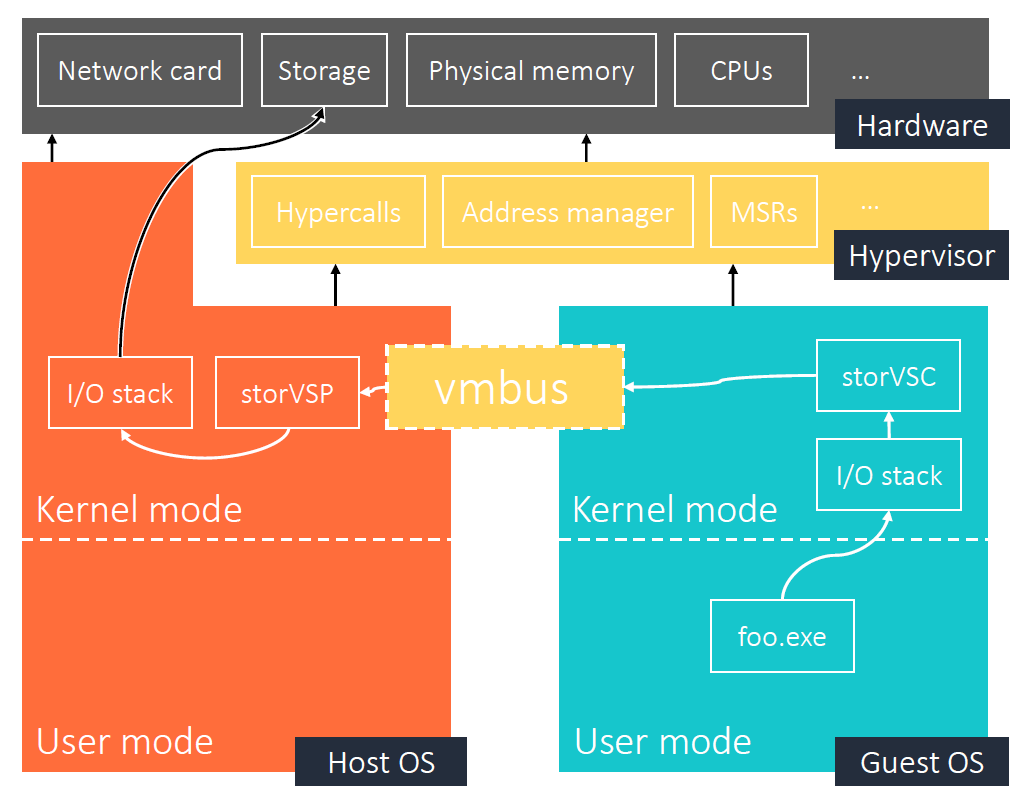
通过进攻性安全研究加固Hyper-V（附现场逃逸视频）

本议题由微软安全工程师[Jordan Rabet](https://www.blackhat.com/us-18/briefings/schedule/speakers.html" \l "jordan-rabet-37155)提供，由冰刃实验室（IceSword Lab）成员闫广禄、秦光远和廖川剑为大家第一时间进行分析解读。议题首先讲解了Hyper-V的guest与host的通信模型；然后具体到vmswitch，包括vmswitch的初始化流程和相关漏洞；又讲解了漏洞的利用过程，包括：控制写入数据、赢取竞争、寻找攻击目标以及绕过KASLR等；最后议题提出了一些加固Hyper-V的方法。

