基于污点分析的二进制程序漏洞检测系统设计与实现*

罗治祥[1]，向　栖[2]，李乐言[1,3]
（1. 工业和信息化部电子第五研究所，广东　广州　511370；2. 东北大学　软件学院，辽宁　沈阳　110819；
3. 智能产品质量评价与可靠性保障技术工业和信息化部重点实验室，广东　广州　511370）

摘　要：针对现阶段二进制程序的静态分析多依赖于人工经验规则导致的低效率问题，以及大多数二进制程序漏洞扫描检测系统性能和可扩展性较低的问题，设计并且实现了一个简易版的基于污点分析的二进制程序漏洞检测系统。与现有的二进制程序漏洞检测系统相比，本文设计的系统改进了 Java 指针分析中提出的算法，使得分析过程的数据抽象部分和指针分析部分得到了分离，进一步简化了污点分析，提高了分析的性能和可扩展性。此外，将漏洞模式单独抽离出来，能够更加方便地进行漏洞模式的自定义。

关键词：二进制程序静态分析；指针分析；污点分析；漏洞模式
中图分类号：TP311　　**文献标识码：**A　　**DOI：**10.19358/j.issn.2097-1788.2023.11.001
引用格式：罗治祥，向栖，李乐言. 基于污点分析的二进制程序漏洞检测系统设计与实现［J］. 网络安全与数据治理，2023，42(11)：1-7.

Design and implementation of binary program vulnerability detection system based on taint analysis

Luo Zhixiang[1]，Xiang Qi[2]，Li Leyan[1,3]
（1. Fifth Research Institute of Ministry of Industry and Information Technology，Guangzhou 511370，China；
2. Software College，Northeastern University，Shenyang 110819，China；3. Key Laboratory of Intelligent Product Quality Evaluation and Reliability Assurance Technology，Ministry of Industry and Information Technology，Guangzhou 511370，China）

Abstract：Given the inefficiencies associated with the heavy reliance on manual heuristic rules for static analysis of binary programs at the current stage，and the low performance and scalability of existing binary program vulnerability scanning systems，this paper has designed and implemented a vulnerability detection system for binary programs based on taint analysis. Compared to the existing binary program vulnerability detection systems，the system designed in this paper improves the algorithm proposed in the Java pointer analysis，allowing the data abstraction and pointer analysis portions of the process to be separated. This further simplifies the taint analysis，enhancing its performance and scalability. Additionally，this paper extracts vulnerability patterns separately，making it much easier to customize vulnerability patterns.

Key words：static analysis of binary programs；pointer analysis；taint analysis；vulnerability patterns

0　引言

近年来，随着计算机技术的不断发展，软件行业迅速发展，软件的体量、种类不断增大。同时，由于程序代码编写不规范，编写过程存在疏忽，或者缺乏软件安全方面的意识等，导致软件的安全性承受巨大威胁。针对这一现状，构建漏洞检测系统对软件进行安全检测是最行之有效的办法之一，可在一定程度上发现并且及时消除潜在的漏洞。

软件的安全检测方法通常可以分为动态测试[1]和静态分析[2]两大类，静态分析这一方法相较于动态测试具有更高的覆盖率以及性能效率。现有的静态分析方案大多都基于源码级别[3]进行分析，虽然可以在比较高的程度上满足软件安全需要，但是在许多真实的安全测试场景中，需要分析对象大多属于常见的二进制文件，比如商业软件、车机固件、嵌入式系统固件等。此时安全研究人员难以获得相应的源代码，源码级静态分析方案不

*基金项目：国家重点研发计划 NQI 专项（2022YFF0607100）

再适用。现今商业化的二进制程序漏洞分析系统基本不具备可二次开发的扩展性，而开源的二进制程序漏洞分析系统，比如 angr[4]、BAP[5]、BinAbsInspector 等优秀的静态分析工具存在一些适用性问题。其中，angr 和 BAP 正逐步发展为通用分析框架，而不仅仅专注于二进制漏洞扫描，这使得内部分析算法变得复杂，不便于后续的扩展和优化。BinAbsInspector 是基于 Ghidra 的插件，使用 Java 编写开发，提供的 API 比较有限，在可扩展性上有所欠缺，普通的二进制安全研究员无法有效地基于该工具进行自定义开发。因此，构建一款在性能和可扩展性上较为突出，同时也满足于真实场景需要的二进制程序漏洞检测系统刻不容缓。

