代码审计系统的误报率成因和优化

肖芫莹，游耀东，向黎希
（中国电信股份有限公司研究院，上海 200122）

摘　要：目前，代码审计已经成为网络安全建设中举足轻重的环节，基于自动化源代码检测的代码审计系统已经得到了广泛的应用，但仍存在诸多缺点。总结了当前代码审计系统的不足之处，简述了不同静态源代码检测算法的原理，并分析检测报告中出现误报的原因，提出了相应的优化思路，描述了优化方案的技术原理及其应用场景。

关键词：代码审计；静态检测技术；网络安全
中图分类号：TN915.9
文献标识码：A
doi: 10.11959/j.issn.1000−0801.2020324

Causes and optimization of the false alarm rate of code review system

XIAO Yuanying, YOU Yaodong, XIANG Lixi
Research Institute of China Telecom Co., Ltd., Shanghai 200122, China

Abstract: Code review technology has become a pivotal part in the construction of network security. Analysis of the test reports obtained by the current code auditing system shows that there are many false positives in the report. The shortcomings in the development of the code audit system were summarized, the principles of different detection algorithms were briefly described, the causes of false alarm rates were analyzed, corresponding optimization ideas were proposed, the technical principles of optimization were explained, and the application scenarios of optimization schemes were described.

Key words: code review, static analysis technology, network security

1　引言

随着网络的普及以及企业数字化转型的新形势，网络与信息安全的挑战日益严峻。企业业务系统经过多年发展，呈现多样化、复杂化、重要性高的特点。企业内部大量系统通过网络互联，在为业务提供支撑的同时，也导致威胁暴露面增加，威胁风险大幅度提高。各种业务系统由大量不同语言类型、不同平台的代码构成，这些代码造成的安全漏洞是被黑客入侵的最主要原因。同时，伴随着互联网产业的发展，大量使用开源软件成为趋势。而开源软件大部分来自企业外部，

收稿日期：2020−10−29；修回日期：2020−12−10

特别是国外，这些开源软件暗藏的漏洞同样是黑客攻击的重点。Gartner 统计，几乎 75%的黑客攻击来自代码漏洞。这导致供应链上代码的漏洞也成为了重大的安全威胁。

代码安全检测分二进制代码检测和源代码安全检测，基于二进制代码的检测适用度较差，要求苛刻，故本文主要讨论源代码的安全检测。若直接对源代码进行人工审查，效率低下且测试结果差异巨大，故代码审计系统多采用自动化的源代码检测和人工审计相结合的方法。主流的自动化源代码检测技术包括 SAST（static analysis security testing，静态源代码安全扫描）、DAST（dynamic analysis security testing，动态源代码安全扫描）、IAST（interactive application security testing，交互式扫描）。DAST 的适用范围较窄，IAST 的提出较晚，形成的成熟应用软件较少，而 SAST 伴随代码安全审计技术的提出发展至今，已经较为成熟，也成为业内占有率较高的自动化检测方式。SAST 通过对代码静态特征进行扫描，识别代码中的安全缺陷风险，提供分析和修复建议，帮助企业识别系统源代码中的安全风险，提早修复代码中的安全漏洞，从而保障系统安全性，提升企业整体安全保障能力。

目前，基于 SAST 的代码安全审计主要和企业安全开发流程相结合，内嵌于软件开发流程为企业提供安全保障。由于技术自身特点，基于代码缺陷特征值的静态检测的检测结果中安全缺陷误报率较高。同时由于软件开发中代码数量的日趋增长以及 DevOps 等新型开发模式的引入，企业对于开发迭代速度的要求逐步提高，误报率较高的问题成为制约代码安全审计推广和应用的一大阻碍。本文结合目前代码审计技术特点和软件开发应用场景，探讨如何降低基于 SAST 的代码审计中的缺陷误报率，为企业应用代码安全审计推广提供参考指导。

2 代码安全审计误报率成因分析

2.1 代码安全审计方法简介

代码安全审计可以通过人工审查和自动化扫描方式进行。目前，伴随着 DevOps 等的快速迭代，持续集成开发等软件开发流程的应用，基于自动化安全测试的安全审计成为代码安全审计的主流，并且在强调“测试左移”概念的如今，为了尽可能在更早的环节中进行测试，对代码审计的效率和精度也提出了更高的要求。

