# 面向小程序模板的漏洞挖掘方法研究

杨雲腾，史一哲，杨哲慜

（复旦大学 计算机科学技术学院 系统软件与安全实验室，上海 200433）

E-mail：yangzhemin@fudan.edu.cn

**摘 要：**小程序模板在开发过程中被广泛使用，但其固有的安全漏洞导致大量模板小程序面临安全风险。由于小程序模板代码闭源且使用情况不透明，现有方法难以对小程序模板固有漏洞以及衍生的小程序漏洞进行挖掘。为此，本文提出了一种面向小程序模板的融合多阶段聚类分析和模板行为分析的漏洞挖掘方法。该方法通过提取代码特征汇总相同模板开发的小程序，采用基于图匹配的启发式方法来识别相似的程序行为，即模板行为，以此构建小程序模板过程间控制流图，从而快速定位并系统性评估模板开发行为导致的小程序漏洞及其扩散现象。基于该方法本文实现了一个面向小程序模板的漏洞挖掘工具 MTVMiner，从小程序集中提取出 11468 套小程序模板过程间控制流图，准确率达 91.23%，并将其用于密钥泄漏漏洞检测，检出 690 套存在密钥泄漏漏洞的模板，影响超 2 万个小程序。

**关键词：**小程序模板；图匹配；数据流分析；漏洞检测

**中图分类号：**TP311 **文献标识码：** A

## Research on Vulnerability Mining Method for Mini-program Templates

YANG Yunteng, SHI Yizhe, YANG Zhemin

（System and Security Laboratory, School of Computer Science, Fudan University, Shanghai 200433, China）

**Abstract:** Mini-program templates are widely used in the development, but their inherent vulnerabilities expose a large number of mini-programs to security risks. Due to the closed-source nature of mini-program template code and the lack of transparency in their usage, existing methods struggle to detect both the inherent vulnerabilities in the templates and the derived vulnerabilities in mini-programs. To address this, this paper proposes a vulnerability mining method for mini-program templates, which combines multi-stage clustering analysis and template behavior analysis. This method extracts code features to group mini-programs developed by same template, and uses a graph-matching-based heuristic approach to identify similar program behaviors, i.e., template behaviors for constructing an inter-procedural control flow graph of mini-program templates. By analyzing inter-procedural control flow graph of templates, this method can quickly locate and systematically evaluates mini-program vulnerabilities and their spread caused by template development. Based on this method, we have implemented a vulnerability mining tool called MTVMiner for mini-program template, which extracts 11,468 inter-process control flow graphs from collection of mini-programs with an accuracy rate of 91.23%. MTVMiner has been used for key leakage vulnerability detection, identifying 690 templates with key leakage vulnerabilities, affecting over 20,000 mini-programs.

**Keywords:** MiniApp template; graph matching; data flow analysis; vulnerability detection

## 0 引 言

小程序以其易用和易开发的特点得到市场青睐并依托于宿主应用迅速发展，目前已成为主流的应用形态。据统计数据[1]显示，仅 2024 年 6 月，微信小程序的月活用户数量达到 9.3 亿。

随着小程序用户规模的持续增长，大量线下实体商家希望利用小程序进行引流，但绝大多数商家缺乏小程序开发能力。为此，微信平台开放小程序服务市场[2]，允许第三方服务商为商家提供小程序开发服务。为提高开发效率，第三方服务商会根据应用场景预设小程序的结构和基本功能，包括页面布局、功能模块和样式设计，以便商家根据需求进行定制和扩展。这种预设的小程序被称为小程序模板，而在此基础上进行定制化开发的小程序则被称为模板小程序。

收稿日期：2023-12-30 基金项目：国家自然科学基金项目（62172104）资助. 作者简介：**杨雲腾**，男，2000 年生，硕士研究生，研究方向为小程序安全研究；**杨哲慜**，男，1984 年生，博士，副教授，CCF 会员，研究方向为隐私保护、恶意攻击感知、软件漏洞发现。

然而，部分第三方服务商安全意识薄弱，其开发的小程序模板存在安全漏洞。随着这类模板被广泛使用，其自身的安全漏洞也随之大范围扩散，导致模板小程序的数据和业务安全遭受威胁。例如，Li[3]等人发现，采用相同模板开发的小程序存在相似的隐私过度分享行为，影响数百个小程序，造成大量用户隐私数据泄漏。因此，如何高效挖掘小程序模板固有漏洞及其衍生的小程序漏洞，仍是当前亟待解决的安全问题。

出于商业保护的考虑，小程序模板代码通常闭源，这使得现有的漏洞挖掘方法[4-6]缺乏明确的分析对象。因此，小程序模板本身的安全漏洞只能通过分析具体的模板小程序实例来发现。由于小程序模板使用信息不透明，本文难以像传统组件复用检测[18-19]那样，直接检索和定位相关的模板小程序。此外，模板小程序的定制化代码和模板代码高度耦合，漏洞溯源的复杂性进一步提高，漏洞源自小程序模板还是其定制化代码变得难以界定。

针对上述问题，本文提出了一种面向小程序模板的融合多阶段聚类分析和模板行为分析的漏洞挖掘方法。首先，基于模板小程序通常由第三方服务商开发以及相同模板开发的小程序具有相似的代码特征这一观察，本文筛选出潜在的模板小程序并设计了基于页面布局、代码逻辑和代码交互的多阶段聚类分析，将相同模板开发的小程序归类至同一簇中。然后，本文采用基于图匹配的启发式方法，从相同模板开发的小程序的过程间控制流图（ICFG, Interprocedural Control Flow Graph）中识别模板行为，即公共的节点和控制流，并以此构建小程序模板过程间控制流图。在构建小程序模板过程间控制流图的过程中，部分非公共节点和控制流被舍弃，导致小程序模板过程间控制流图中部分数据流缺失或断裂。为此，本文基于数据流映射分析对小程序模板过程间控制流图中的数据传播进行补充和修复，以提高其在真实安全分析的可用性。最后，本文以密钥泄漏检测为例对小程序模板固有漏洞进行挖掘，并基于模板固有漏洞的触发路径高效识别模板开发行为衍生的小程序漏洞。

