# 背景介绍

软件漏洞是攻击计算机系统最常见的媒介。尽管人们在设计、构建和部署没有缺陷的软件方面付出了很大的努力，但即使是相对简单的软件系统中也仍然有漏洞存在。更加严重的是，即使在可信计算基中也可能存在足以令攻击者绕开系统中的安全机制的严重漏洞。

在所有的软件漏洞中，内存破坏型的漏洞是最常出现，也是最严重的漏洞之一。这有以下几个原因：首先，大量的程序是由C或C++这种类型不安全的语言编写的，这使得内存破坏型漏洞极为常见；同时，内存破坏漏洞的攻击能力非常强，很多攻击形式都利用内存破坏漏洞来完成攻击，可以导致的结果下至加密密钥口令等关键数据的泄露，上至系统最高权限被攻击者夺取。由于这些原因，内存破坏型的漏洞今天仍然是最常被利用的漏洞之一。根据微软公司2015年的报告[1]，基于内存破坏的攻击方式占据了远程代码执行类CVE（Common Vulnerabilities and Exposures）的主导地位。

鉴于内存破坏攻击的严重性与广泛性，研究者在如何防范这类攻击上进行了大量的工作。一般来说，现有的防御措施可以分为两类：内存错误的检测和特定攻击形式的缓解对策。

第一类防御措施致力于检测内存破坏攻击成立的源头。由于内存破坏攻击需要依靠内存错误来触发，因此如果能够阻止内存错误的发生，那么就可以从根源上阻止内存破坏公积。但是为了根除内存错误，需要在软硬件上付出比较很大的额外开销，根据Szekeres等人的总结[2]，这类措施带来的额外开销可以达到50%甚至超过100%。而当设计者需要在性能与安全之间进行取舍时，往往是安全特性要为性能让位，因此大多数的内存错误检测措施只用在离线的漏洞检测或者崩溃分析中。

第二类防御措施则为特定形式的攻击提供针对性的防御措施，致力于阻止攻击者实行有害系统安全的行为。内存破坏攻击的形式可以分为四类：代码破坏、控制流劫持、数据导向攻击和信息泄露攻击。对于每一种攻击形式，研究者们开发了有针对性的防御手段。例如，数据执行保护（Data Execution Prevention，DEP）机制要求一块内存区域不得同时具有写权限和执行权限，从而阻止控制流劫持中的代码注入攻击。针对性的防护措施具有额外开销小的优点，通常这类措施只会带来大约10%或更低的额外开销。但是针对性措施也有其缺点，那就是只能防止一种类型的攻击，这就导致这类防御措施可以被绕开。

总的来说，内存破坏型攻击的防御措施有这样的特点：能够提供强有力安全保障的解决方案会拖慢系统速度，而效率更高的解决方案则只能提供有限的防护。

# 攻击类型

为了应对内存破坏攻击，我们首先需要了解内存破坏攻击是如何引发并利用内存错误的。下图是内存破坏攻击的一般攻击模型，其中每个矩形框中表示的是攻击中的一步，最下面一行的椭圆框中表示的是具体的攻击种类，菱形框则表示在具体攻击方式中的选择。



图 1 内存破坏攻击的攻击模型

## 制造内存错误

内存破坏攻击首先需要制造内存错误，即图1中的最初两步。第一步中攻击者将制造一个无效的指针，第二步中攻击者将使用这个指针进行读写，即“解引用”（dereference），从而触发一个内存错误。制造无效指针的方法有两大类，一类是从空间上令指针无效，也就是制造一个越界的指针；另一类是让指针在时间上无效，例如解引用一个指向被删除对象的指针，这样的指针称为悬垂指针。解引用越界指针所引发的内存错误称为“空间错误”，而由悬垂指针引发的内存错误则称为“时间错误”。

空间错误方面，攻击者可以利用多种程序漏洞来制造越界指针。例如，攻击者可以触发一次指针的分配失败，如果这个失败事件因为程序漏洞而没有受到检查，那么攻击者就得到了一个空指针，并可以进一步利用这个空指针在内核空间中展开攻击{Kemerlis, 2012 #158}。另一种常见的制造越界指针的方法是在循环中不断地增加或减小数组指针的值，如果程序中没有进行边界检查，或者边界检查不完整，那么攻击者就得到了一个超出数组边界的指针，这种方法称为缓冲区上溢出和缓冲区下溢出。攻击者还可以通过整数的溢出、舍入错误、符号错误或者错误的类型转换来制造非法的数组下标，从而引发一个指针越界访问。最后，攻击者还可以利用已经触发的内存错误来制造更多的越界指针，从而扩大攻击的规模，如图1中向后的循环所示。

