# CFG2VEC：用于跨架构软件逆向工程的层次图神经网络

Shih-Yuan Yu[1], YonatanGizachew Achamyeleh[1],Chonghan Wang[1],Anton Kocheturov[2],Patrick Eisen[2], Mohammad Abdullah AlFaruque[11]加利福尼亚大学欧文分校电气工程与计算机科学系，美国加利福尼亚州欧文{*shihyuay, yachamye, chonghaw, alfaruqu*}*@uci.edu*[2]西门子科技，美国新泽西州普林斯顿，{*anton.kocheturov, patrick.eisen*}*@siemens.com*

arXiv:2301.02723v1 [cs.SE] 6 Jan 2023

摘要——**任务关键型嵌入式软件对社会基础设施至关重要，但随着技术进步，也可能出现新的安全漏洞。当安全问题出现时，逆向工程师（REs）会使用软件逆向工程（SRE）工具分析易受攻击的二进制文件。然而，现有工具的支持有限，逆向工程师需要经历耗时、成本高且容易出错的过程，并凭借经验和专业知识理解软件行为及漏洞。为改进这些工具，我们提出了*cfg2vec*，一种基于层次图神经网络（GNN）的方法。为了表示二进制文件，我们提出一种新颖的图中图（GoG）表示，将控制流图和函数调用图的信息结合起来。我们的*cfg2vec* 利用层次 GNN 和基于孪生网络的监督学习架构，学习如何表示由各种 CPU 架构编译而来的每个二进制函数。我们评估了*cfg2vec*从剥离符号的二进制文件中预测函数名称的能力。结果表明，*cfg2vec* 在预测函数名称方面比最先进方法高出 24.54%，在训练数据更多时，性能甚至可提升 51.84%。此外，*cfg2vec* 在所有 CPU 架构上均持续优于最先进方法，而基线则需要多次训练才能达到相近性能。更重要的是，我们的结果表明，*cfg2vec* 能够处理由未见过的 CPU 架构构建的二进制文件，说明我们的方法可以推广所学知识。最后，我们通过将其实现为*Ghidra* 插件，并在解决 DARPA*Assured MicroPatching*（AMP）挑战期间使用，证明了其实际应用价值。**

索引词—**软件逆向工程；二进制分析；跨架构；机器学习；图神经网络；**

## I. 引言

在任务关键型系统中，嵌入式软件对于操控物理过程以及执行可能对人类操作员造成风险的任务至关重要。近年来，物联网（IoT）创造了价值19万亿美元的市场，并使联网设备的数量大幅增长，在 2025 [1]–[3]年达到约350亿台。然而，尽管物联网推动了技术发展，却也无意中使任务关键型系统暴露于新型漏洞之下 [4]–[6]。据报道，物联网网络攻击数量在 2019 [7],增长了300%，而发现的软件漏洞数量则从1.6k增加到100k [8]。正如[9], 所示，其后果可能十分严重：心脏出血漏洞 可能导致最多64K内存泄露，不仅威胁个人信息安全，也威胁组织的信息安全。此外，*Shellshock* 是一个 Bash 命令行界面 Shell 漏洞，存在至今已有30年，如今仍对企业构成威胁 [11], [12]。对于任务关键型系统，即使意外中断仅持续数小时或数分钟，也可能造成数百万美元的损失 [13]。因此，及时分析受影响的软件并为漏洞打补丁至关重要。

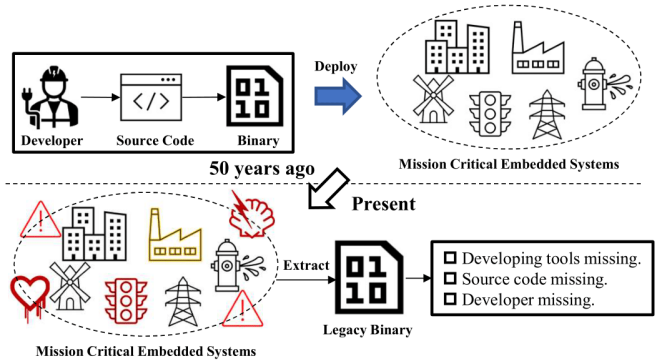



图 1. 遗留软件生命周期。

然而，由于任务本身至关重要，任务关键型系统通常会使用能够持续运行数十年的软件。随着时间推移，这些系统会成为遗留系统，新发现的威胁数量也可能不断增加（如图 1 所示）。通常，对于遗留软件而言，原始开发环境、维护支持或源代码可能已不复存在。为解决漏洞，供应商会根据当前软件版本（例如，版本 0.9）提供源代码更改形式的补丁。然而，遗留系统中唯一可用的数据，是基于其源代码构建的二进制文件（例如，版本 0.1）。这种版本差距给遗留二进制文件应用补丁带来了挑战，因此，给遗留软件应用补丁的唯一办法就是直接进行二进制分析。如今，如图 2 所示，逆向工程师（逆向工程师）必须借助软件逆向工程 （软件逆向工程）

本材料基于国防高级研究计划局（DARPA）和太平洋海军信息战中心（NIWC Pacific）在合同编号N66001-20-C-4024项下支持的研究成果。文中表达的观点、意见和/或研究结果仅代表作者观点，不应被理解为代表美国国防部或美国政府的官方观点或政策。

发行声明“A”（获准公开发布，不受发行限制）。

工具，例如 *Ghidra* [14], *HexRays* [15], 和 *radare2* [16], 先将二进制文件反汇编并反编译为更高层次的表示（例如, C语言或 C++）。通常，这些工具会利用调试信息、字符串、符号表和二进制文件来重建函数名和变量名，使逆向工程师无需访问源代码即可重建软件的结构和功能 [17]。对于逆向工程师而言，这些符号蕴含着源代码的上下文信息，能为他们理解程序逻辑提供宝贵线索，从而帮助他们为易受攻击的二进制文件打补丁。然而，在内存有限的任务关键型遗留系统中，为了缩减二进制文件的体积，通常会剔除符号。由于从剥离符号的二进制文件 中恢复符号并非易事，大多数反编译器都会为代码元素分配毫无意义的符号名。至于理解软件语义，逆向工程师必须凭借自身经验和专业知识处理这些信息，并解读每个代码元素的语义。

近期的研究利用机器学习（ML）应对这些挑战，旨在从原始二进制文件中恢复程序信息。例如， [18], 和 [19] 将代码特征与函数名称关联起来，并对这些代码特征与相应源代码级信息之间的关系进行建模（ [19], 中的变量名，以及 [18]中的变量名和函数名称）。与此同时， [20]和 [21]采用编码器-解码器网络结构，根据指令序列和控制流，从剥离符号的二进制函数中预测函数名称。然而，这些方法都不支持跨架构调试信息重建。另一方面，也有研究致力于在机器学习模型中实现跨平台能力 [22]–[24]。这些研究专注于对二进制代码相似性进行建模：从带有属性特征的控制流图（控制流图）中提取实值向量，然后计算由不同 CPU 架构构建的二进制函数特征向量之间的结构相似性。

本文作为西门子、约翰斯·霍普金斯大学应用物理实验室（JHU/APL）、BAE Systems（BAE）和 UCI 之间的一项跨行业与学术界联合倡议的一部分，提出了 *cfg2vec*。该方法利用层次化图神经网络 （GNN）重建每个二进制函数的名称，旨在提升任务关键型系统中快速为遗留二进制文件打补丁的能力。我们的*Cfg2vec* 构建了一种图中图 （GoG）表示，将控制流图和函数调用图相结合，以建模二进制函数表示与其语义名称之间的关系。此外，得益于孪生网络架构的设计，*cfg2vec* 能够处理跨架构二进制文件，如图 3 所示。跨架构反编译的一个重要应用场景是打补丁，其目标是识别已知漏洞或错误并应用补丁。然而，当存在错误的软件能够针对多种硬件架构各异的设备进行编译时，就会出现架构差异问题。例如，要为十年前针对某种特殊嵌入式架构编译、且易受心脏出血漏洞 [10]等已知攻击影响的剥离符号的二进制文件打补丁，颇具挑战性。尽管软件中提供了参考补丁，但参考架构可能并非随时可得，也可能没有相关文档，或者供应商可能已不复存在。在这种情况下，映射不同架构间的代码特征会非常有帮助。即使参考补丁是针对不同类型的 CPU 架构构建的，这也有助于识别存在漏洞的剥离符号的二进制文件与其参考补丁之间的代码相似性。对于*cfg2vec*，我们的目标贡献如下：

