1. 概述

安全开发周期，即 Security Development Lifecycle (SDL)，是微软提出的从安全角度指导软 件开发过程的管理模式。SDL 不是一个空想的理论模型。它是微软为了面对现实世界中安全挑战，在实践中的一步步发展起来的软件开发模式。

那么，SDL 到底是什么，它是如何和传统的软件开发模式结合？在回答这些问题之前，我们先来阐述一个基本问题，本文的读者，你，为什么要考虑 采用 SDL 来指导软件开发？

2. 为什么要从安全的角度来指导整个软件开发流程？

读者也许会说：对，软件安全是很重要。但是这些是微软或其它大型软件开发厂商进行软件开发时应该考虑的问题，和我好像没什么直接关系吧。有这 种想法并不奇怪。

回答：现在安全问题已经不局限于操作系统软件。所有的应用程序，都面临着安全挑战。

2.1 互联网环境下的安全挑战

安全领域是计算机技术中的发展最为活跃的一个分支。每一项新技术的出现，也带来了相应的风险。不同的时代，不同的技术，就 有不同的安全挑战。

最近几年来，计算机的攻击模式的变化，从没有特定攻击目的，向有特定的目的变化。不再是简单的要登上报纸的头版，或者是恶作剧，而是有特定的 目的，即窃取用户的机密信息，如银号账号，密码等，以获取经济上的利益。

目的的变化，导致了手段的变化。若干年前，传统的攻击手段主要是针对操作系统的安全漏洞，因为攻击操作系统可以导致散布病毒的最广泛的途径。 但是，“最广泛”，已经不是攻击的目的。于是，攻击的软件系统，已经从操作系统，扩展到图像处理，办公处理，备份软件，反病毒软件，web 应用等等各类应 用程序。举个例子，最流行的一个攻击方式，sql injection，针对的不是操作系统，而是典型的 web 应用程序。

特别的，如果开发的应用程序：

有面向网络的功能界面

有面向数据库的应用

有不同级别的权限控制

有存放重要/敏感信息

就更应该考虑在软件开发流程加入安全方面的考虑。

2.2 传统的软件开发流程的局限性

传统的软件开发流程中，如瀑布模型，中心围绕着产品功能，完全没有安全方面的考虑。因此，无法开发出安全的软件也就不足为奇了。微软自身的软 件开发流程，在 SDL 提出以前，就是一个很好的例子。它可以造就功能上相对完善的软件，如 Windows 的早期版本，但是无法满足在安全上的需要。

需求分析。传统的软件开发流程的需求分析，往往有这个一个倾向，用户一旦使用软件，可以直接使用的功能越多越好，所允许的用户工作环境越灵活 越好。但是，实际中，一定是可以直接使用的功能越多越好吗？往往需要在灵活方便，和安全性能上作一定的平衡。例如，对于不经常需要的功能，缺省配置下是不 是应该关闭？

软件设计。传统的软件设计过程，考虑中心是如何有效，正确的实现功能。往往约定来自相关模块的数据是值得信赖的，只在最外围的数据模块接口处 对用户数据加以校验。与之带来的后果就是，如果一个模块出现安全问题，整个软件系统就处于不设防状态。

软件编码和评估。同样，软件编码的实践主要集中如如何有效，正确的实现功能。没有针对安全问题的编码和复查指导，如特定 API 函数的副作用， 堆栈溢出错误等等。没有对应的编译工具和程序静态分析工具以检查通常的代码安全错误。

测试。按照传统的黑箱/白箱方式设计数据，是无法模拟一个恶意的攻击数据的。这是因为传统的测试数据是根据功能文档来设计的，数据范围限于正 常数据和边界情况，以模拟通常的用户使用环境。

