一种基于IR模拟执行的密码学API误用检测方法

何亚非[1,2],占力戈[1,2],聂 宇[1,2],傅建明[1,2*],彭国军[1,2]
(1. 武汉大学国家网络安全学院,湖北武汉 430072;2. 空天信息安全与可信计算教育部重点实验室,湖北武汉 430072)

摘 要: 密码学算法是现代软件系统中保护数据机密性与完整性的核心机制。然而,密码学应用程序编程接口(Application Programming Interface,API)的误用,例如使用可预测密钥或不安全密码算法,严重影响着软件系统的安全性,可能导致软件破解、网络攻击等后果,表明了检测密码学API误用的必要性。现有密码学API误用检测研究主要通过分析API参数值来识别误用现象,根据分析方式可分为动态分析方法和静态分析方法。动态分析方法能获取运行时的参数值,但受代码覆盖率低的限制,易遗漏未触发的代码,从而导致漏报。相比之下,静态分析方法具备更高的代码覆盖率,但现有的静态检测方法在理论与实现上仍存在显著局限:它们大多依赖简单的常量传播或模式匹配,仅能识别直接赋值的简单参数形式,而难以解析经过复杂指令(如字符串拼接、编码转换等"值变化"操作)处理的目标参数,导致当密码学参数经过复杂变换时,现有静态分析方法会出现大量漏报的情况,形成分析盲区。针对上述问题,本文提出并实现了ParamScope。这是一种基于中间表示(Intermediate Representation,IR)解释模拟的Java密码学API误用静态检测方法。首先,ParamScope实现了一种"赋值驱动"的程序切片算法。该算法基于SootUp程序分析工具构建,通过引入严格的赋值模式约束,能够精准提取密码学参数的依赖路径。其次,ParamScope实现了轻量级IR解释器对切片后的路径进行模拟执行,并引入包含真实逻辑的Android核心类库,结合Java反射机制,在静态分析中实现了对复杂方法调用的语义还原。该方法具有静态分析的高覆盖率优势,同时有效解决了静态环境下复杂指令导致的参数值分析难题。实验表明,ParamScope在公开数据集上的参数值重建精确率达到97.31%,且误用检测准确率达到96.2%,优于现有的先进静态与动态工具;在真实Android应用程序中的检测实验进一步显示,ParamScope有效识别出大量被编码或隐藏的真实参数典型案例,并报告其中的误用现象,相较于现有的静态工具CogniCrypt额外检测出约27%的误用案例,且单个应用平均分析用时仅约4.85 min。综上,ParamScope兼具静态分析的高覆盖率与复杂参数解析的高精准度,为密码学API误用检测提供了一种高精度、高效率的新型解决方案。

关键词: 静态分析;程序切片;模拟执行;Java程序分析;密码学API;误用检测
基金项目: 国家自然科学基金(No.62272351, No.62572354)
中图分类号: TP311.5;TP309.7 **文献标识码:** A **文章编号:** 0372-2112(2026)03-1132-15
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250813

A Cryptographic API Misuse Detection Method Based on Intermediate Representation Simulation

HE Yafei[1,2], ZHAN Lige[1,2], NIE Yu[1,2], FU Jianming[1,2*], PENG Guojun[1,2]
(1. *School of Cyber Science and Engineering*, *Wuhan University*, *Wuhan*, *Hubei* 430072, *China*;
2. *Key Laboratory of Aerospace Information Security and Trusted Computing of the Ministry of Education*, *Wuhan University*, *Wuhan*, *Hubei* 430072, *China*)

Abstract: Cryptographic algorithms serve as the core mechanisms for protecting data confidentiality and integrity in modern software systems. However, the misuse of cryptographic application programming interfaces (APIs), such as using predictable keys or insecure cryptographic algorithms, severely compromises software security, leading to software cracking and network attacks, which highlights the necessity of detecting such misuses. Existing studies on cryptographic API misuse detection primarily identify such misuses by analyzing API parameter values, and can be categorized into dynamic and static methods. While dynamic approaches can retrieve precise runtime parameter values, they suffer from low code coverage, leading to false negatives caused by untriggered code paths. In contrast, static analysis offers higher code coverage, but existing static methods face significant theoretical and practical limitations: they mostly rely on simple constant propagation or pattern matching, which only allow them to identify directly assigned parameters. Consequently, they struggle to resolve target parameters processed by complex "value-transformation" instructions (e.g., string concatenation or encoding conversion), leading to substantial blind spots and false negatives when analyzing cryptographic parameters that undergo complex

收稿日期:2025-09-16;录用日期:2026-03-09;责任编辑:覃怀银
*通信作者:傅建明

transformations. To address these limitations, this paper proposes ParamScope, a static detection method for Java cryptographic API misuse based on intermediate representation (IR) interpretation and simulation. First, ParamScope implements an "assignment-driven" program slicing algorithm built upon the SootUp framework, which incorporates strict assignment pattern constraints to precisely extract the dependency paths of cryptographic parameters. Second, it utilizes a lightweight IR interpreter to simulate the execution of the sliced statements, and integrates core Android libraries containing actual implementations. By combining this with Java reflection mechanisms, it achieves the semantic restoration of complex method calls during static analysis. This approach leverages the high coverage of static analysis and effectively resolves the challenge of statically analyzing parameter values derived from complex instructions. Evaluations on public datasets demonstrate that ParamScope achieves parameter value reconstruction and misuse detection accuracies of 97.31% and 96.2% respectively, outperforming state-of-the-art static and dynamic tools. Furthermore, experiments on real-world Android applications reveal that ParamScope effectively identifies typical cases of encoded or hidden real parameters, and reports the misuses within them. It detects approximately 27% more misuses compared to the leading static tool, CogniCrypt, with an average analysis time of only about 4.85 minutes per application. In summary, ParamScope combines the high coverage of static analysis with the high precision of complex parameter resolution, thereby providing a novel, precise, and efficient solution for cryptographic API misuse detection.

Keywords: static analysis; program slicing; simulation execution; Java program analysis; cryptographic application program interface; misuse detection

Foundation Item(s): National Natural Science Foundation of China (No.62272351, No.62572354)

0 引言

密码算法是现代软件系统中保护数据机密性与完整性的核心机制[1]。加密措施可防止敏感信息被未授权访问,这些算法已成为构建可信环境不可或缺的工具[2]。开发者可采用Java官方密码库(Java Cryptography Architecture, JCA)或第三方库(如Bouncy Castle)的密码学API(Application Programming Interface)快速实现加解密功能。然而,尽管密码库提供了极大便利,但与之相关的安全风险也受到广泛关注。

密码库的高复杂性与开发者的实现错误被公认为是导致密码误用的主要原因[3-4]。例如,Logjam漏洞[5]源于TLS协议出于兼容性原因对出口级512位Diffie-Hellman参数的支持,攻击者通过降级攻击使客户端使用弱密钥交换,从而在可行时间内破解会话密钥,解密加密通信。此外,广泛使用的OpenSSL库[6]也存在多项密码误用漏洞,例如CVE-2016-2183(SWEET32)[7]因继续支持强度不足的3DES密码而导致攻击者利用生日攻击恢复敏感数据,破坏会话的机密性。已有研究表明[8],88%的Java应用存在密码学应用程序编程接口(API)误用现象。为应对由此引发的安全问题,研究人员已实现多种检测工具[8-15]。

密码学API的正确使用高度依赖于传入API的参数[11],例如算法名称、密钥和盐值。这些参数的配置不当可能导致严重漏洞[16-18]。因此,近期研究主要聚焦于评估这些参数的安全性[15]来检测误用。动态方法(如RVSec[19])在运行时获取并验证传入API的实际参数值[18],具有高精确率。然而,由于部分功能或特定代码在自动化执行下难以被触发,动态分析的代码覆盖率十分受限。例如,RVSec在实验中仅达到44%的代码覆盖率[19],动态工具受此局限性影响,导致出现大量漏报现象。为此,CryptoGuard[11]和CogniCrypt[12]等近期研究专注于静态分析方法。静态分析方法无需执行程序,基于调用关系实现跨方法分析[20],理论上能够覆盖全部代码库,分析目标软件的功能和行为,从而能识别动态方法遗漏的密码学API误用。尽管现有静态检测器普遍表现优于动态方法,但由于静态分析的内在限制,其在解析动态赋值的"值变化参数值"时仍存在不足。

本文针对上述局限,提出一种静态分析的Java密码学API误用检测方法ParamScope。该方法首先通过静态分析识别所有密码学API调用点,继而通过细化的程序切片与中间表示(Intermediate Representation,IR)解释技术精确重建密码学API参数值。程序切片算法基于赋值语义分析执行精确的逆向数据流分析,从密码学API调用点出发,逆向追踪所有可能影响目标参数的值,并提取IR语句切片。随后,IR解释步骤将逐步解析IR语句并模拟对应Java代码,尽可能还原代码的真实语义,从而更准确地分析密码学API参数。总体而言,ParamScope的优势在于结合了静态分析的高覆盖率与动态分析的高准确率,且无需动态执行整个应用程序。

