



南京大學  
NANJING UNIVERSITY

---

## 自动化测试大作业实验报告

---

基于变异测试的模糊器评估

第 17 组 MMSD 组

### 小组成员信息

学号	姓名
201250123	刘晓旭
201250125	刘承杰
201250151	王骁
201250152	赵凝晖

文件地址: <https://box.nju.edu.cn/d/480d0f92fb69419d856c/>

2022 年 11 月 29 日

# 目录

<b>第一章 总体描述</b>	<b>1</b>
1.1 实验简介 . . . . .	1
1.2 工具介绍 . . . . .	1
1.3 实验环境 . . . . .	1
1.4 文档结构和实验结果 . . . . .	2
1.5 提交项目文件描述 . . . . .	2
<b>第二章 工具安装</b>	<b>4</b>
2.1 AFL 的安装 . . . . .	4
2.2 Mull 的安装 . . . . .	4
2.3 gnuplot 的安装 . . . . .	4
2.4 SQLite 的安装 . . . . .	4
<b>第三章 总体实验思路</b>	<b>5</b>
3.1 AFL 的使用 . . . . .	5
3.2 Mull 的使用 . . . . .	6
3.3 自动化执行脚本 . . . . .	8
3.3.1 执行 Mull 的输入 . . . . .	8
3.3.2 变异得分计算 . . . . .	9
<b>第四章 各项目遇到的问题 and 解决方法</b>	<b>10</b>
4.1 libxml2 . . . . .	10
4.1.1 automake 版本问题 . . . . .	10
4.1.2 找不到 pkg.m4 文件 . . . . .	10
4.1.3 缺少 python3.8 . . . . .	11
4.1.4 无法导入 python.h . . . . .	11
4.1.5 Mull 无法执行脚本文件 . . . . .	12
4.2 binutils . . . . .	13
4.3 matio . . . . .	13
4.4 stb . . . . .	13
4.5 openssl . . . . .	14
4.6 fmt . . . . .	15

4.7	expat . . . . .	16
4.8	lrzip . . . . .	16
4.9	zziplib . . . . .	17
4.10	splite . . . . .	17
4.11	exiv2 . . . . .	17
4.12	xpdf . . . . .	17
4.13	podofu . . . . .	18
4.13.1	安装时找不到依赖库 . . . . .	18
4.13.2	对于使用 Cmake 构建的项目的解决办法 . . . . .	18
4.14	w3m . . . . .	19
4.14.1	w3m 的安装 . . . . .	19
4.14.2	w3m 的种子 . . . . .	19
<b>第五章 结果分析</b>		<b>20</b>
5.1	实验目标汇总 . . . . .	20
5.2	数据处理方法 . . . . .	20
5.3	结果展示 . . . . .	21
5.4	结果分析 . . . . .	25

# 第一章 总体描述

## 1.1 实验简介

本次实验通过选择 libxml2、binutils、matio、stb、openssl、fmt、expat、lrzip、zziplib、splite、exiv2、xpdf、podofo 和 w3m 这 14 个开源项目，并对上述 14 个项目通过首先利用 AFL 对其进行模糊测试，产生测试输入，然后通过编写自动化脚本利用 Mull 运行 AFL 产生的测试输入，记录变异杀死情况，并对运行结果进行分析并绘制统计图的过程从变异杀死的角度对模糊器（AFL）进行评估。

## 1.2 工具介绍

本次实验工具为 AFL 和 Mull。

AFL 号称是当前最高级的 Fuzzing 测试工具之一，由 lcamtuf 所开发。在众多安全会议白帽演讲中都介绍过这款工具，以及 2016 年 defcon 大会的 CGC（Cyber Grand Challenge，形式为机器自动挖掘并修补漏洞）大赛中多支队伍利用 AFL fuzzing 技术与符号执行（Symbolic Execution）来实现漏洞挖掘，其中参赛队伍 shell-phish 便是采用 AFL（Fuzzing）+ angr（Symbolic Execution）技术。

变异测试是一种基于故障的软件测试技术。它通过计算变异分数和显示语义覆盖率的差距来评估测试套件的质量。它通过创建原始程序、变异体的几个略有修改的版本来做到这一点，并对每个变异体运行测试套件。如果测试套件检测到变化，或以其他方式幸存下来，则变异体被视为被杀死。如果至少有一个测试开始失败，变异体就会被杀死。

原始程序的每个变异都基于一组变异运算符。变异体是更改或删除原始程序中现有语句或表达式的预定义规则。每个规则都是确定性的：应用于同一程序的同一组变异运算符会导致相同的变异体集。Mull 是变异测试常用的测试工具。

在绘制结果图像时使用 gnuplot 工具。该工具是一个命令行的交互式绘图工具，用户通过输入命令，可以逐步设置或修改绘图环境，并以图形描述数据或函数，使得可以借由图形做更进一步的分析。

在存储数据结果时使用 SQLite。SQLite 是一个进程内的轻量级嵌入式数据库，它的数据库就是一个文件，实现了自给自足、无服务器、零配置的、事务性的 SQL 数据库引擎。

## 1.3 实验环境

本次实验均采用 VMware Workstation 16 PRO 运行 Ubuntu 20.04LTS 虚拟机运行环境，其中虚拟机配置的内存为 6G~8G 不等。

## 1.4 文档结构和实验结果

本文档接下来将从一下几个方面对整个实验进行描述进行描述

- AFL 和 Mull 的安装和配置过程
- 对整个实验的共性过程进行整体性介绍
- 分别对各项目的 AFL 和 Mull 的使用过程中需要注意的要点、踩的坑和得到的结果进行详细描述
- 最终结果分析

## 1.5 提交项目文件描述

提交的项目文件夹的文件结构如下图所示。

```
/
├── 1_libxml2
├── 2_binutils
├── 3_matio
├── 4_stb
├── 5_openssl
├── 6_fmt
├── 7_expats
├── 8_lrzip
├── 9_zziplib
├── 10_splite
├── 11_exiv2
├── 12_xpdf
├── 13_podofo
├── 14_w3m
├── AT-Fuzzing_Subjects.xlsx
├── mull.yml
├── mull_shell.sh
├── analysis_shell.sh
├── 变异得分
└── 实验报告.pdf
```

其中前 14 个文件夹为各项目文件夹；AT-Fuzzing\_Subjects.xlsx 文件为补充过的各项目汇总表，即 5.1 节提到的表格；mull.yml 文件为部分变异测试使用的变异规则，在 3.2 节有详细说明；mull\_shell.sh 为执行变异测试输入的自动化脚本，在 3.3.1 节有详细描述；analysis\_shell.sh 为分析处理变异得分结果的脚本，在 3.3.2 节有描述；变异得分文件夹为各项目的变异得分文件，以及对应的散点图，在 5.3 节有

所展示；实验报告.pdf 为本文件。

每一个项目文件夹中的内容如下（以 1\_libxml2 为例）

```
1_libxml2
├── 项目源码
├── AFL
│   ├── myin
│   └── myout
├── afl-fuzz 结果
└── mull 结果
```

AFL 文件夹中有 myin 和 myout 两个子文件夹，myin 文件夹里是模糊测试的种子文件，myout 文件夹里是模糊测试生成的文件；afl-fuzz 结果中的是模糊测试的可视化结果图；mull 结果是变异得分的 SQLite 文件。

对于 stb、openssl 和 fmt 这三个需要自己构建 C/C++ 文件的项目还有相应的 C/C++ 文件。

## 第二章 工具安装

### 2.1 AFL 的安装

1. 首先安装 Clang、LLVM、GCC 和 autogen。分别输入命令

```
1 sudo apt-get install clang
2 sudo apt-get install llvm
3 sudo apt-get install gcc
4 sudo apt-get install autoconf automake libtool
```

2. 安装 AFL：首先下载 AFL 安装包

```
1 get http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz
```

