软件安全：

chapter3：调试分析工具

1、Pe
2、Pe 数据节分类：
3、加壳
4、加壳分类
5、虚拟内存 VA=RVA+imagebase
6、程序再定位 虚地址到实地址
7、Pe 文件注入
8、软件破解

Chapter4：软件漏洞

1、缓冲区
2、缓冲区溢出漏洞
3、栈溢出漏洞
4、堆溢出漏洞
5、Dword shoot
6、SEH 结构溢出漏洞
7、单字节溢出漏洞
8、整数溢出：不单独用于攻击，用于绕过目标程序的条件检测
9、存储/运算溢出 符号问题
10、虚函数攻击
11、

Chapter5：漏洞利用

1、Shellcode 核心是利用程序漏洞劫持进程控制权，实现控制流劫持，以便执行 shellcode
2、Exploit
3、Payload 触发漏洞 将控制权转移到目标程序
4、Shellcode 编码植入示例：
5、Shellcode 编码：
6、Shellcode 编码必要性：
7、编码方法：网页编码（base64）/二进制机器代码/异或编码

8、Windows 安全防护技术：
9、ASLR 关键地址随机化
10、GS STACK protection：在函数缓冲区和返回地址添加一个随机数，调用函数时检查这个数
11、DEP 数据执行保护 限制堆栈区代码为不可执行 （软件、硬件）

12、Safe SEH 检查 异常处理函数-合法 seh 函数表/ 异常处理句柄在不在栈上/有效性
13、SEH op 检测栈中 SEH 结构链表合法性

SEH： 异常处理句柄 handle 不在栈上
SEH 结构： 在栈上

14、**地址定位技术：**
15、静态 shellcode 地址 每次启动要用的程序 函数调用分配栈帧固定
16、 基于跳板指令
漏洞在动态链接库 调用时被动态加载 在内存中的栈帧地址是变化的
-解决：利用 jmp esp 指令作为跳板
-使缓冲区溢出，令 jmp esp 的指令地址 覆盖 返回地址 再覆盖相邻的高位置地址写入 shellcode

17、内存喷洒技术：（堆喷洒 heap spray）

DEP 功能可防范 heap spary

在 shellcode 前加入大量滑板指令 slide code（NOP），向系统申请大量内存，反复填充该代码段，再利用漏洞使程序跳转其中，最终执行 shellcode

18、API 函数自搜索：shellcode 自身具备
19、返回导向编程（ROP） 基于代码复用技术 从已有库/可执行文件提取指令片段 构建恶意代码
20、绕过其他安全防护： GS（没有保护异常处理器） ASLR（相同的 dll 本地攻击） SEH（利用未开启 SEH 保护的模块作为跳板绕过））

