**软件安全开发编码规范**

1. **代码编写**
2. 开发人员应保证工程中不存在无用的资源（如代码、图片文件等）。
3. 代码中每个类名上的注释必须留下创建者和修改者的名字。
4. 每个需要import的类都应使用一行import声明，不得使用import xxx.\*。
5. System.out.println()仅在调试时使用，正式代码里不应出现。
6. 开发人员编写代码时应遵循以下命名规则：

* Package 名称应该都是由一组小写字母组成；
* Class 名称中的每个单词的首字母必须大写；
* Static Final 变量的名称全用大写，并且名称后加注释；
* 参数的名称必须和变量的命名规范一致；
* 使用有意义的参数命名，如果可能的话，使用和要赋值的字段一样的名称。

1. 代码应该用unix的格式，而不是windows的。
2. exit 除了在 main 中可以被调用外，其他的地方不应被调用。
3. 代码中应尽量使用interfaces，不要使用abstract类。
4. 在需要换行的情况下，尽量使用 println 来代替在字符串中使用的"\n"。
5. 涉及HTML的文档，尽量使用XHTML1.0 transitional文件类型，其中所有HTML标签都应关闭。
6. 在HTML、JavaScript、XML代码中，缩进应为两个空格，不得使用Tab。
7. HTML标签的name和id属性的命名方式应与Java变量名相同。
8. 在需要经常创建开销较大的对象时，开发人员应考虑使用对象池。
9. 在进行log的获取时开发人员应尽量使用isXXXEnabled。
10. log的生成环境上尽量避免输出文件名和行号。
11. 产品中不要包含后门代码，隔离系统中的后门代码，确保其不能出现在产品中。作为一种特殊的调试代码，后门访问代码是为了使开发者和测试工程师访问一部分终端用户不能访问的程序代码。但是，如果后门代码被留到产品中，对攻击者来说，它就是一条不需要通过正常安全手段来攻陷系统的通路。
12. **JAVA安全**

遵循下面列出的准则有利于编写更加安全的代码。但是总体来说，这些准则不能对安全性做出任何保证。遵循这些准则可能好的实践，但是即使遵循了这些准则，写出的代码仍然可能是不安全的。风险永远存在，不管在编写代码时是如何的警觉。

这些准则的目标，不是为了保证代码的安全性，而是为了消除若干特定类型攻击带来的风险。遵循这些准则，某些特定类型的攻击将无法实现；但是其它类型的攻击仍然可能成功。因此遵循这些准则仅仅是安全的第一步。当书写可能和非守信链接或混用的代码时，应当仔细的考虑如下准则：

* + 静态字段
  + 缩小作用域
  + 公共方法和字段
  + 保护包
  + 尽可能使对象不可变（immutable）
  + 序列化
  + 清除敏感信息

1. 静态字段

避免使用非final的公共静态变量，应尽可能地避免使用非final公共静态变量，因为无法判断代码有无权限改变这些静态变量的值。

一般地，应谨慎使用可变的静态状态，因为这可能导致设想中应该相互独立的子系统之间发生不曾预期的交互。

1. 缩小作用域

作为一个惯例，尽可能缩小成员方法和成员变量的作用域。检查包访问权限成员（package-private）能否改成私有成员（private），保护访问成员（protected）可否改成包访问权限成员（package-private）/私有成员（private）等等。

1. 公共方法/字段

公共变量应当避免使用，访问这些变量时应当通过getter/setter法。在这种方式下，必要时可以增加集中的安全检查。

任何能够访问或修改任何敏感内部状态的公共方法，务必包含安全检查。

参考如下代码段，该代码段中不可信任代码可能修改TimeZone的值：

private static TimeZone defaultZone = null;

public static synchronized void setDefault(TimeZone zone)

{

defaultZone = zone;

}

1. 保护包

有时需要整体上保护一个包以避免不可信任代码的访问，本节描述了一些防护技术：

防止包注入：如果不可信任代码想要访问类的包保护成员，可能通过在被攻击的包内定义自己的新类用以获取这些成员的访问权的方式。防止这类攻击的方式有两种：

* 1. 通过向java.security.properties文件中加入如下文字防止包内被注入恶意类。

...

package.definition=Package#1 [,Package#2,...,Package#n]

...

当检测到代码试图在包内定义新类时，类装载器的defineClass方法会抛出异常，除非代码被赋予以下权限：

...

RuntimePermission("defineClassInPackage."+package)

...

