DOI: 10.15907/j.cnki.23-1450.2014.10.113

文章编号：1008-8717（2014）10-0142-04

白盒代码安全审计方法浅析

李 俊
（牡丹江大学信电学院，黑龙江 牡丹江 157011）

摘 要：白盒代码安全审计是常用的安全审计方式之一，是基于静态遍历代码逻辑的一种审计方式。本文将总结一些在白盒代码安全审计过程中的经验和过程方式，这些内容有助于了解其一系列流程，并总结分析出要点难点。借鉴这些资料，相信可以减少很多白盒代码安全审计不必要的工作量。

关键字：白盒；代码安全；审计
中图分类号：TP319 **文献标识码**：A

一、先期准备

1. 知识准备

在白盒代码安全审计之前，需要有一定的知识准备。首先需要参与审查人员对被审代码所使用的技术有一定的安全相关知识储备，审查人需要了解此技术常见的漏洞体现形式。并且对被审代码中所使用到的第三方库或者第三方软件的机理和安全性有一定了解。只有这样，才能有针对性地规划代码审计中的要点和重点，并有效地提高工作效率。

2. 工具准备

如果审计是在 Unix 环境下，那么常见的工具基本就齐全了。我们可以使用 Grep、find 命令结合 shell 编程，来完成一定量的搜索查找工作。同时，在审计环境中还需要一些常用的脚本工具的解析器（PHP、PERL、PYTHON），为测试时自己编写一些脚本时使用。另外还可以使用网上的一些白盒审计工具，比如常用的 pixy 等，使用这些工具进行辅助性的白盒审计。

3. 测试环境准备

在拿到被审代码之后，可以视情况，在本地搭建一个模拟在线服务器配置的环境。被审代码所需的服务器和脚本，相关的数据库，一切以线上环境为基准，做到尽量模拟真实环境。以辅助验证白盒检查出来的结果。如果被测产品线可以直接提供主机模拟线上的测试环境，则需要阅读日志，线上系统源代码查看、数据库使用、应用程序账号等相关权限，以支持验证最终测试结果。如果有可能，最好还有可远程调试的接口，可以方便的打印追踪变量。

4. 关键词字典

关键词字典是在白盒代码安全审计过程中最重要的部分之一，审查人在测试过程中，依照这些关键词对应用程序代码进行匹配，并根据匹配结果进行排查和检测。根据被审代码所使用的技术和底层框架不同，审计时的关键词字典也不同，需要根据技术的发展和经验的提升实时更新。一般情况下，关键词字典内容包括：a.可能在系统上执行命令或代码的函数；b.能够获取用户输入的方法；c.可能对系统变量或环境变量造成覆盖的函数；d.可能泄露或打印系统信息的函数；e.直接向用户端输出的函数；f.一切涉及到数据库执行操作的函数；g.可以在服务端操作文件的函数；h.可能操作环境变量的函数；i.操作用户文件上传的函数；j.可能远程调用或包含资源的函数。

5. 规范规则

SSL 组已经制订了一系列系统开发安全编码规范，在审核相应技术开发的应用程序时，需要首先熟悉了解安全编码规范，依照安全编码规范中的内容，完善关键词字典，并在代码审计过程中时刻注意检查规范的执行情况。如果暂时没有此项技术

收稿日期：2014-05-17
作者简介：李俊（1978—），女，哈尔滨工业大学毕业，牡丹江大学信电学院副教授，研究方向：计算机教学。

的安全规范，可以在白盒审核的过程中，逐步总结出一些内容，留待以后整理并形成一个完整的安全编码规范。

二、基本方法

在代码安全审计的实施阶段，有一系列常用的方式方法，一般来说，在下文中提到的头两种方式自上而下和自下而上两种方法可以检测绝大多数安全漏洞。比较适用于日常的审核。

1. 自上而下

所谓自上而下的方法，这是基于应用程序的生命周期而言的。应用程序收到用户请求，并处理逻辑操作，最后输出结果返回给用户。

同样的道理，所谓的自上而下，就是跟踪所有的用户或环境变量的输入。包括$_GET、$_POST、$_FILES、$_COOKIE、$_ENV、$_SERVER 等所有可能直接或间接被用户控制的变量，以及一些可能造成内部变量污染的函数或方法（extract，getenv 等）。从接受参数开始，一步步顺着代码逻辑遍历跟踪，一直到找到可能有安全威胁的代码，或者，直到所有的输入都被过滤或限定为安全为止。

值得注意的是，如果应用程序使用了第三方的框架或者库，则很有可能其接受的输入参数都被框架或库函数包装了一层或多层代码。在对这些输入进行跟踪的时候，注意也要跟踪这些库函数，否则可能会造成一定区域的盲点。