为评估ParamScope的有效性,本文在4个具有代表性的公开基准数据集上进行实验,并与3种现有的工具(CryptoGuard、CogniCrypt、RVSec)进行对比。实验结果表明,ParamScope在4个基准集上均显著优于

现有工具，平均准确率为96.2%，F_1分数为96.9%。本文在真实应用中的进一步实验表明，ParamScope比现有工具多检出27%的密码误用案例，并可有效识别现有方法难以精准分析的“因值变化指令而改变的密码学API参数”。

本文的主要贡献如下。

（1）提出一种Java程序的密码学API误用静态检测方法，采取改进的程序切片策略提升分析精度，并通过IR解释技术，以更精确地获取密码学API参数值，尤其是能够更好地分析已有工具忽略的值变化指令对API参数的影响。

（2）对现有程序切片策略进行改进，通过识别赋值模式精确追踪定义-使用链，既能够分析值传播指令，也能分析值变化指令对参数值的改变，克服了现有工具对值变化指令分析不足的缺陷。

（3）实现了ParamScope工具原型，并在4个密码学基准集和真实应用上对ParamScope、CryptoGuard、CogniCrypt、RVSec进行实证对比。结果表明，ParamScope取得了最好的准确率与F_1分数，比现有工具多检出27%的误用案例。

1 研究背景

本章首先阐述密码学API误用的定义、常见类型及其安全危害，继而介绍本文分析的基础——Jimple中间表示（IR）及其核心指令特性，随后综述现有的静态与动态检测工具及研究现状，最后通过具体的动机案例分析当前技术在处理复杂“值变化”指令时的局限性，从而引出本文的研究动机与解决思路。

1.1 密码学API误用

密码学API误用是指“程序中存在可执行的密码学API使用模式违反了安全规则的实例”。

程序P包含密码学API调用集C_{api}。对于任意调用点$c \in C_{\text{api}}$，定义其上下文属性（如参数值、调用顺序等）为集合A_c。设R为密码学标准的安全规则全集，其中规则$r \in R$定义了A_c的合法约束域D_r。当存在调用点c及规则r，使得c的实际属性$a \in A_c$满足$a \notin D_r$时，程序P的调用点c存在密码学API误用。

现有方法聚焦于评估密码学参数的安全性[15]来检测误用，即评估调用点c的参数值属性$a_{\text{arg}} \in A_c$是否满足参数安全性规则$R_{\text{arg}} \in R$。

密码学API参数安全性规则R_{arg}包含两类：（1）值域一致性规则R_{con}（Value Consistency Rules），即参数的实际值必须在安全的白名单值集合内；（2）生成随机性规则R_{ran}（Generation Randomness Rules），即参数的来源必须满足密码学要求的安全随机性。

密码学API参数安全规则的具体场景及误用的危害性包含如下4点。

（1）硬编码密钥。此类误用指应用程序中存在硬编码密钥[21]，包括对称密钥、私钥或基于口令的加密（Password-Based Encryption，PBE）密钥。攻击者可利用这些固定凭据解密敏感数据或访问KeyStore文件。

（2）弱加密算法。此类误用指使用已被证明不安全的密码算法（如MD5、RC4），这将导致密码被破解、数据机密性受损。已有研究证明了这些算法的安全性缺陷[22]，并报告了相关的成功攻击案例[16-17,23]，开发者在实现密码学功能时应遵循密码算法的选择标准。

（3）弱密码学配置。此类误用指不安全的密码学配置，包括基于口令的加密（PBE加密）中迭代次数不足[24]、使用固定初始化向量（IV）[25]、公钥加密中密钥强度不足[26-28]、伪随机数发生器（Pseudo-Random Number Generator，PRNG）中使用固定种子[29]。此类误用会降低密码强度，使系统易受暴力破解和密钥恢复攻击（如RSA加密中的密钥分解攻击）。

（4）使用不安全的随机数源。此类误用指在安全敏感场景中使用不安全的伪随机数发生器（例如通过java.util.Random和java.lang.Math生成伪随机数）。Debian OpenSSL漏洞[30]（CVE-2008-0166）表明，低熵PRNG[31]会显著缩减密钥空间，从而使攻击者恢复密钥。

1.2 中间表示（IR）

静态分析工具通常将源代码转换为IR以简化分析过程。本文采用SootUp[32]框架提供的Jimple IR[33]作为分析基础。Jimple是一种强类型的、基于三地址码的Java语言中间表示，它消除了Java字节码中复杂的堆栈操作，将程序逻辑简化为显式的变量赋值与指令序列。ParamScope主要关注以下3类Jimple IR的指令类型，这些指令构成了赋值驱动切片的基础。

（1）赋值语句（AssignStmt），如“x=y”或“x=c”。其中，y为其他变量，c为常量。该指令表示将右操作数的值（变量、常量或表达式结果）传递给左操作数。ParamScope针对赋值语句IR的分析是识别“值传播”模式的主要依据。

（2）调用语句（InvokeStmt），如“x.method（args）”，表示方法调用。Jimple显式区分了静态调用（staticinvoke）、虚方法调用（virtualinvoke）等调用类型。ParamScope针对调用语句IR的分析是识别“值变化”模式的主要依据。

（3）标识语句（IdentityStmt），如“x:=@param0”，用于在方法入口处将参数或this引用绑定到局部变量，为ParamScope追踪跨方法的参数传递提供变量映射。

1.3 相关工作

Java应用密码学API误用检测的相关研究主要分为动态分析和静态分析两类。

动态方法通过实际运行程序,并获取目标API参数的运行时值来检验其安全性。CryLogger[15]是首个开源的动态密码误用检测工具。它通过在运行时准确记录参数值,并借助日志分析组件识别误用行为。为降低CryLogger运行时开销并提高其检测效率,RV-Sec[19]采用运行时验证技术,同步检测与验证参数值,避免了将检测与验证分离而产生的额外开销,提升了动态分析的有效性。然而,动态分析通常需要完整的执行程序,导致显著的运行时开销和不完全的代码覆盖。这些局限使许多研究转向静态方法。

静态方法通过分析程序的源代码或中间表示,并依据预定义的安全规则来验证密码学API参数的安全性。CryptoLint[8]是该领域的开创性工作,提出了6项安全规则,并通过静态切片分析参数值。为进一步降低误报率,CryptoGuard[11]基于Soot框架[34]执行需求驱动的数据流分析,通过定义-使用关系追踪安全关键的密码值,并融入了实际编程规范中的启发式洞察。此外,CryptoGuard将检测能力扩展至16项规则,涵盖密码学API及SSL/TLS误用。CogniCrypt[12-13]采用更严格的领域特定语言,对API使用规范进行了更细粒度的建模。该工具为常用JCA类定义了使用模式,并将安全的使用模式规则转化为静态代码分析,以识别密码学API误用。

在研究检测方法的同时,现有研究还对这些工作的有效性进行了广泛的实证评估。实证研究MASC[35-36]定义了12个突变运算进行用例的突变测试,用于评估静态检测工具的有效性。研究结果表明,现有工具难以分析经“值变化指令”影响而发生改变的密码学API参数,尤其是当值变化指令需要通过函数调用实现时。这些问题导致了已有的静态方法具有局限性。

1.4 研究动机

为说明现有工具的局限性与本文方法的优势,考虑下列现实场景。某应用功能涉及加解密,使用JCA库的“MessageDigest.getInstance(String algorithm)”密码学API对用户口令进行哈希处理。然而,如代码1所示,由于该应用开发者密码学知识不足,选择了不安全的MD5哈希算法。为了规避直接静态检测,增加逆向难度,其将算法名称“MD5”编码为其十六进制ASCII形式“4D4435”,并在运行时解码。然而,此措施仅能隐藏算法名称,无法消除安全隐患,攻击者仍可利用MD5算法的弱点破解用户口令。在此案例中,密码学API参数“MD5”受到值变化指令(即“decodeASCII()”)的影响,从“4D4435”转化为“MD5”。而已有的静态方法无法捕捉此类赋值语义,导致无法解析“4D4435”的实际值,进而导致漏报。

代码1 ParamScope的动机案例

Listing 1 Motivation example of ParamScope

```
// 其中 decodeASCII("4D4435")返回 "MD5"
public String getAlgorithm(){
return decodeASCII("4D4435");
}
// 误用:使用了不安全的哈希算法 "MD5"
MessageDigest.getInstance(getAlgorithm());// 该代码在语义上等同于:
MessageDigest.getInstance("MD5");
```