针对上述问题，Tan[6]等基于 Java 提出了一种较为简易的指针分析[7]算法，该算法在使用 Datalog[8]引擎支撑之后，在构建好的程序抽象数据上进行分析，使得性能得到较大的提升。对于获取二进制程序抽象数据而言，普遍的做法是反编译[9]二进制程序生成 IR 中间表示，进一步对其进行优化和转换得到其 AST 语法树，最后依据 AST 语法树得到程序抽象数据，比较好的工具有 IDA Pro、Ghidra、Binary Ninja 等。本文设计系统选取了 IDA Pro 进行程序抽象数据获取，使用 IDA Pro 特有的 IDAPython，从原生角度适配了 IDA Pro 提供各项 API 接口，从可扩展性上兼具了 Python 脚本跨平台、易扩展等优点；从性能方面，Python 库中的 pyDatalog 可以替代常见的 Datalog 引擎，且性能方面相差无几。

1　方案设计

1.1　本文研究工作

本文将以污点分析为基础，对 IDA Pro 中插件 Hex-Rays 反编译时的语法树 AST 中相关的变量赋值、函数调用等进行抽象解析，使之成为特定的数据结构。随后使用 Datalog 对抽象之后的数据进行指针分析，在此基础上加入污点分析，通过 Datalog 得到相关指针结果，依据用户提供的污点 source 和定义的 sink，结合缓冲区溢出[10]、格式化字符串[11]、堆内存使用不当等漏洞的特征，进一步检测出二进制程序中的漏洞。由于依赖的底层数据是基于 Hex-Rays 反编译出的语法树，因此是否支持不同架构的二进制程序只会取决于 Hex-Rays 是否能够对该架构程序进行反编译。此外由于使用 Datalog 进行污点分析，其效率和准确度也会得到相应提高。综上所述，本论文基于污点分析，结合 Hex-Rays 反编译的语法树以及 Datalog 对二进制程序进行漏洞检测，更加具有应用价值和研究意义。

1.2　系统架构

依据上述的程序语言以及逻辑构思，本系统构建的整体数据流程业务如图 1 所示。

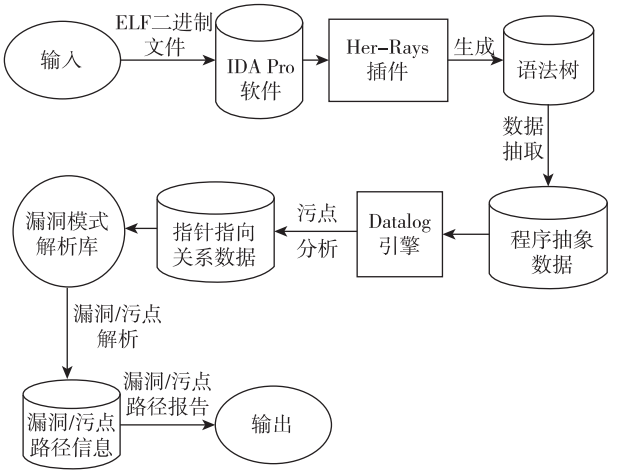


图 1　系统架构图

本系统采用 C/S 架构，并且以 MVC 设计模式进行设计，设计的系统架构如图 2 所示，由上至下系统主要分为用户交互层、数据处理层、指针/污点分析层、业务处理层四个主要层次。

在用户交互层中，依据 IDA Pro 提供的各种接口，利用 IDAPython 结合 PyQt 进行界面开发，旨在提供一个对用户友好的界面处理。该视图控制层中的主要功能有漏洞模式配置，用来选取漏洞分析模式以及管理污点数据库。还有审阅界面，当分析完成时，可以在此界面对漏洞报告进行查看、导出，以及污点路径的相关审阅工作。

在数据处理层中，将对程序进行一些预处理工作，获取程序对应的语法树方便后续进行分析。然后还需要对程序代码进行数据抽象处理，分为 New/Assign 等多种数据，将程序解析成能够使得数据分析引擎进行分析的各项数据。最后需要对这些程序代码抽象出来的数据进行存储。

