软件安全—软件漏洞机理与防护

V8 软件安全设计与构建

陈泽茂

武汉大学国家网络安全学院

chenzemao@whu.edu.cn

提纲

8.1 软件安全开发概述

8.2 软件安全需求分析

8.3 软件安全设计

8.4 软件安全编码

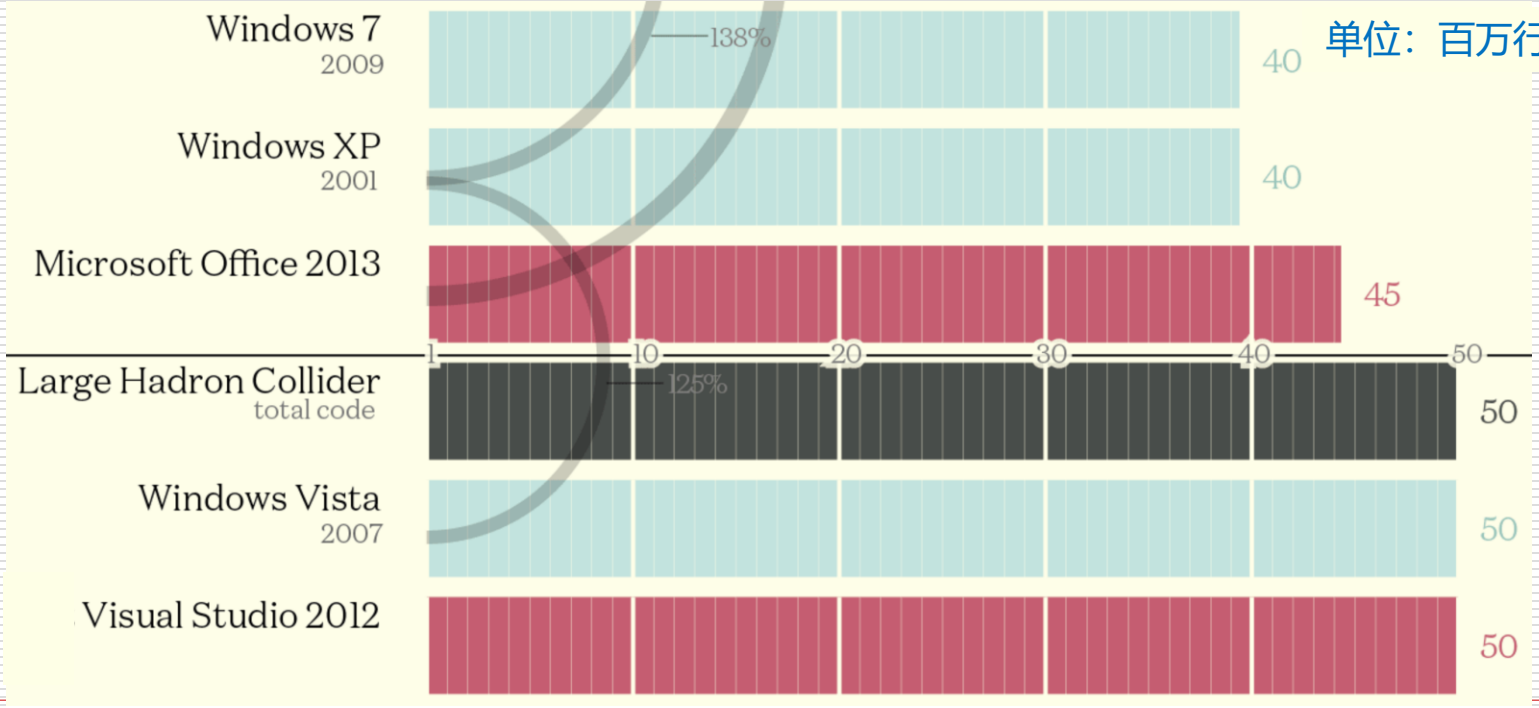
8.5 软件安全测试

8.1.1 软件漏洞广泛存在

- Web浏览器的千行漏洞数
  - Google Chrome: 380 CVE in 6 239 930 LoC（0.06 个漏洞/千行代码)
  - Firefox: 395 CVE in 8 000 969 LoC (0.05个漏洞/千行代码）
- 开源编程语言的千行漏洞数
  - Python: 3 exploitable CVSS>=7 in 862830 LoC（0.003个漏洞/千行代码)
  - PHP: 122 exploitable CVSS>=7 in 3761587 LoC（0.03个漏洞/千行代码)

https://security.stackexchange.com/questions/21137/average-number-of-exploitable-bugs-per-thousand-lines-of-code

