

基于函数语义分析的软件补丁比对技术

曹琰¹, 刘龙¹, 王禹², 王清贤¹

(1. 数学工程与先进计算国家重点实验室, 河南 郑州 450000;

2. 河南工程学院, 河南 郑州 450000)

摘要: 基于结构化的补丁比对是软件漏洞辅助分析的重要方法。在分析总结已有补丁比对技术及反补丁比对技术的基础上, 针对结构化比对存在无法进行语义分析而导致误报的问题, 提出了基于函数语义分析的软件补丁比对方法。利用传统的结构化比对方法, 在函数级进行语法差异比较得到最大同构子图; 通过程序依赖分析, 构建函数输入输出之间的路径包络, 基于符号执行以包络为对象计算函数输出特征; 通过函数摘要进行语义级比对, 结合最大同构子图的匹配函数结果, 进一步分析得出发生语义变化的函数。最终, 通过实验比对测试, 验证了所提方法的可行性和优势。

关键词: 漏洞分析; 补丁比对; 符号执行; 语义分析

中图分类号: TP393

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2019051

Software patch comparison technology through semantic analysis on function

CAO Yan¹, LIU Long¹, WANG Yu², WANG Qingxian¹

1. State Key Laboratory of Mathematical Engineering & Advanced Computing, Zhengzhou 450000, China

2. Henan University of Engineering, Zhengzhou 450000, China

Abstract: Patch comparison provides support for software vulnerability, and structural comparison has been developed. Based on summarizing binary files comparison and anti-comparison methods, comparison technology was proposed by semantic analysis on function to address the issue that structural comparison cannot carry on semantic analysis. Through traditional structural comparison, syntax differences in function-level were analyzed to find the maximum common subgraph. Then, the path cluster was built between the input and output of the function depend on program dependency analysis. Function output characteristics was established based on symbolic execution. Semantic differences of functions were compared by functional summaries. Based on the maximum isomorphic

收稿日期: 2018-09-27; 修回日期: 2019-01-14

通信作者: 曹琰, vspyang@hotmail.com

基金项目: 国家重点研发计划基金资助项目(No. 2017YFB0803202, No. 2016QY07X1404)

Foundation Item: The National Key Plan R&D Program of China (No. 2017YFB0803202, No. 2016QY07X1404)

论文引用格式: 曹琰, 刘龙, 王禹, 等. 基于函数语义分析的软件补丁比对技术[J]. 网络与信息安全学报, 2019, 5(5): 56-63.

CAO Y, LIU L, WANG Y, et al. Software patch comparison technology through semantic analysis on function[J]. Chinese Journal of Network and Information Security, 2019, 5(5): 56-63.

subgraph, the matched functions which there are possible semantic changes between was further analyzed. Ultimately, the experimental results showed the feasibility and advantages of the proposed method.

Key words: vulnerability analysis, patch comparison, symbolic execution, semantic analysis

1 引言

软件厂商为了维护信息产品安全性, 针对已发现存在的软件漏洞, 开发相应的修补代码, 称为“安全补丁 (security patch)”。使用补丁修补漏洞, 即用安全的程序代码替换原有不安全的程序片段, 使同一文件因为补丁发布在不同版本之间存在差异。补丁比对技术就是通过比较可执行文件的差异, 揭示漏洞的确切位置和成因, 而这些信息一般在安全公告中未指明。通过对补丁比对, 进一步分析漏洞的成因和触发机理, 有助于软件开发过程中规避已出现的漏洞模式。

补丁比对技术的发展已有近 20 年的历史。1999 年的 BMAT^[1]是补丁比对技术的开端, 研究在两个可执行文件中函数匹配的问题, 该方法严重依赖于符号文件。

指令级别的同构比较受编译器优化和函数匹配的影响较大, 需要进一步改进, 而利用调用图及控制流图进行比较的结构化比较方法可以很好地解决匹配问题, 并且在一定程度上解决编译器优化所带来的问题。2004 年, Flake^[2]提出了基于结构化比较的二进制文件比较方法。同时, Flake 还提出了使用函数的指纹 (fingerprints) 进行比较, 开发了著名的 Bindiff。