使用 `tar xvf afl-latest.tgz` 解压，在解压得到的文件夹下执行 `make` 和 `sudo make install`

3. 测试 AFL 是否安装成功：输入 `afl-f`，然后按 TAB 自动补全，如果能自动补全为 `afl-fuzz`，则说明安装成功

### 2.2 Mull 的安装

1. Mull 的安装：本实验安装 `mull-12`，其依赖 `Clang-12`，且 `Clang` 的版本必须是 12

```
1 curl -sLf 'https://dl.cloudsmith.io/public/mull-project/mull-stable/
  setup.deb.sh' | sudo -E bash
2 sudo apt-get update
3 sudo apt-get install mull-12
4 sudo apt-get install clang-12
```

2. 安装验证：输入 `mull-runner-12 --version` 以验证安装是否正确

### 2.3 gnuplot 的安装

使用命令 `sudo apt-get install gnuplot`

### 2.4 SQLite 的安装

安装数据库：`sudo apt-get install sqlite`

安装数据库图形化界面：`sudo apt-get install sqlitebrowser`

## 第三章 总体实验思路

### 3.1 AFL 的使用

1. 首先根据项目地址下载源代码并解压；
2. 对于 C 语言代码需要在项目代码文件夹下输入

```
1 CC=afl-gcc ./configure --disable-shared
```

对程序进行插桩，同理对于 C++ 代码则输入

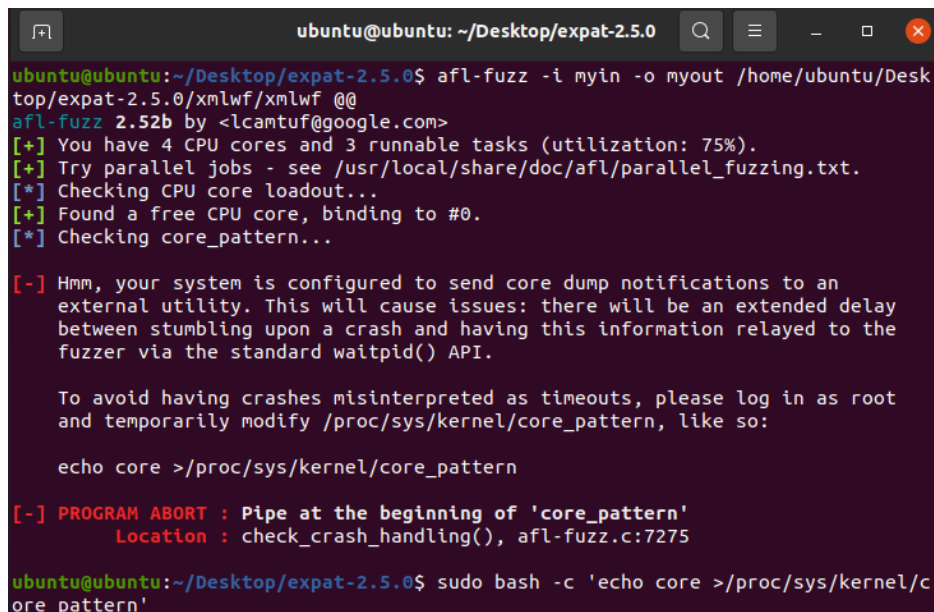
```
1 CXX=afl-g++ ./configure --disable-shared
```

如果程序是用 autoconf 构建的，则需要先运行 ./autogen.sh 生成 MakeFile；

3. 然后 make clean all 清除上次的 make 命令所产生的 object 文件及可执行文件并编译所有的目标；
4. 在编译的过程中，对于不同的软件会有不同的依赖，根据报错提示的内容安装即可；
5. 创建一个输入文件夹与输出文件夹，例如 myin 与 myout。myin 中放入需要的输入文件，例如 readelf 需要 elf 文件，大部分文件都可以在 AFL 目录下的 testcase 中找到；myout 存放执行 AFL 后生成的测试输入；
6. 输入命令 afl-fuzz -i myin -o myout <待测文件的绝对路径> @@，若存在参数，则加在前面。例如：

```
1 afl-fuzz -i myin -o myout /home/wh014/Desktop/binutils-2.39/binutils/readelf -a @@
```

这里有一点需要注意，当第一次执行 afl-fuzz 时，会出现如下报错信息



```
ubuntu@ubuntu: ~/Desktop/expat-2.5.0
ubuntu@ubuntu:~/Desktop/expat-2.5.0$ afl-fuzz -i myin -o myout /home/ubuntu/Desktop/expat-2.5.0/xmlwf/xmlwf @@
afl-fuzz 2.52b by <lcantuf@google.com>
[+] You have 4 CPU cores and 3 runnable tasks (utilization: 75%).
[+] Try parallel jobs - see /usr/local/share/doc/afl/parallel_fuzzing.txt.
[*] Checking CPU core loadout...
[+] Found a free CPU core, binding to #0.
[*] Checking core_pattern...

[-] Hmm, your system is configured to send core dump notifications to an
external utility. This will cause issues: there will be an extended delay
between stumbling upon a crash and having this information relayed to the
fuzzer via the standard waitpid() API.

To avoid having crashes misinterpreted as timeouts, please log in as root
and temporarily modify /proc/sys/kernel/core_pattern, like so:

echo core >/proc/sys/kernel/core_pattern

[-] PROGRAM ABORT : Pipe at the beginning of 'core_pattern'
    Location : check_crash_handling(), afl-fuzz.c:7275

ubuntu@ubuntu:~/Desktop/expat-2.5.0$ sudo bash -c 'echo core >/proc/sys/kernel/core_pattern'
```

图 1: 第一次执行 afl-fuzz 的报错信息



需要根据提示输入如下命令，且每次重启后都需要执行

```
1 sudo bash -c 'echo core >/proc/sys/kernel/core_pattern'
```

最终 afl-fuzz 执行界面如下（以 libxml2 为例），每个项目的执行时间都在一个小时以上，不再单独展示。

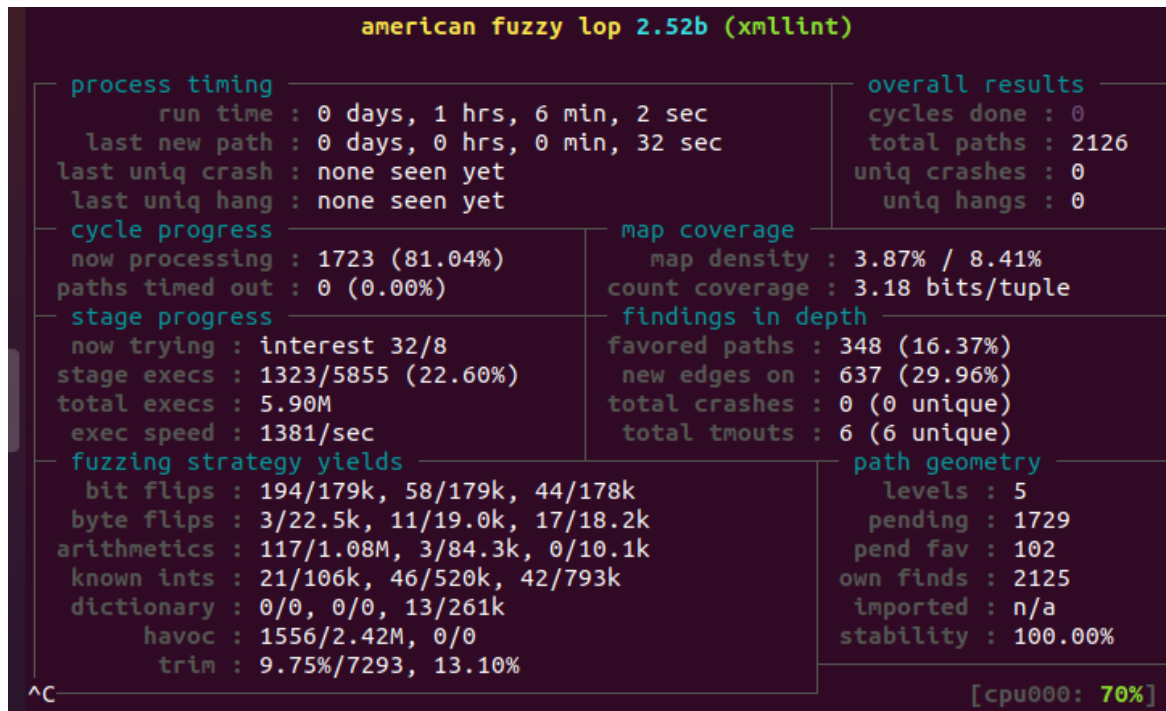


图 2: afl-fuzz 执行界面（以 libxml2 为例）

### 3.2 Mull 的使用

1. 在进行 Mull 插桩前需要新解压一个项目文件，不能在原有进行完 AFL 的文件基础上继续操作，因为 CFLAG 已经被更改，会发生 CFLAGS 冲突，产生如下报错：

```
1 configure: error: C compiler cannot create executables
```