2.3 传统软件知识结构的不足

以往对于软件开发的教育，如软件工程，数据结构，编译原理，系统结构，程序语言等等，并没有针对软件安全方面的知识。

如果开发人员对安全问题没有的足够理解，不了解安全设计的基本原理，安全漏洞的常见类型，如何设计针对安全的测试数据，所开发的代码就自然就 更有可能出现安全漏洞。

例如，以下这段代码[2，p147]就展示了 Windows 系统 RPC 调用中的安全漏洞。它就是导致冲击波病毒（Blaster）爆发的根源。是不是没想到这么简单？那么，这段 代码的问题是什么？

```
HRESULT GetMachineName(WCHAR *pwszPath)
{
 WCHAR wszMachineName[ N + 1 ];
 ...
```

```
LPWSTR pwszServeName = wszMachineName;

while (*pwszPath != L'\\')

*pwszServerName++ = *pwszPath++;

...

}
```

如果你已经看出问题，恭喜你！大多数开发人员在没有经过安全培训前，是不知道正确答案的。代码的问题在于对 pwszPath 的字符串长度没有 验证，从而导致 wszMachineName 堆栈变量的溢出。

这里要特别强调的，这里指的软件安全的知识，不是指对于安全功能的理解和使用。例如，加密解密的原理，SSL 的使用等等。了解这些知识是有用 的，但是仅仅掌握 SSL 的使用，并不能保证设计出的网络软件是安全的。

2.4 实例：来自微软自身的数据

Windows Server 2003 是微软第一次大规模实施 SDL 在操作系统的开发上。虽然 Windows Server 2003 还不是从头至尾都严格按照 SDL 的规范开发，但我们已经可以从图 1 展示的数据中看出，SDL 在很大程度上减少了操作系统软件的安全漏洞。数据来源 于[1]。