结构化的比较方法可以一定程度上避免因优化而使指令变化所带来的影响而且跨平台。结构化比较的基本思想影响深远, 依然是当前补丁比对的主流方法, 已在 Sabre Security Bindiff、PatchDiff2、TurboDiff 等工具中使用, 得到了较好的效果。潘璠等^[3]通过改进固定点传播算法, 避免错误匹配的传播, 且对错误匹配进行修正, 优化了结构化比对结果。肖雅娟^[4]提出了基于函数

内部和外部紧宽约束特征进行函数相似性比较。

为了应对图形匹配效率问题, 研究人员考虑将函数代码特征转化为数字化编码。Liu 等^[5]提出了基于 DNN 的二进制代码相似性检测方法, 将函数代码特征编码为 64 bit 向量, 以数字向量比较计算二进制函数相似度。Xu 等^[6]提出了基于网络嵌入的二进制函数数字化向量方法。

研究人员也将结构化比对技术应用于指导模糊测试和 Concolic (Concrete & Symbolic 的合成) 测试^[7-8]; 基于补丁比对技术对修补方式进行识别, 还可以开展针对未打补丁原文件的利用代码自动生成技术的研究^[9-11]。

为了防止被攻击者利用, 出现了各类对抗二进制比对的混淆、欺骗技术^[12-13], 如改变符号、扰乱指令顺序、扰乱 CFG、使用 proxy 调用、调用不返回、共享基本块等方法。这些方法的基本思想是针对结构化匹配的方法, 混淆结构化签名的相关信息, 使之比较结果出现差错。

如果混淆技术使用于软件补丁代码的编写过程中, 则基于结构化比对的准确率会降低。为了实现更为精准的分析, 在一定程度上抵抗代码混淆, 使结构化差异分析更加精确, 相关研究人员提出了在结构化比对结构基础上, 分别利用基本块语义和 trace 语义信息进一步加强比对结构的精确度^[14-16]。

基于结构化或静态特征的比对本质是语法分析。如果要防止混淆技术带来的不利影响, 就不仅能在代码结构特征上比较, 更要比较文件间的语义功能变化情况, 这就需要进行语义级的比对。已有的语义比对方法主要在基本块级实现, 用于修正函数比对结果。但基本块粒度对漏洞信息包含不全, 只能局部表现某些语义差异。函数级的

分析能够更加精准反映漏洞触发语义。本文设计实现了面向二进制函数级的语义差异分析技术, 基于函数符号执行的输入输出差异, 判断函数语义; 通过路径包络和函数摘要分析提高函数级符号执行的效率。

2 问题的提出

通过比对原文件 (unpatch) 文件和补丁文件 (patched) 文件之间代码结构的差异, 掌握修补漏洞的方式。

如图 1 所示, 把可执行文件看作函数的集合, 令 $unpatch = M_U \cup N_U$, $patched = M_P \cup N_P$, 其中 N_U 、 N_P 分别表示原文件和补丁文件中不匹配的函数集合, M_U 、 M_P 分别表示原文件和补丁文件中匹配的函数集合, $M_U = M_{UC} \cup M_{UN}$, $M_P = M_{PC} \cup M_{PN}$, 其中, M_{UC} 和 M_{UN} 分别表示原文件中匹配且发生改变的函数和匹配且无任何变化的函数; M_{PC} 和 M_{PN} 分别表示补丁文件中匹配且发生改变的函数和匹配且无任何变化的函数。 M_{UN} 和 M_{PN} 中的函数本质上是同一函数, 即补丁前后没发生变化。

在进行比较分析后, 重点关注的是不匹配部分和发生改变的匹配部分的集合 $K = N_U \cup N_P \cup M_{UC} \cup M_{PC}$, 进一步分析, K 是漏洞或补丁最有可能出现的位置。

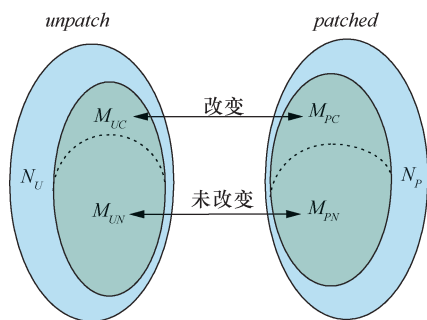


图 1 补丁比较差异示意

本文提出的基于函数语义分析的软件补丁比对方法是符号执行技术与图同构匹配的结合, 其主要步骤如下。

步骤 1 对函数调用图、控制流图进行同构比较。找出两个调用图之间的最大同构子图, 目的是找出漏洞或补丁存在的可疑函数集合 $K = N_U \cup N_P \cup M_C$, 其中 $M_C = M_{UC} \cup M_{PC}$ 。

