基于动态污点分析的二进制程序导向性模糊测试方法

张 斌，李孟君，吴 波，唐朝京
（国防科学技术大学 电子科学与工程学院，湖南 长沙 410073）

摘 要：传统模糊测试中，由于不同的输入可能重复测试相同的状态空间，导致其效率严重低下。提出一种基于动态污点分析与输入分域技术相结合的二进制程序导向性模糊测试技术，可以对典型安全敏感操作与一般模块函数进行导向性模糊测试，很好地解决了传统模糊测试效率低下的问题。实现了二进制导向性模糊测试的原型系统TaintedFuzz，实验证明，该系统能够对二进制程序中存在的典型安全漏洞进行高效地发掘。

关键词：安全漏洞；导向性模糊测试；动态污点分析；输入分域
中图分类号：TN915.08-34 **文献标识码**：A **文章编号**：1004-373X(2014)19-0089-06

Method of binary oriented fuzzy testing based on dynamic taint analysis

ZHANG Bin, LI Meng-jun, WU bo, TANG Chao-jing
(College of Electronic Science and Engineering, National University of Defence Technology, Changsha 410073, China)

Abstract: Since traditional fuzzy testing may test the same state space repeatedly due to the different input, and lead to a low efficiency, a binary oriented fuzzy testing technique based on dynamic taint analysis combined with input field classification technology is presented in this paper, which can perform the oriented fuzzy testing for typical security-sensitive operation and general module function, and serve as a good solution to the problem of low efficiency of the traditional fuzzy testing. The prototype system TaintedFuzz was also realized for binary oriented fuzzy testing. The experiment proves that the method is capable of exploring the typical security vulnerabilities in the binary program efficiently.

Keywords: security vulnerability; oriented fuzzy testing; dynamic taint analysis; input field classification

0 引 言

随着现代信息系统规模和复杂度的逐步增加，计算机软件安全问题变得尤为突出，而软件安全漏洞又是威胁信息系统安全的核心问题，因此，如何快速准确地发掘软件安全漏洞成为安全界的热点研究问题。根据研究对象的不同，软件安全漏洞挖掘分为针对开源软件的源码级别漏洞挖掘和针对闭源软件的二进制级别漏洞挖掘。出于对自身商业利益和知识产权的保护，大部分软件厂商并不向外开放其软件源代码，因此针对二进制程序进行安全漏洞发掘是当前研究的一大主流方向。目前，针对二进制程序有效的发掘方法有补丁对比技术、模糊测试技术、静态分析技术和动态分析技术[1]。

模糊测试(Fuzzing)技术是一种通过向目标系统提供非预期的输入并监视异常结果来发现软件漏洞的方法[2]。模糊测试首先通过正常样本构造出一系列畸形测试用例，然后针对目标系统使用这些测试用例逐一进行测试并实时监控目标状态，通过对目标异常信息的分析来发掘目标系统中存在的安全漏洞。模糊测试是一种通过输入空间的完全遍历来驱动程序在状态空间进行完全遍历的测试方法，具有测试简单、易于部署等特点。然而，输入空间与状态空间的非对等性导致不同的输入可能驱动程序重复测试相同的状态空间，因此模糊测试具有效率低下的天然缺陷。

传统模糊测试采用完全黑盒的测试方法，在畸形测试用例的生成过程中，没有用到程序的内部信息[3]，造成大部分畸形测试用例对程序状态空间进行了重复性测试。如何提高模糊测试的效率是当前模糊测试的主要研究方向，本文提出并实现了针对二进制程序的导向性模糊测试原型系统TaintedFuzz。该系统采用基于动态污点分析与输入分域技术相结合的导向性模糊测试方法，利用程序在动态执行过程中的运行时信息指导模糊测试畸形样本的生成，很好地解决了传统模糊测试效率低下的问题。

1 导向性模糊测试技术

畸形样本生成是模糊测试技术的关键。正常样本数据由安全相关数据和安全无关数据组成，安全相关数据是指与程序安全敏感操作相关的数据，安全无关数据

收稿日期：2014-03-04

指对安全敏感操作不产生影响的数据[3]。典型安全敏感操作包括字符串处理函数(如strcpy、memcpy)、内存分配函数(如malloc、HeapAlloc)以及用户根据具体需求指定的安全操作。对安全无关数据的变异通常不会导致软件异常,而对安全相关数据的变异往往能够触发软件中潜在的安全漏洞,因此,首先通过静态分析技术获得安全敏感函数的调用点;然后利用动态污点分析技术对数据依赖关系进行动态跟踪,根据安全敏感函数的参数类型分析对应的安全相关数据;最终通过对安全相关数据的变异实现对目标程序的导向性模糊测试。