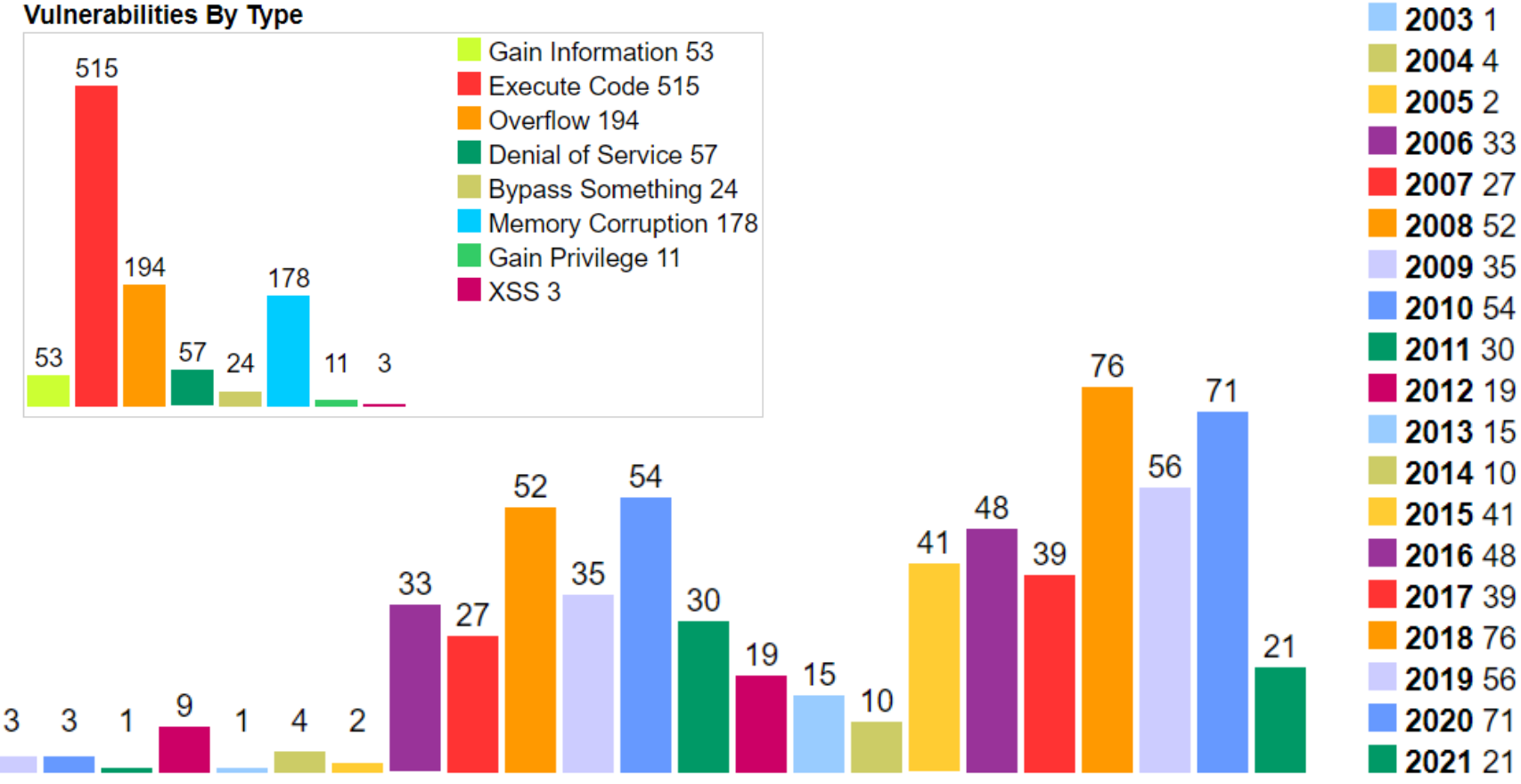
8.1.1 软件漏洞广泛存在



来源：https://informationisbeautiful.net/visualizations/million-lines-of-code/

MS Office漏洞数量统计

https://www.cvedetails.com/product/320/Microsoft-Office.html?vendor_id=26



8.1.1 软件漏洞广泛存在

- 软件漏洞的生成
  - 分析设计阶段：缺乏风险分析，引用不安全的对象，强调易用和功能、性能。
  - 开发实现阶段：开发人员在技术实现中有意或者无意引入缺陷。
  - 配置运维阶段：安全配置不当