密码学API参数值的安全性判定主要存在两类挑战。

(1)复杂生成逻辑。密码学API参数值可能呈现复杂的生成逻辑,ParamScope通过将赋值行为建模为两类来应对此挑战。一是值传播(Value Propagation,VP),指直接赋值,如str=“MD5”。二是值变化(Value Transformation,VT),指动态地进行值修改,如案例中的“decodeASCII()”。传统静态分析难以准确地分析“decodeASCII()”方法的内部逻辑,导致难以重建经“值变化指令”影响的实际值。Reardon等人[37]的研究指出,Android应用及各种软件SDK中广泛应用值变化指令来保护数据安全并规避静态分析。这些指令为静态分析带来了巨大挑战,产生了大量超过传统静态分析能力的密码学API参数值,CryptoGuard等现有的静态分析工具都无法检测到此类误用。这表明了现有静态方法“难以分析值变化指令”的局限性。

(2)误用判定的非局部性。虽然误用的触发点位于局部代码(API调用处),但它是否误用的判定逻辑具有全局性:密码学API参数值的属性通常由程序中非API调用点处的复杂的数据流操作决定(例如跨方法、跨类的赋值、外部输入)。本文的方法通过结合细化的、跨方法的程序切片与IR解释,解析值变化并精确重建参数值。

以上述案例为例,程序切片从密码学API调用点反向追踪变量“algo”,识别到其跨方法的赋值语句“decodeASCII(“4D4435”)”并提取相关的程序IR切片;IR解释阶段将IR语句进行解释和模拟,并通过Java反射实现赋值语句中的方法调用。该过程解析出传入密码学API的参数实际值“MD5”,使得ParamScope能够精确重建关键参数值。

2 ParamScope概述

图1展示了ParamScope的整体架构,该架构包含6个步骤:(1)预处理阶段从反编译代码中获取中间

表示(IR),构建方法调用图并分析类继承结构,以实现跨方法且精确的IR分析;(2)调用点识别阶段通过静态分析扫描IR,以精确识别所有密码学API调用点;(3)程序切片阶段从调用点出发进行后向切片,分析与目标参数相关的赋值语句,以获取影响目标参数值的IR语句集合;(4)值重建阶段对IR语句进行建模并通过IR解释和模拟恢复实际参数值;(5)误用判定阶段根据预定义的值域一致性规则R_{con}和生成随机性规则R_{ran}对重建的参数值进行误用行为检测;(6)可视化报告阶段提供数据流可视化与误用检测报告,生成误用检测结果及重建值的数据流图。

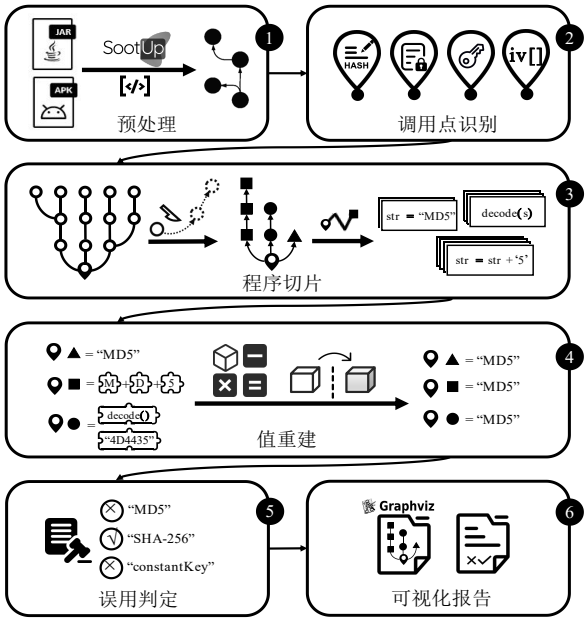

图1 ParamScope系统概述

Figure 1 System overview of ParamScope

在整个流程中,ParamScope有效解决了复杂赋值语义与继承关系分析的核心挑战,从而确保了密码学API误用高精度的检测结果。

3 ParamScope设计

本章阐述ParamScope各步骤的设计方案。

3.1 预处理与调用点识别

在预处理阶段,ParamScope利用dex2jar[38]将Android应用Dalvik字节码转换为Java字节码,以实现Java代码兼容的分析。随后,ParamScope基于SootUp[32]将Java字节码转换为Jimple IR,为后续赋值驱动的程序切片与IR解释提供基础。此外,构建更完整的调用关系既需要包含显式声明的方法,还需要分析类继承的方法,ParamScope通过类层次分析记录类继承关系并构建方法调用关系。

ParamScope沿用了Weiser等人[39]提出的"切片准则"。切片准则定义为$c=<s_{target}, V_{target}>$。其中,$s_{target}$表示切片起点,即目标密码学API的调用点;$V_{target}$表示关注变量集,即关注的目标参数变量集合。当目标参数集合包含多个参数时,ParamScope针对每个目标参数独立建立切片准则$\{c_{s0}, c_{s1}, \cdots, c_{sn}\}$,按序分别进行程序切片、值重建、误用检测。

表1列举了ParamScope切片准则集合C中每条切片准则c的目标API和目标参数,包括对称加密、PKCS#8私钥、基于口令的加密(PBE)及密钥库配置、加密与哈希算法、PBE迭代次数、初始化向量(IV)、公钥参数以及伪随机数生成器(PRNG)种子。

3.2 赋值驱动的程序切片

ParamScope的赋值驱动程序切片模型$<C, P>$由切片准则集合C与赋值模式约束P构成,其中$P=\{VP_{I\sim III}, VT_{I\sim III}\}$,定义了IR层的赋值模式约束。通过细化的程序切片模型,ParamScope克服了在复杂控制流和数据流结构中准确识别与切片准则相关的代码片段的核心挑战[40]。

考虑被切片程序包含以下元素:指令集S,即程序中所有IR语句的集合;变量集V,包含程序中的局部变量v、参数arg、字段f;表达式E,包含IR中二元运算op、方法调用call。

定义追踪集T,初始化为目标API参数变量x,即关注变量集V_{target}中的参数变量。对于语句$s \in S$,定义左值为l_s,右值为r_s。具体而言,约束P完整地覆盖了Java三地址码IR中的赋值语义,包含以下赋值模式。

(1)$VP_{\text{Ⅰ}}$(立即值与变量传播)。$s: t \leftarrow v$或$s: t \leftarrow c$,其中$t \in T, v \in V, c$为常量。

(2)$VP_{\text{Ⅱ}}$(方法参数传播)。$s: t \leftarrow \arg_i$,其中$t \in T$,$\arg_i \in V$,且$\arg_i$为当前方法的形参。

(3)$VP_{\text{Ⅲ}}$(字段传播)。$s: t \leftarrow y.f$或$s: t \leftarrow C.f$,其中$t \in T, y.f, C.f \in V$,$y.f$表示实例字段,$C.f$表示静态类字段。

(4)$VT_{\text{Ⅰ}}$(表达式衍生值)。$s: t \leftarrow e$,其中$e \in E$,e表示IR中二元运算、类型转换、方法调用返回值等表达式。

(5)$VT_{\text{Ⅱ}}$(引用类型副作用)。$s: l_s \leftarrow \text{call}(t_r)$,其中$t_r \in T$,$\text{call}(t_r)$为方法调用,将正在追踪的引用类型变量$t_r$(如byte[])传入方法内部并进行修改,如"Arrays.fill(arr, val)"。

(6)$VT_{\text{Ⅲ}}$(静态字段副作用)。$s: l_s \leftarrow$call,其中call为方法调用,且被调用者方法内部存在语句s满足$s: C.f \leftarrow$val且$C.f \in T$。

传统的程序切片算法缺乏Java语言的约束规则,通常只进行简单的定义-使用分析和变量追踪。ParamScope赋值驱动的程序切片精准建模了Java语

言Jimple IR中赋值语义,聚焦于对目标API参数存在赋值语义的语句,在切片过程中仅保留符合约束P的赋值依赖。

在切片算法上,传统的程序切片基于完整的数据与控制流依赖计算切片语句。相比之下,ParamScope基于方法IR语句列表和方法间调用关系执行轻量、快速的程序切片,以s_{target}为起点,提取目标API参数的后向数据依赖图(Backward Data Dependence Graph)。ParamScope引入了赋值模式约束P作为过滤准则,按需进行后向遍历并构建一棵无环的、聚焦于赋值的值依赖树(Value Dependency Tree),该值依赖树仅包含由约束P限定的赋值模式语句。针对方法调用的反射,ParamScope对网络通信、用户输入等API的模拟进行限制,并将其标记为外部输入。ParamScope完整保留了目标参数生成所需的所有数据依赖并限制了方法调用API,确保了转译指令的可执行性。