2. 在插桩前需要编写配置文件 mull.yml，如果没有配置文件或者未检测到，会提示警告，并使用默认的配置。考虑到默认的配置过于复杂，运行时间过长，而且还会因为内存问题不断被 kill，产生如下报错：

```
1 /bin/bash: line 1: 189469 Killed
```

为次我们选择扩充虚拟机的内存到最大，但是仍然会出现内存溢出的问题，因此对于大型目标项目换成了相对简单的配置文件：

```
1 mutators:
2   - cxx_add_to_sub
```

3. 插桩：在目标文件目录下打开终端，对于 C 语言代码依次输入以下指令：

```
1 export CC=clang-12
```

```

2 export CFLAGS="-fexperimental-new-pass-manager -fpass-plugin=/usr/lib/
   mull-ir-frontend-12 -g -grecord-command-line"
3 ./configure
4 make

```

对于 C++ 代码则需要输入 `export CXX=clang++-12` 同样地，对于使用 `autoconf` 构建的项目需要再设置好 `CFLAGS` 后运行 `./autogen .sh` 以生成 `./configure` 文件，然后运行 `./configure`。

4. 运行：`mull-runner-12 <target> <inputfile>`，同时 `mull` 提供将运行结果存储到 SQLite 数据库文件中的选择

```

1 mull-runner-12 --reporters=SQLite --report-dir=<report_dir> <target> <
   inputfile>

```

由于通过 AFL 生成的输入过多，需要采用自动化脚本的形式读取输入执行 Mull，相应的脚本将在 3.3 节详细说明。

对于上述 Mull 的过程描述，还有几点需要补充说明。

首先，在设置 CC 时，`clang12` 的与 `mull` 相关的参数需要在 `CFLAGS` 中，不能直接在 CC 中设置；同时设置的等号两边不能有空格，否则报错

```

1 bash: export: '=': not a valid identifier

```

其次，根据官方文档的描述，在设置 `CFLAGS` 时的命令应该为

```

1 export CFLAGS="-o0 -fexperimental-new-pass-manager -fpass-plugin=/usr/lib/
   mull-ir-frontend-12 -g -grecord-command-line"

```

即设置编译优化等级为 `-o0`，但在实际尝试的时候，发现这样会产生 `CFLAGS` 错误

```

1 C compiler cannot create executables

```

这是因为项目的优化等级均为 `-o1` 或 `-o2`，因此需要删掉指令中的 `“-o0”`。

再次，在 `make` 时有可能会遇到报错

```

1 '.rodata' can not be used when making a PIE object
2 ld returned 1 exit status

```

这是库在建立连接时出了问题，导致文件无法正常编译。如果使用 `-no-pie` 命令，则文件无法正常通过编译，最终的解决办法是将 `CFLAGS` 命令修改为：

```

1 export CFLAGS="-fexperimental-new-pass-manager -fpass-plugin=/usr/lib/mull
   -ir-frontend-12 -g -grecord-command-line -fpic"

```

即加上 `-fpic` 后即可正确链接所使用的库。

最后，对于带参数的项目（如 `binutils` 中的 `readelf` 项目）无法被 `mull` 正常运行，如图 3 所示。

```
wh014@ubuntu:~/Desktop/bin$ mull-runner-12 /home/wh014/Desktop/bin/binutils/readelf -a /home/wh014/Desktop/binutils/out_1/queue/id:000000,orig:small_exec.elf
mull-runner-12: Unknown command line argument '-a'. Try: 'mull-runner-12 --help'
mull-runner-12: Did you mean '-h'?
```

图 3: Mull 的参数无法正确识别

如果不加参数则会提示需要一个参数，如图 4 所示。

```
[warning] Original test failed
status: Failed
stdout: ''
stderr: 'Usage: readelf <option(s)> elf-file(s)
Display information about the contents of ELF format files
Options are:
-a --all                Equivalent to: -h -l -S -s -r -d -V -A -I
-h --file-header        Display the ELF file header
-l --program-headers    Display the program headers
--segments              An alias for --program-headers
-S --section-headers    Display the sections' header
--sections              An alias for --section-headers
-g --section-groups     Display the section groups
-t --section-details    Display the section details
-e --headers            Equivalent to: -h -l -S
-s --syms               Display the symbol table
--symbols               An alias for --syms
--dyn-syms              Display the dynamic symbol table
--lto-syms              Display LTO symbol tables
--sym-base=[0|8|10|16] Force base for symbol sizes. The options are
                        mixed (the default), octal, decimal, hexadecimal.
-C --demangle[=STYLE]  Decode mangled/processed symbol names
                        STYLE can be "none", "auto", "gnu-v3", "java",
```

图 4: Mull 不加参数无法运行

这是因为命令格式不正确，将目标执行需要的参数识别成 mull-runner 的参数，因此需要在参数前加上 “--”，即（以 readelf -a 为例）

```
1 mull-runner-12 readelf -- -a inputfile
```

### 3.3 自动化执行脚本

整个项目中总共实现了两个自动化脚本，分别用来自动化执行 Mull 的输入和对 Mull 的结果进行变异得分的计算。

#### 3.3.1 执行 Mull 的输入

```
1 #!/bin/bash
2 function get_files(){
3     for element in `ls $1`
4     do
5         dir_or_file=$1$element
6         if [ -d $dir_or_file ]
7         then
8             getdir $dir_or_file
9         else
10            mull-runner-12 --reporters=SQLite --report-dir=$report_dir
11                $target_file $dir_or_file
12            fi
13    fi
```

```

12     done
13 }
14 #注意: input_dir路径需要写最后一个反斜杠/
15 input_dir="" #这里填写输入文件所在的绝对路径, 下同
16 target_file="" #这里填写需要执行的项目名和参数
17 report_dir="" #这里填写生成的SQLite文件存储的路径
18 get_files $input_dir

```

对于上述 shell 脚本, 直接在命令行中执行 `./shell.sh` 时可能出现无法执行的情况: `bash: ./shell.sh: Permission denied`, 需要改成 `bash` 执行该脚本, 即

```

1 bash shell.sh

```

### 3.3.2 变异得分计算

```

1 #!/bin/bash
2 function get_files(){
3     for element in `ls $1`
4     do
5         dir_or_file=$1$element
6         if [ -d $dir_or_file ]
7         then
8             getdir $dir_or_file
9         else
10            sqlite3 $dir_or_file <<EOF
11 create view killed_mutants as select * from mutant where status <> 2;
12 create view mullscore as select round((select count(*) from killed_mutants
13     ) * 1.0 /(select count(*) from mutant) * 100);
14 .output $dir_or_file.csv
15 select* from mullscore;
16 .output stdout
17 EOF
18
19         fi
20     done
21 cat $input_dir*.csv > $target_name.csv
22 rm $input_dir*.csv
23 }
24 #注意: input_dir路径需要写最后一个反斜杠/
25 input_dir="" #这里填写Mull结果的数据库文件的目录
26 target_name="" #生成csv文件的文件名
27 get_files $input_dir

