**应用开发安全指南**

本文为收大量资料集整理并修改，有些涉及技术方面的已经比较旧，欢迎大家一起完善和提建议。

# **概述**

本指南是IT安全保障体系建设规范的一个组成部分，全面阐述了IT系统应用开发整个软件生命周期所必须遵照的设计、编码、测试方面的安全要求，阐述了不同开发环境和编码语言条件下安全开发的相关规范要求。

## 目的

本指南针对应用系统应当遵循的应用开发安全标准进行了规范性说明，旨在指导应用系统设计人员、代码开发人员和安全检查管理人员进行应用安全开发的安全配置，以提高应用系统的安全防护能力。

## 适用范围

本指南适用于代码开发项目，作为在IT系统开发、设计环节中所遵照执行的依据。

## 适用对象

本配置指南的适用人员包括：系统应用开发人员及安全检查管理人员。

# **应用系统设计安全**

## 应用系统架构安全设计要求

在应用系统设计阶段，应充分考虑该架构的安全性,包括B/S、C/S等形式的安全，主要体现在应用数据和用户会话的安全，还应当考虑应用系统自身体系架构内部的安全，以及与外系统接口的安全。针对某些特殊应用，还需考虑复、抗攻击等安全机制。所有的安全设计都是为了保证系统的稳定性和连续性，有针对性的解决自身安全问题。

### 应用系统自身架构安全

1. 自身结构中各组件之间通讯过程的安全机制

组件之间的通讯包括命令级的和数据级的，应充分考虑：

* + - 传输命令和数据所采用的协议的安全性。应根据组件之间通讯内容安全性要求程度的不同选择不同安全性要求的协议；
    - 考虑程序的模块之间的安全通讯机制；
    - 不应使用标准的服务端口或者常见病毒、蠕虫等使用的服务端口。

1. 认证与访问控制机制，应考虑：
   * + 组件之间的信任机制；
     + 用户的身份认证机制；
     + 对于组件资源的访问控制机制；
     + 不通用户对资源的权限控制机制。
2. 组件内重要文件和数据的安全防护机制：

存在于组件内部的重要数据资源应当考虑其相应的安全防护机制，这些重要的数据资源包括：

* 配置文件；
* 用户数据，包括文件数据及数据库中的数据；
* 临时文件和数据；
* 与外系统或者系统内部其他组件接口用的数据文件。

对这些重要数据的存取安全性设计，包括:

* 文件和数据存放是否加密及采用的加密方式。

### 应用系统与外系统接口的安全

应用系统与外系统的接口安全设计，主要应考虑以下几个要素：

1. 与外系统的之间通讯中的安全机制。应充分考虑：

* 传输命令和数据所采用的协议的安全性。应根据系统之间通讯内容安全性要求程度的不同选择不同安全性要求的协议；
* 建议不使用默认的服务端口或者常见病毒、蠕虫等使用的服务端口，传输过程使用加密传输。

1. 与外系统的认证与访问控制机制，应考虑：

* 系统之间的信任机制；
* 会话凭据的有效时间；
* 可以访问资源的权限控制；
* 对于系统之间资源的访问控制机制。

1. 对外系统安全机制的符合性，应考虑：

* 如果外系统采用的接口方式经评估认为是安全的，本系统应当沿用其接口规范进行设计开发；
* 如果外系统采用的接口方式经评估认为存在安全缺陷，应商定采用更加安全的接口方式；
* 在考虑接口安全性的同时，也应当注意接口方式对双方系统性能、磁盘、连接数等各种性能指标和资源的影响。

### 应用系统其他的安全机制

除了上述基本的安全架构设计内容外，针对不同的应用，以及应用系统的重要程度，可以补充考虑以下几种安全机制：

* + 1. 针对Web应用的页面保护与恢复机制。

利用专用的安全产品，或者系统自身设计时就考虑到了对于Web页面进行静态保护和监控问题，当监控到网页被篡改时能够实时恢复页面。

* + 1. 针对特殊数据的完整性检查和监控机制。

应用系统自身的审计机制。这一点也可算作是应用系统的安全功能设计的一部分，参见相关章节的要求。

* + 1. 针对重要系统的DDOS攻击的解决方案。
    2. 应用系统安全性分析。
    3. 除了使用技术手段保证接口安全的同时，也需要保证接口相关代码和文件不被上传到互联网或泄漏给非必要人员。

任何系统都会存在一定的安全缺陷，关键在于风险和缺陷是否可以被容忍，因此，在应用系统设计完成后，应当就其安全性问题进行自我分析和评价。

## 应用系统软件功能安全设计要求

除了在架构上考虑的安全机制外，这些安全机制及相关的安全功能也应当分配在应用系统软件的各部件中。应用系统在开发中应该考虑如下几个方面的安全功能：

* 安全审计；
* 通讯安全（此部分内容在架构中进行了设计）；
* 数据保护；
* 认证与授权；
* 资源保障；
* 最小权限。

### 认证与授权功能的设计

1. 应用软件应包含用户身份认证体系的强度设计,重要系统应使用双因素或多因素认证措施，加强系统安全性：

* 用户名、口令认证；
* 一次性口令、动态口令认证；
* 证书认证；（可选）
* 生物特征的认证（签名、声音、指纹、虹膜、视网膜等）。（可选）

1. 应用软件应包含认证失败后的处理方式设计
   * + 连续失败3次，将锁定登录账号一个小时。账号锁定后可以由系统管理员解锁，也可以在一段时间后自动解锁；
     + 通知用户认证失败，防止黑客暴力猜测；
     + 验证码的功能，需要一定复杂度，防止被机器识别；
     + 账号复杂度提醒功能。
2. 应用软件应包含用户权限分配和管理功能设计。
   * + 系统编码中要实现读、写、执行三个权限的分离设计；
     + 系统查看、配置、修改、删除、登录、运行等权限设计；
     + 数据访问范围的权限设计；
     + 针对不同用户的数据库表权限隔离；
     + 应用功能模块使用权限的设计。
3. 应用软件应包含接口设计，应明确系统的内部结构和外部接口，对于每一个对外接口应详细说明：
   * + 需要通信的对方系统的安全状况和可信程度；
     + 需传送的数据的保密性和完整性要求；
     + 对传送数据的合法性检验规则；
     + 对通信可靠性的要求；
     + 与外部系统的互相认证方面的需求；
     + 信息传输过程的加密需求。

### 数据安全功能

1. 应用系统的数据安全功能，应当根据安全需求进行功能设计，内容涉及：数据库的安全、数据采集、数据传输、数据处理、数据存储、数据备份和恢复的安全。对重要的敏感数据应进行加密和完整性保护。
2. 应用软件应包含输入的安全性设计，主要指对错误输入、恶意输入进行处理。
3. 应用软件应包含输出的安全性设计。

### 安全审计功能

1. 应用系统具备如下安全审计功能：

* 审计功能的启动和关闭；
* 变更审计功能的配置信息；
* 至少应进行审计的事件：进入和退出的时间(登录、退出系统)、异常的系统使用行为(失败登录)、系统维护行为、敏感行为和其它安全功能要求的审计内容；
* 每个审计记录中至少记录如下信息：事件的日期和时间、事件的类型、主题标识、事件的结果(成功、失败)和事件相关信息。