考虑到编译器的优化工作,如函数内联、无用代码删除、循环体展开等,这些优化工作可能会将一些安全敏感函数内联到模块函数当中,造成模块函数中出现漏洞的可能性增大。因此,对模块函数进行模糊测试时,发掘目标程序中的安全漏洞具有重要意义。然而,不同于安全敏感函数,这些模块函数往往不具有调试信息,其函数参数类型不可获得,所以无法根据其函数类型对其相关数据进行变异。提出基于动态污点分析与输入分域技术相结合的导向性模糊测试方法,通过对目标程序处理正常样本过程的学习,获取所有处理输入的模块函数及对应的污点输入;并通过基于执行迹的分域技术,完成对未知格式输入的输入域的划分;最终根据输入域划分的结果对模块函数的污点输入进行变异,实现对目标程序的导向性模糊测试。

本文实现了基于动态污点分析与输入分域技术的导向性模糊测试原型系统TaintedFuzz,整体框架如图1所示。系统分为前端信息收集、中端测试点分析及后端Fuzzing引擎三大模块。系统前端模块利用动态污点分析技术收集目标软件在执行正常测试样本时的运行时信息;中端模块通过对前端收集的预处理信息进行分析处理,获得后端Fuzzing引擎的测试点(check点);后端Fuzzing引擎通过对check点的变异测试,完成对目标软件的导向性模糊测试。

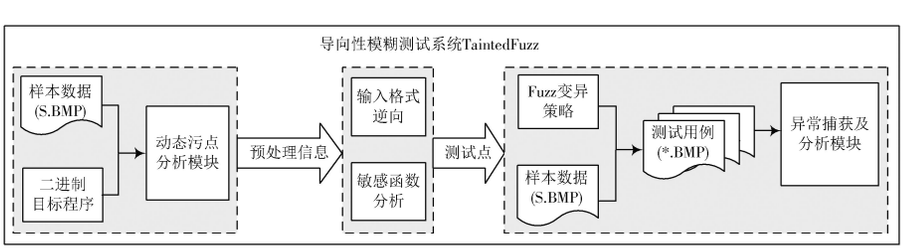

图1 导向性模糊测试系统整体框架

1.1 在线安全敏感操作分析

1.1.1 细颗粒度动态污点分析技术

TaintedFuzz采用细颗粒度动态污点分析技术(Fine-Grained Dynamic Taint Analysis)完成对目标程序执行过程的监控及信息获取。

动态污点分析技术本质上是一种信息流分析,其基本思想是在程序的动态执行过程中,追踪并收集程序对特定输入数据的处理过程。传统动态污点分析技术仅能够分析内存、寄存器或指令是否被单一污点源污染,无法满足针对复杂输入软件的分析。TaintedFuzz通过对污点数据进行字节粒度的标定与传播,完成对目标程序的细颗粒度污点分析。