时间错误方面，常见的制造悬垂指针的方法是利用异常处理程序或消息处理程序。例如在2011年发现的一个CVE中[CVE, 2011 #164]，攻击者可以改写“226 Transfer Complete”消息，从而在服务器的消息处理程序中触发一个漏洞。这个漏洞会导致指向应答消息的指针多于预期，在应答消息占用的空间被释放后，将会存在未被重新初始化的指针，利用这一指针攻击者可以在服务器上远程执行代码。在这个例子里，攻击者在对象已经被删除后再次“使用”了对象。由于这个特点，时间错误也被称为“use-after-free”漏洞，因为攻击者在悬垂指针指向的内存区域已经被释放后对指针进行了解引用。

制造出无效的指针后，攻击者需要对该指针进行读或写操作来触发内存错误，这是图1中的第二步。无论是越界指针还是悬垂指针，他们能够制造出的内存错误是相似的，能够进行的攻击也是相似的。只有一点限制，那就是对于悬垂指针来说，已经被释放的内存对象所占用的空间需要被新的内存对象重新使用。

1. 读无效指针

当程序使用一个攻击者所制造的无效指针进行读操作时，所读取到的数据可能是由攻击者所控制的数据，也可能是系统中的敏感数据，这取决于攻击者将要实行什么样的攻击。例如，考虑下面一段代码，攻击者可以在user\_input中制造一个越界指针，这个指针指向了位于jump\_table之外的代码片段，通过这种方式，攻击者改变了程序正常的控制流，使程序执行自己所希望的代码片段。

|  |
| --- |
| func\_ptr jump\_table[3] = {fn\_0, fn\_1, fn\_2};  jump\_table[user\_input](); |

攻击者也可以通过无效指针来直接读取敏感数据，从而直接泄露系统信息。一个典型的例子是利用printf函数的格式化字符串所进行的格式化字符串攻击。假如printf函数的格式字符串被攻击者所控制，那么攻击者可以构造一个特殊的格式化字符串，使得printf函数在根据格式化字符串构造参数表的过程中制造出无效的指针。举一个简单的例子，在下面的printf用法中，如果用户输入的是”%x”，那么printf就会将栈中的内容打印出来。这样攻击者便利用越界指针直接获取了程序栈中的内容。另外，在这种攻击方式中，攻击者也可以利用格式化字符串中的%n参数来对程序中的变量内容直接进行修改。

|  |
| --- |
| printf(user\_input); |

另一种通过读取无效指针来进行攻击的方式是制造一个悬垂指针，这个指针曾经指向的对象类型是A，在原来的内存对象被释放，内存空间重新分配后，这个指针指向一个由攻击者所控制的B类型对象。当程序按照原本的A类型对象调用虚函数时，查找的虚函数表内容其实是攻击者控制的新对象内容，这样一来，攻击者就可以通过操纵新对象内容来破坏虚函数的调用过程。另一种情况是，在重新分配的内存空间中包含了敏感数据，这时就可以利用A类型的指针直接对敏感数据区域进行读取，从而导致信息泄露。

1. 写无效指针

越界指针也可以用于修改内存中的内容。理论上来说，越界指针可以指向内存空间中的任何位置，因此如果攻击者将越界指针用于写操作，那么内存中的任何数据都有可能被攻击者修改，这包括了其他的指针甚至程序代码。但是在实际应用中，由于存在内存访问权限等各种限制，越界指针的攻击能力会受到一定的削弱，但仍然可以用于破坏其他指针从而制造更多的内存错误，其攻击能力仍然十分强大。

写越界指针的常见形式是缓冲区溢出攻击以及各种数组下标类的漏洞。攻击者可以利用缓冲区溢出攻击来覆盖函数的返回地址，从而将控制流转向由自己控制的代码区域；或者可以利用类似的攻击手段来破坏一个类的虚函数表，在这个类进行虚函数调用的时候劫持控制流。

# 参考文献

[1] RAINS T, MILLER M, WESTON D. Exploitation trends: From potential risk to actual risk; proceedings of the RSA Conference, F, 2015 [C].

[2] SZEKERES L, PAYER M, WEI T, et al. SoK: Eternal War in Memory; proceedings of the 2013 IEEE Symposium on Security and Privacy, F 19-22 May 2013, 2013 [C].