1. 应用系统应支持数据查阅审计功能：按照主题、事件查阅；应用系统应明确用户能够查阅审计数据用户。
2. 在意外情况出现时，应有措施保证审计数据的可用性，当审计记录溢出时采取保护行动。

### 容错功能设计

1. 应用软件应包含各模块的出错处理设计。
2. 应用软件应包含可能出现的各种异常情况的安全处理设计，如：错误信息不回显给用户。
3. 应用软件应包含抗网络攻击的能力的设计及系统脆弱性分析。
4. 对于应用软件本身的资源及服务的优先保障设计。

## 应用系统存储安全设计要求

在应用系统存储安全设计时，应对系统的存储容量、存储介质、存储备份内容、存储备份方式、存储设备功能要求及相关的存储技术统筹进行考虑。

### 应用系统的存储容量设计

应依据对于应用数据的测算，估算应用系统的存储容量，建议在存储容量估算时应考虑以下要求：

* 在实际估算值上预留30%的存储余量，并考虑未来的应用存储量的增长需求。
* 考虑到应用系统自身的审计数据的容量、保存期限以配置相应的存储设备。
* 对于应用系统中的临时数据和过渡数据，应当设计其保存的时间，并以此考虑这部分的存储容量要求。

### 应用系统的存储介质选择

应用系统的存储介质主要包括但不限于：磁带、光盘、磁盘、磁盘阵列和云存储。具体存储介质的选择应依据应用系统的业务种类及存储周期的要求，采用不同的介质。

1. 对于应用系统的交易数据，应采用高性能、高可靠的存储介质，如磁盘、磁盘阵列、云存储等进行存储；
2. 对于应用系统的历史数据，应采用可靠、稳定的存储介质，如磁带、光盘等进行存储。

### 应用系统存储备份对象

应用系统对于其储存备份的对象设计，应包括如下内容：

1. 系统数据的备份：应包括Web服务器的网站内容、Mail服务器的邮件实时备份、数据库、文件服务器中的文件以及其他数据；
2. 系统的完全备份：应包括关键的、需要快速恢复的设备，通过磁带机的完全备份，应实现快速的灾难恢复；
3. 系统的冗余主机备份：对于关键并且不能停止的服务设备（如计费服务、Web、Mail服务器），应考虑使用多台主机进行冗余备份，以保证当任何一台主机发生故障时，服务器仍可提供服务；
4. 系统配置的备份：应包括关键路由器的配置、防火墙的配置、各类服务器操作系统的安全配置以及各类服务器（如Web、Mail、文件服务器等）中间件和容器（如Apache、tomcat、nginx、weblogic等）的配置。

### 应用系统存储备份方式

应用系统应当根据不同的阶段，系统数据不同的重要程度，对数据采取不同的备份方式：

1. 完全备份

使用备份介质对整个系统进行完全备份，包括系统和数据。这种备份方

式的优点是直观，容易被人理解，而且当数据丢失时，可以快速恢复丢失的数据。它也有不足之处：

* 定期对系统进行完全备份，因此在备份数据中有大量的重复信息，占用了大量的存贮空间，增加了备份成本；
* 需要备份的数据量大，因此备份所需要的时间较长。
* 建议在关键性应用系统的实施前、实施后、变更以及升级等重要操作时，对操作系统进行完全备份。针对信息较小的不断变化的，且变化的内容大于50％的，定期进行完全备份。

1. 增量备份

每次备份的数据只是相当于上一次备份后增加和修改过的数据。没有重复的备份数据，节省备份介质的空间，缩短了备份时间。这种备份的优点很明显，同时也存在某些不足之处，即当发生灾难时，恢复数据比较麻烦。

建议在关键性应用系统正常运行维护阶段，针对变化的、不断增加的信息，定期进行增量备份。

1. 差异备份

每次备份的数据只是相当于上一次完全备份后新增加和修改过的数据，即采用完全备份和差异备份相结合备份策略。如：每周日进行一次完全备份，而周一至周六进行差异备份。其优点为：没有重复的备份数据，即节省备份介质的空间，缩短了备份时间；缺点为：当发生灾难时，恢复数据比较麻烦。

建议应用系统的正常运行维护阶段，针对不断变化的（变化的内容小于

50％）系统，定期进行差异备份。

1. 按需备份

按需备份是指在正常的备份安排之外，额外进行的备份操作，这种备份方式可以弥补冗余管理以及长期转存的日常备份的不足。因此它是一种非常灵活、重要的备份方式，在应用系统的各个阶段，如果备份的内容较少，可以采用按需备份。

建议应用系统在下列情况下采取按需备份：

* 只需要备份很少的几个文件、目录、数据库或数据库中的表；
* 备份服务器上必要的配置文件。

1. 排除备份

排除备份是指排除不需要的文件后再进行备份。从本质上讲，排除备份不是一种备份方法，只是减少备份冗余的一种方法。

建议应用系统在下列情况下考虑排除备份：

* 有些文件非常大，但并不重要；
* 某些文件总是导致备份异常或出错。

1. 备份恢复测试

不管是全备、增备、差异、按需还是排除备份，都应该定期对已被分的数据进行恢复测试，保证备份数据的可用性，在关键时刻可以及时并快速的恢复系统。

### 应用系统的存储设备功能要求

应用系统存储设备的功能要求应包括如下内容：

1. 存储设备应保证数据的高可用性和完整性要求；
2. 存储设备应具有在多主机环境下工作的能力；
3. 存储设备应能方便地做到快速备份和恢复，重要系统应做到双机备份、支持热插拔；
4. 存储设备应有简便的、功能强大的管理工具，做到对整个存储系统的监视与控制。

## 应用系统通讯安全设计要求

1. 应采用安全通信协议对重要数据进行安全传输（尤其是账号、口令信息），如使用SSL/TLS、HTTPS、SFTP和IPSec、SCP等安全协议进行通信：终端与服务器端之间的WWW服务，建议使用HTTPS安全通信协议；终端与服务器端之间的FTP服务，建议使SFTP安全通信协议；终端与服务器端之间的Telnet服务，建议使SSH安全通信协议。
2. 终端应用程序采用加密传输机制对重要信息进行传输。
3. 终端应用程序采用完整性检查对业务的重要数据或敏感数据进行检查。
4. 终端应用程序应采用抗抵赖攻击技术对重要的交互信息进行保护。
5. 终端应用程序使用固定的通信端口。
6. 对于需要映射到公网的端口，不要使用原端口，在条件允许的情况下使用白名单访问。

## 应用系统数据库安全设计要求

1. 应从以下方面进行数据库的选型：

* 数据库、应用系统的运行环境；
* 数据库的稳定性、安全性（多级安全）；
* 数据库的容量（最多支持的库的数目、表的数目、记录数目）；
* 数据库的存取速度；
* 是否支持多种备份方式；
* 是否支持数据库的导入和导出。

1. 应明确数据库相关的用户管理、资源管理、特权管理和角色管理，明确各种用户的资源权限，并建立规范的权限文档。
2. 数据库原则上应及时更新重要补丁。在安装补丁前应先在测试环境进行，提前进行数据备份，充分准备回退方案和应急预案。
3. 数据库的配置应符合相应的基线配置要求。
4. 应及时修改数据库的默认密码或将默认账号锁定、删除。
5. 数据库的账号应根据业务和维护需要进行合理分配，避免账号共用。
6. 数据库每个用户之间的权限需要隔离。
7. 需要对数据库操作进行审计，包含：账号、操作、时间等指标。