整个安全审计的核心是源代码的安全扫描，目前使用的代码安全自动化扫描测试方式主要如下。

（1）SAST

SAST 是一种白盒测试方式，主要通过对程序中数据流、控制流、语义等信息进行分析，与特定的安全缺陷特征规则库进行匹配，从而得到缺陷检测报告，帮助开发人员尽早发现代码中存在的安全缺陷，并提供代码缺陷信息以及修改方式帮助开发人员进行修复。主要特点是扫描结果漏报率较低但是误报率较高。基于 SAST 的测试工具发展较为成熟，它是目前使用得最多的代码安全扫描方式。

（2）DAST

DAST 是一种黑盒测试方式，它模拟黑客攻击的方式，通过动态数据流对程序进行测试，从运行状态发现程序存在的安全漏洞。它的主要特点是扫描结果误报率低，但是漏报率较高，并且漏洞的检出率取决于测试用例，也就是动态数据流数据准备的充分性。

（3）IAST

IAST 是一种黑白盒结合的测试方式。它不仅模拟了黑客攻击的方式，还和现有测试用例相结合，从运行状态发现程序存在的安全漏洞，同时定位对应的代码缺陷爆发点。主要特点是准确率较高、误报率低，同时保证漏报率在较低水平。

但由于这一扫描方式较为新颖，在国内外都缺少成熟的产品和完善的指导，方案成本较高，难以落地，但在未来有可能会替代原有方案，以提供更为准缺的扫描报告。

2.2　静态代码检测方法概述

静态代码安全检测流程如图 1 所示。目前，静态测试方法中普遍使用污染传播模型。该模型基于用户的输入是被污染的数据的假设，通过分析数据流，构建数据的输入点到安全缺陷漏洞，也就是爆发点的路径，判断代码是否存在安全风险。基本的分析方法是首先定义跨语言的抽象化的程序结构模型，然后基于已知安全漏洞信息库形成的安全缺陷特征，逐步扫描程序结构模型，查找代码中是否存在能够匹配对应的安全漏洞的部分，形成代码漏洞报告。这其中运用的技术包括语义分析、数据流分析、控制流分析、配置分析等，它们提取代码中的信息，将提取后的信息和已知的安全缺陷特征规则库进行匹配，在匹配中使用对比分析技术可以分为基于代码相似性、基于符号执行、基于规则以及基于机器学习 4 类，不同缺陷类型或检测工具会运用不同的分析技术。检测完成后，代码审计系统同时根据安全缺陷规则库中的信息，对缺陷形成原因和修正方案进行说明。帮助安全审计和开发人员判断，修正

