一 libbcrypt项目

1.1 项目介绍

网址：https://github.com/rg3/libbcrypt

128 Star 45Fork

bcrypt 密码散列C 库。该库是一个简单的包装，为bcrypt密码哈希算法实现提供了方便的可重入接口。该源代码的精确副本包含在crypt_blowfish子目录中。bcrypt 库函数支持如下功能：

- 生成可以用于验证密码的哈希值。
- 专门设计用于验证密码并避免定时攻击。

bcrypt（以及crypt_blowfish）最重要的特性是它可以适应将来处理器性能的提高，使您可以任意增加检查密码的处理成本，同时仍保持与旧密码哈希的兼容性。 现在，您将要使用的bcrypt哈希值比传统的基于Unix DES或FreeBSD风格的MD5哈希值强几个数量级。

主要函数

```
int bcrypt_gensalt(int workfactor, char salt[BCRYPT_HASHSIZE])
```

该函数期望`workfactor`在4到31之间，并使用一个char数组来存储生成的`salt`。 char数组通常至少应具有`BCRYPT_HASHSIZE`字节。 如果提供的`workfactor`不在先前的范围内，它将默认为12。如果可以正确生成`salt`，则返回值为零，否则为非零。

```
int bcrypt_hashpw(const char *passwd, const char salt[BCRYPT_HASHSIZE],
                  char hash[BCRYPT_HASHSIZE]);
```

该函数希望对密码进行哈希处理，使用salt来对密码进行哈希处理，并使用char数组保留结果。
salt和hash参数都应至少具有BCRYPT_HASHSIZE字符的空间。
它还可以用于验证哈希密码。 在这种情况下，在salt参数中提供预期的哈希，并验证输出哈希与输入哈希相同。 但是，为避免定时攻击，最好在验证密码时使用bcrypt_checkpw。
如果可以对密码进行散列，则返回值为零，否则为非零。

```
int bcrypt_checkpw(const char *passwd, const char
hash[BCRYPT_HASHSIZE]);
```

该函数需要一个密码和一个哈希来验证密码。 内部实现已调整，以避免定时攻击。
如果有错误，则返回值为-1；如果提供的密码与给定的哈希值匹配，则返回值为零；如果未找到错误且密码不匹配，则返回值为大于零。

1.2 项目流程

- 在github上下载源码

```
git clone https://github.com/rg3/libbcrypt.git
```

- 修改libbcrypt文件夹内的Makefile文件

```
CC = $(AFL_HOME)/afl-gcc
#cc = gcc
```

```
CFLAGS = $(shell grep '^CFLAGS = ' crypt_blowfish/Makefile | cut -d= -f2-)
.PHONY: crypt_blowfish

all: pass_match_driver.a pass_match_driver pass_mismatch_driver

pass_match_driver: pass_match_driver.c crypt_blowfish
	$(CC) $(CFLAGS) -DTEST_BCRYPT -c pass_match_driver.c
	$(CC) -o pass_match_driver_test pass_match_driver.o crypt_blowfish/*.o

pass_match_driver.a: pass_match_driver.o crypt_blowfish
	ar r pass_match_driver.a pass_match_driver.o crypt_blowfish/*.o

pass_match_driver.o: pass_match_driver.c
	$(CC) $(CFLAGS) -c pass_match_driver.c

pass_mismatch_driver: pass_mismatch_driver.c crypt_blowfish
	$(CC) $(CFLAGS) -DTEST_BCRYPT -c pass_mismatch_driver.c
	$(CC) -o pass_mismatch_driver_test pass_mismatch_driver.o
crypt_blowfish/*.o

pass_mismatch_driver.a: pass_mismatch_driver.o crypt_blowfish
	ar r pass_mismatch_driver.a pass_mismatch_driver.o crypt_blowfish/*.o

pass_mismatch_driver.o: pass_mismatch_driver.c
	$(CC) $(CFLAGS) -c pass_mismatch_driver.c

crypt_blowfish:
	$(MAKE) -C crypt_blowfish

clean:
	rm -f *.o pass_match_driver pass_match_driver.a pass_mismatch_driver
pass_mismatch_driver.a *~ core
	$(MAKE) -C crypt_blowfish clean
```