2. 自下而上

自下而上的方法和自上而下的方法正好相反，是根据敏感函数的关键词字典，从应用点回溯其接受的参数，一步步向上跟踪，直到排除嫌疑或发现安全隐患为止。

对于此方法，需要对这些敏感函数的内部机理和使用方式非常了解，这样才能判断某些非法参数的输入是否会有安全风险，会造成什么样的风险。

3. 逻辑路径覆盖

逻辑路径覆盖是基于代码业务逻辑的一种遍历测试方式。方法是根据代码的逻辑和生命周期，使用调试器或者人工遍历所有可能的路径，从中发现安全隐患。逻辑路径覆盖方法的主要目标是发现程序员在编写代码的过程中所引起的逻辑上的漏洞，通常情况下这些问题很难在黑盒或灰盒测试中显现出来，在上文提到的两种审计方法中也较难被发现。

由于逻辑路径覆盖方式审计所需要花费的人力成本非常高，此方法的产出和投入之比非常的低。在通常情况下，在经过常规检测之后的代码逻辑安全问题更为稀少和隐蔽，难以被安全审计人员发现。而外界恶意攻击者由于无法得到我们的产品代码，更难从简单的黑盒测试中找到这些问题。因此建议，除非是安全性要求等级非常高的应用，其他应用无需过多的使用此类方法进行审计。

4. 框架安全

在审计应用的代码安全时，如果此应用使用了公司内部开发的或第三方框架或者代码库，应该在对应用程序核心代码开始审计前先对框架的代码进行审计，并优先了解框架中数据获取，数据传输，数据过滤，数据输出，文件上传，敏感操作调用，数据库操作等内容的运作原理。其主要目的有三个：

a.检测底层库中的安全漏洞和隐患；

b.依据现行安全编码规范对框架或代码库进行评估和总结，完善现有规范和底层库；

c.在底层库中找出可能引发安全问题的敏感函数或方法，并归纳为一个字典。在接下来审查该应用核心代码或者其他使用到此框架的应用时，需要将此字典加入关键词字典加以分析和检测。

5. 灰盒测试

在白盒安全审计的过程中，还需要结合灰盒测试技术对白盒审查中的结果进行核实和补充。例如，在拥有测试环境的情况下，可以基于发现的安全问题在测试环境中重现漏洞，并探索研究出测试用例，一并归入审计报告中，方便应用负责人修复。

三、审计原则

1. “所有的输入都是不可信的”

这是一句至理名言，所谓“病从口入”，也是一样的道理。本质上，应用程序就是一个将用户的输入进行处理分析再展示的过程，因此每个输入都有可能被敏感操作处理。90%以上的安全问题都是由输入的数据引起的。安全审计的过程中，审计人员具体操作时，一定要控制跟踪好所有的输入数据。每个输入都需要排查，需要严格限定其格式和内容。

2. 照着规范来，否则就可能存在风险（同时随时补充规范）

在审查前需要制定相应技术的安全规范，并在审查中检查应用程序是否严格遵守现有的安全规范。即使没有造成安全漏洞，但是其没有遵守规范时，仍然存在安全风险，需要在白盒安全审计报告中指出。

同时，在白盒安全审计的过程中，随时归纳当前应用程序安全控制的要点和盲点，这样既可以在报告中指出，同时也可随时补充加强现有的安全规范。

3. 从危害大的问题查起，做到同成本下功效最大化。

我们可以将漏洞类型进行分级量化，并放在一个直角坐标系中，其中它的 x 轴代表“危害性”，y 轴代表“可检测性”。如下图所示：

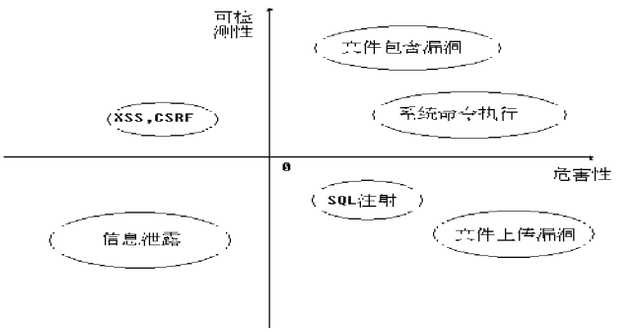

图 1. 常见 Web 应用程序漏洞

在相同成本的情况下，我们需要提高白盒审计的效率，同时将安全风险降低到最低。在审计的过程中，应该将第一目标设定为第一象限（高风险易检测），先排查被审应用在此象限内所有可能出现的安全隐患，然后再继续第二目标：第四象限（高风险难检测），接着是第三目标第二象限（低风险易检测），最后才为第三象限（低风险难检测）。

在实践过程中，应该根据被审应用的实际情况，及时调整当前任务的重点和时间分配。

4. 高安全性要求情况下，尽量做到同一代码两次或多次检查（不同人）

冗余的审计是必要的，如果条件允许，尽量多安排一位审核人员参与审核，并在不受另一位审核人员思路干扰的情况下进行白盒审核。这样可以避免在思维惯性或者高度疲劳下漏报、错报等问题，高安全性要求能够得到保证。