指针/污点分析层是主要的漏洞分析模块，获取上述得到的程序代码抽象数据，然后依据编写的 Datalog 程序对程序抽象后的各类数据进行指针/污点分析，最终输出相关的分析结果。

业务处理层主要依据指针/污点分析层得到的相关分析结果，建立相关的漏洞模式，依据漏洞模式的相关规则，尝试检测出漏洞，最后导出相关漏洞报告。针对污点数据 Sources/Sinks 处理，该层提供显示污点路径以及污点路径表格化展示功能。

1.3　算法整体设计

1.3.1　功能模块设计

本系统的功能模块设计如图 3 所示，主要包括程序输入模块、程序预处理模块、污点分析模块、漏洞检测模块四个部分。

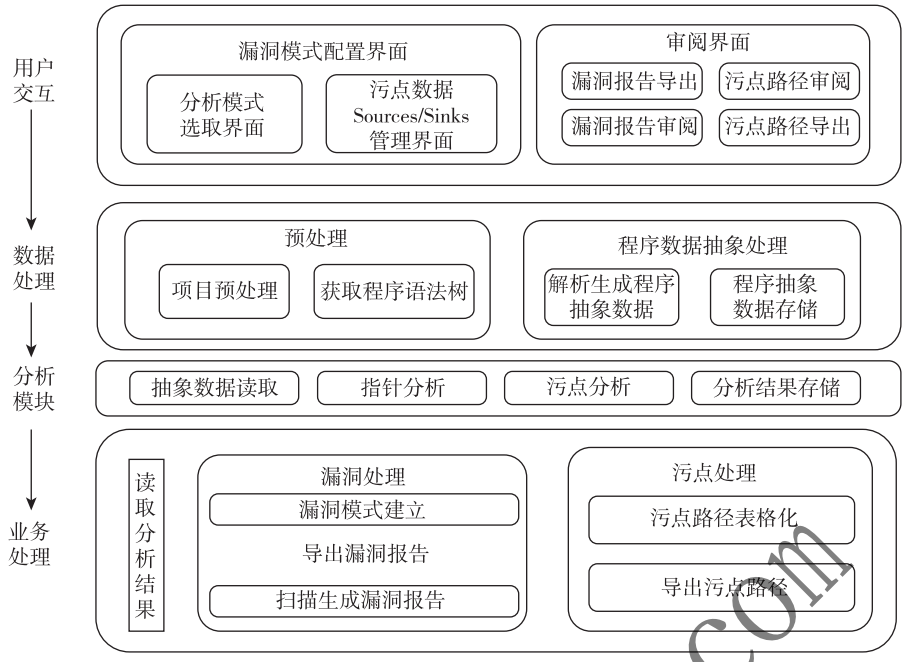


图 2　系统架构图

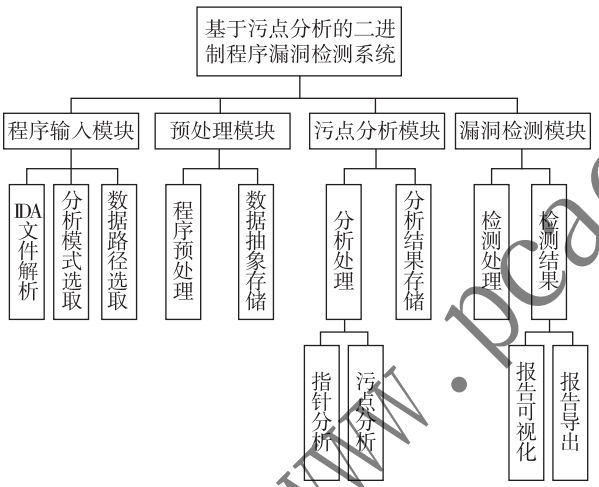


图 3　功能模块设计图

1.3.2　详细设计

（1）程序输入模块

在该模块中，主要的功能为在 IDA Pro 中导入二进制程序文件，选取漏洞分析模式，配置污点数据库 Sources/Sinks。其中漏洞分析模式主要有如下三种：

① 漏洞分析模式

该模式将会自动分析当前二进制程序文件，依据指针分析结果和相关的漏洞模式来寻找漏洞，并且提供漏洞分析报告析。

② 仅 Source 分析

用户可以在污点数据 Sources/Sinks 的管理界面使用下拉框选取污点数据。比如选取 main 函数中变量 v1 作为 Source，均以数据下列框、按钮等形式来选取。随后即可进行污点分析，依据用户给出的污点数据集合和指针分析结果，分析出污点数据可能的流向，并且提供相关的数据流报告。