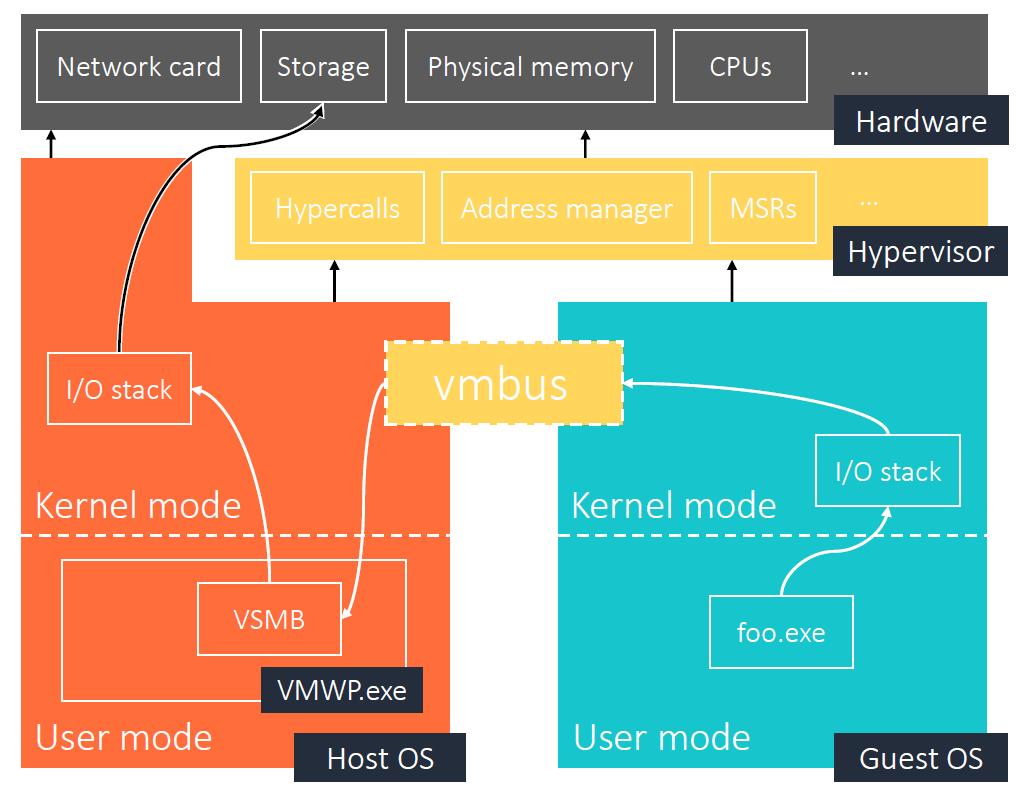
首先我们介绍一下Hyper-V中Guest如何访问硬件资源。

Hyper-V中Guest访问硬件资源通常需要Host协助，下面是一个guest访问存储资源的流程图：

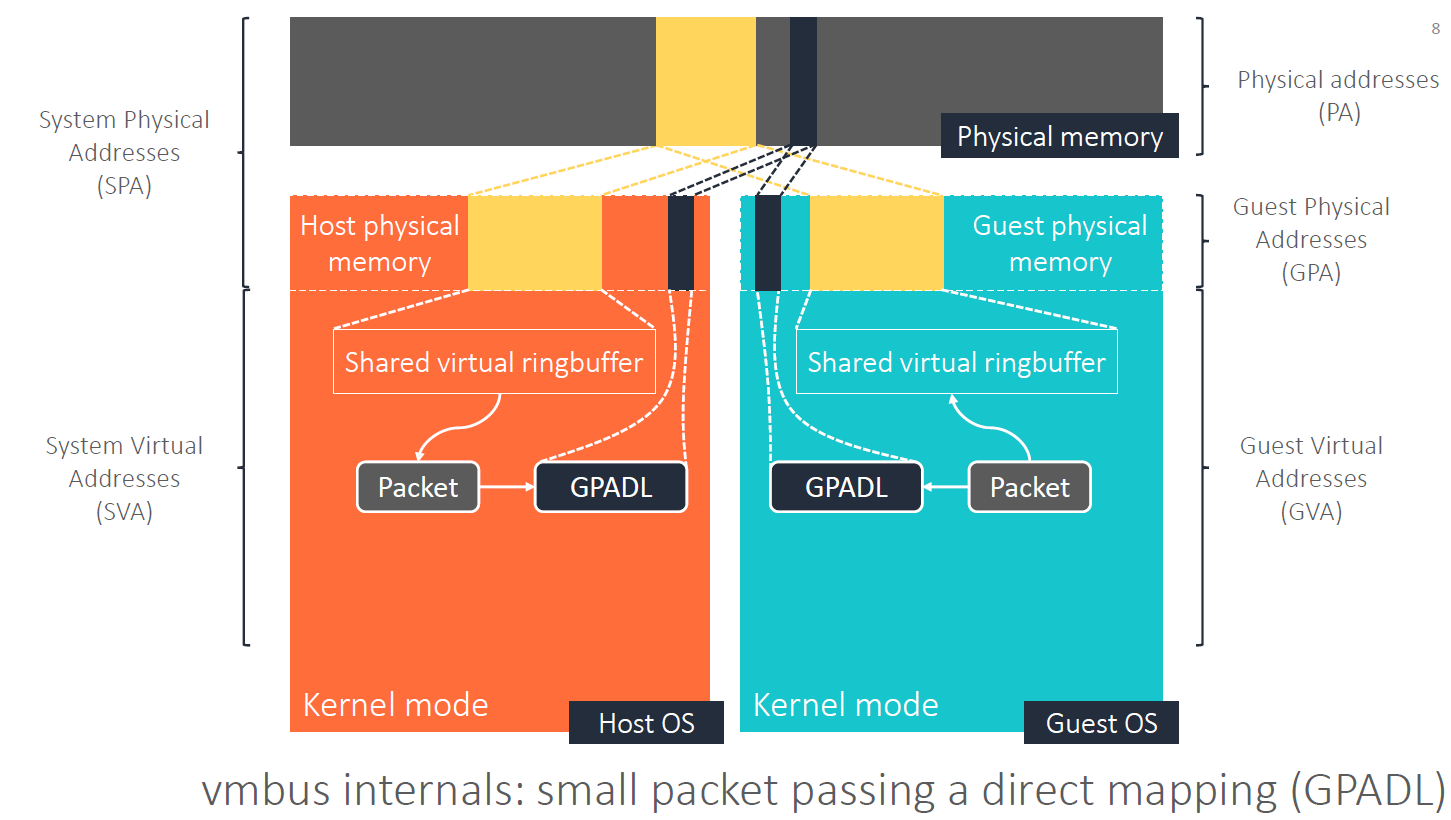


Guest中的进程通过I/O stack与guest内核中的storVSC（Virtualization Services Client组件）进行交互，而storVSC通过vmbus与Host内核中的StorVSP（Virtualization Services Provider组件）进行通信，StorVSP通过host的I/O stack进行硬件操作，这样就隔离了guest对硬件的直接访问。

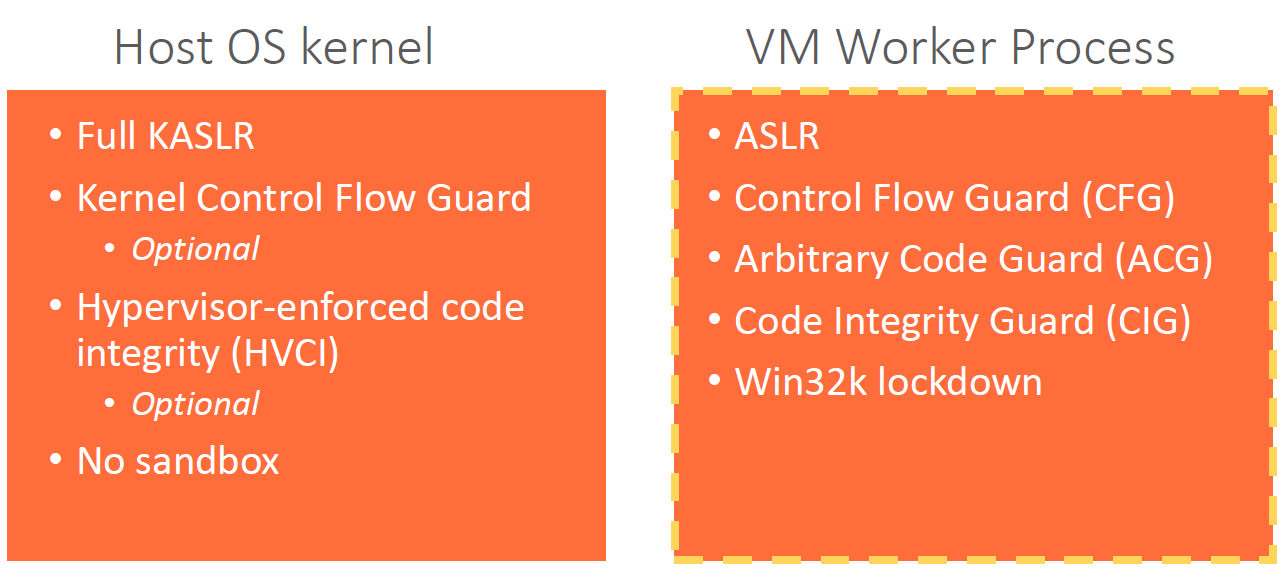
Virtualization Services Provider也可以在用户态，如下图，由虚拟机进程VMWP.EXE中的VSMB提供服务。



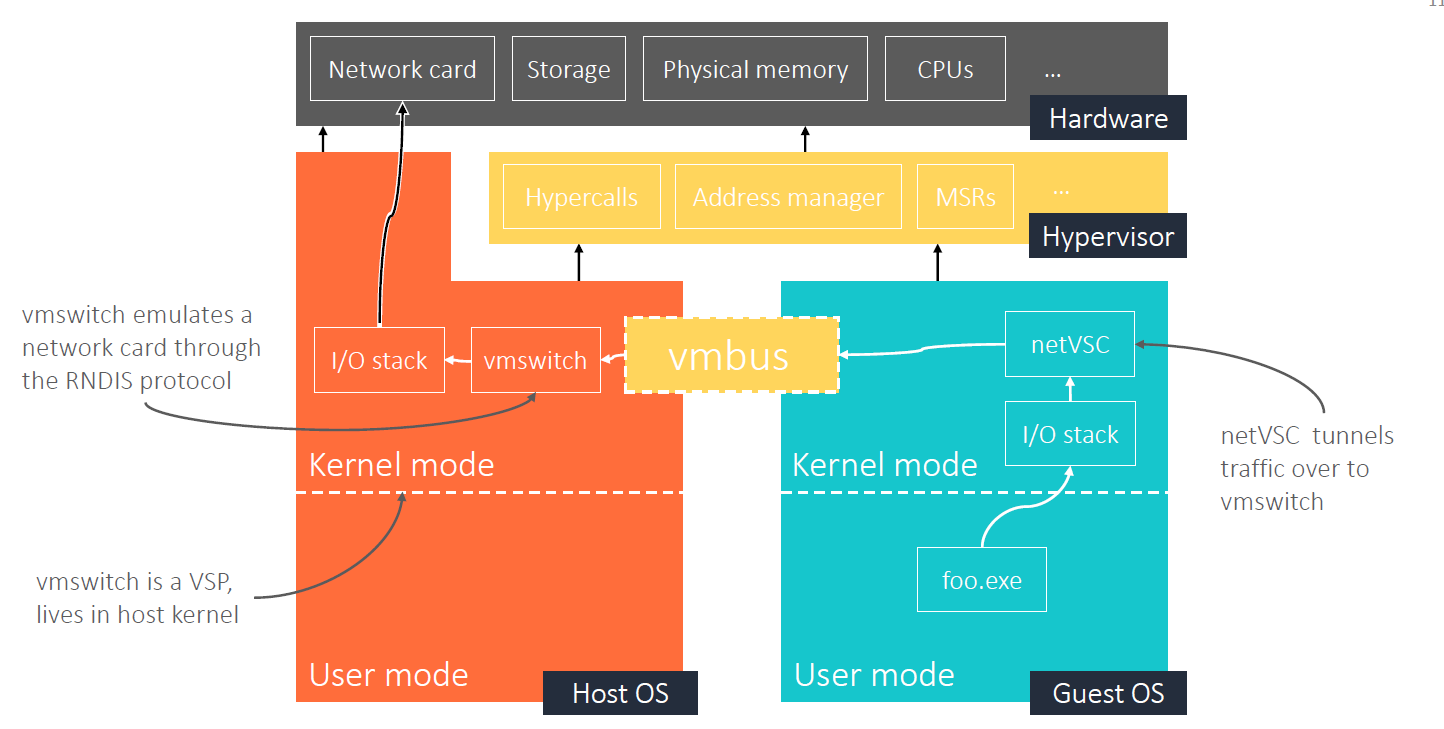
VMBUS是通过共享内存的方式为Virtualization Services Client和Virtualization Services Provider提供通信服务。



