## 第一节介绍

恶意软件的分析是计算机安全的一个根本问题。它提供了对恶意软件的功能和功能的必要详细了解，从而形成有效的对策和缓解策略的基础。由专业和技术精湛的对手创建，现代恶意软件越来越复杂和复杂。高级恶意软件家族如Stuxnet [26]，Uroburos [29]和Regin [50]都是当前恶意软件复杂程度和复杂程度的例证。这些恶意软件家族显示了恶意软件作者隐藏其活动的非凡长度，并使恶意软件的手动逆向工程变得更加困难和困难。

由于程序无法识别另一个程序的非平凡属性，自动恶意软件分析的通用问题是不可判定的[44]。由于这种限制，安全研究集中在自动分析特定类型的功能，例如加密功能的识别[13]，自动协议逆向工程[12]，[53]以及基于DGA的恶意软件的检测[2]。作为这种限制的另一个结果，安全分析人员经常需要采取手动逆向工程来对恶意软件进行详细和彻底的分析，这是一个困难和耗时的过程。作为补救措施，安全研究人员已经开始探索在分析过程中帮助分析师而不是替代它们的方法。所提出的方法通过正确识别二进制文件[4]，[47]中的功能来加速分析过程，可靠地提取二进制代码[7]，[35]，[37]，[42]，[58]，将模糊化的可执行代码[ 20]，[56]，并通过反编译恢复二进制代码的高级抽象[45]，[57]。

解码提供了一种有吸引力的方法来帮助恶意软件分析，使分析能够以二进制代码的高级，抽象形式进行。在高层次上，反编译包括一系列*抽象恢复*机制，以恢复二进制代码中不容易获得的高级抽象。然后可以对反编译程序代码执行手动和自动分析，从而减少所需的时间和精力。为实现这一目标，研究界已经解决了恢复源代码重建所需的高级抽象的原则性方法。这包括用于恢复数据类型的方法[39] ，[40] ，[48]和高级控制流结构，如二进制代码中的if-then-else结构和while循环[45]，[57]。虽然已经取得了重大进展，但是最先进的反编译器仍然创建非常复杂的代码，而不是专注于可读性。

在本文中，我们认为以人为中心的方法可以显着提高反编译器的有效性。为此，我们提出了几种语义保留代码转换来简化反编译代码。改进的可读性使得反编译代码更容易理解，从而可以加速恶意软件的手动逆向工程。我们的方法的关键观点是，可以利用以前的反编译阶段恢复的抽象来设计强大的优化。驱动这些优化的主要直觉是，反编译的代码更容易理解，如果它可以以类似于程序员编写的方式形成。基于这种直觉，我们设计优化来简化表达式和控制流结构，消除冗余，并根据代码中使用的方式为变量提供有意义的名称。另外，我们开发一个通用的查询和转换引擎，允许分析人员轻松地编写代码查询和转换。我们已经在最先进的学术反编译器Dream [57]之上实现了我们的可用性扩展。我们称之为扩展版本Dream ++。

虽然在改进反编译器方面做了大量工作，但是对这些方法的评估从未包括用户研究，以验证优化是否真正帮助真正的恶意软件分析人员。Cifuentes等人的开创性工作[16]和许多后续的作品[15]，[17]，[24]，[27]，[45]，[57]都基于一些机器可测量的可读性来评估反编译器质量诸如反编译代码中的goto语句数量或与输入程序集相比反编译代码小得多的度量。此外，以前的大量工作对少数，小型，有时自写作的手动定性评估，

在本文中，我们介绍了恶意软件分析的第一个用户研究。我们与21名学生进行了一次研究，他们完成了恶意软件分析课程和9个专业恶意软件分析师。该研究包括从独立恶意软件专家获得的真实恶意软件样本的6个逆向工程任务。我们的研究结果表明，我们改进的反编译器Dream ++产生了显着更易于理解的代码，并且表现优于领先的行业和学术反编译器：Hex-Rays [30]和Dream。使用Dream ++参与者解决了3× 更多的任务比使用Hex-Rays和 2×更多的任务比使用梦想。专家和学生们对Dream ++的评价显着高于竞争对手。

总而言之，我们做出以下贡献：

* *反编译器的可用性扩展*。我们提出几种语义保留代码转换，以简化和提高反编译代码的可读性。我们的优化利用了以前的反编译阶段恢复的高级抽象。我们已经将我们的技术实现为最先进的学术反编译器Dream [57]的扩展。我们称之为扩展反编译器Dream ++。
* *评估反编译器质量的新可用性度量*。我们建议将*人为因素*纳入一个指标，以评估反编译器对恶意软件分析的有用性。尽管以前的工作已经提出了几种衡量反编译器质量的定量指标，但令人惊讶的是，人为因素尚未得到研究。鉴于手工逆向工程是反编译研究背后的主要驱动因素，这是一个严肃的监督，我们希望我们的方法也将成为未来研究的基准。
* *用户学习评估*。我们进行第*一次*用户研究，以评估反编译器的恶意软件分析的质量和有用性。我们与受过恶意软件分析培训的学生以及专业的恶意软件分析师进行研究。结果提供了统计学上显着的证据，Dream ++在成功分析的任务量方面优于领先的行业反编译器Hex-Rays和原始Dream反编译器。

## 第二节问题陈述和概述

本文的重点是提高反编译器创建的代码的可读性，以加速恶意软件的分析。代码可读性对于人们正确理解代码的功能至关重要[10]。我们对恶意软件分析师进行了几次非正式访谈，以确定对可读性产生负面影响的最先进的反编译器的缺点。我们还进行了认知演练，逐步通过重组Hex-Rays和Dream生成的恶意软件代码，了解这两个反编译器的问题。我们将发现的问题分为三类

**1）复数表达式**

最先进的反应堆通常产生过于复杂的表达。这样的表达式很少在人类编写的源代码中找到，因此很难理解。这包括

**a）复杂逻辑表达式**

控制结构内部使用逻辑表达式（例如，if-then或while循环）来决定要执行的下一个代码。复杂的逻辑表达式使得很难理解代码中执行的检查以及基于它们的决策。

**b）变量数量**

解码代码通常包含太多变量。这使得分析复杂化，因为必须跟踪大量的变量。虽然反编译器应用了死代码消除步骤，但仍然错失了删除冗余变量的机会。在许多情况下，可以将多个变量合并为单个变量，同时保留代码的语义。

**c）指针表达式**

数组访问操作通常作为涉及指针算术和转换操作的引用表达式来恢复。而且，使用address-of操作符（例如，\*（＆v + i））将对堆栈上分配的数组的访问恢复为表达式。

我们在第三节介绍我们处理这些问题的方法。

**2）卷积控制流程**

程序的可读性在很大程度上取决于其排序控制的简单性[23]。两个问题往往使反编译器回收的控制流程复杂化

**a）重复/内联代码**

二进制代码通常包含重复的代码块。这通常是由编译期间的宏扩展和函数内联引起的。因此，分析师可能会多次分析相同的代码块。

[](http://ieeexplore.ieee.org/mediastore/IEEE/content/media/7528194/7546461/7546501/7546501-fig-1-large.gif)

**图。1：**

摘自simda恶意软件系列的域生成算法的反编译代码。此示例显示生成域名的主循环。在高级别上，从两个数组中随机挑出字母。选择复制字母的数组是基于循环计数器是偶数还是奇数。

**b）复杂环路结构**

反编译器使用的控制流结构化算法通过分析控制流图来识别循环。因此，它们恢复了编译器生成的结构，该结构针对效率进行了优化，但不可读性。在这个阶段停止，防止反编译器从人类编写的源代码中看到的循环更可读的形式。

我们在第五节解决这些问题。我们优化的核心是我们在第四节中描述的代码查询和转换框架。

**3）缺乏高级语义学**

高级语义（如变量名称）在编译过程中丢失，无法由反编译器恢复。因此，反编译器通常会将默认名称分配给变量。此外，一些常量在给定的上下文中具有特殊的含义，例如，由API函数使用或作为文件类型的魔术数字。在第六节中，我们描述了几种基于代码中如何使用变量和常量有意义的名称的技术。