表1　ParamScope的切片准则列表

Table 1　List of slicing criteria for ParamScope

类名	目标方法	目标参数及其描述
java.security.KeyPairGenerator	KeyPairGenerator getInstance (String)	参数0,算法名称
	void initialize(int)	参数0,公钥密码密钥大小
java.security.KeyStore	void load (InputStream, char[])	参数1,密钥存储库密钥
	void store (OutputStream, char[])	参数1,密钥存储库密钥
	void setKeyEntry (String, Key, char[], Certificate[])	参数2,密钥存储库密钥
	Key getKey (String, char[])	参数1,密钥存储库密钥
	KeyStore getInstance (File, char[])	参数1,密钥存储库密钥
java.security.MessageDigest	MessageDigest getInstance (String)	参数0,算法名称
java.security.SecureRandom	void init (byte[])	参数0,种子字节数组
	void setSeed (byte[])	参数0,种子字节数组
	void setSeed (long)	参数0,种子长整型值
java.security.Signature	Signature getInstance (String)	参数0,算法名称
java.security.spec.DSAGenParameterSpec	void init (int, int)	参数0,DSA算法素数P位长度
	void init (int, int)	参数1,DSA算法素数Q位长度
java.security.spec.ECGenParameterSpec	void init (String)	参数0,椭圆曲线算法名称
java.security.spec.RSAKeyGenParameterSpec	void init (int,BigInteger)	参数0,RSA密钥大小
	void init (int,BigInteger)	参数1,RSA公钥指数
java.security.spec.X509EncodedKeySpec	void init (byte[])	参数0,X509标准编码密钥
javax.crypto.Cipher	Cipher getInstance (String)	参数0,密码算法名称
javax.crypto.Mac	Mac getInstance (String)	参数0,消息认证算法名称
javax.crypto.SecretKeyFactory	SecretKeyFactory getInstance (String)	参数0,密码算法名称
javax.crypto.spec.DHGenParameterSpec	void init (int, int)	参数0,DH素数模数位长度
	void init (int, int)	参数1,DH指数位长度
javax.crypto.spec.IvParameterSpec	void init (byte[])	参数0,初始化向量字节数组
javax.crypto.spec.PBEKeySpec	void init (char[])	参数0,密码字符数组
	void init (char[], byte[], int)	参数1,盐值
	void init (char[], byte[], int)	参数2,迭代次数
javax.crypto.spec.PBEParameterSpec	void init (byte[], int)	参数0,盐值
	void init (byte[], int)	参数1,迭代次数
javax.crypto.spec.SecretKeySpec	void init (byte[], String)	参数0,密钥字节数组
	void init (byte[], String)	参数1,算法名称

算法1详述了值依赖树的构建过程。该算法采用基于工作列表的迭代策略,输入为程序的调用关系CG和单个目标API调用点,输出为构建的值依赖树T_{target}根节点。算法首先在行1~4分别初始化为目标参数变量,初始化根节点T_{target},初始化赋值模式约束P和工作列表初始任务。行5~7检查当工作列表不为空时,取出当前任务各变量,将当前节点语句赋值为空。行8~11获取当前节点的所有IR语句并后向遍历,检查此语句是否为目标调用点的前序语句及是否满足赋值模式约束P。若满足,行12~13将此语句

加入切片并更新追踪集合 V_{track}，删除定义集，加入使用集。遍历结束时，行 14 检查追踪集合 V_{track} 是否仍包含未解析参数。若包含，则触发跨方法分析。行 15 ~ 16 从调用关系 CG 获取当前方法的所有调用者并进行遍历。行 17 ~ 19 将跨方法追踪的参数映射到新的目标 API 和参数，创建相连的新子节点以构建树结构，并将新的追踪任务压入工作列表。随着循环的进行，算法将逐节点地构建出包含目标参数所有依赖的完整值依赖树 T_{target}，最终由行 20 返回构建的值依赖树根节点。

算法 1 赋值驱动的程序切片算法

输入：调用关系 CG，单个目标 API 调用点 c_{target}
输出：值依赖树根节点 T_{target}
Procedure ProgramSlicing (CG, c_{target}):
1. $V_{track} \leftarrow$ getTargetArg (c_{target});
2. $T_{target} \leftarrow$ createNode(c_{target});
3. $P \leftarrow \{VP_{Ⅰ\sim Ⅲ}, VT_{Ⅰ\sim Ⅲ}\}$;
4. worklist.push (c_{target}, V_{track}, T_{target});
5. **WHILE** worklist ≠ 空集 **DO**:
6. (c_{target}, V_{track}, node) ← worklist.pop();
7. node.slice ← 空集;
8. $S \leftarrow$ getStatements(getCaller(c_{target}));
9. **FOREACH** 后向遍历语句 $s \in S$ **DO**:
10. **IF** findCallSite(c_{target}, s) **THEN**:
11. **IF** MatchPattern (P, s) is True **THEN**:
12. node.slice ∪ s;
13. $V_{track} \leftarrow (V_{track} \setminus \text{Def}(s)) \cup \text{Use}(s)$;
END IF
END IF
END FOREACH
14. **IF** V_{track} ≠ 空集 **THEN**:
15. callers ← getCallers (CG, c_{target});
16. **FOREACH** caller ∈ callers **DO**:
17. $V'_{track} \leftarrow$ mapArgsToParams (caller, V_{track});
18. node.child ∪ createNode (c_{caller});
19. worklist.push (caller, V_{track}, node.child);
END FOREACH
END IF
END WHILE
20. 返回 T_{target}

3.3 目标参数值重建

ParamScope 采用轻量级 IR 解释和模拟策略来重建目标参数值。在程序切片完成后，ParamScope 首先从算法 1 中生成的值依赖树中提取正向的值依赖路径，存在多条路径时，每条路径对应一个独立的重建值。ParamScope 不动态执行整个程序，仅在 IR 解释模拟过程中对其中的方法调用进行反射执行。ParamScope 维护了一个变量内存表，沿 IR 解释模拟的执行轨迹逐条更新变量状态。算法 2 详述了这一值重建模拟过程。

算法 2 输入为算法 1 构建的值依赖树及其对应的 API 调用点，输出为重建的目标参数值集合。行 1 ~ 3 初始化变量内存表、目标参数值集合和目标参数在变量内存表中对应的变量名。行 4 从值依赖树按正向提取各 IR 解释的路径，并在行 5 ~ 7 遍历路径中所有 IR 语句。行 8 ~ 9 对于包含方法调用的语句采取反射进行调用，不包含时，由行 10 直接解析其值传递行为，由行 11 在内存变量表更新对应的定义和实际值。行 12 在每条路径解释执行结束时更新重建的目标参数值集合，由行 13 返回结果。

在参数值重建的过程中，ParamScope 还处理了 Android 方法反射缺失的挑战：由于部分 Android 框架库方法在 Android 各版本 SDK 中仅包含存根而无实际实现，若在分析过程中采用 SDK 中的 Android 框架库，通过反射调用这些方法将无法模拟执行并抛出异常。为克服此限制，ParamScope 集成了 Robolectric[41] 所提供的包含了实际实现版本的 android.jar 来应对此挑战。

算法 2 值重建算法

输入：值依赖树根节点 T_{target}，对应调用点 c_{target}
输出：重建目标参数值集合 V_{target}
Procedure ReconstructValues(T_{target}, c_{target}):
1. $V_{objs} \leftarrow$ 空集;
2. $V_{target} \leftarrow$ 空集;
3. $v_{target} \leftarrow$ getTargetVar(c_{target});
4. $P \leftarrow$ getAllForwardPaths(T_{target});
5. **FOREACH** 路径 $p \in P$ **DO**:
6. **FOREACH** 节点 node∈p **DO**:
7. **FOREACH** 语句 s∈node **DO**:
8. **IF** containsMethodInvoke(s) **THEN**:
9. val ← interpretReflection (s, V_{objs});
ELSE
10. val ← interpretIR (s, V_{objs});
END IF
11. updateVal(val, V_{objs}, Def(s));
END FOREACH
END FOREACH
12. $V_{target} \cup V_{objs}$.get(v_{target});
END FOREACH
13. 返回 V_{target}

3.4 误用检测报告

在完成值重建后，ParamScope 根据值域一致性规则 R_{con} 和生成随机性规则 R_{ran} 判断是否存在误用。对

于值域一致性规则R_{con},直接验证重建的值是否符合安全规范。对于生成随机性规则R_{ran},ParamScope采用混合策略验证该值是否被安全随机化。首先在切片结果中匹配是否存在已知的不安全模式(如java.util.Random);然后在IR解释过程中执行"重复生成测试",即对参数生成逻辑解释模拟两次,若输出结果相同,则判定该值为固定值。

在报告阶段,ParamScope将生成所有检测到的误用报告、所有重建的目标参数值,为每个API调用点生成易读的数据流语句图。该数据流图使用DOT语言规范[42]描述,并通过Graphviz[43]进行可视化渲染。

4　评估

ParamScope由约5 000行Java代码实现,所有评估均在搭载Intel Xeon Gold 6338 CPU,并配备256 GB内存的Ubuntu 20.04服务器上运行。

本文评估围绕以下研究问题(RQ)展开。

RQ1:ParamScope在程序切片和值重建精确率如何?细化的程序切片和反射处理方法调用做出的贡献如何?