③ Source/Sinks 分析

用户创建了对应的污点数据集合后，还可以选取用户特定感兴趣的函数放入 Sinks 集合中，比如选取 main 函数中的 read 函数作为 Sink 等。随后开始进行污点分析，判断在 Sinks 集合是否被污点数据污染，并且提供相关的污染报告。

（2）预处理模块

该模块的主要功能为获取程序语法树、程序数据抽象、抽象数据存储。基于 Tan[6] 等提出指针分析算法改进的 Datalog 抽象数据规则算法如算法 1 所示。

算法 1：Datalog 抽象数据规则算法

```
1   New (x: V, o: O, m: M, c_ x: C)
2   Assign (x: V, y: V, c_ x: C, c_ y: C)
3   Store (x: V, f: F, y: V, c_ y: C)
4   Load (y: V, x: V, f: F, c_ y: C, c_ x: C)
5   Argument (l: S, i: N, ai: V, thisM: M)
6   Parameter (m: M, i: N, pi: V)
7   MethodReturn (m: M, ret: V)
8   CallReturn (l: S, r: V, thisM: M)
9   Methods (l: C, mName: M)
10  VCall (l: S, x: V, k: M, thisM: M)
```

由于需要确定各个函数中对应的变量所指向对象，因此加入了 thisM 属性来进行区分。

依据上述的数据抽象规则，使用 IDAPython 与 IDA Pro 提供的 API 进行交互，遍历语法树中所有可达函数，将对应的变量声明、存储、赋值、上下文等信息进行抽象，以 csv 格式保存在文件中。

（3）污点分析模块

污点分析模块中主要包含抽象数据存储、指针分析、污点分析、指针分析结果存储几个功能。从文件中获取到存储的抽象数据之后，就可以按如下步骤进行分析。

① 指针分析

基于 Tan[6] 提出指针分析算法改进的 Datalog 指针分析规则算法如算法 2 所示。

算法 2：Datalog 指针分析规则算法

1 VarPointsTo（v:V,o:obj,c_ var:C,c_ obj:C,m:method).

2 FieldPointsTo（Oi:obj,f:field,Oj:obj,c_ Oi:C,c_ Oj:C).

3 Reachable（m:method).

4 CallGraph（l:stmt,m:method,thisM:method).

5 Reachable（m),CallGraph（l,m,thisM）< - VCall（l,m,thisM),Reachable（thisM).

6 VarPointsTo（x,o,c_ xo,c_ xo,m）< - Reachable（m),New（x,o,m,c_ xo).

7 VarPointsTo（x,o,c_ x,c_ o,m）< - Assign（x,y,c_ x,c_ y), VarPointsTo（y,o,c_ y,c_ o,m).

8 VarPointsTo（y,Oj,c_ y,c_ Oj,m）< - Load（y,x,f,c_ y,c_ x), VarPointsTo（x,Oi,c_ x,c_ Oi,m), FieldPointsTo（Oi,f,Oj,c_ Oi,c_ Oj).

9 FieldPointsTo（Oi,f,Oj,c_ Oi,c_ Oj）< - Store（x,f,y,c_ y), VarPointsTo（x,Oi,c_ x,c_ Oi,m), VarPointsTo（y,Oj,c_ y,c_ Oj,m).

10 VCall（l:stmt,m:method,thisM:method)

11 Argument（l:stmt,i:number,ai:V,thisM:method).

12 Parameter（m:method,i:number,pi:V).

13 Methods（label:C,mName:method).

14 VarPointsTo（pi,o,label,c_ o,m）< - Methods（label,m), CallGraph（l,m,thisM), Argument（l,i,ai,thisM), Parameter（m,i,pi), VarPointsTo（ai,o,c_ ai,c_ o,thisM).