作为说明这些问题的示例，我们考虑如图1所示的代码。该图显示了由三个反编译器生成的Simda恶意软件系列的域生成算法（DGA）的反编译代码：Hex-Rays（图1a），Dream（图1b）和我们改进的反编译器Dream ++（图1c）。在这里，由于空间限制，我们只显示计算域2的主循环。如片段所示，Hex-Rays和Dream生成的代码相当复杂，难以理解。在Hex-Rays生成的代码中，循环变量i从不在循环中使用，循环以break语句结束。此外，对循环计数器的奇偶校验的恢复检查涉及复杂的低级表达式（行26-30）。访问char数组（v37和v30）使用指针运算，运算符地址和取消引用运算符。

Dream产生了一个稍微更可读的代码，但仍然有一些问题。这里，恢复的环路结构不是最佳的并且可以进一步简化。由于v18的初始值为零，所以if语句和所附的do-while循环的条件在第一次迭代时是相同的。这样可以将整个构造转换成更可读的循环。

最后，在本文中开发的优化进一步降低了代码的复杂性。从图1c可以看出，代码包含一个简单的for循环，具有清除初始化步骤，条件和增量步骤。对于每个循环迭代，根据循环计数器的奇偶校验（i％2 == 0），从两个字符数组（v1和v2）中选择一个字母，并将结果存储在输出数组（v3）中。

范围。Dream ++基于Dream反编译器，它使用IDA Pro [33]来反汇编二进制代码并构建二进制中的函数的控制流图。可以说，由此产​​生的反汇编并不完美，如果二进制文件被故意混淆，可能会包含错误。对于本文的范围，我们假定提供给反编译器的程序集是正确的。如果二进制代码被模糊化，可以使用[7]，[37]，[58]等工具来提取二进制代码。此外，最近的方法如[20]，[56]可以用于在将二进制代码提供给反编译器的输入之前对其进行解混淆。

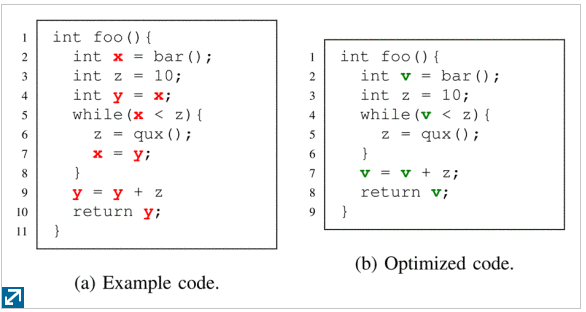
我们的方法的高级概述如下。首先，使用Dream对二进制文件进行反编译。该阶段对每个功能进行反编译，并生成相应的控制流图（CFG）和抽象语法树（AST）。AST中的每个节点表示DREAM的中间表示（IR）中的语句或表达式。我们的工作从这里开始 我们开发三类语义 - 保留代码转换，以简化代码并增加可读性。这些类别是*表达式简化，控制流简化*和*语义感知命名*。在以下部分中，我们详细讨论我们的优化。

## 第三节表达简化

在本节中，我们介绍我们的优化来简化表达式，并从反编译代码中删除冗余。

**A.一致性分析**

一致性分析是我们从反编译代码中删除冗余变量的方法。关键概念是识别代表相同值的变量，并且可以在保留语义的同时用单个代表变量代替。我们将这些变量表示为*一致变量*。Dream已经执行了几个优化来消除诸如表达传播和死代码消除之类的冗余。然而，存在传统的死代码消除算法不能去除冗余代码的场景。一个突出的例子是编译器发出指令以临时保存一些稍后恢复进一步使用的值。根据控制结构，这可能导致反编译器IR中的相应变量之间的循环依赖，防止去除死代码。作为说明这些情况的示例，我们考虑图2a所示的代码示例。在这个例子中，第4行和第7行在变量x和y之间复制一个值。另外，用单个代表（如变量v）替换x和y并不会改变程序的语义。此外，这种替换会导致两个可以安全删除的v = v（行4和7）的简单的复制语句，从而产生更紧凑和可读的代码，如图2b所示。不改变程序的语义。此外，这种替换会导致两个可以安全删除的v = v（行4和7）的简单的复制语句，从而产生更紧凑和可读的代码， 如图2b所示。不改变程序的语义。此外，这种替换会导致两个可以安全删除的v = v（行4和7）的简单的复制语句，从而产生更紧凑和可读的代码， 如图2b所示。



**图。2：**

一致性分析

这个简单的例子说明了代码在变量一致性表征中的不同属性。总之，需要涵盖以下几个方面。

**1）相同类型**

一致变量具有相同的数据类型。这个要求是必要的，以避免由于隐式类型转换而改变语义。例如，如果转换不会保留语义y 是类型短。

**2）非干扰定义**

用单个代表替换全等变量不会改变达到使用这些变量的程序点的定义。请注意，这并不要求一致变量的生存范围不会干扰。例如，第7行的x的定义位于y的生存范围内，即在y（行4）的定义和对应的y（行9）的使用之间。然而，定义是一个简单的复制语句x = y，因此使用第9行的x或y中的任何一个保留语义。

**3）一致显示语句**

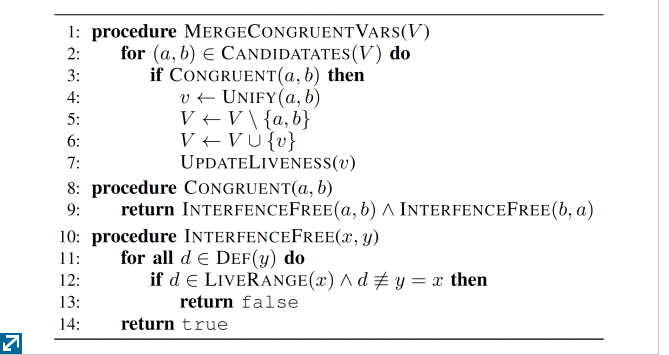
以前的检查足以保证统一变量的语义保留属性。然而，无限制地将其应用于所有变量可能会对可读性产生负面影响。相同类型的两个非干扰变量可能具有不同的语义（例如，一个整数变量用作循环计数器，第二个整数用作缓冲区的大小）。不合并这些变量使我们能够根据变量在代码中的使用方式给出每个变量的代表性名称。基于这一点，我们将同余分析限制在代码中包含用于相同目的的指示的变量中。也就是说，我们只检查与x = y形式的复制语句相关的变量，我们将其表示为*一致性显示*语句。

这些检查的核心是关于变量活动的信息。为此，我们执行定点*实时变量*分析，编译器设计中的标准问题[41]，p。443.在高水平上，活动变量分析确定程序中每个点处哪些变量是*活*的。一个变量v 在程序中的特定点生活 p∈P 如果 p 位于执行路径上的定义 v 和使用 v 这不包含重新定义 v。该组程序点构成变量的*生存范围*。

LIVERANGE(v)={p∈P:v live at p}

查看来源

**算法1一致性分析**

[](http://ieeexplore.ieee.org/mediastore/IEEE/content/media/7528194/7546461/7546501/7546501-alg-1-large.gif)

算法1通过首先计算候选变量对的集合来实现这个想法，即，在一致揭示语句中涉及的相同类型的变量，然后检查这些对的一致性。对于每个候选对(x,y)，算法检查它们是否没有干扰定义。特别是程序Interferencefree检查每个变量的定义y 不在于生活范围 x，或者是表单的副本 y=x。同样的检查也在定义中进行x。当确定两个一致变量时，程序Unify 1）选择一个代表变量v; 2）代表所有出现的并发变量; 和3）删除由此统一（形式v = v）导致的平凡副本。接下来，这组变量V已更新。最后，新添加的变量的活动信息更新如下：

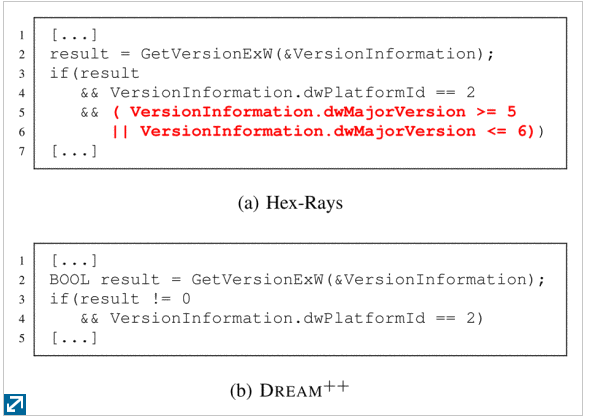
LIVERANGE(v)=LIVERANGE(x)∪LIVERANGE(y)