基于该方法，本文设计并实现了一款面向小程序模板的漏洞挖掘工具 MTVMiner。该工具能够提取小程序代码特征并按照模板类型对数据集中的小程序进行高效且准确的分类，然后采用基于图匹配的启发式方法从相同模板开发的小程序中识别模板行为，以构建小程序模板过程间控制流图，进而用于漏洞挖掘。本文在小程序数据集中对 MTVMiner 进行了测试和评估，成功提取出 11468 套小程序模板过程间控制流图。通过抽样评估，MTVMiner 在多阶段聚类分析中的准确率达到 97.32%，在模板行为分析中的准确率达到 91.23%。本文基于小程序模板过程间控制流图实现了一个密钥泄漏漏洞检测工具，并对提取出的小程序模板过程间控制流图进行扫描，结果显示 690 套模板存在密钥泄漏漏洞，占提取模板总量的 6.02%，影响 21755 个小程序。实验结果表明，本文提出的漏洞挖掘方法能有效挖掘小程序模板固有漏洞及其衍生的小程序漏洞，同时揭示了模板开发导致的漏洞扩散现象具有一定的普遍性。

## 1 相关工作

### 1.1 小程序静态分析

静态分析是指在不运行程序的条件下，进行程序分析的方法。静态分析广泛应用于代码优化[7]、隐私泄漏检测[8]等场景。静态分析会根据语言特性、应用特点，分析目标，设计相应的算法对程序进行分析。

小程序代码由前端和后端两部分组成：1）前端代码运行在客户端，与用户直接交互，能够处理系统资源；2）后端代码主要是实现数据存储和提供小程序云服务。由于后端代码基本无法获取，目前主要研究对象均为前端代码。小程序前端代码会以 wxapkg 的形式发送到客户端，如图 1 所示。wxapkg 的结构主要由全局配置文件、资源文件以及页面文件三部分组成。小程序模板定制化开发是围绕页面文件进行，其中 XML 布局文件和 JS 交互逻辑文件最为重要，并且 XML 布局文件和 JS 交互逻辑文件也是静态分析的重要输入。

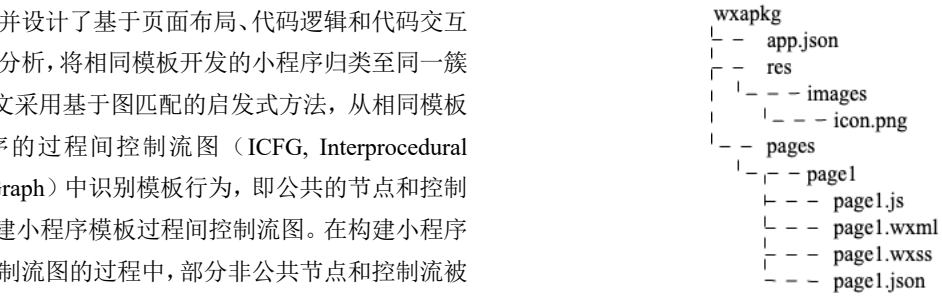


图 1 wxapkg 文件代码结构

Fig.1 Code structure of wxapkg

小程序使用 JS 语言作为开发语言，因此 Soheil[10]等人设计的 JS 静态分析工具 JAW 同样适用于小程序。JAW 通过构建 JS 程序的混合属性图进行静态分析。针对小程序本身的跨语言、跨页面和跨小程序等特性，Wang[11]等人设计了一款小程序静态分析工具 TaintMini。TaintMini 通过构建一种全新的通用数据流图来展示小程序间以及小程序页面间的数据流关系，使得小程序间的污点分析更加精确。由于小程序框架代码闭源，现有工具无法补全框架代码和交互代码之间的数据流和控制流信息。为此，叶瀚[12]等人利用知识图谱技术对小程序官方文档进行数据抽取来指导小程序函数调用图的精确构建。

现有的小程序静态分析技术真要针对单个小程序应用进行分析，而本文提出的模板行为分析方法则需要对多个小程序的程序行为进行差异分析，分析过程更为复杂。

### 1.2 小程序漏洞研究

目前小程序相关研究工作已经披露许多高质量漏洞，根据漏洞触发的场景，分为前端漏洞和混合漏洞。

前端漏洞是指在用户界面（UI）或客户端代码中存在的安全问题。小程序交互逻辑代码由 JS 编写，传统的 JS 空指针问题同样会出现在小程序前端。Liu[13]等人在已有 JS 动态分析框架基础上实现 WeJalangi 以用于微信小程序的动态分析，并对开源小程序代码的空指针问题进行可用性测试。Lu[14]等人探究了宿主应用资源管控的问题，并介绍了新型的用户界面欺骗攻击模型：宿主应用并未与小程序完全隔离，恶意小程序可以伪装宿主应用的用户界面来窃取用户敏感信息。Yang[15]等人探究了小程序间的通信机制，发现许多小程序接受外部数据而不对数据发送方进行身份校验，提出跨小程序请求伪造攻击模型并设计工具 CMRFScanner 对该漏洞的在野情况进行评估。

混合漏洞同时涉及前端和后端的安全问题，结合利用会造成更为严重的危害。Zhang[16]等人探究了小程序主密钥泄漏的问题，并就密钥前端泄漏导致的账号劫持、促销滥用、服务窃取等后端安全问题进行说明。Zhang[17]等人探究了小程序对宿主应用的特权接口访问机制，并就域名、appid、功能三种校验机制的缺漏提出了三种身份混淆攻击。

本文的核心目标并非研究小程序漏洞，而是提出一种针对小程序模板固有漏洞及其衍生的小程序漏洞的挖掘方法。为验证方法有效性，本文会以真实漏洞为例进行实验评估。

### 1.3　第三方组件检测及其风险识别研究

小程序模板开发本质上第三方服务商借鉴传统的组件复用理念所进行的一种软件开发实践。因此，小程序模板开发导致的漏洞扩散，本质上是由组件复用所导致的软件供应链安全问题。目前，已有大量研究工作聚焦于应用软件中第三方组件的检测和风险识别。

