COOP基本框架

1. 使用基本符号执行查找Vfgadgets

使用IDA工具对受害者程序进行反汇编，并使用调试符号静态识别C++模块中的所有vtable。如果调试符号不可用，就是用一种简单有效的启发式算法：将每个address taken的函数指针数组视为潜在的vtable，并检查所有已经识别的、不超过一定数量基本块的虚函数，将他们视为潜在的Vfgadgets。

1. 使用识别的Vfgadgets实现攻击语义（使用SMT求解器对齐重叠对象）

对于潜在的目标Vfgadgets，使用单一静态分配来分析基本块的语义，分析基本块的I/O行为，并根据不同的语义行为设计代码重用攻击。

1. 在缓冲区分配伪造对象

使用SMT求解器来对齐重叠伪造对象：在缓冲区中，对象的重叠字段实现了Vfgadgets之间的即时数据流。不同I/O功能的Vfgadgets进行重叠组合后，可以实现图灵完备的攻击操作。

COOP优点

1. 不会间接或直接跳转到non address-taken位置。COOP中，控制流仅通过现有的间接调用分派给应用程序内的、现有的、并且是address-taken的函数。
2. 函数返回方式符合C++常规程序的调用堆栈返回方式。胁持中，COOP既不会注入新的，也不会直接修改现有的返回地址以及其他代码指针，仅仅操纵或注入现有的vptr（即指向代码指针的指针）。
3. 使用间接分支的特征被良好隐藏。COOP攻击中可能会执行高比例的间接分支，易被检测执行间接分支频率的启发式算法发现。但是，COOP攻击依赖的ML-G组件本身是合法的C++虚函数，启发式算法为识别COOP攻击，将产生大量误报检测，防护成本过高。
4. 不会胁持堆栈指针。COOP仅仅胁持C++对象的vptr，不会胁持堆栈上易被检查修改的指针，不受阻止恶意返回的影子调用栈的影响。
5. 不注入新代码指针，不操纵现有代码，因此不受代码指针保护方案的影响。
6. 控制流与数据流与常规C++程序相似。通过胁持主循环ML-G组件，COOP攻击完全符合C++常规程序中访问对象的函数调用流程，即访问虚表，再选取目标虚函数进行执行并返回，不考虑C++语义的CFI无法防御COOP。另一方面，COOP复用vtable，最大程度上避免了畸形控制流。并且在适当放宽要求的前提下，可以通过修改vtpr的偏移量，执行任意数量任意种类的Vfgadgets。
7. 能广泛应用于C++应用。无论是x86或x64平台，COOP都具有良好的Vfgadgets传参手段。并且可以通过制作伪造对象，其vptr不指向实际的vtable，而是指向已加载模块的导入表或导出表，以便ML-G将WinAPI函数作为虚函数调用。
8. COOP是图灵完备的。COOP攻击可以通过使用W-COND-G vfgadget 来在满足某些条件的情况下，重写ML-G的索引，或者修改下一个待遍历的伪对象的指针，以实现条件分支与循环。

COOP不足

1. COOP依赖胁持C++对象以及其vptr，并且为了数据兼容，需要得知对象的二进制布局。如果无法胁持C++对象，或者无法修改vptr，抑或无法正确分析出胁持对象的二进制布局，就无法开展攻击。
2. COOP需要程序掌握足够大小的缓冲区。
3. COOP对vfgadgets的要求较为严格。如W-G（写入数据至目标地址）需要要求使用当前类对象的某个字段作为复制操作的长度。
4. COOP的攻击严格依赖虚函数的功能。因此，极难实现对全部API的攻击调用，例如，调用WinExec（）的虚函数应该接近不存在。另外，对于其他虚函数可能触及的API，如果COOP通过修改vptr指向已加载模块的导入表或导出表来调用WinAPI，首先这表现出异常控制流与数据流，其次受限于调用约定，伪造对象的指针总是以第一个参数传递给WinAPI，能实现的操作非常有限。最后，如果用非常量参数调用C风格函数指针的虚函数，大多数WinAPI的调用是可以实现的，但是，此类Vfgadgets相对罕见。
5. 潜在Vfgadgets的寻找，建立在对具有短基本块虚函数的分析上。一方面，这导致攻击方式相对简单；另一方面，大量充满潜力的Vfgadgets无法被挖掘。
6. 传参方式局限性较大。X64下，要求ML-G不修改参数传递寄存器。传递多参数时，仅适用于使用ebp而非esp访问堆栈上的局部变量的ML-ARG-G。