TaintedFuzz对目标程序的动态监控及信息收集主要由三个部分构成:污点数据的标定、污点数据的传播以及污点分析策略。图2描述了其基本过程。

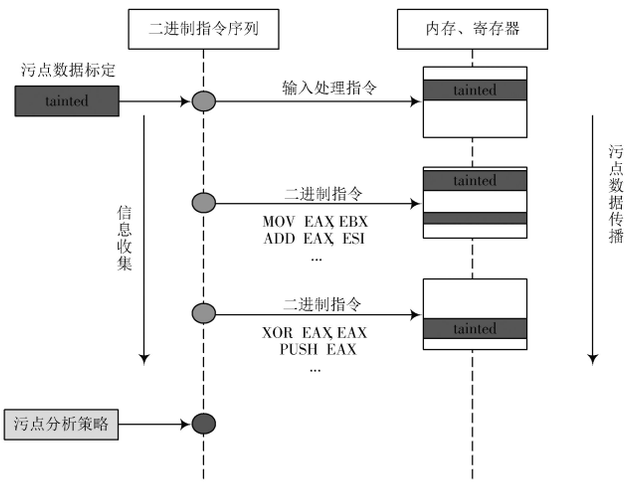

图2 动态污点分析过程

TaintedFuzz依靠劫持特定的Windows API函数完成对目标程序污点数据的标定及注入。如针对命令行输入型程序可以拦截GetCommandLine的返回值,该函数返回一个指向命令行参数的指针,通过对该指针指向的内存区域进行污点标记,完成对命令行参数的污点标记;针对文件输入型程序,TaintedFuzz通过拦截Windows系统中典型的文件操作函数(如fread、ReadFile、MapViewOfFile等)并分析这些函数的参数或返回值,完成对文件输入型程序的污点标定。TaintedFuzz通过对输入数据的每个字节赋予一个特定的污点标签,来完成字节粒度上多污点源的标定。污点标签结构为二元组<s,o>。其中:s表示污染源,如文件、命令行等;o表示该污点字节在整个污染源中的偏移量。在完成污点数据的标定之后,TaintedFuzz监控并分析每条CPU指令,来实现对目标代码执行的动态监控,并通过影子内存(Shadow Memory)[4]技术实现对内存和寄存器污点信息的存储及传播。在程序的执行过程中,TaintedFuzz将

根据需要对相应的信息进行收集,并在达到需要分析的位置(sink点)之后,根据特定的污点分析策略对收集到的信息进行分析。

1.1.2　安全敏感函数分析

TaintedFuzz在细粒度动态污点分析基础上实现了数据依赖关系的动态跟踪,因此在监控程序动态执行的同时,可以在线完成典型安全敏感操作的分析工作。TaintedFuzz首先通过静态分析技术获得安全敏感函数的调用点,然后在动态污点分析时,根据这些安全敏感函数的参数类型分析对应的安全相关数据,如对于strcpy,其函数原型为:

```
char*strcpy(char*strDestination,const char *strSource)
```

其中strDestination表示写入地址,strSource表示源地址。TaintedFuzz通过对strSource指向的内存地址进行污点检查,获取对应的安全相关数据,并记录当前运行时信息,运行时信息记录格式为(Danger_func,TID,Module,InvokeEIP,RETAddr,StackBase,StackLimit,arg1,arg2,…),Danger_func代表安全敏感函数;TID、Module分别表示当前线程ID与模块名称;InvokeEIP与RETAddr分别表示安全敏感函数调用地址与返回地址;StackBase与StackLimit分别表示当前线程栈的栈基址与栈界限;arg1等代表当前安全敏感函数参数。下面给出了具体的执行结果:

```
(1, 1856, cmd_overflow.exe, 0x0040e791, 0x0040e796, 0x0012e000, 0x00130000, 0x0012ff10, 0x00380bbd)
T(16 17 18 19 20 21 22 23)
```

其中1表示当前安全敏感函数为strcpy;T(16 17 18 19 20 21 22 23)表示strSource参数所指向的内存被偏移量为(16 17 18 19 20 21 22 23)的污点数据所污染。那么这8个连续污点数据将被作为检测点,用来指导后端的模糊测试。这样可以避免对输入空间的盲目枚举,提高模糊测试的效率。

1.2　针对模块函数的测试用例生成方法

模糊测试的有效性完全取决于测试用例的有效性。例如,针对安全敏感操作的测试用例生成,可以在已知敏感操作函数参数类型的情况下,根据参数类型对安全相关数据进行变异,这样生成的测试用例更容易触发安全漏洞。然而,如何针对未知参数类型的模块函数进行有效的测试用例生成,却成为模糊测试的难题。在对输入进行输入域划分的基础上,结合程序循环复杂度提出一种针对模块函数的测试用例生成方法,该方法可以针对复杂模块函数生成有效的测试用例。

1.2.1　基于执行迹的输入分域技术

作为输入驱动的测试方法,模糊测试的有效性完全依赖于畸形数据的生成方式。传统模糊测试畸形测试用例产生的方法有两种:基于变异的畸形数据生成方法和基于规范的畸形数据生成方法[2]。基于变异的生成方法通过对正常数据进行随机修改/破坏而生成畸形数据,其主要优势在于不依赖具体的格式而且易于实现,然而这种测试方法生成的测试数据对程序状态空间覆盖率比较低,严重影响模糊测试的效率;基于规范的畸形数据测试方法依赖于测试人员对文件/协议格式的先验知识,然而如何获取畸形数据生成规范却成为这种方法的瓶颈,对于未公开或非常复杂的输入格式,如何构造出有效的测试用例成为基于规范的畸形数据生成方法的难点。在文献[5-7]对协议格式逆向工程(Protocol Format Reverse Engineering)已经取得一定成果的基础上,提出一种基于执行迹的输入分域方法,通过学习目标程序对正常样本的处理过程,完成对输入数据的域划分。