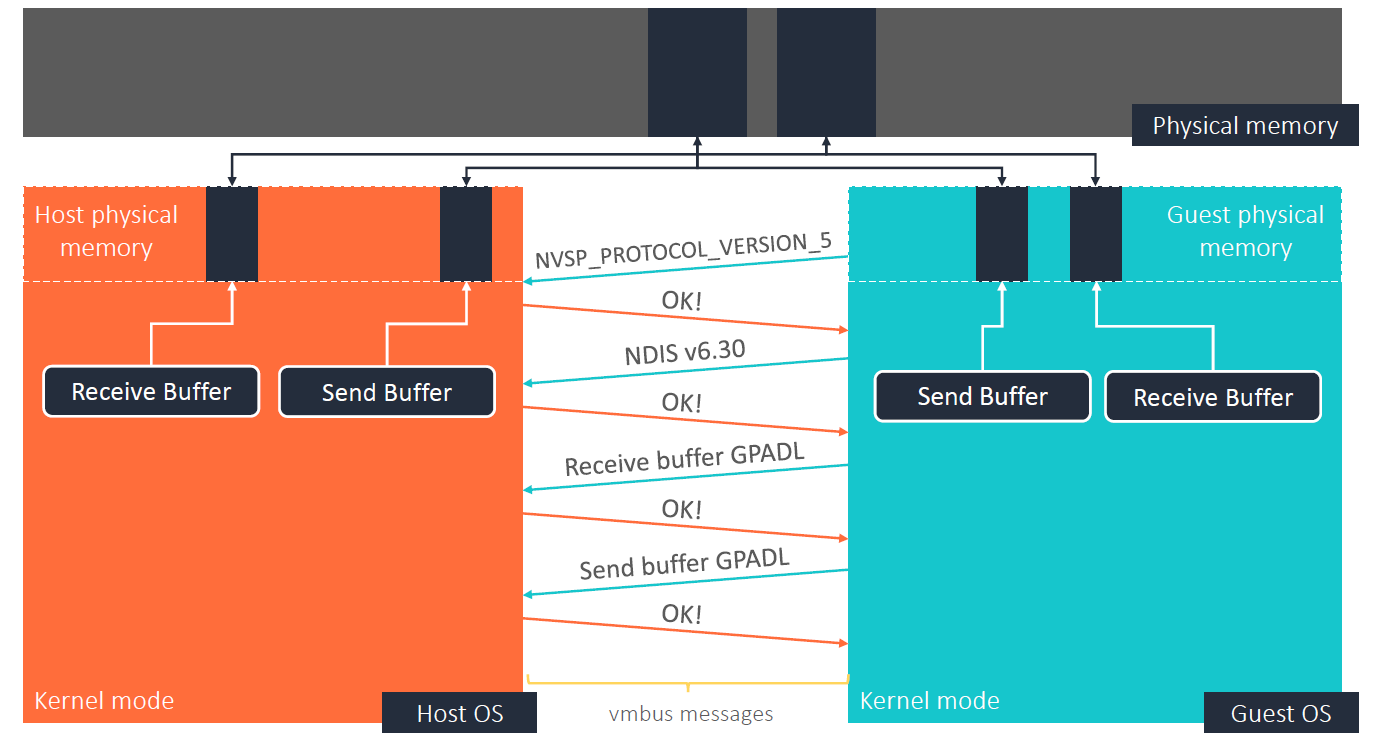
另外为了防止guest利用漏洞进行攻击，Host OS提供了很多缓解机制，包括：