- 我们提出用图的图（GoG）表示二进制函数，并展示了这种表示在从剥离符号的二进制文件中重建函数名称方面的实用性。
- 我们提出了一种新颖的方法*cfg2vec* ，它使用分层图神经网络（GNN）对二进制文件中的控制流和函数调用关系进行建模。
- 我们提出在训练时使用跨体系结构损失，使*cfg2vec* 能够捕获不依赖于体系结构的二进制文件表示。
- 我们在 GitHub 仓库中发布*cfg2vec*： https://github.com/AICPS/mindsight cfg2vec。
- 我们将*cfg2vec* 集成到一个实验性 Ghidra 插件中，以协助在真实场景中为 DARPA*Assured MicroPatching*（AMP）挑战赛的二进制文件打补丁。

本文结构如下：第二节讨论相关工作和基础知识，以便更好地理解本文。第三节介绍*cfg2vec*，包括问题形式化、数据预处理以及主要流程。第四节展示实验结果。最后，我们在第五节总结全文。

## II. 相关工作

本节介绍软件逆向工程的背景，探讨利用机器学习改进逆向工程的相关研究，并介绍用于二进制分析的图学习。

### *A.* 软件逆向工程

软件逆向工程（SRE）旨在无法访问源代码的情况下理解程序的行为，常用于恶意软件检测 [25], [26], 漏洞发现以及遗留软件中的缺陷打补丁等多种应用 [27], [28]。逆向工程师（*REs*）用来检查程序的一种主要工具是反汇编器，它可将二进制文件转换为低级汇编代码。这类工具包括*GNU Binutils* 的 *objdump* [29],*IDA* [15], 、Binary Ninja [30], 和 Hopper [31]。然而，即使有这些工具，逆向工程专家仍需付出相当大的认知努力，才能在汇编级进行推理。

近来，逆向工程师使用 反编译器 （如 *Hex-Rays* [32], 或 *Ghidra* [14]）进一步将反汇编器的输出转换为类似 C 或 C++等高级编程语言的代码，从而逆转编译过程，减轻理解汇编代码的负担。通过分析汇编指令，这些反编译器可以运用程序分析和启发式方法，重建二进制文件中的变量、类型、函数和控制流结构。然而，即使这些反编译器生成了更高层级的输出以便更好地理解代码，反编译结果仍不完整。

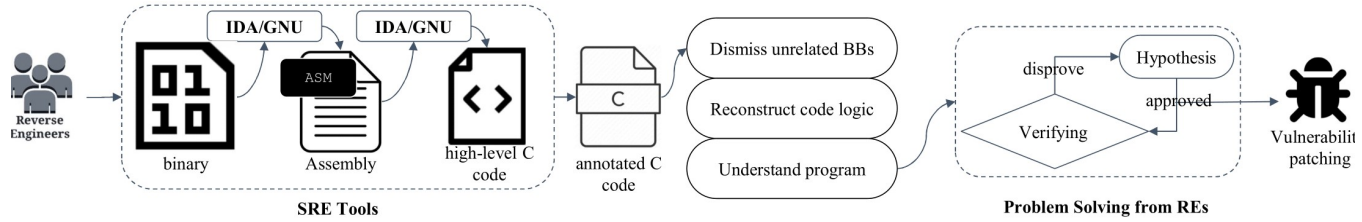



图 2. 解决安全问题的逆向工程流程。

原因在于，编译过程会丢弃源代码级信息并降低其抽象层级，以换取更小的占用空间、更快的执行速度，甚至满足安全方面的考量。注释、变量名、函数名和惯用结构等源代码级信息对于理解程序至关重要，但通常无法从这些反编译器的输出中获得。

如图 2 所示，逆向工程师使用反汇编器或反编译器生成高级源代码。此外， [33]表示逆向工程师会做笔记，并为与漏洞相关的关键函数命名。由此，他们可以基于机器生成的高级源代码创建带注释的源代码。在注释源代码时，逆向工程师还会分析与漏洞相关的重要部分，并忽略常规指令或无关代码。同时，理解函数之间的逻辑流也是他们完成任务时必须重点解决的另一项主要工作。经过分类、注释和理解后，逆向工程师会尝试多种可行的补救措施，以找到修复漏洞的正确补丁。

### *B.* 逆向工程中的机器学习

软件二进制分析是提升安全性的直接第一步，因为开发者通常以二进制文件的形式部署软件 [34]。通常，专家会通过理解编译源信息、函数签名和变量信息来进行补丁修复或漏洞分析。然而，在编译后，这些信息通常会被有意剥除或混淆（例如，混淆）。在这种情况下，软件二进制分析变得更具挑战性，因为开发者必须凭借自身经验和专业知识恢复源代码级信息。早期针对二进制文件的恢复工作侧重于人工补全，但逆向工程效率低、成本高且容易出错。

随着机器学习 (ML)的推理能力显著提升，应用机器学习并重建更高层次的源代码信息，以替代基于人工的方法，已受到研究界的广泛关注。例如， [35] 是首个采用基于神经网络和基于图的模型的方法，通过预测函数类型来帮助逆向工程师理解二进制文件。 [36]也利用神经网络预测函数名，并聚合二进制文件向量各个片段的相关特征。随后，它分析源代码（例如 Java）中各函数与其对应函数名之间的联系，以进行函数名称预测。另一方面， [18], 并未使用神经网络。它结合了基于决策树的分类算法和基于概率图模型的结构化预测，通过分析符号名、类型和位置来匹配函数名。然而， [18] 只能从预先确定的封闭集合中进行预测，无法推广到新的名称。

由于用于命名函数的语言与自然语言相似，近期研究开始倾向于采用自然语言处理（NLP） [20], [21], [37]。具体而言，这些模型根据库中的函数名预测语义词元，并在推理时将函数名纳入其中。其基本前提是，每个词元都以某种方式对应函数的属性和功能。 [20] 使用控制流图（CFG）预测函数名。它将静态分析与长短期记忆网络和Transformer神经网络模型相结合，从而实现函数命名。然而，由于数据集包含的数据分布不平衡且特征不足，其规模有限，妨碍了性能的进一步提升。[37] 旨在解决数据集的局限性。它提供了*UbuntuDataset* ，其中包含来自 2.2 万个软件的 900 多万个函数。 [21] 通过构建大型数据集展现了该框架的有效性。该框架在建模和实现函数名称预测时，考虑了汇编代码中细粒度的序列和结构信息。同时， [21]在保持基本语义不变的情况下，降低了数据（指令或单词）的多样性，这与自然语言处理中的词干提取和语义处理类似。然而，这些研究在预测任务上的精确率较低，例如， [21], 正确预测函数名子词元的比例仅约为 41%。此外，未知函数的推理指标要低得多， [21],这使逆向工程师难以在实践中从中受益。

尽管许多现有工作能够重建源代码级信息，但它们都不支持重建跨平台调试信息。交叉编译在软件开发中日益普及。例如，硬件制造商通常会在运行于不同体系结构的多个设备上重复使用同一固件代码库 [38]。如果我们拥有针对一种体系结构编译的剥离符号的二进制文件，以及针对另一种体系结构编译、包含调试符号的相似程序的二进制文件，那么能够执行跨架构函数名预测/匹配的工具将大有裨益。我们可以利用包含调试符号的二进制文件，预测剥离符号的二进制文件中的函数名，从而显著助力调试。能够提取二进制文件架构无关特征的工具

也有助于恶意软件检测，因为恶意软件的源代码可以针对不同体系结构进行编译 [38], [39]。比较不同体系结构的两个二进制文件会更加复杂，因为它们使用不同的指令集、调用约定、寄存器集等。此外，由于不同体系结构的行为略有差异，通常无法直接比较来自不同体系结构的汇编指令 [40]。通过从仅针对一种体系结构编译的二进制文件中学习恶意函数的特征，跨架构函数名预测将有助于在针对不同体系结构编译的程序中找到该恶意函数。上述工具并非架构无关，因此无法将其用于此类应用。为了解决上述缺陷、帮助开发更高效的反编译器，并让逆向工程更易于开展，我们提出了*cfg2vec*。通过采用跨架构孪生网络架构，我们的*cfg2vec* 能够学习提取稳健特征，其中涵盖平台无关特征，从而实现跨架构二进制文件的函数名称重建，提升最先进方法的水平。