算法的核心思想是程序在解析输入时总是将不同的输入域(Input Field)放置在不同的执行环境(Execution Context)中处理,也就是说,属于同一个输入域的相邻字节将在同样的执行环境中处理。因此,TaintedFuzz在已获取到每个输入字节处理信息(执行迹、调用栈等)的基础上,通过基于执行迹的输入分域算法完成对输入中各个域的区别与划分。

算法将污点输入的处理信息表示为四元组,<i,s,t,c>。其中i代表指令地址;s表示当前指令的调用栈信息;t表示当前指令的污点集合信息;c为当前进程线程信息。TaintedFuzz采用动态维护影子栈的方法完成对程序运行时调用栈的获取,对进程中每个线程影子栈进行分别维护。影子栈的维护通过插桩函数调用指令,如CALL和函数返回指令如RET完成。若将每条处理信息表示为preInfo[i],则每条处理信息中的四元信息将分别表示为preInfo[i].i、preInfo[i].s、preInfo[i].t以及preInfo[i].c。具体输入分域算法如下:

```
Algorithm2 Input Format Reverse
1:Initial:IFtree = ROOT  -  Input Format Tree
2:Input:preInfo
3:Output:IFtree
4:Input_Format_Reverse (preInfo){
5:    p=preInfo[0].t; i = 1;
6:    while(preInfo is not EOF){
7:        q=preInfo[i].t;
8:        if(q == p){}
9:        else if ((q.min == p.max || q.min ==p.max+1) &&
10:       preInfo[i].s == preInfo[i-1].s)
11:       p=UNION(p, q);
12:   else{
13:       Vnode =CreateNoe(p);
14:       Find Unode in IFtree where Unode contains Vnode, but not its children;
15:       Insert Vnode into IFtree as the child of Unode;
```

```
16:     if (Unode.children != NULL){
17:       Find NeedMoves in Unode.children which are subset of Vnode;
18:       Insert NeedMoves into IFtree as the child of Vnode;
19:     }
20:      p=q;
21:   }
22:  i++;
23:  }
24:  return IFtree;
25: }
```

算法将整个输入看作一个输入格式树。首先将树初始化为包含所有污点标签的ROOT节点。对于每条信息preInfo[i],如果当前信息污点集合与前一条信息污点集合连续或“伪连续”(伪连续是指当前信息污点集合最小值与前一条信息污点集合最大值相等),并且当前调用栈与前一条调用栈相同,则认为当前污点信息集合与前一条污点信息集合属于同一个输入域(第9、10行);如果污点集合信息不连续或调用栈不同,算法首先在树中找到一个包含当前污点集合信息的最小节点(第14行),然后将当前污点信息作为该节点的子节点插入到树中;同时,如果该节点有子节点,则将子节点中是当前污点信息子集的子节点移动为当前污点信息集合节点的子节点(第17~19行)。

用于测试的文件格式如图3所示。

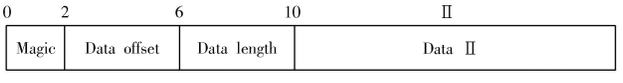

图3 测试文件格式

该测试文件格式前两个字节表示文件格式,第2~5个字节表示数据偏移位置,第6~9个字节表征数据域的长度,第10个字节以后为数据域。测试程序为简单的文件读入解析程序,经过输入域划分之后的结果如图4所示。可以看到,基于执行迹的输入分域算法可以很好地识别与划分输入域。

1.2.2 输入分域与循环复杂度结合的测试用例生成法

考虑到目标程序中模块函数的数量比较庞大,在有效时间内对目标程序中所有模块函数进行完全模糊测试将会消耗大量资源,因此,为了提高模糊测试效率,发掘更多的安全漏洞,结合程序分析理论,对模块函数进行筛选,选择其中比较复杂的函数进行选择性的分析与测试。

循环复杂度(Cyclomatic Complexity)也称为条件复杂度或圈复杂度,用于表示程序复杂度[8]。程序的循环复杂度是其线性独立路径的数量,一般通过程序的控制流图来定义,其定义为 $M=E-N+2P$,其中 E 代表流图中边的个数,N 代表流图中节点个数,P 代表流图中连接组件的个数[9]。例如,在图5给出的简单控制流图中,$E=9$,$N=8$,$P=1$,因此其循环复杂度为 $9-8+(2\times1)=3$。