介绍完这些背景知识后，我们以vmswitch作为一个例子来进行深入学习。Vmswitch作为VSP提供虚拟机的网络服务，它基于RNDIS协议。



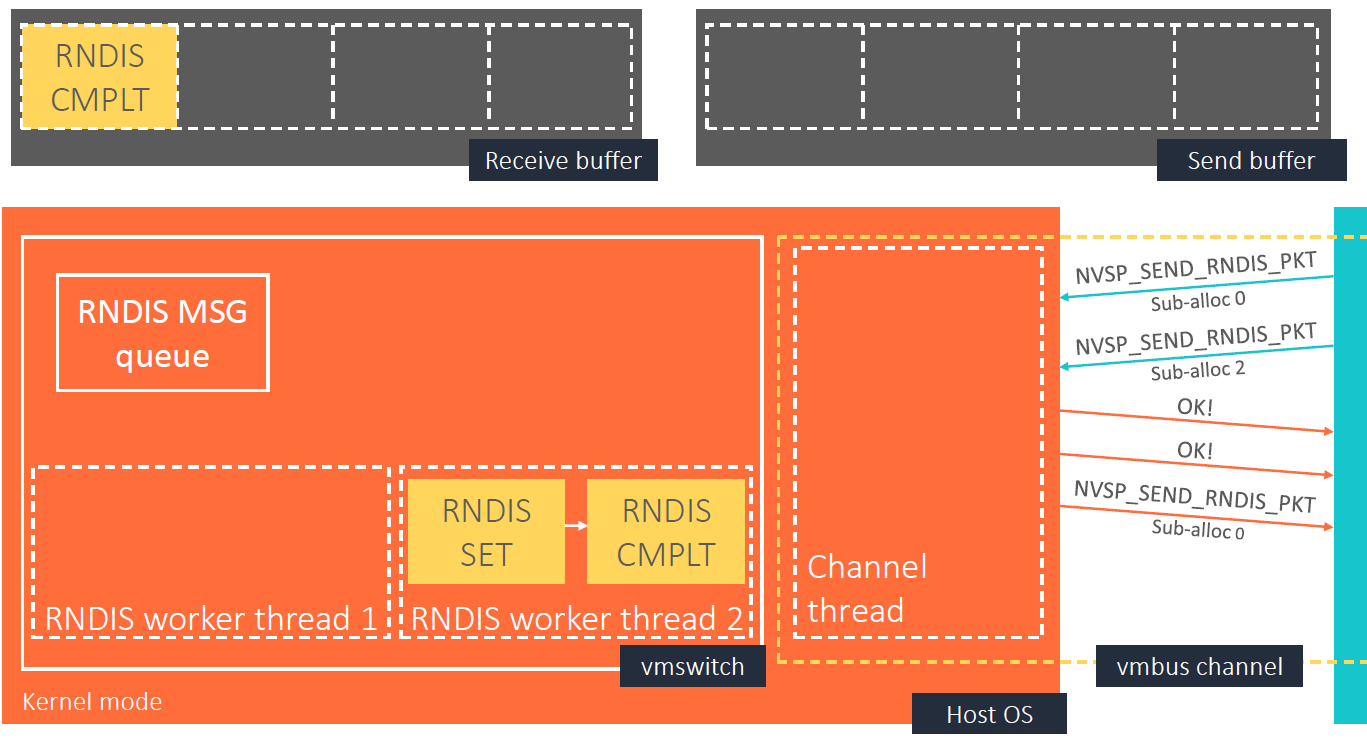
它的初始化过程如下所示：



实际上它通过共享内存建立了接收缓冲区和发送缓冲区，这里的接收和发送是相对与guest而言。

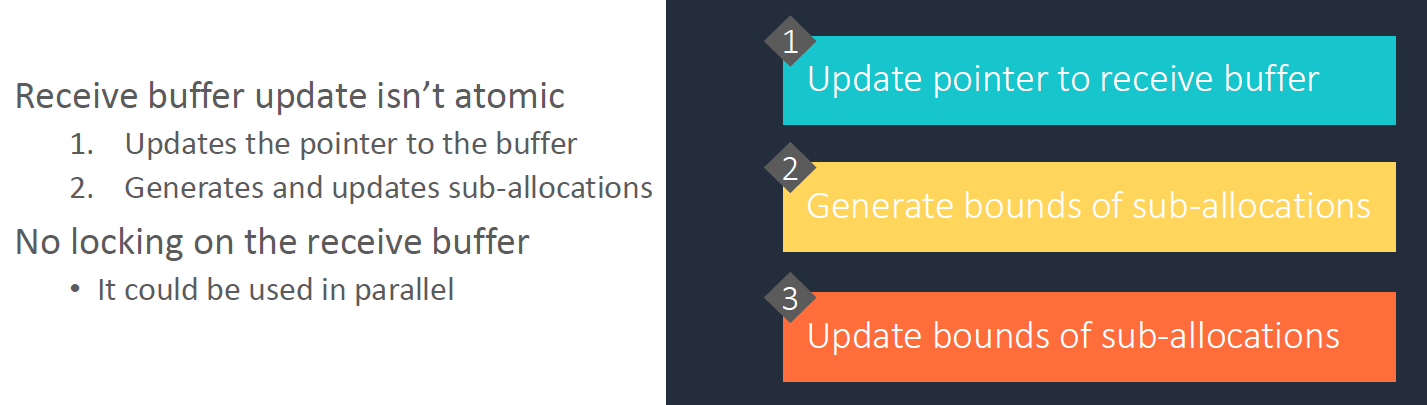
在发送RNDIS包时，需要进一步把发送/接收缓冲区切分成子块（sub-allocation），netVSC通过发送缓冲区将query包发送给vmswitch，而vmswitch在处理完后将cmplt包通过接收缓冲区发回给netVSC。

那么vmswitch又是如何处理RNDIS消息的呢？实际上vmswitch有多个线程来处理guest的数据包，同时还有一个RNDIS消息队列来存储guest的数据包。首先vmswitch有一个channel thread，它负责将发送缓冲区的数据包存储进RNDIS消息队列中，然后这些数据包会被分配给不同的线程进行处理。最后，当处理完成时，相应的cmplt数据包会被存储到接收缓存区中。

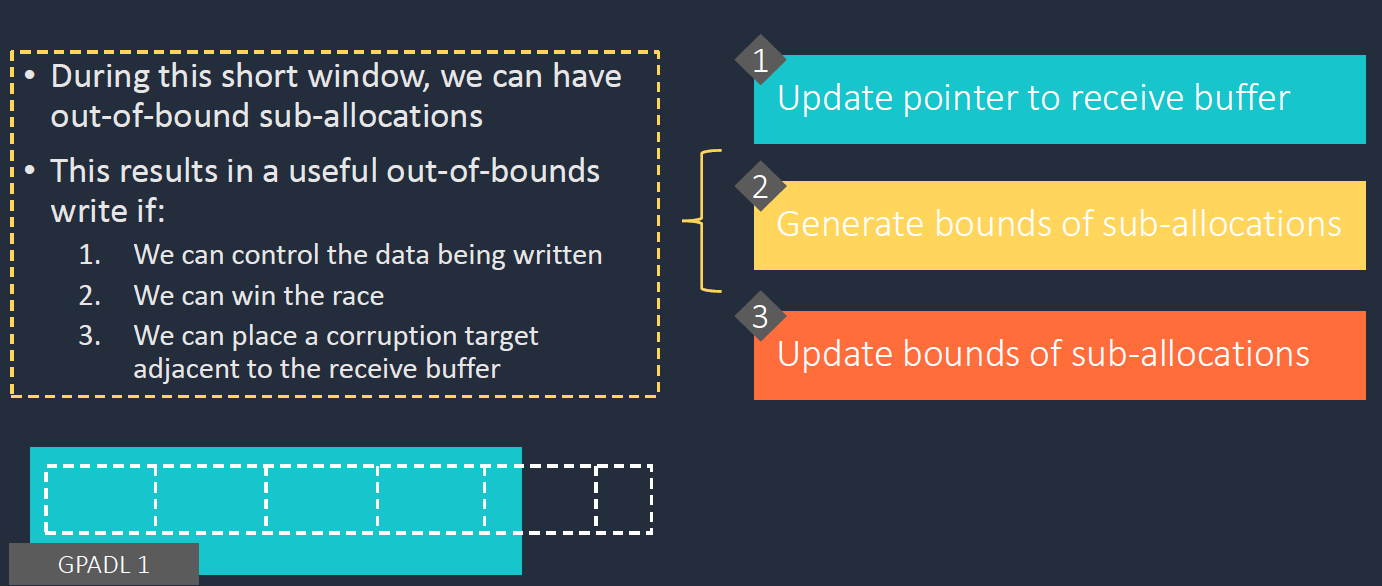


接下来我们来看一下vmswitch中的初始化序列漏洞。

漏洞的原因在于：vmswitch初始化过程中，利用GPADL建立共享内存时需要更新接收缓冲区指针，而这个更新过程并非原子操作，可能导致竟态攻击（update 接收缓冲区指针与update bounds of sub-allocations的竞争），进而造成out-of-bounds。



如果满足下面的条件，那么这个漏洞就很可能可以被利用。



**首先是我们能否控制被写入的数据（即，写到越界位置的数据）**。

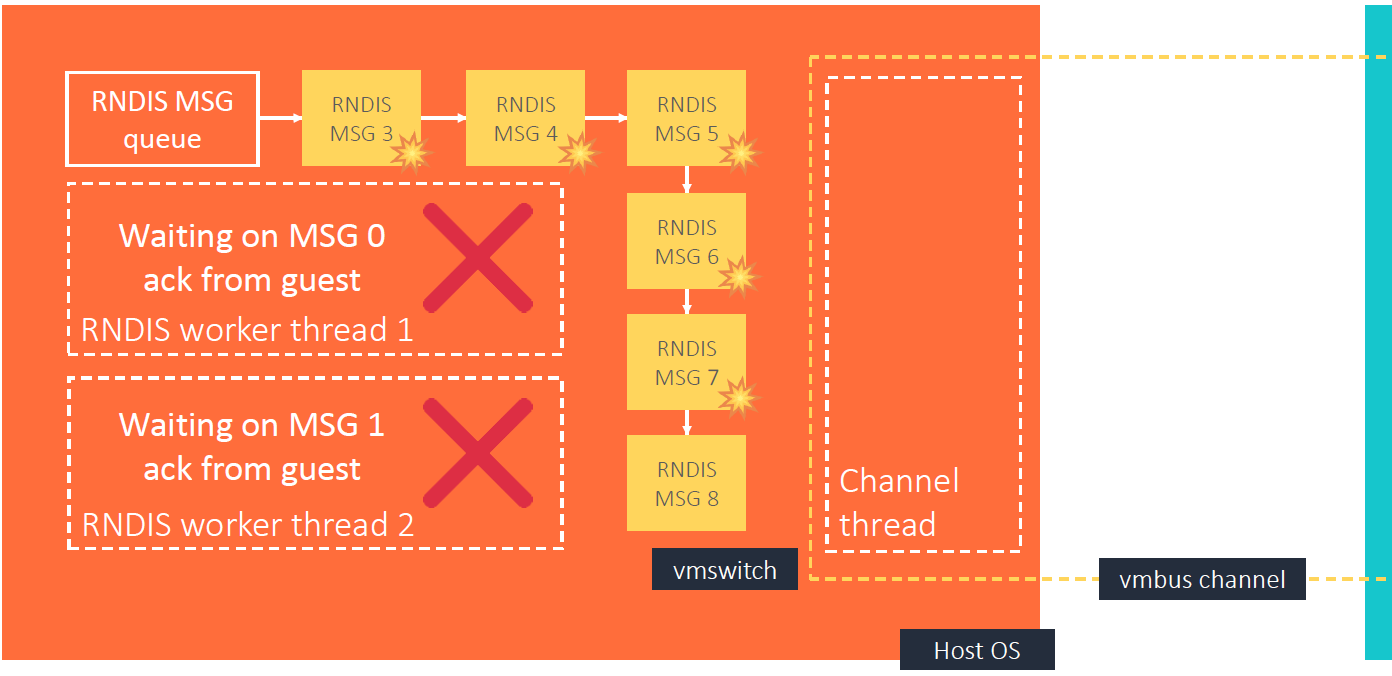
作者通过RNDIS control message responses来控制写入的内容。