查看来源

不是我们不要求所有程序点的一致变量必须具有相同的值。它们的活动范围不会干扰的点可能具有不同的值。例如，虽然变量x和y的不同值在图2a所示的代码中达到return语句，但是x不在第9行和第9行。这使我们能够为x和y使用相同的变量。

**条件简化**

此步骤的目标是在反编译代码中找到最简单的高级逻辑表达式。这些表达式对于理解程序的控制流程非常重要，因为它们在控制语句中使用，例如if-then-else语句或while循环，以决定接下来要执行的代码。简化逻辑表达式有两个方面的帮助：第一，它有助于将语义上等效的高级条件恢复到编译器发出的低级检查。第二，它有助于清除由原始代码中的错误引起的任何误解。



**图。3：**

摘录自stuxnet示例的反编译代码。代码检查Windows操作系统的版本。

**低级检查**

在编译期间，编译器使用一个名为*tiling*的转换将高级程序语句减少到汇编语句中。因此，每个高级语句都可以转换成一系列语义上等效的汇编指令。在此过程中，高级谓词被转换为可以有效执行的语义上等效的低级检查。作为一个例子，我们考虑中所示的代码图1a。第30行分配的右侧是检查变量v19是偶数还是奇数的复数表达式。这不像源代码中使用的常用操作，但它相当于计算v19％2 = 0的高级操作。

**守则中的错误**

200新新新新200新新新200新新新200新200新新200新200新新新200新200新新200新新200新200新新200新新200新新200新新200新新200新新200新新200新新200新新新200新色新款200 恶意软件分析师假设他们分析的代码执行一些有意义的任务，他们需要找出。他们也知道，恶意软件通常使用几个技巧来隐藏其真正的功能。有了这种观念，当分析师观察到一个看似无用的代码时，他们需要进行双重检查，以排除旨在使代码看起来没用的技巧的可能性。结果，有一段时间是浪费的。图3a所示的Stuxnet恶意软件系列的简单示例说明了这种情况。此代码检查Windows操作系统的版本，这是*环境目标恶意软件中* 的常见过程[55]。新新新旗新旗新新旗新新旗旗新新旗新新旗新新旗新新旗新新旗新新旗新新旗新新旗200新新新旗新新200新新新新旗新新200新新旗新新200 任何整数大于5或小于6.最可能的是，恶意软件作者打算使用AND表达式，但没有出于某种原因。简化此表达式导致如图3b所示的代码。

为了提供通用的简化方法，我们将我们的技术基于Z3定理证明器[21]。我们的做法如下。首先，我们将Dream IR中的逻辑表达式转化为Z3定理证明器的语义等效符号表达式。为了实现忠实的表示，我们将变量建模为固定大小的位向量，具体取决于它们的类型。位向量理论允许对无符号和有符号双补数运算的精确语义进行建模。在这个转换过程中，我们保持每个符号变量和原始逻辑表达式中它所代表的相应变量之间的映射。其次，我们使用定理证明来简化和规范化符号表达式。最后，我们使用映射来构建Dream IR中的逻辑表达式的简化版本。

**C.指针转换**

使用指针算术通过指针取消引用访问值可能会令人困惑。另外，访问堆栈上分配的缓冲区可能会导致难以理解的卷积反编译代码。

**指针到阵列转换**

在这里，我们使用观察结果，在C中，指针可以像数组名称一样进行索引。该表示清楚地将指针变量与用于从起始地址计算偏移量的表达式分开。为了保证此转换的语义保留属性，我们搜索代码中一致访问的指针类型的变量。也就是说，使用指针变量读取或写入的所有数据具有相同的类型τ。在这种情况下，取消引用这些变量可以表示为具有类型元素的数组τ。这里所得到的偏移表达式必须根据类型的大小进行调整τ。例如，如果一个指针p 一直用于访问4字节的整数，则表达式（如\*（p + 4 \* i））可以转换为更可读的p [i]形式。

**参考指针转换**

在这一步中，我们将仅与*地址 - 运算符*（＆）结合使用的变量转换为指针变量。Dream中的第一步之一是*可变恢复*，可从二进制代码恢复各个变量。例如，函数通常在堆栈上分配一个空间来存储局部变量。然后将访问此堆栈帧中的值的表达式作为局部变量进行恢复。为了提高效率，当编译时已知最大大小时，缓冲区通常会分配给堆栈。在这种情况下，变量恢复步骤表示缓冲区作为局部变量v，访问此缓冲区内的项目的表达式使用运算符地址＆v表示，导致难以理解的反编译代码。例如，从堆栈上分配的缓冲区中读取一个字符表示为\*（＆v37 + dwSeed / v15％6）（ 图1a中的第31行）。如果只使用地址表达式（即＆v）在代码中访问变量v，则使用指针变量v\_ptr替换这些表达式。这创造了进一步简化指针解引用表达式的机会，其中生成的指针变量涉及数组索引。前面的例子可以表示为v37\_ptr [dwSeed / v15％6]。这创造了进一步简化指针解引用表达式的机会，其中生成的指针变量涉及数组索引。前面的例子可以表示为v37\_ptr [dwSeed / v15％6]。这创造了进一步简化指针解引用表达式的机会，其中生成的指针变量涉及数组索引。前面的例子可以表示为v37\_ptr [dwSeed / v15％6]。

## 第四节代码查询和转换

我们随后的优化的核心是我们通用的方法来搜索代码模式并应用相应的代码转换。我们的方法背后的主要思想是利用逻辑编程的推理能力来搜索反编译输出中的模式。为此，我们将反编译代码表示为描述相应抽象语法树的属性的逻辑事实。这种基于逻辑的表示使我们能够优化将搜索模式建模为逻辑规则，并通过代码库高效地执行复杂查询。可用性是一个关键的设计目标，因此我们使系统的用户能够使用普通的C代码来定义搜索规则，并提供一个规则编译器来将它们编译成我们引擎所需的逻辑规则。我们使用平台无关，免费的SWI-Prolog实现[52]。在下文中，我们将详细描述我们的方法。

**A.梦想红外的基于逻辑的表示**

此步骤作为Dream生成的抽象语法树（AST）作为输入，并输出相应的逻辑事实，表示为*s代码事实*。我们将每个AST节点表示为在AST中描述其属性和嵌套顺序的代码事实。表I显示了DREAM中间表示（IR）3中所选语句和表达式的代码事实。谓词符号（事实名称）表示AST节点类型。第一个参数是相应节点的唯一标识符。第二个参数是父节点的唯一标识符（例如，包含if语句）。节点ID和父ID代表反编译代码的分层句法结构。

我们通过遍历输入AST生成代码事实，并为每个被访问节点产生相应的代码事实。代码事实存储在事实基础中F，这将在搜索代码模式时被查询。作为说明代码事实概念的简单示例，我们考虑图4a所示的代码示例。功能体的相应代码事实如图4c所示。body是两个语句的序列（*id* = 3）：一个if-then-else语句（*id* = 4）和一个return语句（*id* = 14）。这两个语句具有序列节点作为其父节点，并且序列中的顺序由序列码事实内相应ID的顺序表示。

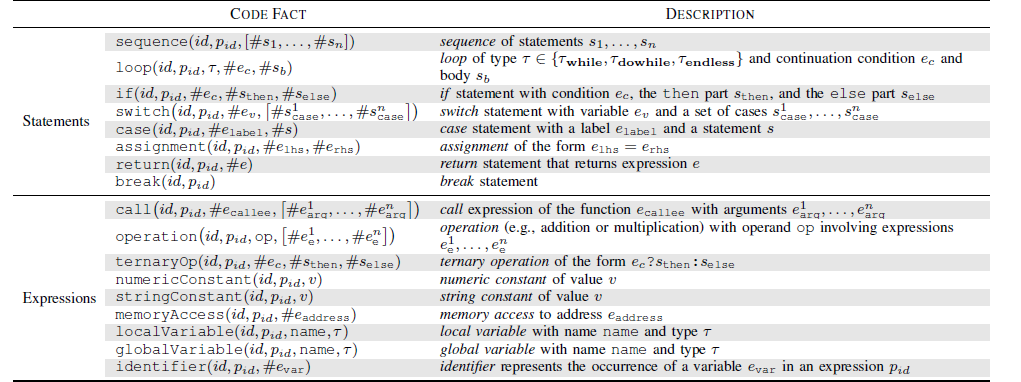
**B.转型规则**

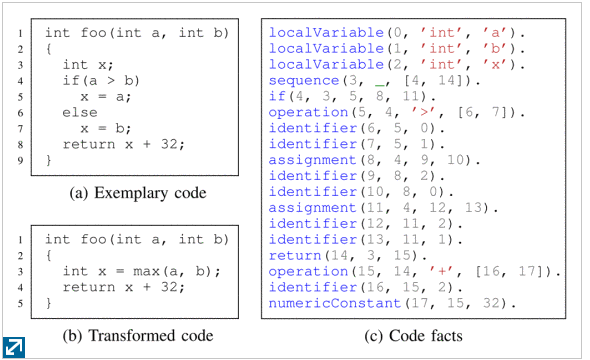
基于逻辑的代码表示使我们能够优雅地将搜索模式建模为形式的*推理规则*

P1P2…PnC

查看来源推理规则栏的顶部包含前提 P1,P2,…,Pn。如果所有房屋都满足，那么我们可以在酒吧下方的结论C。场景描述了我们搜索的代码模式的属性。在代码查询的情况下，结论是简单地指示搜索模式的存在。对于代码转换，结论表示识别的代码模式的转换形式。