循环复杂度和软件缺陷个数具有高度的正相关性,因此循环复杂度可以用来预测软件中安全缺陷的个数[10]。提出基于输入分域与循环复杂度相结合的测试用例生成方法,首先通过对动态污点分析执行迹进行分析,获得被污染的函数及其污点信息集合;然后,为了提高模糊测试中发现漏洞的概率,根据循环复杂度,对被污染函数进行排序,选取其中复杂度较高的函数;最后,根据输入分域的结果对这些函数的污点信息进行分析和输入域划分,用于指导模糊测试的测试用例生成。

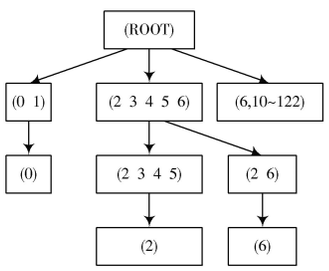

图4 输入域划分结果

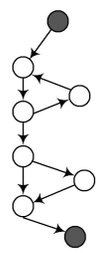

图5 程序控制流图

如表1所示,在对动态污点分析的执行迹进行分析的基础上,针对循环复杂度较高的Func1,Func2,Func3,可以根据输入分域的结果将其对应的污点信息进行输入域划分,如(2,3,4,5)四个字节很可能表示一个整形数据,(10,11,12,13,14,15)连续多字节表示字符串数据,那么用这些信息指导生成的测试用例将会更容易触发复杂函数中的安全漏洞。

表1 被污染函数复杂度排序

污点信息集合	函数	循环复杂度
(2,3,4,5,8,9)	Func1	40
(10,11,12,13,14,15)	Func2	37
(1,5,6,7)	Func3	23
…	…	…

1.3 模糊测试引擎

在完成安全相关数据偏移及类型识别之后,TaintedFuzz系统将识别的关键测试点送至模糊测试引擎中进行变异。TaintedFuzz的模糊测试引擎主要负责畸形测试用例的生成和针对目标程序的安全测试。TaintedFuzz的模糊测试引擎实现了针对不同数据类型(如longint,int,shortint,char,string等)的变异,并且每种数据类型均具有多种变异策略,如针对char类型的深度变异、针对string类型的长度变异,针对整形的临界安全值变异及随机变异等;安全测试部分采用调试器技术实现对目标程序执行的动态监控及异常捕获,并且可以将异常发生时的CPU状态及执行上下文记录下来,供安全

人员进行分析。

2　TaintedFuzz系统实现及实验结果

2.1　TaintedFuzz系统实现

TaintedFuzz原型系统基于二进制代码分析平台Bitblaze[11]的TEMU[12]动态污点分析组件实现。TEMU是一个基于QEMU虚拟化技术开发的全系统动态二进制污点分析平台,在TEMU中可以运行操作系统及应用程序,并通过相应接口完成对相关二进制代码执行的细粒度监控与分析。TEMU架构如图6所示,其中语义提取器用于提取仿真系统的操作系统级语义信息,包括进程、线程、模块及符号信息;动态污点分析引擎采用影子内存技术实现对物理内存、CPU寄存器、磁盘及网络接口缓冲区数据的污点标记及传播;同时用户还可以自定义插件,以扩展TEMU功能。TaintedFuzz通过编写命令行污点标记模块、文件污点标记模块、执行上下文记录模块、敏感函数在线分析插件,实现针对二进制程序的执行迹及运行时信息的维护与记录。

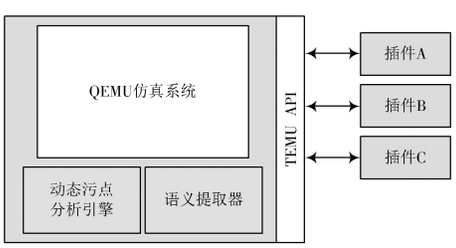

图6　TEMU架构

TaintedFuzz模糊测试引擎基于模糊测试框架antiparser实现。antiparser是一个由python开发的跨平台模糊测试框架,主要用于针对不同数据类型实现畸形测试用例的生成[2]。TaintedFuzz扩展了antiparser数据的生成方式,以提高生成数据的有效性;并在antiparser的基础上实现了针对命令行输入型程序与文件输入型程序的模糊测试引擎,可以完成对二进制程序执行的监控及异常捕获。TaintedFuzz模糊测试引擎的测试及异常捕获模块基于调试器技术实现,通过集成轻量级调试器pydbg[2]完成对异常的捕获及对异常发生时现场环境的记录。