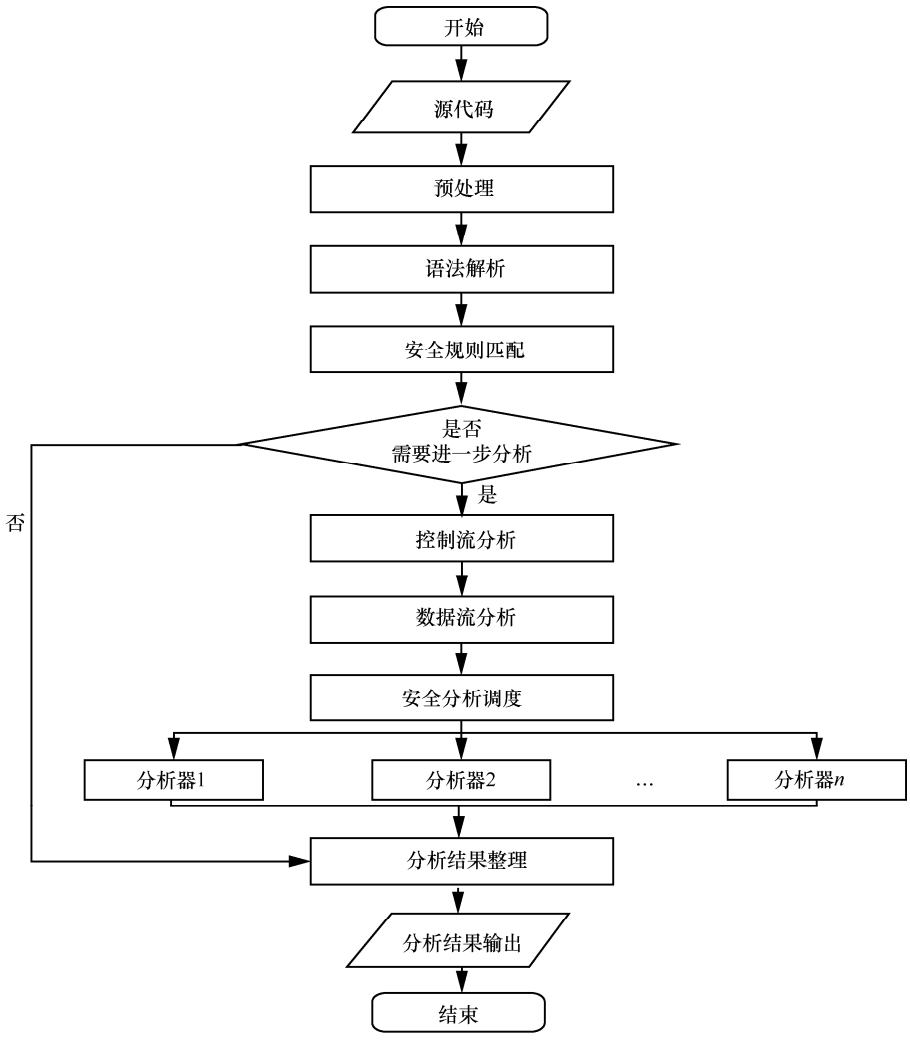


图 1　静态代码安全检测流程

代码缺陷。

主流静态检测工具分类见表 1，可以看到其中大部分来自国外，在代码质量检查和安全性检测方面都有覆盖，有开源软件供使用，也有精度更高的商用软件。静态检测的总体流程大同小异，具体实现细节中的匹配算法决定了结果的精度，而商业软件中的代码受专利保护，若是使用商业软件，在考虑减少误报时难以直接从检测软件本身入手。

表 1　主流静态测试工具分类

类别	工具
编码规范检测	Parasoft C++ Test
	LDRA Testbed
	IBM Rational Rule Checker
	CppCheck
	Gimpel Software Pc-lint
	…
安全性检测	Fortify SCA
	Secure Yun-ZBG
	CheckMarks
	CoBOT
	…
运行时错误检测	Matlab PolaySpace
	Parasoft insure++
	…

目前，各大国内高校也在进行代码检测相关的研究，提出了基于深度学习进行代码检测的方法，通过训练源代码缺陷分类器完成检测工作，但这停留在框架构建的阶段，还未形成可以被推广使用的检测工具。对静态检测工具本身的改进任重而道远。

2.3　静态代码检测方法误报率原因分析

静态代码测试的优点是缺陷检出率较高，在基于已知安全漏洞信息库的基础上形成的安全缺陷规则库，能有效地检测绝大部分安全缺陷漏洞。但是，另一方面，由于对于程序输入数据外部条件无法判断，以及对用户校验数据的规则很难做出完备的判断，导致静态代码测试存在较高的误报率。例如：

（1）程序外部输入数据已经在外部进行了安全校验，但是静态代码扫描程序无法获得相关信息，还是认为该数据为可能污染数据，存在安全隐患，报告为安全缺陷；

（2）程序内部对输入数据进行了安全判断和处理，使用了用户自定义程序逻辑和方法，但是警惕代码扫描程序无法了解该情况，从而依然报告为安全缺陷。

静态代码测试技术自身的特点导致其对用户自定义的安全处理逻辑以及运行环境中的可信数据流无法感知，这是造成误报率高的主要原因。

3　代码审计降低误报率技术原理

3.1　静态代码检测改进——白名单方法

由于程序设计语言本身是复杂灵活的，而静态检测依赖已经建立的模式和规则库，它们通常具有一定落后性，故对于现存的模式和规则库进行补充，能够以较为简单的方式提高检测的准确率，降低检测的误报率。规则库的补充可以从不同的角度进行。

（1）补充白名单方法规则库

代码中用于处理数据的方法被定义为白名单方法。

比如当前非常广泛地出现在源代码中的注入缺陷。它是 OWASP Top10 中的一项，它主要的作用原理是由于用户输入的文件未经过校验，有可能造成用户输入的恶意内容对系统进行作用，引发重大安全事故，造成严重后果。为了在代码层面杜绝事件的发生，开发人员在编写提供上传功能的部分代码时，通常需要先明确需求，从而确定允许上传文件的类型和内容，并在代码中进行处理。处理的主要方式有格式过滤和内容转义，

其中格式过滤的机制有黑白名单的方式，黑名单机制即拒绝特定格式，白名单机制与白名单方法的定义略有不同，指的是只接收特定格式的文件。白名单机制的安全性较高，可以有效预防新型的缺陷利用方式。

