doi 10.3969/j.issn.1671-1122.2012.10.018

一种 Flash 运行时注入查找工具的设计

卢阳[1]，叶超[2]，严寒冰[3]，文伟平[1]

（1. 北京大学 软件与微电子学院信息安全系，北京 102600；
2. 北京市东城区公安消防支队，北京 100061；
3. 国家计算机网络应急技术处理协调中心，北京 100029）

摘 要：为提高 Flash 运行时系列软件的质量，提升查找注入的效率，文章在之前注入查找技术的基础上，重新设计了一个自动化查找注入的工具。该工具的前端查找程序整合了路径查找、运行进程和查找算法策略等模块，可以跨平台运行。同时，该前端查找程序可灵活的运用优化策略进行查找，具有良好的适应性和扩展性。该工具的后端分析程序可根据实际测试环境情况来计算，从而选择最优算法作为查找策略，并且可对导致注入的源头信息进行分析，进一步缩小查找范围，结合选取的算法提出更为优化的查找策略。该后端分析程序还包括了校验注入和计算 Build 的置信水平 用于整个测试的反馈。文章综合了各类查找算法和 Flash 运行时系列软件的工作特点，提升了注入查找能力，在实践中具有较强的应用价值。

关键词：注入查找；二分查找；安全机制；置信水平

中图分类号：TP393.08 文献标识码：A 文章编号：1671-1122（2012）10-0077-06

The Design of Analysis and Search Tool of Flash Runtime Injection

LU Yang[1], YE Chao[2], YAN Han-bing[3], WEN Wei-Ping[1]

(1. *Department of Information Security, SSM, Peking University, Beijing* 102600*, China;*
2. *Fire brigade of Dongcheng District, Beijing* 100061*, China;*
3. *National Computer network Emergency Response technical Team / Coordination Center, Beijing* 100029*, China*)

Abstract: In order to improve quality of Flash Runtime series software and the efficiency of injection finding, this article has designed a new tool for the work of injection finding on the basis of former injection finding technology. The front finding program of the tool integrates path search, launching process and finding algorithms strategy module. It can run a cross-platform. At the same time, the front end finding program, which has a good adaptability and expansibility, can adopt the strategy flexibly to find the injection. According to actual test environment, the after end analysis program of tools calculates to select the optimal algorithm as search strategy. It can explore and analyse the source of the injection of information, further narrow search range, combined with the selection of the optimization of the algorithm is put forward more search strategy. The after end analysis program includes the injection and calculation of calibration build confidence level for the entire test of feedback. This article comprehensively various search algorithms and Flash runtime series software characteristics of the work, and enhance the search into the ability, in the practice has strong application value.

Key words: injection finding; binary search; security mechanism; confidence level

1 研究现状

由于互联网的飞速发展和用户的需要，Flash 运行时下面的 Flash Player 和 AIR 的 Build 引入了很多新特性的代码。同时，不断地修复 Bug 的过程也会造成代码的频繁改动，这一改动对 Flash 运行时产品的稳定性、健壮性很有可能造成负面的影响。我们把这种由于新特性的加入和 Bug 修复而改动代码，造成了以往工作正常的特性失效叫做注入（Injection）。这种所谓的注入有别于信息安全的概念中所表述的注入。这种注入的现象是：在一个连续的 Flash player 版本的序列中，由某个版本的 Flash player 开始，某个特性失效，而之前的版本的这个特性工作正常，那么将这个开始失效的版本叫做注入版本（Injection build）或者注入点（Injection point）。如果失效的版本表示为“1”，而未失效的版本表示为“0”，那么这个按时间从较早到现在的排序为“00…0011…11”。

Adobe 内部把测试 Flash player 或 AIR 的子版本统称为 Build。下文所描述的 Build 就是指子版本的 Flash player 或 AIR 的安装文件或者安装包。

收稿时间：2012-07-17

基金项目：国家自然科学基金资助项目 [61170282]