15 CallReturn（l:stmt,r:V,thisM:method).

16 MethodReturn（m:method,ret:V).

17 VarPointsTo（r,o,c_ ret,c_ o,thisM）< - CallGraph（l, m, thisM), MethodReturn（m, ret), VarPointsTo（ret,o,c_ ret,c_ o,m), CallReturn（l,r,thisM).

将不再需要 Tan[6] 等提出的指针分析算法中的 Dispatch/ThisVar 变量，因为在反编译得到的语法树中，其函数名是唯一标识的，不存在相同的函数名称。

由于 Hex-Rays 反编译得到的语法树较为复杂，所提供的 API 种类繁多，实际进行数据抽象的流程较为复杂，因此图 4 展示的仅为数据抽象时的流程简图。

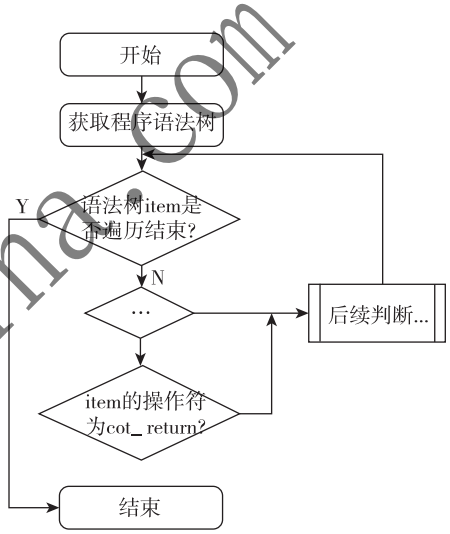


图 4 程序数据抽象流程简图

② 污点分析

同样也是基于 Tan[6] 等提出指针分析算法改进的 Datalog 污点分析规则，对应有数据污染源 Source（m：M），程序代码关键关注点 Sink（m：M，i：N），污染数据点 Taint（l：S，t：T）。即把所有的污点数据产生点上下文`l：S′关联到产生的污点数据 tainted data，即 t：T 中。传递污点数据算法如算法 3 所示。

算法 3：Datalog 污点传递规则算法

1 VarPointsTo（r,t）< - CallGraph（l,m),Source（m),CallReturn（l,r),Taint（l,r).

2 TaintFlow（j,l,i）< - CallGraph（l,m),Sink（m,i),Argument（l,i,ai),VarPointsTo（ai,t),Taint（j,t).

TaintFlow 代表 j 点产生的污点数据在流向 l 点调用的函数，其参数位置为 i。但是由于 JAVA 中一切即对象的思想，因此所有的函数过程传递都会用上污点传递，而在 C 中并不是很适合。所以，当指针分析结果中相关对象是被污染的对象，就将其加入到污点流动集合中。进而设计定义 TaintVar（m：M，v：V），TaintVar 依据用户选定的污点数据集合进行构造，代表用户选中函数 m 中的 v 变量作为污点 Source。随后依据污点数据集合，将对应变量指向的对象进行污染，方便后续进行指针分析时传递污点对象，具体设计如算法 4 所示。

算法 4：Datalog 污点对象流动算法

```
1  TaintObj（o:obj,c_ obj:context,m:method）
2  TaintObj（o,c_ xo,m） <- Reachable（m）,
New（x,o,m,c_ xo）,TaintVar（m,x）.
3  TaintTranslate（v:var,c_ v:context,o:obj,m:
method）
4  TaintTranslate（x,c_ x,o,m） <- VarPointsTo
（x,o,c_ x,c_ xo,m）,TaintObj（o,c_ xo,m）.
```

通过 TaintObj 将对象进行污染，当指针分析结果中存在污点对象，则标注出对应指向污点对象的变量。在对应的 IDAPython 脚本中获取到 Sinks 集合，判断集合中调用的函数参数是否包含指向污点对象的变量即可。最后完成分析结果存储工作，方便进行下一步的漏洞分析或者污点分析报告。

（4）漏洞检测模块

针对漏洞分析模块通常需要设定一些常见漏洞模式，比如从 IDA Pro 提供的语法树中获取 read 函数列表，判断调用 read 函数读取字符时读取数量是否大于当前缓冲区的大小，或者 strcpy 中的源字符串长度是否大于目标字符串长度等。