2.2　实验结果

实验针对未知参数类型的模块函数和典型安全敏感操作的导向型模糊测试,分别选取两个不同的目标程序进行测试。针对模块函数的测试程序代码片段如下:

```
Test Code 1:
1:char * get_Data(char *arg_[], int len){
2:      char dest[10];
3:      memset(dest, 0, sizeof(dest));
4:      char * a = NULL;
5:      if (len > 0){
6:          strcpy(dest,arg_[2]);
7:          a=dest;
8:      }
9:      return a;
10:}
```

程序从命令行参数中获取数据字符串arg_[2],并将其复制到缓冲区当中;然而在复制时未对参数的长度进行合理的验证(第5行),如果命令行第二个参数长度大于10则会造成严重的缓冲区溢出。程序中虽然显式调用了安全敏感函数strcpy,但是由于编译器的优化,使strcpy被内联到当前模块中,无法通过静态分析的方式获得这些安全敏感函数的调用点。虽然现有一些可以通过函数摘要匹配的安全敏感函数识别的算法[13],但由于编译器的多样化造成内联代码的多样化,因此这些算法仍无法完全解决安全敏感函数的识别问题。TaintedFuzz通过学习目标代码对输入的解析过程,从而完成对命令行参数的输入域划分,划分之后的输入格式如图7所示(输入命令行参数为"soft_metric.exe 8 AAAAAAAA"):

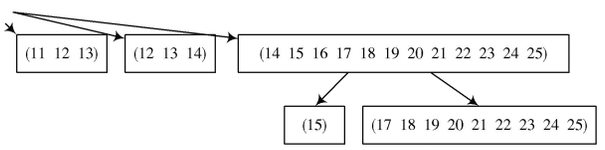

图7　输入域划分结果

同时,通过分析执行迹信息得知,get_Data函数的污点集合信息为(15,17,18,19,20,21,22,23,24)的非连续字节数据(考虑到目标程序中用户自定义函数较少,因此跳过函数循环复杂度排序),那么根据输入划分的结果,判定偏移量为15的参数"8"和偏移量为(17,18,19,20,21,22,23,24)的参数"AAAAAAAA"分别属于不同的输入域;然后通过导向型模糊测试引擎分别对偏移量为15的单字节进行深度变异和偏移量为(17,18,19,20,21,22,23,24)的连续字节进行字符串变长变异;最终模糊测试引擎成功触发该漏洞。

针对典型安全敏感操作造成的安全漏洞,实验选取VUPlayer 2.49的.m3u缓冲区溢出漏洞,该漏洞在EDB数据库的编号为EDB-ID:30336。VUPlayer是Windows平台下一款功能强大的媒体播放工具,支持多种媒体格式。该软件2.49版本在解析.m3u的文件名时存在严重的缓冲区溢出漏洞。

TaintedFuzz随机选取简单的样本.m3u文件,如图8所示,文件仅包含一条播放记录。通过对VUplayer解析样本.m3u文件的过程进行动态污点分析及在线安全敏感函数分析后,获得安全敏感函数调用记录如下:

```
CALL lstrcpyA() at 0x00453313: (Dest: 0x0012eb4c, Scr: 0x00ba93e4)
```

PID：1700，MODULE：VUPlayer.exe，TID：1696，RETADDR：0x00453319.

TID： 1696， StackBase： 0x00129000， StackLimit：0x00130000.

Writing to current stack.

Taint(0~10)

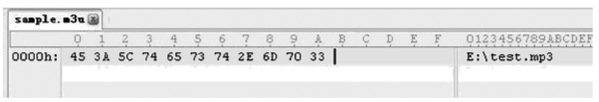

图8 样本文件

VUPlayer在0x00453313处通过调用lstrcpyA函数将当前.m3u文件的文件名(第0~10字节)拷入到当前的线程栈当中(0x0012eb4c)；在对危险函数lstrcpyA的参数进行分析后，TaintedFuzz将测试点(第0~10字节)送入到导向性模糊测试引擎当中作为字符串进行变异，模糊测试引擎成功触发该漏洞。异常发生时上下文环境如图9所示。