**第二个问题是赢得竞争。**

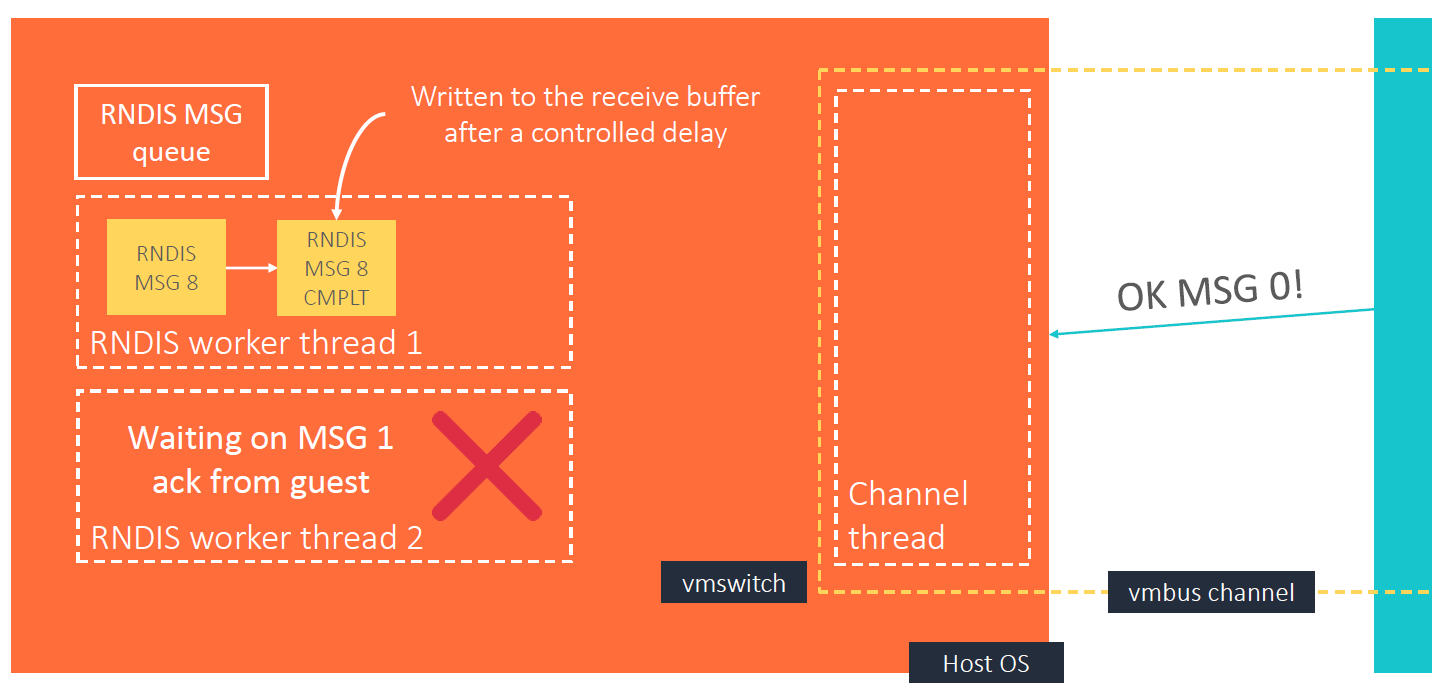
之前我们说过，host实际上是由多个线程来处理guest的数据包。当一个线程处理完guest数据包后会生成cmplt数据包，并将cmplt数据包发回给guest，只有guest发送了应答之后，线程才会继续处理下一个数据包。那么所有的数据包都需要guest应答吗？答案是否定的，例如畸形的

RNDIS\_KEEPALIVE\_MSG数据包。这样我们就有了以下的攻击思路：

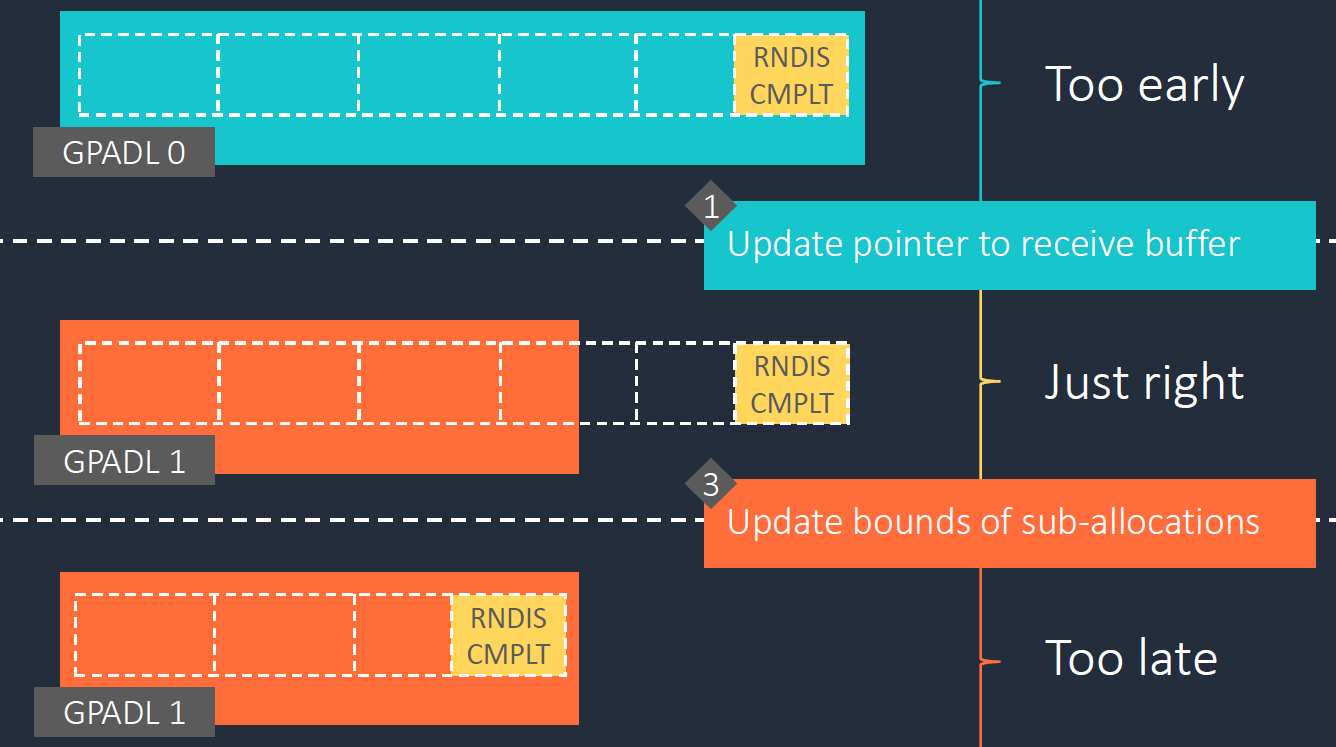
1. 阻塞所有RNDIS工作线程
2. 串联N个畸形的RNDIS\_KEEPALIVE\_MSG数据包，来获得适当的延迟
3. 在最后面放置一个有效的RNDIS数据包



这样就可以在一个可控的延迟后向接收缓存区写数据。（guest通过发送OK MSG 0让线程1恢复执行）



那么问题来了，N要如何选才能赢得竞争？



如果N选的过低，cmplt数据包就会落在上图的“Too early”阶段，此时应当适当的增加N；如果N选的过高，cmplt数据包就会落在上图的“Too late”阶段，此时应当适当的减少N。作者说通常少于10次尝试就能让N收敛，N也会根据不同的机器而发生变化。