## 应用系统数据安全设计要求

### 数据采集安全

应根据数据采集的内容、采集的频率、数据精确度要求、时间特性等来进行数据采集的安全要求设计，数据采集服务器和采集主机应考虑30%的系统开销及冗余。

### 数据传输安全

1. 应按照数据的类型、数据的重要程度、网络的安全状况等综合因素，对

数据的传输采取不同的安全保护，包括但不限于防火墙、IDS、IPSEC-VPN、病毒防护等安全措施。

1. 应了解数据传输存在安全隐患的网络或设备，对存在安全隐患的网络采取必要的安全技术，包括但不限于安全通信协议、加密算法、完整性检查算法以及抗抵赖攻击方法等。
2. 应制定数据传输安全的检查方式，包括但不限于数据传输安全抗主动攻击能力检查、被动抗攻击的能力检查。
3. 应保障“数据传输安全”有关的重要配置参数安全，包括但不限于口令、加/解密算法、加/解密密钥等。
4. 应采用安全通信协议对数据进行安全传输，如使用SSL/TLS、HTTPS、SFTP和IPSec等安全协议进行通信。
5. 对传输的信息进行不同等级的加密保护，即根据网络或设备的风险、传输内容安全要求的不同，选择不同安全强度的加密算法对信息进行加密传输。建议使用RSA等高强度的密码算法对非常重要的信息（如口令、加密密钥）进行加密传输；对于普通数据的传输，可以采用RSA、3DES等相对安全的加密算法进行加密传输。
6. 应防止对所传输数据进行未经授权的任何形式的修改，即对业务的重要数据或敏感数据，建议使用MD5、SHA等算法对数据完整性进行保护。
7. 对重要的交互信息，建议采取抗抵赖技术，包括但不限于数字签名技术。
8. 为了配合网络其它安全设备，建议采用基于用户名/口令的认证技术、VLAN技术、MPLS技术等安全技术手段。

### 数据处理安全

1. 应根据数据的类型、数据的处理方式、数据的安全性要求、与其它接口有关的敏感等级、数据相关业务应用的重要性程度来进行数据处理过程的安全性设计。
2. 应对原始数据进行检错和校验操作，保证原始数据的正确性和完整性。
3. 数据在转换过程中，应采用通用的标准格式，应考虑相关的不同系统和不同应用的格式需求。
4. 数据处理过程应提供处理数据的状态信息和数据处理过程的动态信息。
5. 数据处理过程应具备异常处理功能，在任一环节发现问题，均应能及时回退，必要时可以人工处理。
6. 数据处理的中间过程和中间结果不能暴露给第三方。

# **应用系统开发安全**

## 安全原则

1. 保护最薄弱的环节原则：保护最易受攻击影响的部分；
2. 纵深防御原则：不同层面、不同角度之间需要相互配合；
3. 最小权限原则：只授予执行操作所需的最小权限；
4. 最小共享原则：使共享文件资源尽可能少；
5. 权限分离原则：授予不同用户所需的最小权限，并在它们之间形成相互制约的关系；

## 需求管理阶段

1. 根据业务目标分析并制定系统安全目标；
2. 确认安全需求规格说明。

## 系统设计阶段

1. 根据安全目标执行威胁建模，识别威胁和风险；
2. 根据威胁建模制定对应开发安全任务：

* 确定安全体系架构，设计安全协议和安全接口；
* 确定访问控制与身份鉴别机制，定义主体角色和权限；
* 信息输入的安全过滤，信息输出的校验和控制；
* 数据结构安全设计，选择加密方法和算法；
* 确定敏感数据保护方法；
* 内部处理逻辑安全设计；
* 评估内部通信机制，确定完整性机制。

## 系统实现阶段

1. 开发环境安全管理要求：

* 软件系统开发、测试禁止在生产环境中进行；
* 开发环境中的开发用机应进行统一安全配置及时进行系统补丁升级和漏洞修复；
* 软件程序不得篡改应用软件所运行的环境或平台中任何安全配置、安全文件和安全程序。

1. 编码安全要求（后面会有详细讲解）：

* 遵循代码编写安全规范，根据代码编写安全规范以及安全设计方案进行系统开发；
* 遵循通用安全编程准则，包括输入验证、缓存溢出、安全调用组件和程序编译等；
* 遵循机密性要求，保护用户访问信息的机密性，严禁在客户端存放敏感数据
* 避免内存溢出，严格检查和验证输入输出信息等；
* 遵循结构化异常处理机制，捕捉并处理程序异常，防止系统信息泄露
* 遵循代码脆弱性防范要求，包括缓冲区溢出、SQL注入、跨站脚本攻击、XML注入攻击、HTTP HEAD注入等。

1. 开发流程安全要求：

* 开发过程中应对阶段性开发成果进行有效管理；
* 开发过程中应定期进行代码静态分析，使用代码审核工具对源代码进行检测，并报告源代码中存在的安全弱点；
* 开发人员不得超越其规定权限进行开发，不得在程序中设置后门或恶意代码程序。

## 系统测试阶段

1. 测试内容应包括代码的安全测试和安全功能测试；
2. 代码的安全测试是指使用代码测试工具或渗透测试来识别代码的安全脆弱性，并应按照其提供的修复建议进行修复；
3. 安全功能测试主要包括身份认证和访问控制的功能测试；
4. 测试系统环境应尽可能模拟生产环境并与生产环境进行安全隔离；
5. 真实数据不得直接在测试环境中使用，须进行适当修改或屏蔽，在测试完成之后须立即从测试应用系统清除运行信息；
6. 测试人员编制安全测试方案，构造安全测试用例；
7. 验收测试不得由开发人员兼岗。

## 系统上线阶段

1. 系统上线须在内部验收通过后进行；
2. 需进行上线前试运行，确认应用系统是否符合上线要求；
3. 上线成功后，记录上线的日期和内容。

## 文档管理

1. 源代码的变更和版本发布进行统一控制，对程序资源库的任何修改、更新和发布都需经部门主管领导授权和批准；
2. 应指定专人妥善保管程序源代码及相关技术文档。

## 外包管理

1. 应与外包开发单位签署相关知识产权保护协议和保密协议；
2. 外包开发单位进行系统开发过程中须严格遵循本制度的相关安全要求；
3. 在系统开发过程中须指派专人监督审核外包开发单位在各个阶段安全要求的执行情况；
4. 外包开发单位在系统开发完成后提供程序源代码和相关技术文档，不得将计算机系统采用的关键安全技术措施和核心安全功能设计对外公开；
5. 应对开发完成后的应用软件进行审查或检测。

# **应用系统编码安全**

## 基本代码安全要求

### 输入验证

对函数入口参数的合法性和准确性进行检查，具体如下：

* 在B/S环境下，应进行服务端的验证而不仅仅是客户端的验证（例如基于Javascript的验证）。通过在客户端和服务器之间放置一个代理服务器，可以很容易绕过客户端验证。有了代理服务器，攻击者可以在数据被客户端“验证”后修改数据（与“中间人”攻击类似）。
* 在实际的校验中，输入校验首先定义一个有效（可接受）的字符集，然后检查每个数据的字符是否在有效范围内。如果输入中包含无效的字符，应用程序应该返回错误页面并说明输入中包含无效字符。这样进行验证的原因是定义无效的字符集比较困难，并且一些不应该有效的字符通常不会被指出。
* 另外，边界检查（例如字符串的最大长度）应该在字符有效性检查以前进行，边界分析可以防止大多数缓冲区溢出漏洞。
* 从环境变量获得的数据也需要进行验证，同时避免在环境变量中存放敏感数据（例如密码）。