chapter7:漏洞挖掘

静态分析：词法分析 符号执行 五点分析
动态分析：模糊测试 符号执行 污点分析

4、**符号执行**：（代价小、效率高　　路径爆炸）

用符号值代替具体值 模拟程序执行

收集过程中语义信息 探索可达路径 分析错误

变量符号化 / 程序执行模拟（运算、分支语句）/ 约束求解

5、**污点分析**：（分析准确率高/输入参数漏洞　　--------　路径多 分析性能低）

标记程序中的数据（外部输入或内部数据）为污点 跟踪流程

静态：不执行程序 模拟进程
动态：执行程序 实时传播并检测污点标记

确定污点源　　->　　标记分析污点
（外部不可信数据）

污点源　/　　传播规则　　/　　污点检测
（污点扩散规则+清除规则）　　在程序执行敏感点进行污点判定
Sourse　　sinks

6、**词法分析**：（简单/性能高　　---------- 表面词法检测 漏报误报率高）
基于文本/字符的匹配分析

将 代码文本 和 归纳好的缺陷模式 匹配来发现漏洞

7、数据流分析：

数据沿程序执行路径流动的信息分析技术
分析程序执行路径上的数据流动/可能取值

过程内分析：函数内代码分析（ 流不敏感　　/　流敏感　　/　　路径敏感 ）
按行号　　按 CFG 拓扑排序　　实际可执行路径
程序控制流图

过程间分析：函数间数据流　（　上下文不敏感　　/　　上下文敏感　）

忽略调用位置/参数取值　　分析不同位置同一函数

【基于数据流的漏洞分析】
代码建模 -> 生成抽象语法树/CFG　-->漏洞分析规则分析

7、模糊测试（　　---------畸形数据生成随机　数据覆盖率不高）

漏洞检测技术）：

输入大量畸形数据来监测目标系统异常　{黑盒测试}

基于生成：按特定文件格式/协议规范　生成用例
基于变异：在合法用例基础上进行变异策略

步骤：（1）确定测试对象/输入数据
（2）生成测试数据
（3）检测测试数据
（4）监测程序异常

（5）确定可利用性

【智能模糊测试】
解决随机性 基于符号执行/污点分析 针对性进行测试数据生成
以最小代价生成最可能产生漏洞的执行路径集合

步骤：　反汇编 --》 中间语言切换 --》采用智能技术分析 数据/可执行路径关系 --》利用分析数据对执行路径进行测试

【AFL 模糊测试】
AFL：模糊测试工具 基于覆盖引导 记录输入样本的代码覆盖率　增加发现漏洞概率

流程：（1）源码编译插桩 记录代码覆盖率
（2）

Chapter8：漏洞挖掘进阶

1、程序切片

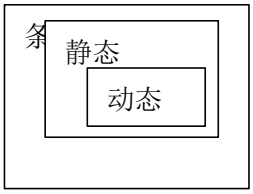
提取程序中满足一定约束的代码片段
前向（和运行方向一致）/后向

CFG 控制流程图：

程序依赖关系提取 --》 切片规则制定 --》切片生成

（控制/数据依赖）

静态切片 / 动态切片 / 条件切片
（特定输入）



2、程序插桩：

在程序中插入自定义代码
（断言是一种特殊插桩）

源代码 / 静态二进制 / 动态二进制
（运行前/收集信息处） （运行前/对编译后二进制机器码） （运行时/截获二进制指令）

3、程序 hook

过滤消息 在函数执行前先执行自己的挂钩函数 监控函数调用
过滤关键函数调用

调试法/注入法

消息 hook： 拦截单个 / 所有进程 消息
线程钩子 -----》 系统钩子

API hook： 对 API 函数进行 hook
IAT：导入表 EAT： 导出表
IAThook、代码 hook

【符号执行：续】

符号具体执行： 程序变量值、程序指令计数、路径约束条件 pc

路径条件分支选择情况 （初始化为 true）

遍历符号执行树

【符号传播】 建立符号变量传播关系、变化对应内存地址数据

[约束求解器]： SAT 求解器 / SMT 求解器
求解布尔表达式

Z3 是一个约束求解器

【污点分析】

Source / sinks / sanitizer
污点源 污点汇聚点 无害处理

识别污点源 --》 污点传播分析 --》 无害处理

（前提） **显式**流分析：how 跟随变量间**数据依赖**传播 （让数据不再产生危害）
隐式流分析：how 跟随变量间**控制依赖**关系传播

chapter10：web 安全基础

1、Web 基础：

http 协议：超文本传输协议

Html： **超文本标记语言**
Head body
网页信息 具体内容

Javascript： 直译式脚本语言

http 会话管理：
会话： 终端用户与交互系统进行通讯的过程
Session

多次 http 连接间维护用户与同一用户发出的不同请求之间关联的情况称为维护一个会话
http 协议是无状态的通信协议 本质因为 http 为短连接的

http 会话的实现机制：通过 cookie 与 session 来实现
浏览器 服务器

Web 编程语言：
Web 动态语言：

http 请求：
Form 表单：
与服务器信息交互的四种方法：增　删　改　查
Put delete post get

Post 安全性比 get 高

Php 语言：　解释性语言

Php 连接数据库：
连接数据库 --》 执行 sql 操作 --》 关闭连接

Cookie 和 session：

cookie 在客户端浏览器：
Session 保存在服务器：

Session id 保存在 cookie 中

十大 web 安全威胁：

注入、 跨站脚本、 遭破坏的身份认证和会话管理、（窃取用户访问 http 时的用户名和密码/用户会话　得到 session id 或用户身份信息　冒充用户）

不安全的直接对象引用、伪造跨站请求、安全配置错误、
不安全的加密存储、没有限制的 URL 访问、传输层保护不足和未验证的重定向和转发

会话劫持攻击：session ID 在生命周期内被窃取 登入目标账户
会话保持攻击：窃取 session 并一直保持一个有效的 session 成为一个永久的“后门”

Chapter11：web 渗透实战基础

1、文件上传漏洞：hacker 上传一个可执行文件（木马/病毒/恶意脚本/webshell）到服务器并执行；
【原理】 文件上传功能未严格限制用户上传文件后缀及类型

2、webshell： 以网页文件形式存在的一种 命令执行环境/网络后门

3、跨站脚本（XSS）攻击：

反射式跨站脚本 ： 非持久型/参数型 将恶意脚本附加到 url 地址参数

持久式跨站脚本 ： 存储式/持久型 利用 web 应用程序进行存储
（更威胁）

在 web 应用程序的网页中显示未经编码的攻击者脚本

存储式 xss 脚本不是来自 web 应用程序请求

4、【SQL 注入漏洞】

原理：将 sql 代码插入到输入参数 再将参数传递到后台

漏洞存在原因： 网页对输入信息未做有效筛查和处理

Chapter12： 文件包含漏洞：

本地文件包含 / 远程文件包含
合法文件、日志文件 攻击者服务器上的文件

序列化：对象/数组等数据结构转化为可存储格式
对象-》字符串

【Php 反序列化漏洞】 ： 应用中传给 unserialize 的参数是用户可控 hacker 可精心构造一个序列化字符串 利用 php 魔术方法控制对象内部变量/函数