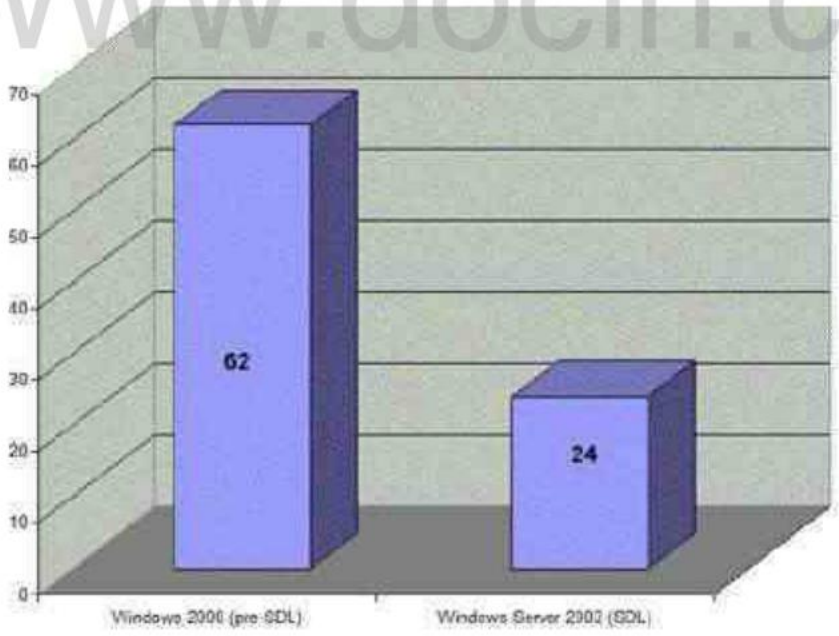

图 1：Windows 2000和Windows Server 2003 安全漏洞数目对比

3. 如何将 SDL 应用于传统软件开发模式

3.1 SDL 综述

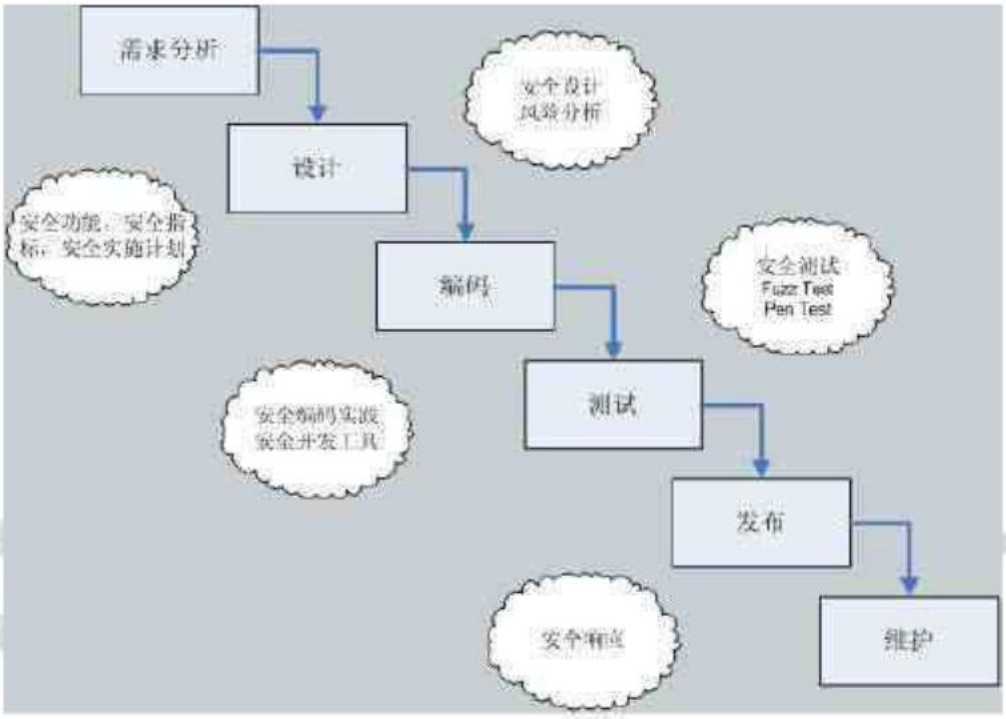

图2：SDL开发模式

图 2 是一个简化的 SDL 开发模型。SDL 的核心理念，就是将软件安全的考虑，集成在软件开发的每一个阶段：需求分析，设计，编码，测试，和维 护。

为什么说这是一个简化的 SDL?

传统的软件开发模型，如瀑布模型，是一个简化的模型。在实际开发中，需求分析，设计，开发，测试，是一个反复循环的螺旋开发模式 （spiral model）。需求分析和设计文档会经历若干次的修改。在每一次修改过程中，都应考虑对软件安全方面的影响。

完整的 SDL 模型，还包括若干更为细化的阶段。如在产品正式发布前的 Final Security Review （FSR）阶段等等。有兴趣的读者可以参考[2]获取进一步信息。

这个模型中没有包括工程人员的培训教育。一个再好的流程，如果没有相应的工程人员来实施，也会是纸上谈兵。对工程人员的教育，一是观念上的改 变，

加强安全意识。二是基本安全知识的掌握。

3.1.1 SD3+C 原则

SD3+C 是微软归纳的实施 SDL 中的基本原则。

安全设计（Secure by Design）。软件的设计和实现需考虑如何保护其本身（和存 储的信息）抵御外部攻击。

安全配置（Secure by Default）。软件的缺省配置运行环境应考虑如何降低安全风险。重要的一个假设是软件自身代码总存在安全漏洞。那么，如何减少这些安全漏洞的危害？例 如，是否可以运行于普通用户权限，而不需要管理员权限？是否可以缺省关闭某些高风险的代码模块？