```

## 第四章 各项目遇到的问题 and 解决方法

在第三章中已经介绍了相对通用的执行流程，但每个项目都有每个项目的特点，在实际执行的过程中都会遇到不同的问题需要解决，本章将针对每个项目介绍在实验中遇到的问题及其解决办法。

### 4.1 libxml2

#### 4.1.1 automake 版本问题

在 AFL 执行的过程中，该项目需要采用 autoconf 方式执行，执行后报错如下

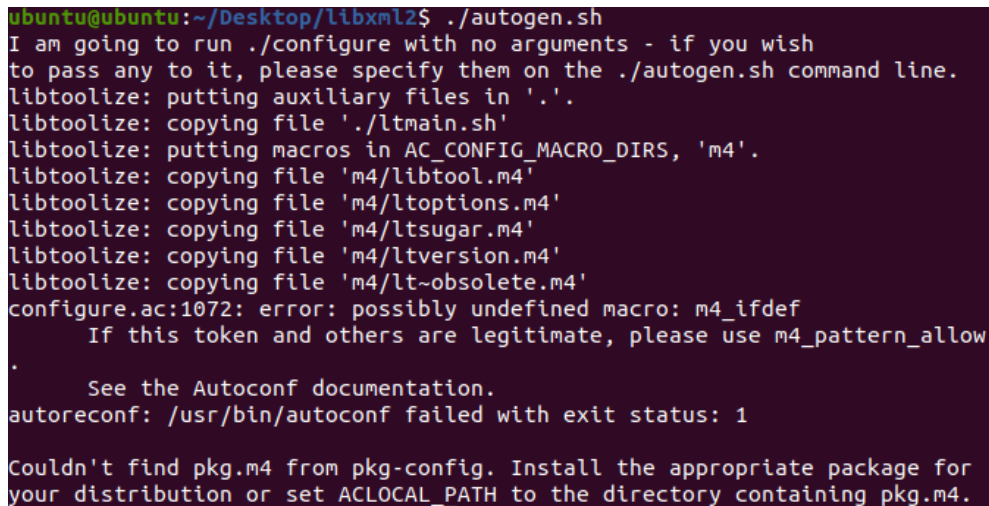
```
1 configure.ac: 42: error: require Automake 1.16.3, but have 1.16.1
```

这是 automake 的版本问题，需要先下载一个 automake 1.16.3，下载过程如下

```
1 wget https://ftp.gnu.org/gnu/automake/automake-1.16.3.tar.gz
2 tar -zxvf automake-1.16.3.tar.gz
3 cd automake-1.16.3
4 ./configure
5 make
6 make install
```

#### 4.1.2 找不到 pkg.m4 文件

解决上一个问题后，重新执行，出现如下报错



```
ubuntu@ubuntu:~/Desktop/libxml2$ ./autogen.sh
I am going to run ./configure with no arguments - if you wish
to pass any to it, please specify them on the ./autogen.sh command line.
libtoolize: putting auxiliary files in '.'.
libtoolize: copying file './ltmain.sh'
libtoolize: putting macros in AC_CONFIG_MACRO_DIRS, 'm4'.
libtoolize: copying file 'm4/libtool.m4'
libtoolize: copying file 'm4/ltoptions.m4'
libtoolize: copying file 'm4/ltsugar.m4'
libtoolize: copying file 'm4/ltversion.m4'
libtoolize: copying file 'm4/lt-obsolete.m4'
configure.ac:1072: error: possibly undefined macro: m4_ifdef
      If this token and others are legitimate, please use m4_pattern_allow
.
      See the Autoconf documentation.
autoreconf: /usr/bin/autoconf failed with exit status: 1

Couldn't find pkg.m4 from pkg-config. Install the appropriate package for
your distribution or set ACLOCAL_PATH to the directory containing pkg.m4.
```

图 5: 找不到 pkg.m4 文件

这个问题是因为大多数 .m4 文件都在 /usr/share/aclocal/ 目录下，但实际上 configure 的默认 aclocal 路径为 /usr/local/share/aclocal，因此需要将 /usr/share/aclocal/ 下的 \*.m4 文件都拷贝到 /usr/local/share/aclocal 目录下：

```
1 cp /usr/share/aclocal/*.m4 /usr/local/share/aclocal/
```

### 4.1.3 缺少 python3.8

再次执行 `./autogen.sh` 发现提示缺少 python3.8

```
configure: error: Package requirements (python-3.8) were not met:
No package 'python-3.8' found

Consider adjusting the PKG_CONFIG_PATH environment variable if you
installed software in a non-standard prefix.

Alternatively, you may set the environment variables PYTHON_CFLAGS
and PYTHON_LIBS to avoid the need to call pkg-config.
See the pkg-config man page for more details.

Configure script failed, check config.log for more info.
```

图 6: 缺少 python3.8

下面是安装 python3.8 的过程

```
1 wget https://www.python.org/ftp/python/3.8.0/Python-3.8.0a4.tar.xz
2 tar Jxf Python-3.8.0a4.tar.xz
3 cd Python-3.8.0a4
4 ./configure prefix=/usr/local/python3
5 make
6 sudo make install
```

以下是将系统 python 环境切换为当前安装的 python3.8 的过程，如果未安装过其他版本的 python 则不需要执行

```
1 sudo rm /usr/bin/python
2 sudo ln -s /usr/local/python3/bin/python3.8 /usr/bin/python
```

此时已经安装好了 python3.8，但重新执行后依然报上图的错误，仔细研究报错信息，应该设置 python 的环境变量

```
1 export PYTHON_CFLAGS="/usr/bin/python3.8"
2 export PYTHON_LIBS="/usr/lib/ovthon3.8"
```

再次重新执行，终于能够执行成功生成 configure

### 4.1.4 无法导入 python.h

执行插桩命令时会出现如下报错，这是因为 python 没有添加到环境变量中，程序找不到这个文件，解决办法如下

```
1 # 打开.bashrc文件
2 cd ~
3 sudo vim .bashrc
4 # 在末尾添加
5 export C_INCLUDE_PATH=/usr/include/python3.8:$C_INCLUDE_PATH
6 export LUS_INCLUDE_PATH=/usr/include/python3.8:$PLUS_INCLUDE_PATH
7 # 刷新
```

```
8 source .bashrc
```

```
afl-cc 2.52b by <lcantuf@google.com>
libxml.c:15:10: fatal error: Python.h: No such file or directory
  15 | #include <Python.h>
      |
compilation terminated.
make[4]: *** [Makefile:599: libxml.lo] Error 1
make[4]: Leaving directory '/home/ubuntu/Desktop/libxml2/python'
make[3]: *** [Makefile:725: all-recursive] Error 1
make[3]: Leaving directory '/home/ubuntu/Desktop/libxml2/python'
make[2]: *** [Makefile:494: all] Error 2
make[2]: Leaving directory '/home/ubuntu/Desktop/libxml2/python'
make[1]: *** [Makefile:1594: all-recursive] Error 1
make[1]: Leaving directory '/home/ubuntu/Desktop/libxml2'
make: *** [Makefile:781: all] Error 2
```

图 7: 无法导入 python.h

#### 4.1.5 Mull 无法执行脚本文件

按照 Mull 的执行流程最终执行时, 会出现如下错误

```
ubuntu@ubuntu:~/Desktop/newlibxml/libxml2$ mull-runner-12 /home/ubuntu/Desktop/n
ewlibxml/libxml2/xmllint /home/ubuntu/Desktop/libxml2/myout/queue/id:000000,orig
:title.xml
[info] Using config /home/ubuntu/Desktop/newlibxml/libxml2/mull.yml
[error] Cannot create SymbolicFile from: /home/ubuntu/Desktop/newlibxml/libxml2/
xmllint
[error] Error messages are treated as fatal errors. Exiting now.
```