ICS 35.040
L 80

GB

中华人民共和国国家标准

GB/T 33561—2017

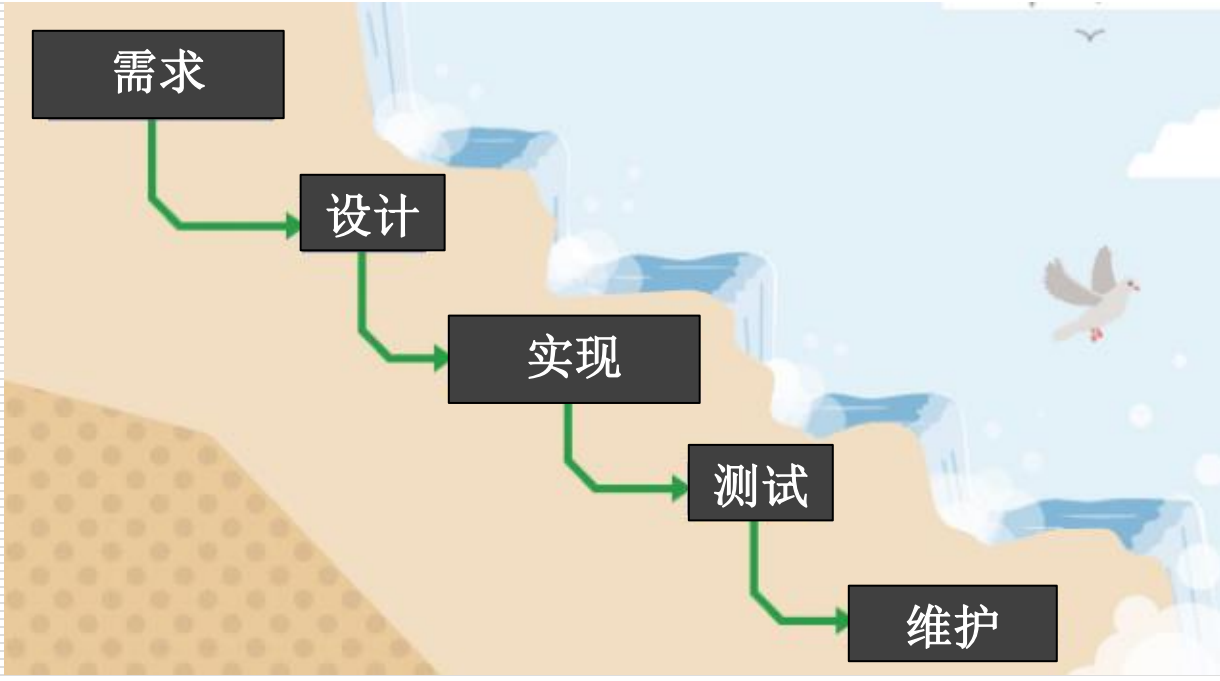
信息安全技术　安全漏洞分类

Information security technology—Vulnerabilities classification

8.1.2 传统软件开发模型

- 瀑布模型

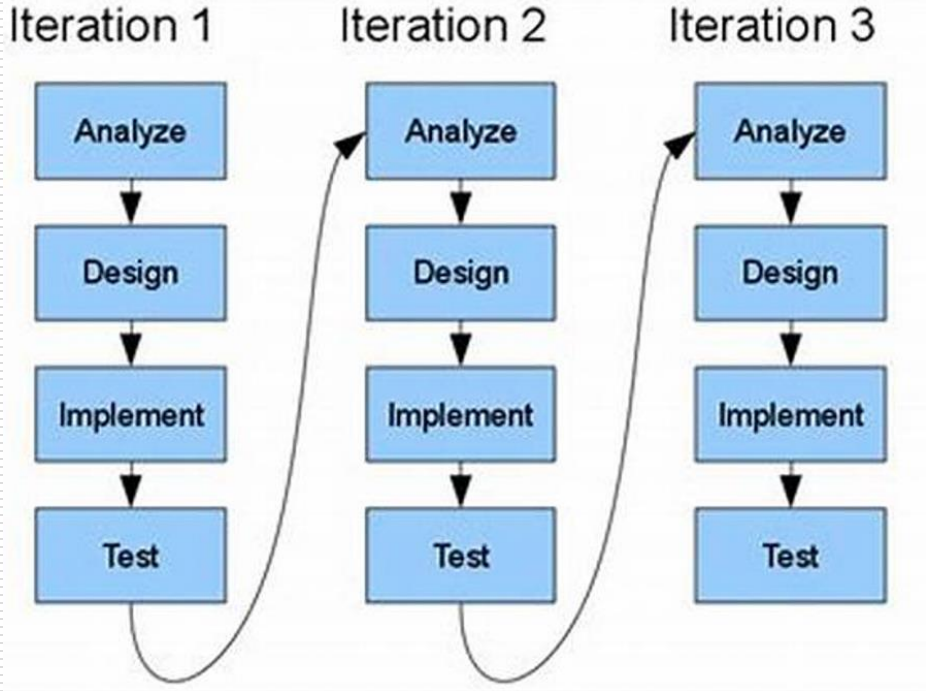
  将软件生命周期按顺序划分为相互连接的若干阶段，直至得到最终的软件产品。



8.1.2 传统软件开发模型

- 迭代模型

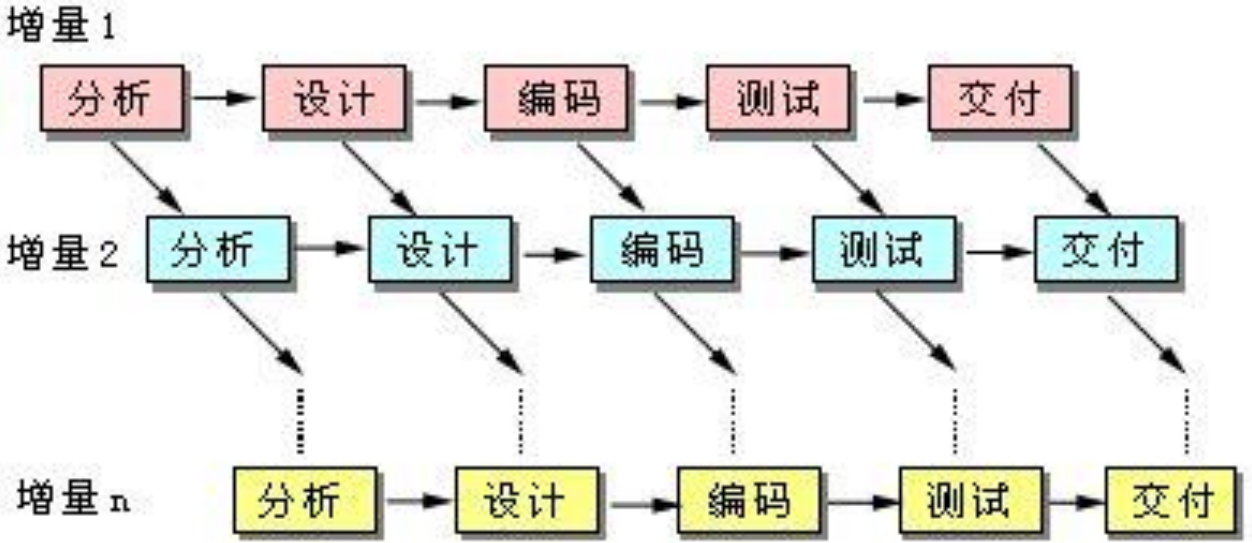
  按软件的细化程度来划分软件产品的不同阶段。先将产品的整个框架建立起来，然后随着项目的推进，不断细化或完善。



8.1.2 传统软件开发模型

- 增量模型

把要开发的软件模块化，每个模块作为一个增量，分批次地分析、设计、编码和测试这些增量组件。



8.1.2 传统软件开发模型

- 传统软件开发模型的特点：以产品功能为中心
  - 需求分析：以按时交付用户所需功能为主要目标
  - 软件设计：假设来自相关模块的数据可信
  - 软件编码：力图正确、高效地实现功能
  - 软件测试：确认软件产品与软件需求、设计规格和编码规范等开发要求的一致性。

8.1.2 传统软件开发模型

- 传统软件开发模型的局限：软件安全文化缺失
  - 市场将按期交付和软件功能作为衡量软件开发工作的主要指标
  - 用户方没有施加安全方面的压力
  - 开发者缺乏安全培训

传统软件开发模型的局限性

https://www.synopsys.com/blogs/software-security/software-security-training-resources-infographic/

8.1.3 软件开发的安全保证思路

- 在软件开发生命周期中，开发后期改正错误的代价远高于开发早期。

Table 5-1. Relative Cost to Repair Defects When Found at Different Stages of Software Development (Example Only)

X is a normalized unit of cost and can be expressed terms of person-hours, dollars, etc.

Requirements Gathering and Analysis/ Architectural Design	Coding/Unit Test	Integration and Component/RAISE System Test	Early Customer Feedback/Beta Test Programs	Post-product Release
1X	5X	10X	15X	30X

NIST: The Economic Impacts of Inadequate Infrastructure for Software

8.1.3 软件开发的安全保证思路

- 基本思路：在软件开发生命周期的各阶段进行风险评估和风险管理，做到安全问题早发现早解决。
- 软件开发各阶段的安全工作
  - 需求阶段：分析软件安全需求
  - 设计阶段：遵循安全设计原则
  - 编码阶段：遵循安全编码规范
  - 测试阶段：检验安全需求、安全设计、安全编码规范得到正确实施。

提纲

8.1 软件安全开发概述

8.2 软件安全需求分析

8.3 软件安全设计

8.4 软件安全编码

8.5 软件安全测试

8.2.1 软件安全需求

- 软件需求
  - 功能需求
    - 开发人员必须在软件产品中实现的供用户使用的功能。
  - 非功能需求
    - 不直接提供具体功能，但与系统总体特性相关
    - 质量属性要求：易用性，可靠性，时效性，安全性，…
    - 约束性要求：时间约束，空间约束，标准遵循，…

8.2.1 软件安全需求

- 软件安全需求的概念
  - 安全功能需求：为了实现用户的安全目标，软件应该做什么；
  - 安全保证需求：应该如何提高软件的安全性，消除或减少软件的安全脆弱性。
  - 合规类需求：指南遵从，标准遵从，法律遵从。

8.2.1 软件安全需求

- 软件安全需求的特点
  - 安全需求不一定来自软件用户的要求或兴趣
  - 安全需求分析不能局限于软件本身，而要立足系统全局
  - 软件安全需求具有适度性
    - 安全机制的经济性原则
    - 防护措施与业务重要性相适应

8.2.2 软件安全需求分析方法

- 基本步骤
  - 理解分析对象
  - 进行风险评估
    - 分析软件的脆弱点
    - 分析软件面临的安全威胁
    - 定性或定量计算软件面临的风险
  - 确定安全需求

8.2.2 软件安全需求分析方法

- 基于威胁建模的分析方法
  - 识别威胁，评估风险。
  - 确定必要安全机制，确保威胁环境下业务目标的实现。
  - 平衡软件的安全性与可用性。
  - 形成软件安全需求报告

8.2.3 威胁建模

- 威胁建模步骤
  - 理解要分析的软件
  - 识别软件面临的威胁
  - 量化软件面临威胁的风险级别
  - 提出风险消减措施

8.2.3.1 理解要分析的软件

- 理解软件自身的体系结构
- 分析软件与外部实体的交互
- 绘制数据流图
- 识别攻击者可能感兴趣的资产

8.2.3.2 识别软件面临的威胁

- 应用安全框架（ASF）分析法：从防御方的视角识别威胁

类别	脆弱性举例	威胁举例
审计与日志	未进行登录审计	用户抵赖其操作
认证	使用弱口令	字典攻击
授权	权限分离不充分	数据篡改
配置管理	管理员太多	未授权使用管理界面
加密	密钥长度太短	加密被破解
异常管理	向客户端泄露过多信息	泄露系统敏感信息
输入和数据验证	用未验证输入生成SQL查询	SQL注入攻击
敏感数据	在代码中存储秘密信息	信息泄露
会话管理	允许延长会话生命期	会话重放攻击

8.2.3.2 识别软件面临的威胁

- STRIDE分析法：从攻击者视角识别威胁

威胁	安全属性	定义	举例
仿冒（S）	认证	冒充人或物	冒充其他用户账号
篡改（T）	完整性	修改数据或代码	修改订单信息
抵赖（R）	审计	不承认做过某行为	不承认修改行为
信息泄露（I）	保密性	信息被泄露或窃取	用户信息被泄露
拒绝服务（D）	可用性	消耗资源、服务可不用	DDOS 导致网站不可用
权限提升（E）	授权	未经授权获取、提升权限	普通用户提升到管理员