* 1. 另一种方式是通过将包放到封闭的JAR（sealed Jar）文件里。

（参看http://java.sun.com/j2se/sdk/1.2/docs/guide/extensions/spec.html）

通过使用这种技巧，代码无法获得扩展包的权限，因此也无须修改java.security.properties文件。

防止包访问：可以通过限制包访问但同时仅赋予特定代码访问权限防止不可信任代码对包成员的访问。通过向java.security.properties文件中加入如下文字可以达到这一目的：

...

package.access=Package#1 [,Package#2,...,Package#n]

...

当检测到代码试图访问上述包中的类时，类加载器的loadClass方法会抛出异常，除非代码被赋予以下权限：

...

RuntimePermission("defineClassInPackage."+package)

...

1. 尽可能使对象不可变（immutable）

尽可能使对象不可变。如果对象必须改变，使得它们可以克隆并在方法调用时返回副本。如果方法调用的返回对象是数组、向量或哈希表等，牢记这些对象并非不可变，调用者可以修改这些对象的内容并导致安全漏洞。此外，不可变的对象因为不用上锁所以能够提高并发性。

不要返回包含敏感数据的内部数组引用。

这个不可变惯例的变型，在这儿提出是因为是个常见错误。即使数组中包含不可变的对象比如说是字符串，也要返回一个副本，这样调用者不能修改数组中包含的到底是哪个字符串。在方法调用返回时，返回数据的拷贝而不要返回数组。

1. 不要直接在用户提供的数组里存储

这是不可变惯例的另一个变型。构造器和方法可以接受对象数组，比如说PubicKey数组，这个数据存储到内部之前应当克隆，并保存克隆后的数据，而不是直接将数组引用赋给同样类型的内部变量。如果缺少这个步骤，在使用了有问题的构造器创建了对象后，用户对外部数组所作的任何修改都将更改对象的内部状态，尽管对象应该是不可变的。

1. 序列化

对象在序列化后、反序列化之前，都不在Java运行时环境的控制之下，也因此不在Java平台提供的安全控制范围内。

在实现接口Serializable时务必将以下事宜牢记在心：

* transient

直接引用系统资源的句柄和包含了地址空间相关信息的字段应当使用关键字transient修饰。资源，如文件句柄，如果不被声明为transient，该对象在序列化状态下可能会被修改，从而在被反序列化后获取对资源的不当访问。

* 特定类的序列化/反序列化方法

为了确保反序列化对象不包含违反一些不变量集合的状态，类应该定义自己的反序列化方法并使用接口ObjectInputValidation验证这些变量。

如果一个类定义了自己的序列化方法，它就不能向任何DataInput/DataOuput方法传递内部数组。所有的DataInput/DataOuput方法都能被重写。注意默认序列化不会向DataInput/DataOuput字节数组方法暴露私有字节数组字段。

如果Serializable类直接向DataOutput(write(byte [] b))方法传递了一个私有数组，那么黑客可以创建ObjectOutputStream的子类并覆盖write(byte [] b)方法，这样他可以访问并修改私有数组。下面示例说明了这个问题。

示例类：

public class YourClass implements Serializable {

private byte [] internalArray;

....

private synchronized void writeObject(ObjectOutputStream stream) {

...

stream.write(internalArray);

...

}

}

黑客代码：

public class HackerObjectOutputStream extends ObjectOutputStream{

public void write (byte [] b) {

Modify b

}

}

...

YourClass yc = new YourClass();

...

HackerObjectOutputStream hoos = new HackerObjectOutputStream();

hoos.writeObject(yc);

* 字节流加密

另一种保护位于虚拟机之外的字节流的方式是对序列化产生的流进行加密。字节流加密可以防止解码和读取被序列化对象的私有状态。如果决定加密，需要管理好密钥，密钥的存储以及密钥交付给反序列化程序的方式，等等。

* 需要注意的其它事宜

如果不可信任代码在创建对象时受到约束，务必确保不可信任代码在反序列化对象时受到相同的约束。牢记对象反序列化是创建对象的另一途径。

比如说，如果applet创建了frame，在该frame上创建了警告标签。如果该frame被应用程序序列化并被applet反序列化，务必使该frame在反序列化后标有相同的警告标签。

