**软件漏洞分析技术读书报告**

姓名：仝志欣

|  |
| --- |
| **本周预期完成任务：**阅读软件漏洞分析技术的第十三章内容 |
| **本周实际完成进度：**基本看完智能灰盒测试，并做好读书笔记。了解了智能灰盒测试的一般原理和流程，分析示例，了解相关工具。 |
| **详细内容：**   1. **基本原理**   **1. 基本概念**  智能灰盒测试一般使用符号执行和污点分析等程序分析方法，大大增加了代码的覆盖能力，有针对性地检测某些安全敏感点的行为，大大增加了漏洞发现的概率。  在智能灰盒测试中，影响测试效率主要有两个关键因素：**一是通过动态符号执行等技术获得路径约束；二是对约束的求解。**  **2. 过程**  现代智能灰盒测试技术的基本思想是:采用中间表示、污点分析、符号执行、执行路径遍历、路径约束求解等技术手段,在不同程度上改进模糊测试的弊端。智能灰盒测试思想的出发点在于如何以尽可能小的代价找出程序中最有可能产生漏洞的执行路径集合,从而避免盲目地对程序进行全路径覆盖测试。通过外部输入点(接受不可信的输入数据的位置抵达内部敏感点(程序内部潜在漏洞的位置)的执行路径是最有可能产生漏洞的危险路径,而该路径的寻找过程可以采用逐步逼近的优化策略。然后再利用所需的测试路径集合对程序进行测试。  智能灰盒测试具体流程:①利用静态分析方法识别出程序内部的输入函数和敏感点;②通过搜索策略检测包含输入函数和敏感点的危险路径;③使用敏感点逼近序等算法筛选危险路径;④利用符号执行引擎求解危险路径的路径约束;⑤输出覆盖目标程序危险路径的测试集;⑥再对覆盖目标的测试集进行测试,以检查程序的安全性。具体可参考p332的流程图。   1. **方法实现**   **主体技术—动态符号执行；生成合适路径条件、准确生成测试用例的核心技术—路径控制与定向遍历、路径约束求解和漏洞触发数据生成。**   1. **动态符号执行**   动态符号执行技术的一种实现方法是将动态插装技术和符号执行技术结合起来,利用动态插装技术获取执行路径,然后对该路径进行符号执行,进而进行漏洞检测。同时,收集该路径上的路径约束集合,然后将该约束集合中的某个约束取反,获得另一个具体的路径输入。通过插装技术修改对应路径输人信息,进行下一次具体执行,在路径分支节点利用具体执行的路径选择信息来指导符号执行的路径选择。   1. **具体执行和符号执行**   现在更常使用用的是将两者混合使用，单独的具体执行速度快但是深度不够，而符号执行深度够但直接检测的速度较慢，因为很容易碰到不能确定的约束条件。   1. **动态符号执行原理**   动态符号执行是以具体数值作为输入来模拟程序代码，动态符号执行时通过不断地约束求解和修改约束条件进行不断迭代从而完成所有路径上的遍历。  主要有以下步骤：  加载目标程序,驱动程序开始执行,在程序的执行过程中从当前执行路径中收集所有的约束条件。  在程序执行完毕后对收集到的约東条件,按照一定的路径选择策略,对其中的某个约束条件进行取反,得到新的约束条件集,该约束条件集对应一条新的可执行路径。  采用约束求解器对新生成的约束条件集进行求解,产生新的测试用例,该用例可驱动程序执行新的可达路径。  由动态符号执行生成的新的测试用例,可引导程序执行不同的路径,在每一次的实际运行过程中,通过代码插装对程序运行时的状态信息进行监测,作为安全性分析的数据输入。   1. 动态符号执行状态建模   变量列表  路径条件  约束满足问题  线性规划问题   1. **路径控制与定向遍历**   **对于智能灰盒中的动态符号执行技术，最关键的部分是如何减少约束数和单个路径条件的规模，路径控制与定向遍历正是为了避免去执行和验证一些和漏洞不相关的路径，尽量使程序执行路径逼近指定的测试目标。**   1. **漏洞相关路径的识别**   程序中漏洞的存在形式,一般都会具有一定的特征,即漏洞模式,通过形式化的方法,对程序的漏洞模式进行形式化描述,继而总结岀漏洞模式的特征,然后利用模型检测等方法对程序进行分析,识别出漏洞模式中跟漏洞相关的路径,以便后续处理。  漏洞相关路径的形式化描述  漏洞模式。对一个特定程序的安全漏洞可以从多方面进行分类。本书从实际角度出发，将漏洞大致分为几类：文件操作类型、内存覆盖、逻辑错误、外部命令执行问题。  漏洞相关路径的识别方法：基于静态分析的识别方法、基于动态分析、基于动静态结合的识别方法（通常先获得测试程序的控制流图,然后将控制流图转换成一定的中间形式,如通过 PDL(procedure design language)语言描述程序执行流程,为动态分析等后续操作提供信息支撑。在动态分析过程中,则可以借助于全局信息,更快地找到漏洞相关的程序路径。）