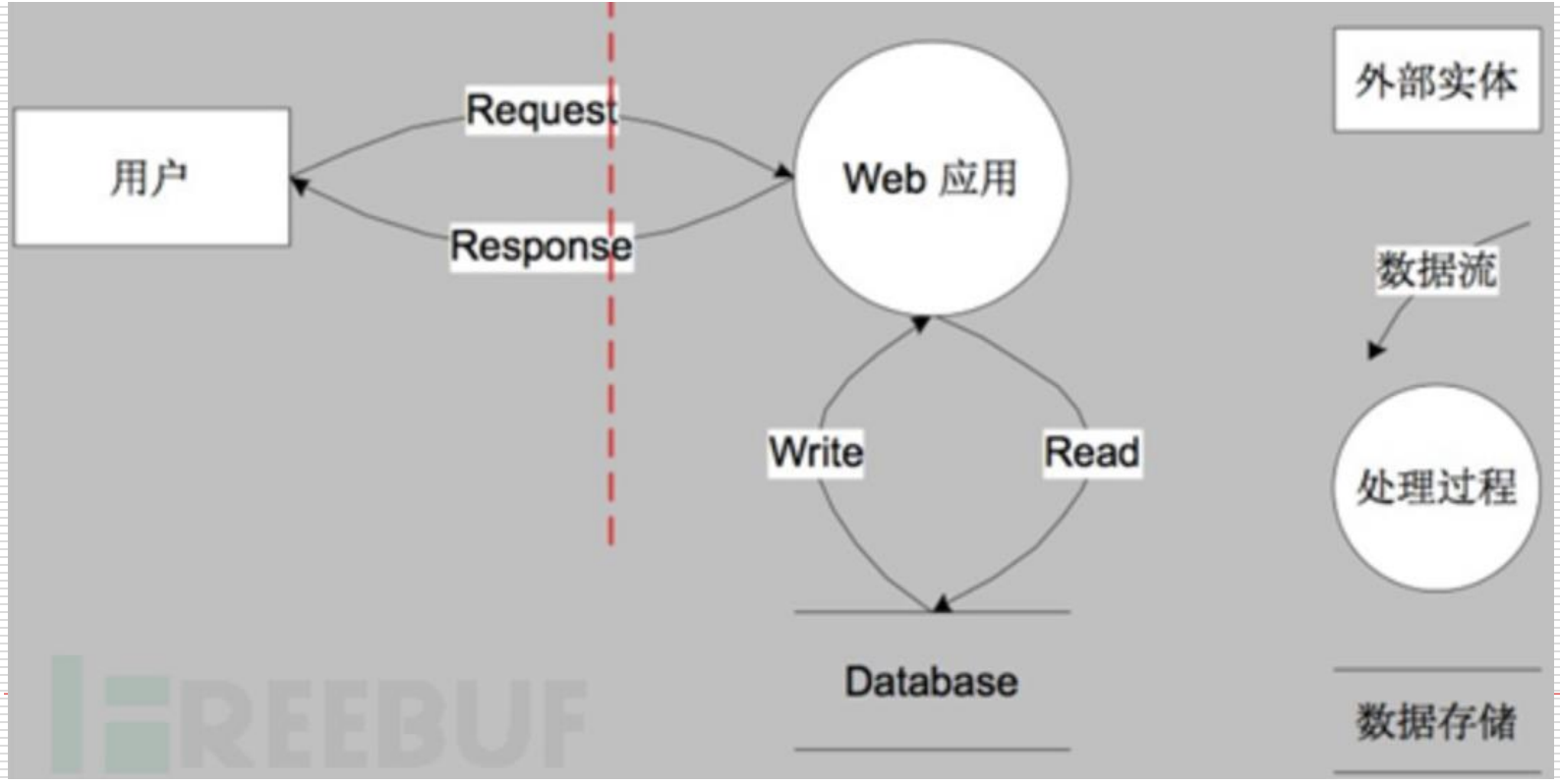
https://www.freebuf.com/articles/es/205984.html

8.2.3.2 识别软件面临的威胁

- STRIDE分析法
  - 绘制数据流图

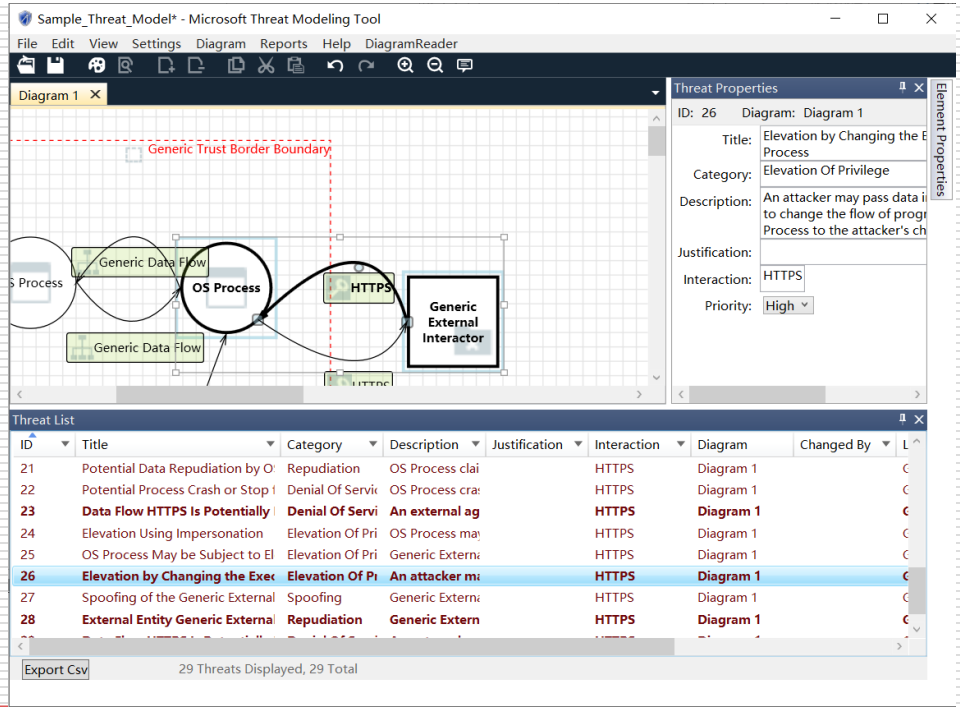
a) 外部实体：系统控制范围之外的用户、软件或设备
b) 处理过程（进程）：任务或执行过程
c) 数据存储：存储数据的内部实体
d) 数据流：外部实体与处理过程，处理过程与处理过程，或者处理过程与数据存储之间的交互

