### CFG 机制及缺点

**郑瑞辰 2018141531071**

CFG 是一个称作流程控制技术的措施，旨在解决内存损坏漏洞，针对的是间接跳转的保护，比如call eax，jmp eax等。CFG扩展了先前的漏洞缓解技术，例如GS，DEP和ASLR微软实现的CFG主要关注缓解间接调用和调用不可靠目标的问题，不可靠的目标有明显特征：在大部分情况下，它不是一个有效的函数起始地址。微软的CFG实现是基于间接调用的目标必须是一个可靠的函数的起始位置。

**1. CFGBitmap是怎么生成的**

在Windows 10中，有了CFG的支持，它指向ntdll!LdrpValidateUserCallTarget函数。这个函数使用目标地址作为参数。Guard CF function table：函数的相对虚拟地址（RVA）列表的指针，其包含了程序的代码。每个函数的RVA将转化为CFGBitmap中的“1”位，CFGBitmap的位信息来自Guard CF function table。在这里，编译器完成了CFG的整个工作。剩下的是OS的支持使CFG机制起作用。在OS引导阶段，第一个CFG相关的函数是MiInitializeCfg。这个进程是system，MiInitializeCfg函数的前置工作是创建包含CFGBitmap的共享内存。调用时间可以在NT内核阶段1内存管理器初始化时找到（MmInitSystem）。 然后获得压缩RVA列表信息的函数且保存到映像的Control\_Area结构，PE映像第一次加载到系统。NT内核将调用MiRelocateImage来重定位。MiRelocateImage将调用MiParseImageCfgBits。在函数MiParseImageCfgBits中，PE映像的压缩的RVA列表被计算且存储在PE映像节中的Control\_Area数据结构。在系统引导期间一个PE映像只发生一次。当PE再次加载进进程，NT内核将调用MiRelocateImageAgain。因为它的压缩的RVA列表已经保存了（且不需要再次计算）。MiParseImageCfgBits被用来计算启用CFG编译的模块的压缩的RVA列表。在深入这个函数之前，我们将看一下这个函数调用的上下文。函数MiParseImageCfgBits将在MiRelocateImage函数中调用。

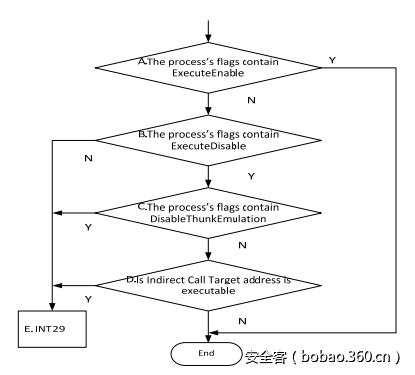
在CFGBitmap共享内存对象被创建后，CFGBimap共享内存对象被映射来作为两种用途：

1. 用来写共享模块的bits。这个映射是临时的；在bits写入完成后，映射将释放。通过这个映射写入的bits信息是共享的，意味着它能被操作系统内所有的进程读取。这个映射发生在MiUpdateCfgSystemWideBitmap函数中。
2. 用来写私有的bits和读取校验间接调用的bits。通过这个映射写入的bits是私有的，意味着它只能被当前进程读取。这个映射的生存周期与进程的生命周期相同。

如果PE映像的RVA列表准备好了且CFGBitmap节也映射了，就可以将RVA列表翻译为CFGBitmap中的bits。MiCfgBitMapSection是一个共享内存对象，包含了CFGBitmap的共享的bitmap的内容。它与每个进程共享。当它在自己的虚拟内存空间中映射MiCfgBitMapSection时，每个进程看见的内容都相同。

当创建一个进程，NT内核调用PspPrepareSystemDllInitBlock函数来写CFGBitmap映射的地址和全局变量的长度，其数据结构是PspSystemDllInitBlock结构。PspSystemDllInitBlock是修正过的地址并且从用户模式和内核模式都能访问。用户模式可以访问硬编码的PspSystemDllInitBlock全局变量的CFGBitmap字段。

在所有的准备都完成后，如果间接调用的目标地址相关的位在CFGBitmap中不是“1”，将触发CFG。进程将采取行动处理这个异常。处理函数是RtlpHandleInvalidUserCallTarget。这个函数使用间接调用的目标为唯一的参数。



如果一个进程需要自定义CFGBitmap，它能调用ntdll中的NtSetInformationVirtualMemory。在内核中函数MiCfgMarkValidEntries实现了这个功能。MiCfgMarkValidEntries以一个缓冲区和长度作为参数。缓冲区中的每个单位是8字节。头四个字节是目标地址，其想在CFGBitmap中设置相关的位，且后四个字节是设置“0”或“1”的标志。MiCfgMarkValidEntries自定义的CFGBitmap只在当前进程能看见。

1. **CFG能否绕过**

CFGBitmap空间地址存储在修正过的地址中，其能被用户模式代码获得。这在CFG实现中讨论过。这是很重要的安全问题，但是被简单的放过了。如果主模块没有开启CFG，即使加载的启用了CFG的模块，进程也不会受保护。

如果一个进程的主模块禁用了DEP，能绕过CFG访问处理，即使间接调用的目标地址是不可靠的。

在CFGBitmap中的每个bit在进程空间中表示8个字节。因此如果一个不可靠的目标地址少于8个字节，CFG将认为是可靠的。

如果目标函数是动态生成的，CFG的实现不能保护。这是因为NtAllocVirtualMemory将在CFGBitmap中为所有分配的可执行的内存空间设置为“1”。通过MiCfgMarkValidEntries自定义的CFGBitmap解决这个问题是可能的。