### SQL语句

如果应用程序需要连接后端数据库，使用存储过程而不能在代码中使用SQL

语句，使用程序以外的嵌入在代码中的SQL语句调用特别危险，难以防止攻击

者使用输入域或者配置文件（由应用程序载入）来执行嵌入式的SQL攻击。当然，输入验证有助于缓解这种风险。

### 注释代码

当应用程序在实际环境中开始应用时，应该删除所有的注释代码。注释代码

是用来调试或者测试的，它们不是最终应用程序的一部分。无论如何应该在实际的环境中删除它们以避免意外的执行（一般注释标识被删除后就无法激活休眠的代码，但还是存在可能性的，所以强烈建议执行这项工作）。

### 错误消息

所有为用户显示的错误信息都不应该暴露任何关于系统、网络或应用程序的

敏感信息。如果可能，应使用包含编号的一般的错误信息，这种信息不返回给访问用户，只返回404错误，如“发生了错误（代码1234），请您与系统维护部门联系。

### URL内容

对于Web应用，不能在URL上暴露任何重要信息，例如密码、服务器名称、IP地址或者文件系统路径（暴露了Web服务器的目录结构），这些信息可以在攻击时被使用。

### 设置PATH变量

设置PATH为一个已知的值，而不是仅仅使用启动时的缺省值。攻击者可

以在攻击应用程序时使用PATH变量，例如试图执行一个任意的程序，这些可

以应用于大多数其他的语言。

### 其他要求

1. 禁止使用未经授权和验证的代码。
2. 使用第三方代码，应对代码安全性进行评估和测试。
3. 测试用的“后门”，应在发布版中去除。
4. 规范代码的格式。
5. 规范变量、函数的命名；
6. 规范程序的书写格式，确保程序的易读性。
7. 对代码进行版本控制，确保代码的可用性。
8. 防止程序员非授权修改代码
9. 对代码的访问权限进行严格的权限控制；
10. 禁止在程序中添加隐藏“恶意”的代码，防止与应用系统相关的程序员对系统的非授权修改。
11. 应用系统不应在程序或进程中固化账号和口令。
12. 系统应具备对口令猜测的防范机制和监控手段。
13. 避免应用程序以错误的顺序运行，或者防止出现故障时，后续程序以不正常的流程运行。
14. 采用正确的故障恢复程序，确保正确处理数据。
15. 采取会话控制或批次控制，确保更新前后数据文件的一致性，例如：检查操作前后文件打开和关闭的数目是否一致。
16. 检查执行操作前后对象的差额是否正常，如：句柄处理，堆栈等系统资源的占用与释放等。
17. 严格验证系统生成的数据。
18. 在网络传输过程中检查下载/上传的数据或软件的完整性。
19. 检查文件与记录是否被篡改。例如通过计算哈希值（HASH）进行对比。
20. 禁止私自讲代码上传到互联网。

## Web编程安全基本要求

### 输入检查安全

1. 限制用户输入HTML和Script(JavaScript、VBScript)代码。输入恶意HTML或Script（JavaScript、VBScript）代码可能会对其他浏览者造成混淆、欺骗或恶意破坏的结果。
2. 检查用户输入数据的长度。输入超出限定长度的数据，可能造成服务器端程序溢出。
3. 防止用户输入特殊字符改变SQL语义。输入含特殊字符的字串，篡改SQL语句的语义，可能造成SQL查询执行不该执行的操作，以此绕
4. 过身份认证获取非法权限、甚至对数据进行破坏。
5. 限制用户能够访问的最顶层目录。编写对服务器端文件、目录操作的程序时应该注意限定此类程序能够访问的最顶层目录，防止用户构造输入字串借助程序功能访问服务器关键文件导致泄漏服务器敏感信息。
6. 对所有类型的用户输入都要做检查，并严格限定什么是合法的用户输入，限定一个合法输入的范围，同时过滤有可能造成危险的特殊字符。
7. 对不可信任域发送到可信任域的数据一定要进行检查。
8. 尽可能在服务器端完成用户输入检查，不能轻易相信客户端脚本的检查结果。虽然客户端的Script脚本能完成一部分的用户输入检查功能，但这种检查的结果是不可信任的，攻击者可以自己制作表单程序绕过客户端脚本验证，将非法数据提交到服务器。
9. 在输入变为输出时，也要对特殊字符做检查和转换。

### 敏感数据的存放和传递安全

1. 敏感数据不能存放在Web页中。
2. 不能把敏感的数据存储在cookie、隐藏字段或者潜在地可能会被用户修改的地方。
3. 客户端向服务器端提交敏感数据应该经过加密（例如使用SSL），尽量不能明文传输。
4. 密码等敏感信息存放在数据库中应该加密，并采用健壮的加密算法。
5. 防止数据库被攻破后泄漏用户密码。
6. 敏感数据需要脱敏显示。

### 缓冲区溢出安全

1. 所有的输入都必须进行正确的有效性检测。
2. 必须保证数组没有越界，增加数组操作函数的边界检查。
3. 安全地使用字符串处理函数，慎用有安全隐患的字符串处理函数
4. 使用Format字符串的时候特别注意Unicode和ANSI的大小不一致的情

况。

1. 注意字符串结束符的保护。
2. 仔细研究库函数内部的缓冲区分配，明确其限制。不能使用realpath()

等函数，如果功能需要必须使用时，一定要检查试图规范化的路径的长度，确保其不长于MAXPATHLEN。

1. 时刻进行边界检查。建议使用一些检查工具：Purify、Stackguard等检查代码，保证没有缓冲区溢出的问题。

### 格式化字符串安全

1. 使用固定的格式化字符串，或者来自可信源的格式化字符串。
2. 要检查并限定locale的请求为合法值。
3. 不能将用户输入直接作为格式化字符传给格式化函数。

### 整数溢出安全

1. 对于涉及到内存分配大小的计算，要进行仔细检查，确保计算不会产生溢出。
2. 对于涉及到数组索引的计算，要进行仔细检查，确保计算不会产生溢出。
3. 要使用无符号整数表示数组偏移和内存分配大小。

### SQL注入代码安全

1. 要检查输入的有效性和可信度。
2. 要使用参数化的查询、占位符、或者参数绑定来构造SQL语句。
3. 要在程序之外存储数据库的连接信息，比如经过保护的配置文件或者Windows注册表。
4. 即使使用的是存储过程，也不能使用字符串连接来构造SQL语句。
5. 不能在存储过程内部使用字符串连接来构造SQL语句。
6. 不能在存储过程内部执行不可信的参数。
7. 不能简单地双写单引号或者双引号。
8. 不能使用高权限账号连接数据库，比如sa或者root。
9. 不能在程序或者连接字符串中存储登录口令。
10. 不能在Web根目录下存储数据库配置信息。
11. 应从数据库中删除对所有用户自定义表的访问权限，同时只对存储过程授权，然后使用存储过程以及参数化的查询来构造查询字符串。