RQ2:ParamScope在基准数据集上是否比现有工具表现更好?

RQ3:ParamScope在真实世界应用数据集上是否比现有工具表现更好?

RQ4:ParamScope的检测效率如何?

RQ5:ParamScope能否识别现有工具遗漏的"值变化"案例?

本文将ParamScope和对比工具在基准数据集和Android应用数据集上进行4项评估,包括值重建、基准数据集性能检测、Android应用数据集性能检测、检测时间,以回答RQ1至RQ4,并总结值变化案例并补充案例验证以回答RQ5。

4.1　实验设置

4.1.1　数据集

在基准数据集中,本文选取了4个包含密码学误用的开源基准数据集。在后文表格中,将使用其缩写进行表示。

ApacheCryptoAPI-Bench(APA-B)[44]由8个包含密码学案例的真实Apache项目构成。

CryptoAPI-Bench(CRY-B)[44]关注常见的密码学API中的参数值传播模式(如直接值传播、基于字段的值传播和过程间值传播等)。

MASC-MinimalFlaws(MAS-B)[35]揭示静态检测器缺陷的最小案例集,这些案例通过12个突变测试类型生成。

OWASP-Bench(OWA-B)[45]由OWASP基金会开发,用于评估漏洞检测工具的效果,包含涉及密码学误用的漏洞案例。

在Android应用数据集中,本文选择了2024年Google Play社交、购物、视频和金融类别下载量前100名的应用。4个类别共400个应用,排除重复和其他格式的应用程序后,最终数据集包含327个与所有工具兼容的APK文件。

4.1.2　基准工具

基于近期包含静态Java密码误用检测器评估的学术研究[2,19,35,44,46],本文实验选择了两个现有的静态工具(CryptoGuard和CogniCrypt)和一个近期的动态工具(RVSec)。RVSec发布时间较新,采用基于运行时验证的方法,且现有研究[19]中已证明它能有效检测密码学API误用。

4.1.3　评估指标

本文将对不同实验的评估指标分别进行阐述。

对于切片精确率实验和值重建精确率实验,本文通过人工审查的方式建立了基准真值。在切片精确率实验中,人工验证每个用例切片结果是否精确地包含了源代码中完整的目标API参数赋值语义;在值重建精确率实验中,人工验证每个用例源代码中的目标API参数真实值和ParamScope值重建结果是否一致。上述验证结果分为"正确"和"错误"两种互斥类型,本文为了与后续误用检测评估保持一致,分别记作TP(True Positive)与FP(False Positive):TP指结果与源代码真实语义完全匹配,FP指结果与源代码真实语义存在偏差。静态分析工具的高代码覆盖率保证了覆盖目标API所有调用点,ParamScope将对所有调用点进行程序切片和值重建,因此不存在漏报,TN和FN为0。切片精确率和值重建精确率指标为精确率$=\frac{TP}{TP+FP}$。

在误用检测实验中,准确检测误用既要求准确的值重建,也需要明确实现的安全规则,因此误用检测是一个二元分类任务。其检测结果分为4种情况:TP(True Positive),即正确识别出的误用案例;FN(False Negative),即未被检测出的误用案例;FP(False Positive),即被错误分类为误用的正确使用案例;TN(True Negative),即正确识别出的正确使用案例。误用检测的评估指标如下:$\mathrm{Prec}=\frac{TP}{TP+FP}$,$\mathrm{Rec}=\frac{TP}{TP+FN}$,$\mathrm{Acc}=\frac{TP+TN}{TP+FP+TN+FN}$,$F_1=\frac{2\times\mathrm{Prec}\times\mathrm{Rec}}{\mathrm{Prec}+\mathrm{Rec}}$。

4.2　切片精确率与值重建精确率

4.2.1　切片精确率

此实验在基准数据集上检验ParamScope的程序切片精确率。同时,为了评估细化的程序切片对切片精确率的贡献,本文实现了一个移除了程序切片中针

对 $VT_{Ⅰ}$ 至 $VT_{Ⅲ}$（值变化语义）分析的变体，记作 ParamScope-NoVT。

表 2 实验结果记录了完整 ParamScope 与移除细化程序切片后的程序切片精确率。

表 2 ParamScope 及 ParamScope-NoVT 变体的切片精确率

Table 2 Slicing precision: ParamScope vs. ParamScope-NoVT

基准	案例数	ParamScope-NoVT		ParamScope	
		TP数	精确率/%	TP数	精确率/%
APA-B	99	90	90.91	98	98.99
CRY-B	268	144	53.73	257	95.90
MAS-B	36	9	25.00	34	94.44
OWA-B	1 382	1 034	74.82	1 351	97.76
总计	1 785	1 277	71.54	1 740	97.48

对切片不正确案例进行验证，45 例不准确切片案例的赋值行为依赖于程序控制流。其中，路径敏感用例在 CryptoAPI-Bench 中导致 11 例，在 OWASP-Bench 中导致 31 例。数组的循环赋值在 ApacheCryptoAPI-Bench 中导致 1 例切片不准确，在 MASC-MinimalFlaws 中导致 2 例切片不准确。

对比 ParamScope 及此变体的实验结果，由于该变体无法捕获方法调用及表达式中的赋值行为（VT 赋值模式），程序切片结果中遗漏了大量关键定义语句，总体切片精确率下降了 25.94%，表明了细化切片方法能够保证目标 API 参数赋值语义的完整性。

此外，为了验证切片的精简度，此实验统计了所有案例的切片 IR 指令数。ParamScope 的切片结果平均指令数仅为涉及方法指令的 12.7%。这表明，赋值驱动切片在保持语义完整性的同时，有效剔除了与参数生成无关的代码。

4.2.2 值重建精确率

此实验在基准数据集上检验 ParamScope 参数值重建的精确度。根据已知研究，ParamScope 是首个检测密码学 API 误用的同时报告参数值重建结果的静态分析工具。相较于 ParamScope，CogniCrypt 和 CryptoGuard 对参数值进行了一定程度的分析，但不包含完整的值重建步骤，因此无法评估其值重建精确率。

同时，为了评估 ParamScope 在值重建阶段“通过反射处理方法调用”对值重建精确率的贡献，本文实现了 ParamScope-NoRefl 变体，该变体移除了值重建阶段中“基于反射机制模拟方法调用赋值”的功能。

表 3 的实验结果记录了完整 ParamScope 与移除“反射分析方法调用”变体的值重建精确率。

表 3 ParamScope 及 ParamScope-NoRefl 变体的值重建精确率

Table 3 Reconstruction precision: ParamScope vs. ParamScope-NoRefl

基准	案例数	ParamScope-NoRefl		ParamScope	
		TP数	精确率/%	TP数	精确率/%
APA-B	99	90	90.91	96	96.97
CRY-B	268	169	63.06	255	95.15
MAS-B	36	11	30.56	34	94.44
OWA-B	1 382	1 034	74.82	1 351	97.76
总计	1 785	1 304	73.05	1 736	97.31

对值重建有误的案例进行验证，所有 49 个有误结果均由两个原因引起：第一，“控制流相关的赋值”原因，导致了 45 例切片不精确，进而导致值重建有误，与第 4.2.1 节误差分析相同；第二，部分案例依赖于基准集源代码中缺失的外部依赖值，在 CryptoAPI-Bench 中，缺失外部属性“pass.key”导致 2 例误报。在 ApacheCryptoAPI-Bench 中，缺失外部常量导致 2 例误报，且该缺失模块未在数据集中给出。

对比 ParamScope 及此变体的实验结果，变体在评估方法调用产生的赋值语句时，无法正确通过反射重建其赋值行为，导致目标 API 参数值重建中断。例如，MASC-MinimalFlaws 关注通过方法调用对参数值进行修改，移除反射组件后，值重建结果精确率仅为 30.56%。

4.3 基准数据集性能评估

此实验在基准数据集上评估 ParamScope 和各对比工具的误用检测效果指标。

表 4 的实验结果表明，ParamScope 取得了 96.22% 的准确率和 96.85% 的 F_1 分数，效果优于 CryptoGuard、CogniCrypt 和 RVSec。

表 4 ParamScope 与 CryptoGuard、CogniCrypt、RVSec 在基准数据集上的性能评估比较 单位：%

Table 4 Performance evaluation comparison of ParamScope, CryptoGuard, CogniCrypt, and RVSec on benchmark datasets unit: %