**表I：** Dream IR的基于逻辑的谓词。每个谓词都有一个id来唯一地表示相应的语句或表达式。每个代码事实的第二个参数是父id pid，表示包含AST节点的id。对于语句或表达式，我们用#e表示的idee

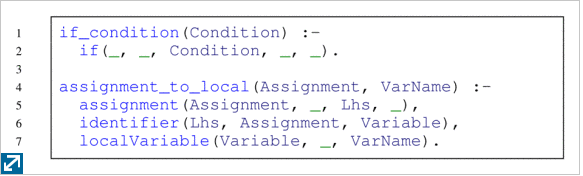
[](http://ieeexplore.ieee.org/mediastore/IEEE/content/media/7528194/7546461/7546501/7546501-table-1-large.gif)

[](http://ieeexplore.ieee.org/mediastore/IEEE/content/media/7528194/7546461/7546501/7546501-fig-4-large.gif)

**图。4：**

代码表示。

我们认识到推理规则是Prolog规则，这使我们能够询问关于代码事实代表的程序的Prolog查询。图5显示了两个简单的示例，说明了将代码搜索模式建模为Prolog规则的想法。规则if\_condition搜索if语句中使用的条件表达式。规则参数是表示从匹配模式中提取的信息片段的Prolog变量。规则机构代表必须履行的处所，以便规则返回匹配。在高级别，当执行查询时，Prolog尝试找到一个令人满意的赋值给规则的变量，使其与事实保持一致。例如，在图4c中的事实基础上执行的查询if\_condition（Condition） 返回匹配{Condition = 5}，与图4a中的if语句的条件相对应的代码事实的id 。这个统一是通过匹配规则的前提与if语句的相应代码事实来完成的。

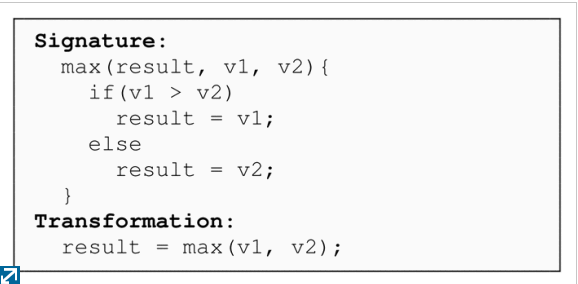
[](http://ieeexplore.ieee.org/mediastore/IEEE/content/media/7528194/7546461/7546501/7546501-fig-5-large.gif)

**图。5：**

样品搜索模式。

逻辑规则的一个非常强大的方面是相应的查询可以用于多个目的。例如，第二个规则assignment\_to\_local搜索给定其名称的局部变量的赋值。使用具体的变量名称，查询返回对应变量的所有赋值（例如，assignment\_to\_local（Assignment，'x'））。另一方面，对于名称使用Prolog变量，查询会将所有赋值返回给所有变量（例如，assignment\_to\_local（Assignment，Name））。

转换规则可以用正常的C代码编写。图6显示了一个示例转换规则，用于搜索计算两个值中最大值的if语句，并通过调用max库函数来替换它们。转换规则由*规则签名*和*代码转换*两部分组成。规则签名描述了要搜索的代码模式，并将其作为正常的C函数声明写入：表示为*规则参数*的参数列表表示需要通过Prolog推理引擎匹配实际变量的变量，以便转换代码可以构建。函数体表示代码模式。

[](http://ieeexplore.ieee.org/mediastore/IEEE/content/media/7528194/7546461/7546501/7546501-fig-6-large.gif)

**图。6：**

样本转换规则。

我们将转换规则编译成可由Prolog的推理引擎使用的逻辑规则。为此，我们解析规则体，然后遍历生成的AST。对于每个访问的AST节点，我们生成相应的代码事实。在这里，我们使用Prolog变量来生成事实标识符。然后，当推理引擎找到匹配项时，这些变量将被绑定到事实库中的实际标识符。最后，编译规则存储在规则库中R 并在查询库中对应查询 Q。

**C.应用转型**

我们首先用代码库初始化Prolog F 和规则库 R。然后，我们迭代地在查询集中应用查询Q。如果找到匹配项，则推理引擎*将*规则参数*统一*到相应代码事实的标识符。在这种情况下，我们构造等效的转换代码。为此，我们首先解析变换字符串来构造相应的AST。在此过程中，我们对每个规则参数使用相应的AST节点，根据初始代码库的原始变量获得变换后的代码。例如，将图6中的样本规则应用到图4c所示的事实基础中，返回一个匹配：{result = x，v1 = a，v2 = b}。这使我们能够通过函数调用x = max（a，b）来替换完整的if语句来获取图4b所示的代码。最后，我们更新事实基础F 使其与AST保持一致。

代码查询和转换引擎是我们后续的代码优化的基础，它们标识某些代码模式并应用相应的转换，旨在简化代码并提高可读性。

## 第五节控制流简化

在本节中，我们介绍我们的技术来简化反编译代码的控制流程。

**循环变换**

编译器优化通常会改变源代码中循环的结构。虽然这种优化旨在提高效率，但是由此产生的循环结构变得不太可读，从而降低了反编译代码的质量。一个众所周知的循环优化是*反转*，它将循环改变为一个循环中的一个循环执行的循环中的标准循环，该循环包含在一个if条件中，在执行循环的情况下将跳转次数减少2。也就是说，形式的循环（e）{...}被转换为if（e）{do {...} whi le（e）;}。这样做会重复条件检查（增加代码的大小），但效率更高，因为跳转通常会导致流水线停顿。另外，如果初始条件在编译时已知为真，并且无副作用，则可以跳过if guard。

这里我们观察到，while循环比do-while循环更可读，因为从一开始就清楚连续条件。此外，一些while循环可以进一步简化为for循环，其中初始化语句，连续条件和增量语句从一开始就是清楚的。基于此观察，我们分析了do-while循环，并检查它们是否可以转换为while循环。在这里，我们区分情况：

**守卫的Do-While循环**

如果（c1）{do {...} while（c2）;}的形式的循环转换为while（c2）{...}，如果可以证明 c1==c2在循环的第一次迭代开始时。注意c1 和 c2不必是相同的逻辑表达式。例如，我们考虑图7a所示的代码示例。条件\*（\_ BYTE \*）v7！= 0和\*（\_ BYTE \*）（v8 + v7）！= 0都在循环条目下产生相同的布尔值。请注意，此时变量v8的到达定义为v8 = 0。

**无保护的Do-While循环**

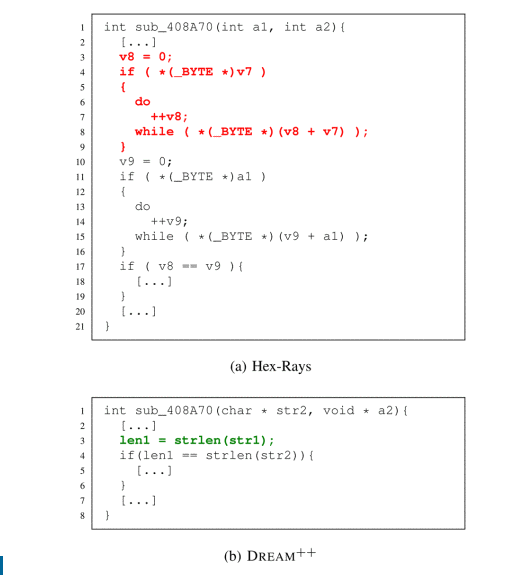
对于这些循环，我们只检查循环条件是否为第一次迭代。在这种情况下，循环可以转换为while循环。