而在静态代码检测中，无论是黑名单还是白名单机制，由于开发人员有不同的代码编写习惯，以及检测模式和规则库落后于新式的代码防注入手段，这一部分用于对上传内容进行过滤和转义的代码难以被检测到，从而产生误报缺陷。

白名单方法规则库就是对这一问题的解决方案。这一方案要求开发人员对数据处理的代码进行封装，形成特定方法。这一方案的具体实现则分为两部分：一是在静态代码检测方法的数据流分析步骤后添加过滤模块；二是在过滤模块中人工添加代码中所使用的白名单方法。这一规则库扩展能够有效降低规则库的落后性，解决部分问题。

添加了白名单方法规则库的代码审计系统将自动对检测得到的缺陷进行过滤。若缺陷中涉及数据的路径上存在白名单方法，代表它被相应的白名单方法处理过，是安全的数据来源，则此缺陷被认定为误报，将不出现在最终的检测结果中。这种方法通过定制代码审计系统，降低了误报率。

（2）增加其他形式的规则库

根据相同的思路，同样可以增加其他形式的规则库，如文件或者方法过滤规则库。这是由于开发人员会编写一些在业务上线时不会用到的测试代码，通过将这些无须检测的测试文件的名称或者方法的名称添加到过滤规则库中，系统将在预处理时跳过这一部分代码的检测，这会缩短检测时间，并从另一个角度降低开发人员和审计人员在代码检测阶段的负担。当然，随着信息安全的发展，对于检测规则库本身的不断优化也必不可少，这要求代码审计工作者充分熟悉检测规则，并时刻关注网络安全领域。

3.2　黑白盒综合——插桩模式

除了上文提到了通过制定白名单等方法降低误报率，提高准确度，还可以利用一种基于灰盒测试理论的黑白盒联动测试方法。这种测试方法融合了 DAST 和 SAST 的优势，漏洞检出率极高、误报率极低，同时可以定位到 API（application programming interface，应用程序接口）和代码片段。

目前，软件安全检测方法主要集中在 SAST 静态白盒测试和 DAST 动态黑盒测试，这两种测试方法具各有其优缺点，白盒测试覆盖率高但误报较多，后期排查误报耗时耗力，在一定程度上降低了实用性，比如：SAST 扫描后可能报告 100 个缺陷，其中 20 个是真的缺陷，80 个是误报，需要投入大量人力排查；黑盒测试着眼于程序外部结构，从输入数据与输出数据的对应关系出发进行测试，测试结果准确率高，但无法定位漏洞的具体代码行数和产生漏洞的原因，需要比较长的时间进行漏洞定位和原因分析。因此，在规定的时间和成本内，最大限度地提高软件安全性一直是业务关注的问题。

黑白盒联动的安全检测主要模式是插桩模式，插桩模式是在保证目标程序原有逻辑完整的情况下，在特定的位置插入探针，在应用程序运行时，通过探针获取请求、代码数据流、代码控制流等，基于请求、代码、数据流、控制流综合分析判断漏洞，如图 2 所示，具体如下所示。

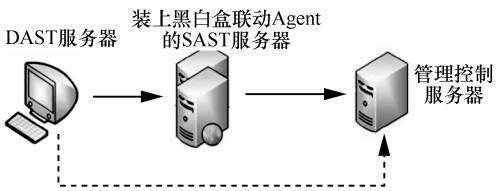


图 2　黑白盒联动示意图

步骤 1　插桩 Agent 部署位置：SAST 工具所在服务器（代码审计工具）。

步骤 2　插桩需与 DAST 扫描器关联，由

DAST 扫描器触发。

步骤 3 Agent 需集成到 SAST 系统，在源代码中部署传感器。

步骤 4 在 DAST 扫描器扫描期间检查 Web 应用程序执行时的源代码，在后端抓取应用程序，提供 100％的爬行覆盖率，查找并测试在黑盒 DAST 扫描期间未发现的隐藏输入。