### C. 二进制分析的图学习

图学习已成为多个领域中的一种实用方法 [41]–[44]。尽管传统机器学习能够有效捕获欧几里得数据中的隐藏特征，例如图像、文本或视频，但我们的工作更侧重于核心数据为图结构数据的应用。图可能是不规则的，并且可能包含数量不定、无序排列的节点；此外，节点的邻居数量也可能各不相同，这使得深度学习中的数学运算（例如二维卷积）难以应用。传统机器学习方法中的运算只能通过将非欧几里得数据投影到低维嵌入空间来实现。在图学习中，图嵌入（GE）可以将图转换为一个向量（图的嵌入）或一组向量（节点或边的嵌入），同时保留图的相关信息和结构信息 [41]。图神经网络（GNN）是一种旨在以端到端方式解决图相关任务的模型，其核心思想是通过聚合节点自身的表示及其邻居的表示来生成该节点的表示 [42]。图神经网络通过堆叠多个图卷积层、图池化层和图读出操作，从高维图结构数据中生成低维图嵌入。

在软件二进制分析中，许多方法将控制流图(CFG)用作主要表示。例如，*Genius* 通过提取每个基本块(BB)的原始属性，为每个二进制函数构建属性控制流图(ACFG)表示。基本块是一段直线代码序列，除入口和出口外，不存在分支；在 ACFG 中表示为 [22]。*Genius* 通过二分图匹配算法衡量一对属性控制流图之间的相似度，然后根据相似度对这些属性控制流图进行聚类。*Genius* 利用码本，根据相似度检索属性控制流图的嵌入表示。另一种方法*Gemini*提出了一种基于深度神经网络的模型，并采用孪生架构，以比当时其他最先进模型更高的效率和准确率来建模二进制相似性 [23]。*Gemini* 接收一对从已知代码漏洞对应的原始二进制函数中提取的属性控制流图，随后在其网络架构中使用共享的*Structure2vec*模型对它们进行嵌入。完成嵌入后，*Gemini*使用损失函数训练模型，该损失函数计算两个嵌入表示之间的余弦相似度。*Gemini*的表现优于*Genius* 等模型以及二分图匹配等其他方法。在相关文献中，还有其他研究将函数调用图(FCG)作为二进制分析中用于恶意软件检测的主要数据结构 [45]。我们的 *cfg2vec* 结合控制流图和函数调用图，提取相关的平台无关特征，从而为跨架构高级信息重建任务（例如函数名）生成图的图(GoG)表示。

## III. CFG2VEC 架构

本节首先介绍问题形式化。接下来，如图4所示，我们描述 *cfg2vec* 如何从每个软件二进制文件中提取图中图（GoG）表示。最后，我们介绍*cfg2vec*中的网络架构。

### A. 问题形式化

在我们的工作中，给定一个由不同 CPU 架构编译得到的二进制代码，记为 $p$，我们提取图中图（GoG）表示 $\mathcal{G} = (\mathcal{V}, \mathcal{A})$，其中 $\mathcal{V}$是节点集合，$\mathcal{A}$ 是邻接矩阵（如图 3 所示）。$\mathcal{V}$ 中的节点表示函数，$\mathcal{A}$ 中的边表示它们之间的交叉引用关系。也就是说，节点 $f_i \in \mathcal{V}$中的每个节点都是一个控制流图（CFG），我们将其表示为 $f_i = (B, A, \phi)$，其中 $B$ 中的节点表示基本块，$A$ 中的边表示它们之间的依赖关系。$\phi$ 是一个映射函数，用于将汇编形式中的每个基本块映射到其提取出的相应属性 $\phi(v_i)$；$= C^k$ 其中 C 是一个数值，k 是该基本块（BB）的属性数量。控制流图结构旨在提供较低基本块级别的更多信息，而 GoG 结构则用于恢复 CFG 之间更高层次函数级别的信息。图3展示了一个部分 GoG 结构的示例，并进一步观察了其中一个 CFG 节点以及另一个单独的 CFG 基本块节点，同时展示了与该基本块节点对应的特征集合。我们的目标是设计一种高效且有效的图嵌入技术，用于重建 $f_i \in \mathcal{V}$中每个函数的函数名称。

### B. 用于图提取的 *Ghidra Data ToolKit*

为了提取*cfg2vec*所需的结构化表示，我们借助最先进的反编译器*Ghidra* [14] 以及*Ghidra* 无头分析器[1]。无头分析器 是*Ghidra* 的命令行版本，允许用户通过命令行界面执行*Ghidra* 支持的许多任务（例如分析二进制文件）。为了从二进制文件中提取 GoG

[1]*Ghidra* 无头分析器文档：https://ghidra.re/ghidra docs/analyzeHeadlessREADME.html

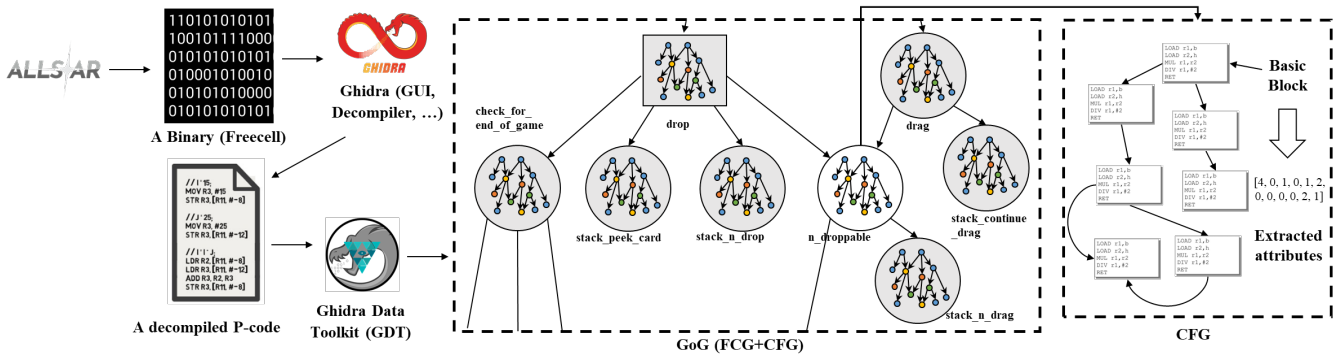



图 3. 一个 由 amd64 CPU 架构下编译的 Freecell 软件包中的二进制文件构成的 图中图(GoG) 示例

表示，我们开发了*Ghidra Data Toolkit* （GDT）；GDT 是一组基于 Java 的元数据提取脚本，用于对*Ghidra* 无头分析器进行插桩。首先，GDT 以编程方式分析给定的可执行文件，并将提取的信息存储在 Ghidra 数据库中。Ghidra 提供了一组 API，用于访问数据库并检索有关已分析二进制文件的信息。GDT 使用这些 API 导出每个函数的信息，例如 Ghidra 的 PCode 和调用图。具体而言，*FunctionManager* API 使我们能够操作二进制文件中每个反编译函数的信息，并获取函数之间的交叉调用依赖关系。对于每个函数，我们使用另一个 Ghidra API*DecompInterface*[2] 来提取与函数中每个基本块相关的 12 项属性。这些属性具体对应于指令总数，包括算术、逻辑、转移、调用、数据传输、静态单赋值（SSA）、比较和指针指令，以及不属于上述类别的其他指令；还包括该基本块中的常量和字符串总数。最后，通过整合所有信息，我们为每个二进制文件构建 GoG 表示$\mathcal{G}$ $p$。我们重复这一过程，直到所有二进制文件都转换为 GoG 结构。随后，我们以批次形式将得到的 GoG 表示输入模型，批次大小记为 B。

### *C.* 层次图神经网络

从 GDT 中提取$\mathcal{G}$ 后，我们将其输入层次网络架构（受 [46] 启发）中，该架构同时包含控制流图图嵌入层和*GoG* 图嵌入层，如图 4 所示。对于每个 GoG 结构，我们将其表示为$\mathcal{G}$ $=(\mathcal{V},\mathcal{A})$，其中$\mathcal{V}$是与$\mathcal{G}$ 相关联的一组函数，而$\mathcal{A}$ 表示$\mathcal{V}$中函数之间的调用关系。$\mathcal{V}$中的每个函数都采用控制流图 $f_i = (B, A, \phi)$ 的形式，其中每个节点 $b \in B$都是一个基本块，以定长属性向量 $b \in R^d$表示；$d$是我们前文提到的维度。 $A$ 编码了这些基本块之间成对的控制流依赖关系。