**第三个大问题是寻找一个攻击目标，这个攻击目标需要邻近接收缓冲区。**

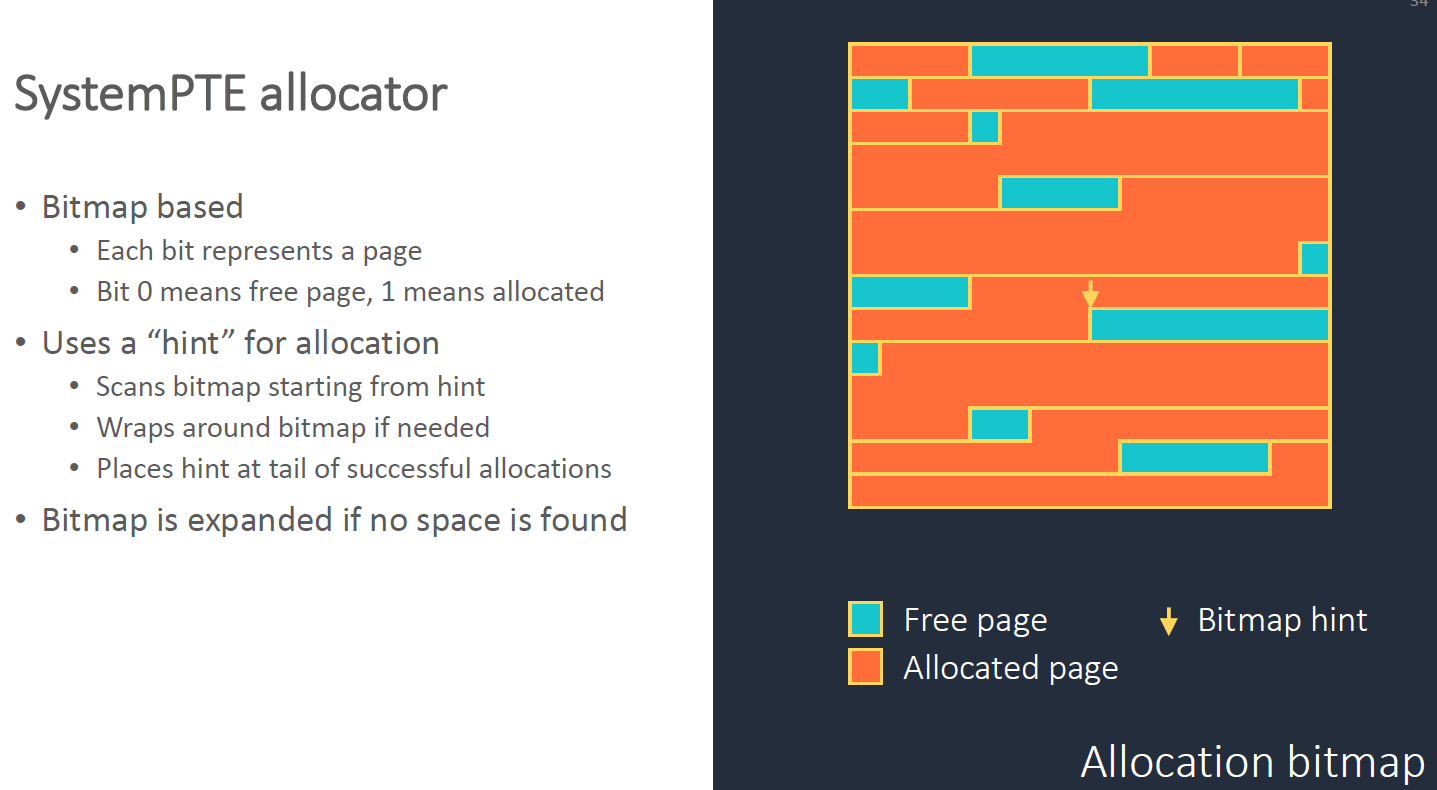
首先确定我们的接收缓冲区所在的内存位置。GPADL利用MDL映射物理地址，而这些MDL又会被映射到SystemPTE区域。通过调试发现，与这些MDLs相邻的是其它MDL和内核栈，因此我们选取内核栈作为攻击目标。

Windows内核栈共包含了7个页面，其中6个页面作为栈空间，而最后一个页面在底部作为guard page。内核栈也在SystemPTE区域进行分配，并且可以通过内核栈构建rop，所以它是一个很好的攻击目标。

那么我们又会产生如下几个子问题。

1. SystemPTE区域是如何进行内存分配的？
2. 我们能否把内核栈设置在我们指定的区域（即与接收缓冲区存在一个指定的偏移）？
3. 我们可以设置内核栈吗？如何衍生线程？

首先说一下SystemPTE的内存分配器：

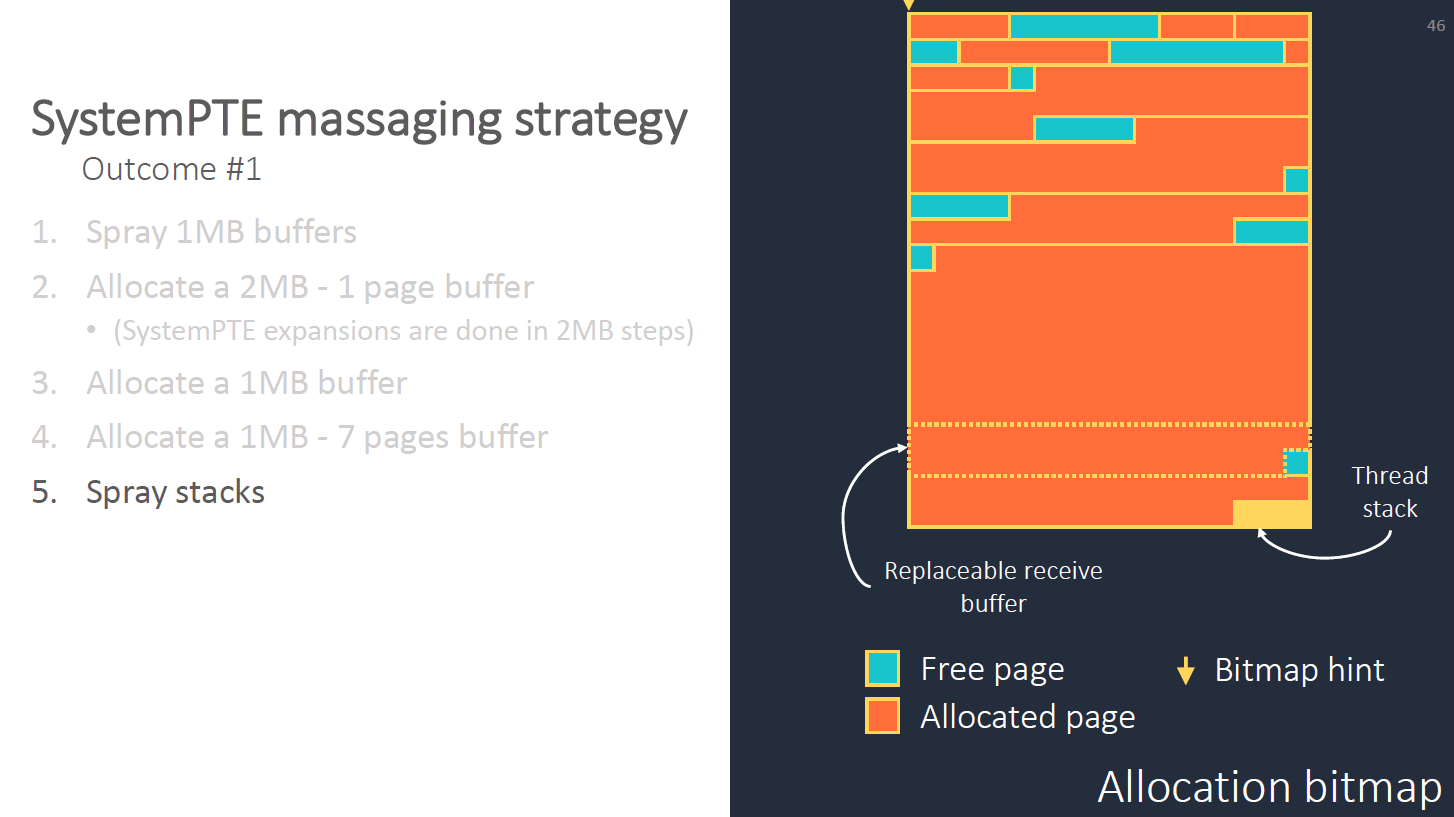


可以看到SystemPTE allocator实际是基于一个可扩展的bitmap来记录内存的分配。

接下来我们就要寻找一个内存分配基元，来帮助我们布局内存。实际上这个基元可以直接利用接收/发送缓冲区来构造：我们可以映射任意数量和任意大小（实际数量和大小还是有限制的）的MDL，同时我们还能借助NVSP\_MSG1\_TYPE\_REVOKE\_RECY\_BUF和NVSP\_MSG1\_TYPE\_REVOKE\_SEND\_BUF来撤回缓冲区。因此我们就有了内存分配和释放的基元来帮助我们操控这个区域。但是我们还需要一种方法来生成新的内核栈，以便于我们去操控它，所以我们还需要栈分配基元。