### 命令注入代码安全

1. 在输入命令传递给命令处理程序之前要进行验证。
2. 如果输入验证失败，要安全地处理失败信息。
3. 不能向任何命令解释器传递未验证的输入信息，即使这些输入仅仅是数据信息。
4. 避免使用正则表达式来进行输入验证，应手工去写一些简单而又清晰的验证代码。

### 异常处理代码安全

1. 要检测每个安全相关函数的返回值。
2. 对于每一个更改用户设定或者及其设定的函数，都要检查其返回值。
3. 要有从错误条件中进行恢复的考虑，避免拒绝服务攻击。
4. 不能一次性处理所有的异常，要将异常情况进行分类处理，避免在异常处理代码中的漏洞发生。

### 跨站脚本代码安全

1. 要对所有基于Web的输入进行输入验证和可信度验证。
2. 在没有验证合法性之前，不能对基于Web的输入进行回显。
3. 不能在cookie中存储敏感数据。

### 保护网络流量的代码安全

1. 要使用强大的初始认证机制。
2. 对应用程序所产生的所有网络流量都要执行过程中消息认证。
3. 尽可能使用SSL/TLS进行网络加密传输。

### .应用中的弱口令代码安全

1. 确保口令在网络上认证时不被窃听。
2. 要在登录失败时给出错误提示，并记录失败口令尝试。
3. 尽可能使用基于hash强壮的单向加密函数进行口令存储。
4. 为用户更改口令提供安全的机制。
5. 不得使用默认账号和默认口令，若使用，必须在首次登录后进行修改。
6. 不得在程序、后台存储明文的口令。
7. 口令要有一定的强度，应当满足系统的账号口令策略要求。

### SOCKET网络编程安全基本要求

1. 在socket函数调用时，明确参数中绑定的端口、IP地址和网卡接。Windows环境下，在遇到多个网卡的情况时，需要通过注册表来获得网卡接口和IP地址的信息，包括WindowsNT和windows2008。
2. 判断连接的合法身份。即，为防止恶意的连接以及可能是无效的连接，建议在socket连接期间，判断连接的对端是否是合法的真正的连接。
3. 对于UDP连接，可以获得连接对方的IP地址和端口，从而可以判断对方的有效性和合法性；对于TCP连接，由于每次连接需要三次握手，而且还有超时机制，存在两种方式来控制。
4. 对于TCP连接，需要尽量在三次握手完成前完成判断，同时防止端口扫描的攻击。
5. 尽可能确保socket应用能通过合理设置的防火墙。
6. 在可能的情况下，尽量减少socket连接数目。
7. 尽量不能使用回拨的技术。
8. 尽量采用有连接状态的协议，例如TCP协议。由于防火墙一般采取禁止一切的策略，对于UDP协议比较难以设置。
9. 在一个应用程序中，尽量使用同一种协议，不能使用多种协议。
10. 尽量将客户端和服务器端的端口做成可以配置，不能硬编码在程序中。

## JAVA安全开发要求

JAVA语言安全规范参考OWASPTOP10要求，本指南列举了常见的

JAVA开发安全要求。

### 防范跨站脚本（XSS）

跨站脚本是最普遍的Web应用安全漏洞。当应用程序在发送给浏览器的页面中包含用户提供的数据，但没有经过适当验证或转译，就容易导致跨站脚本漏洞。

攻击者能在受害者浏览器中执行脚本以劫持用户会话、危害网站、插入恶意内容和重定向用户等。

已知三种著名跨站漏洞是：1）存储式；2）反射式；3）基于DOM。

反射式跨站脚本通过测试或代码分析很容易找到。

防范措施：

* + 1. 验证输入

检查每个输入的有效性，主要检查输入类型和数据的长度。

* + 1. 编码输出

对验证输入的另一面就是编码输出。编码输出是指确保字符被视为数据，而不是作为HTML元字符被浏览器解析。这些技术定义一些特殊的“转义”字符，没有正确转义的数据它仍然会在浏览器中正确解析。编码输出只是让浏览器知道数据是不是要被解析，达到攻击无法实现的目的。需要编码的部分：HTML实体、HTML属性、JavaScript、CSS、URL。

### 防范SQL注入

简单来说，注入往往是应用程序缺少对输入进行安全性检查所引起的，攻击者把一些包含指令的数据发送给解释器，解释器把收到的数据转换成指令执行。注入漏洞十分普遍，通常能在SQL查询、LDAP查询、Xpath查询、OS命令、程序参数等中出现。

注入能导致数据丢失或数据破坏、缺乏可审计性或是拒绝服务，注入漏洞有时甚至能导致完全接管主机。

SQL注入包含了SQL注入、XPATH注入、LDAP注入、OS命令注入等。

### 防范恶意文件执行

恶意文件执行是一种能够威胁任何网站形式的漏洞，只要攻击者在具有引入（include）功能程式的参数中修改参数内容，Web服务器便会引入恶意程序从而受到恶意文件执行漏洞攻击。

攻击者可利用恶意文件执行漏洞进行攻击取得Web服务器控制权，进行不法利益或获取经济利益。

### 不安全的直接对象引用

所谓“不安全的对象直接引用”，即Insecure direct object references，意指一个已经授权的用户，通过更改访问时的一个参数，从而访问到原本其并没有得到授权的对象。Web应用往往在生成Web页面时会用它的真实名字，且并不会对所有的目标对象访问时检查用户权限，所以这就造成了不安全的对象直接引用的漏洞。以下是不安全的对象直接引用示例：

* 攻击者发现他自己的参数是6065，即?acct=6065；
* 他可以直接更改参数为6066，即?acct=6066；
* 这样他就可以直接看到6066用户的账户信息了；
* 这种漏洞能损害参数所引用的所有数据。除非名字空间很稀疏，否则攻击者很容易访问该类型的所有数据。

### 防范跨站请求伪造

跨站请求伪造，也被称成为“one-click attack”或者session riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。它与XSS非常不同，并且攻击方式几乎相左。XSS利用站点内的信任用户，而CSRF则通过伪装来自受信任用户的请求来利用受信任的网站。与XSS攻击相比，CSRF攻击往往不太流行（因此对其进行防范的资源也相当稀少）和难以防范，所以被认为比XSS更具危险性。

攻击者能让受害用户修改可以修改的任何数据，或者是执行允许使用的任何功能。

### 信息泄露和错误处理不当

应用程序常常产生错误信息并显示给使用者。很多时候，这些错误信息非常有用，因为它们揭示实施细则或有用的开发信息。泄露太多的细节（如错误堆栈跟踪信息、SQL语句等等）；

 登录失败后，通知用户是否用户ID或密码出错——登录失败可能是由于ID或密码错误造成的。这为一个对关键资产发动蛮力攻击的攻击者提供重要信息。

### 残缺的认证和会话管理

与认证和会话管理相关的应用程序功能往往得不到正确实施，这就导致攻击者破坏密码、密钥、会话令牌或利用实施漏洞冒充其他用户身份。

这些漏洞可能导致部分甚至全部账号遭受攻击。一旦攻击成功，攻击者能执行合法用户的任何操作，因此特权账号会造成更大的破坏。

编程要求：

* 使用内置的会话管理功能；
* 通过认证的问候；
* 使用单一的入口点；
* 确保在一开始登录SSL保护的网页；
* 获取注销的权利；
* 添加超时；
* 确保你使用的是安全相关的功能；
* 使用强大的认证；
* 不进行默认身份验证。