基准	ParamScope				CryptoGuard				CogniCrypt				RVSec			
	Prec	Rec	Acc	F_1	Prec	Rec	Acc	F_1	Prec	Rec	Acc	F_1	Prec	Rec	Acc	F_1
APA-B	98.08	96.23	95.65	97.14	80.00	60.38	57.97	68.82	81.97	94.34	79.71	87.72	97.62	77.36	81.16	86.32
CRY-B	92.20	98.48	92.17	95.24	85.71	81.82	74.70	83.72	83.02	100.00	83.73	90.72	95.38	93.94	91.57	94.66
MAS-B	100.00	100.00	100.00	100.00	100.00	72.73	73.53	84.21	96.77	90.91	88.24	93.75	100.00	93.94	94.12	96.88
OWA-B	97.70	96.40	96.82	97.04	92.77	82.58	87.08	87.37	63.79	92.42	67.49	75.48	100.00	49.05	72.41	65.82
总计	96.79	96.92	96.22	96.85	90.91	80.43	83.44	85.35	68.90	93.83	70.90	79.46	98.48	60.99	76.05	75.33

CogniCrypt 采用名为 CrySL 的领域特定语言（DSL）进行检测，该语言定义了 JCA 类的使用规范并将非规范行为标记为误用，这种基于白名单的检测策略使其仅达到 70.9% 的低准确率。CryptoGuard 采用

启发式策略降低误报率,实现了83.4%的更高准确率。RVSec采取了动态方法,实现了最高精确率,但由于动态方法的低覆盖率和对随机值进行验证规则的缺失,其召回率结果更低。相比之下,ParamScope在基准中平均检测准确率较CryptoGuard提升12.8%,较CogniCrypt提升了25.3%,表明ParamScope能更精确地解析“受值变化指令影响”的密码学API参数值,在检测更多误用案例的同时减少误报。

深入分析误报和漏报情况表明,由于采用了根据参数值和安全规则检测误用的方法,值重建的精度直接影响了误用案例判断的准确度。ParamScope所有的分析错误案例均可归因于值重建实验中的两个误差原因,所有的误报和漏报均由控制流敏感用例和外部缺失依赖引起。

4.4　Android应用数据集性能评估

此实验在Android应用数据集上对ParamScope、CryptoGuard和CogniCrypt进行误用检测以对比检测效果。考虑到RVSec动态分析需要额外配置和大量运行时开销,难以实现大规模自动化分析,因此此实验排除了RVSec。

在测试资源配置上,本实验为3个工具配置了相同的资源设置:最大堆大小为32×10^9,最大栈大小为2×10^6,分析超时时间为1 h。

表5描述了ParamScope与CryptoGuard和CogniCrypt这3个工具在327个真实应用中分析的误用实例数量。表5中,第一行“App结果数”表示工具分析成功的App数量(由于部分应用加壳、复杂度过高,分析工具可能发生崩溃、超时、无结果等现象);第二行和第三行“对比误用数”表示将ParamScope与CryptoGuard和CogniCrypt分别比较时工具检出的误用数量,并且为了确保分别比较的对比公平性,仅在两个工具均成功完成分析的“重合应用”集合中比较误用检出数量。

表5第一行“App结果数”结果表示,在总共327款App中,ParamScope、CryptoGuard和CogniCrypt分别成功分析了220款、150款和94款。

表5第二行“对比误用数1”表示ParamScope与CryptoGuard的对比结果。二者都分析成功136款应用,经本文验证,CryptoGuard报告517例误用,而ParamScope报告685例,检测效果提升约32%。

表5第三行“对比误用数2”表示ParamScope与CogniCrypt的对比结果。二者都分析成功69款应用,经本文验证,CogniCrypt报告了320例误用,而ParamScope报告407例,检测效果提升约27%。

4.5　检测时间

此实验在基准数据集和Android应用数据集上评估了ParamScope和对比工具的分析时间。

表5　ParamScope与对比工具在App数据集的误用检测

Table 5　Misuse detection comparison on the App dataset

	ParamScope	CryptoGuard	CogniCrypt
App结果数	220	150	94
对比误用数1	685	517	—
对比误用数2	407	—	320

表6在基准上进行的实验结果表明,在基准中,ParamScope实现了最高的平均检测效率,对基准的平均分析时间仅需44.5 s;CogniCrypt需要将CrySL规则转换为静态分析,基准平均分析时间为46.3 s;而CryptoGuard依赖复杂的启发式策略,平均分析时间为221.3 s。

表6　ParamScope、CogniCrypt、CryptoGuard的检测时间　单位:s

Table 6　Detection time of ParamScope, CogniCrypt, and CryptoGuard

unit: s

基准	ParamScope	CogniCrypt	CryptoGuard
APA-B	118.0	122.5	171.3
CRY-B	6.2	5.7	315.6
MAS-B	3.8	4.8	8.8
OWA-B	50.0	52.0	389.3
平均值	44.5	46.3	221.3

在进行App分析时,单个App的分析时间可以从3个阶段进行评估。(1)预处理,包括字节码转换(dex2jar)和SootUp初始化工作时间;(2)调用图构建,ParamScope构建程序调用关系及继承关系图分析时间;(3)误用分析,涵盖程序切片、参数值重建、误用检测及结果报告时间。

表7在Android应用数据集中的实验结果表明,ParamScope平均分析单个App耗时291 s(约4.85 min),分析时间中位数为184 s(约3.07 min)。此外,本文统计了各阶段的分析时间占比,其中调用图构建阶段平均占总分析时间的81.60%。结果表明,ParamScope在实际App中的分析效率与现有的静态方法相当。

表7　ParamScope在App数据集中的分析时间分布　单位:s

Table 7　Analysis time distribution of ParamScope on the App dataset

unit: s

分析阶段	最小值	最大值	平均值	中位值
预处理	1	2 583	46	14
调用关系构建	6	1 494	216	145
值误用分析	1	854	29	12
总分析时间	8	3 071	291	184

4.6　“值变化”参数值分析

ParamScope关注了现有静态分析工具在“值变化”参数分析的局限性,本节总结了针对Android应用数据集真实应用代码场景中“值变化”参数值案例的

分析,并补充了案例验证实验。

4.6.1 Android应用中的“值变化”参数值

在分析成功的220个App中,ParamScope在46个App中发现了以下3类“值变化”参数值。

(1)编码值。在25个App中,存在将参数值通过ASCII编码、Base64编码或自定义方法进行编码后,动态运行时解码使用的案例。

(2)隐藏数据。在16个App中,存在参数值通过包装类、数据类或集合等间接方式获取的案例。

(3)语义派生值。在23个App中,存在通过程序语义生成参数值的案例,包括StringBuilder拼接值、方法副作用影响的值以及哈希衍生的密钥。

接下来详细分析3个来自实际Android应用中的案例,这些参数值被ParamScope成功重建并正确报告,但现有工具存在误报或漏报。

代码2展示了一个下载量超500万次的流行购物应用中存在的编码值案例。经过Base64编码的字符串“UlNBL0VDQi9QS0NTMVBhZGRpbmc=”在程序运行时通过自定义方法解码为“RSA/ECB/PKCS1Padding”,指该程序应用了RSA公钥加密算法,采取电子密码本(ECB)模式和PKCS#1填充模式。Daniel Bleichenbacher提出的选择密文攻击[47]证明了此加密方案存在严重的安全风险。然而,现有静态工具均无法检测该案例,导致漏报。

代码2 案例1:编码值

Listing 2 Case 1: Encoded value

```
// 案例一,来自某超500万次下载量的流行购物应用
// Jimple中间表示:
$s4 = new java.lang.String
$s3 = staticinvoke <com.c.a.a.a.b: byte[] a(java.lang.String)>
("UlNBL0VDQi9QS0NTMVBhZGRpbmc=")
specialinvoke $s4.<java.lang.String: void <init>(byte[])>($s3)
l2 = staticinvoke <javax.crypto.Cipher: javax.crypto.Cipher getInstance(java.lang.String)>($s4)
// Java代码:
Cipher l2 = Cipher.getInstance(new String(com.c.a.a.a.b.a
("UlNBL0VDQi9QS0NTMVBhZGRpbmc=")));
// ParamScope值重建结果:
"RSA/ECB/PKCS1Padding"
```

代码3展示了一个下载量超1亿次的流行购物应用中存在的隐藏参数值案例:方法“a()”接受输入值“668”并返回一个密码算法名称字符串。逆向分析证实,类“a.a.a.a”内嵌了超100 KB的压缩数据,由方法“a()”接收索引值并检索对应的数据。此案例返回值对应算法类型“RSA with OAEPPadding”算法符合安全规范,但采用白名单策略的CogniCrypt无法验证该安全值,而错误地导致误报。