作者简介：卢阳（1986-），男，安徽，硕士研究生，主要研究方向：系统与网络安全；叶超（1982-），女，北京，硕士，主要研究方向：高层居民住宅小区消防安全管理思考和建设；严寒冰（1975-），江西，男，高级工程师，博士，主要研究方向：计算机网络安全等；文伟平（1976-），男，湖南，副教授，博士，主要研究方向：网络攻击与防范等。

为了提升测试的效率，本文围绕其核心框架对问题逐步深化分析，解决注入查找方式从人工到自动、从自动到智能的问题。

本文首先调研了当前主要软件保护技术，并对当前软件保护技术的不足进行了分析。其次，介绍了注入在 Adobe Flash 运行时总体测试的重要性及其作用。并陈述了现今的大规模的 Build 编译与效率低下的手工查找的矛盾性。再次，介绍了在原有研究的基础上新的自动查找注入工具的总体设计，并说明了前端查找程序和后端分析程序运行的不同环境。说明了前端查找程序的设计，包括 Build 路径查找模块、运行进程模块、算法实现模块的设计并说明了查找流程。在前端基本查找实现的情况下，通过对优化算法与实际场景相结合，得出采用哪种算法是最高效的。最后，给出了该思路的具体应用场景，以供更多的同行做进一步的研究和参考。

2 Flash 运行时注入查找相关介绍

2.1 Flash运行时测试总体框架

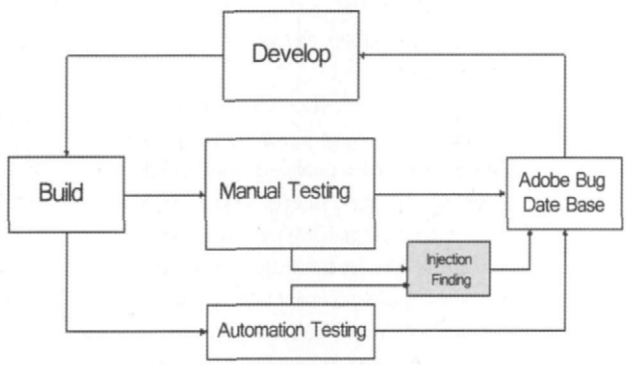

图1 Adobe Flash 运行时主体测试框架图

由测试团队来检验其质量。经测试团队通过手工的和自动的方法测试，将开发团队编译出的新 Build 中的 Bug（如果是注入类的 Bug 还需要找出注入 Build 的版本号）填写到 Adobe Bug 数据库中。最后开发团队从 Adobe Bug 数据库中获取 Bug 信息而修复 Bug 并编译出修复完的 Build。如此的过程循环往复，滚动式地推进 Flash 运行。

2.2 Adobe之前研究介绍

Adobe 之前有一个 Java 编写的注入查找工具，也可以跨平台运行。该工具简单地模拟人工查找注入的流程，采用单一的二分查找 [1] 算法来查找。

优点：实现了自动化的注入查找，该工具的运行界面较为简洁，输入需要查找的注入的范围即可进行查找。在环境稳定的情况下，该工具的查找效率较高。

缺点：只能从输入的路径文件夹下查找，不能在多个文件夹下查找，所以无法适应如今的多个文件夹、文件命名灵活的查找条件。该工具只采用了一种查找算法，并且在某些情况下这种查找算法效率会很低。因而此工具无法满足现在的 Flash 运行时软件的大规模、多版本的测试需求。

3 一种 Flash 运行时注入查找工具的设计

3.1 总体设计

3.1.1 运行环境

Flash 运行时注入查找工具分为前端查找程序和后端分析程序。前端查找程序运行在测试机上，后端分析查找程序一般运行在测试人员的工作机上（一般为 Windows 平台）。

测试机的平台主要有 Windows 系列、Mac 系列、ubuntu，也包括 Opensuse、Redhat、Solaris，需要在这些平台上进行 Flash Player 和 AIR 的注入自动查找。针对基于 Android 平台的手机和 PDA、IOS 平台上的注入自动查找程序还有待开发。