针对第三方组件的检测，目前工作主要分为两种。一种是已知第三方组件特征对未知应用软件进行匹配识别，例如 Wu[18]等人通过分析代码特征来构建安卓应用与第三方库之间的类对应关系，并使用函数及其调用链操作码的相似性提高类对应关系的准确率，以对抗各类代码混淆。另一种则是直接对应用软件中的第三方组件进行提取，例如 Li[19]等人利用安卓应用内部代码的依赖性来检测和分类候选库，并利用特征哈希来对抗第三方库的混淆问题。上述方法都是通过先检测第三方组件，再根据第三方组件的漏洞报告去判断使用该组件的应用软件是否存在安全风险。Ohm[20]等人则基于使用恶意第三方组件的应用在安装过程中会引入大量新的第三方组件这一观察进行实验研究，并设计了动态分析工具 BuildWatch。

由于小程序模板闭源，无法直接获取其内部知识，且模板代码与定制化代码高度耦合，导致现有的第三方组件检测方法和风险识别方法难以适用于小程序模板。

## 2　基本架构

为能够系统性研究小程序模板固有安全漏洞及其引发的小程序漏洞，本文提出了一种融合多阶段聚类分析和模板行为分析的面向小程序模板的漏洞挖掘方法，并基于该方法实现了一款漏洞挖掘工具 MTVMiner，其系统架构如图 2 所示。

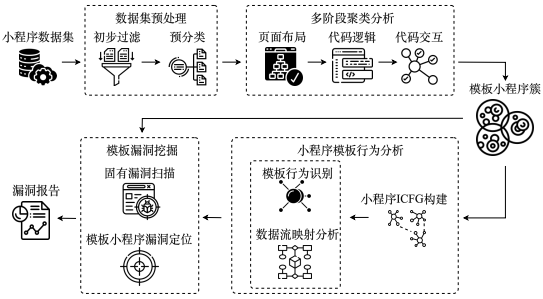


图 2　MTVMiner 系统架构示意图

Fig.2　Overall architecture of MTVMiner

在多阶段聚类分析前，MTVMiner 会基于小程序的授权信息筛选出潜在的模板小程序并基于授权服务商信息的对其进行预分类。在多阶段聚类分析模块中，MTVMiner 根据小程序的页面布局、代码逻辑和代码交互三个代码特征来判断其是否由相同模板开发。由于小程序数据集规模庞大，MTVMiner 分阶段提取小程序的特征，逐步进行聚类分析，从而显著提高聚类效率。最后，MTVMiner 将相同模板开发的小程序汇总至同一个簇。

在模板行为分析模块中，MTVMiner 首先从小程序簇中抽样部分模板小程序，并分别构建其过程间控制流图。接着，MTVMiner 对抽样的模板小程序过程间控制流图进行模板行为识别，即通过图匹配技术分析其中公共节点和控制流，从而构建出小程序模板过程间控制流图。此外，MTVMiner 将原始模板小程序和小程序模板的过程间控制流图进行数据流映射分析，修复小程序模板过程间控制流图提取过程中断裂的数据流，以提高其在漏洞挖掘中的准确性和可用性。

在模板漏洞挖掘模块中，MTVMiner 首先利用小程序模板的过程间控制流图去挖掘其固有漏洞，即模板漏洞。若检测出小程序模板存在安全漏洞，MTVMiner 则对该模板开发的小程序进行漏洞定位。在漏洞定位过程，MTVMiner 根据模板漏洞的触发范围，对小程序进行局部的过程间调用图构建，以提高漏洞检测效率。由于小程序漏洞的多样性，本文仅以常见的小程序密钥泄漏为例进行实验分析和评估。

## 3　设计与实现

### 3.1　多阶段聚类分析

由于模板使用信息不透明，本文需要基于庞大的小程序数据集进行分析，以覆盖尽可能多的小程序模板。鉴于小

程序数据集规模庞大，本文采用多阶段递进式的聚类方式，逐步缩小聚类范围，从而提高聚类效率。

3.1.1 数据集预处理

为了提高分析效率，MTVMiner 对初始小程序数据集进行预处理，主要分为模板小程序过滤和预分类两部分。小程序的第三方服务商以及授权信息会在其“更多资料”模块中公开。基于小程序是否授予第三方服务商开发权限，MTVMiner 过滤出潜在的模板小程序。考虑到相同服务商开发的小程序通常使用其自研的模板，MTVMiner 根据服务商信息对潜在的模板小程序进行预分类。

3.1.2 基于页面布局的聚类分析

针对相近的业务场景，第三方服务商通常使用相同小程序模板进行定制化开发。业务场景中的“场景”可以理解为用户与小程序交互的页面。因此，本文推断相同模板开发的小程序其页面结构高度相似性。由于小程序的页面和自定义组件代码都是由 JS、WXML、WXSS 和 JSON 四部分组成，MTVMiner 需将两者区分。理想情况下，MTVMiner 可从全局配置文件中直接提取小程序的页面路径。然受限于解包工具，部分小程序的全局配置文件可能无法解析。MTVMiner 通过检查页面和组件对应 JSON 文件中的 components 属性进行区分，如果 components 属性为 True，表示其为自定义组件；反之，则为页面。基于上述方法，MTVMiner 成功获取绝大多数小程序的页面路径。

基于提取到的小程序页面路径，MTVMiner 采用深度优先遍历算法遍历页面 WXML 文件的标签序列，自定义组件标签会被替换为其 WXML 文件的标签序列。如果两个页面文件的标签序列相似度超过一定阈值，本文认为这两个页面相互映射。根据两个小程序映射页面数量和页面总数，本文能够计算小程序间的页面相似度。基于小程序间的页面相似度，MTVMiner 采用聚类算法对小程序集进一步划分，并将同一簇中小程序间的页面映射表输出。为保证聚类效果，本文为聚类算法添加相似度约束，即簇中小程序间的相似度必须超过一定阈值，后续的聚类分析同样适用。