针对COOP的体系结构安全启示

1. 普通的CFI方法并不能避免胁持虚函数的攻击。不考虑C++语义来推导应该强制执行的CFG，即使用二进制代码，使间接调用只能转到address-taken函数。所有仅约束间接调用和跳转的防御方法，面对COOP都是无效的，因为COOP并不会注入新的vtable或代码指针，因此也不会跳转至non address-taken位置。
2. 注意到，COOP通过修改vptr实现危险的虚函数调用，那么只需要阻止与对象无关的虚函数调用就可以有效制止COOP。GCC的C++-aware virtual table verification（VTV）技术根据C++类层次结构严格限制每个vcall站点的目标。LLVM的简介函数调用检查（IFCC）中，每个间接调用站点都与一组有效的目标函数相关联（如果目标函数是address-taken且签名与调用站点兼容，则目标有效）
3. 考虑到COOP攻击依赖于Vfgadgets中基本块的语义，当从源码中考虑精确的C++语义时，可以可靠的防止COOP控制流。例如，VfGuard在vcall站点检查vptr是否指向任何已知vtable的开头，有效的限制了COOP的可用Vfgadgets集合。
4. C++数据结构的细粒度随机化。由于COOP攻击在选取初始对象时，需要得知对象的二进制布局。那么，在应用程序启动时随机化C++对象布局，例如在字段之间插入随即大小的填充，就能使COOP伪造的对象不可用。另外，对vtable的位置和结构的细粒度随机化也对COOP起到有效防御。
5. 代码隐藏。目前常见的代码隐藏方式，如随机重新排序基本块等，对COOP无效。需要进行的是细粒度的代码隐藏，使COOP无法通过反汇编的方式分析基本块语义。
6. COOP攻击需要控制足够大小的缓冲区。通过为C/C++应用提供内存安全形式的系统可以构成对一般控制流胁持攻击的强大防御。

附：

阅读记录

Counterfeit Object-oriented Programming On the Difficulty of Preventing Code Reuse Attacks in C++ Applications

COUNTERFEIT OBJECT-ORIENTED PROGRAMMING 面向伪造对象的编程

COOP是一种代码复用攻击方式，旨在攻击使用C++或其他面向对象编程语言开发的应用。

目标

目前的代码复用攻击方法，大多依赖于一些常规代码中不会出现的控制流或数据流模式，如：

C-1: 间接或直接跳转到non address-taken位置（即不使用函数指针执行到的代码位置，比如虚函数）

C-2: 不符合调用堆栈的函数返回方式

C-3: 过度使用间接分支

C-4: 胁持堆栈指针（暂时）

C-5: 注入新代码指针，或者操纵现有代码

这些特性很容易被低级、语言无关的保护方法检测到。因此COOP的设计目标为：

G-1: 不暴露特性C-1 - C-5

G-2: 表现出和正常C++代码执行相似的数据流和控制流

G-3: 能广泛应用于C++应用

G-4: 图灵完备

对手模型

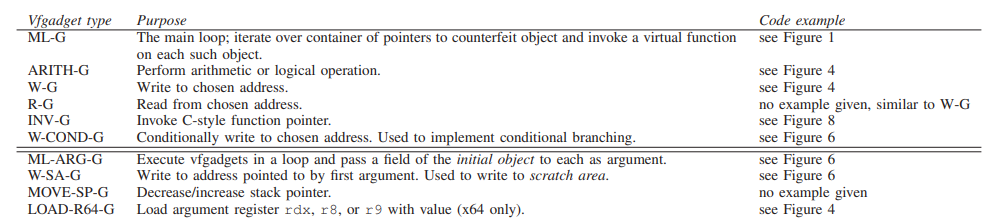
对于COOP，我们假设攻击者控制了一个有vptr（即有虚函数）的C++对象，并且能推断出该对象的基地址，或者可以控制足够大小的缓冲区。进一步的，能推断出一组二进制布局（部分）已知的C++模块的基地址。