3.1.2 框架结构

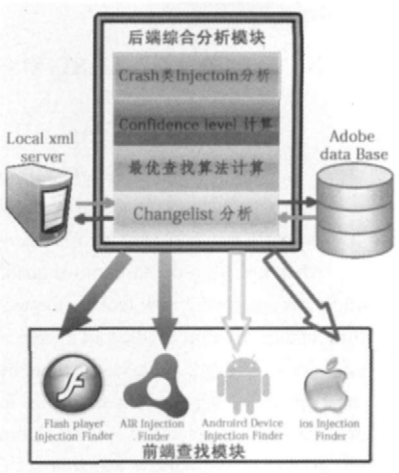

图2 系统的结构图

如图 2 所示，在系统总体结构中包含了两个大模块，前端查找模块和后端综合分析模块。前端查找模块用于在测试平台上运行找出注入 Build，而后端综合分析模块用来计算和选取最优查找算法和通过分析 Changelist 来提供注入查找信息。

前端查找模块包括：

1）Flash Player 注入查找模块：该模块用来在测试机上运行查找在该种配置下的注入 Build；

2）AIR 注入查找模块：用来查找在测试机上运行的程序，查找该配置下的注入 Build。

后端综合分析模块包括：

1）Crash 类注入分析模块：用来对 Windbg 分析 Crash log 后得到的调用栈文本过滤 [2]，以便查看信息；

2）置信水平计算模块：用来完成一项测试任务后，输入配置和 bug 情况，计算出在该种配置条件下 build 的置信水平；

3）最优查找算法计算模块：依据环境情况计算出最优的查找算法；

4）Changelist 分析模块：分析 Changelist 纯文本，在按照

特性归类的过程中需要查询 Adobe bug database，最后的结果存储于本地Windows Share Server上，其作用为辅助查找算法。

3.1.3 查找流程

Flash 运行时注入查找工具在使用时，分成三个阶段，如图 3 ：查找前策略计算阶段、查找过程及策略使用阶段、查找后校验及信息反馈阶段。

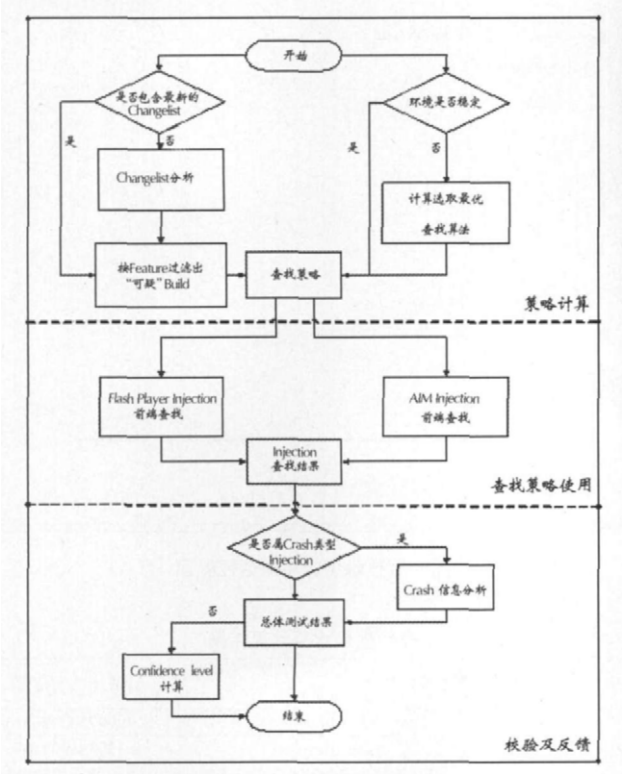

图3 总体框架查找流程

3.2 Flash Player及AIR 注入查找模块的设计

Flash Player 注入查找模块的方法是模拟人工的查找方法一步一步进行的，如图 4。这个模块采用基于 Flex[3] 框架的 AS 3.0 语言编写，采用 Cairngorn 框架做一些消息映射。其开发环境为 Adobe 的 Flash Builder 4.5。