图 8: Mull 无法执行脚本文件

这是因为 Mull 的目标应该是可执行文件, 而不是 shell 脚本。为了解决这个问题, 需要将该脚本转换为可执行文件。这里采用 shc 转换。

shc 的全称为 shell script compiler, 可以将 shell 脚本编译为二进制可执行文件, 从而达到隐藏源代码的目的。其的工作过程分为两步: 将 shell 脚本转化为 C 语言源码、将 C 语言源码进行编译链接得到二进制文件。其安装方式为

```
1 sudo apt-get install shc
```

使用方法为

```
1 shc -f xmllint.sh
```

得到两个文件: xmllint.sh.x 和 xmllint.sh.c。

然后需要对 xmllint.sh.c 文件重新进行插桩、编译, 方法为单文件编译的方法, 即

```
1 clang-12 -fexperimental-new-pass-manager -fpass-plugin=/usr/lib/mull-ir-
frontend-12 -g -grecord -command-line xmllint.x.c -o xmllintt
```

最后重新执行 Mull 即可, 结果如下



```

ubuntu@ubuntu:~/Desktop/newlibxml/libxml2$ mull-runner-12 /home/ubuntu/Desktop/n
ewlibxml/libxml2/xmllint /home/ubuntu/Desktop/libxml2/myout/queue/id:000000,ori
g:title.xml
[info] Using config /home/ubuntu/Desktop/newlibxml/libxml2/mull.yml
[warning] Could not find dynamic library: libc.so.6
[info] Warm up run (threads: 1)
[#####] 1/1. Finished in 242ms
[info] Filter mutants (threads: 1)
[#####] 1/1. Finished in 0ms
[info] Baseline run (threads: 1)
[#####] 1/1. Finished in 14ms
[info] Running mutants (threads: 2)
[#####] 2/2. Finished in 22ms
[info] Survived mutants (1/2):
/home/ubuntu/Desktop/newlibxml/libxml2/xmllint.x.c:963:12: warning: Survived: Re
placed + with - [cxx_add_to_sub]
        return 1 + (argc - a);
                ^
[info] Mutation score: 50%
[info] Total execution time: 323ms

```

图 9: libxml2 的最终执行结果

## 4.2 binutils

该项目的六个 target 按照第三章的提到的过程和补充说明执行没有遇到其他问题，因此不再赘述。

## 4.3 matio

该项目也是常规执行步骤，target 为 tools/matdump，输入的种子为.mat 文件

## 4.4 stb

该项目并不能为 C/C++ 库文件，并不能直接执行，而是需要编写 C++ 文件，对该文件进行 AFL 和 Mull 插桩进行模糊测试和变异测试。

编写的项目文件 stb.cpp 如下：

```

1 #include <iostream>
2 #define STB_IMAGE_IMPLEMENTATION
3 #include "stb_image.h"
4 #define STB_IMAGE_WRITE_IMPLEMENTATION
5 #include "stb_image_write.h"
6 #define STB_IMAGE_RESIZE_IMPLEMENTATION
7 #include "stb_image_resize.h"
8 #include <string>
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <vector>
12
13 using namespace std;
14
15 int main(int argc, char *argv[]) {
16     std::cout << "Hello, STB_Image" << std::endl;
17

```



```

18     string inputPath = argv[1];
19     int imageWidth, imageHeight, nrComponents;
20
21     // 加载图片获取宽、高、颜色通道信息
22     unsigned char *imagedata = stbi_load(inputPath.c_str(), &imageWidth, &
23         imageHeight, &nrComponents, 0);
24     std::cout << "测试 STB_Image 读取到的信息为: imageWidth = "<<
25         imageWidth <<
26         ",imageHeight = "<<imageHeight << ", nrComponents = " <<
27         nrComponents
28         << std::endl;
29
30     // 输出的宽和高为原来的一半
31     int outputWidth = imageWidth / 2;
32     int outputHeight = imageHeight / 2;
33     // 申请内存
34     auto *outputImageData = (unsigned char *)malloc(outputWidth *
35         outputHeight * nrComponents);
36     std::cout << "测试 STBIR_RESIZE, 改变后的图片尺寸为: outputWidth = "<<
37         outputWidth<<", outputHeight = "<< outputHeight << std::endl;
38     // 改变图片尺寸
39     stbir_resize(imagedata, imageWidth, imageHeight, 0, outputImageData,
40         outputWidth, outputHeight, 0, STBIR_TYPE_UINT8, nrComponents,
41         STBIR_ALPHA_CHANNEL_NONE, 0,
42         STBIR_EDGE_CLAMP, STBIR_EDGE_CLAMP,
43         STBIR_FILTER_BOX, STBIR_FILTER_BOX,
44         STBIR_COLORSPACE_SRGB, nullptr
45     );
46
47     return 0;
48 }

```

AFL 的执行命令为

```

1 afl-g++ -o stb stb.cpp
2 afl-fuzz -i min -o myout /home/ubuntu/Desktop/stb/stb @@

```

Mull 的执行命令为

```

1 clang++-12 -fexperimental-new-pass-manager -frass-plugin=/usr/lib/mull-ir-
  frontend-12 -g -grecord-command-line stb.cpp -o stb

```

## 4.5 openssl

该项目同样需要自己编写 C 语言文件，如下：

```

1 #include <openssl/md5.h>

```

```

2 #include <string.h>
3 #include <stdio.h>
4
5 int main(int argc, char *argv[]){
6     MD5_CTX ctx;
7     unsigned char outmd[16];
8     char buffer[1024];
9     int len=0;
10    int i;
11    FILE * fp=NULL;
12    memset(outmd,0,sizeof(outmd));
13    memset(buffer,0,sizeof(buffer));
14    char *filename = argv[1];
15    fp=fopen(filename,"rb");
16    if(fp==NULL){
17        printf("Can't open file\n");
18        return 0;
19    }
20
21    MD5_Init(&ctx);
22    while((len=fread(buffer,1,1024,fp))>0){
23        MD5_Update(&ctx,buffer,len);
24        memset(buffer,0,sizeof(buffer));
25    }
26    MD5_Final(outmd,&ctx);
27
28    for(i=0;i<16;i++){
29        printf("%02X",outmd[i]);
30    }
31    printf("\n");
32    return 0;
33 }

```

同时在使用 `afl-gcc` 和 `clang-12` 编译时需要指定链接库，否则会找不到 `openssl` 和 `md5` 库，需要加上 `-I/home/ubuntu/Desktop/openssl`。

## 4.6 fmt

该项目同样需要自己编写 C++ 文件，如下：

```

1 #include "fmt/core.h"
2 #include "fmt/ranges.h"
3 #include "iostream"
4 #include "fstream"
5 #include "string"
6

```

```

7 using namespace std;
8 int main(int argc, char * argv[]){
9     char *filename = argv[1];
10
11     ifstream infile;
12     infile.open(filename, ios::in);
13     if (!infile.is_open()){
14         cout << "读取文件失败" << endl;
15         return 0;
16     }
17     string buf;
18     while (getline(infile,buf)){
19         cout << buf;
20         int x = stoi(buf);
21         x++;
22         x--;
23         x += 1;
24         x -= 1;
25     }
26     return 0;
27 }

```

在编译时同样需要指定 `-lfmt` 链接库。

这里需要指出一点，`fmt` 是格式化输出库，上述文件单纯是为了凑出 Mull 可以变异的操作符而编写的。整体项目和后续变异没有什么关系，变异体能够被杀死的原因是通过 AFL 生成的输入在第 20 行转换成 `int` 时报错导致程序崩溃，因此后续将不会对该项目进行具体分析描述。

## 4.7 expat

该项目在 Mull 中同样遇到了目标是 shell 脚本的情况，解决办法参考 4.1.5 节。

## 4.8 lrzip

该项目在运行 `./configure` 时，报错提示

```
1 configure: error: Could not find zlib library - please install zlib-dev
```