步骤 2 对集合 M_C 中的函数对进行语义比较。进一步排除结构特征发生变化而语义无变化的函数对, 重点关注真正语义发生变化的函数对, 以更加精准定位漏洞所在函数。

3 基于图形同构的函数级结构化比较

基于结构特征签名信息的比较是目前主流的同构匹配方法。选用能够表达结构化特征的属性集合作为函数签名, 可选的函数签名属性包括函数名、节点的入度/出度、递归属性、循环体 (while 循环、for 循环等)、相同字符串引用、可归约结构的素数乘积、参数个数、标准应用程序编程接口 (API, application programming interface) 函数调用、栈布局、远程过程调用 (RPC, remote procedure call) 函数等。函数签名间的欧几里得距离用于表示相似程度。

根据上述的设计签名, 对于原文件中的函数 A 和补丁文件中的函数 B, 如果 A 和 B 匹配, 需要同时满足以下 3 个条件。

- 1) A 和 B 签名的欧几里得距离满足阈值。
- 2) 补丁文件中除了 B 没有其他的函数与 A 签名的欧几里得距离满足阈值。
- 3) 原文件中除了 A 没有其他的函数与 B 签名的欧几里得距离满足阈值。

在进行了函数同构比较后, 找到了最大同构子图。接下来, 在最大同构子图的基础上, 对已匹配函数未发生结构变化的函数对 M_C 进行语义比较, 以判断发生了语义变化的函数对。

4 基于符号执行的函数级语义比较

函数级语义比较的基本思路是: 基于程序依

赖图 (PDG, program dependence graph) 分析函数输入、输出之间的依赖关系, 构建路径包络; 基于符号执行计算函数输出的符号值, 每个路径包络对应一个输出的符号值; 根据函数输入、路径包络和输出关系建立函数摘要, 通过比对函数摘要信息, 判断两个函数间的输入、输出之间是否存在相似关系, 进而判断函数语义 (功能) 等价性。

4.1 函数功能相似性的判定原理

程序语义反映的是功能, 如果两个函数的功能相同, 则语义无差别。而函数的功能可以使用输入输出的映射关系来刻画。对于相同的输入, 不同的函数执行后能够得到相同的输出, 可认为它们的功能相同。为了实现函数级的语义分析, 本文使用静态符号执行技术, 是单元程序分析的重要方法。在函数内以符号值代替具体值, 实现函数的符号化模拟执行。即对某个函数静态模拟执行时, 所有的函数参数进行符号化, 用符号值代入运算; 在执行结束后, 收集函数所有输出形态 (全局变量、返回值、引用参数、指针等) 的运算符号值。

本文中函数的输入指的是函数参数, 以及全局变量和函数中定义的变量; 输出指的是参与函数内部执行的全局变量、函数返回值、引用参数、指针等。一般来说, 语义相同的函数在相同的参数符号化条件下, 也有相同的输出形态的符号值对应。

4.2 基于路径包络分析的执行结果计算

函数的输出因为路径条件的约束可能会产生

多个, 即多条执行路径可能会产生不同的输出值。对于执行路径比较复杂的函数, 如函数体包含的子调用、循环等较多时, 如果遍历每条路径将输入参数代入符号值运算, 以计算函数输出符号值, 显然模拟计算带来的开销和庞大的路径导致计算量很大, 效率较低。为了快速分析全部的输出值, 可以对控制程序执行路径的约束条件进行归约, 将对函数输出符号值产生相同程序依赖影响的路径集合构建成路径包络, 在路径包络内部计算输出符号值的计算, 而无须遍历全部路径。

4.2.1 基于 PDG 的路径包络构建

路径包络归约是为了提取影响目标点符号值的关键路径控制条件, 而去除对目标点符号值无影响的路径条件, 即无用控制条件。路径包络的构建是建立在对程序控制和数据依赖分析基础上的, PDG 能够反映程序的控制依赖关系和数据依赖关系。因此, 可以根据 PDG 的相关分析, 建立路径包络。本文中研究的目标点是 4.1 节提到的函数输出, 下文均用目标点阐述。