要在循环条目中检查逻辑表达式的值，我们计算到达循环条目的循环变量的定义集合。为此，我们执行一个固定点进程内*到达定义*分析，编译器设计中的一个标准问题[41]，p。通常，循环变量的达成定义是表示循环计数器的初始值的常量值的赋值。这使得很容易在逻辑表达式中替换这个初始值，并检查循环条目的等价性。

**功能概述**

功能内联是众所周知的编译器优化，其中所有对某些功能的调用都被替换为功能代码的就地副本。这改善了运行时性能，因为完全消除了调用和从函数返回的开销。在代码混淆的背景下，内联是一种强大的技术[19]。它通过两种方式使逆向工程变得更加困难：首先，相同代码的几个重复项遍布该程序。因此，分析人员最终分析了相同代码的几个副本。第二，内部抽象，如程序中的函数之间的调用关系被消除。

反向功能内联对于恶意软件的手动分析很有价值。作为一个简单的例子说明了功能概述的好处，我们考虑了图7所示的Cridex恶意软件系列的摘录代码。图7a所示的十六进制反码代码中的两个循环中的每个循环通过将每个字符的计数器递增1来计算字符串的长度，直到找到终止空字符。Dream ++将这两个块标识为strlen库函数的实现，并用相应的函数调用替换它们，如图7b所示。这个简单的例子可以深入了解功能概述代码分析的好处。



**图。7：**

摘录自cridex恶意软件家族的代码，显示代码内联技术。

1. *紧凑的代码*。通过等效函数调用替换代码块可以消除重复的代码块，从而产生更紧凑的反编译输出。整个代码块被名称直接显示代码块的功能的函数调用所替代。此外，块内使用的临时变量从代码中删除，减少了分析人员应该跟踪的变量数。
2. *有意义的变量名*。概述功能有已知的接口，包括其参数的名称。这些名称代表其语义，并显示关于变量作业的重要信息。我们利用这些信息给出有意义的名称反编译输出中的变量。
3. *改进的类型恢复*。从二进制代码恢复类型的方法，如[39]，[40]依靠*类型接收器*作为可靠的起点。类型接收器是程序中指定变量的类型已知的点。这包括对其签名已知功能的调用。概述函数生成一个新的类型接收器，可以用来提高类型推理算法的性能。
4. *恢复相互依赖关系*。函数概述隐含地恢复内联函数和调用它的函数之间的调用关系。也就是说，它识别程序中调用该函数的点。呼叫关系对于手动逆向工程非常重要。在分析给定功能后，恶意软件分析人员可以得出关于调用功能的结论。

原始的Dream反编译器包含硬编码的签名，用于最小的字符串函数集，这些签名很难扩展。我们利用我们的代码查询和转换引擎轻松地为复制，比较，计算长度和初始化缓冲区的几个函数包含多个转换规则。例如，我们处理strcpy，strlen，strcmp，memset。对于字符串函数，处理8位和16位字符版本。我们还包括使用缓冲区长度作为参数的字符串函数的版本的签名。

我们系统的用户可以轻松添加新的转换规则来处理新功能。当分析师观察到重复的代码模式时，她可以简单地编写一个转换规则，通过一个名称代表它的功能的函数调用代替整个代码块。将概述同一块的所有其他副本。代码块不仅作为函数内联的结果复制。在C中，*类似函数的宏*是接受参数的预处理器宏，与普通函数调用一样使用。这些宏由预处理器处理，因此保证内联。

## 第六节语义感知命名

在本节中，我们介绍了反编译代码中变量级别的可读性改进。

**有意义的名字**

变量名称在分析源代码时起着重要的作用。这些名称揭示了有关变量目的的有用信息，以及如何在程序中使用变量。我们根据它们发生的上下文给变量有意义的名称。这里我们区分以下情况：

**标准库呼叫**

使用定义良好的API，标准库调用是变量名称的重要来源。例如，从Internet下载数据并将其保存到文件的Windows API URLDownloadToFile需要五个参数。其中一个名为szURL的参数表示要下载的URL。名为szFileNarne的第二个参数表示要为下载创建的文件的名称或完整路径。通过分析库函数调用和返回，我们重命名用作参数或返回值的变量，直接向分析人员揭示其目的。

**基于上下文的命名**

代码中使用变量的方式可以深入了解其目的。我们分析使用变量为其提供有意义的名称的上下文。更具体地说，我们区分以下情况：

1. *循环计数器*。我们查询用于*计数循环*的反编译代码，即更新其体内变量的循环，然后在其连续条件中测试相同的变量。将循环中的计数变量重新命名为i，j或k。将其他循环的计数变量重命名为计数器。
2. *数组索引*。我们重新命名用作索引索引的数组的变量
3. *布尔变量*。包含评估逻辑表达式结果的变量将重命名为cond。这表示它们代表测试变量名中的条件。

当识别出可以使用相同名称的多个变量时，我们将下标添加到默认名称以具有唯一的名称。例如，如果三个循环计数器被识别，它们被重命名为counter1，counter2和counter3。

**B.命名常数**

常数是逆向工程过程中的重要角色。例如，一些加密算法使用魔术数字，并且几种文件格式包括用于识别文件类型的魔术数字。标准库函数也将特定常量赋予特殊含义。通常这些数字在源代码中具有文本表示形式。我们使用两个来源来识别这些特殊常数。

**库API常量**

C标准库和Windows API中的许多函数定义了特殊的命名常量。这些常数具有特定的含义，因此具有代表性的名称。在编译期间，编译器将常量的符号表示替换为相应的数值。例如，函数CreateFile使用常量GENERIC\_READ来请求对打开的文件的读取访问。这在二进制中变为0×80000000。为了恢复符号，容易记住这些常量的名称，我们检查各种库函数的命名常量的出现。例如，这将使函数调用CreateFileA（f，0×80000000，1，...）转换为更易读的形式CreateFileA（f，GENERIC\_READ，FILE\_SHARE\_READ，...）。

**文件魔数**

对于许多文件类型，文件以短字节序列（大多数为2到4个字节长）开始，以唯一标识其类型。在文件中检测这些常量是区分许多文件格式的简单而有效的方式。例如，DOS MZ可执行文件格式及其后代（包括NE和PE）以两个字节4D 5A（字符'MZ'）开头。恶意软件通常在运行时从C＆C服务器下载文件，并可以检查其收到的文件类型。我们检查这些常量是否在流量控制语句的条件下使用。

## 第七节用户研究设计

我们研究的目标是测试我们改进的反编译器Dream ++生成的代码与学术前身Dream和行业标准Hex-Rays 4的可读性。我们计划进行用户研究，参与者必须用不同的反编译器解决许多逆向工程任务。参与者的任务是分析代码片段并回答关于代码功能的许多问题。我们的基于网络的学习平台在分屏显示代码以及问题。参与者可以编辑代码来帮助分析。

为了衡量用户感知，在每个任务之后，我们询问参与者的反馈意见和关于可读性的几个问题。每个参与者得到了不同反编译器生成的一些代码片段，而不被告知使用哪个反编译器，所以我们将对代码进行不偏不倚的评估。只有在研究结束时，我们向用户展示了所有反编译器并行生成的代码，并要求他们给出反编译器的总体评级。

**A.任务选择**

为了最大限度地减少偏见的风险，即潜意识地选择有利于我们的反编译器的任务，我们接触了三名独立的专业恶意软件分析人员进行任务选择。我们知道分析师，但没有参与研究或反编译器的工作。我们告诉他们，我们想进行恶意软件分析研究，并要求他们为我们提供他们自己在工作中必须分析的恶意软件代码片段。我们要求代码段执行某种功能，应该是可以理解的，而不需要其余的恶意软件代码。总共有8个代码片段。其中两个片段包含基于XOR的加密/解密算法，因此我们删除了其中的一个，并留下了7个恶意软件代码段。我们进行了一项预研，将在下面进行更详细的描述，以测试任务。对于这个预研究，偶数任务是可取的，所以我们添加了一个附加的代码段。根据预研究的结果，我们删除了这个和另外一个片段，所以我们剩下6个片段。*我们将这些片段分成两组： 中*（三个任务）和*硬*（三个任务）。在下文中，我们将详细介绍任务。五*和硬*（三个任务）。在下文中，我们将详细介绍任务。五*和硬*（三个任务）。在下文中，我们将详细介绍任务。五