基本方法

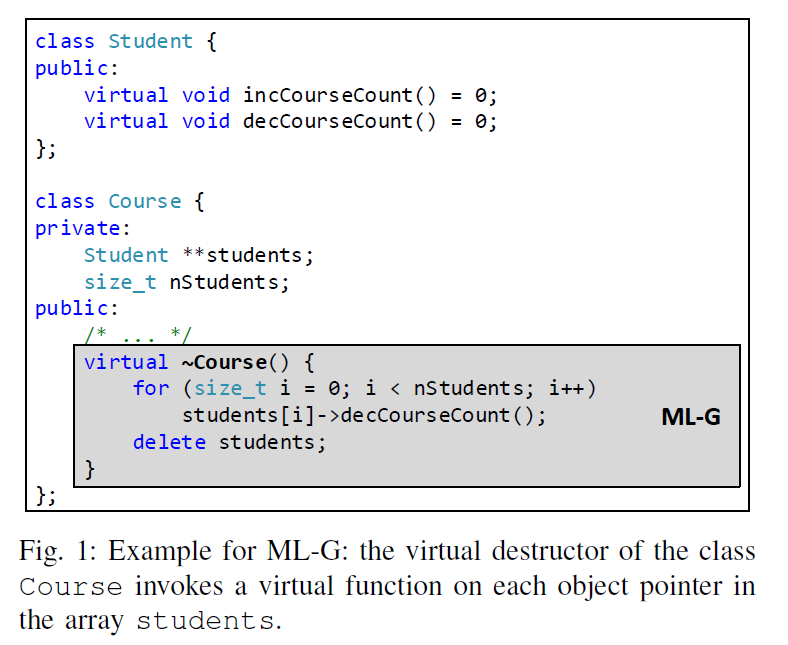
**初始对象（Initial object）**：目标程序中被劫持的C++对象。

**伪造对象（Counterfeit object）**：包含攻击者选择的vptr和数据字段，并被攻击者注入到可控内存中。

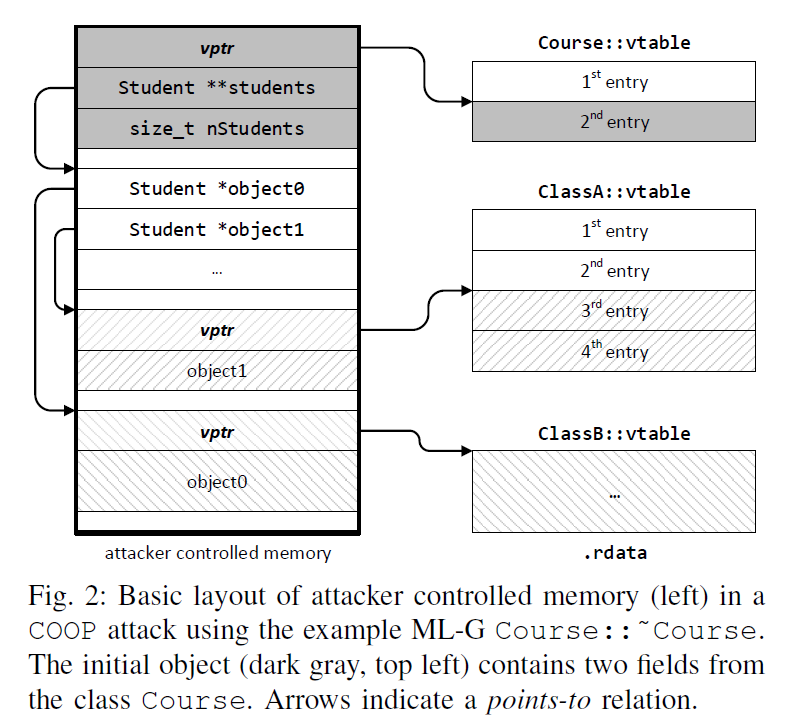
**Vfgadgets**：COOP攻击中使用的虚函数。攻击者通过源代码分析或二进制代码的逆向工程在实际攻击之前识别应用程序中的有用 vfgadgets。即使源代码可用，也需要在二进制级别确定 vfgadget类的实际对象布局，因为编译器可能会删除或填充某些字段。只有这样攻击者才能注入兼容的伪造对象。