*1）CFG* 图嵌入层：我们的网络架构首先将一个批次 GoG 中的所有函数输入*CFG* 图嵌入层，该层由多个图卷积层和一个图读出操作组成。该层的输入是一个函数 $f_i = (B, A, \phi)$，输出则是表示该函数的定维向量。对于每个基本块$b_k$ ，我们令 $b_k^0 = b_k$，并使用如下所示的图卷积操作将 $b_k^t$ 更新为 $b_k^{t+1}$ ：

$$b_k^{t+1} = f_G(W b_k^t + \sum_{b_m \in A_k} M b_m^t)$$

其中， $f_G$ 是非线性激活函数，例如 ReLU；$A_k$ 是与 $b_k$ 相邻的基本块列表； $W \in R^{d\times d}$ 和$M \in R^{d\times d}$ 是在训练过程中需要学习的权重。我们执行 $T$ 轮这样的卷积，这可以作为模型中的可调超参数。在更新过程中，每个基本块利用其邻居的表示，逐渐将控制流依赖关系的全局信息聚合到自身的表示中。我们将每个基本块的最终表示记为 $b_k^T$。为获取函数 $f_i$的表示，我们应用图读出操作，例如求和读出，如下所述，

$$g^{(T)} = \sum_{b_k \in B} b_k^T \tag{1}$$

我们将 $g^{(T)}$ （即控制流图嵌入）的值赋给 $f_i$。图读出操作也可以替换为均值读出 或最大值读出。

*2) GoG* 图嵌入层：将所有函数转换为固定长度的图嵌入后，我们将$\mathcal{G}$ 输入 cfg2vec 的第二层，即*cfg2vec*的*GoG* 图嵌入层。在这里，对于每个函数 $f_i$ ，我们使用$\mathcal{F}$ 和*C*语言执行另外 $L$ 轮图卷积。更新过程如下所示，

$$f_k^{(l+1)} = f_{GoG}(U f_k^l + \sum_{f_m \in C_k} V f_m^{(l)}) \tag{2}$$

其中， $f_{GoG}$ 是非线性激活函数， $C_k$ 是函数 $f_k$ 的相邻函数（调用）列表，而 $U \in R^{d\times d}$ 和 $V \in R^{d\times d}$ 是在训练过程中需要学习的权重。最后，我们将 $f_k^{(L)}$ 作为同时考虑控制流图和 GoG 图结构的表示。我们使用这些更新后的表示进行跨架构函数相似性学习。

[2]Ghidra API DecompInterface 文档：https://ghidra.re/ghidra docs/api/ghidra/app/decompiler/DecompInterface.html

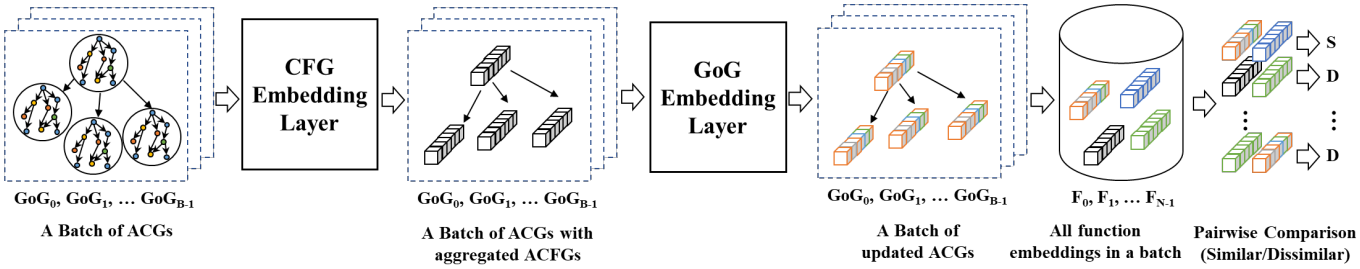



图 4. 采用有监督层次图神经网络方法的*cfg2vec*架构。

*3*）基于孪生网络的跨架构函数相似性：给定一批 GoG $B = \{GoG_1, GoG_2, ..., GoG_B\}$，我们应用层次图神经网络来获取更新后的函数嵌入集合，记为 $B_F = \{f_1^T \quad f_2^{(T} \quad \dots \quad f_K^T\}$ $()$。我们使用余弦相似度计算每对函数的相似度，记为 $\hat{} \hat{}$ $y \in [-1, 1]$。 $J$ 表示 $Y$ 与真实标签 $y$之间的损失函数，该标签表明一对函数是否具有相同的函数，计算如下：

$$J(\hat{y}, y) = \begin{cases} 1 - y, & \text{if y=1,} \\ MAX(0, \hat{y} - m), & \text{if y=-1,} \end{cases} \tag{3}$$

最终损失 $L$计算如下：

$$L = H(Y, \hat{Y}) = \sum_i (J(\hat{y}_i, y_i)), \tag{4}$$

其中， $Y$ 表示真实标签（相似性或ˆ不相似性）， $Y$表示相应的预测结果。更具体地说，如果两个函数相同，但使用不同的 CPU 架构进行编译，我们将这对函数记为相似。 $m$ 是一个常数，用于防止学习到的嵌入发生失真（默认为 0.5）。为了保持正、负训练样本之间的平衡，我们开发了一种自定义批处理算法。该函数利用以下知识：将某个软件包的二进制文件添加到给定批次后，寻找并添加同一软件包针对不同体系结构构建的二进制文件，并将其作为正样本加入所提供的批次中。同时，它还会从另一个软件包中加入一个二进制文件作为负样本。这样可以使任意批次中的正、负样本保持均衡比例。最后，我们使用损失 $L$ 通过*Adam* 优化器更新神经网络中的所有相关权重。训练完成后，我们使用该模型执行函数名称重建任务。

## IV. 实验结果

在本节中，我们评估*cfg2vec*预测函数名称的能力。我们首先介绍数据集准备和训练设置流程，然后展示 *cfg2vec* 与基线在预测函数名称方面的比较。尽管许多基线候选方案都试图解决同一问题 [18]，[20]，[21]，[37]，，但有些方案需要购买 IDA Pro 付费版本才能预处理数据集，有些甚至没有开源其实现。因此， [18] 是唯一可行的选择，因为使用我们的数据集运行其他模型几乎是不可能的。接下来，我们还展示了针对*cfg2vec*的消融研究结果。此外，我们证明了我们的*cfg2vec* 在架构无关预测方面优于基线。最后，我们展示了一个实际应用案例：将我们的*cfg2vec* 集成为*Ghidra* 插件应用，以协助解决具有挑战性的逆向工程任务。所有实验均在一台服务器上进行，该服务器配备 Intel Core i7-7820X CPU @3.60GHz、16GB RAM 和两块 NVIDIA GeForce GTX Titan Xp GPU。

### *A.* 数据集准备

我们的评估数据来源是由静态分析研究组装标注库（ALLSTAR）数据集，该数据集由应用物理实验室 （APL）托管 [47]。其中包含 30,000 多个 Debian Jessie 软件包，这些软件包是为软件逆向工程研究而针对 i386、amd64、ARM、MIPS、PPC 和 s390x CPU 架构预构建的。作者在 Docker 中使用经过修改的*Dockcross* 脚本，为每种受支持的体系结构构建各个软件包。随后，他们将每次构建生成的 ELF 文件及其符号、相应的源代码、头文件、中间文件（.o、.class、.gkd、.gimple）、系统头文件和系统库一并保存。

为了构建数据集，我们选取了那些包含针对 amd64、armel、i386 和 mipsel CPU 架构编译的 ELF 二进制文件的软件包。i386 和 amd64 广泛用于通用计算机，尤其分别应用于 Intel 和 AMD 产品中。MIPS 和 ARM 在嵌入式系统、智能手机及其他便携式电子设备中至关重要 [48]。在实践中，我们排除了 ALLSTAR 数据集中仅支持一种 CPU 架构的软件包。此外，由于本地计算资源有限，我们剔除了过大、难以处理的软件包。我们使用*Ghidra* 反编译器和 Linux file 命令检查每个选定的二进制文件是否包含真实标注符号信息，并移除了不包含此类信息的文件。最后，我们构建了主要数据集，命名为*AS-4cpu-30k-bin*数据集，其中包含来自 1117 个软件包、涵盖 4 种 CPU 架构的 27572 个预构建二进制文件，如表 I 所示。

我们的初步实验发现，随机划分数据集时，评估过程存在数据泄漏问题。因此，我们在*AS-4cpu-30k-bin* 数据集上采用了非随机的变体训练集-测试集划分，比例为 4:1，选取约 80% 的二进制文件作为训练数据集，其余作为测试数据集。我们参考了 [23] 中的划分方法，旨在确保二进制文件