但此时执行 `sudo apt-get install zlib-dev` 是不可以的，正确的安装命令是

```
1 sudo apt -y install zlib1g-dev
```

此时依然缺三个库，这三个库根据提示命令安装即可

```

1 sudo apt -y install libbz2-dev
2 sudo apt -y install liblz2-dev
3 sudo apt -y install liblz4-dev

```

## 4.9 zziplib

本项目是由 Cmake 构建的 C 语言项目，具体执行方法详见 4.13.2 节，这里不再赘述，只需要把 afl-g++ 换成 afl-gcc，clang++-12 换成 clang-12。

其 target 为 bins/zzcat。

## 4.10 splite

该项目为数据库，其 Mull 的执行过程极慢，执行一个变异分支大概需要 10 分钟，本次只跑完一个输入的变异测试就需要整整一天，执行完全部输入则需要 14 天整，因此将不会对该项目进行具体分析。

## 4.11 exiv2

本项目是由 Cmake 构建的 C 语言项目，具体执行方法详见 4.13.2 节，这里不再赘述。

本项目的 target 为 exiv。

## 4.12 xpdf

在运行该项目的时候，对于 fuzz 指令进行了几十次的尝试，尝试了的部分指令如下

```
1 afl-fuzz -i myin -o myout xpdf/pdfimages @@ image-root
2 afl-fuzz -i myin -o myout xpdf/pdfimages -png @@ image-root
3 afl-fuzz -i myin -o myout xpdf/pdfimages -list @@ png
```

均未能成功，同时也根据提示尝试了带一张图片、多张图片、不带图片的初始种子，也都没能成功，都会报错提示 odd! check syntax.。最终也没找到解决办法。

```
american fuzzy lop 2.52b (pdfimages)

process timing
  run time : 0 days, 0 hrs, 0 min, 17 sec
  last new path : none yet (odd, check syntax!)
  last uniq crash : none seen yet
  last uniq hang : none seen yet
cycle progress
  now processing : 0 (0.00%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 255/256 (99.61%)
  total execs : 16.1k
  exec speed : 878.7/sec
fuzzing strategy yields
  bit flips : 0/32, 0/31, 0/29
  byte flips : 0/4, 0/3, 0/1
  arithmetics : 0/222, 0/0, 0/0
  known ints : 0/26, 0/84, 0/44
  dictionary : 0/0, 0/0, 0/0
  havoc : 0/15.4k, 0/0
  trim : 97.73%/12, 0.00%

overall results
  cycles done : 57
  total paths : 1
  uniq crashes : 0
  uniq hangs : 0
map coverage
  map density : 0.02% / 0.02%
  count coverage : 1.00 bits/tuple
findings in depth
  favored paths : 1 (100.00%)
  new edges on : 1 (100.00%)
  total crashes : 0 (0 unique)
  total tmouts : 1 (1 unique)
path geometry
  levels : 1
  pending : 0
  pend fav : 0
  own finds : 0
  imported : n/a
  stability : 100.00%

[cpu000: 55%]
```

图 10: xpdf 项目无法运行 fuzz 报错

## 4.13 podof0

### 4.13.1 安装时找不到依赖库

在根据官方教程安装 podof0 时，发生找不到 mem.h 和 BaseTsd.h 的问题

```
42 /home/st1234/Desktop/podof0-0.9.8/CMakeFiles/CMakeTmp/CheckIncludeFile.c:1:10: fatal error:
   BaseTsd.h: No such file or directory
43   1 | #include <BaseTsd.h>
     |         ^~~~~~
44
45 compilation terminated.
```

图 11: podof0 安装时找不到 mem.h 和 BaseTsd.h

经过检查发现是缺少一系列的库。安装好这些库后需要重启，否则依然会找不到。

### 4.13.2 对于使用 Cmake 构建的项目的解决办法

该项目是用 Cmake 构建的 C++ 项目，因此不能使用第三章所提到的方法。在 AFL 时应该输入如下命令

```
1 cmake CMakeLists.txt -DCMAKE_C_COMPILER=afl-gcc -DCMAKE_CXX_COMPILER=afl-g
   ++
2 make
3 make install
```

在输入命令之前还需更改 CMakeLists: 在第二行添加 add\_compile\_options(-lm), 即

```
1 CMAKE_MINIMUM_REQUIRED(VERSION 2.6)
2 add_compile_options(-lm)
3 #***** IMPORTANT ***** IMPORTANT *****
4 # Look at http://www.vtk.org/Wiki/CMake\_HowToDoPlatformChecks
5 # and the other wiki entries before you add anything. You might not need to.
6 #*****
7
8 #
9 # Project name and version
10 #
11 PROJECT(PoDoFo)
```

图 12: 修改 CMakeLists

最终即可成功 make

在 Mull 执行时，应该输入的命令为

```
1 cmake -DCMAKE_CXX_FLAGS="-fexperimental-new-pass-manager -fpass-plugin=/
   usr/lib/mull-ir-frontend-12 -g -grecord-command-line"
2 make
3 make install
```

然后即可执行 Mull 的运行脚本。

## 4.14 w3m

### 4.14.1 w3m 的安装

1. 首先下载 w3m: <https://sourceforge.net/projects/w3m/files/>, 输入 `./configure` 命令编译安装
2. 出现错误提示缺少 `gc.h`, 所以需要下载 `gc`: 进入网址[http://www.hpl.hp.com/personal/Hans\\_Boehm/gc/gc\\_source/](http://www.hpl.hp.com/personal/Hans_Boehm/gc/gc_source/)下载并解压, 解压进入软件目录并分别输入 `./configure`、`make && make install`安装
3. 现在 w3m 可以正确运行 `./configure`, 但运行 `make`构建时会出现 `func.c`类似的错误
4. 为了解决上述问题,需要安装 `libgc`:<http://rpmfind.net/linux/rpm2html/search.php?query=libgc1>, 选择版本为 `libgc1-6.3-2mdk.i586.rpm`
5. 首先在视窗界面下,右键用安装软件的形式打开,并安装,输入命令 `/mktable 100 functable.tab > functable.c`
6. 最后再重复 w3m 的安装步骤即可成功安装