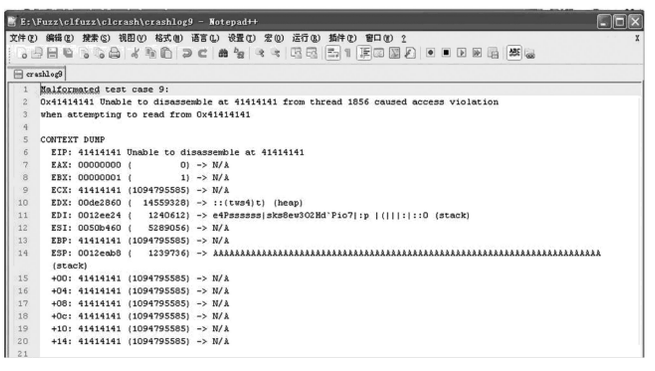

图9 异常现场环境

3 结 语

本文提出了一种基于动态污点分析的二进制程序导向性模糊测试技术。该技术可以根据动态污点分析的结果，对安全敏感相关字段进行有效地广度或深度变异。同时，针对未知参数类型函数，特别是内联了安全敏感操作的模块函数，提出了一种基于输入分域与循环复杂度的测试用例生成方法，该方法利用基于执行迹的输入分域方法。首先通过对目标程序处理正常样本过程的学习，完成输入字段的划分；之后，为了提高漏洞发现的效率，根据模块函数的循环复杂度对模块函数进行排序；选择复杂度较高的模块函数，对其根据输入分域结果进行测试用例生成。该方法可以针对复杂模块函数生成有效的测试用例，提高了模糊测试的效率。实现了基于动态污点分析的二进制模糊测试的原型系统TaintedFuzz，实验证明该系统针对二进制目标程序中的模块函数和典型安全敏感操作可以进行导向性模糊测试，能够对二进制程序中存在的典型安全漏洞进行高效地发掘。

参 考 文 献

[1] 文伟平，吴兴丽，蒋建春.软件安全漏洞挖掘的研究思路及发展趋势[J].信息网络安全，2009 (10)：78-80.

[2] SUTTON Michael.模糊测试：强制性安全漏洞发掘[M].北京：机械工业出版社，2009.

[3] 王铁磊.面向二进制程序的漏洞挖掘关键技术研究[D].北京：北京大学，2011.

[4] SCHWARTZ E J, AVGERINOS T, BRUMLEY D. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask) [C]// 2010 IEEE Symposium on Security and Privacy (SP). [S.l.]: IEEE, 2010: 317-331.

[5] LIN Z, JIANG X, XU D, et al. Automatic protocol format reverse engineering through context-aware monitored execution [J]. NDSS, 2008, 8: 1-15.

[6] LIN Z, ZHANG X, XU D. Reverse engineering input syntactic structure from program execution and its applications [J]. IEEE Transactions on Software Engineering, 2010, 36(5): 688-703.

[7] CABALLERO J, YIN H, LIANG Z, et al. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis [C]// Proceedings of the 14th ACM conference on Computer and Communications Security. [S.l.]: ACM, 2007: 317-329.

[8] KANER C, BOND W P. Software engineering metrics: What do they measure and how do we know [J]. Methodology, 2004, 8: 6-18.

[9] MCCABE T J. A complexity measure [J]. IEEE Transactions on Software Engineering, 1976 (4): 308-320.

[10] HATTON L. The role of empiricism in improving the reliability of future software [J/OL]. [2012-08-29]. http://www.computer.org/csdl/.

[11] SONG D, BRUMLEY D, YIN H, et al. BitBlaze: A new approach to computer security via binary analysis [J]. Information Systems Security: Lecture Notes in Computer Science, 2008, 5352: 1-25.

[12] YIN H, SONG D. Temu: Binary code analysis via whole-system layered annotative execution, UCB/EECS-2010-3 [R]. Berkeley: EECS Department, University of California, 2010.

作者简介：张 斌(1991—)，男，陕西乾县人，硕士研究生。主要研究方向为通信与信息安全。

李孟君(1983—)，男，湖南株洲人，博士研究生。主要研究方向为通信与信息安全。

吴 波(1985—)，男，四川资中人，博士研究生。主要研究方向为通信与信息安全。

唐朝京(1962—)，男，江苏常州人，教授，博士生导师。主要研究方向为通信与信息安全。