Vmswitch依赖于系统工作线程来处理异步任务，这些工作线程在内核维护的线程池中，只有当所有线程都处于busy状态时，额外的线程才会被加入。所以我们的核心思想是：多次快速的触发异步任务。如果产生的任务足够快，那么就会有新的线程加入。有几类vmswitch消息依赖于系统工作线程，例如我们使用的NVSP\_MSG2\_TYPE \_SEND\_NDIS\_CONFIG。

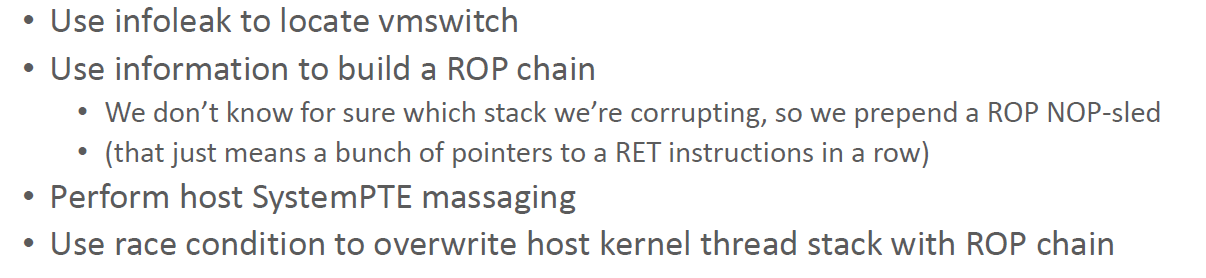
这种方法通常会让我们创建5个线程，如果在系统的线程池中已经存在很多个线程了又该如何？最好的方法是结束这里的线程，可惜的是我们没有这样的途径。幸运的是，有可以利用的bug来实现类似的目标。这个bug可以造成系统工作线程死锁。如之前所述，NVSP\_MSG1\_TYPE\_REVOKE\_RECY\_BUF或NVSP\_MSG1\_TYPE\_REVOKE\_SEND\_BUF可以用来撤销缓冲区，但是当多个撤销消息被处理时，除了最后一个工作线程，其它的工作线程都会被永久死锁。因此我们可以利用这个bug来锁住“任意数量的”系统工作线程。这样我们就可以根据这个bug来实现一个受限的线程栈喷射。这样我们就可以通过下面的步骤来分配一个靠近接收缓冲区的内核栈。



最后我们还要绕过KASLR。

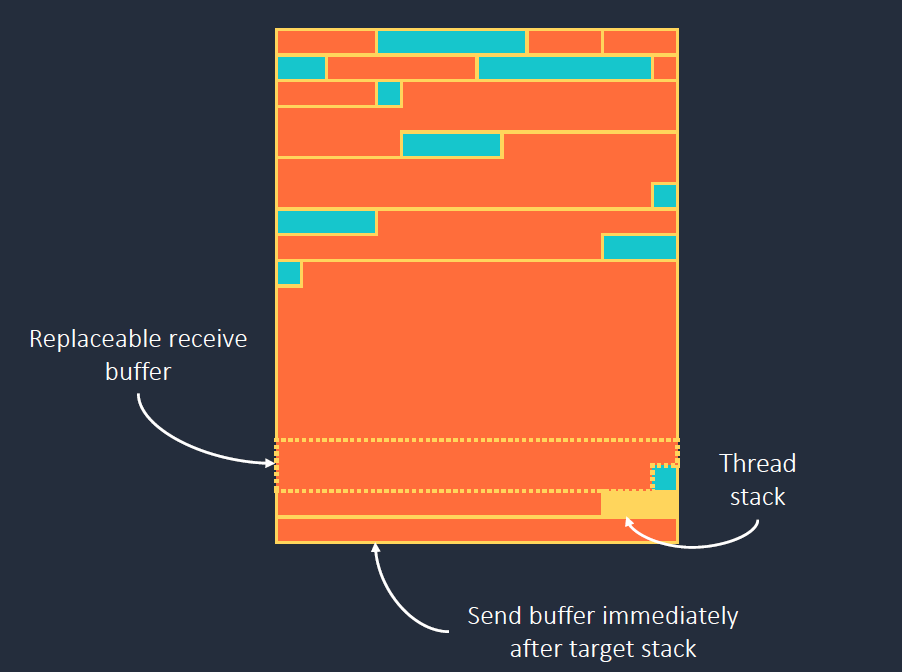
这里用到一个信息泄露的漏洞，造成信息泄露漏洞的结构体是nvsp\_message，它在栈上进行分配，它只初始化了前8个字节，却返回了sizeof(nvsp\_message)大小，因此32字节未被初始化的栈内存会被发回给guest，造成信息泄露。通过泄露的信息，我们能够获得vmswitch的返回地址，进而构造rop链。