### 不安全的加密存储

保护敏感数据已经成为网络应用的最重要的组成部分，加密的敏感数据已是非常常见安全保护手段。不加密的应用程序、设计不当或者使用不恰当的密码技术等可能导致披露敏感数据。

* 攻击者能够取得或是篡改机密的或是私有的信息；
* 攻击者通过机密秘密的窃取从而进行进一步的攻击；
* 造成企业形象破损，用户满意度下降，甚至面临法律诉讼等。编程要求：
* 验证你的结构；
* 识别所有的敏感数据；
* 识别敏感数据存放的所有位置；
* 确保所应用的威胁模型能够应付这些攻击；
* 使用加密手段来应对威胁；
* 使用一定的机制来进行保护
* 文件加密；
* 数据库加密；
* 数据元素加密；
* 正确的使用这些机制；
* 使用标准的强算法；
* 合理的生成，分发和保护密钥；
* 准备密钥的变更；
* 验证实现方法；
* 确保所有的证书、密钥和密码都得到了安全的存放；
* 有一个安全的密钥分发和应急处理的方案。

### 不安全的通信

对于不加密的应用程序的网络信息传输，需要保护敏感的通信。加密（通常SSL）必须用于所有身份验证的连接，特别是通过Internet访问的网页，以及后端的连接。否则，应用程序将暴露身份验证或会话令牌。

* 攻击者能够取得或是篡改机密的或是私有的信息；
* 攻击者通过这些秘密的窃取从而进行进一步的攻击；
* 造成企业形象破损，用户满意度下降，甚至面临法律诉讼等。
* 编程要求：
* 提供合理的保护机制；
* 对于敏感数据的传输，对所有连接都要使用TLS；
* 在传输前对单个数据都要进行加密；（如XML-Encryption）；
* 在传输前对信息进行签名；（如XML-Signature）；
* 正确的使用这些机制；
* 使用标准的强算法；
* 合理管理密钥和证书；
* 在使用前验证SSL证书。

### 限制URL访问失效

这个漏洞事实上也是与认证相关的，与我们前面提到的不安全的直接对象引用也是类似的，不同在于这个漏洞是指系统已经对URL的访问做了限制，但这种限制却实际并没有生效。常见的错误是，我们在用户认证后只显示给用户认证过的页面和菜单选项，而实际上这些仅仅是表示层的访问控制而不能真正生效，攻击者能够很容易伪造请求直接访问未被授权的页面。

编程要求：

* 如果URL不是公开的，那么必须限制能够访问的授权用户；
* 加强基于用户或角色的访问控制；
* 完全禁止访问未被授权的页面类型（如配置文件、日志文件、源文件等）；
* 验证你的构架；
* 在每一个层次都使用简单肯定的模型；
* 确保每一层都有一个访问机制；
* 验证你的实现；
* 不能使用自动化的分析工具；
* 确保每个URL都被外部过滤器或其他机制保护；
* 确保服务器的配置不允许对非授权页面的访问。

## PHP安全开发要求

### 变量滥用

PHP-4.1.0发布的时候建议关闭register\_globals，并提供了7个特殊的数组变量来使用各种变量。对于从GET、POST、COOKIE等来的变量并不会直接注册成变量，必需通过数组变量来存取。PHP-4.2.0发布的时候，php.ini默认配置就是register\_globals=Off。这使得程序使用PHP自身初始化的默认值，一般为0，避免了攻击者控制判断变量。通过以下解决方法实现：

配置文件php.ini设置register\_globals=Off。

要求程序员对作为判断的变量在程序最开始初始化一个值。

### 文件打开

如非特殊需要，把php的文件操作限制在Web目录里面。以下是修改apache配置文件httpd.conf的一个例子：

<Directory/usr/local/apache/htdocs>php\_admin\_valueopen\_basedir/usr/local/apache/htdocs

</Directory>

重启apache后，/usr/local/apache/htdocs目录下的PHP脚本就只能操作它自己目录下的文件了，否则PHP就会报错：

Warning:open\_basedirrestrictionineffect.Fileisinwrongdirectoryinxxxonlinexx.

使用safe\_mode模式也能避免这种问题，前面已经讨论过。

### 文件包含

要求程序员包含文件里的参数尽量不能使用变量，如果使用变量，就一定要严格检查要包含的文件名，绝对不能由用户任意指定。如前面文件打开中限PHP操作路径是一个必要的选项。另外，如非特殊需要，一定要关闭PHP的远程文件打开功能。修改php.ini文件：

allow\_url\_fopen=Off

重启apache。

### 文件上传

PHP-4.0.3以后提供了is\_uploaded\_file和move\_uploaded\_file函数，可以检查操作的文件是否是用户上传的文件，从而避免把系统文件拷贝到Web目录。

使用$HTTP\_POST\_FILES或$\_FILES数组来读取用户上传的文件变量。严格检查上传变量。比如不允许是php脚本文件。把PHP脚本操作限制在Web目录可以避免程序员使用copy函数把系统文件拷贝到Web目录。move\_uploaded\_file不受open\_basedir的限制，所以不必修改php.ini里upload\_tmp\_dir的值。把PHP脚本用phpencode进行加密，避免由于copy操作泄漏源码。严格配置文件和目录的权限，只允许上传的目录能够让nobody用户可写。

对于上传目录去掉PHP解释功能，可以通过修改httpd.conf实现：

<Directory/usr/local/apache/htdocs/upload>php\_flagengineoff

#如果是php3换成php3\_engineoff

</Directory>

重启apache，upload目录的php文件就不能被apache解释了，即使上传了php文件也没有问题，只能直接显示源码。

### 命令执行

解决方法：

要求程序员使用escapeshellcmd()函数过滤用户输入的shell命令。启用safe\_mode可以杜绝很多执行命令的问题，不过要注意PHP的版本一定要是最新的，小于PHP-4.2.2的都可能绕过safe\_mode的限制去执行命令。

变量类型缺陷逻辑比较时注意变量类型。

必要的时候使用"==="，那么连变量类型一起比较。

### 警告及错误信息

修改php.ini中关于Errorhandlingandlogging部分内容：

error\_reporting=E\_ALLdisplay\_errors=Offlog\_errors=Onerror\_log=/usr/local/apache/logs/php\_error.log

然后重启apache，注意文件/usr/local/apache/logs/php\_error.log，必需可

以让nobody用户可写。

### PHP与MySQL组合的SQL注入

解决方法：

要求程序员对所有用户提交的要放到SQL语句的变量进行过滤。

即使是数字类型的字段，变量也要用单引号扩起来，MySQL自己会把字串处理成数字。

在MySQL里不能给PHP程序高级别权限的用户，只允许对自己的库进行操作。

### 跨站脚本

解决方法：

确认输入

strip\_tags()

htmlspecialchars()

清除危险的插入点。

### 禁用无用的函数

如果觉得有些函数还有威胁，可以设置php.ini里的disable\_functions（这个选项不能在httpd.conf里设置），比如：disable\_functions=phpinfo,get\_cfg\_var