路径包络本质是满足相同程序依赖关系的一组路径集合, 可由路径约束条件及其布尔值进行表示。如路径包络 $\langle (2, T), (4, T), (6, F) \rangle$ 表示第 2、4、6 分支分别取值为 T 、 T 、 F 时的路径集合。

探索从函数入口点到目标点具备相同运算符号值的路径集合, 如图 2 所示, 基本流程是: 通过入口点和目标点的公共依赖域分析, 建立局部程序依赖关系; 通过反向数据依赖分析, 探索与目标点具有数据依赖关系的语句集合; 如果没有

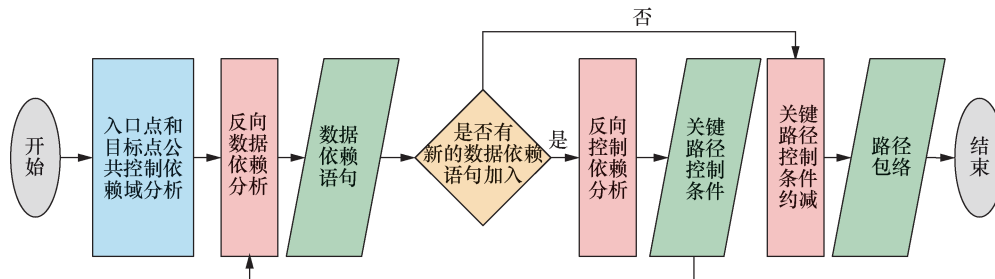


图 2 路径包络构建流程

新的数据依赖语句加入则结束, 否则, 通过反向控制依赖分析, 探索与数据依赖语句具有控制依赖关系的关键路径控制条件; 以关键控制路径条件为目标点, 探索与之具有数据依赖关系的语句集合; 当新的数据依赖语句不再发生变化时, 对具有两个布尔值的关键路径控制条件进行消减(无用控制条件), 构建路径包络。

针对包络构建流程, 本文提出了面向函数的路径包络构建算法(PCE)。该算法主要功能是在PDG上分析程序的控制依赖和数据依赖关系, 根据函数入口点 e 和函数输出点 t , 找到 e 到 t 的路径包络集合 PC 。

算法 路径包络构建算法

输入 F : 待分析函数

E : 函数入口点

T : 函数输出点

输出 PC —归约的路径包络

- 1) $PC = \Phi$
- 2) $DataDependenceCut(e, t)$
- 3) $PC = GatherPC(e, t')$
- 4) $PCEtoT = FindRS(e, t)$
- 5) $PC = PC \oplus PCE\ to\ T$
- 6) $DataDependenceCut(R_{ifather}, t)$
- 7) $PC1 = GatherPC(R_{ifather}, t)$
- 8) $PC = PC \oplus PC1$
- 9) return PC

路径包络构建算法描述如下。

步骤 1 初始化归约的路径包络为空。

步骤 2 基于 PDG 分析以 e 为根节点的树上所有节点与节点 t 的数据依赖关系, 剪除与节点 t 数据依赖无关的控制依赖域节点及其子树。

步骤 3 生成 e 到 t' 的路径包络, t' 是 e 的直接后继节点, 并且是 t 的祖先。

步骤 4 基于 PDG 上从 e 到 t 的路径, 收集控制依赖域节点, 同时收集从 e 到 $R_{ifather}$ 路径条件。

$R_{ifather}$ 是在从 e 到 t 的路径上, 距离 t 最近的控制依赖域节点的祖先。

步骤 5 生成从 e 到 $R_{ifather}$ 的路径包络。

步骤 6 基于 PDG 分析以 $R_{ifather}$ 为根节点的子树上的所有节点和 t 的数据依赖关系, 剪除与节点 t 数据依赖无关的控制依赖域节点及其子树。

步骤 7 生成从 $R_{ifather}$ 的直接后继节点到 t 的路径包络 $PC1$ 。

步骤 8 将 e 到 $R_{ifather}$ 的路径包络与 $PC1$ 联合, 生成 e 到 t 的路径包络。

步骤 9 输出归约的路径包络结果。

其中, 子算法 $DataDependenceCut$ 和 $GatherPC$ 在文献[17]中已经提出。

4.2.2 函数执行结果符号值计算

本文采用符号执行以实现函数功能的相似性比较, 见 4.1 节。因为只需要比较函数输出时的符号值是否相同, 无须进行具体值的运算。但在符号执行过程中, 会遇到外部调用、循环等问题, 如果处理不当, 会影响输出符号值的计算结果。