如果属于同一软件包，就会被划分到同一个集合中，即训练集或测试集。这种变体划分方法使我们能够真正评估*cfg2vec* 。

接下来，我们利用第三节-B 中提到的 GDT，将 *AS-4cpu-30k-bin* 数据集中的二进制文件转换为图中图 (GoG) 表示。值得注意的是，我们一次处理与同一个软件包相关的一批二进制文件，因为开发者可能会在同一软件包的不同模块中定义用户函数，同时将原型声明放在该软件包的主模块中。对于这种情况，*Ghidra* 确实会识别出两个函数实例：一个仅包含函数声明，另一个则包含实际的函数内容。由于这两个实例对应同一个函数名，且其中一个只包含虚拟指令，因此它们可能会给数据集引入噪声，进而影响模型的学习。为解决这一问题，我们的 GDT 还会从同一软件包的其他二进制文件中搜索函数体。如果找到，GDT 就会将该用户函数关联到包含实际内容数据的函数图节点。除了用户函数之外，二进制文件中还可能存在库函数调用；对于动态加载的二进制文件，在同一软件包中搜索这些函数的函数体则会失败。在这种情况下，*Ghidra* 会将这些函数识别为*ThunkFunctions*[3]，它们只包含一条虚拟指令。作为变通方法，我们从数据中移除了这些 *ThunkFunctions* ，因为它们可能会误导模型的学习。采用这一变通方法表明，我们的模型能够为用户函数和静态链接函数预测函数名。

表 I 实验中使用的数据集统计信息。

| 数据集 / 数量 | pkg/bin | 函数节点/边[1] | 基本块节点/边[2] |
|---|---|---|---|
| *AS-4cpu-30k-bin* | 1117/27,572 | 51.17/97.14 | 14.12/19.98 |
| *AS-3cpu-9k-bin* | 633/9,000 | 44.01/79.06 | 12.24/17.07 |
| *AS-i386-3k-bin* | 633/3 ,000 | 45.31/87.70 | 11.45/15.97 |
| *AS-amd64-3k-bin* | 633/3 ,000 | 42.28/74.07 | 12.28/17.18 |
| *AS-armel-3k-bin* | 633/3,000 | 44.45/75.41 | 13.00/18.07 |

[1] 每个二进制文件中的平均函数数和边数[2] 每个函数中的平均基本块数和边数

我们参照其实现[4]，使用自己的数据集进行了实验 [18] 。由于 [18] 在实验中使用了包含 3,000 个二进制文件的数据集，我们也采用了相同规模，准备了规模较小但相近的数据集。具体而言，我们从主要数据集 *AS-4cpu-30k-bin*中进行下采样，创建了 *AS-3cpu-9k-bin* 数据集，其中包含适用于 i386、amd64 和 armel CPU 架构的 9,000 个二进制文件。此外，由于 [18] 一次仅支持一种 CPU 架构，我们随后按不同 CPU 架构拆分*AS-3cpu-9k-bin* 数据集，生成了三个用于测试 [18] 的训练数据集：*AS-i386-3k-bin*、*AS-amd64-3k-bin*和 *AS-armel-3k-bin*。训练时，我们使用 Linux strip 命令将原始数据转换为三种形式：原始二进制文件（调试）、带调试信息的剥离符号二进制文件（已剥离）以及不带调试信息的剥离符号二进制文件（已剥离，无符号表），以符合 [18] 要求的数据格式。评估时，我们从主要数据集中为每种 CPU 架构抽取 100 个二进制文件，分别标记为 *AS-amd-100-bin*、*AS-i386-100-bin*、*AS-armel-100-bin*和*AS-mipsel-100-bin*。我们还有另一个名为 *AS-noMipsel-300-bin* 的评估数据集，其中包含大约 300 个为 amd64、i386 和 armel 平台生成的二进制文件。表 I 总结了所有这些数据集的统计信息，包括软件包数、二进制文件数、函数节点、边和基本块节点的平均数量。接下来的章节将详细介绍我们如何在实验中使用这些数据集。

### *B.* 评估：函数名称预测

表 II 展示了*cfg2vec* 在预测函数名称方面的结果。对于基线，我们采用了 [18] 的最佳设置，即寄存器或栈偏移的特征维度均为 100，并使用我们准备的数据集进行训练。对于*cfg2vec*，我们在两个图嵌入层中均使用了三个 GCN 层和一个 GAT 卷积层。评估时，我们计算 p@k（例如，前 k 项精确率）指标，该指标指预测函数名列表中前 k 项的平均命中率。具体而言，我们将 GoG 中表示的每个二进制文件输入训练好的模型，将每个函数转换为嵌入 $f \in F$ 并获取其函数嵌入 $h_f$。然后，我们计算 $h_f$ 与所有其他函数嵌入之间的成对余弦相似度，从中选出嵌入与 $h_f$最相似的 k 个名称，组成前 k 项列表。如果真实函数名位于函数名称预测的前 k 项列表中，则视为命中；否则视为未命中。在实验中，我们将 top-k 值设为 5，因此模型可以为二进制文件中的每个函数推荐最有可能的五个名称。

如表 II 所示，使用*AS-3cpu-9k-bin* 数据集训练的*cfg2vec*在推断函数名称时，预测准确率（例如，p@1）可达到 69.75%。对于 [18],，我们必须针对每种 CPU 架构分别训练其模型，因为它无法以跨架构方式进行训练。即便如此，对于 amd64 二进制文件， [18]的精确率也仅为 29.32%；而对于 i386 和 armel，其表现分别为 52.64% 和 53.65%。这一结果表明，无论如何，我们的*cfg2vec* 都优于[18]。此外， [18] 只能给出一个预测，而我们的*cfg2vec* 会提供五个选项，让用户（例如，逆向工程师）可以灵活地从预测概率最高的 k 个名称中选出他们认为最适合该函数的名称。表 II 中的 p@2 至 p@5 表明，我们的*cfg2vec* 能够为用户提供充分的函数名称线索。例如，使用我们的 *AS-3cpu-9k-bin* 数据集训练的*cfg2vec*，在所有 CPU 架构二进制文件上的 p@5 精确率可达到 70.50%。我们还使用更大的数据集对*cfg2vec* 进行了实验。从表 II 可以看出，*cfg2vec* 在正确预测函数名称（例如，p@1）方面可带来 5.04% 的性能提升。此外，使用*AS-4cpu-30k-bin* 数据集训练*cfg2vec* 时，性能提升可增至 28%。我们认为，在更大的数据集上训练意味着使用更多样化的二进制文件进行训练。这使我们的模型能够获取更多知识，从而能够

[3]ThunkFunction 手册：https://ghidra.re/ghidra docs/api/ghidra/program/model/listing/ThunkFunction.html [4]Debin 的 [18]代码仓库：https://github.com/eth-sri/debin

表 II cfg2vec*cfg2vec* 针对 [18] 的函数名称预测性能评估。

| 模型 | 训练数据集 | 测试数据集 | P@1[1] | P@2[1] | P@3[1] | P@4[1] | P@5[1] |
|---|---|---|---|---|---|---|---|
| *cfg2vec* | *AS-4cpu-30k-bin* | *AS-noMipsel-300-bin* | 97.05% | 99.47% | 99.47% | 99.47% | 99.47% |
| *cfg2vec* | *AS-4cpu-20k-bin* | *AS-noMipsel-300-bin* | 74.22% | 75.76% | 75.78% | 75.78% | 78.78% |
| *cfg2vec* | *AS-3cpu-9k-bin* | *AS-amd-100-bin* | 69.18% | 69.98% | 69.98% | 69.98% | 69.98% |
| | | *AS-i386-100-bin* | 69.41% | 70.39% | 70.39% | 70.39% | 70.39% |
| | | *AS-armel-100-bin* | 70.66% | 71.04% | 71.11% | 71.11% | 71.11% |
| | | *AS-noMipsel-300-bin* | 69.75% | 70.47% | 70.50% | 70.50% | 70.50% |
| [18]-amd64[2] | *AS-amd64-3k-bin* | *AS-amd-100-bin* | 29.32% | - | - | - | - |
| [18]-i386[2] | *AS-i386-3k-bin* | *AS-i386-100-bin* | 52.64% | - | - | - | - |
| [18]-armel[2] | *AS-armel-3k-bin* | *AS-armel-100-bin* | 53.65% | - | - | - | - |

[1] P@k 衡量实际函数名是否位于预测函数名的前 k 位中。[2] 这些模型仅提供排名第一的函数名称预测，因此只有 P@1 值。

提取出更稳健的二进制函数特征。总之，这一结果表明，与基线相比，我们的模型能够有效地为用户提供反编译代码中与上下文相关的函数名称。

表III *cfg2vec* 与其消融变体的比较。

| Arch | [18] | *GCN-GAT* | *2GCN* | *2GCN-GAT* | *cfg2vec* |
|---|---|---|---|---|---|
| amd64 | 29.32%[1] | 61.59% | 69.49% | 69.56% | 70.66% |
| armel | 52.64%[2] | 66.40% | 68.59% | 68.92% | 69.19% |
| mipsel | 53.65%[3] | 66.47% | 68.17% | 68.56% | 69.41% |
| 总体 | 45.20% | 64.82% | 68.75% | 69.01% | 69.75% |