可以指定多个函数，用逗号分开。重启apache后，phpinfo,get\_cfg\_var函数都被禁止了。建议关闭函数phpinfo,get\_cfg\_var，这两个函数容易泄漏服务器信息，而且没有实际用处。

### 禁用某些类

这个选项是从PHP-4.3.2开始才有的，它可以禁用某些类，如果有多个用逗号分隔类名。disable\_classes也不能在httpd.conf里设置，只能在php.ini配置文件里修改。

### 限制脚本操作路径

前面分析例程的时候也多次提到用open\_basedir对脚本操作路径进行限制，这里再介绍一下它的特性。用open\_basedir指定的限制实际上是前缀，不是目录名。也就是说"open\_basedir=/dir/incl"也会允许访问"/dir/include"和"/dir/incls"，如果它们存在的话。如果要将访问限制在仅为指定的目录，用斜线结束路径名。例如："open\_basedir=/dir/incl/"。

可以设置多个目录，在Windows中，用分号分隔目录。在任何其它系统中用冒号分隔目录。作为Apache模块时，父目录中的open\_basedir路径自动被继承。

### 其他安全配置

1. 取消其它用户对常用、重要系统命令的读写执行权限

一般管理员维护只需一个普通用户和管理用户，除了这两个用户，给其它用户能够执行和访问的东西应该越少越好，所以取消其它用户对常用、重要系统命令的读写执行权限能在程序或者服务出现漏洞的时候给攻击者带来很大的迷惑。记住一定要连读的权限也去掉，否则在linux下可以用/lib/ld-linux.so.2/bin/ls这种方式来执行。

如果要取消程序如果是在chroot环境里，这个工作比较容易实现，否则，这项工作还是有些挑战的。因为取消一些程序的执行权限会导致一些服务运行不正常。PHP的mail函数需要/bin/sh去调用sendmail发信，所以/bin/bash的执行权限不能去掉。

1. 去掉apache日志其它用户的读权限：

apache的access-log给一些出现本地包含漏洞的程序提供了方便之门。通过提交包含PHP代码的URL，可以使access-log包含PHP代码，那么把包含文件指向access-log就可以执行那些PHP代码，从而获得本地访问权限；

如果有其它虚拟主机，也应该相应去掉该日志文件其它用户的读权限；

当然，如果你按照前面介绍的配置PHP那么一般已经是无法读取日志文件了。

1. 保持运行环境干净。
2. 不能在Web目录放测试文件。

## C/C++安全开发要求

C本质上是不安全的编程语言。例如如果不谨慎使用的话，其大多数标准的字符串库函数有可能被用来进行缓冲区攻击或者格式字符串攻击。但是，由于其灵活性、快速和相对容易掌握，它是一个广泛使用的编程语言。下面是针对开发安全的C语言程序的一些规范。

### 缓冲区溢出

避免使用不执行边界检查的字符串函数，因为它们可能被用来进行缓冲区溢出攻击。下面是应该避免使用的函数。同时，也列出了每个函数相应的比较安全的替换方式。

不使用strcpy()，使用strncpy()；

不使用strcat()，使用strncat()；

不使用sprintf()，使用snprintf()；

不使用gets()，使用fgets()。

在上面的前三个中函数中，每个替代函数的“n”表示了使用的缓冲区的大小。最后一个函数的“f”，表示格式，它允许用户指定期望的输入的格式。这些替换方程强制程序员定义使用的缓冲区的尺寸以及确定输入的类型。

### 格式化字符串攻击

该类攻击往往与缓冲区溢出相关，因为它们主要利用了某些函数的假设，例如sprintf()和vsprintf()假设缓冲区的长度是无限的。然而即使使用snprintf()替换sprintf()也无法完全保护程序不受格式化字符串的攻击。这些攻击通过直接将格式说明符（formatspecifiers）(%d，%s，%n等)传递到输出函数接收缓冲区来进行。

例如，以下的代码就是不安全的snprintf(buffer,sizeof(buffer),string)这种情况下，可以在字符串中插入格式说明符来操纵内存的栈，来写入攻击者的数据（这些数据中包含小的程序代码，并可由处理器接着执行）。对以上的例子建议使用下面的代码。

snprintf(buffer,sizeof(buffer),“%s”,string)进行格式字符串攻击不太容易。首先攻击者必须能获得内存栈的内容情况（或者从应用导出或者使用调试器），然后必须知道如何精确访问特定的内存空间来操纵栈中的变量。

执行外部程序推荐使用exec()函数而不是system()函数来执行外部程序。这是因为system()接收整个命令行的随机的缓冲区来执行程序。

snprintf(buffer,sizeof(buffer),"emacs%s",filename);

system(buffer);

在以上的例子中，可以通过使用分号利用文件名变量在sehll中插入额外的命令（例如文件名可以是/etc/hosts;rm\*，这将在显示/etc/hosts目录文件的同时，删除目录中的所有文件）。而exec()函数只保证第一个参数被执行：

execl("usr/bin/emacs","usr/bin/emacs",filename,NULL);

上面的例子保证文件名仅仅作为一个参数输入Emacs工具，同样它在Emacs命令中使用完全的路径而不是使用可以被攻击者利用的PATH环境变量。

### 竞争条件

进程需要访问资源时（无论是磁盘、内存或是文件）通常需要执行两个步骤：

1、首先测试资源是否空闲可用；

2、如果可用，就访问该资源，否则它等到资源不再使用为止再去访问它。当另一个进程在步骤1和2之间想要访问同一个资源时就出现问题了。

这会导致不可预测的结果。进程可能会被锁定，或者一个进程劫持获得了另一个进程的较大的权限而导致安全问题。攻击主要集中在有较大权限的程序上（称为setuid程序）。竞争条件攻击通常利用程序执行时可以访问到的资源。另外权限低的程序也存在安全风险，因为攻击者可能会等待有较高权限的用户执行那个程序(例如root)，然后进行攻击。

下面的建议有助于缓解竞争条件(racecondition)攻击：

在进行文件操作时，利用那些使用文件描述符的函数而不能使用那些使用文件路径的函数（例如使用fdopen()而不能使用fopen()）。文件描述符使得恶意的用户在文件打开时或是在原始的进程对文件进行操作前，无法使用文件连接（符号式的或是物理的）来改变文件。

在写文件甚至在读文件时使用fcntl()和flock()函数来对文件加锁，这样它们就不能被其他进程访问。它几乎可以建立原子级的操作。

谨慎操纵临时文件，因为它往往会导致竞争条件。

### 检验有效的返回值

检验有效的返回值非常重要。一个例子是旧的/bin/login的实现中不检验错误的返回值，导致当它找不到/etc/passwd文件时返回root的访问权限。如果该文件损坏了，那么这种情况是合理的，但如果该文件存在只是无法访问，那么这就是一个大问题。

## Perl安全开发要求

多年以来，Perl已经成为用于系统管理和WebCGI开发的功能最强的编程语言之一（几乎可以使用Perl做任何功能的程序）。但其扩展应用，即作为Internet上CGI的开发工具，使得它经常成为Web服务器上的攻击目标。

另外，大多数CGI脚本有着比一般用户更高的权限，导致它更容易受攻击。下面列举了一些开发者（特别是CGI程序员）可以使用的主动的预防性的措施来增强Perl代码的整体安全性（请注意：这不是Web服务器CGI脚本安全性的指导原则）。