代码3 案例2:数据隐藏

Listing 3 Case 2: Hidden data

```
// 案例二,来自某超1亿次下载量的流行购物应用
// Jimple中间表示:
$stack4 = staticinvoke <a.a.a.a.a: java.lang.String a(int)>(668)
l3 = staticinvoke <javax.crypto.Cipher: javax.crypto.Cipher
getInstance(java.lang.String)>($stack4)
// Java代码:
Cipher l3 = Cipher.getInstance(a.a.a.a.a(668));
// ParamScope值重建结果:
"RSA/ECB/OAEPWithSHA-256AndMGF1Padding"
```

代码4展示了一个下载量超1亿次的流行购物应用中存在的语义派生值案例,该字节数组通过引用传递语义的代码在“Arrays.fill()”方法中被方法副作用置为全零,并被构建为固定密钥。ParamScope追踪了该赋值链,并准确重建出“SecretKeySpec”实例中的固定密钥。现有工具因未能结合语义构建值及引用类型分析而无法识别此类固定密钥,导致误报。

代码4 案例3:语义派生值

Listing 4 Case 3: Semantically derived value

```
// 案例三,来自某超1亿次下载的流行购物应用
// Jimple中间表示:
l3#0 = newarray (byte)[16]staticinvoke <java.util.Arrays: void
fill(byte[],byte)>(l3#0, 0)
specialinvoke l0#3.<javax.crypto.spec.SecretKeySpec: void
<init>(byte[],java.lang.String)>(l3#0, "AES")
// Java代码:
byte[] l3#0 = new byte[16];
Arrays.fill(l3#0, 0);
SecretKeySpec l0#3 = new SecretKeySpec(l3#0, "AES");
// ParamScope值重建结果:
"[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]"
```

4.6.2 案例验证

为验证这些“值变化”密码学API参数值对静态分析的影响,本文通过逆向工程针对案例研究中识别的每类情况构建了9例最小误用用例,并在ParamScope、CogniCrypt和CryptoGuard中测试,以验证这些功能能否检测到这些误用。这些案例包括以下3类:3例编码值(包含一个Base64编码的固定密钥、一个ASCII编码字符串及一个自定义解码实现案例);3例隐藏数据(包含将算法名称封装于密码专用类中、从数据类中获取及包装于List类中的案例);3例语义构建值(包含通过方法副作用修改引用类型、通过哈希指令派生固定密钥及通过StringBuilder构建值的案例)。

表8结果总结了各静态工具在这些代表性用例的检测结果,表明ParamScope通过解析以往工作未处理的“值变化”参数值,在所有从真实应用中构建的最小用例中均准确检出误用,有效提升了静态检测方法的准确性。

在逆向工程方法上,本文使用jadx、dex2jar[38]和IntelliJ反编译器对应用进行手工逆向分析,以理解各模式的代码逻辑并重建代表性案例。为确保符合伦理规范,本文仅关注与所观察到的赋值模式相关的代码,而避免对整个应用进行详细的逆向分析。

表8　静态检测器中的案例验证

Table 8　Case validation in static detectors

案例	CryptoGuard	CogniCrypt	ParamScope
Base64编码案例	○	◐	●
ASCII编码案例	○	○	●
自定义实现编码案例	○	○	●
字段存储字符串案例	●	●	●
数据类存储字符串案例	○	○	●
List类存储字符串案例	○	◐	●
引用类型修改案例	○	◐	●
Hash派生值案例	○	◐	●
StringBuilder构建字符串案例	○	○	●

注:○表示未检测到误用;◐表示检测到误用,但未精确检测出参数值;●表示检测到误用,且精确检测出参数值。

5 讨论

5.1 优势及理论分析

ParamScope在基准数据集与真实应用中展现出的优异性能,可归因于“赋值驱动的切片”与“IR模拟解释”的协同。具体而言,ParamScope具有以下优势。

(1)赋值驱动的程序切片构建了最小可执行语义依赖。从程序切片理论分析,相较于传统程序切片基于完整的数据流与控制流依赖计算切片语句和仅进行简单的变量追踪,ParamScope通过引入赋值模式约束P,构建了目标参数值的最小数据依赖指令集。切片精确率实验验证了ParamScope的切片策略在基准上的有效性及赋值模式约束P对切片策略的贡献。

(2)“IR解释执行”在切片构建的子程序上进行解释和模拟执行,并引入了反射机制来解释执行IR语句中的方法调用。现有静态方法主要基于抽象解释或数据流分析,在处理复杂的字符串操作时往往无法精确建模其语义而采取过近似策略,导致未知结果。ParamScope采取“IR解释模拟”精确重建了“值变化”指令(如字符串拼接、编码转换),消除了抽象语义不足的分析盲区。值重建精确率实验验证了值重建的有效性及反射机制对值重建过程的贡献。

5.2 局限性

尽管ParamScope效果出色,但仍面临静态分析中的两个长期挑战。首先,路径敏感性缺失。为了提高分析效率,本文和现有的工作相同,分析不具备路径的敏感性,可能导致切片过程包含无关路径或遗漏关键路径,并影响参数值重建的准确性。其次,别名问题。ParamScope未实现对变量别名的跟踪分析,若别名导致密码学API参数值被间接修改,可能导致值重建有误,引发误报和漏报,已有的静态工作也存在此类问题。总体而言,ParamScope的轻量级切片策略在精度与效率间取得了良好平衡,确保了实际应用的可行性。

5.3 有效性威胁

在内部有效性上,ParamScope的分析结果可能受到多种抗静态分析与代码保护技术的影响。由于其基于中间表示进行分析,结果的准确性依赖于SootUp和dex2jar的反编译质量。反编译对抗手段(如应用加壳)可能导致反编译失败,从而跳过分析。在外部有效性上,ParamScope针对JCA密码学库误用检测。在Java环境进行分析可能无法涵盖Android特定资源(如本地化字符串资源、Android安全类),从而影响值重建结果和误用检测精度。未来工作需针对Android平台特性进行分析,并扩展到其他密码学库。

5.4 未来工作

本文的未来工作将围绕以下方向展开。首先,为提升对Android特定误用检测的能力,将增强对Android特定误用(如“android.security”包中的安全相关类)的检测规则。其次,计划将检测范围拓宽至其他常用密码库(如Bouncy Castle),以扩展适用性。最后,针对Java反射机制在现代开发框架中的广泛应用,将研究通过反射实现的API调用的分析方法,以支持此类场景的检测。ParamScope高精度的参数值重建能力为反射API调用的参数分析提供了基础,有

助于实现对反射过程的深入检测。

6 结束语

本文提出并实现了首个能够同时处理值传播与值变化赋值约束的密码学 API 误用检测方法 ParamScope。该方法从目标 API 调用点出发，通过赋值模式驱动的细粒度程序切片，对获得的 IR 指令进行解释，模拟程序执行逻辑以重建参数值并识别密码学 API 误用。实验结果表明，ParamScope 的参数值重建达到了 97.31% 的精确率，其误用检测达到了 96.22% 的准确率，性能优于现有方法。在对 327 款热门 Android 应用的检测实验中，ParamScope 平均每款应用分析耗时 4.85 min。此外，ParamScope 检出了大量“受值变化指令影响的密码学 API 参数”的典型案例，且相比于 CryptoGuard 和 CogniCrypt 分别多检测出约 32% 和约 27% 的误用案例。因此，ParamScope 为 Java 密码学 API 参数值重建与误用检测提供了一种高精度、可扩展的静态分析解决方案。

参考文献

[1] Katz J, Lindell Y. Introduction to modern cryptography: principles and protocols[M]. Boca Raton: Chapman & Hall/CRC, 2007.

[2] Zhang Y, Kabir M M A, Xiao Y, et al. Automatic detection of Java cryptographic API misuses: Are we there yet?[J]. IEEE Transactions on Software Engineering, 2023, 49(1): 288-303.

[3] Meng N, Nagy S, Yao D D, et al. Secure coding practices in Java: Challenges and vulnerabilities[C]//Proceedings of the 40th International Conference on Software Engineering. New York: ACM, 2018: 372-383.

[4] Nadi S, Krüger S, Mezini M, et al. Jumping through hoops: Why do Java developers struggle with cryptography APIs[C]//Proceedings of the 38th International Conference on Software Engineering. New York: ACM, 2016: 935-946.

[5] Adrian D, Bhargavan K, Durumeric Z, et al. Imperfect forward secrecy: How diffie-Hellman fails in practice[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2015: 5-17.

[6] Durumeric Z, Li F, Kasten J, et al. The matter of heartbleed[C]//Proceedings of the 2014 Conference on Internet Measurement Conference. New York: ACM, 2014: 475-488.