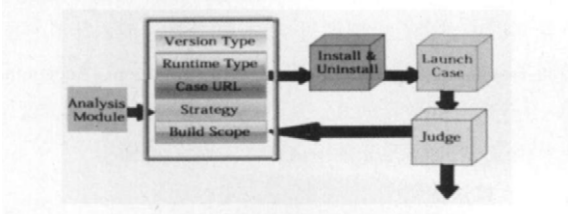

图4 Flash Player & AIR 注入查找工具的总体运行流程图

第一步，先是得到了一个由后端分析模块提供的 xml 文件，包含策略信息。它包含两个内容 ：一是由后端的策略计算出来的根据实际情况采用什么算法最优的信息 ；二是得到了以这个产生注入的 case 的特性为过滤条件过滤出的 Changelist 的结果。这些过滤出来的 Changelist 的结果是与该 case 的相同的特性代码改动的信息，所以在掌握了这个有效信息后可以将其中记录的 build 定义为“ 可疑的 ” Build。

表1 有关特性 A的Changelist项

Feature :A		A		A		A			A						A
Sub Version	1	2	…	50	51	52	…	…	100	101	…	…	150	…	200

以上的信息都需要事先加载到内存中，以便在每次循环时调用。

第二步，需要输入测试该注入所需的一些信息，注入 Finder 运行步骤如表 2 所示。

表2 Flash Player 注入 Finder 运行步骤

Version Type	String	选择发布版本还是调试版本即 :Release or Debugger。
Runtime Type	String	运行模式。Flash Player有:ActiveX, Plugin, Standalone 和 AuthPlay，而 AIR 本身就是一种运行模式。
Case URL	String	测试案例的链接。
Build Scope	Int[]	测试人员选取的查找注入的范围。

第三步，在得到了以上数据后，查找的工作可以正式开始。以表 3 的配置为例。

表3 Flash Player 注入 Finder 配置信息举例

输入信息项 ：	举例 ：
Version Type	发布版
Runtime Type	Firefox 浏览器的 plugin 插件模式
Case URL	URl
Strategy	二分法
Build Scope	X.0.1----X.0.200

首先，需要调用卸载程序将本机上的 Flash Player 卸载。其次，依据算法的不同和输入的 build 范围来选择需要安装某个版本的 Build，例如按照二分算法，应该选用 X .0.100 这个 Build 来安装。

若为 AIR 注入查找则需要跳过卸载的步骤，因为 AIR 程序是直接运行起来不需要装卸。

第四步，在安装了新的 Build 后，用指定的浏览器 Firefox 来打开该 case 的链接运行该条 case。同时判断通过与否并将结果以 <string :安装 Build 版本号, bool :case 通过与否 > 的形式发给下一个子模块，例如此时的结果为 <X.0.100,true>。

在判断 case 通过与否的过程是对 Build 的功能性测试，这个需要采用自动化测试工具或者在该 case 较为复杂的情况下需要人工的去判断通过与否，并将结果输入给下一个模块。

第五步，此模块为判断上一步给过来的结果是不是注入 Build。判断的方法是 ：如果模块中没有存储 Build 的信息，则将传过来的信息记录下来，此时记录了<X.0.100,false>。如果是原来已经记录了一条信息，那么将其与当前得到的这条信息比较，型如 <X.0.50,true> 和 <X.0.51,false> 的两条记录，则可断定 X .0.51 即是我们需要找出的注入 Build ；若前后两条记录分别为 <X.0.100,false> 和 <X.0.150,false>，那么可以断定注入 Build 应在其前方，所以将原来的区间的截止位置换位为 X.0.100，然后继续用此方法查找。图 5 为相关查找的示意图。

3.3 后端综合分析各模块的设计

3.3.1 置信水平计算

置信水平(Confidence Level) 是软件测试人员对在测试期间的产品的综合性评估，一般以 1-10 的分数来计分，从而通

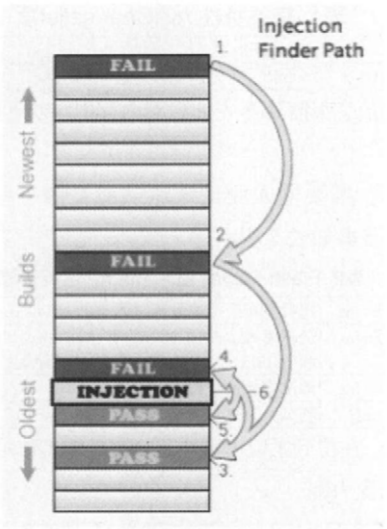

图5 Flash Player 注入查找运行示意图

过汇总统计来体现某个版本的软件产品在一个测试周期中的稳定性、功能性的逐步完善过程。并且这个统计的分数必须超过某个标准线，产品才可以发布出去，否则需要进一步完善，直到符合标准。