鉴于函数内部单元符号执行特点和语义比对的需求, 在处理外部调用和循环问题时可简化处理。

1) 外部调用: 不跟进外部调用的执行, 将调用的子函数名直接作为符号带入后续符号执行过程。包含子函数名的符号值相当于黑盒运算。

2) 循环计数: 将循环计数与函数变量或输入参数关联, 循环体内的运算结果与循环计数关联。

因为语义分析是针对函数输出符号值的比较, 不需要进行约束求解, 因此可以接受符号化的子函数调用和循环计数。

4.3 基于函数摘要分析的语义比对

在补丁文件发布时, 可能加入针对控制流图或函数调用图的混淆机制^[14-15], 干扰补丁分析结果。为了有效识别混淆代码, 剪除不可能执行到达的分支路径以及由此带来的函数伪输出, 需要对包络的关键路径条件进一步分析, 发现可能存

在的约束冲突，将该包络产生的函数输出标记为伪输出，从函数执行结果的比对中剔除。

函数摘要技术一般用于频繁执行某个函数时，只在第一次执行或未执行时静态分析，建立该函数的输入输出映射关系，以便在后续调用该函数时可以复用摘要信息，代替实际执行该函数。

在本文中，通过路径包络的构建，建立函数输入、输出符号值之间的映射关系。函数摘要用三元组<Input, Constraint, Output>表示，如下。

1) **Input**: 表示函数输入，这里的输入包括全局变量、函数体内定义的局部变量、函数参数等对应的符号值。

2) **Constraint**: 表示路径包络对应的关键路径条件及其取值。

3) **Output**: 表示当输入符号值为 Input，沿着在 Constraint 约束下程序路径执行时，得到的程序输出符号值。这里的输出指的是参与函数内部执行的全局变量、函数返回值、引用参数、指针等。

函数摘要构建时，为了解决混淆代码的影响，需要对路径包络的关键约束条件进行相容检查，发现可能存在的约束冲突，从而将不可能产生的函数输出和对应的 Output 和约束条件 Constraint 删除，在一定程度上可以缓解混淆代码的影响。

约束相容性检查后，对处理过的函数摘要进行比较。相同语义的函数对，可能因为分配的输

入符号不同，导致输出符号值不同。这里对输出符号值的比较不能用简单的匹配方式实现，必须找到一组相应的序列映射，才可能判断函数是否有语义差别。具体的形式化定义描述如下。

判定规则函数功能相同：如果对于任意两个函数 f_A 和 f_B , f_A 的输出 $Output_A$ 和 f_B 的输出 $Output_B$ 满足 $E(x)=x$ 且 $E(x)$ 是双射 (其中, $x \in Output_A$, $E(x) \in Output_B$), 那么定义 f_A 和 f_B 功能相同。

5 系统实现

按照前文设计思路，本文实现了补丁比对工具 SymDiff，其基本构成如图 3 所示。

1) 反汇编器

系统使用的反汇编器是 IDA Pro，本文设计的补丁比对插件 SymDiff 是在 IDA PRO 5.5 版本上实现的。系统需要使用 IDA 的反汇编及插件功能。

2) 函数级同构比较

运用图同构匹配方法进行结构化比对，生成最大同构子图。函数签名模块从待比较文件中提取签名信息，利用同构算法在函数调用图级进行分析，生成比对结果。详细的过程见第 3 节内容。

3) 函数级语义比较

在最大同构子图基础上，运用符号执行的方法对函数输入输出特征进行语义级比对。也就是说，对于集合 M_C 中的函数对进行语义比较，找出发生了功能变化的匹配函数对，从而找到补丁具体修补的指令。具体方法见第 4 节内容。