[1] [18]-amd64 模型的评估结果。[2] [18]-i386 模型的评估结果。[3] [18]-armel 模型的评估结果。

我们还尝试了各种消融网络配置，以研究 *cfg2vec* 的各个组件对性能的贡献。首先，我们从原始实验配置中移除一个图卷积网络层，简化了*cfg2vec* 。如表III所示，我们将此配置称为*2GCN-GAT* ，其性能略微下降了 0.75%。随后，我们在*2GCN-GAT*配置的基础上进一步移除图注意力网络层，将其称为*2GCN*。我们再次观察到性能略有下降（<1%）。接着，我们从*2GCN-GAT*中再移除一个图卷积网络层，构建*GCN-GAT* 配置。对于*GCN-GAT*，我们观察到性能大幅下降（4.2%），这表明图卷积网络层的数量可能是影响性能的关键因素。具体而言，我们发现，图卷积网络层从 1 层增加到 2 层，可使预测准确率提升 4% 以上。然而，当图卷积网络层的数量增加到三层以上时，我们没有观察到显著的性能提升。因此，我们保留了包含三层图卷积网络层的原始*cfg2vec* 模型。总而言之，如表III所示，所有这些消融模型的表现仍优于 [18], , 这归功于我们为数据集中每个二进制文件构建的 GoG 表示。

### *C.* 评估：架构无关预测

表 IV 展示了我们的*cfg2vec*在跨架构支持方面的能力。由于 [18] 一次只能针对一种 CPU 架构进行训练，因此我们在实验中对其进行了多次训练。具体而言，我们使用三个数据集对 [18] 进行了训练：*AS-amd64-3k-bin*、*AS-i386-3k-bin*和 *AS-armel-3k-bin*，并将训练所得的模型分别命名为 [18]-amd、64, [18]-i386和 [18]-armel。对于这些基线模型，我们观察到，它们在使用相同 CPU 架构构建的二进制文件上测试时表现良好，但在使用不同 CPU 架构构建的二进制文件上表现较差。例如， [18]-amd64 对 amd64 二进制文件的准确率为 29.3%，但在 i386 和 armel 二进制文件上的表现较差（分别为 13.8% 和 7.1%）。类似地， [18]-i386 对 i386 二进制文件的准确率为 52.6%，但在 amd64 和 armel 二进制文件上的表现较差（分别为 6.2% 和 1.1%）。最后， [18]-armel 对 armel 二进制文件的准确率为 53.6%，但在 amd64 和 i386 二进制文件上的表现较差（分别为 11.8% 和 8.9%）。我们使用*cfg2vec* 生成的 top-1预测（即 p@1）作为比较指标，因为 [18] 每个函数只生成一个预测。从结果来看，*cfg2vec* 在测试的三种 CPU 架构上均优于 [18] 。*cfg2vec* 在所有 CPU 架构上都表现稳定，这表明我们的*cfg2vec* 支持跨架构预测。

为了评估模型泛化所学知识的能力，我们使用 *AS-mipsel-100-bin* 数据集测试了所有模型。该数据集包含基于另一种著名 CPU 架构 mipsel 构建的二进制文件，而我们的*cfg2vec* 此前并未针对该架构进行训练。对于 [18],，在测试此前未训练过的 CPU 架构生成的二进制文件时，其性能较低。例如，在使用*amd64* 二进制文件训练、使用 i386 二进制文件评估时， [18] 的准确率最高仅为 13.84%。如表 IV 所示，在我们的工作中，*cfg2vec* 使用 amd64、i386 和 armel 二进制文件训练，但在 mipsel 二进制文件上测试时，准确率达到 36.69%。对于 [18], , 它甚至不支持分析 mipsel 二进制文件。总之，这些结果表明，在跨架构二进制文件的函数名预测任务中，我们的*cfg2vec* 优于基线，并且能够更好地泛化到由未见过的 CPU 架构生成的二进制文件。为了进一步研究*cfg2vec*的跨架构性能，我们使用三个数据集对其进行训练，每个数据集均包含针对两种不同架构构建的二进制文件。随后，我们根据这些二进制文件所基于的架构为训练所得的模型命名：*cfg2vec*-armel-i386、*cfg2vec*- amd64-i386 和 *cfg2vec*-armel-amd64。这些结果表明，我们的模型在所有这些场景下都能很好地完成函数名称预测任务，包括在针对未知 CPU 架构编译的二进制文件上进行测试。

表 IV CFG2VEC 与 [18]的跨架构比较

| 模型 | 测试数据集 | P@1 |
|---|---|---|
| *cfg2vec*-3-架构 | *AS-amd-100-bin* | 69.18% |
| | *AS-i386-100-bin* | 70.66% |
| | *AS-armel-100-bin* | 69.41% |
| | *AS-mipsel-100-bin** | 36.69% |
| *cfg2vec*-amd64-armel | *AS-amd-100-bin* | 68.53% |
| | *AS-i386-100-bin** | 39.23% |
| | *AS-armel-100-bin* | 69.11% |
| | *AS-mipsel-100-bin** | 32.21% |
| *cfg2vec*-amd64-i386 | *AS-amd-100-bin* | 68.59% |
| | *AS-i386-100-bin* | 69.09% |
| | *AS-armel-100-bin** | 34.20% |
| | *AS-mipsel-100-bin** | 38.26% |
| *cfg2vec*-armel-i386 | *AS-amd-100-bin** | 42.96% |
| | *AS-i386-100-bin* | 67.45% |
| | *AS-armel-100-bin* | 63.86% |
| | *AS-mipsel-100-bin** | 36.61% |
| [18]-amd64 | *AS-amd-100-bin* | 29.32% |
| | *AS-i386-100-bin** | 13.84% |
| | *AS-armel-100-bin** | 7.08% |
| | *AS-mipsel-100-bin** | - |
| [18]-i386 | *AS-amd-100-bin** | 6.23% |
| | *AS-i386-100-bin* | 52.64% |
| | *AS-armel-100-bin** | 1.05% |
| | *AS-mipsel-100-bin** | - |
| [18]-armel | *AS-amd-100-bin** | 11.82% |
| | *AS-i386-100-bin** | 8.86% |
| | *AS-armel-100-bin* | 53.65% |
| | *AS-mipsel-100-bin** | - |

* 表示该数据集未用于训练。

### D. CFG2VEC 的实际应用

本节将展示*cfg2vec* 如何帮助逆向工程师应对国防高级研究计划局（DARPA）的*AssuredMicroPatching* （AMP）挑战赛二进制文件。AMP 计划旨在通过整合现有源代码样本、原始构建过程信息和历史软件制品，实现对旧式任务关键型系统二进制文件的快速打补丁，增强反编译能力，并引导其实现逆向工程师（RE）设定的特定目标。

*1) MINDSIGHT* 项目：我们的跨行业与学术界合作项目由西门子、JHU/APL、BAE 和 UCI 共同发起，旨在开发通过施加同态变换使反编译源代码可理解（MINDSIGHT）。我们的团队致力于构建与*Ghidra*集成的自动化工具链，旨在使反编译过程能够实现：(1) 以较粗粒度识别模块单元；(2) 准确重建符号名；(3) 将二进制文件提升为风格化 C 代码；(4) 采用系统且可扩展的方法分析代码相似性；以及 (5) 使用先进的嵌入式软件二进制数据集对新的反编译技术进行基准测试。迄今为止，我们团队开发了开源工具*CodeCut*[5]，，以提高*Ghidra*的模块识别准确率和完整性，并提供自动化的、基于脚本的反编译分析工具链，帮助逆向工程师进行专家解读。此外，我们还开发了同态变换语言（HTL），用于描述对抽象语法树（AST）语言的变换及其组合规则。我们的工具集成于*Ghidra*后，可让开发者在保持语义等价的同时，对反编译代码进行句法变换。其核心思想是使用 HTL 将*Ghidra* AST 转换为 GCC AST，从而将反编译得到的二进制文件提升为高级 C 表示。这个过程有助于逆向工程师理解二进制代码。*cfg2vec* 是 MINDSIGHT 项目开发的另一款工具，可重建函数名称，免去逆向工程师手动猜测的工作。