插桩模式解决了传统 DAST 漏报和无法精确定位漏洞位置的问题。但需内嵌组件，此组件在被测应用程序中监测应用程序的反应，由此进行漏洞定位和降低误报。

4 代码审计应用场景

4.1 代码审计流程

代码审计的流程并非单一人员的工作，系统需要实现源代码自动化检测，也需要能够对检测结果进行人工审核。目前，代码审计系统的检测流程包括文件检测、缺陷审计、缺陷修复和缺陷复审。检测时由审计人员指定检测详情，选择审计用的引擎和相关规则集，在得到检测报告中对代码的安全性进行评估，对特定的缺陷进行人工审计，根据缺陷的来源、跟踪、爆发点处的代码确定缺陷是否为误报，是否需要在当前版本中进行解决，并分配给开发人员进行修改。修改后，审计人员还需要对代码进行复审，确定缺陷已被解决。

同时，代码审计也并非一蹴而就的过程，系统支持对版本控制工具如 git、SVN 中不同分支的代码进行检测，或者定时检测。从而通过对比检测结果，能够了解代码的安全性是否有所上升。通常，随着代码审计工作的进行，代码的安全性能够显著上升。

代码审计也并不是独立于系统生命周期的流程，在业务发展的过程中，系统将不断更新换代，新的攻击方式的发现也层出不穷，源代码安全风险的控制也随之动态变化，这就要求代码审计不断跟进系统建设，采用更好的手段及时对缺陷进行检测和修复。

4.2 在迭代开发中的应用

代码审计的流程涉及多次的检测，以及对检测出的缺陷进行审计和修复，并且需要根据实际情况判断是否需要继续进行审计，这同迭代开发的思路十分契合。迭代开发是一种在不断地迭代中逐步完善系统的灵活开发方式。它弥补了传统开发方式的缺点，在每一个迭代周期中实现产品的一部分功能，以解决问题。每次设计和实现的阶段叫作一个迭代。迭代开发能降低风险，通过早期的反馈及时修改代码，在持续的测试和集成中完善产品。

而代码审计以在开发过程中从根本上解决可能引发信息安全事故的缺陷为目的，要求及早发现缺陷并在发现缺陷时及时修改代码，故在迭代中加入代码审计能够更有效率地提高代码安全性。

以需求–设计–开发–交付的迭代为例，同代码审计结合的迭代开发在开发过程中不仅只关注功能实现，还考虑了代码的安全测试，进行代码的检测、缺陷的审计和修复。同时，为了降低误报率、减轻开发人员和审计人员的负担，在检测前根据开发的情况需添加白名单方法，对误报缺陷进行过滤。

每次迭代过程中的检测能够显著降低修改缺陷需要付出的时间和人工成本，做到遏制任意高位缺陷的苗头。对于添加了相应缺陷误报率降低模块，如白名单添加模块的代码审计系统，每次迭代后的交付过程中对误报缺陷进行的总结，能够通过修改代码和添加白名单方法及时反馈到代码审计系统中，修正代码审计系统中规则库的落后部分，从而更好地在下一个迭代周期中提供检测和审计服务，降低下一个迭代中检测结果的误报率。迭代周期流程如图 3 所示。

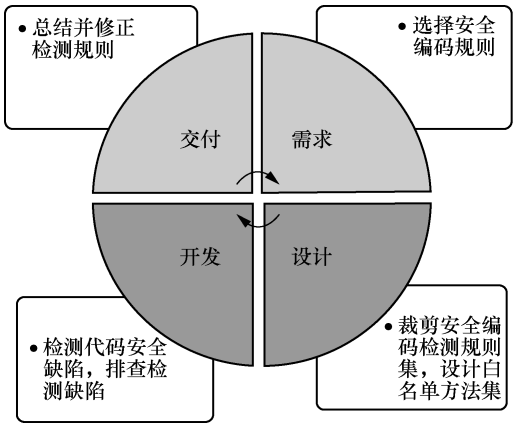


图 3　迭代周期流程

在迭代开发周期中实行缺陷审计，并进行缺陷的排查和白名单方法的设计，并统计不同周期中检测报告的误报率，能够观察到误报率的显著下降。同时，缺陷个数也会随之下降。在一系列的迭代后，系统的功能将会趋于完善，系统的安全性也将趋于完备，从而达到从代码层面消除可能造成安全事故的缺陷的目的。

随着信息安全逐渐受到重视，安全性成为了当前系统上线前的硬性要求，将安全评估引入开发中将会给整个开发组带来相应的压力，代码审计系统中检测结果的误报也会使开发的推进变得更为艰难。而在引入有效的误报解决手段后，这种随着不断迭代贯穿开发生命周期的代码审计也使安全性的硬性要求分散为一个个的阶段性要求，也就更为容易达到，鼓励了团队在安全开发上的积极性。