一旦缓冲区溢出、格式化字符串漏洞、堆内存使用不当等漏洞的模式规则被满足，即生成对应格式的报告，以 csv 的格式展示，设计如表 1 所示。

在污点路径显示功能设计中，依据污点分析结果，以特定的格式生成对应的污点路径表格，以便后续给用户展示，设计如表 2 所示。

如表 2 所示，能看出选中的污点对象 Obj2 在 main 函数中流经了 b、c 两个变量，更加便于用户着重分析污点数据的相关变更。最后将相关报告导出显示完成所有的污点/漏洞分析。

表 1　漏洞报告

漏洞名称	漏洞类型	漏洞产生点	漏洞产生关键变量	关键变量指向的对象
栈溢出	CWE787	10	main. v1	Obj1
二次释放	CWE415	15	main. v2	Obj2

表 2　污点路径报告

流经污点变量名称	流经污点变量产生/赋值地址	污点对象	流经污点变量所在函数
b	0x40023	Obj2	main
c	0x40034	Obj2	main

2　系统测试与分析

2.1　测试环境

本文进行系统测试的测试环境如表 3 所示。

表 3　系统测试环境

环境	项目	配置及版本信息
硬件环境	CPU	AMD Ryzen 7 4800H with Radeon Graphics @ 2.90 GHz
	内存	48 GB DDR4
	硬盘	1 TB
软件环境	操作系统	Windows Server 2022 Standard Evaluation
	反编译平台	Hex-Rays v7.6.0.210427
	静态分析工具	IDA v7.6_ SP1

2.2　功能测试

本节将使用二进制程序漏洞检测系统针对常见漏洞进行功能测试。在 ubuntu20.04_x64 平台使用 gcc_v9.4.0 编译代码，使用命令为 gcc ./test.c -o stackOverflow，得到程序 stackOverflow，放入 IDA Pro 进行分析。

分析过程中在 IDA Pro 的输出窗口会输出过程数据，如图 5 所示。点击查看报告，如图 6 所示。

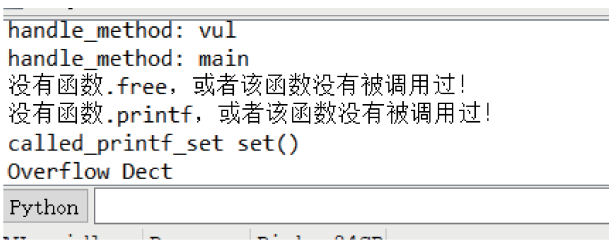


图 5　分析过程输出图


污点路径报告

	流经污点变量名称	流经污点变量产生/赋值地址	污点对象	流经污点变量所在函数
1	a1	4457	v4-localObj	vul
2	v4	4503	v4-localObj	main

图 6　栈溢出污点报告图

其中 v4-localObj 污点对象流经了 main. v4 和 vul. a1，符合程序的逻辑。切换分析模式，漏洞分析结果如图 7 所示。


漏洞报告

	漏洞名称	漏洞类型	漏洞产生点	漏洞产生关键变量	关键变量指向的对象
1	溢出	CWE787	4495	vul.a1	v4-localObj

图 7　栈溢出漏洞报告图

此外针对格式化字符串、二次释放漏洞也进行了相关的测试，得到结果和栈溢出漏洞测试类似。

综上所述，本节涉及的污点路径探索测试、漏洞检测测试，从整体上来说大致符合预期，能够检测出较为简单的漏洞以及污点路径。

2.3　性能测试

本文选取 Juliet 测试集，就 x86、x64 两种架构与 cwe_checker 的测试结构进行比较对照。此外，由于本文实现的系统是漏洞检测原型系统，在漏洞模式检测上仅实现 CWE-787、CWE-415、CWE-134 三种 CWE 中指定函数造成的漏洞检测，因此，本文从 Juliet 中随机筛选了 CWE-134 中 125 个关于 printf 的格式化字符串漏洞、CWE-415 中 151 个关于 malloc-free 的二次释放漏洞，CWE-787 中 135 个关于 read 函数的溢出写漏洞测试用例进行测试。表 4 和表 5 展示的是 x86 和 x64 架构下的三种 CWE 的检测能力与 cwe_checker 的对比结果，其中缩写英文对应为 VDT。本文实现的原型系统为 vulDector，统计得到如图 8 所示漏报率误报率检测对比统计图。结果表明，本文设计的原型系统的性能测试结果在目前所支持的 CWE 类型上取得一定优势。