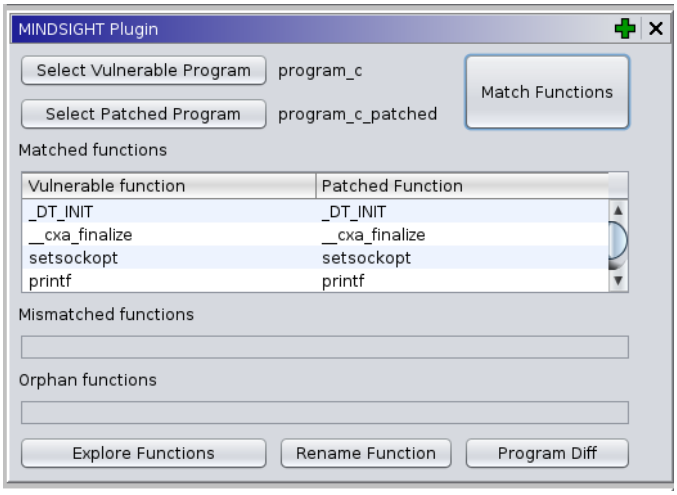



图 5. 集成到 Ghidra 中的插件截图。

*2) cfg2vec* 插件：在 *MINDSIGHT* 项目中，我们将 *cfg2vec* 集成到 *Ghidra* 反编译器中，作为一款插件应用。我们的 *cfg2vec* 插件通过为每个没有名称的函数提供一系列可能的函数名称，帮助逆向工程师理解二进制文件。具体而言，与所有 *Ghidra* 插件一样，我们的 *cfg2vec* 插件基于 Java 开发，其核心推理模块通过 Python 3.8 中的 REST API 实现。从 *Ghidra* 反编译器中提取剥离符号的二进制文件的元数据后，会将其发送至 *cfg2vec*端点，由该端点计算并返回所有函数的推断映射。图 5 展示了我们的 *cfg2vec* 插件的用户界面。在此场景中，用户必须提供易受攻击的二进制文件以及包含额外调试信息（例如函数名称）的参考二进制文件。点击“匹配函数”按钮即可触发 *cfg2vec* 功能，并在三个表格中显示函数映射结果：

- 匹配表：显示相似函数的映射关系。
- 不匹配表：显示不相似函数的映射关系，因此这些函数是打补丁的候选对象。
- 孤立表：显示置信度较低的函数的映射关系。

这些分组减轻了逆向工程师的工作量。他们无需检查所有函数，只需专注于为候选函数（不匹配的函数）和孤立函数打补丁。“浏览函数”按钮会打开 Ghidra 的函数浏览器，让用户并排比较这两个函数，如图 5 所示。该工具允许用户在 C语言和汇编语言之间切换，从而帮助确认或修改表格中的映射。关于

[5]*CodeCut* 的代码库：https://github.com/DARPAMINDSIGHT/CodeCut

*cfg2vec*的函数预测，“重命名函数”按钮会选取表格中当前选中的行，并将已修补的二进制文件中的函数名称赋予易受攻击的二进制文件中的对应函数。点击“匹配函数”按钮时，我们会为这两个程序（易受攻击的程序和已修补的程序）调用 FCG 和控制流图生成器。

*3）AMP* 挑战二进制文件的使用场景：DARPA AMP 挑战要求逆向工程师修补一个漏洞，该漏洞涉及一种弱加密算法：通信流量采用已弃用的密码套件进行加密，即三重数据加密标准或 3DES [49]。在这项挑战中，逆向工程师必须分析易受攻击的二进制文件，找出需要修补的函数和指令；本例中，需要将 *3DES* 密码套件相关的函数调用和指令替换为高级加密标准 [50]对应的内容。所有这些步骤都在反编译二进制层级完成；易受攻击的二进制文件经过编译器优化，并且已剥除调试信息和函数名称。此外，这些二进制文件有时会与 GNU C 库 [51]或 OpenSSL 等库静态链接，从而向二进制文件引入许多额外函数（其中有些函数永远不会被调用或使用）。面对这些复杂情况，逆向工程师要理清所有这些函数、找出问题并成功修补，并非易事。我们 *cfg2vec* 插件的直接用途，是让用户选取一个信息已剥除的目标函数，并查看其可能的函数名预测，或查看可用参考二进制文件中的匹配函数，以确认该函数是否处于逆向工程师解决问题时的关键路径上。如图 5 所示，我们的插件允许用户查看信息更完整的已修补（参考）二进制文件中的函数，与易受攻击且已剥除信息的二进制文件中的函数之间可能的匹配项。逆向工程师随后可以利用这些信息，为相应函数做好记录，从而更高效地完成工作。使用该工具的逆向工程师给我们的主要反馈是，他们希望拥有这样的功能。不过，该工具的准确率和易用性还不够高，尚无法充分发挥其潜力。

## 五、结论

本文提出了*cfg2vec*，一种基于层次图神经网络的软件逆向工程方法。基于*Ghidra*，我们的*cfg2vec* 插件可以提取二进制文件的图中图 （GoG）表示，将控制流图（CFG）和函数调用图（FCG）中的信息结合起来。*Cfg2vec* 利用分层图嵌入框架，学习编译为各种体系结构的二进制代码中每个函数的表示。最后，我们的*cfg2vec* 利用学到的函数嵌入进行函数名称预测，在所有测试二进制文件上的平均表现比最先进方法 [18] 高出24.54%。通过增加数据量，我们的模型性能提升了51.84%。虽然 [18] 需要针对每种 CPU 架构分别训练一次，但我们的*cfg2vec* 只需训练一次，便能在所有体系结构上持续取得更好的表现。此外，我们的模型能够更好地泛化所学知识 [18] 到由未经训练的 CPU 架构构建的二进制文件。最后，我们证明我们的*cfg2vec* 可以协助现实世界中的逆向工程师解决 *Darpa Assured MicroPatching*（AMP）挑战。


## 致谢

本材料基于国防高级研究计划局（DARPA）和太平洋海军信息战中心（NIWC Pacific）在合同编号 N66001-20-C-4024 下资助的工作。文中表达的观点、意见和/或研究结果仅代表作者的立场，不应被视为代表美国国防部或美国政府的官方观点或政策。


## 参考文献


[1] K. Zhidanov、S. Bezzateev、A. Afanasyeva、M. Sayfullin、S. Vanurin、Y. Bardinova 和 A. Ometov，“面向智能手机和受限物联网设备的区块链技术：未来展望与实现”，载于《2019年IEEE第21届商业信息学会议（CBI）》，第2卷。IEEE，2019年，第20–27页。[2] A. Panarello、N. Tapas、G. Merlino、F. Longo 和 A. Puliafito，“区块链与物联网集成：系统综述”，《Sensors》，第18卷，第8期，第2575页，2018年。[3] S. Dange 和 M. Chatterjee，“物联网僵尸网络：对物联网网络的最大威胁”，载于《数据通信与网络》。Springer，2020年，第137–157页。[4] S. R. Chhetri、J. Wan 和 M. A. Al Faruque，“信息物理系统的跨域安全”，载于《2017年第22届亚洲及南太平洋设计自动化会议（ASP-DAC）》。IEEE，2017年，第200–205页。[5] A. Sargolzaei、A. Abbaspour、M. A. Al Faruque、A. Salah Eddin 和 K. Yen，“网络化控制系统的安全挑战”，载于《可持续互依网络》。Springer，2018年，第77–95页。[6] A. Barua、L. Pan 和 M. A. A. Faruque，“Bayesimposter：基于贝叶斯估计的工业控制系统 BSS 冒充攻击”，载于《年度计算机安全应用会议》，2022年，第440–454页。[7] F-secure，“2019年上半年攻击态势”，2019年。[在线]。网址：https://blog-assets.f-secure.com/wp-content/uploads/2019/09/12093807/2019 attack landscape report.pdf [8] “CVE-2014-0160”。来源：MITRE，CVE-ID CVE-2014-0160，2013年12月3日。 [在线]。网址：http://cve.mitre.org/cgi-bin/cvename. cgi?name = 20T4 0160 CVE- - I. Ghafoor、I. Jattala、S. Durrani 和 C. [9] M. Tahir，“嵌入式系统的 OpenSSL Heartbleed漏洞分析”，载于《第17届IEEE国际多主题会议2014》。IEEE，2014年，第314–319页。[10] “Heartbleed漏洞”。2014年。 [在线]。网址：http: //heartbleed.com/[11] “Shellshock：关于 Bash漏洞，你需要知道的一切”，2014年。 [在线]。网址：https://www.symantec.com/ [12] “深入解析Shellshock：为何这个老漏洞仍挥之不去”，《Security Intelligence》，2020年。 [在线]。网址：https: //securityintelligence.com/articles/shellshock-vulnerability-in-depth/ [13] S.-H. Kim、M. A. Cohen、S. Netessine 和 S. Veeraraghavan，“为任务关键型系统的不频繁修复与恢复签订合约”，《Management Science》，第56卷，第9期，第1551–1567页，2010年。[14] N. S. Agency，“Ghidra——软件逆向工程框架”。2019年。[在线]。网址：https://www.nsa.gov/resources/everyone/ghidra/ [15] S. Hex-Rays，“IDA反汇编器”，2017年。[16] R. Team，《Radare2手册》。GitHub，2017年。[17] A. Keliris 和 M. Maniatakos，“ICSREF：工业控制系统二进制文件自动逆向工程框架”，arXiv预印本 arXiv:1812.03478，2018年。[18] J. He、P. Ivanov、P. Tsankov、V. Raychev 和 M. Vechev，“Debin：预测剥离符号的二进制文件中的调试信息”，载于《2018年ACM SIGSAC计算机与通信安全会议论文集》，2018年，第1667–1680页。[19] J. Lacomis、P. Yin、E. Schwartz、M. Allamanis、C. Le Goues、G. Neubig 和 B. Vasilescu，“Dire：一种为反编译标识符命名的神经网络方法”，载于《2019年第34届IEEE/ACM自动化软件工程国际会议（ASE）》。IEEE，2019年，第628–639页。[20] Y. David、U. Alon 和 E. Yahav，“利用增强型控制流图对剥离符号的二进制文件进行神经网络逆向工程”，《ACM程序设计语言汇刊》，第4卷，OOPSLA期，第1–28页，2020年。