5　结束语

本文介绍了代码审计系统的应用场景、代码扫描实现原理和优化思路。在代码安全方案中，审计人员和开发人员面临的最主要的问题是如何高效地确定代码是否确实存在风险，从而在开发效率和代码安全中取得平衡。本文提出的优化方案结合代码审计中静态扫描的原理，针对实际使用中遇到的问题，过滤误报缺陷，尽量减少审计工作量。

保障代码安全归根结底是为了保障业务系统的安全，静态代码审计系统虽然发展较早，但由于其硬性不足，目前已有在快速地迭代开发中表现乏力的情况出现。本文提出了相应的审计信息迭代方案，随着其他技术的逐步发展，日后的研究将不限于代码审计系统本身，而是将结合多种信息安全保障手段对结果进行修正（如组件安全检测等）。这将使代码审计系统突破技术瓶颈，更好地为信息安全保驾护航。

参考文献：

[1] 罗琴灵. 基于静态检测的代码审计技术研究[D]. 贵阳: 贵州大学, 2015.
LUO Q L. Research on code audit technology based on static detection[D]. Guizhou: Guizhou University, 2015.

[2] 王优楠. 一种 XSS 漏洞灰盒检测方案的设计与实现[D]. 成都: 电子科技大学, 2017.
WANG Y N. Design and implementation of a XSS vulnerability grey box detection scheme[D]. Chengdu: University of Electronic Science and Technology of China, 2017.

[3] 何斌颖, 杨林海. Web 代码安全人工审计内容的研究[J]. 江西科学, 2014, 32(4): 536-537.
HE B Y, YANG L H. A study of Web artificial code security audit[J]. Jiangxi Science, 2014, 32(4): 536-537.

[4] 梁业裕, 徐坦, 宁建创, 等. 代码审计工作在整个安全保障体系中的重要价值[J]. 计算机安全, 2012(12): 32-34.
LIANG Y Y, XU T, NING J C, et al. The important value of code audit work in the whole security system[J]. Network and Computer Security, 2012(12): 32-34.

[5] 柳毅, 洪俊斌. 基于网络爬虫与页面代码行为的 XSS 漏洞动态检测方法[J]. 电信科学, 2016, 32(3): 87-91.
LIU Y, HONG J B. A dynamic detection method based on Web crawler and page code behavior for XSS vulnerability[J]. Telecommunications Science, 2016, 32(3): 87-91.

[6] 乔涛. 跨站脚本漏洞检测与防御技术研究[D]. 扬州: 扬州大学, 2017.
QIAO T. Research on cross-site scripting vulnerability detection and defense technology[D]. Yangzhou: Yangzhou University, 2017.

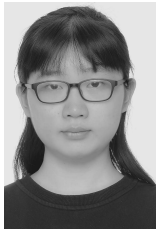
[7] 许波. 代码安全审计工作之我见[J]. 信息科技探索, 2020(5): 130-132.
XU B. My view on code security audit[J]. Public Communication of Science & Technology, 2020(5): 130-132.

[8] 王晓萌. 深度学习源代码缺陷检测方法[J]. 北京理工大学学报, 2019(11): 1155-1156.
WANG X M. Source code defect detection based on deep learning[J]. Transactions of Beijing Institute of Technology, 2019(11): 1155-1156.

[9] 李珍, 邹德清, 王泽丽, 等. 面向源代码的软件漏洞静态检测综述[J]. 网络与信息安全学报, 2019(2): 2-4.
LI Z, ZOU D Q, WANG Z L, et al. Survey on static software vulnerability detection for source code[J]. Chinese Journal of

Network and Information Security, 2019(2): 2-4.
[10] 黄显果, 王鹏, 刘静静, 等. 基于工具检测的源代码静态测试技术研究[J]. 软件研发与应用, 2019(5): 17-19.
HUANG X G, WANG P, LIU J J, et al. Research on source code static testing technology based on tool detection[J]. Computer Programming Skills & Maintenance, 2019(5): 17-19.

[作者简介]



肖芫莹（1997− ），女，现就职于中国电信股份有限公司研究院应用安全研究所，主要研究方向为代码审计。

游耀东（1979− ），男，现就职于中国电信股份有限公司研究院应用安全研究所，主要研究方向为安全开发 SDL、应用安全以及代码审计。

向黎希（1984− ），女，现就职于中国电信股份有限公司研究院应用安全研究所，主要研究方向为代码审计。