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。

2） 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。

3） 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。