3.1.3 基于代码逻辑的聚类分析

由于相同页面布局背后的代码逻辑可能完全不一致，MTVMiner 基于代码对簇中小程序进一步聚类。由于小程序的逻辑代码是以函数块形式存在，MTVMiner 将小程序代码划分成函数粒度的比较单元，来计算小程序间的相似度。此外，本文整理了小程序中常见的通用组件，如表 1 所示，并将其作为噪声进行过滤，以提高小程序间相似度计算的准确率。

为提高相似函数映射效率，MTVMiner 将函数分为页面函数和工具函数。页面函数是指页面的逻辑代码，必须与其映射页面的函数进行匹配，自定义组件的函数则归类到使用自定义组件的页面；工具函数是指除页面函数外其余函数，通常是工具类函数，必须与其他小程序的工具函数进行匹配。本文提取函数基本特征作为摘要，用三元组（函数名，参数长度，语句类型序列）表示。如果两个函数参数长度相同，语句类型序列相似度超过设定阈值，本文认为这两个函数相互映射。在函数映射过程中，MTVMiner 优先对同名函数进行匹配操作，若同名函数映射，则直接将其添加至函数映射表；其余未成功映射的函数进行交叉匹配，若存在某个函数与多个函数都符合映射条件，取语句类型序列相似度最高的函数作为映射函数并添加至函数映射表。根据两个小程序映射函数数量和函数总数，本文能够计算小程序间的代码相似度。基于小程序间的代码相似度，MTVMiner 采用聚类算法对小程序集进一步划分，并将同一簇中小程序间的函数映射表输出。

表 1 第三方 JS 文件中的关键字

Table1 Keywords within third-party JS

| 文件类型 | 函数文件名关键字 |
|---|---|
| SDK 文件 | sdk, min.js, ald-stat, wxcharts.js, bmap, qqmap, amap |
| 通用工具类 | aes.js, des.js, base64.js, rsa.js, rsa_encry.js, rsaEncrypt.js, wxparse |
| 组件或库文件 | @, __, miniprogram_npm, node_modules, npm/, lib/, libs/, bundle.js, taro.js, vendor.js, runtime.js |

3.1.4 基于代码交互的聚类分析

即使两个函数在函数摘要层面相似，但其交互方式的差异仍可能导致错误的聚类结果。为此，MTVMiner 利用小程序间函数调用链的相似度来对小程序集进一步划分。MTVMiner 使用 JAW[10]提取小程序的显式调用链，并基于叶瀚[12]等人抽取的框架控制流信息来提取小程序的隐式调用链。在提取到的隐式调用链中，本文仅考虑 UI 控件与小程序函数间的调用链以及小程序函数与框架函数间的调用链，并不考虑框架函数间的调用链。

本文归纳了三种调用链类型，分别是小程序函数间的调用链，UI 控件与小程序函数间的调用链以及小程序函数与框架函数间的调用关系，表 2 列举图 3 中出现的这三类调用链。针对这三种函数调用链，本文对其映射规则进行如下定义：

1）小程序函数间的调用链映射：两条调用链的调用函数和被调函数符合 3.1.3 函数相互映射条件。

2）控件与小程序函数间的调用链映射：两条调用链的控件其类型和所属页面相同，控件绑定的小程序函数符合相互映射条件。

3）小程序函数与框架函数间的调用链映射：由于框架函数很少被混淆，两条调用链使用的框架函数相同且小程序函数符合函数相互映射条件。

根据两个小程序映射调用链数量和调用链总数，本文能够计算小程序之间的调用链相似度。基于小程序间的调用链相似度，MTVMiner 采用聚类算法对小程序集进一步划分，并将同一簇中小程序间的调用链映射表输出。

表 2　小程序函数调用链示例

Table 2　Example of miniapp function call chain

| 调用链类型 | 调用链示例 |
|---|---|
| 小程序函数间 | confirm → addCountryCode |
| 控件与小程序函数 | < button bindtap = “toSubmit” > → toSubmit |
| 小程序与框架函数 | toSubmit → wx.navigateTo |

## 3.2　模板行为分析

### 3.2.1　　小程序过程间控制流图构建

本文采用过程间控制流图来描述小程序的整体程序行为，从而便于从多个模板小程序中识别相似的程序行为。首先，MTVMiner 利用 Esprima[21]工具将函数转换为抽象语法树，并对语句重新建模，将其表示为节点对象。之后，MTVMiner 利用 Stxy[22]工具生成语句间的控制流关系连接语句节点。基于 3.1.4 中提取到小程序显式和隐式调用链，MTVMiner 将页面函数、页面控件和框架函数进行连接。最后，MTVMiner 对特定框架接口进行建模，实现跨页面函数连接，进一步增强小程序过程间控制流图的完整性。

图 3 展示了小程序数据收集页面和数据确认提交页面的过程间控制流图构建过程。首先，MTVMiner 将页面中逻辑函数分别进行控制流图构建，其中节点会根据语句类型重新建模，并包含语句关键信息，控制流关系用实线表示。随后，MTVMiner 对函数延伸的调用边进行补充。若函数被控件节点和同页面其他函数调用，则用实线将两个节点进行连接，例如图 3 中的 confirm 与 addCountryCode 函数。最后，MTVMiner 根据页面跳转接口和依赖引入接口建模的结果，用虚线连接页面间函数的调用边。

### 3.2.2　　基于图匹配的模板行为识别

模板行为是指在基于相同模板开发的小程序中，共同展现的程序行为特征。这些行为体现在过程间控制流图中，表现为相似的节点和控制流结构。为此，本文采用基于图匹配的启发式算法，逐步匹配同一簇中模板小程序过程间控制流图的节点和控制流，识别其中公共节点和控制流，并以此构建小程序模板的过程间控制流图。此外，本文发现当参与模板行为识别的小程序达到一定数量时，构建出的模板过程间控制流图趋于稳定。因此，本文随机抽样部分模板小程序进行模板行为识别，具体细节将在实验与评估中阐述。