安全部署（Secure in Deployment）。 软件需提供相应的文档和工具，指导用户如何安全的使用。

交流（Communications） 。开发人员需要对发布产品中的安全漏洞准备响应方案。

3.2 需求分析：设定安全目标

在需求分析阶段，加入以下的安全考虑：

产品提供的安全功能

产品如何安全的与用户（或其它软件模块）交互

特别的，安全方面的考虑对产品开发计划的影响。

产品的风险评估和威胁模型（threat modeling）

产品的缺省功能配置

3.3 安全设计

在安全设计阶段，特别加入以下两方面的考虑。

减少攻击界面。例如，对一个网络软件的设计，它需要监听那些网络端口，是否可以减少监听端口的数目？那些 用户可以与这些端口建立连接，是否要加强身份验证？

深层防御。底层模块的设计中，假设上层模块有可能出现安全漏洞。对传递的数据考虑进一步校验。

3.4 如何避免代码中的安全问题

每个开发人员都需要遵循安全编码规范。

使用最新的编译器和编译选项。对于微软的最新 C/C++编译器，使用以下两个编译选项：

/GS 选项。加入检测函数堆栈缓存溢出错误额外代码。

/SAFESEH 选项。加入额外的异常处理信息，以确保代码所调用的是合法的，而不是被非法篡改过的异常处理程序。

禁止使用特定的危险 API。许多安全漏洞都是由于不当使用危险的 API 函数导致的。常见的危险函数如 strcpy, strcat, sprintf, strlen 等等。这些危险函数在最新的 C 语言运行库中已经被标为“deprecated”。读者可以参考[3]获取进一步的信息。替换这些函数可以考虑 使用 StrSafe 定义的函数。限于篇幅，这里就不给出具体离子了。[4]包括使用 StrSafe 的详细信息。

使用静态语言分析工具以扫描安全漏洞

定期进行安全代码复查

3.5 安全测试：模拟恶意输入

安全测试引入了 Fuzz 测试这个概念。它的主要目的是创建恶意的输入数据，以模拟软件被恶意攻击时的行为。Fuzz 测试可包括的对象可以是文 件测试，网络数据测试，用户界面输入数据测试，等等。

我们用 AVI 文件来举例一个典型的 Fuzz 测试。下面是 AVI HEADER 的定义：

```
typedef struct _avimainheader {
...
 DWORD dwStreams;
 DWORD dwSuggestedBufferSize;
 DWORD dwWidth;
 DWORD dwHeight;
...
} AVIMAINHEADER;
```

假定 dwSuggestedBufferSize 的通常范围为 1k – 4k，那么，在正常情况下，以下代码不会出现问题。

dwSuggestedBufferSize = pBuffer->GetDW();

m_pSuggestedBuffer = new char [dwSuggestedBufferSize];

但是如果我们在 fuzz 测试中恶意设定这个输入变量值为 0xFFFF，那么，就可能发现它导致了系统的内存分配错误，从而可以避免一个客户端 /服务器端的 DOS（deny of service）安全漏洞。

同样基于 fuzz 原理，我们根据 AVI HEADER 的结构信息，设计其它的测试数据，如：

恶意的 dwStreams

恶意的图像尺寸：dwWidth，dwHeight

无或多个 AVI HEADER，等等

3.6 安全响应和维护：紧急反应

当前任何一个软件开发模式（包括 SDL 在内），都无法确保发布的软件没有安全漏洞。因此，需要事先制订对应的相应模式，包 括：

（内部或外部发现的）安全漏洞以何种途径汇报

如何评估安全漏洞的严重级别

开发安全补丁的流程

测试安全补丁的流程

发布安全补丁的流程

如何在以后开发中避免类似的安全漏洞，等等

4. 总结

实践证明，SDL 可以有效的减少软件的安全漏洞，提高软件的安全。但是它不是万能的。实施 SDL 开发的软件也不能完全避免安全漏洞。例 如，Windows Vista 就是严格按照 SDL 开发的，但是它依然存在安全漏洞。只不过我们期望，它的安全漏洞的数目和严重程度，要比以往的操作系统减少许多。

安全领域中，没有免费的午餐。SDL 的实施，需要时间，精力，教育，管理等多方面的支持。但是，如果不购买这顿午餐的话，晚餐账单可能会比想 象的要高的多。

限于篇幅，这里只能对 SDL 作一个泛泛介绍。如果需要进一步了解，可以参阅附录中的

参考文献。