以往的置信水平是由人员参照某个标准文档主观的去给出分数，若某项分数由有经验的测试人员来给出，那么可以认为是相对准确。但不可能每个人都是如此的资深和有经验，所以需要有一套相对准确的计算工具来给出一个相对确切的符合实际的分数。

置信水平的值分布符合 Logistic 函数的形状，只要对 Logistic 函数的参数加以调整[4]，可得到较为准确的置信水平。

3.3.2 置信水平的计算方法及计算工具实现

首先，计算 Logistic 函数[5, 6]的各参数的值。

参数 L、k、a 都是常量，经多次调整后，将点 (0,10)、(3,8.5)、(6,6.5) 带入。求得 L=12.01721，a=0.239488，k=0.201724。随后与点 (9,4.5) 比较，得出在该情况下，当 x=9 时，y=4.384019，|y-4.5|<0.25，可以认为这些参数有效。可以计算出其拐点为 (6.68091,6.011176)，这个拐点很重要，表示在 x=6.68091 之前的是递减的，递减的速度越来越快，而在 x=6.68091 也是递减的，递减的速度越来越慢。

其次，对型如 Non-uLLu Manual \tAS2+AS3 \t Certification \t Win\t Win 7 \t IE 9 64bit \tActiveX \tRelease 64bit \t1"的文本进行分析，提取“ Certification ”、“ Win ”、“ Win 7 ”、“ IE 9 64bit ”等关键词，然后在程序中采用 switch 来逐步获取其权值。

接着，对输入的 Bug number 来处理，将其转换为整形。此时采用函数 y=，t 取 1.75，来对 Bug number 处理。因为 Bug number 不能简单的累加，同时又得保证当 x → + ∞时，y → + ∞，x 增长，y 也增长，且该函数一阶导数 >0，二阶导数 <0，刚好符合 Bug number 的处理要求。

随后，在得到了以上的值，调用 .Net 画出其在曲线上的点。

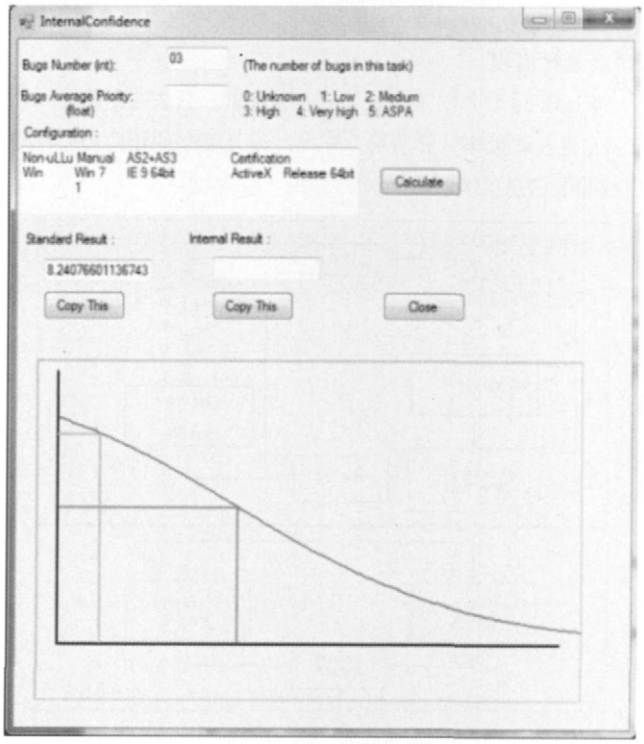

图6 置信水平计算对话框界面

最后，检验多组值。

表4 置信水平计算结果

序号	Bug 数量	配置(仅标出test level ,其它与图中配置相同)	计算出的置信值
1	0	Certification	8.99997503586514
2	1	Certification	8.330341901387
3	1	Smoke	7.94092351752163
4	1	Acceptance	8.17891076641814
5	3	Certification	7.51624280675981
6	10	Certification	6.10605339435213
7	20	Certification	5.12085681824741