数据流图示例



8.2.3.2 识别软件面临的威胁

- STRIDE分析法
  - 分析数据流图中每个元素面临的威胁
  - 微软Threat Modeling Tool可以提供帮助。



8.2.3.3 量化软件面临的威胁

- DREAD法：用于评估STRIDE所识别的威胁
  - D - Damage potential：威胁被利用后的潜在破坏性

    破坏性从小到大，可选5个取值：0, 5, 8, 9, 10
  - R - Reproducibility：威胁利用的可复现性

    复现难度从难到易，可选4个取值：0, 5, 7.5, 10
  - E - Exploitability：威胁利用的难度

    利用难度从难到易，可选4个取值：2.5, 5, 9, 10

https://haiderm.com/application-threat-modeling-using-dread-and-stride/

8.2.3.3 量化软件面临的威胁

- A - Affected Users：受该威胁影响用户的数量

  受影响用户数从少到多，可选5个取值：0, 2.5, 6, 8, 10

- D - Discoverability：发现该威胁的容易程度

  威胁的发现难度，从难到易，可选4个取值：0, 5, 8, 10

- 基于DREAD的威胁风险值计算
- Risk = (D + R + E + A + D) / 5
- Risk值在0~10之间，越大则表示该威胁的风险越高。

8.2.3 威胁建模

- 威胁建模步骤
  - 理解要分析的软件
  - 识别软件面临的威胁
  - 量化软件面临威胁的风险级别
  - **提出风险消减措施**

提纲

8.1 软件安全开发概述

8.2 软件安全需求分析

8.3 软件安全设计

8.4 软件安全编码

8.5 软件安全测试

8.3.1 概述

- 软件安全设计的重要性

Design flaws account for 50% of software security defects in most software.

——Gary McGraw

https://www.darkreading.com/application-security/building-security-into-software/a/d-id/1338067

8.3.1 概述

- 软件安全设计原则
  - 攻击面最小化原则
  - 最小权限原则
  - 纵深防御原则
  - 默认安全原则
  - 失效安全原则
  - 开放设计原则
  - 简单原则

8.3.2 攻击面最小化原则

- 攻击面（Attack Surface）的概念
  - 信息系统的攻击面：攻击面是信息系统的特征，它允许攻击者探测、攻击信息系统，或维持其在信息系统中的存在。

A Glossary of Common Cybersecurity Terminology.
https://niccs.cisa.gov/about-niccs/cybersecurity-glossary

8.3.2 攻击面最小化原则

- 软件的攻击面
  - 软件中可被本地或远程的未授权用户访问的所有功能。比如：软件的用户输入界面，软件为用户提供的各种服务，软件与外部系统的交互协议，等。
  - 软件每多一个功能，其攻击面便扩大一分，所面临的安全风险便增加一分。

8.3.2 攻击面最小化原则

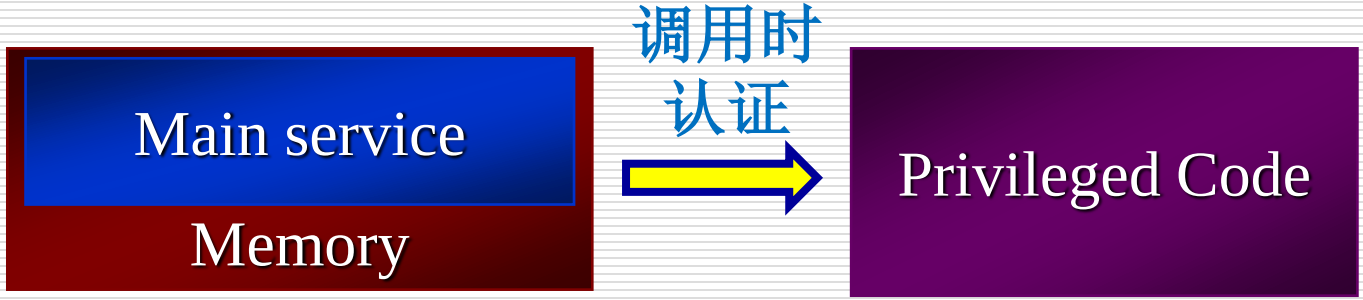
- 攻击面最小化：尽量减小攻击面，从而降低被攻击概率。
- 如何减小软件的攻击面
  - 去掉非必要的软件功能
  - 禁用非必要的远程访问途径
  - 实施认证访问，禁用匿名访问权限
  - 避免向外部暴露程序内部处理过程
  - 减少与外部环境的交互
  - 实施最小权限管理

8.3.3 最小权限原则

- 仅授予用户（进程）执行任务所必需的权限
  - 评估应用程序功能所需的最低权限，进行最少授权.
  - 如果个别情形需要较高权限，可以通过权限撤销机制，即在需要时授予，任务完成后撤销.

8.3.3 最小权限原则

- 实例：Windows XP之后重构服务提升安全性.
  - 若服务不执行特权操作，则直接移除其LocalSystem权限。
  - 否则，将服务重构为两部分：主服务、特权代码。



8.3.4 纵深防御原则

- 纵深防御（defense in depth）
  - 将信息系统划分成局域计算环境、区域边界、网络传输、支撑性基础设施等4个域，实现对信息系统的多层防护。
  - 多层次布防，多种安全方案并举、配合，形成整体。
  - 不是安全机制的简单叠加，而是在正确的地方部署正确的安全方案。

8.3.4 纵深防御原则

- 实例：多种手段并用提升用户认证的安全性
  - 用户名/口令
  - IP地址验证
  - Captcha验证
  - 登录审计
  - 蛮力攻击检测

8.3.5 软件安全设计的其它原则

- 失效安全原则
  - 在出错情形下，不能给予用户额外权限，也不能向用户展示敏感信息。
  - 反例：Web应用把后台异常的详细信息暴露给前端用户。

8.3.5 软件安全设计的其它原则

- 默认安全原则
  - 在默认设置下，软件必须是安全的。
  - 用户熟悉安全配置后，可根据自身情况调整安全设置。
  - 实例
    - Win 7之后，DEP默认开启，但用户可改变DEP策略。
    - Win 10默认启用Windows Defender，但用户可关闭。

8.3.5 软件安全设计的其它原则

- 开放设计原则
  - 安全性不能依赖设计中的秘密.
  - 反例：将登录口令、密钥等秘密信息硬编码在程序中。
- 简单原则
  - 设计越复杂，程序实现越容易出错.
  - 功能越复杂，用户操作越容易出错.

提纲

8.1 软件安全开发概述

8.2 软件安全需求分析

8.3 软件安全设计

8.4 软件安全编码

8.5 软件安全测试

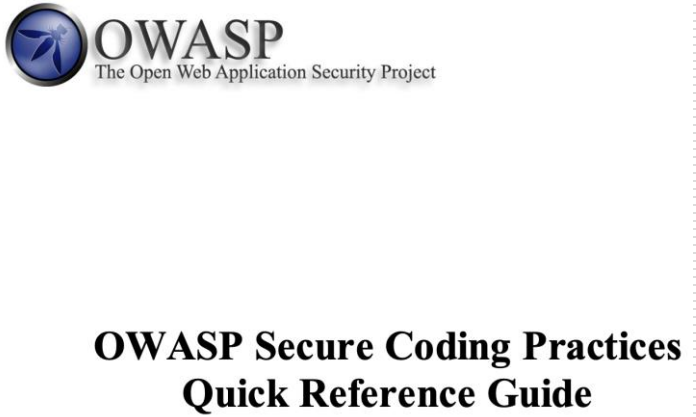
8.4.1 概述

- 不安全编码示例

```
int main(int argc, char *argv[])
{
    char dst[128] = {0x00};
    if (argc > 1)
        strcpy(dst, argv[1]);
    return 0;
}
```

8.4.1 概述

安全编程规范



8.4.1 概述

- 安全编码原则
  - 验证输入
  - 弃用不安全的函数
  - 谨慎处理错误
  - 确保代码生成安全

8.4.2 验证输入

- 对所有外部数据的可信性持怀疑态度
  - 所有输入数据都要进行安全检查后方能使用。
- 常见输入
  - 用户输入
  - 配置参数
  - 网络数据
  - 进程间通信
  - 文件

8.4.2 验证输入

- 输入验证的重点
  - 可能引发缓冲区溢出的输入：数组下标，内存偏移，内存分配的尺寸，数据拷贝长度；文件的内容和格式，等
  - 可能影响代码执行路径的输入：直接或间接影响循环、分支等判断条件，...
  - 可能导致非预期命令执行的输入：作为命令执行参数，用于拼装SQL语句，用于构造格式化字符串，...