- 修改libbcrypt/crypt_blowfish文件夹内的Makefile文件

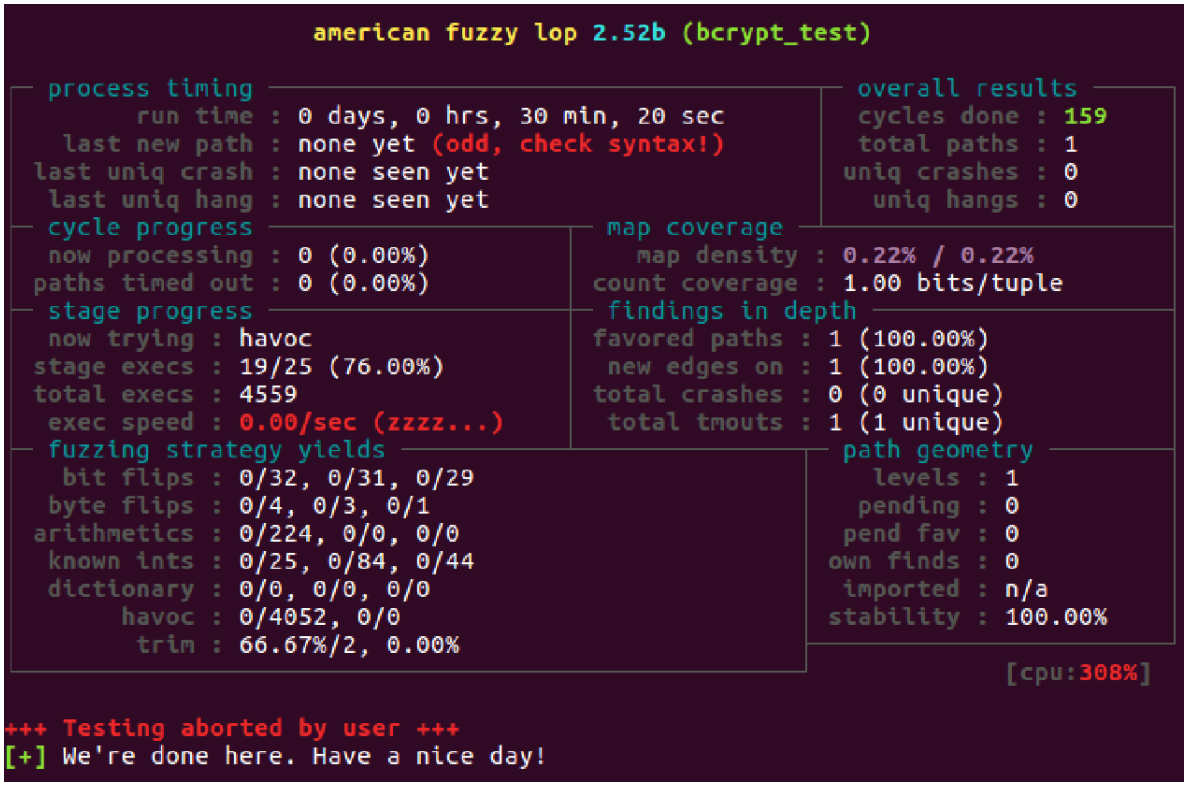
```
CC = $(AFL_HOME)/afl-gcc
#cc = gcc
AS = $(CC)
LD = $(CC)
RM = rm -f
CFLAGS = -W -Wall -Wbad-function-cast -Wcast-align -Wcast-qual -Wmissing-
prototypes -Wstrict-prototypes -Wshadow -Wundef -Wpointer-arith -O2 -fomit-
frame-pointer -funroll-loops
ASFLAGS = -c
LDFLAGS = -s

BLOWFISH_OBJS = \
    crypt_blowfish.o x86.o

CRYPT_OBJS = \
    $(BLOWFISH_OBJS) crypt_gensalt.o wrapper.o

TEST_OBJS = \
    $(BLOWFISH_OBJS) crypt_gensalt.o crypt_test.o

TEST_THREADS_OBJS = \
```

```
    $(BLOWFISH_OBJS) crypt_gensalt.o crypt_test_threads.o

EXTRA_MANS = \
    crypt_r.3 crypt_rn.3 crypt_ra.3 \
    crypt_gensalt.3 crypt_gensalt_rn.3 crypt_gensalt_ra.3

all: $(CRYPT_OBJS)

man: $(EXTRA_MANS)

$(EXTRA_MANS):
    echo '.so man3/crypt.3' > $@

crypt_blowfish.o: crypt_blowfish.h
crypt_gensalt.o: crypt_gensalt.h
wrapper.o: crypt.h ow-crypt.h crypt_blowfish.h crypt_gensalt.h

.c.o:
    $(CC) -c $(CFLAGS) $*.c

.S.o:
    $(AS) $(ASFLAGS) $*.S

clean:
    $(RM) crypt_test crypt_test_threads *.o $(EXTRA_MANS) core
```