如表 4 中数据所示，Bug 的数量越多，计算出的值越小，这一点与实际相符。第一条 0 bug 计算出一个 9.0 的值，说明产品在测试期最高分就是 9.0，只有在产品即将发布出去的最后一轮测试中才会有出现超过 9.0 分的分数，而对于其它配置相同而Test level变化的三组值说明了Certification、Acceptence、Smoke 中产生相同数量的 Bug 对 Build 的影响程度不同，这与项目要求相符，其计算出来的值也在误差允许范围内。

3.3.2 查找算法优化

针对 Flash Player 或 AIR 的注入 Build 的查找可以将其抽象为针对型如“ 00…0011…11 ”的数组查找变化点的问题。前面的“ 00…00 ”表示连续版本的 Flash Player 或 AIR 的某个特性工作正常，后面的“ 11…11 ”表示连续版本的 Flash Player 或 AIR 的这个特性工作不正常，我们所要找的就是这个由“ 0 ”变化到“ 1 ”的这个“ 1 ”在数组中的位置。

对此类型，这里提出了三种算法如表 5 ：顺序（逆序）查找法，二分查找法，N 分查找法。

表5 查找算法分类及说明

算法名称	算法说明
顺序（逆序）查找法	顺序（逆序）地从起始端（结尾端）开始，依次向后查找，直到找到第一个“1”，返回该“1”的位置。
二分查找法	在已知首尾两个分别为“0”和“1”的情况下，选取数组中正中部的那个元素，判断其为“0”或“1”。如果为“0”则选取后半段，反之选取前半段，再对选取的这半段用同样的方法查找。如此循环并最终得出由“0”变化到“1”的那个“1”的位置。
m分查找算法（m>2）	与二分本质相同，其同样选取首位分别为“0”和“1”的区间，然后将其等分为N个区间，并判断这N-1个分界点的值，并选取这N个区间中首尾分别为“0”、“1”的区间继续如此操作，直到找出由“0”变化到“1”的那个“1”的位置。

假设在某个型如“00…0011…11”的数组一共有n个元素，并且这个由“0”变化到“1”的这个位置在该数组中的位置是等概率的，那么则有以下的算法复杂度比较：

表6 查找算法的分类及复杂度说明

算法名称	算法的复杂度	平均复杂度
顺序（逆序）查找算法	0(n)	0()
二分查找算法	0()	0(-1+)
m分查找算法（m>2）	0(m*)	0(m*)

从表6中可以比较出，二分查找算法是最优的，其次是m分查找算法，最差的是顺序（逆序）查找算法。

以上是理想的情况下，只考虑了将检验该位是否为“0”或“1”定义为一个单位复杂度。而还原到实际问题中该判断的过程对应于Flash Player注入Finder的判断测试case在安装了现Build下的某个特性是否工作正常。在判断这个case通过与否之前还有一步操作需要下载和安装新的Build。下载、安装Build与检验case通过与否这三个操作每次都要按顺序执行一次，可以视为一个绑定的操作，这个绑定的操作对于手动测试来说，话费时间是固定的。与下载Build操作不同，可以一次性下载一个Build，也可以一次性下载m-1（m>2）个Build，这两种操作在手动测试中花费测试人员的时间差不多，并且这个操作花费时间与以上所提到的绑定的三步操作是同一个量级。所以m分查找就有可能超越二分查找成为最优。

在Adobe Flash测试团队的日常工作中，针对注入查找的手动查找方式，测试人员平均时间开销如表7所示。

表7 注入手动查找步骤及说明

绑定的操作序号	操作内容	时间花费
操作一	1、卸载build 2、安装build 3、检验case通过与否	=20s
操作二	打开浏览器、按各级目录找到所要下载的build，下载一个或m-1个build，关闭浏览器	=50s