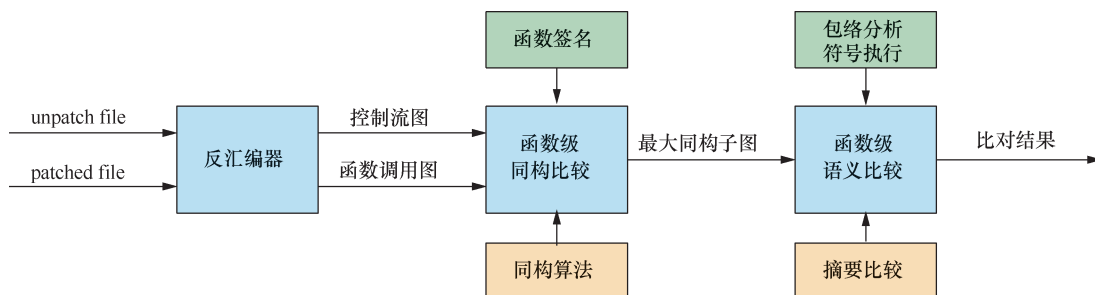


图 3 SymDiff 系统结构

6 实验分析

1) 测试目的

本节通过 SymDiff 与当前常用的开源补丁比对工具 PatchDiff2 进行比对测试，以验证本工具在确定可疑函数精度方面的优势。

2) 测试工具

自主开发的补丁比对分析工具 SymDiff 和开源工具 PatchDiff2 如表 1 所示。

表 1 比对测试工具

工具名称	来源
SymDiff	自主研发
PatchDiff2	由 Tenable Network Security 发布的免费比对工具

3) 测试对象

CVE-2006-3439、CVE-2008-4205、CVE-2010-2746、CVE-2017-6178 这 4 个漏洞对应的原文件和补丁文件，如表 2~表 5 所示。

表 2 CVE-2006-3439 文件

文件类型	文件名称	版本
原文件	netapi32.dll	5.00.2195.6949
补丁文件		5.00.2195.7105

表 3 CVE-2008-4205 文件

文件类型	文件名称	版本
原文件	netapi32.dll	5.1.2600.2180
补丁文件		5.1.2600.2952

表 4 CVE-2010-2746 文件

文件类型	文件名称	版本
原文件	comctl32.dll	6.00.2900.5512
补丁文件		6.00.2900.6028

表 5 CVE-2017-6178 文件

文件类型	文件名称	版本
原文件	USBPcap.sys	1.0.0.7
补丁文件		1.2.0.1

4) 测试方法

使用两个工具对 4 个比较对象进行补丁比对，记录漏洞所在函数/修补函数对数量 x ，记录可疑函数对 M_C 数量 y 。通过 x 和 y 的比值，判断比对工具对可疑函数筛选的精度。

5) 测试结果

M_C 中的函数往往是漏洞存在或修补的位置，也是漏洞分析人员关注的重点。如果该函数集合范围较小，能提高人工分析效率。由于结构化比对存在误报，将结构化差异误认为语义差异，导致 M_C 函数过多。本次测试使用漏洞及修补函数对与 M_C 函数对做比较，得出的比例作为比对结果的精度。表 6 是比对测试结果，显示 SymDiff 工具具有优势，这也源于使用函数级的语义比较修正了函数同构匹配的结果。

表 6 比对测试结果

文件名	工具	漏洞/修补函数对 x	M_C 函数对 y	$\frac{x}{y}$ 的比值
CVE-2006-3439	SymDiff	1	7	14.3%
	PatchDiff2	1	7	14.3%
CVE-2008-4205	SymDiff	1	2	50%
	PatchDiff2	1	3	33.3%
CVE-2010-2746	SymDiff	1	2	50%
	PatchDiff2	1	3	33.3%
CVE-2017-6178	SymDiff	5	6	83.3%
	PatchDiff2	5	8	62.5%

7 结束语

本文阐述了函数级语义比对在软件补丁比对中的应用。软件补丁比对分析是软件漏洞分析的重要方法，可以对 1day 漏洞进行快速定位，分析成因。经典的结构化比对方法虽然获得了较好的效果，但是随着反二进制比对混淆代码技术的出现，比对的难度不断增加。为了应对这些挑战并综合考虑符号执行技术本身的瓶颈，本文提出了将函数级符号执行运用于语义差异分析的方法，