1. 本地方法

应从以下几个方面检查本地方法：

* + 返回什么
  + 需要什么参数
  + 是否绕过了安全检查
  + 是否是公共的，私有的等
  + 是否包含能绕过包边界的方法调用，从而绕过包保护

1. 清除敏感信息

当保存敏感信息时，如信用信息，尽量保存在如数组这样的可变数据类型中，而不是保存在字符串这样的不可变对象中，这样使得敏感信息可以尽早显式地被清除。不要指望Java平台的自动垃圾回收来做这种清除，因为回收器可能不会清除这段内存，或者很久后才会回收。尽早清除信息使得来自虚拟机外部的堆检查攻击变得困难。

1. **数据库安全**
2. 开发人员应尽量使用PreparedStatement，并且使用占位符？来表示参数。在使用set命令时，数据库驱动程序会对参数中的关键字进行转义。严格禁止将参数和SQL语句做拼接。
3. 只给数据库用户授予其需要的最小权限，以保障数据库服务器的安全。
4. 当使用JDBC操作数据库时，涉及到的资源包括ResultSet、Statement、Connection都必须及时关闭。
5. ResultSet、PreparedStatement、Connection必须依次关闭，同时三者的close方法都应提示异常，且每个close方法都必须用try、catch来实现。
6. 数据库关闭的原则是：谁创建的资源，谁负责关闭。
7. 应在try代码块中及时关闭数据库资源，同时finally的代码块中也要关闭资源，或者将一个try代码块拆分为多个try代码块，保证每个资源都能在使用完以后立即关闭。
8. 数据库表名、字段名必须大写。
9. 对于返回较大结果集的查询，必须禁止SELECT \*，在其他查询中也应避免使用。
10. 编写可以移植的SQL语句，原则如下：

* 不得使用某个数据库专用的关键字、函数等；
* 当必须要使用某个数据库特定的特性时，需在程序运行时，先判断当前数据库的类型，然后再根据数据的不同使用其特性；
* 可以使用各种数据库都支持的函数包括MIN、MAX、AVG、COUNT；
* 尽量使用简单的SQL语句，当因为特殊情况需要使用非常见SQL语句时，应该在多种数据库下测试。

1. 优化SQL语句时开发人员应遵循以下原则：

* 使用合适的SQL语句以避免不必要的关联；
* 使用JDBC批量更新来优化insert和update的性能；
* 必要时可以使用对象缓存技术，但是技术方案需要通过讨论并且获得批准后方可执行。

1. 不得将数据库的用户名和密码以明文形式存储在配置文件中。
2. 对于存储于数据库中的重要数据以密文形式存放，可以大大增强数据的安全性。
3. **WEB安全**
4. 独立、完整且集中的输入验证
5. 校验全部的程序输入
6. 校验全部的输入长度
7. 校验全部的输入类型
8. 不使用任何方式处理失败的数据
9. 对HTTP所有内容进行校验
10. 校验向用户输出的数据
11. 只相信服务器端校验，客户端校验只能作为补充
12. 使用安全、统一的编码或转义方式
13. 设定有安全的权限边界
14. 校验被调用的后台命令
15. 校验被调用的文本或配置文件
16. 在HTML中，一些特殊字符在页面上显示时必须转义。
17. 用户界面须支持主流浏览器，避免因某类浏览器的安全问题或者在非IE浏览器下用户界面不能常驻。
18. 用户界面应该包含公司或者产品标识。
19. 尽量使用 POST 而不是 GET方式。使用 HTTP POST 方法来保证 Request 参数的安全。
20. 创建一个默认的错误页面 。对所有的异常构造统一的错误页面，包括 HTTP错误和未经处理的异常。
21. 在默认错误页面中使用通用的错误消息。要确定错误提示信息不会泄露系统信息和出错原因等敏感信息。精心构造错误提示信息来防止诸如用户 id，网络，应用程序以及服务器环境的细节等重要的敏感信息的泄漏。
22. 使用较强的会话标识符，如使用包含至少 128 位安全随机数密码的会话标示符。
23. 除了完全公开的Web页面，对于其他所有Web页面，需要验证访问用户是否具有访问权限。
24. **日志安全**
25. 对每个重要的行为都记录日志。如认证尝试、重要传输、重要数据更改、管理行为等。在上述事件的日志定义行为中，要注意失败事件的关注程度至少要与成功事件一样，因为这些失败事件经常会发生在安全事件之前。
26. 保护日志文件。安全地保存日志文件，主要方法是将日志文件独立保存于应用程序目录外，同时通过严格的权限设置来控制对日志文件的访问。