5. 灵活思维，多角度、大广度、大深度思考

在做代码审计的过程中，一定要保持思维的灵活连续性，不能随着程序员的思维走，一定要站在一个恶意用户的角度，从各个方向对代码的所有问题进行排查，在检查到一些异常问题时，在解决问题的同时，也要思考问题的成因，是否会波及到其他应用等。

四、简单例子

以下是一个简单的白盒代码安全审核的例子，在这些例子中，笔者使用到了上文所叙述的一系列内容。可以见到，灵活应用以上方法，将会大大提高代码安全审核的排查速度和准确率。

下面以 space 应用的代码安全审计做一个简单的代码安全审核例子，首先在开始前需要对应用有一定的了解，拿到源代码之后，先看一下目录结构，了解一下其代码组织结构：

经过了解，其中 phpsrc 为根目录，其下 conf 目录中存放着所有子项目的配置文件，目录中有 lib 的则是程序中所使用到的公司内部公共类库，而其他目录则为 space 的各个子项目。在子项目目录文件夹下，page 目录是子项目的入口，存放着最上层的业务逻辑代码，inc 目录下是子项目的一些预定义变量，phpunit 目录里是测试用例文件，可以忽略不看，剩下的目录中都是子目录可能用到的类。

既然已经知道了代码的目录结构，那么就可以开始按照常见的关键词进行匹配了。

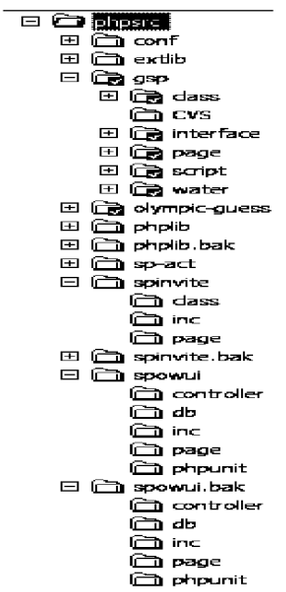

图 2. space 应用源代码目录结构

经过排查，我们看到当我们搜索 system 函数的时候，发现在 system 中有代入变量。那么，这里很有可能就有安全隐患，我们继续跟踪，按照自下而上的方法，继续回溯跟踪被传入 system 的参数。

```
F:\work\review\space\phpsrc\sp-
act\interface\class\SPMSDataInterface.class.php
00108: $from = $request['from'];
00109: $to = $request['to'];
00110:
00111: $tmp = '/tmp/act_css_tmp_' . $uri;
00112:
00113: system("/usr/bin/wget $from -O $tmp");
00114:
00115: if (file_exists($tmp)) {
00116: // $path = '/home/space/apache/htdocs/css/{ActName}.css';
00117:
00118: $targetCss = "/home/space/apache/htdocs/css/$uri.css";
00119: $bakCss = $targetCss . ".bak." . date('Ymd_His');
00120: system("cp $targetCss $bakCss");
00121: # system("mv $tmp $targetCss");
00122: system("cp $tmp $targetCss");
00123: }
00124: }
00125:
00126:
00127:
00128: if ('test' == $myFunc) {
00129: $name = $request['name'];
00130: system("echo '$name\n' >> /tmp/llk.txt");
00131: }
00132:
00133: echo $myFunc;
00134: return;
00135:
```

图 3.搜索“system”出的结果

首先，按照最大化效率的原则，我们先匹配可　　能对服务器直接进行命令操作的一些关键词函数：

```
$uri = $request['uri'];  这里的 request 参数都没有过滤
$from = $ request['from'];
$to = $request['to'];

$tmp = ' tmp act_css_tmp_' ,$uri;
system(" usr bin wget $from -0 $tmp");
```

直接被执行了，可以直接插入命令，以下的 system 函数同样有此问题。

图 4.追踪参数

很幸运，在 system 代码的前两行就回溯到了放入函数中的参数：$form 和$tmp 变量。这两个变量是直接从 request 用户输入中获取的，并且未做任何过滤！很明显，这里是一个可以直接在服务器上插入执行系统命令的高危漏洞。从上文可见，一个典型的高危漏洞，可以通过关键字匹配，人工自下而上回溯追踪，非常便利的检测出来。

五、其他内容

1. 在分析的过程中，需要对被发现的安全风险进行分级，需要测试用例，需要漏洞代码，并在审计报告中体现出来。

2. 日常代码审计的流程：申请项目审计　前期准备（测试环境，知识准备，工具准备）　第一次关键词字典准备　第三方库审计（可选）　第二次关键词字典准备（可选）　应用程序代码审计　复查（可选）　验证测试　总结报告　代码安全审计完成。

参考文献：

[1]李昕,等.开源软件安全问题与对策[J].计算机安全,2008,(4).

[2]梁婕,等.基于静态分析技术的源代码安全检测模型[J].计算机应用研究,2008,(9).