[7] Bhargavan K, Leurent G. On the practical (In-) security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2016: 456-467.

[8] Egele M, Brumley D, Fratantonio Y, et al. An empirical study of cryptographic misuse in Android applications[C]//Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. New York: ACM, 2013: 73-84.

[9] Muslukhov I, Boshmaf Y, Beznosov K. Source attribution of cryptographic API misuse in Android applications[C]//Proceedings of the 2018 on Asia Conference on Computer and Communications Security. New York: ACM, 2018: 133-146.

[10] Wang J N, Guo S Q, Diao W R, et al. CrypTody: Cryptographic misuse analysis of IoT firmware *via* data-flow reasoning[C]//Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses. New York: ACM, 2024: 579-593.

[11] Rahaman S, Xiao Y, Afrose S, et al. CryptoGuard: High precision detection of cryptographic vulnerabilities in massive-sized Java Projects[C]//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2019: 2455-2472.

[12] Krüger S, Nadi S, Reif M, et al. CogniCrypt: Supporting developers in using cryptography[C]//2017 32nd IEEE/ACM International Conference on Automated Software Engineering. Piscataway: IEEE, 2017: 931-936.

[13] Krüger S, Späth J, Ali K, et al. CrySL: An extensible approach to validating the correct usage of cryptographic APIs[J]. IEEE Transactions on Software Engineering, 2021, 47(11): 2382-2400.

[14] Li Y, Zhang Y Y, Li J R, et al. iCryptoTracer: Dynamic analysis on misuse of cryptography functions in iOS applications[C]//Network and System Security. Cham: Springer, 2014: 349-362.

[15] Piccolboni L, Di Guglielmo G, Carloni L P, et al. CRYLOGGER: Detecting crypto misuses dynamically[C]//2021 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2021: 1972-1989.

[16] Stevens M, Bursztein E, Karpman P, et al. The first collision for full SHA-1[C]//Advances in Cryptology - CRYPTO 2017. Cham: Springer, 2017: 570-596.

[17] Vanhoef M, Piessens F. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS[C]//Proceedings of the 24th USENIX Security Symposium. Washington: USENIX Association, 2015: 97-112.

[18] Ami A S, Moran K, Poshyvanyk D, et al. “False negative - that one is going to kill you”: Understanding industry perspectives of static analysis based security testing[C]// 2024 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2024: 3979-3997.

[19] Torres A, Costa P, Amaral L, et al. Runtime verification of crypto APIs: An empirical study[J]. IEEE Transactions on Software Engineering, 2023, 49(10): 4510-4525.

[20] 杜瑞颖, 陈晶, 吴聪, 等. 基于敏感组件函数调用图的安卓重打包恶意软件检测方法[J]. 电子学报, 2025, 53(7): 2372-2388.
Du Ruiying, Chen Jing, Wu Cong, et al. A detection method for Android repackaged malware based on sensitive component function call graph[J]. Acta Electronica Sinica, 2025, 53(7): 2372-2388. (in Chinese)

[21] Biringa C, Kul G. Detecting hard-coded credentials in software repositories via LLMs[J]. Digital Threats: Research and Practice, 2025, 6(3): 1-16.

[22] Alfardan N J, Bernstein D J, Paterson K G, et al. On the security of RC4 in TLS[C]//Proceedings of the 22nd USENIX Security Symposium. Berkeley, CA: USENIX Association, 2013: 305-320.

[23] Stevens M, Lenstra A, De Weger B. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities[M]//Advances in Cryptology - EUROCRYPT 2007. Berlin: Springer, 2007: 1-22.

[24] RFC2898: PKCS #5: Password-based cryptography specification version 2.0[S].

[25] Al Fardan N J, Paterson K G. Lucky thirteen: Breaking the TLS and DTLS record protocols[C]//2013 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2013: 526-540.

[26] Lenstra A K, Verheul E R. Selecting cryptographic key sizes[J]. Journal of Cryptology, 2001, 14(4): 255-293.

[27] RFC 5639: Elliptic curve cryptography (ECC) brainpool standard curves and curve generation[S].

[28] RFC3447: Public-key cryptography standards (PKCS) #1: RSA cryptography specifications version 2.1[S].

[29] Gutmann P. Software generation of practically strong random numbers[C]//Proceedings of the 7th conference on USENIX Security Symposium - Volume 7. New York: ACM, 1998: 19.

[30] Heninger N, Durumeric Z, Wustrow E, et al. Mining your Ps and Qs: Detection of widespread weak keys in network devices[C]//Proceedings of the 21st USENIX Conference on Security Symposium. New York: ACM, 2012: 35.

[31] Krawczyk H. How to predict congruential generators[J]. Journal of Algorithms, 1992, 13(4): 527-545.

[32] Karakaya K, Schott S, Klauke J, et al. SootUp: A redesign of the soot static analysis framework[C]//Tools and Algorithms for the Construction and Analysis of Systems. Cham: Springer, 2024: 229-247.

[33] SootUp. Jimple[EB/OL]. [2025-12-31]. https://soot-oss.github.io/SootUp/v1.1.2/jimple/.

[34] Vallée-Rai R, Gagnon E, Hendren L, et al. Optimizing Java bytecode using the soot framework: Is it feasible?[C]// Compiler Construction. Berlin: Springer, 2000: 18-34.

[35] Ami A S, Cooper N, Kafle K, et al. Why crypto-detectors fail: A systematic evaluation of cryptographic misuse detection techniques[C]//2022 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2022: 614-631.

[36] Ami A S, Ahmed S Y, Redoy R M, et al. MASC: A tool for mutation-based evaluation of static crypto-API misuse detectors[C]//Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2023: 2162-2166.

[37] Reardon J, Feal Á, Wijesekera P, et al. 50 ways to leak your data: An exploration of Apps’ circumvention of the Android permissions system[C]//Proceedings of the 28th USENIX Security Symposium. Berkeley, CA: USENIX Association, 2019: 603-620.

[38] Pan B. Dex2jar v2.1[EB/OL]. [2025-02-07]. https://github.com/pxb1988/dex2jar.

[39] Weiser M. Program slicing[J]. IEEE Transactions on Software Engineering, 1984, SE-10(4): 352-357.

[40] 刘天阳, 石剑君, 叶嘉威, 等. P-Slicer: 面向路径表示学习的程序切片方法[J]. 电子学报, 2025, 53(11): 3894-3909.
Liu Tianyang, Shi Jianjun, Ye Jiawei, et al. P-Slicer: A program slicing approach based on learning path representations[J]. Acta Electronica Sinica, 2025, 53(11): 3894-3909. (in Chinese)

[41] Robolectric. Robolectric[EB/OL]. [2025-02-17]. https://robolectric.org/.

[42] Gansner E R, Koutsofios E, North S. Drawing graphs with dot[EB/OL]. (2015-01-05) [2026-03-10]. https://www.graphviz.org/pdf/dotguide.pdf.

[43] Ellson J, Gansner E, Koutsofios L, et al. Graphviz: Open source graph drawing tools[M]//Graph Drawing. Berlin: Springer, 2002: 483-484.

[44] Afrose S, Xiao Y, Rahaman S, et al. Evaluation of static vulnerability detection tools with Java cryptographic API benchmarks[J]. IEEE Transactions on Software Engineering, 2023, 49(2): 485-497.
[45] OWASP benchmark project[EB/OL]. [2025-02-17]. https://owasp.org/www-project-benchmark.
[46] Chen Y, Liu Y, Wu K L, et al. Towards precise reporting of cryptographic misuses[C]// Proceedings of the Network and Distributed System Security Symposium (NDSS). San Diego: Internet Society, 2024: 241032.
[47] Bleichenbacher D. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1[C]// Advances in Cryptology - CRYPTO '98. Berlin: Springer-Verlag, 1998: 1-12.

作者简介

何亚非 男,2001年9月出生于四川省巴中市。现为武汉大学国家网络安全学院硕士研究生。主要研究方向为软件安全。
E-mail: yaf2023@whu.edu.cn

傅建明 男,1969年9月出生于湖南省宁乡市。现为武汉大学国家网络安全学院教授。主要研究方向为系统安全、网络安全等。
E-mail: jmfu@whu.edu.cn

占力戈 男,1996年3月出生于福建省南平市。现为武汉大学国家网络安全学院博士研究生。主要研究方向为软件安全。
E-mail: ligezhan@whu.edu.cn

彭国军 男,1979年11月出生于湖北省荆州市。现为武汉大学国家网络安全学院教授。主要研究方向为网络与信息系统安全等。
E-mail: guojpeng@whu.edu.cn

聂 宇 男,1986年11月出生于江西省景德镇市。现为武汉大学国家网络安全学院博士研究生。主要研究方向为移动安全隐私保护、软件安全、数据安全等。
Email: yu.nie@whu.edu.cn