随着小程序功能的拓展，小程序过程间控制流图中节点和控制流的数量相应增多，不可避免地导致图匹配算法的时间和空间复杂度升高。为此，MTVMiner 对小程序过程间控制流图进行拆分，拆分为函数控制流图及其延伸的调用边。在后续的图匹配过程中，MTVMiner 利用聚类分析生成的函数映射表，对映射函数之间的控制流图及其延伸的调用边进行匹配分析，从而有效降低图匹配算法的复杂度。

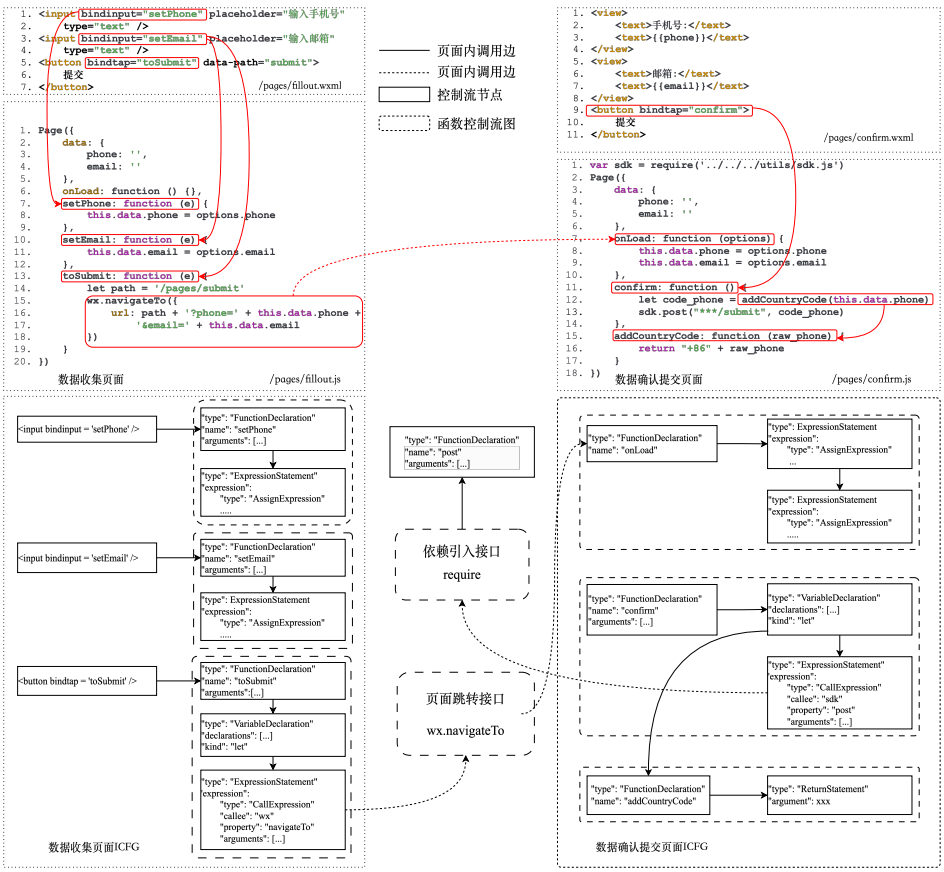


图 3　小程序过程间控制流图构建

Fig.3　Construction of miniapp interprocedural control flow graph

函数控制流图延伸的调用边匹配相对简单，因为调用边的另一端只有两种情况：控件节点和另一个函数控制流图。对于第一种情况，只需判断控件所在页面是否映射且控件类型是否相同。若两个条件都满足，则认为该调用边为公共边并予以保留；否则，删除该调用边。对于第二种情况，只需根据函数映射表判断是否存在映射关系。若条件满足，则认为该调用边为公共边并予以保留；反之，删除该调用边。

函数控制流图匹配的复杂度则取决于函数的内部结构。如图 3 所示，数据收集页面的函数 toSubmit，其控制流图由三条语句节点构成，且这三条语句节点是顺序执行的，形成了一个单向链表结构。在这种情况下，函数控制流图匹配算法相对简单，只需按顺序匹配图中的公共节点，并依次连接这些节点，即可生成公共函数控制流图。如图 4 所示，小程序的数据处理函数包含复杂的条件判断语句，其对应的控制流图结构存在复杂的分支。若采用完全遍历的方式去匹配分支中的语句节点，必然会降低图匹配算法的效率。本文观察发现，函数控制流图的复杂结构是由控制流语句造成的，包括循环语句、条件语句、跳转语句和异常处理语句，如表 3 所示。因此，本文将函数控制流图中控制流语句节点进行提取，并构建函数的控制依赖图，如图 4 所示。控制依赖图相较于之前的函数控制流图，分支减少近一半。因此，函数控制流图的匹配便可以转为函数控制依赖图的匹配。MTVMiner 首先对控制依赖节点进行匹配，并对控制依赖节点的匹配规则进行如下定义：若两个控制依赖节点需满足语句类型相同，且其循环或控制条件所使用的表达式一致，则两个控制依赖节点相互映射。此外，对于异常处理语句，需确保捕获的异常或错误类型相同。根据上述控制依赖节点匹配规则，MTVMiner 识别出函数控制流图中的公共控制依赖节点，并基于这些节点提取出控制依赖节点间的公共边，生成公共的控制依赖图。对于控制流依赖节点间的语句块通常是顺序执行的，MTVMiner 能轻松完成匹配，生成公共语句块。最后，MTVMiner 对公共控制依赖图和公共语句块进行整合，便可获取公共的函数控制流图。

表 3 语句类型分类

Table 3 Classification of statement type

| 语句分类 | 语句类型 |
|---|---|
| 顺序执行语句 | ExpressionStatement, VariableDecalaration, ThrowStatement |
| 循环语句 | ForInStatement, ForStatement, WhileStatement, DoWhileStatement, |
| 条件语句 | IfStatement, SwitchStatement |
| 跳转语句 | BreakStatement, ContinueStatement, ReturnStatement |
| 异常处理语句 | TryStatement |