- 编写`pass_match_driver.c`和`pass_mismatch_driver.c`两个驱动测试文件（见`2.3节`）
- 编译并进行测试

```
make clean
make
mkdir IN
cd IN
echo "Hello World" > test1.txt
cd ..
afl-fuzz -i IN/ -o OUT ./pass_match_driver_test @@
```

- 执行结果如图所示：



1.3 驱动编写

- `pass_match_driver.c`验证密码散列正确性（密码可以正确匹配）
- `pass_mismatch_driver.c`验证密码无法破解（错误密码无法匹配）

两个文件的编写见附件

二、c-algorithms项目

2.1 项目介绍

网址：https://github.com/fragglet/c-algorithms

2.1K Star 565 Fork

与其他现代编程语言相比，C编程语言仅包含非常有限的标准库。 本库是可以在C项目中使用的通用计算机科学数据结构和算法的集合。
该代码已获得ISC许可（BSD许可的简化版本，功能相同）的许可。 这样它可以在任何项目中重用，无论是专有项目还是开源项目。

2.2 项目流程

- 在github上下载项目

```
git clone https://github.com/fragglet/c-algorithms.git
```

- 将`test`文件夹中的下列文件，拷贝到`src`文件夹中

```
alloc-testing.c
alloc-testing.h
framework.c
framework.h
```

- 将Makefile文件和两个测试驱动程序（`test-binary-heap.c`和`test-trie.c`）拷贝至`c-algorithms`目录
- 编译并进行测试

```
# 建立 obj 文件夹保存生成的*.o文件
mkdir obj
make clean
make
mkdir IN
cd IN
echo "Hello World" > test1.txt
cd ..
afl-fuzz -i IN/ -o OUT ./test_binary_heap @@
```

1.3 驱动编写

- `test-binary-heap.c`验证堆实现的正确性
- `test-trie.c`验证`Trie`树实现的正确性

两个文件的编写见附件

三 log.c项目（过于简单，暂时废弃）

3.1 项目介绍

网址：https://github.com/rxi/log.c

654 Star 166 Fork

使用C99实现的简单日志记录库

3.2 项目流程

- 在github上下载源码

```
git clone https://github.com/rxi/log.c.git
```

- 在`src`文件夹内添加简单的测试代码

```
# 文件名：logtest.c
#include <stdio.h>
#include <stdlib.h>

#include "log.h"

int main(int argc, char *argv[]) {
    log_set_level(LOG_INFO);
    if(argc == 1){
        log_debug("Hello, World!");
        log_warn("%d", argc);
    }
    int i = 1;
    while (i < argc){
        log_fatal(argv[i]);
```

```
        i += 1;
    }
    return 0;
}
```

- 编写Makefile文件：

```
CFLAGS ?= -Wall -Os -fPIC
CC = $(AFL_HOME)/afl-gcc

all : logtest logtest.o log.o

logtest : logtest.o log.o
    $(CC) $(CFLAGS) -o $@ $^

log.o : log.c log.h
    $(CC) -c log.c

logtest.o : logtest.c log.h
    $(CC) -c logtest.c

clean:
    rm -f *.o logtest
```

- 执行如下命令进行编译和模糊测试：

```
make clean
make
mkdir IN
cd IN
echo "Hello World" > test1.txt
cd ..
afl-fuzz -i IN/ -o OUT ./logtest @@
```

- 执行结果如图所示：