### Taint验证

Perl版本5.x包含一个叫做TaintChecking的数据验证措施。如果起用该功能，将不允许通过用户输入（任何程序外的输入）来操纵其他的外部程序（例如通过管道将数据导入另一个程序执行））。一般而言，程序员不能信任输入脚本和程序的数据（叫做Tainted数据），因为无法保证它不会产生危害（有意或者无意的）。Taint验证可以通过在命令行参数加入“-T”来开启。

例如：你可以在Perl脚本的第一行这样加入“-T”：

#!usr/bin/perl5-T

Tainted数据包括命令行参数、环境变量和来自文件的数据。引用tainted

数据的变量也称为tainted数据。如果脚本试图通过不安全的方式来使用tainted数据会产生一个致命错误（对这种情况称为“不安全的依赖”(Insecuredependency)或者其他的说法）。启用tainted验证在有些情况下会导致脚本停止运行，常常是由于Perl解释器要求所有脚本引用的外部程序的完全路径必须在PATH环境变量中列出，同时PATH中包含的每个目录除了目录的所有者及相应的所有者用户组外无法修改。Taint验证对于环境比较敏感，但是只要可能的话，应该使用taint验证，特别是代码执行其他程序功能时（例如在CGI脚本的情况下）。

### 安全模块

如果不但输入数据不可信而且实际的代码也不可信，例如用户从网站上下载了一个ActiveX控件，而它实际是一个特洛伊木马(Trojanhorse)。这种情况下taint验证就不起作用。安全模块让程序员可以在Perl脚本中将不同的代码模块与安全对象联系。每个安全对象对于运行的每块代码建立了一个限制的环境。这与chroot在一个进程中只能在整体目录结构的一个子目录中运行类似。而saft对象限制perl代码只能在perl包结构的某些特定包中运行。如何使用安全模式超出了本文的范围，但是程序员应该在任何时候尽量使用这一功能。

### 警告参数

使用-w参数可以在Perl解释脚本时显示所有的警告信息。警告可以在以下情况产生：只使用了一次的变量或者完全没有使用过得变量，未定义的文件句柄，未关闭的文件句柄，或者将非数值变量传递到数据变量。该功能不是针对安全处理的，但是可以有助于调试直接或者间接对安全有危害的错误。一般推荐总是使用-w参数。可以在taint验证时在第一行这样使用-w参数：

#!usr/bin/perl5-Tw

# **应用系统测试安全**

应用系统在正式上线前应对安全性进行测试，验证应用系统的安全性是否符合安全设计及安全需求，应用系统的安全测试包括以下内容：

## 测试前的要求

测试之前，首先要确定测试大纲，在测试大纲中应当明确应用安全方面的测试内容、手段及方案（即测试用例）。包括：

* 安全性测试的环境要求；
* 安全性测试软件、测试设备要求；
* 安全性测试人员要求；
* 安全性测试的时间要求；
* 安全性测试的内容；
* 安全性测试的输入数据；安全性测试的预期结果；
* 安全性测试的详细过程；
* 安全性测试的风险和风险规避方案。

测试大纲中安全性方面的描述是进行安全性测试的一个重要依据。大纲内容的编写应当依据应用系统需求说明书及应用系统设计说明书中，有关安全方面的内容来编写相应的测试用例。

## 测试方法及内容要求

1. 应根据应用系统的具体情况，设计安全性测试的具体方法和内容，通常主要的测试方法和内容分为功能性测试、压力测试、渗透性测试、审核性测试。
2. 功能性测试要求：对应用系统的安全功能点进行测试，确保安全功能的有效性、正确性。
   * 这些功能点应全部包含在测试大纲中，在大纲的测试用例中体现具体的测试方法。测试大纲中的这些内容应当覆盖了需求说明书中安全功能部分的要求。
   * 测试的具体内容主要涉及以下几个方面：
     + 安全审计
     + 通信安全
     + 数据保护
     + 身份认证
     + 容错及资源分配方面的要求
3. 压力测试：对应用系统的安全功能进行压力测试，确保安全功能可以满足设计的需要。
   * 压力测试的具体应全部包含在测试大纲中，在大纲的测试用例中体现具体的测试方法。
   * 测试大纲中的这些要求应当参照设计说明书中所描述的设计目标进

行测试用例的设计，包括但不限于：

* + - 应用系统服务器端和单个终端进行安全数据传输的最大容量。
    - 应用系统服务器端能够与多少终端同时进行安全数据传输。
    - 服务器承受的最大并发连接数。

1. 渗透性测试：模拟攻击手段，对应用系统抵抗攻击的能力进行测试。主要测试应用程序及应用系统配置上的安全漏洞。
   * 渗透测试的具体应全部包含在测试大纲中，在大纲的测试用例中体现具体的测试方法。
   * 测试大纲中渗透测试方法的选择需要依据设计说明书中描述的应用系统采用的开发工具及应用系统安装所需要的基础应用平台，如iis、apached等，来设计不同的测试方法，包括但不限于：
     + 对webserver的渗透攻击测试或者工具扫描性测试。
     + SQL注入、跨栈攻击。
2. 审核性测试：
   * 代码审核。这不是必需的测试手段，如果条件允许，可以针对容易出现代码漏洞的程序，特别是关于身份认证等方面的部分代码进行审核。
   * 可以采用开发相关的专用扫描工具，或者编制相应的脚本程序，检查代码中的不安全函数的使用，如：strcpy()，strcat()等。
3. 测试过程中，应详细记录测试过程发生的每一件事情，列出测试过程中

发现的问题。这些信息包括：发现了什么，在哪里发现的，当时的环境，这些问题是否可重现。

1. 应根据测试的过程和测试结果，提出被测试系统、测试过程等方面的改进说明。
2. 应确保测试用例、测试内容和测试结果的保密性。

## 测试环境及人员安全要求

1. 测试环境的物理、硬软件环境要求应当可以模拟真实环境。
2. 为确保测试环境的安全，应将测试环境与开发环境、生产环境相隔离，避免测试工作对业务的影响。
3. 测试数据如果选择是真实数据，应限定使用该数据的测试人员的数量，并在测试完成后全部删除。
4. 系统测试和验收通常需要大量的（真实数据）尽可能靠近实际运行数据的测试数据。应避免使用含有个人信息的业务数据库。如果要使用其中信息，在用之前应使其失去个性化。当把运行数据用于测试目的时，应采取以下措施保护运行数据。

* 如果该数据由测试人员使用，则需要相关授权业务人员参与，或者监督。
* 每次把运行信息复制到测试应用系统都应有单独的授权。
* 应记录运行信息的复制和使用，以提供一种检查追踪。

1. 在与其他系统的互操作性测试中，应充分考虑对其他系统的影响，选择适当的时间、方法。

## 测试后的要求

测试完成后，在正式上线前，应进行安全检查，消除测试用的一切后门、用户名及口令等，包括但不限于：

1. 测试用的测试数据，即测试过程中的临时数据，特别注意要删除授权访问的那些重要测试数据。
2. 测试用的账号、口令，特别是有高级权限的账号。
3. 测试用时临时开通的系统服务、防火墙端口及策略。
4. 与外系统测试时，要求外系统上临时开启的服务、账号等。