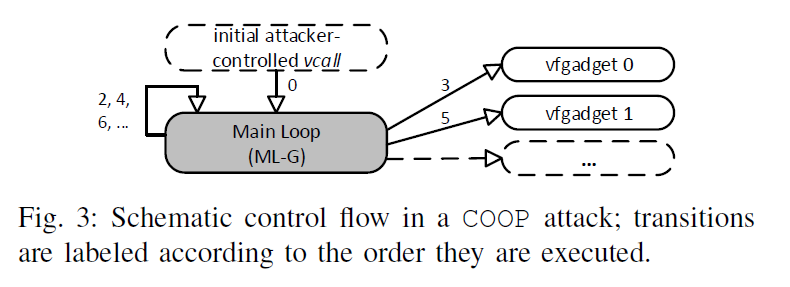
**主循环（Main loop）**:COOP依赖的特殊主循环cfgadget（ML-G）。ML-G是一种遍历指向C++对象的指针的容器，并在每个对象上调用虚函数。



**初始对象的布局**：攻击者将初始对象布局为ML-G对象的类的结构。攻击者在可控内存中构建完整的类对象，并在内部构造不同类型的伪造对象。初始对象包含ML-G类的字段的子集，即，使ML-G按预期工作所需的所有数据字段。



**伪造Vptr（Counterfeit vptrs）**：COOP攻击中的控制流与数据流应该类似于常规C++程序，因此避免引入假的vtable并重用现有的vtable。严格遵守G-2规则的话，示例程序应只使用在现有vtable中的2nd entry。略微放宽G-2规则，让vptr不必严格指向现有vtable的开头，而是前后偏移一点（如object1），便可以使用任意数量的vfgadget。

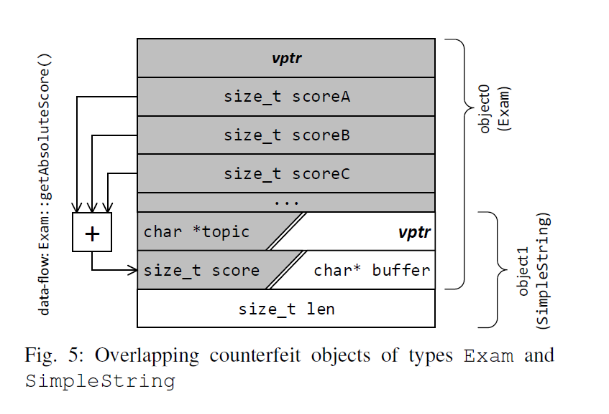
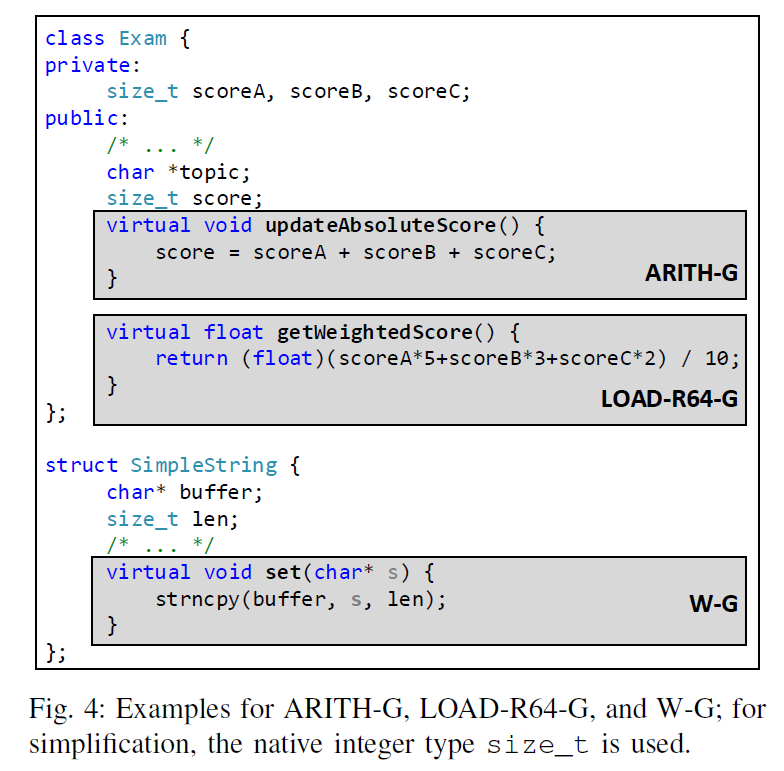