8.4.3 弃用不安全的函数

- 不安全函数举例

  C标准库中的许多函数，未检查目标缓冲区的大小，很容易导致缓冲区溢出漏洞。

字符串拼接	strcat, wcscat, strncat
字符串拷贝	strcpy, wcscpy, strncpy, wcsncpy
内存拷贝	memcpy, wmemcpy, memmove, wmemmove
字符串格式化	sprintf, swprintf, wsprintf, wnsprintf

Security Development Lifecycle (SDL) Banned Function Calls
https://docs.microsoft.com/en-us/previous-versions/bb288454(v=msdn.10)?redirectedfrom=MSDN

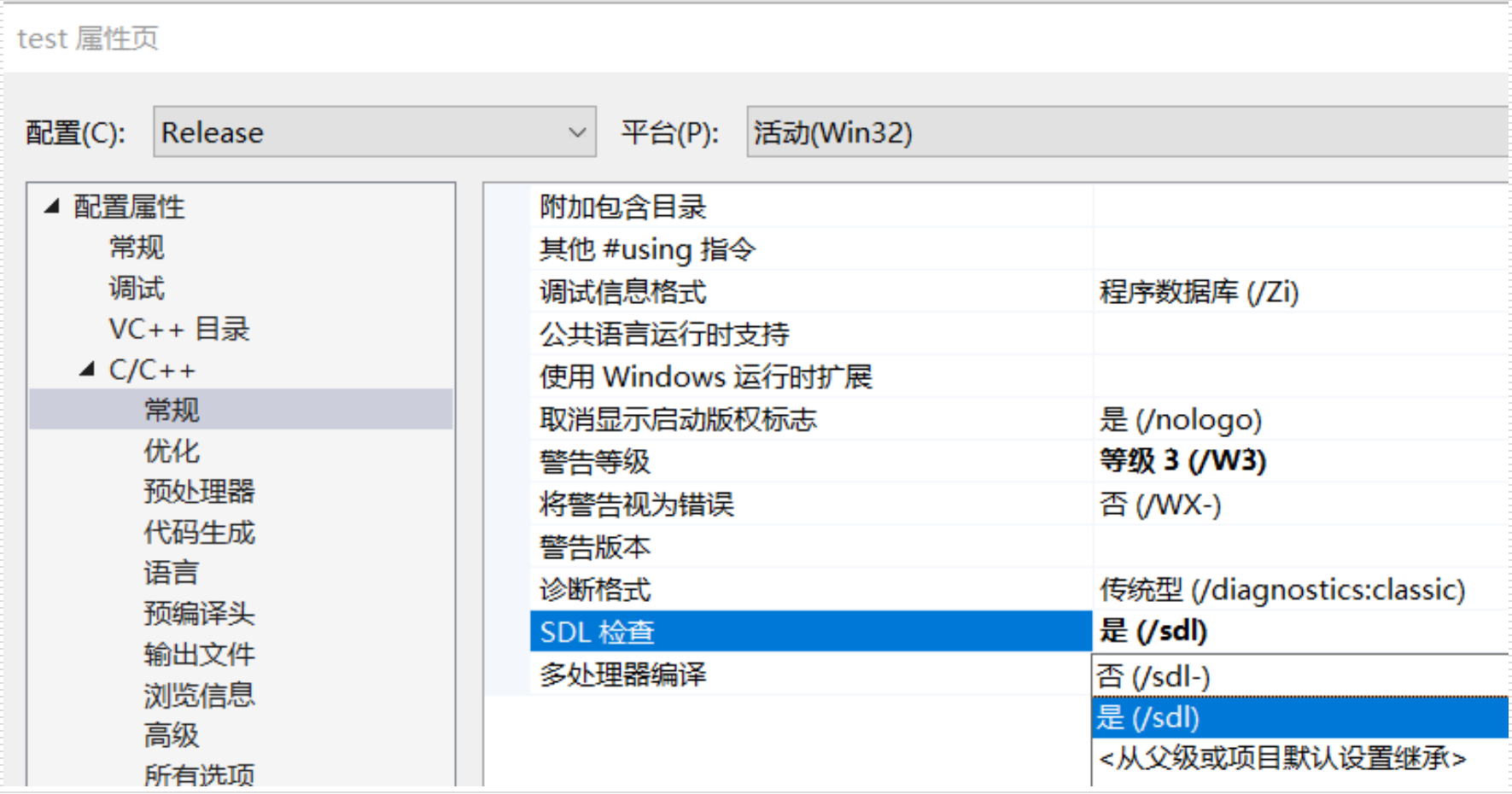
8.4.3 弃用不安全的函数

- 检查代码中是否使用了禁用函数。工具：
  - #include ”banned.h”
    - 在banned.h中列出要禁用的函数，例：

```
pragma deprecated (memcpy, RtlCopyMemory, CopyMemory, wmemcpy)
pragma deprecated (lstrlen)
```

    - https://github.com/x509cert/banned/blob/master/banned.h

启用编译器中的检查选项



stdafx.h test.cpp

test (全局范围) main()

```
1   // test.cpp : 定义控制台应用程序的入口点。
2   //
3
4   #include "stdafx.h"
5   #include <string.h>
6
7   int main()
8   {
9       char buf[128];
10      strcpy(buf, "12345678");
11      return 0;
12  }
```

174 %

输出

显示输出来源(S): 生成

```
1>------ 已启动全部重新生成: 项目: test, 配置: Debug Win32 ------
1>stdafx.cpp
1>test.cpp
1>d:\myproj\test\test\test.cpp(10): warning C4996: 'strcpy': This function or variable may be unsafe. Consider using strcpy_s instead.
1>c:\program files (x86)\windows kits\10\include\10.0.15063.0\ucrt\string.h(136): note: 参见“strcpy”的声明
```

8.4.3 弃用不安全的函数

- 用安全版本的函数来替代对应的不安全函数.
  - 方案一：使用Windows定义的安全字符串函数。

Strsafe.h function	Replaces
StringCchCatEx	strcat, wcscat
StringCchCopy, StringCchCopyEx	strcpy, wcscpy, lstrcpy
StringCchPrintf, StringCchPrintfEx	sprintf, wsprintf, wnsprintf
……	……

