第4章作业

姓名：袁昊男 学号：2018091618008

1、简述常见的漏洞利用技术以及每种技术分别适用的场合。

**答：**(1) 利用跳板覆盖返回地址。当子函数执行完成后，pop出上一函数的返回地址到EIP。ESP往往指向EIP+8，将shellcode放在此处即可以让程序执行。之所以会使用到跳板技术，是因为在静态定位shellcode 的过程中，需要将函数的返回地址覆盖成为shellcode的地址，但是shellcode是填充在缓冲区内部，而且该地址硬编码，在下一次程序重新装载之后其地址就会发生变化，也就是常说的“帧移位”。此时再按照之前的硬编码数据进行地址覆盖就会造成程序EIP 指向其他数据，导致程序崩溃。为了解决这个问题，提出了跳板技术进行shellcode的动态定位。要实现shellcode的动态定位，就必须找到一个不变的量，以此为参考进一步找到shellcode的位置。如将函数返回地址处的指令修改为jmp esp的地址（也就是将函数的返回地址重定向），当然也可以修改为其他指令。跳转指令jmp esp执行之后，将之后的地址空间覆盖为shellcode，这样再jmp esp指令执行后，就会继续执行shellcode。适用场合：当系统打开ASLR时，使用硬编码地址就无法成功利用漏洞。在这种情况下就可以使用这种技术，程序必须关闭NX。

(2) ret2libc技术。用libc中的system()的地址覆盖返回地址，同时在栈上布置好参数，程序返回时就会产生一个shellcode。适用场合：当系统打开DEP保护时，不能直接在栈上放shellcode。就可以使用几乎每个系统都会自带的libc库中的代码。

(3) ROP技术。面向返回语句的编程方法，它借用libc代码段里面的多个retq前的一段指令拼凑成一段有效的逻辑，从而达到攻击的目标。适用场合：当系统打开DEP保护时，不能直接在栈上放shellcode。经过充分研究并证明ROP方法是图灵完备的，换句话说， ROP可以借用libc的指令实现任何逻辑功能。

(4) 绕过GS保护机制。检测cookie前触发异常、尝试覆盖其它在cookie被检测前就被引用的数据、通过同时替换.data和cookie、覆盖父函数的栈数据等方式来绕过GS保护机制，从而实现栈溢出。适用场景：系统开启了GS保护机制。

2、简要说明在Shellcode中进行API自搜索和代码重定位的作用和过程。

**答：**(1) API自搜索：作用是在编写shellcode时，一般需使用一些API函数，这些函数的入口地址位于系统的动态链接库中，由于不同操作系统的动态链接库的加载地址不同，shellcode中需增加API自搜索技术。过程：

Win32程序都会加载ntdll.dll和kernel32.dll。首先定位kernel32.dll中的API地址：

1) 通过段选择字FS在内存中找到当前的线程控制模块TEB。

2) 线程控制块中偏移位置为0x30的地方存放着指向进程控制块PEB的指针。

3) 进程控制块中偏移地址0x0c的地址存放着指向PEB\_LDA\_DATA结构体的指针，其中存放着已经被装载的动态链接库信息。

4) PEB\_LDA\_DATA结构体偏移地址为0x1c的地方存放着指向模块初始化链表的头指针InInitializationOrderModulelist。

5) 模块初始化链表InInitializationOrderModulelist中按顺序存放着PE装入运行时初始化模块信息，第一个链表节点是ntdll.dll，第二个链表节点就是kernel32.dll。

6) 找到属于kernel32.dll的节点后，在此基础上再偏移0x08就是kernel32.dll在内存中的加载基地址。

7) 从kernel32.dll的加载基地址开始偏移0x3c的地方就是其PE头。

8) PE头偏移0x78的地方存放着指向函数导出表的指针。

9) 按以下方式在导出表中算出所需函数的入口地址:

导出表偏移0x1c处的指针指向存储导出函数偏移地址（RVA）的列表；导出表偏移0x20处的指针指向存储导出函数函数名的列表（0x24存序号地址表）；函数的RVA地址和名称按顺序放在RVA列表及函数名列表中，根据函数名在函数名称列表中找到函数序号，再根据函数序号在RVA列表中找到函数对应的RVA；函数对应的RVA加上动态链接库的加载地址得到该函数的虚拟地址。

(2) 代码重定位：采用call指令实现。如call Next会执行以下语句：push EIP，JMP Next。 即先保存下一句指令的地址，然后跳到Next处执行。而Next后面紧跟shellcode，所以push EIP就会把紧跟后面的shellcode的地址保存在堆栈中，然后跳到Next执行。而在Next处立即pop EBP，就会把保存的EIP（其实就是shellcode的地址）赋给EBP，这样EBP就是shellcode的地址，实现了动态重定位。

3、简述漏洞挖掘方法中白盒测试、黑盒测试和逆向分析的异同。

**答：**白盒测试：使用白盒测试，需要和客户组织一起工作，来识别出潜在的安全风险。白盒测试的最大好处是测试者将拥有所有的内部知识，并可以在不需要害怕被阻断的情况下任意地实施破坏。而白盒测试最大的问题在于无法有效地测试客户组织的应急响应程序，也无法判断出他们的安全防护计划对检测特定攻击的效率。白盒测试适用于时间比较紧急，或是特定的渗透测试环境如情报收集并不在范围之内的测试场景。

黑盒测试：经过授权的黑盒测试是设计成为模拟攻击者的入侵行为，并在不了解客户组织大部分信息和知识的情况下实施的。黑盒测试可以用来测试内部安全团队检测和应对一次攻击的能力。黑盒测试比较费时，同时对技术要求比较高。在安全业界的渗透测试眼中，黑盒测试能更逼真地模拟了一次真正的攻击过程。黑盒测试依靠测试人员的能力探测获取目标系统的信息，作为一次黑盒测试的渗透测试者，通常不需要找出目标系统的所有安全漏洞，而只需要尝试找出并利用可以获取目标系统访问权代价最小的攻击路径，并保证不被检测到。

异同：白盒测试和黑盒测试是对软件或系统进行渗透测试的两种方法，逆向分析是渗透测试所需要的工具，其中黑盒测试更需要借助逆向分析来获取目标软件或系统的源代码和输入数据的标准格式。

4、简述fuzzing测试的原理。

**答：**Fuzzing测试，也称为模糊测试，或健壮性测试，是安全测试的一种，主要是指发现系统潜在的安全性缺陷的测试，也可以称之为漏洞的发掘。Fuzzing测试是一种特殊的黑盒测试方法，与同样基于黑盒的功能性测试不同，Fuzzing测试的主要目的是发现系统的未知漏洞（0day漏洞）。Fuzzing技术源于软件测试中的黑盒测试技术，它的基本思想是把一组随机数据作为程序的输入，并监视程序运行过程中的任何异常，通过记录导致异常的输人数据进一步定位软件中缺陷的位置。Fuzzing测试是业界常见的一种安全测试手段，通常是一种黑盒的测试方法，测试前不需要了解被测系统的内部实现细节，也不需要掌握被测系统的源代码，而是通过“尝试各种可能性”来找到被测系统的漏洞。Fuzzing测试需要构造“各种可能性”，也就是测试用例，测试用例越多，覆盖面越广，就越能够测试出被测系统的漏洞。

5、如果Shellcode中不允许出现特定字符，应当如何处理？请给出一种通用方法。

**答：**对shellcode进行编码。如采用异或运算：存储shellcode前对shellcode每一字节与特定值进行异或运算；在跳转至shellcode前再次将异或后的shellcode与特定值进行异或运算。连续两次异或会得到值本身，因此这种方式能避免特殊字符。还有其他多种对shellcode的编码方式，参考资料：https://www.guhei.net/post/jb635。