**1）加密**

编码功能广泛应用于恶意软件以及良性应用。恶意软件可以使用C＆C服务器加密交换的消息，并编码内部字符串以避免静态分析。此任务是从Stuxnet恶意软件中解析包含Stuxnet主DLL的.stub部分的函数。

**2）自定义编码**

此任务是来自Stuxnet恶意软件系列的XOR加密/解密功能。该函数使用0xAE12执行字义XOR，并被许多Stuxnet组件用于伪装一些字符串。

**3）动态解决API**

为避免静态分析，恶意软件通常避免在导入表中列出所需的API功能。相反，它可以在运行时动态解决它们。该任务是Cridex恶意软件的一个功能，它将API函数的名称作为输入，并返回相应的起始地址。

**4）字符串解析**

恶意软件经常从C＆C服务器接收命令和配置文件。因此，需要解析这些命令以从C＆C消息中提取参数和其他信息。此任务是从URLZone恶意软件注入解析函数。该函数检查序列％[A-Z0-9]％的第一次出现的字符串，并返回一个指向该字符串开头的指针及其长度。

**5）下载并执行**

一个非常常见的功能是从C＆C服务器下载可执行文件，然后执行它。例如，每次安装服务的情况就是这样的[11]。该任务涉及分析仙女座恶意软件的更新机制。该片段从远程服务器下载文件，并检查它是否是有效的PE可执行文件或包含可执行文件的Zip存档。在这种情况下，文件保存在磁盘上并执行。

**6）域生成算法**

恶意软件通常配备有域生成算法（DGA）动态生成用于C＆C域名（例如，取决于种子值，例如*作为*当前的日期/时间和Twitter趋势）[2] 。这是一种强大的技术，可以使僵尸网络对攻击和删除尝试更有弹性。该任务包含Simda恶意软件的DGA。

由于空间限制，我们无法在本文中提供我们研究中使用的完整的反编译恶意软件源代码。新新[200 200 200 200 200 200](https://net.cs.uni-bonn.de/fileadmin/ag/martini/Staff/yakdan/sm-oakland-2016.tar.gz) 200：[200EC- .gz](https://net.cs.uni-bonn.de/fileadmin/ag/martini/Staff/yakdan/sm-oakland-2016.tar.gz" \t "_blank)。在这里，我们提供与我们的用户研究相关的所有数据，包括三个反编译器的反编译代码以及一组任务。

**B.预研**

200的X- 200 X- 200 X- 200 X- 200 200 X- 200 X- 200 200 200 200 200 200：200 200 X- 我们还想检查我们的认知演练和非正式采访是否错过了在进行全面研究之前应该处理的任何重要问题。为了规划研究和适当的补偿，我们需要估计每个任务需要多长时间。

由于恶意软件分析是一项非常复杂的任务，需要专门的技能，因此我们招募了成功完成我们的恶意软件新兵训练营的学生。恶意软件启动训练营是我们大学每个学期举办的实验课程，学生将被引入到恶意软件和二进制代码分析领域。对于预先学习，我们招募了其中两名学生。

我们在可用性实验室进行了预先学习，并使用了一个思路协议，要求参与者在执行任务时发出他们的思维过程和感受。我们选择了这个协议来了解用户的思想过程，以及他们所面临的障碍和问题。在预研究中，我们只测试了DREAM ++和Hex-Rays。第一个参与者使用Dream ++反编译了四个任务，并用Hex-Rays四个反编译，并且必须回答关于代码功能的问题。第二个参与者得到了反向选择。作业随机化。在每个任务之后，要求参与者提供有关他们分析的代码质量的反馈。然后，

**1）预研结果**

研究平台只有小错误，参与者无任何问题地理解了任务描述。根据他们的思考反馈，我们没有发现在认知演练期间没有发现任何公开的问题。我们还根据报告的难度对我们的任务进行排名

在主要研究中，我们希望另外测试Dream意味着更少数量的任务样本，因为我们从不会为同一任务显示来自不同反编译器的任何参与者代码。根据预研究，我们估计主要研究应在3小时内完成。对于详细的预研究结果，包括参与者的反馈，我们参照附录E。

**C.方法论**

我们进行了一个学科内设计实验，参与者必须分析三种不同反编译器（DREAM ++，Dream和Hex-Rays）反编译的代码段。首先，我们向参与者详细说明了概念和程序。参与者被允许在研究期间使用互联网，因为这反映了分析师实际工作的方式。目标是删除与反编译代码质量无关的所有方面。该教程之后是一个培训阶段，以确保参与者熟悉系统以避免系统相关的错误。我们提供了一个示例代码片段，并指示参与者重新命名其中的变量，并在线查找库函数的信息。

主要的用户研究几乎与预测不变。方法在以下几点上有所不同。反编译器的名称是盲目的，以免偏见参与者。另外，这项研究没有在实验室进行，而是通过我们的在线学习平台进行。这个决定是出于以下几个原因：首先，并不是所有完成恶意软件新兵训练的学生都在本地生活，我们希望最大限度地发挥我们的招聘池，因为这个级别的参与者很少。第二，我们也想与专业恶意软件分析师进行研究，期望他们来到实验室是不切实际的。我们决定在线进行整个研究，以保持结果的可比性。

**1）变量和条件**

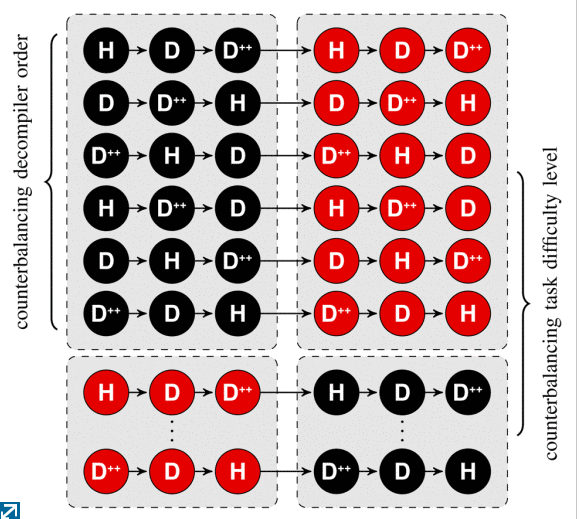
在我们的实验中，我们有两个独立变量：

**闪客**

解码器用于解决给定的任务，并有三个条件：Dream ++，Dream和Hex-Rays。Hex-Rays是恶意软件分析师广泛使用的领先的工业反编译器。因此，我们将Dream ++与Hex-Rays进行比较，以检查我们的方法是否改进了恶意软件分析的当前状态。我们测试了最新的Hex-Rays版本，这篇文章是2.2.0.150413。此外，我们将Dream ++与原始的Dream反编译器进行比较，以评估我们的新扩展的有用性。

**困难**

代表任务难度的受试者因素。根据我们的预测（§VII-B）的结果，我们将其难度分为两组（*中*，*硬*），每组包含三个任务。



**图。8：**

平衡反编译器的顺序和难度级别。每个水平序列中的节点表示一个参与者执行的任务。字母表示任务使用的反编译器，颜色代表任务难度级别：中等（黑色）或硬（红色）。

**2）条件分配**

我们选择了一个学科内设计，因为个人技能是表现的一个强大因素。为了避免在我们的课外设计中学习和疲劳的影响，参与者在每个难度级别内使用反编译器的顺序，以及使用平衡措施设计来排列难度级别。图8显示了我们的平衡设计的细节：在每个难度级别中，订购三个反编译器有六种可能性。两个难度级别也被排列（图中红色与黑色）。平衡难度级别可能排序总数的两倍。我们选择了平衡难度级别而不是任务级别，因为这给了我们一个平衡排列12（3！\* 2！）而不是4320（3！\* 6！）。由于我们不希望招募4320名参与者，我们选择了使用我们柜台所有行的招募12位参与者的倍数。该设计确保每个反编译器和每个难度级别在整个研究中获得相同的曝光，并最大限度地减少整体学习和疲劳效应。这也保证每个参与者获得与每个反编译器相同数量的中等和硬任务。这对于控制参与者之间的个体差异是重要的，并且避免倾斜结果，消除熟练和有动力的参与者使用一个反编译器执行所有任务的可能性，而较不熟练的参与者使用另一个反编译器执行其任务。该设计确保每个反编译器和每个难度级别在整个研究中获得相同的曝光，并最大限度地减少整体学习和疲劳效应。这也保证每个参与者获得与每个反编译器相同数量的中等和硬任务。这对于控制参与者之间的个体差异是重要的，并且避免倾斜结果，消除熟练和有动力的参与者使用一个反编译器执行所有任务的可能性，而较不熟练的参与者使用另一个反编译器执行其任务。该设计确保每个反编译器和每个难度级别在整个研究中获得相同的曝光，并最大限度地减少整体学习和疲劳效应。这也保证每个参与者获得与每个反编译器相同数量的中等和硬任务。这对于控制参与者之间的个体差异是重要的，并且避免倾斜结果，消除熟练和有动力的参与者使用一个反编译器执行所有任务的可能性，而较不熟练的参与者使用另一个反编译器执行其任务。