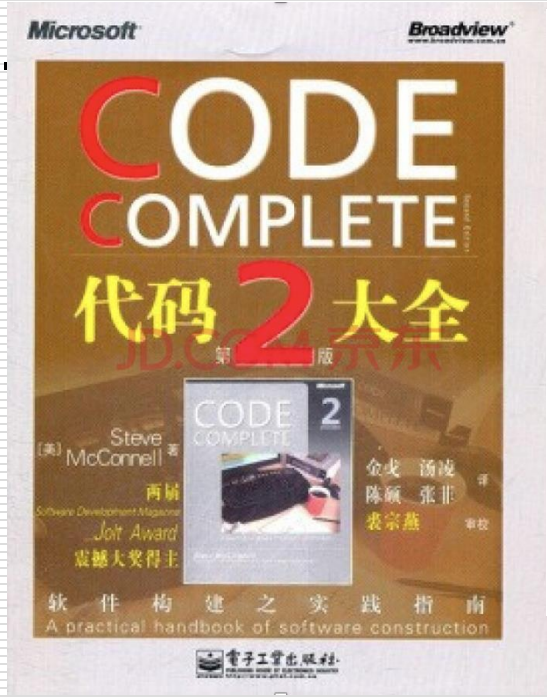
  - 方案二：使用ISO/IEC TR 24731-1定义的安全字符串函数，如strcpy_s、strcat_s、sprintf_s等。

8.4.4 谨慎处理错误

- 尽可能把握所有可能的代码出错环节
  - 用错误码来检查预期可能发生的错误
  - 检查函数调用的返回值，正确处理各种可能情况。
- 最小化向客户端反馈的错误信息
  - 不要泄露系统和软件的配置、版本等信息
  - 不要泄露后台文件和目录的绝对路径
  - 不要泄露口令、重要业务数据等敏感数据

8.4.4 谨慎处理错误

Steve McConnell. 代码大全(第2版). 金戈, 汤凌, 陈硕, 张菲, 译. 北京: 电子工业出版社, 2011.



第8章 防御式编程

8．1 保护程序免遭非法输入数据的破坏

8．2 断言

8．3 错误处理技术

8．4 异常

8．5 隔离程序，使之包容由错误造成的损害

8．6 辅助调试的代码

8．7 确定在产品代码中该保留多少防御式代码

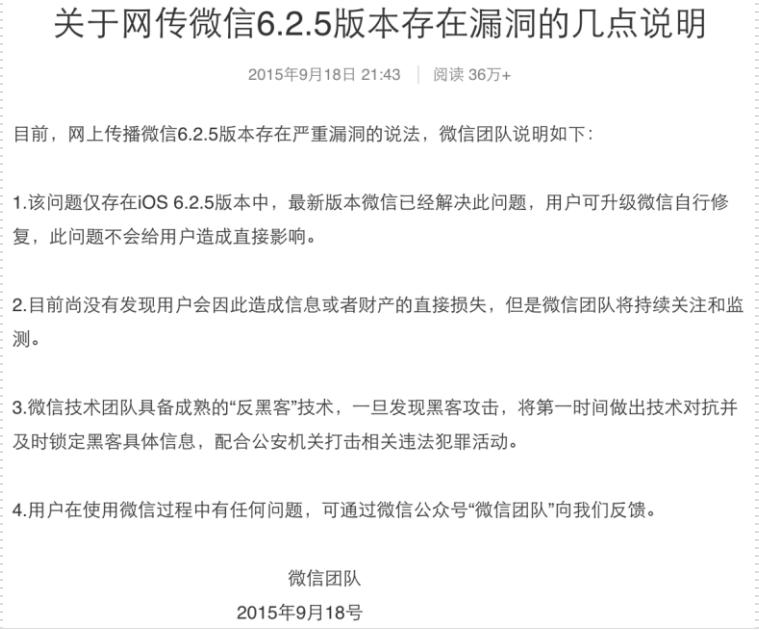
8．8 对防御式编程采取防御的姿态

8.4.5 确保代码生成安全

- 使用可信的开发工具

  例：XcodeGhost病毒事件。

  恶意代码作者在Xcode安装包中植入后门，部分开发者安装了这个“李鬼” Xcode，导致编译出的APP中包含恶意代码。



关于网传微信6.2.5版本存在漏洞的几点说明

2015年9月18日 21:43 | 阅读 36万+

目前，网上传播微信6.2.5版本存在严重漏洞的说法，微信团队说明如下：

1.该问题仅存在iOS 6.2.5版本中，最新版本微信已经解决此问题，用户可升级微信自行修复，此问题不会给用户造成直接影响。

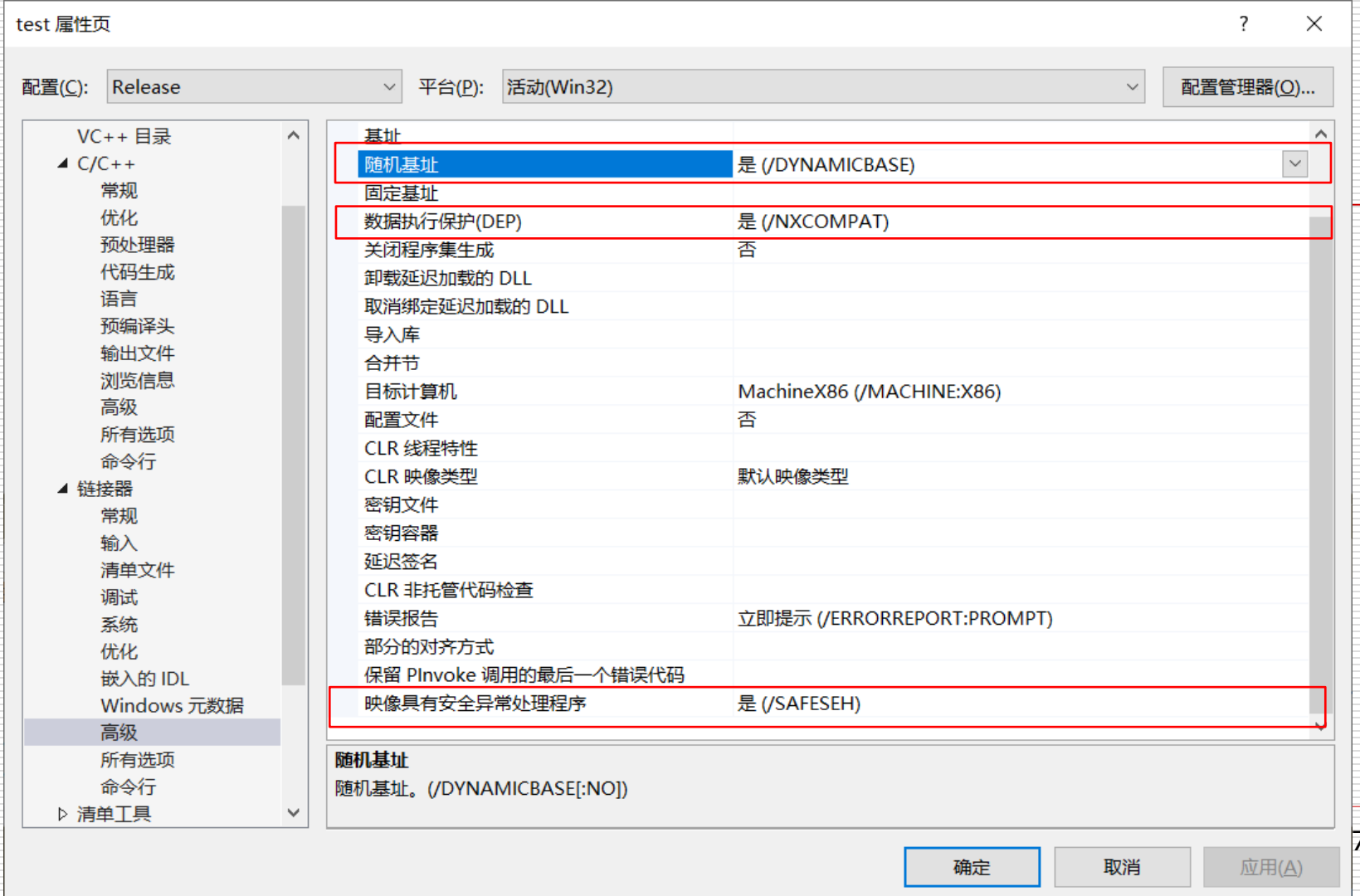
2.目前尚没有发现用户会因此造成信息或者财产的直接损失，但是微信团队将持续关注和监测。