对于二分查找其需要的操作组合是集合：{操作二，操作一，操作二，…，操作一，操作二，操作一}，如此操作一和操作二交替进行。

对于m分查找其需要的操作组合是：{操作二，操作一，操作一，…，操作一，操作二，操作一，操作一，…，操作一}，如此的一个操作二后面跟着1到m-1个操作一。

假设在需要查找的Build区间上共有n个连续版本的Build，那么对于二分查找总的时间开销为：

*(+) ……(1)

对于m分查找，在得到了m-1个分界点的情况下如果采用顺序遍历，在平均情况下的总的时间开销为：

* + ** ……(2)

在m分查找所得到的m-1个分界点后，如果采用二分查找，在平均情况下为：

* + **(-1+) ……(3)

为方便计算，将式（3）的结果近似为：

* + **() ……(4)

此时比较式（1）与式（2）的大小。设n=128或256，这个介于100-300的n的取值是非常符合实际测试场景的，因为在所要查找的build区间中就是包含了这么多个build。

用式（1）-式（2）然后与0比较大小：

*(+) - * - ** ……(5)

式（5）的意义在于求最大值，带入各参数的值，并在最大值时，给出m的值。

n=128，=20，=50

表8 m分查找相对于二分查找的时间节省量

m=	式（5）的值
2	0
3	136.679
4	175
5	188.526
6	192.123
7	190.786
8	186.667
9	180.844
10	173.919
11	166.247
12	158.058
13	149.5
14	140.676
15	131.659
16	122.5
17	113.238
18	103.902
19	94.513
20	85.0881
29	0.0845387
50	-192.158
100	-616.285

可见当m=6的时候式（5）的正值最大，也就是说在此种情况下六分查找比二分查找节省了192.123秒的时间。当然这是在n=128的情况下，如果n较大，这个值也会变大，那么就会节省跟多的时间。整个二分查找所需要的时间是 *(+)=490s，所以节省了192.123/490=39.2%。在build区间内版本较多的情况下，查找注入通常会消耗一个测试人员2-4个小时的时间，所以采用该种m分查找会节省0.8-1.6个小时的时间，所以其作用是相当可观的。

可以看到29分查找与二分查找的时间开销等同，当然在实际操作中不可能采用29分查找，因为其将过多的时间用在了下载build的过程中。超过29分查找的就不如二分查找的效率高。

3.3.3 Changelist 分析

Changelist 是用来体现对源代码逻辑改动的一系列变化的记录。针对源代码的任何改动，将由源代码管理软件自动地保存在 Changelist 中。代码的改动主要是由于新的特性加入和对 Bug 的代码修复，一旦有代码的变动，项目代码管理软件会记录这些改动。其中信息包含了：Change ID、日期、详细描述、测试人员备注、修复的 Bug ID 等关键信息。由于测试人员是通过邮件得到的这个信息，所以这个信息源头是个纯文本，需要从这个纯文本中提取出关键项，同时还需要按照数据库或人工判断加上一个元素——该 Changelist 的所属特性。

Changelist 分析实现如图 7 所示。

第一步，获取 Changelist 信息。从源代码管理服务器上得到 Changelist 纯文本，这个纯文本是以邮件内容的格式存储。由于其包含内容较多，该纯文本的体积也较大。通常每 600 份邮件组成的一个 Changelist 纯文本大约有 5M。