表 4　x86 架构 CWE 检测能力对比

CWE 类型/工具名称	数量	TP	FP	TN	FN
CWE415/VDT	151	70	3	68	10
CWE415/CC	151	24	6	62	59
CWE787/VDT	135	67	8	53	7
CWE787/CC	135	37	4	61	33
CWE134/VDT	125	65	4	50	6
CWE134/CC	125	53	12	34	26

表 5　x64 架构 CWE 检测能力对比

CWE 类型/工具名称	数量	TP	FP	TN	FN
CWE415/VDT	151	69	4	66	12
CWE415/CC	151	23	2	70	56
CWE787/VDT	135	70	6	57	2
CWE787/CC	135	40	3	63	29
CWE134/VDT	125	63	6	46	10
CWE134/CC	125	52	13	32	28

3　结论

本文主要研究了基于污点分析的二进制程序漏洞检测系统设计与实现，尝试通过基于 Hex-Rays 反编译的语法树来进行指针分析，寻找漏洞以及实现污点路径探索功能。

本文根据对目前二进制程序漏洞检测系统的研究现状的参考，首先确定了本系统的总体架构、数据业务流程，根据系统分析设计，对基于污点分析的二进制程序漏洞检测系统进行实现。在实现过程中，采用 IDAPython 语言进行程序编写，使用了 Hex-Rays 提供的相关接口，以及用于指针/污点分析的 Datalog 引擎 pyDatalog。在系统实现之后，依据系统测试相关原理，使用简单的漏洞代码编写的二进制程序对系统的各个功能模块进行了测试。测试过程中，本系统实现的各个功能模块能够稳定运行并投入使用，检测出较为简单的栈溢出、格式化字符串、二次释放等漏洞。

本文具有以下创新点：

（1）在 Hex-Rays 反编译结果的基础上进行指针/污点分析的相关研究还不是很多，在此基础上实现完整可用的基于污点分析的二进制程序漏洞检测系统。

（2）不同于使用 Datalog 进行 C 源代码形式的指针/污点分析，本文使用 Datalog 对二进制源程序的语法树形式进行指针/污点分析研究。

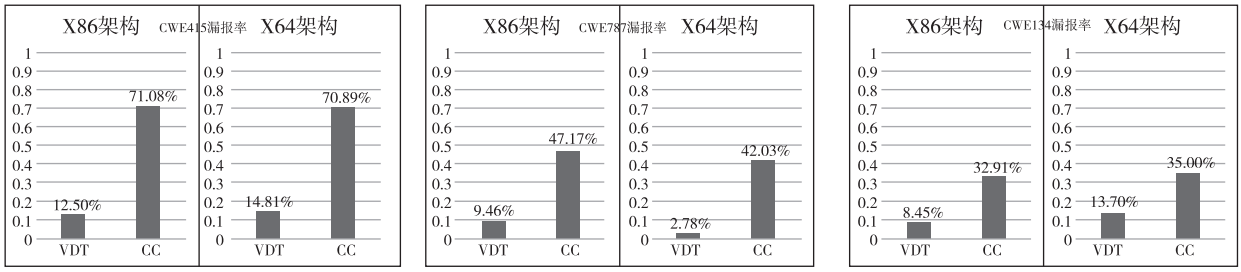
（3）本系统实现的编程语言全部为 Python 脚本，具有高度的可扩展性、兼容性、可移植性等。