### 4.14.2 w3m 的种子

AFL 并没有提供 HTML 文件的种子, 本实验采用了最简单的 HTML 框架作为种子输入。

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title></title>
6 </head>
7 <body>
8 </body>
9 </html>
```

## 第五章 结果分析

本章将从变异得分的角度对结果进行分析。

### 5.1 实验目标汇总

对本实验涉及的 14 个项目的情况汇总如下：

序号	项目	目标 [参数]	备注
1	libxml2	<code>xmllint --valid --recover &lt;input&gt;</code>	
2	binutils	<code>readelf -a</code>	
		<code>objdump -SD</code>	
		<code>cxxfilt &lt;symbol&gt;</code>	
		<code>nm -C</code>	
		<code>size</code>	
		<code>strip &lt;input_file&gt; -o &lt;output_file&gt;</code>	
3	matio	<code>tools/matdump</code>	
4	stb	自己构造的 C/C++ 文件，详见4.4	
5	openssl	自己构造的 C/C++ 文件，详见4.5	
6	fmt	自己构造的 C/C++ 文件，详见4.6	
7	expat	<code>xmlwf</code>	
8	lrzip	<code>lrzip -f &lt;file&gt;</code>	
9	zziplib	<code>bins/zzcat</code>	
10	splite	<code>splite3</code>	运行时间极长，只运行了一部分，详见4.10
11	exiv2	<code>exiv</code>	
12	xpdf	<code>pdftimages &lt;input&gt; &lt;image-root&gt;</code>	运行失败，详见4.12
13	podof	<code>bin/podofocountpages</code>	
14	w3m	<code>w3m</code>	

图 13: 实验目标汇总

### 5.2 数据处理方法

首先通过 3.3.2 节的脚本对每个目标的变异得分进行计算，共计 18 个结果文件（第 12 个项目 xpdf 运行失败），每一个结果文件存储该项目对应其相应的每一个输入所得的变异得分。然后对每一个文件的变异得分绘制散点图并计算平均值，该工作借助 Excel 的数据数据处理功能完成。最后对各项目的变异得分进行汇总展示。