**3）用户感知**

在完成每项任务后，参与者将会看到一个简短的问卷调查表，在那里他们可以评估反编译器生成的代码的质量，以及一个文本字段以获得其他反馈。在这里，参与者可以再次看到代码。我们共问了8个问题，6个关于可读性的问题，2个问题的信任问题。类似于系统可用性分数（SUS）[8]，问题是平衡的（正/负）以最小化响应偏倚，例如“ *这个代码很容易读取* ”和“ *了解这段代码做了什么是艰巨的* ”。完整的问题集可以在附录B中找到。

除了每个任务之后的调查问卷，在研究结束时，我们向参与者询问了从1（最差）到10（最佳）的总体评级。在此步骤中，它们显示了每个任务和每个反编译器的代码片段，以便于直接比较。为了避免偏见参与者，反编译器被命名为M，P和R.

**4）统计检验**

对于所有的统计假设检验，我们选择了共同的意义水平 α=0.05。为了考虑多次测试，所有p值都在Holm-Bonferroni修正版本[32]中报告。

使用Holm-Bonferroni校正的Mann-Whitney U测试（双尾）测试连续测试，如时间间隔或用户评分。我们只是使用反编译器（DREAM ++ vs. Dream and Dream ++ vs. Hex-Rays）来测试所有的成对比较对。效果大小通过平均比较和*通用语言效果大小*方法的使用来报告。使用双尾Holm-Bonferroni校正的巴纳德精确测试来测试分类应急比较。

## 第八节用户研究

在本节中，我们将介绍我们的研究成果：在讨论了参与者的人口统计特征后，我们进行代码分析实验，然后进行用户感知讨论。

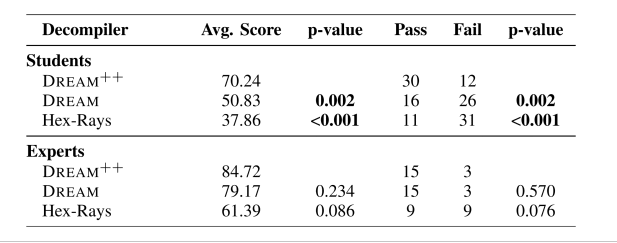
**A.参与者**

我们向我们大学完成了恶意软件新手训练的学生发送了36封邀请函，目的是让24名参加者加入40欧元的赔偿。恶意软件启动训练营是我们大学每个学期举办的实验课程，学生将被介绍到恶意软件和二进制代码分析领域。邀请函可以在附录A中找到。

22名学生参加了这项研究。一名学生开始学习，但只完成了一项任务，所以我们从样本中删除了这名学生，让我们有21名学员。在学生组中，中位年龄为26岁。最年长的参加者是31岁，最小的是19岁。两名参与者未报告年龄。其中一名参加者是女性。多年来恶意软件分析中位数的中位数为1年。一名参与者报告说有14年的恶意软件分析经验，7名参与者报告不到一年。

我们还邀请了来自商业安全公司的31名恶意软件专家。基于非正式谈判，我们被告知，为这些专家提供相同的赔偿不会激励他们，因为时间比金钱更有价值。然而，有人建议，许多人将有内在的动力来帮助，因为在这个领域的任何改进将最终使他们受益。根据这一反馈，我们选择为参与者提供早期访问*学术反编译器*，并在学习之后让他们访问Dream ++。17名恶意软件分析师开始研究。然而，8只看到第一个任务，而没有实际开始研究。共有9名恶意软件分析师参与了该研究，所有男性中位年龄为30岁（2名未披露年龄和性别）。

**表二：**综合实验结果

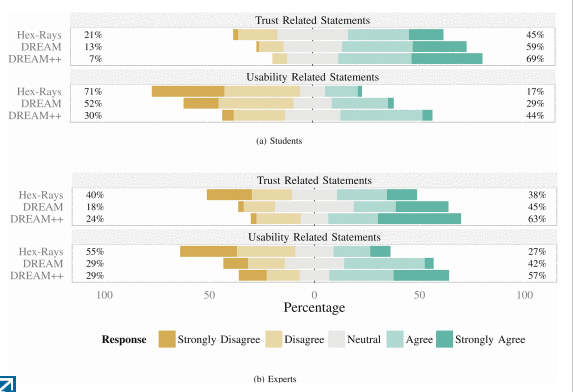
[](http://ieeexplore.ieee.org/mediastore/IEEE/content/media/7528194/7546461/7546501/7546501-table-2-large.gif)

**B.恶意软件分析实验**

我们根据其在理解任务中的重要性对所有问题分配权重，并对所有答案进行了评分。我们进行了一个双程评分方法的校准目的。由于答案包含变量名称并被称为循环结构，因此不可能将这部分评估视为盲目。表二总结了代码分析实验的结果。不出所料，专业分析师表现优于学生。在两组中，与Dream和Hex-Rays相比，使用Dream ++时，与会者表现更好。在学生组中，我们的样本量提供了足够的统计学功能，以确认分数具有统计学显着差异。另一个有趣的观察是，专家们比Hex-Rays比学生做得更好，表明他们已习惯于使用Hex-Rays生成的代码。附录C提供了更详细的表格。

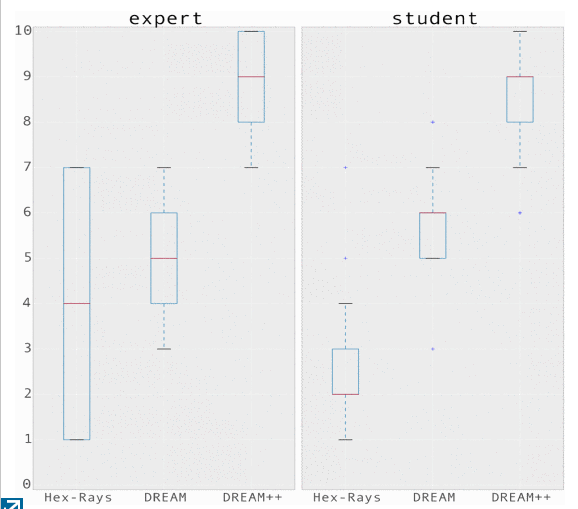
为了获得更好的结果感觉，如果参与者得分超过70％，我们还会将每个任务标示为通行证。这个级别是根据我们的判断来选择的，即这些任务得到了充分的回应，这意味着他们的结果将在恶意软件分析团队中有用。在这里，专业分析师和学生之间的差异变得更加明显。然而，在两组中，Dream ++表现优于竞争对手。

我们还测量完成不同任务所需的时间。由于样本数量有限，我们无法对时间进行统计分析。请注意，只有成功完成的任务才能在该比较中考虑。不幸的是，许多参与者使用Hex-Rays和Dream失败或放弃了任务。尽管如此，参与者通过Dream ++更快地解决任务的趋势，但需要更多的样本才能可靠地量化。任务级别的详细概述，包括平均花费在任务上的时间，可以在附录C中找到。



**图。9：**

与可用性感知相关的陈述（6个问题）和对正确性的信任（2个问题）的聚合参与者协议。



**图。10：**

用于反编译器等级的Boxplot。

**用户感知**

为了衡量用户的感知力，在每个任务之后，我们向参与者询问了8个问题。完整的问题清单见附录三。图9总结了用户感知结果的聚合。在这里，我们区分了两组问题：可用性感知（问题1-6）和信任感知（问题7和8）。信任问题很有趣，因为反编译器确实犯错误或产生误导性的代码，恶意软件分析师常常回到分析汇编代码。因此，我们想看看代码中是否有不同的信任级别。