依据上述方法，MTVMiner 能够获取模板小程序的公共函数控制流图及其延伸的公共调用边，通过对这些公共节点和控制流进行整合，便可构建小程序模板过程间控制流图。为减少后续重复的模板行为分析，MTVMiner 会将小程序模板过程间控制流图拆分为函数控制流图和过程间调用链,并将函数控制流图还原为抽象语法树的格式，分别存储在 func_asts.json 和 call_rels.json 两个文件中。若后续需要对某个小程序模板过程间控制流图进行安全分析，研究人员仅需将该小程序模板的 func_asts.json 和 call_rels.json 的路径作为参数传递给 MTVMiner 并设定构建参数，便可以直接生成小程序模板的过程间控制流图。

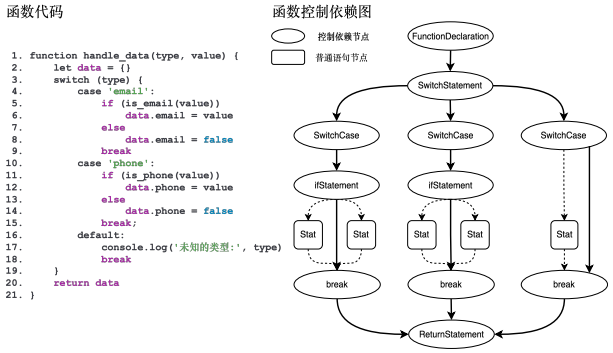


图 4 控制依赖图提取

Fig.4 Extraction of control flow dependency

### 3.2.3 数据流映射分析

尽管 MTVMiner 成功构建出小程序模板的过程间控制流图，但是部分非公共节点和控制流被舍弃，导致过程间控制流图部分数据流断裂或缺失，对后续安全分析的准确性造成影响。如图 5 所示，图中函数均取自图 3 的数据确认提交页面，原先 this.data.phone 的数据传播完整连贯。由于语句节点①作为非公共节点被删除，语句①中 this.data.phone 传递至 code_phone 的数据流断裂。此外，人工分析其他模板小程序中的语句②，可以发现 this.data.phone 被使用。因此，从数据流分析的角度来看，即使语句节点①作为非公共节点被删除，模板过程间控制流图也应将 this.data.phone 与语句②中的 code_phone 建立联系。

为此，本文引入数据流映射分析对小程序模板过程间控制流图构建过程中断裂的数据流进行补全。MTVMiner 利用 TaintMini[11]对原始的模板小程序过程间控制流图和提取的小程序模板过程间控制流图进行数据流建模，挖掘其中数据流断裂的地方，并将导致数据流断裂的语句节点进行提取。MTVMiner 利用数据流分析技术，确认缺失语句节点中变量的传播关系，如 code_phone 受 this.data.phone 的影响。之后，便在后续使用 code_phone 的语句中添加 taint 属性，用于记录语句中 code_phone 变量受 this.data.phone 的影响。通过上述方式，本文提取的小程序模板过程间控制流图在真实安全分析中的可用性得到了进一步提升。当然，该方法会受限于 TaintMini 工具数据流构建的精确性和完整度。

```
var sdk = require('../../../utils/sdk.js')
Page({
    data: {
        phone: '',
        email: ''
    },
    onLoad: function (options) {
        this.data.phone = options.phone
        this.data.email = options.email
    },
    confirm: function () {   非模板语句，删除
      ① let code_phone = addCountryCode(this.data.phone)
      ② sdk.post("***/submit", code_phone)
    },
    addCountryCode: function (raw_phone) {
        return "+86" + raw_phone
    }
})
```

图 5　数据流断裂示例

Fig. 5　Example of data flow break

### 3.3　模板漏洞挖掘

#### 3.3.1　固有漏洞扫描

在模板行为分析模块中，MTVMiner 能够将小程序模板过程间控制流图提取出来。因此，研究人员可以针对小程序模板控制流图进行安全分析，例如静态分析。

本文以小程序密钥泄漏检测为例，详细阐述如何将传统的安全分析方法与小程序模板过程间控制流图相结合。小程序密钥泄漏是指在小程序开发或运行过程中，敏感信息（如 API 密钥、加密密钥、访问令牌等）意外暴露或被恶意利用的情况。这种泄漏可能导致严重的安全问题，例如数据泄露、未经授权的访问、恶意攻击等。Zhang[16]等人研究发现，小程序中存在大量密钥硬编码泄漏的问题。本文对该安全漏洞进行梳理：硬编码密钥通常有固定的格式，并且周围存在相关的代码语义，但是仅根据格式和语义去检测会造成大量假阳性，因此需要分析可疑密钥字段的使用情况。本文首先利用正则表达式去扫描模板过程间控制流图中的可疑密钥字段。然后，本文对可疑密钥字段的进行数据流追踪，分析其具体的使用情况。如果可疑密钥字段被广泛用于小程序框架接口的使用，本文认为该模板存在密钥泄漏问题。

#### 3.3.2　模板小程序漏洞定位

小程序模板固有一旦漏洞被挖掘，其在控制流图上的触发逻辑便可以轻松提取。以密钥泄漏检测为例，MTVMiner 能够从模板过程间控制流图中提取硬编码密钥数据流情况。基于聚类分析生成的页面映射表，MTVMiner 在模板小程序中快速定位能够快速定位模板漏洞所涉及的页面信息。出于漏洞定位效率考虑，MTVMiner 仅需要对漏洞涉及的页面进行局部过程间控制流图构建即可，并能快速生成漏洞报告。

## 4　实验和评估

本文首先对 MTVMiner 多阶段聚类分析模块使用到的相似性阈值和聚类算法进行实验选定并对该模块的准确率进行评估，然后对 MTVMiner 的模板行为识别模块进行准确率评估，最后以硬编码密钥泄漏为例去挖掘小程序模板固有漏洞及其衍生的小程序漏洞，同时揭示模板开发导致的漏洞扩散问题具有一定的普遍性。

### 4.1　实验环境和数据集

本文实验的硬件和软件环境分别如下：硬件环境为 64 个 CPU 核心（Intel Xeon Gold 5218, 2.30GHz），内核版本为 5.15.0，物理内存大小为 256G 的 64 位服务器；软件环境为 Ubuntu 22.04.4 LTS 系统，Python 版本为 3.10.12，Node.js 版本为 16.20.2。