系统实用性方面仍旧存在的不足：

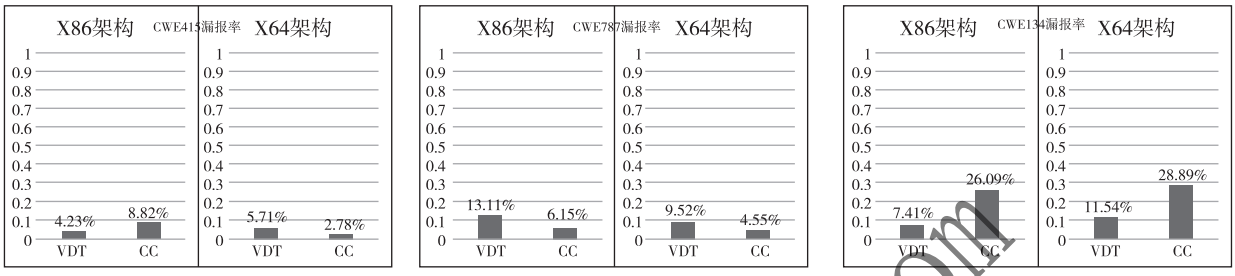
（1）本系统的测试用例都是在 Linux 平台下使用 GCC 进行编译生成的，对于其他平台是否适用还有待进一步检验。

（2）由于本系统只是依据程序中的一些数据结构以及函数调用等关系进行数据抽象，因此具备诸如循环、递归等复杂逻辑时的适用性有待检验。

（3）本系统暂时只实现了栈溢出漏洞、格式化字符串漏洞、二次释放漏洞模式建立，适用范围较小。后续将对基于污点分析的二进制程序漏洞检测系统进行完善。



(a) 漏报率



(b) 误报率

图 8　ds x86/x64 架构下的漏洞检测漏报率/误报率对比图

参考文献

[1] MILLER B P, FREDRIKSEN L, SD B. An empirical study of the reliability of UNIX utilities [J]. Communications of the ACM, 1990, 33 (12): 32 – 44.

[2] CHESS B, WEST J. Secure programming with static analysis [M]. Addison-Wesley Professional, 2007.

[3] ENGLER D, CHEN D Y, HALLEM S, et al. Bugs as deviant behavior: a general approach to inferring errors in systems code [C]// Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, 2001: 57 – 72.

[4] SHOSHITAISHVILI Y, WANG R, SALLS C, et al. SoK: (State of) the art of war: offensive techniques in binary analysis [C]//2016 IEEE Symposium on Security and Privacy (SP), 2016: 138 – 157.

[5] BRUMLEY D, CABALLERO J, LIANG Z, et al. Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation [C]//Proceedings of the 16th Conference on USENIX Security Symposium, 2007.

[6] 谭添, 马晓星, 许畅, 等. Java 指针分析综述 [J]. 计算机研究与发展, 2023, 60 (2): 274 – 293.

[7] BURKE M, CARINI P R, CHOI J D, et al. Flow-insensitive interprocedural alias analysis in the presence of pointers [J]. Again, Proceedings of the International Workshop on Languages and Compilers for Parallel Computing, 1994.

[8] CERI S, GOTTLOB G, TANCA L. What you always wanted to know about Datalog (and never dared to ask) [J]. IEEE Transactions on Knowledge and Data Engineering, 1989, 1 (1): 146 – 166.

[9] CIFUENTES C. Reverse compilation techniques [D]. Brishane: Queensland University of Technology, 1994.

[10] ALEPH ONE. Smashing the stack for fun and profit [J]. Phrack, 1996, 7 (49): 14.

[11] NEWSHAM T. Format string attacks [R]. Guardent, Inc, 2000.

(收稿日期: 2023 – 10 – 10)

作者简介:

罗治祥 (1998 –), 男, 本科, 主要研究方向: 漏洞挖掘、网络安全、车联网信息安全。

向栖 (2003 –), 女, 本科, 主要研究方向: 网络安全、信息安全。

李乐言 (1993 –), 通信作者, 男, 本科, 工程师, 主要研究方向: 信息安全攻防技术和质量评价技术、智能产品及汽车信息安全。

版权声明

凡《网络安全与数据治理》录用的文章，如作者没有关于汇编权、翻译权、印刷权及电子版的复制权、信息网络传播权与发行权等版权的特殊声明，即视作该文章署名作者同意将该文章的汇编权、翻译权、印刷权及电子版的复制权、信息网络传播权与发行权授予本刊，本刊有权授权本刊合作数据库、合作媒体等合作伙伴使用。同时，本刊支付的稿酬已包含上述使用的费用，特此声明。

《网络安全与数据治理》编辑部