反编译器之间的成对比较显示出高的统计学显着性差异 (p<0.001)针对专家和学生的可用性相关问题。与Hex-Rays相比，信任相关问题仅显示出统计学上的显着性差异(p<0.001 在学生组中， p=0.03在专家组）。图9中的颜色编码显示协议级别，右侧和左侧的百分比数字总结了分别评价为正和负的参与者的百分比。

完成所有任务后，我们要求每个反编译器的总体评级。图10显示了具有评级分布概述的箱形图。可以清楚地看到，Dream ++比学生和专家的竞争得分高（调整后）p<0.001对于所有成对比较）。有趣的是，虽然可以清楚地看到，专家们认为Hex-Rays比学生更好，他们对Dream ++也更加热情。这些都是非常有希望的结果，两个组都明确地喜欢我们改进的反编译器生成的代码，即使专家们十六进制，他们给出了梦想++杰出的标记。

**第九节**

相关工作

对二进制代码恢复高层次抽象的反编译方法和开发原理方法进行了大量研究。在高水平上，本文提出的工作有四条相关的研究。首先，从可执行文件中提取二进制代码的方法。第二，关于恢复源代码重建所需的抽象的研究。第三，端对端反编译器的工作。最后，查询代码库和应用转换的技术。

**二进制代码提取**

反编译的基本步骤是正确提取二进制代码。Kruegel等 [37]提出了一种拆卸x86混淆代码的方法。Kinder等 [35]提出了一种通过数据流分析交织多个拆卸轮来实现准确和完整的CFG提取的方法。二进制分析平台BitBlaze [49]及其后续BAP [9]使用值集分析（VSA）[3]来解决间接跳转。运行时封装程序经常被恶意软件编写者用来混淆代码并阻碍静态分析[51]。为了处理这些情况，研究界提出了依靠动态分析来处理严重混淆的方法。这包括提取完整的CFG的方法[42]，从混淆二进制文件中提取二进制代码[58]，去混淆的模糊化可执行代码[20]，[56]。一个密切相关的话题是识别二进制代码中的函数。最近，安全研究开始探索基于机器学习的方法来解决这个问题。Byteweight [4]使用加权前缀树学习功能开始的签名，并通过将二进制片段与签名相匹配来识别功能。Shin等 [47]使用神经网络。解模糊化的可执行代码[20]，[56]。一个密切相关的话题是识别二进制代码中的函数。最近，安全研究开始探索基于机器学习的方法来解决这个问题。Byteweight [4]使用加权前缀树学习功能开始的签名，并通过将二进制片段与签名相匹配来识别功能。Shin等 [47]使用神经网络。解模糊化的可执行代码[20]，[56]。一个密切相关的话题是识别二进制代码中的函数。最近，安全研究开始探索基于机器学习的方法来解决这个问题。Byteweight [4]使用加权前缀树学习功能开始的签名，并通过将二进制片段与签名相匹配来识别功能。Shin等 [47]使用神经网络。并通过将二进制片段与签名相匹配来识别功能。Shin等 [47]使用神经网络。并通过将二进制片段与签名相匹配来识别功能。Shin等 [47]使用神经网络。

**摘要从二进制代码恢复**

源代码重建需要恢复两种类型的抽象：数据类型抽象和控制流抽象。以前的工作涉及到从二进制代码中恢复这些抽象的原理方法。最近的工作提出了静态和动态方法来恢复两个标量类型（例如，整数或短整数）和聚合类型（例如，数组和结构体）。突出的例子包括REWARDS [40]，Howard [48]，TIE [39]和MemPick [31]。其他工作[22]，[27]，[28]，[34]侧重于C ++的具体问题，例如恢复C ++对象，重建类层次结构以及解决虚拟继承产生的间接调用。

控制结构恢复的早期工作依赖于间隔分析[1]，[18]，其将CFG解构为称为间隔的嵌套区域。Sharir [46]随后将间隔分析改进为结构分析。结构分析通过将CFG中的区域与预定义的一组模式或区域模式相匹配来恢复高级控制结构。恩格尔等人 [25]扩展结构分析以处理C特定的控制语句。他们提出了一个单一输入单继承器（SESS）分析，作为结构分析的扩展，以处理在断点之前存在的语句的情况，并在循环体中继续语句。

最近在控制流程结构恢复方面取得了重大进展。施瓦茨等 [45]提出了对香草结构分析的两个改进：首先，*迭代精化*选择一个边，并在算法不能进一步进行时使用goto语句表示。这允许算法找到更多的结构。其次，*语义保留*确保正确的控制结构恢复。亚克丹等 人。[57]提出了与*模式无关的*控制流结构，一种依赖于高级控制结构语义而不是对应流程图形状的方法。

**代码查询和转换**

已经提出了基于一阶谓词逻辑的几种代码查询技术。它们主要用于检测设计模式或有问题的设计模式的软件工程。这些技术支持特定的源语言，他们引入新的语言来建模代码查询，如CrocoPat [5]，[6]和Soul [54]，或者这些工具的用户必须直接写入逻辑规则，如JTransformer [36]。我们的代码查询和转换引擎基于Dream IR，使恶意软件分析人员能够将转换规则直接写入正常的C代码。

**反编译**

Cifuentes为现代反兴奋剂奠定了基础。在博士论文[16]中，她介绍了几种用于反编译二进制代码的技术，涵盖了数据流分析和控制流分析的各种技术。这些技术在dcc中实现，一个用于Intel 80286 / DOS的反编译器.Cifuentes等人 还开发了*asm2c，一个* SPARC组件到C反编译器，并用它来反编译整数SEPC95程序[17]。

Van Emmerik提出在博士论文[24]中使用静态单一赋值（SSA）形式进行反编译。他的工作表明，SSA可以有效地实现许多反编译器组件，如表达传播，死码消除和类型分析。他的技术在开源的Boomerang反编译器中实现。虽然比间隔分析更快，但是它的结构恢复较少。另一个开源反编译器[14]是基于van Emmerik的工作。

Chang等 [15]创建了一个模块化框架，用于构建合作反编译器的管道。由一系列由中间语言连接的反编译器执行解压缩。他们的工作表明可以在反编译的源代码中使用源代码级工具来查找原始C代码中已知存在的错误。

Hex-Rays是*事实上的*行业标准反编译器[30]。Hex-Rays由Ilfak Guilfanov开发，并作为Interactive Disassembler Pro（IDA）的插件构建。由于它是封闭源，对于所使用的确切方法知之甚少。它使用增强版本的香草结构分析，并有一个引擎来识别几个内联函数。还有其他的反编译器可以在线获得，如DISC [38]和REC [43]。然而，我们的经验表明，所有以前提到的反编译器并不像Hex-Rays那样先进​​。

凤凰是由Schwartz等人创建的高级学术反编译器。[45]。它建立在二进制分析平台（BAP）[9]之上，它将顺序的x86汇编指令升级到BIL中间语言。它还使用TIE [39]从二进制代码恢复类型。Phoenix使用增强的结构分析算法，可以正确地恢复比香草结构分析更多的结构。Schwartz等 是第一个衡量反编译器整体的正确性的。他们的方法依赖于检查反编译代码是否可以通过为源代码编写的自动检查。

梦想是Yakdan等人开发的最新学术反编译器。[57]。Dream使用IDA来提取二进制代码并构建CFG。类型恢复基于TIE。它使用一种新颖的控制流结构化算法来恢复高层次的控制结构，而不依赖于模式。Dream是第一个生成无转换反编译输出的反编译器。假设这使Dream成为当前生成最可读代码的反编译器。因此，我们以Dream为依据。

所提出的所有作品共享两个共同特征。首先，他们没有利用恢复的抽象来简化反编译代码，因此他们错过了提高可读性的机会。最多只能实现最小的可读性增强。

**用户研究**

据我们所知，我们首先在恶意软件分析领域进行用户研究。

## 第十节结论

在本文中，我们作了两个贡献。首先，我们创建了大量新颖的以可读性为重点的代码转换，以提高反编译代码的恶意软件分析质量。我们的转换简化了程序表达式和控制流程。他们还根据使用它们的上下文为变量和常量分配有意义的名称。其次，我们通过第一个用户研究验证了我们的改进，涉及学生和专业恶意软件分析师。结果清楚地表明，我们以人为本的反编译方法提供了显着的改进，Dream ++的性能优于Dream和Hex-Rays。尽管取得了巨大的进步，但我们相信我们在这个高度技术性的领域中只能勉强地抓住可以做到的一切。