本文选取生态最为成熟的微信小程序作为实验对象。本文采用 MiniCrawler[9]来构建小程序初始数据集，并根据授权信息进行过滤，共计爬取 330359 个小程序，最终成功解包并被认定为潜在模板的小程序数量为 307784。之后，本文根据小程序的服务商信息，对小程序数据集进行预分类，统计共有 6362 个开发商。

### 4.2　多阶段聚类分析评估

#### 4.2.1　阈值确定和聚类算法选取

在聚类分析阶段，页面布局、函数摘要以及聚类算法的相似阈值均需要测定。以页面布局的相似阈值为例，本文随机抽取 10 个预分类数据集，随机抽样其中 100 个小程序，按照页面映射规则计算小程序页面间相似度。之后，本文从相似度 60%-100%，步长 5%进行进行抽样，每个数据段分别抽样 50 对页面进行人工分析，用于标记真正相似页面对的比例，即准确率。本文将人工分析的准确率与相似度阈值进行整理绘制，如图 4 所示，并以 90%准确率为界确定页面

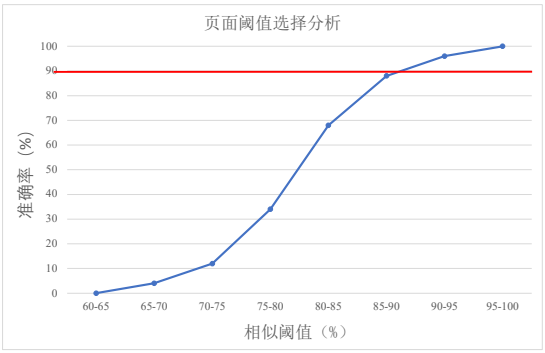


图 6 页面相似阈值选择分析图

Fig. 6 Analysis of page similarity threshold selection

布局相似度阈值，发现页面相似度阈值落在 85%-90%之间。于是，本文选取阈值中间值 88%作为页面布局相似度阈值。以相同的抽样方式，本文依次确认函数摘要相似阈值为 93%，基于页面布局、函数摘要和函数调用链计算小程序间相似阈值分别为 88%、93%和 93%。

此外，本文选取了谱聚类、层次聚类和密度聚类三种聚类算法对同一个预分类数据集进行聚类模拟，　三种聚类算法 F1-score 分别为 0.949、0.987、0.972。因此，本文最终选取层次聚类算法用于多阶段聚类分析。

#### 4.2.2　多阶段聚类分析有效性评估

在相关阈值确定后，本文对预分类数据集进行多阶段聚类分析。本文将簇中小程序超过 3 个的定义为有效簇，数量为 11468 个。本文从中随机抽取 5 个小程序簇来评估多阶段聚类分析的准确率，抽样结果如表 4 所示。

表 4 聚类分析模块准确率评估

Table 4 Effectiveness of cluster analysis

| 小程序数量 | | TP | FP | FN | 准确率 | 召回率 |
|---|---|---|---|---|---|---|
| 簇 | 预分类集 | | | | | |
| 8 | 11 | 5 | 0 | 3 | 100.00% | 62.50% |
| 10 | 11 | 10 | 0 | 1 | 100.00% | 90.91% |
| 20 | 20 | 17 | 3 | 0 | 85.00% | 100.00% |
| 32 | 37 | 31 | 0 | 6 | 97.43% | 83.78% |
| 81 | 81 | 81 | 0 | 0 | 100.00% | 100.00% |

从表 4 可知，多阶段聚类分析平均准确率为 97.96%，平均召回率为 93.51%。相较之下，多阶段聚类分析的召回率偏低，主要归因于阈值设定难以精确把控，但是这一不足并不影响后续的模板行为识别，因此可以接受。

### 4.3 模板行为分析评估

在多阶段聚类分析完成后，本文发现部分小程序簇规模过大，若将簇中所有模板小程序的过程间控制流图全部用于构建模板过程间控制流图，将面临时间和空间复杂度爆炸的问题。为此，本文选取了 5 个大小为 100 的小程序簇，并从中随机抽取 k 个小程序用于模板行为识别。实验结果表明，当 k 值超过 10 之后，识别出的节点和控制流几乎一致。因此，若小程序簇规模超过 10，MTVMiner 随机抽取 10 个模板小程序用于模板行为分析；若簇规模小于 10，则将簇中的所有小程序全部用于模板行为分析。

由于小程序模板缺乏源代码，本文仅能通过人工的方式来验证模板行为分析生成的过程间控制流图的准确性。为此，本文随机选取了 5 套小程序模板中 5 个函数控制流图进行人工分析。具体方法为：通过人工分析和观察的方式，重新提取并构建函数的控制流图，并与抽样的模板函数控制流图进行对比。若节点和控制流存在差异，则记为误报（FP）；若节点和控制流存在缺失，则记为漏报（FN）。相关结果如表 5 所示。

表 5 模板提取模块准确性评估结果

Table 5 Effectiveness results of template extraction

| 抽样控制流数量 | TP | FP | FN | 准确率 | 召回率 |
|---|---|---|---|---|---|
| 79 | 59 | 10 | 10 | 85.51% | 85.51% |
| 80 | 80 | 0 | 0 | 100.00% | 100.00% |
| 86 | 66 | 11 | 9 | 85.71% | 88.00% |
| 111 | 91 | 4 | 16 | 95.79% | 85.05% |
| 119 | 95 | 10 | 14 | 90.48% | 87.16% |

根据表 5 的数据，模板提取的平均准确率为 91.78%，平均召回率为 88.86%。本文进一步对模板提取中的漏报和误报进行了详细分析。误报的主要原因在于抽象语法树中未统计到部分语法结构，例如箭头函数表达式，导致涉及该表达式的语句均提取错误。而漏报则主要源于对 JavaScript 特殊实现方式的考虑不足，例如函数体中出现函数定义的情况，工具仅考虑了函数体中为非函数定义语句的情况，导致函数控制流图缺失。