第二步，通过过滤工具规整 Changelist 中有效信息。首先，在得到了 Changelist 纯文本文件后，由于文本较大，后续的操作较多，可以先将该纯文本分块，分块的过程不能将连接处的邮件拆开。然后，将得到的块拆分为单个邮件，存在 string[] 中。其次，采用正则表达式提取关键项的内容：Change ID、日期、详细描述、测试人员备注、修复的 Bug ID。最后，将内容存放在 xml 文件中，其数据格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<changelist name="changelist" tag="2Windows">
  <Item>
    <ChangeID>100871</ChangeID>
    <date>2011/10/26</date>
    <bugfixed>3005555</bugfixed>
    <detaileddescription></detailddescription>
    <QAtestingNotes></QAtestingNotes>
  </Item>
</changelist>
```

第三步，为每条 Changelist 的项添加一个属性 <feature> 并获得这个属性的内容。首先判断该 Changelist 项是不是为修复 bug 而产生的，如果是则查询该 bug 的特性赋给这个 Changelist 项作为特性的内容。其次，如果不是由 Bug 而产生的，

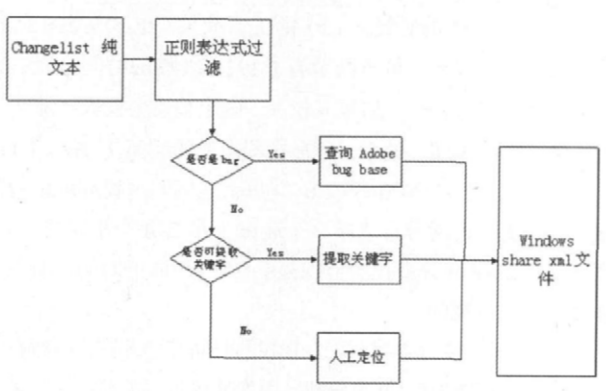

图7 Changelist分析流程图

可对 <detaileddescription> 和 <QAtestingNotes> 进行关键词匹配，如果匹配成功则将该特性赋给该 Changelist 的特性。最后，在以上两种方法都无法获得特性的时候，采用人工读取内容并归类特性。

```
<?xml version="1.0" encoding="utf-8"?>
<changelist name="changelist" tag="2Windows">
  <Item>
    <ChangeID>100871</ChangeID>
    <date>2011/10/26</date>
    <bugfixed>3005555</bugfixed>
    <detaileddescription></detailddescription>
<QAtestingNotes></QAtestingNotes>
<feature>Network</feature>
  </Item>
</changelist>
```

存储到 Windows Share 服务器上的指定文件夹中。

4 结束语

注入查找技术是一个综合性的技术，融合了计算机算法，文本分析技术，概率论与数理统计等相关数学理论。注入查找效率的提升对任何软件的测试流程都具有积极的意义。尤其是针对版本较多、用户群较大、发布频繁的 Adobe Flash 运行时，可以更为明显的减少人工劳动，提高测试效率。因此，本文致力于设计一个完善的针对 Flash 运行时的注入查找工具，该工具可实现基本的自动化查找，同时具有良好的扩展性和稳定性，可适应算法的扩展和测试环境及 Build 测试方式的变化。

综上所述，这种 Flash 注入查找工具有较多优点 如效率高、适应多平台、扩展性良好、维护较为简单、易于操作人员使用等。该工具在 Adobe（中国）研发中心使用，得到较好的效果。我们认为，可以将其设计成适应体积较小而跨平台的软件的注入查找工具，来提高此类软件提供商的测试团队的工作效率。

由于互联网和操作系统的发展，使得跨平台的基于 Web 的应用获得了较大的发展。Flash 运行时系列软件的注入产生的缘由就是功能性和安全性代码的加入和修复，如果能缩短响应用户的问题的时间，对公司的名誉和竞争力会有所提升。我们的目标是将 Flash 运行时的注入查找工具发展下去，提出更好的查找优化方案和扩展到 Android 和 IOS 平台上，以便提高 Flash 运行时系列软件的效率，并使用户在享受互联网娱乐的同时较少遭受安全问题的困扰。 （责编 杨晨）

参考文献：
[1] Trevor McCauley. Injection Finder[M]. Adobe Wiki, 2009.
[2] Thomas Junk. Confidence level compulation for combing searches with small statistics[M]. 1999.
[3] Adobe Systems Incorporated. ACTIONSCRIPT™ 3.0 编程 [M]. 2007.
[4] Jon Louis Bentley. Multidimensional binary search trees used for associative searching[M]. 1975.
[5] Paul David Allison. Logistic Regression Using SAS[M]. 1999.
[6] 李根发，钟笛 . 数字签密综述 [J]. 信息网络安全，2011，(12)：1-7.