动静结合方法改善了单纯静态识别和动态识别的不足,但是也增加了程序的执行开销,也是目前研究的热点,仍有许多需要改进和优化的技术点。   1. 基于路径条件判断的路径控制   离线式路径条件判断  在线式路径条件判断  混合式路径条件判断：混合式路径条件判断结合在线式和离线式两种路径判断方法,在判断中需要进行混合模式符号执行。所谓混合模式符号执行是指采用离线和在线相结合的方法,使用操作系统的内存管理模式,当内存不足时,挂起一个正在执行的路径,保存该路径下次执行需要的内存状态、路径依赖条件等至外部存储空间。当内存空闲时,重新载入被挂起的路径,恢复状态,继续执行   1. 漏洞相关路径的定向遍历   一般说来,有三种方式:第一种是全遍历模式,如广度优先算法和深度优先算法,这种方式实现简单,但是也有明显的缺点,它会遍历许多的无关路径,影响目标测试效率;第二种方式是启发式算法,如以蚁群算法为代表的传统启发式算法,这种方式可以更好地遍历更底层的代码区域,不会像深度全遍历模式,由于循环等因素,出现无法结束的情况,但是由于它本身不具有目标针对性,只是为了遍历更多的路径,因此,可能会出现漏洞相关的路径并没有遍历或遍历很少的情况;第三种方式是现代智能逼近算法,该方法针对传统启发式方法针对性不足的缺点,通过域收敛等方式,确保漏洞相关路径能被优先遍历,该方法能有效地提高漏洞测试效率。  蚁群算法：是一种正反馈算法，蚂蚁最终能够找到最短路径，直接依赖于最短路径上的信息素积累，而信息素的积累是一个正反馈的过程。  智能逼近域收敛算法   1. 路径约束求解   约束求解是智能灰盒测试漏洞挖掘中的关键技术,它依赖于执行路径条件的收集、路径条件约束建模、约束表达式的构造和约束表达式的求解四个方面进行。约束求解本身是一个数学问题,约束求解的输入是约束条件合取范式,输出是可满足性的解。约束求解技术本身和漏洞没有直接关系。在漏洞分析中,利用符号执行生成路径条件,将条件进行规范化,形成约束求解器的标准输入即可。约束求解器内部如何构成是和漏洞没有关系,它只是给出一个判定是肯定还是否定,因此,如何构造约束条件将是和漏洞分析中最重要的部分。   1. 执行路径条件收集   不安全函数主要包含c函数中的一些没有判断输入长度的内存和字符串操作函数，比如：strcpy、strcat、sprintf等，针对参数量固定和不固定，又分为两类（详细见于p290）   1. 路径条件约束建模   路径条件是约束在符号执行中的一种具体形式，通常可以用一阶谓词逻辑公式描述。两类在路径条件约束满足问题中使用的数据结构：基于邻接表的数据结构、胀缩数据结构（详细见于p352和p353）   1. 约束问题表达式构造   包含程序基本块定义、“在同一层次内，mini(t(Bi)=2)—>Bi属于S且Bi是循环的的第一个基本块。”和堆溢出CSP表达式   1. 约束求解的主要技术   数学求解  搜索求解  约束推理技术  约束求解器   1. **漏洞触发数据生成** 2. 漏洞触发条件建模   缓冲区起初漏洞建模  对于堆溢出漏洞，可以用二元组（D,S）标识一个堆空间元素；而对于栈溢出漏洞，为了识别栈空间变量的边界，需要重构函数调用栈，可以以函数栈帧的边界作为局部变量的边界的估计。  整数溢出漏洞建模  i. 调用污点数据引入函数。发生整数溢出操作中,部分操作数和污点数据存在数据依赖关系,而这些污点数据来源于外界可控输入源,例如文件、网络报文、命令行参数等,并且由专z的AP函数引入。典型的污点数据引入函数包括read、 fread、recv  ii. 对污点数据不恰当检查。程序路径上没有检查污点数的范围或者范围检查不完备。  iii. 将溢出结果用于敏感操作。并非所有的整数溢出都会导致安全问题。只有当溢出结果影响了程序的控制流、内存分配、内存访问等操作时,才会进一步引起其他安全问题。   1. 漏洞触发条件与路径条件的求解  * 输入数据符号化 * 路径条件收集 * 漏洞触发条件检查 * 约束条件求解   (3) 触发漏洞的输入数据生成   1. **典型工具**   **1. Dart**  **2. Smart fuzzing**  **3. BitBlaze** |
| **相关文件及参考资料：** 软件漏洞分析技术的第十三章内容 |
| **遇到的问题及该阶段学习的想法：这章内容是读这本书以来直观感受最多最难懂的一章，需要查阅网上资料，并且通过对比，发现这本书有些地方说法较为老旧、晦涩难懂（本人认为）。但是也有很多收获，最多的可能就是智能灰盒测试将前面学到的很多知识进行了统筹，感受到了这些知识综合起来的强大。** |
| **接下来的进度计划：** 补充十三章、阅读第十四章 |