**重叠的伪造对象（Overlapping Counterfeit object）**：使用两个vfgadgets

Exam类中的ARITH-G（算数或逻辑操作）：将三个成员变量的和写入至当中的另一个字段

SimpleString类中的W-G（写入数据至目标地址）：使用当前类对象的某个字段作为复制操作的长度

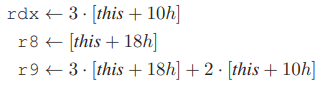
上述两个cfgadgets，一个写入数据，一个读取数据，因此攻击者可以将两个对象的内存重叠，使W-G中使用的length刚好是ARITH-G的计算结果。这允许将攻击者选择的数据写入动态计算的内存地址。



**向vfgadgets传参（Passing Arguments to Vfgadgets）**：

1. windows x64传参：函数的前四个（非浮点）参数通过寄存器rcx，rdx，r8和r9传递。对于C++代码，this-ptr作为第一个参数传递给rcx。所有四个参数寄存器都被定义为调用者保存，因此虚函数通常使用rdx，r8和r9作为临时寄存器，并且在返回时不会恢复或清除他们。于是，在x64上向vfgadgets这样传参：首先，挑选一个合适的vfgadget并执行，以便于将伪造的字段分别写入至函数调用参数传递寄存器。然后执行目标vfgadget，其参数使用上一个vfgadget刚刚伪造的值。

在MSVC的编译下，LOAD-R64-G给攻击者提供了这样的语段：

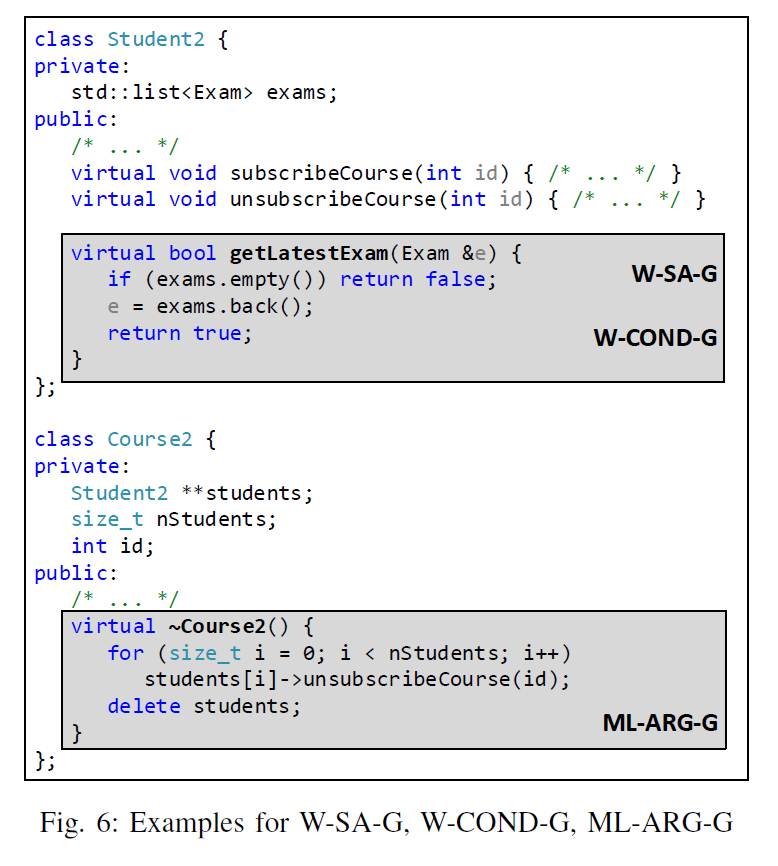


攻击者只需要合适的选择10h和18h偏移处的值，就可以向rdx，r8和r9写入任意的值。

该方法要求ML-G不修改参数传递寄存器。

1. windows x86传参：所有常规参数都通过堆栈传递，而this-ptr在寄存器ecx中传递。因此参数传递依赖于ML-G主循环。ML-G应将初始对象的某个字段作为参数，传入给每个vfgadget。之后，攻击者使用以下方法将目的参数传递给目标vfgadget：