### 4.4 密钥泄漏检测评估

本文以小程序硬编码密钥泄漏漏洞为例，对提取的 11468 套小程序模板过程间控制流图进行扫描，共发现 690 套模板存在密钥泄漏漏洞，占提取模板总量的 6.02%，涉及 21755 个小程序。

为确认小程序的密钥泄漏漏洞是模板开发导致的，本文随机抽样了 5 套模板中的 10 个小程序，对其进行了人工分析。结果显示，小程序密钥泄漏的触发逻辑均与模板密钥泄漏的触发逻辑一致。这表明本文提出的面向小程序模板的漏洞挖掘方法在能够应用于真实的漏洞挖掘，同时也揭示了模板开发所导致的漏洞扩散问题具有一定的普遍性，对小程序生态安全造成一定威胁。

## 5 结语

模板开发作为目前小程序开发的主流方式，但其自身的安全性难以保障。由于小程序模板代码的闭源特性，现有研究无法对其安全漏洞进行挖掘和系统性分析。为解决这一问题，本文提出了一套融合多阶段聚类分析和模板行为分析的方法，旨在挖掘小程序模板固有漏洞以及衍生的小程序漏洞。基于此方法，本文实现了一款漏洞挖掘工具 MTVMiner。MTVMiner 从相同模板开发的小程序过程间控制流图中识别相似的程序行为，并构建小程序模板过程间控制流图，准确率可达 91.23%。此外，本文对提取的小程序模板过程间控制流图进行了真实的硬编码密钥泄漏检测，共发现 690 套模板存在密钥泄漏，并进一步识别出超 2 万个小程序因模板开发也存在密钥泄漏风险，同时揭示了小程序模板开发导致的漏洞扩散具有一定的普遍性，对小程序生态安全造成一定威胁。

**References:**

[1] QuestMobile. 2024 Panoramic Traffic Report[EB/OL]. https://www.questmobile.com.cn/research/report/1815588709535420417

[2] Tecent. Wechat mini-program market[EB/OL]. https://fuwu.weixin.qq.com/

[3] Li S, Yang Z, Yang Y, et al. Identifying Cross-User Privacy Leakage in Mobile Mini-Apps at a Large Scale[J]. IEEE

Transactions on Information Forensics and Security, 2024.

[4] Medeiros I, Neves N F, Correia M. Automatic detection and correction of web application vulnerabilities using data mining to predict false positives[C]//Proceedings of the 23rd international conference on World wide web. 2014: 63-74.

[5] Li Z, Zou D, Xu S, et al. Vuldeelocator: a deep learning-based fine-grained vulnerability detector[J]. IEEE Transactions on Dependable and Secure Computing, 2021, 19(4): 2821-2837.

[6] Alrabaee S, Wang L, Debbabi M. BinGold: Towards robust binary analysis by extracting the semantics of binary code as semantic flow graphs (SFGs)[J]. Digital Investigation, 2016, 18: S11-S22.

[7] Karrer T, Krämer J P, Diehl J, et al. Stacksplorer: Call graph navigation helps increasing code maintenance efficiency[C]//Proceedings of the 24th annual ACM symposium on User interface software and technology. 2011: 217-224.

[8] Yang Z, Yang M, Zhang Y, et al. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection[C]//Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. 2013: 1043-1054.

[9] Zhang Y, Turkistani B, Yang A Y, et al. A measurement study of wechat mini-apps[J]. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2021, 5(2): 1-25.

[10] Khodayari S, Pellegrino G. {JAW}: Studying client-side {CSRF} with hybrid property graphs and declarative traversals[C]//30th usenix security symposium (usenix security 21). 2021: 2525-2542.

[11] Wang C, Ko R, Zhang Y, et al. Taintmini: Detecting flow of sensitive data in mini-programs with static taint analysis[C]//2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023: 932-944.

[12] Ye H, Yang Z M. Research on Call Graph Construction for Miniapp[J]. Journal of Chinese Computer Systems, 2024, 45(09):2228-2234

[13] Liu Y, Xie J, Yang J, et al. Industry practice of javascript dynamic analysis on wechat mini-programs[C]//Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. 2020: 1189-1193.

[14] Lu H, Xing L, Xiao Y, et al. Demystifying resource management risks in emerging mobile app-in-app ecosystems[C]//Proceedings of the 2020 ACM SIGSAC conference on computer and communications Security. 2020: 569-585.

[15] Yang Y, Zhang Y, Lin Z. Cross miniapp request forgery: Root causes, attacks, and vulnerability detection[C]//Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. 2022: 3079-3092.

[16] Zhang Y, Yang Y, Lin Z. Don't leak your keys: Understanding, measuring, and exploiting the appsecret leaks in mini-programs[C]//Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. 2023: 2411-2425.

[17] Zhang L, Zhang Z, Liu A, et al. Identity confusion in {WebView-based} mobile app-in-app ecosystems[C]//31st USENIX Security Symposium (USENIX Security 22). 2022: 1597-1613.

[18] Wu Y, Sun C, Zeng D, et al. {LibScan}: Towards more precise {Third-Party} library identification for android applications[C]//32nd USENIX Security Symposium (USENIX Security 23). 2023: 3385-3402.

[19] Li M, Wang W, Wang P, et al. Libd: Scalable and precise third-party library detection in android markets[C]//2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE, 2017: 335-346.

[20] Ohm M, Sykosch A, Meier M. Towards detection of software supply chain attacks by forensic artifacts[C]//Proceedings of the 15th international conference on availability, reliability and security. 2020: 1-6.

[21] Esprima. ECMAScript parsing infrastructure for multipurpose analysis[EB/OL]. https://esprima.org/, 2024.

[22] Stxy. Deriving control flow graphs from javascript programs[EB/OL]. https://github.com/mariusschulz/styx, 2021.

**附中文参考文献：**

[1] QuestMobile. 2024 年全景生态流量报告[EB/OL]. https://www.questmobile.com.cn/research/report/1815588709535420417

[2] 腾讯. 微信服务市场[EB/OL]. https://fuwu.weixin.qq.com/

[12] 叶瀚,杨哲慜.面向小程序的函数调用图构建方法[J].小型微型计算机系统,2024,45(09):2228-2234.