### 5.3 结果展示

各项目变异得分如下表所示，其中变异测试结果数是指采用 3.3.1 的执行 Mull 输入的脚本得到的结果数量，在某些项目中其值小于模糊测试得到的输入个数的原因是结果文件是以秒为单位生成的，对于执行速度快的项目，一秒会执行多个输入，但这些输入的结果均写在一个数据库文件中，所以会造成数据库文件的数量变少的情况。由于对于一个数据库文件计算变异得分已经取了所有条目的平均值，因此这种存储方式对结果并没有任何影响。

项目序号	项目名称	模糊测试得到输入个数	变异测试结果数	变异得分
1	libxml2	2126	271	99.45%
2	readelf	1522	960	4.58%
		921	920	20.68%
		1749	1749	0.17%
		595	595	50.92%
		590	590	39.76%
		799	799	98.03%
3	matio	2118	670	71.39%
4	stb	259	259	75.69%
5	opensssl	1	1	25%
6	fmt	9	2	0.00%
7	expat	4246	137	100%
8	lrzip	7	4	0.00%
9	zziplib	9	2	38%
10	splite	14	1	100%
11	exiv2	794	37	32.54%
12	xpdf	运行失败		
13	podofu	25	6	100%
14	w3m	1429	1416	1%

为了更加直观地展现实验结果，上表的结果绘制成可视化结果图如下页图 14 所示，其中 xpdf 项目在图中并没有画出。各项目变异得分的汇总簇状图如图 15 所示，xpdf 同样也没有画出。



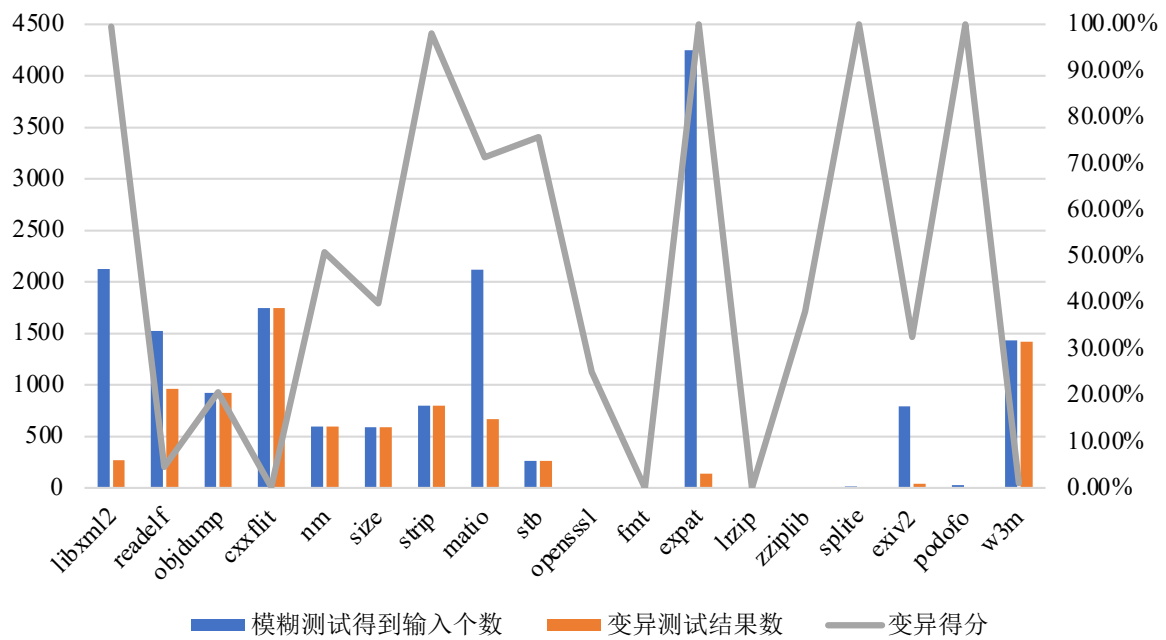


图 14: 各项目结果汇总

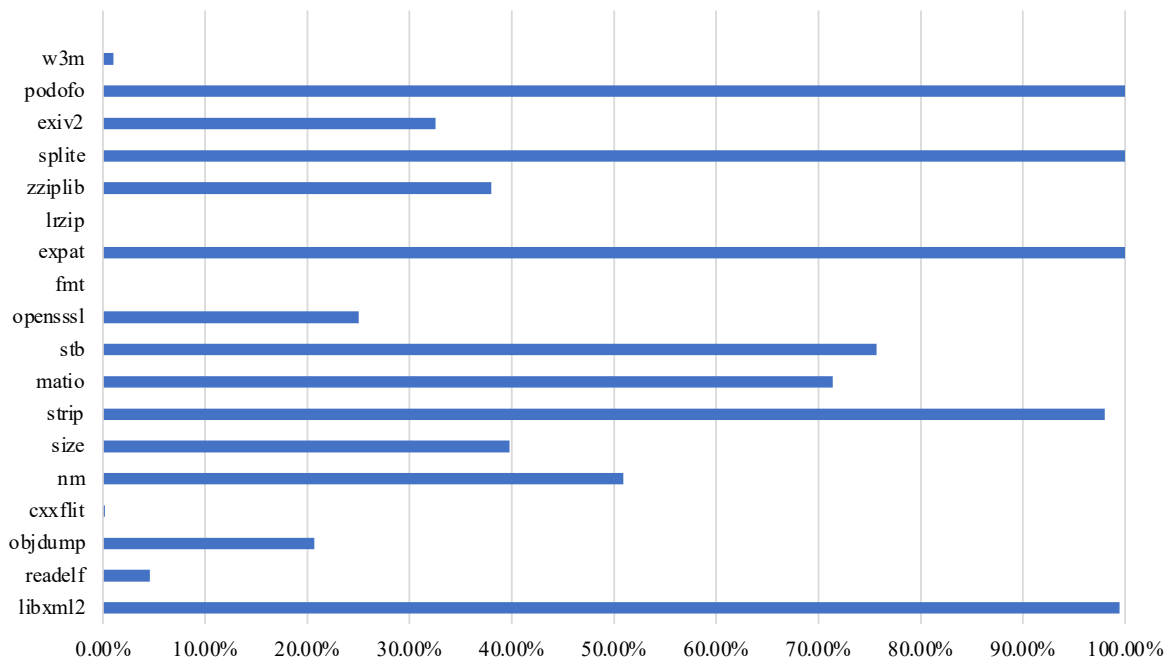


图 15: 各项目变异得分汇总

下页（图 16）展示的是的各项目变异得分分布的散点图，其中的每个点取自根据 Mull 结果数据库文件的一个文件算出的变异得分。由于 openssl 和 splite 只有一个变异测试结果，因此并没有展示在下图中，同样地，xpdf 也没有展示。图 17 是 AFL 的结果展示。

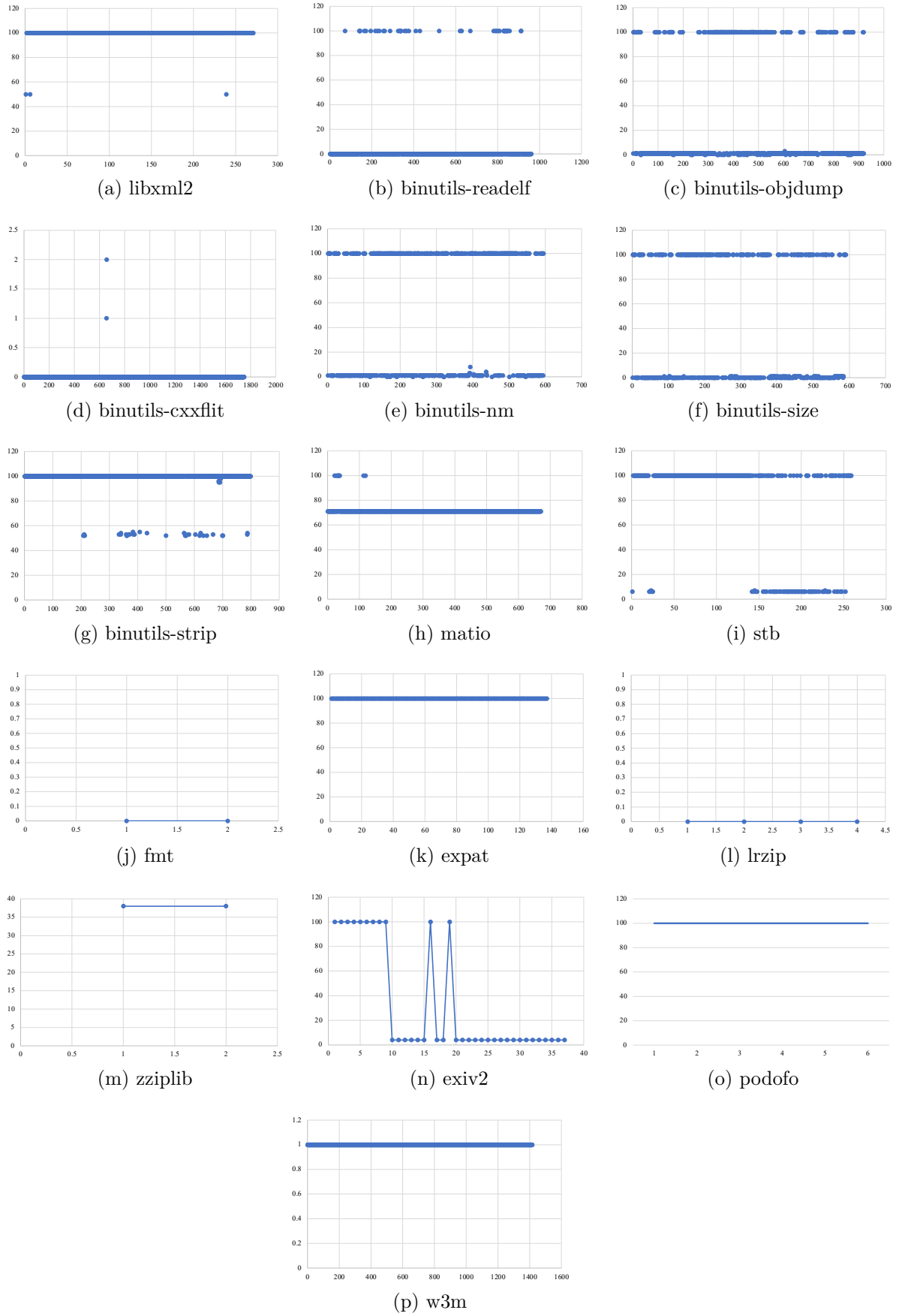


图 16: 各项目变异得分分布散点图

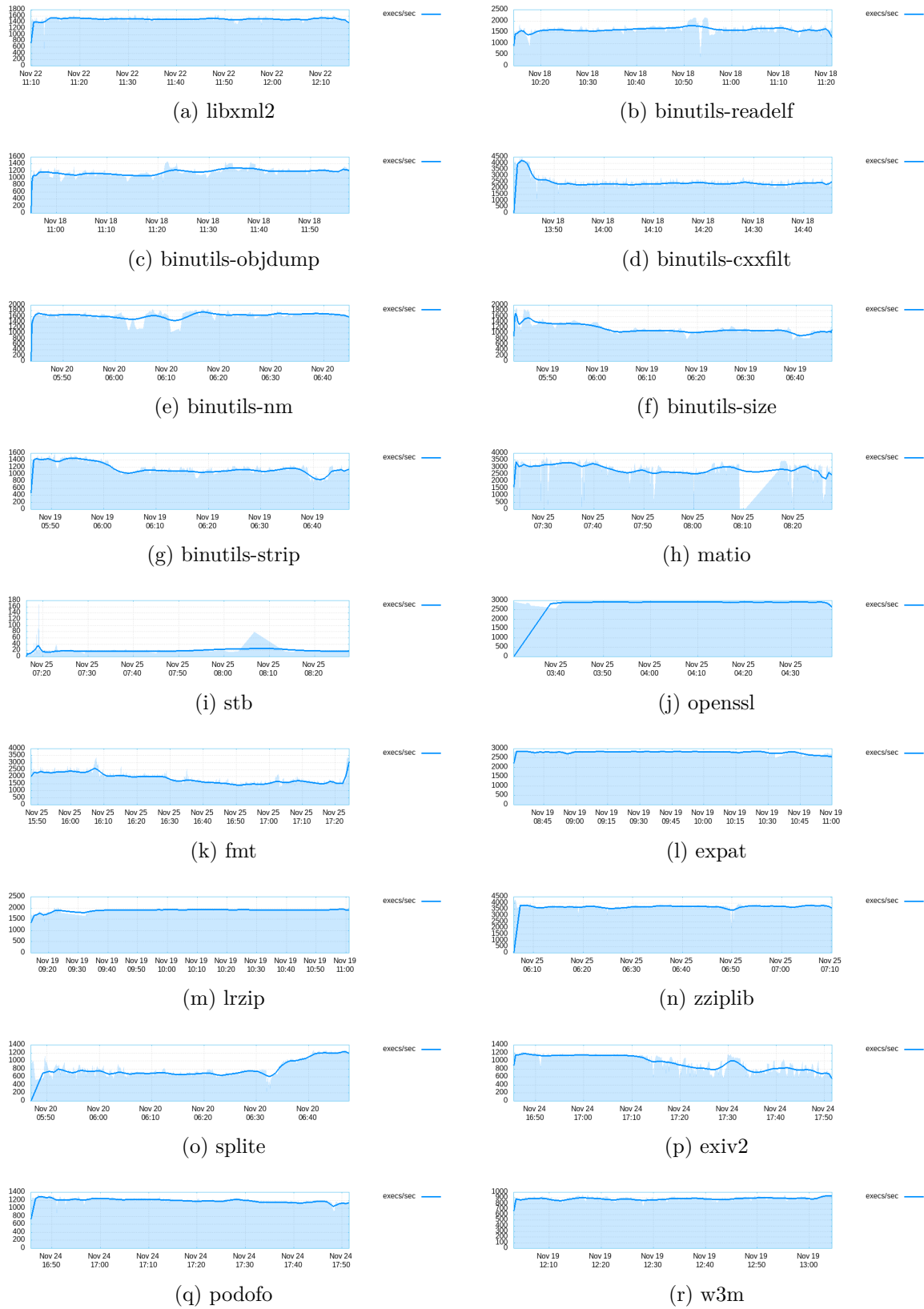


图 17: 各项目 AFL 结果

## 5.4 结果分析

从图 15 中可以看到，各项目的变异得分之间存在很大差异，且同一项目（binutils）的不同 target 见的变异得分也存在很大的差异。对于这些现象，我们认为其原因有三：

1. 所选的项目均为大型开源项目，其代码规模比较大，所选择的变异策略很难覆盖到与实际实现有关的部分；
2. 这些项目非常完善，很难产生严重的崩溃错误；
3. 不同项目的实现逻辑是不一样的，触发崩溃的条件也是不一样的，例如对于 pod-  
ofc 项目，其接受 pdf 文件，但模糊测试得到的测试输入均改变了文件格式，使  
输入均不是 pdf 文件，自然会导致全部杀死，变异得分为 100%。