3.微信技术团队具备成熟的“反黑客”技术，一旦发现黑客攻击，将第一时间做出技术对抗并及时锁定黑客具体信息，配合公安机关打击相关违法犯罪活动。

4.用户在使用微信过程中有任何问题，可通过微信公众号“微信团队”向我们反馈。

微信团队

2015年9月18号


test 属性页
配置(C): Release
平台(P): 活动(Win32)
配置管理器(O)...
VC++ 目录
C/C++
常规
优化
预处理器
代码生成
语言
预编译头
输出文件
浏览信息
高级
所有选项
命令行
链接器
常规
输入
清单文件
调试
系统
优化
嵌入的 IDL
Windows 元数据
高级
所有选项
命令行
清单工具
基址
随机基址 是 (/DYNAMICBASE)
固定基址
数据执行保护(DEP) 是 (/NXCOMPAT)
关闭程序集生成 否
卸载延迟加载的 DLL
取消绑定延迟加载的 DLL
导入库
合并节
目标计算机 MachineX86 (/MACHINE:X86)
配置文件 否
CLR 线程特性
CLR 映像类型 默认映像类型
密钥文件
密钥容器
延迟签名
CLR 非托管代码检查
错误报告 立即提示 (/ERRORREPORT:PROMPT)
部分的对齐方式
保留 PInvoke 调用的最后一个错误代码
映像具有安全异常处理程序 是 (/SAFESEH)
随机基址
随机基址。(/DYNAMICBASE[:NO])
确定
取消
应用(A)

提纲

8.1 软件安全开发概述

8.2 软件安全需求分析

8.3 软件安全设计

8.4 软件安全编码

8.5 软件安全测试

8.5.1 概述

- 软件测试
  - 使用人工或自动的手段来运行或测定某个软件系统的过程
  - 检验软件是否满足规定的需求，或弄清预期结果与实际结果之间的差别。
  - 依据用户需求和功能规格来设计测试用例。

8.5.1 概述

- 软件安全测试
  - 安全功能测试：确定软件安全特性的实现是否与设计预期一致
  - 安全漏洞测试：查找软件实现中存在的安全缺陷，检验软件的抗攻击能力。
- 软件安全测试方法
  - 源代码审查
  - 模糊测试

8.5.2 源代码安全审查

- 以人工和/或自动化的方式，对软件源代码进行系统性的安全检查，以找出其中与安全相关的缺陷。
- 重点关注以下方面的潜在缺陷
  - 认证，授权，加密，日志，安全配置
  - 会话管理，数据验证，错误处理
- 审查方式
  - 人工审查
  - 工具审查

8.5.2 源代码安全审查

- 人工审查
  - 要求审查人员精通代码编程语言，有安全编程经验，了解安全防护机制.
  - 适用于检查对象
    - 高风险组件
    - 安全基础组件：密码算法，安全协议，等.
    - 处理或存储敏感和隐私数据的组件

8.5.2 源代码安全审查

- 工具审查
  - 工具审查规则的制定依据
    - 软件漏洞库：公共漏洞库，设备或软件厂商的漏洞库
    - 软件漏洞报告：OWASP的年度Top10
    - 软件安全编码规范

源代码安全审查工具

- Fortify SCA（ Fortify Static Code Analyzer）
  - 8个脆弱性分析器：缓冲区，配置，内容，控制流，数据流，语义，结构和高阶等分析器。
  - 分析规则：每类脆弱性的分析器都有对应的脆弱性检测规则.

https://www.microfocus.com/en-us/cyberres/application-security/static-code-analyzer

源代码安全审查工具

- RIPS
  - PHP脚本代码静态分析器
  - 能够检测XSS、SQL注入、文件泄露、本地/远程文件包含等Web漏洞。

http://rips-scanner.sourceforge.net/

源代码审查工具

- 阿里巴巴Java开发规范静态检查插件

alibaba / p3c https://github.com/alibaba/p3c Watch 1.2k Sta

Code Issues 44 Pull requests 11 Actions Projects 0 Wiki Security Insights

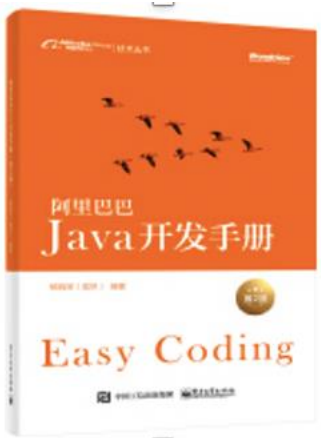
Alibaba Java Coding Guidelines pmd implements and IDE plugin https://github.com/alibaba/p3c/wiki

233 commits 3 branches 0 packages 21 releases 26 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

SeanCai Update issue templates Latest commit 20889ab on Oct 15

.github/ISSUE_TEMPLATE	Update issue templates	last month
eclipse-plugin	v2.0.0	5 months ago
idea-plugin	v2.0.0	5 months ago



8.5.3 模糊测试

- 模糊测试（Fuzz Testing）：基于错误注入的黑盒随机测试技术
  - 向目标程序输入随机或者半随机的畸形测试数据，以引发被测试目标产生异常；
  - 通过分析程序执行结果及检测程序状态，发现目标程序中隐藏的各种安全漏洞；
  - 不关心被测试目标的内部实现。

8.5.3 模糊测试

- 适用对象
  - 适用于测试处理结构化输入（如格式化文件，协议）的程序.
- 测试输入要求
  - 足够的正确性：以防被测试对象直接丢弃
  - 足够的畸形度：以引发被测程序的意外行为

8.5.3 模糊测试

- 几类模糊测试器
  - 文件格式Fuzzer

    AFL, WinAFL, FileFuzzer
  - 参数Fuzzer

    COMRaider，SyscallFuzz
  - 网络协议Fuzzer

    Spike，ProtoFuzz，Dhcpfuzz