尽可能减少结构化比对带来的误差。

参考文献：

- [1] WANG Z, PIERCE K, MCFARLING S. BMAT-a binary matching tool for stale profile propagation[J]. The Journal of Instruction-Level Parallelism (JILP), 2000,2:1-20.
- [2] FLAKE H. Structural comparison of executable objects[C]//Dortmund GI SIG SIDAR Workshop, 2004.
- [3] 潘璠, 吴礼发, 孙传鲁, 等. 一种改进的补丁比较模型的研究与实现[J]. 南京邮电大学学报(自然科学版), 2012, 32(2): 75-83.
PAN F, WU L F, SUN C L, et al. Research and implementation of an improved patch comparison technique[J]. Journal of Nanjing University of Posts and Telecommunications(Natural Science), 2012, 32(2): 75-83.
- [4] 肖雅娟. 二进制代码函数相似度匹配技术研究[D]. 济南: 山东大学, 2016.
XIAO Y J. Research on similarity matching technology of binary code function[M]. Jinan: Shandong University. 2016.
- [5] LIU B C, BINGCHANG L, WEI H, CHAO Z, et al. α Diff: cross-version binary code similarity detection with DNN[C]//IEEE ACM Automated Software Engineering (ASE'18). 2018.
- [6] XU X J, LIU C, FENG Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]//The 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017.
- [7] 王欣, 郭涛, 董国伟, 等. 基于补丁比对的Concolic测试方法[J]. 清华大学学报(自然科学版), 2013, 53(12): 1737-1742.
WANG X, GUO T, DONG G W, et al. Concolic test method based on patch matching[J]. Journal of Tsinghua University(Science and Technology), 2013, 53(12): 1737-1742.
- [8] 王嘉捷, 郭涛, 张普含, 等. 基于软件代码差异分析的智能模糊测试[J]. 清华大学学报(自然科学版), 2013, 53(12): 1726-1730.
WANG J J, GUO T, ZHANG P H, et al. Intelligent fuzzy test based on software code difference analysis[J]. Journal of Tsinghua University(Science and Technology), 2013, 53(12): 1726-1730.
- [9] SHAHZAD M, SHAFIQ M Z, LIU A. X. A large scale exploratory analysis of software vulnerability life cycles[C]//International Conference on Software Engineering. 2012: 771-781.
- [10] FREI S, MAY M, FIEDLER U, et al. Large-scale vulnerability analysis[C]//SIGCOMM Workshop on Large-Scale Attack Defense. 2006: 131-138.
- [11] BRUMLEY D, POOSANKAM P, SONG D, et al. Automatic patch-based exploit generation is possible: techniques and implication[C]//Security & Privac. 2008.
- [12] AVERY J, SPAFFORD E H. Ghost patches: fake patches for fake vulnerabilities[C]//International Conference on ICT Systems Security and Privacy Protection. 2017: 399-412.
- [13] JEONG W O, CHEN S. Binary and anti-binary comparison[C]//XCon Information Security Conference. 2013: 1-27.
- [14] DEBIN G, MICHAEL K R, DAWN S. BinHunt: automatically finding

semantic differences in binary programs[C]//The 10th International Conference on Information and Communications Security. 2008.

- [15] LANNAN L, JIANG M, DINGHAO W, et al. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection[J]. IEEE Transactions on Software Engineering, 2017,43(12): 1157 - 1177.
- [16] ZHENG Z X, BIHUA C, MAHINTHAN C, LIU Y, et al. SPAIN: security patch analysis for binaries towards understanding the pain and pills[C]//The 39th International Conference on Software Engineering. 2017.
- [17] CAO Y, WEI Q, WANG Q X. Research on parallel symbolic execution through program dependence analysis[C]//The Fifth International Symposium on Computational Intelligence and Design. 2012: 222-226.

【作者简介】



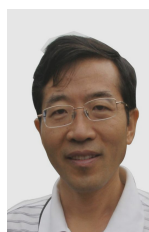
曹琰(1983—), 男, 河南郑州人, 博士, 数学工程与先进计算国家重点实验室讲师, 主要研究方向为网络空间安全。



刘龙(1983—), 男, 河南尉氏人, 数学工程与先进计算国家重点实验室讲师, 主要研究方向为网络空间安全和机器学习。



王禹(1984—), 男, 河北博野人, 博士, 河南工程学院讲师, 主要研究方向为网络空间安全和IPv6。



王清贤(1960—), 男, 河南新乡人, 数学工程与先进计算国家重点实验室教授、博士生导师, 主要研究方向为网络空间安全和软件分析。