[21] H. Gao、S. Cheng、Y. Xue 和 W. Zhang，“基于大规模剥离符号二进制文件的函数名称重分配轻量级框架”，载于《第30届ACM SIGSOFT国际软件测试与分析研讨会论文集》，2021年，第607–619页。[22] Q. Feng、R. Zhou、C. Xu、Y. Cheng、B. Testa 和 H. Yin，“固件映像的可扩展图搜索漏洞方法”，载于《2016年ACM SIGSAC计算机与通信安全会议论文集》，2016年，第480–491页。[23] X. Xu、C. Liu、Q. Feng、H. Yin、L. Song 和 D. Song，“ 基于神经网络的图嵌入：用于跨平台二进制代码相似性检测 ”，载于《2017年ACM SIGSAC计算机与通信安全会议论文集》，2017年，第363–376页。[24] I. U. Haq 和 J. Caballero，“二进制代码相似性综述”，arXiv预印本 arXiv:1909.11424，2019年。[25] K. Yakdan、S. Dechand、E. Gerhards-Padilla 和 M. Smith，“帮助Johnny分析恶意软件：面向易用性优化的反编译器及恶意软件分析用户研究”，载于《2016年IEEE安全与隐私研讨会（SP）》。IEEE，2016年，第158–177页。[26] L. ˇDurfina、J. Kˇroustek 和 P. Zemek，“Psybot恶意软件：逐步反编译案例研究”，载于《2013年第20届逆向工程工作会议（WCRE）》。IEEE，2013年，第449–456页。[27] M. J. Van Emmerik，《用于反编译的静态单赋值》。昆士兰大学，2007年。[28] D. Brumley、J. Lee、E. J. Schwartz 和 M. Woo，“利用保语义结构分析与迭代控制流构造进行原生x86反编译”，载于第22届 { USENIX} 安全研讨会（{USENIX} Security 13），2013年，第353–368页。[29] “Binutils。（2019）objdump”，2019年。 [在线]。网址：https://www. gnu.org/software/binutils/ [30] “Binary Ninja。（2022）交互式反汇编器、反编译器和二进制分析平台”，2022年。 [在线]。网址：https://binary.ninja/ [31] “Hopper。（2022）可用于反汇编、反编译和调试应用程序的逆向工程工具”，2022年。 [在线]。网址：https://www.hopperapp.com/ [32] “Hex-Rays。（2019）Hex-Rays反编译器”，2019年。 [在线]。网址：https://www.hex-rays.com/products/decompiler/ [33] M. Nyre-Yu、K. Butler 和 C. Bolstad，“面向安全领域静态二进制逆向工程的任务分析”，2022年1月。 [在线]。网址：https://scholarspace.manoa.hawaii.edu/handle/ 10125 79608 / W. Shao、[34] Q. Yang、X. Guo 和 R. Cai，“基于机器学习的二进制程序可用信息恢复综述”，载于《2022年第5届人工智能与大数据国际会议（ICAIBD）》。IEEE，2022年，第125–132页。[35] Z. L. Chua、S. Shen、P. Saxena 和 Z. Liang，“神经网络能够从二进制文件中学习函数类型签名”，载于《第26届USENIX安全会议论文集》，系列：SEC'17。美国：USENIX协会，2017年，第99–116页。[36] U. Alon、M. Zilberstein、O. Levy 和 E. Yahav，“ Code2vec：学习代码的分布式表示”，《ACM程序设计语言汇刊》，第3卷，POPL期，2019年1月。 [在线]。网址：https: //doi.org/ 10 1145 3290353 。/ F. Artuso、G. A. Di Luna、L. [37] Massarelli 和 L. Querzoni，“奉函数之名：利用神经网络为剥离符号的二进制文件中的函数命名”，arXiv预印本 arXiv:1912.07946，2019年。[38] K. Redmond、L. Luo 和 Q. Zeng，“受自然语言处理启发、用于二进制代码分析的跨架构指令嵌入模型”，arXiv预印本 arXiv:1812.09652，2018年。[39] D. Vasan、M. Alazab、S. Venkatraman、J. Akram 和 Z. Qin，“ Mthael：基于神经网络高级集成学习的跨架构物联网恶意软件检测”，《IEEE计算机汇刊》，第69卷，第11期，第1654–1667页，2020年。[40] M. Alhanahnah、Q. Lin、Q. Yan、N. Zhang 和 Z. Chen，“用于跨架构物联网恶意软件分类的高效特征签名生成”，载于《2018年IEEE通信与网络安全会议（CNS）》。IEEE，2018年，第1–9页。[41] P. Goyal 和 E. Ferrara，“图嵌入技术、应用与性能：综述”，CoRR，卷号 abs/1705.02801，2017年。 [ ] 1705 02801 在线。网址：[42] http://arxiv.org/abs/ 。Z. Wu、S. Pan、F. Chen、G. Long、C. Zhang 和 S. Y. Philip，“图神经网络综合综述”，《IEEE神经网络与学习系统汇刊》，2020年。[43] R. Yasaei、S.-Y. Yu 和 M. A. Al Faruque，“Gnn4tj：用于寄存器传输级硬件木马检测的图神经网络”，载于《2021年欧洲设计、自动化与测试会议及展览（*DATE*）。IEEE，2021年，第1504–1509页。[44] S.-Y. Yu、A. V. Malawade、D. Muthirayan、P. P. Khargonekar 和 M. A. Al Faruque，《场景图增强的数据驱动风险评估：自动驾驶车辆决策》，*IEEE*智能交通系统汇刊，2021年。[45] M. Hassen 和 P. K. Chan，《可扩展的基于函数调用图的恶意软件分类》，载于第七届*ACM*数据与应用安全和隐私会议论文集，2017年，第239–248页。[46] S. Harada、H. Akita、M. Tsubaki、Y. Baba、I. Takigawa、Y. Yamanishi 和 H. Kashima，《用于预测化学网络的双图卷积神经网络》，*BMC*生物信息学，第21卷，第1–13页，2020年。[47] J. Staff，《静态分析研究组装标注库（allstar）数据集》，2019年12月。 [在线]。地址：https://allstar.jhuapl.edu/ [48] 《Debian GNU/Linux安装指南》，2004年。 [在线]。地址：https://www.debian.org/releases/bullseye/i 386 02 01 186 /ch s . [49] en.html#idm J. Callas，《三重数据加密标准：数据加密标准有多强？》，2017年5月。 [在线]。地址：https://www.techtarget.com/searchsecurity/tip/Expert-advice-Encryption-101-Triple-DES-explained [50] C. Bernstein 和 M. Cobb，《什么是高级加密标准（ AES ）？ 来自 SearchSecurity 的定义》，2021年9月。 [在线]。地址：https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard [51] T. Rothwell 和 J. Youngman，《GNU C参考手册》，自由软件基金会，第86页，2007年。