A-1 传递的参数是一个指针，指向一个临时可写内存，这样vfgadgets可以通过读写这块内存来传递参数。ML-ARG-G是用于传参的主循环，Course2::~Course2传递了一个参数id给其他vfgadget，而在Student2::getLatestExam方法中，将id视作一个指针，并动态修改了指针指向的内存。这样当下一个vfgadget获取到参数后，能读取上一个vfgadget所保留的信息。



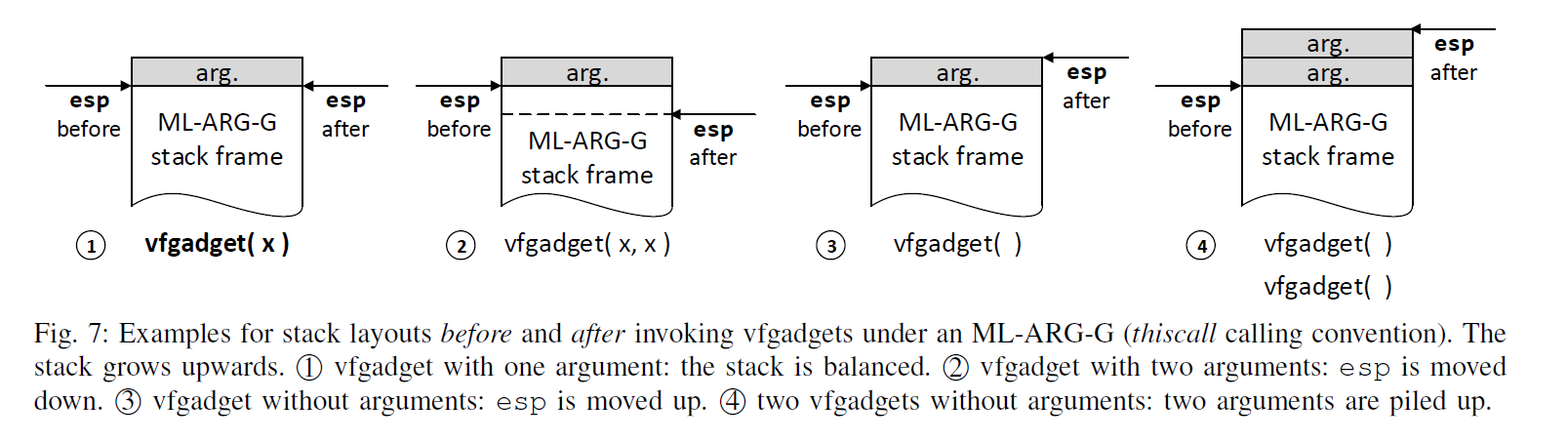
A-2 动态重写参数。需要一个可用的 W-G 类型的vfgadget。

传递多个参数：

若调用的 vfgadget 实际使用的参数个数比 ML-ARG-G 传递的要少时，新传递的参数将被永久压栈（因为不使用参数的 vfgadget 将不会清理栈）。多调用几次（即多压几次栈），那么栈内存上就有构建好的一组参数值。

而若调用的 vfgadget 实际使用的参数个数比 ML-ARG-G 传递的要多时，在 vfgadget 函数返回时，该函数将额外弹出”参数“的栈空间，即向下恢复栈。

该技术仅适用于使用 ebp 而非 esp 访问堆栈上的局部变量的 ML-ARG-G。



**调用API**：

W-1：使用一个能合法调用感兴趣的 WinAPI 函数的 vfgadget。

W-2：调用 WinAPI 函数就像从 COOP 主循环中调用一个虚函数。

W-3：使用一个调用 C 风格函数指针的 vfgadget。

W-1在大多数场景下不可行，例如，调用WinExec（）的虚函数应该接近不存在。

W-2可以通过制作伪造对象，其vptr不指向实际的vtable，而是指向已加载模块的导入表（IAT）或导出表（EAT），以便ML-G将WinAPI函数作为虚函数调用。

