和网络攻防。



赵旭 (1986-), 男, 河南三门峡人, 网络空间安全教育部重点实验室副教授, 主要研究方向为漏洞挖掘和 Web 安全。



邱茜 (1981-), 女, 河南郑州人, 网络空间安全教育部重点实验室副教授, 主要研究方向为网络安全模拟与评估、域间路由安全。