总结一下整个的攻击过程：

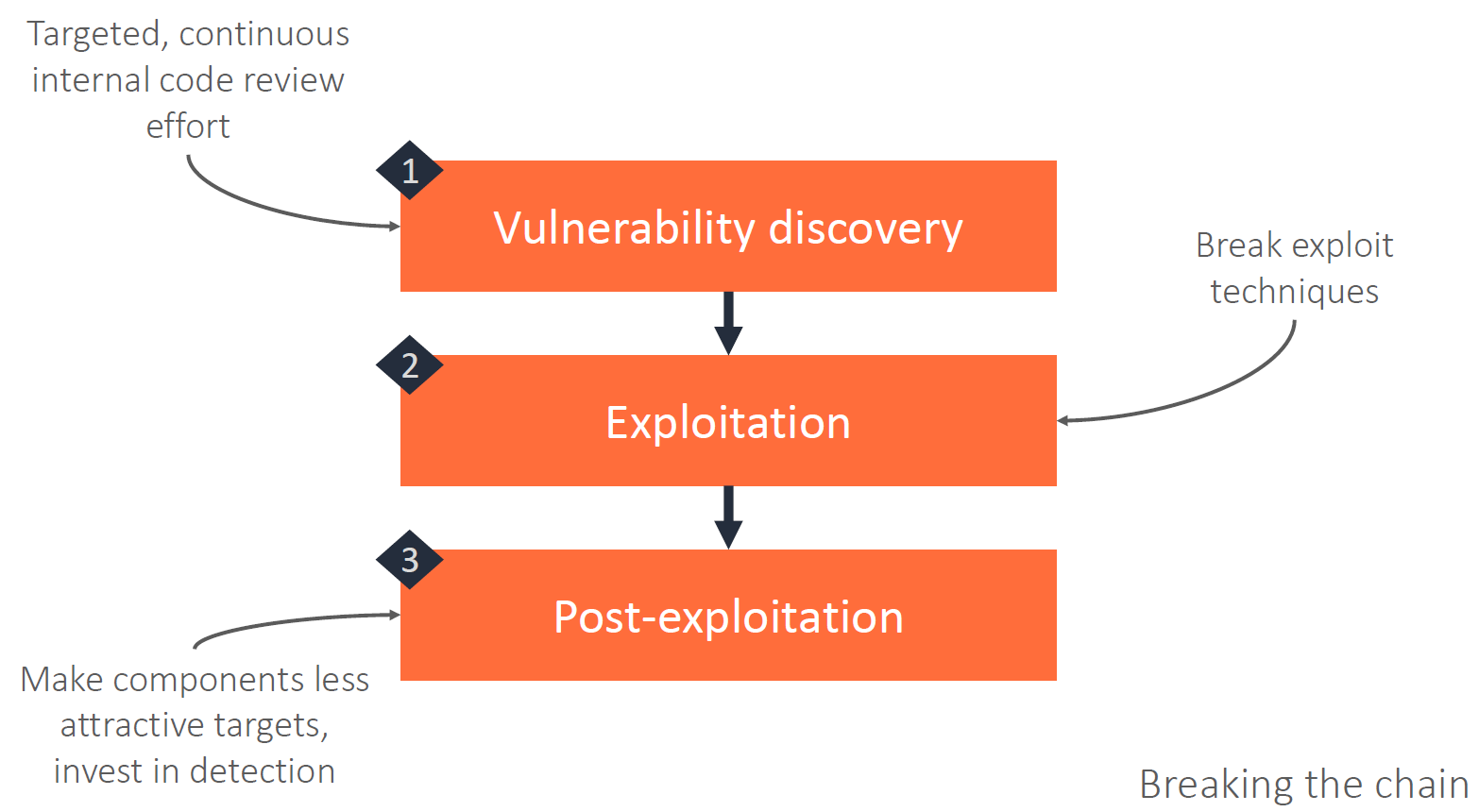


这个信息泄露漏洞只在Windows Server 2012 R2上，Windows 10中并不存在这个漏洞。那么在没有信息泄露的情况下我们如何绕过KASLR呢？实际上作者使用了部分覆盖返回地址的方法。

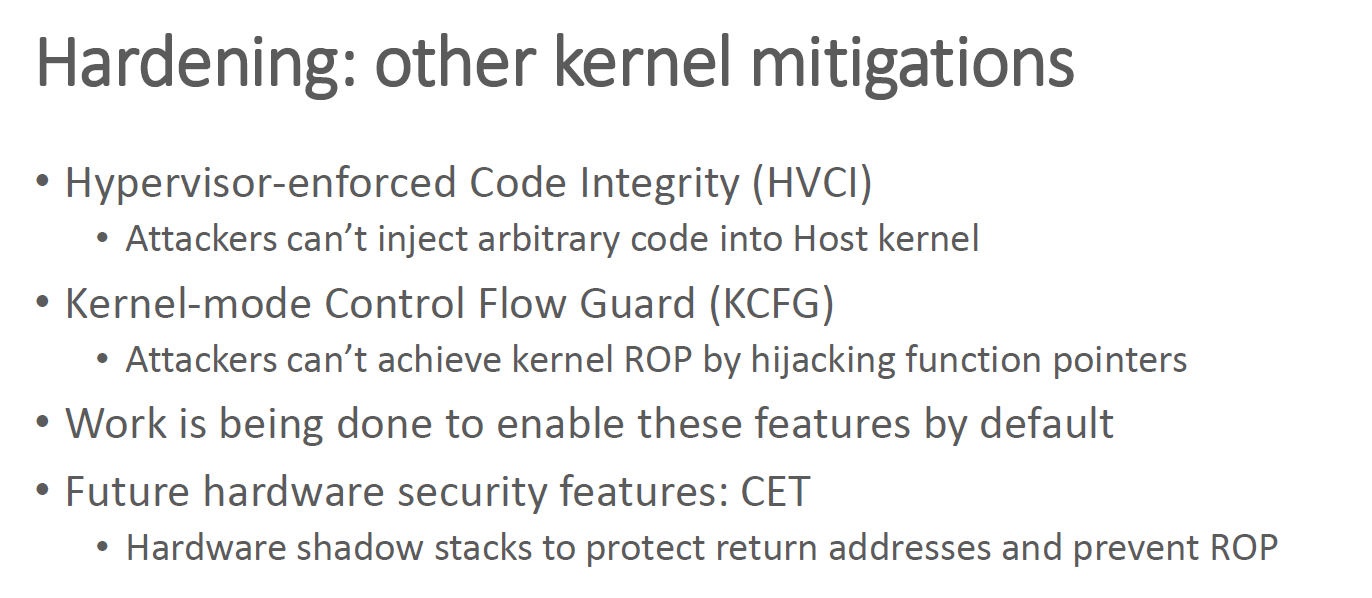
覆盖部分返回地址，从而触发#GP异常，#GP异常的处理函数会使用相同的内核栈来dump出异常信息（包括引发#GP异常的地址），而此时内核栈刚好又会越界到发送缓冲区（与guest共享）中，因此guest可以获得相关信息从而绕过KASLR。

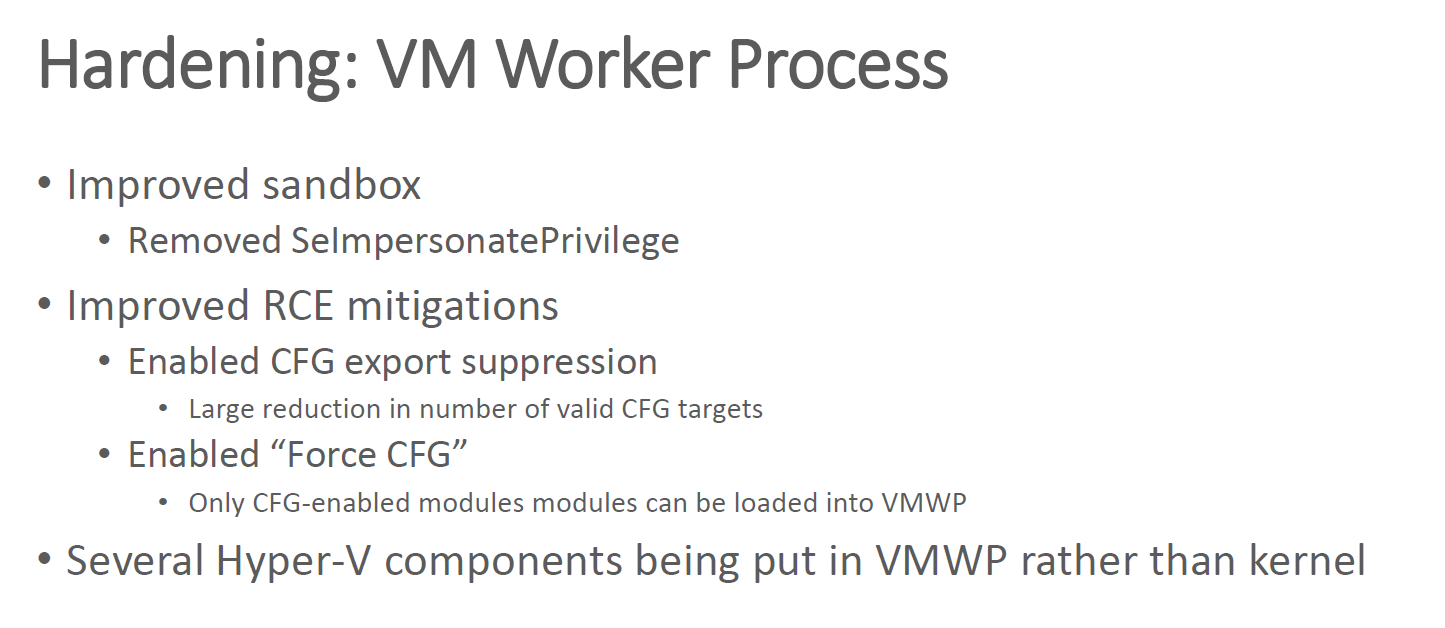


演讲的最后作者给出了一些加固Hyper-V的方法，主要的思路如下：



作者把内核栈进行了隔离，即把内核栈移到了它们自己的区域。另外在内核态和用户态也增加了一些保护机制：





以上就是我们的分析。时间有限，难免疏漏，欢迎大家指正。最后附上一段我们现场录制的逃逸视频。