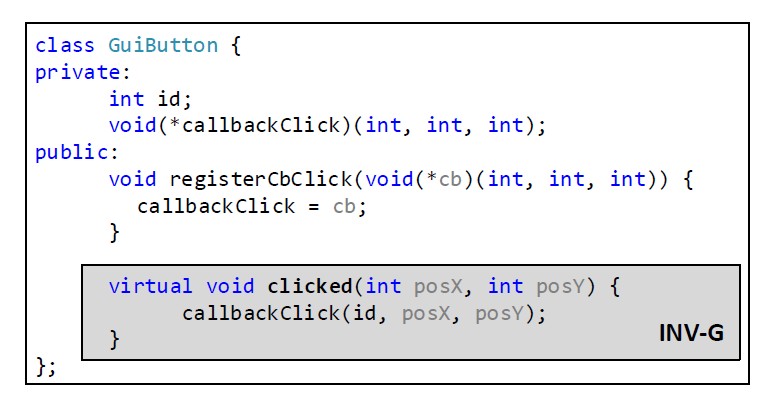
优点：易于实现。例如让 vptr 指向诸如 GOT、IAT、EAT 等表。

缺点：违反目标G-2，没能展示出类似于正常 C++ 代码执行的控制流和数据流；受限于调用约定，伪造对象的指针总是以第一个参数传递给 WinAPI。

W-3需要一种特殊类型的 vfgadget：一个用非常量参数调用C风格函数指针的虚函数。这个特定的vfgadget允许使用至少三个攻击者选择的参数调用任意WinAPI函数。

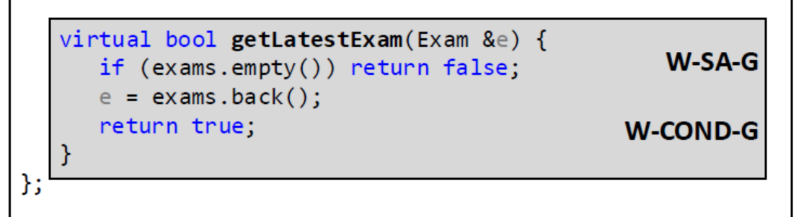
优点：比 W-1 更灵活并且比 W-2 更隐蔽。

缺点：INV-G 是相对的罕见的。



**实现条件分支和循环**：

COOP 攻击可以通过使用 W-COND-G vfgadget 来在满足某些条件的情况下，重写 ML-G 的索引，或者修改下一个待遍历的伪对象的指针。这种重写需要知道对应变量的地址，若索引存放在栈上，则可以通过上面的压栈和弹栈来移动栈指针，达到修改目标地址上索引的目的。



COOP框架

实现COOP攻击的三步：1）识别vfgadgets 2）使用识别的vfgadgets实现攻击语义 3）在缓冲区安排伪造对象

**A 使用基本符号执行查找Vfgadgets**：vfgadget搜索器仅依赖于二进制码，使用IDA工具反汇编。搜索器使用调试符号静态识别C++模块中的所有vtable，或者如果这些符号不可用，则使用一组简单但有效的启发式算法（算法将每个address taken函数指针的数组视为潜在的vtable，并检查所有已识别的，不超过一定数量基本块的虚函数）。实践中使用具有一个或三个基本块的虚函数作为潜在vfgadgets。（优势：更容易评估语义，表现出更少的副作用 缺点：长的vfgadgets可以用于欺骗攻击检测器）

使用单一静态分配（single static assignment）来分析基本块的语义，以紧凑且易于分析的形式反映了基本块的I/O行为。

**B 使用SMT求解器对齐重叠对象**： COOP程序由其对应特定vfgadget的伪造对象的顺序和位置决定。重叠伪造对象的字段实现了vfgadgets之间的即时数据流。COOP编程环境定义了伪造对象和标签，具有相同标签的字节会被映射到缓冲区的相同位置，而不同标签的字节映射到不同位置，而没有标签的字节永不重叠。