

— 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

```
1 | 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。

```
1 | int bcrypt_hashpw(const char *passwd, const char salt[BCRYPT_HASHSIZE],
2 |                  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。

```
1 | 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

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

- 修改libbcrypt文件夹内的Makefile文件

```
1 | CC = $(AFL_HOME)/afl-gcc
2 | #CC = gcc
```

```

3 CFLAGS = $(shell grep '^CFLAGS = ' crypt_blowfish/Makefile | cut -d= -f2-)
4 .PHONY: crypt_blowfish
5
6 all: pass_match_driver.a pass_match_driver pass_mismatch_driver
7
8 pass_match_driver: pass_match_driver.c crypt_blowfish
9     $(CC) $(CFLAGS) -DTEST_BCRYPT -c pass_match_driver.c
10    $(CC) -o pass_match_driver_test pass_match_driver.o crypt_blowfish/*.o
11
12 pass_match_driver.a: pass_match_driver.o crypt_blowfish
13    ar r pass_match_driver.a pass_match_driver.o crypt_blowfish/*.o
14
15 pass_match_driver.o: pass_match_driver.c
16    $(CC) $(CFLAGS) -c pass_match_driver.c
17
18 pass_mismatch_driver: pass_mismatch_driver.c crypt_blowfish
19    $(CC) $(CFLAGS) -DTEST_BCRYPT -c pass_mismatch_driver.c
20    $(CC) -o pass_mismatch_driver_test pass_mismatch_driver.o
    crypt_blowfish/*.o
21
22 pass_mismatch_driver.a: pass_mismatch_driver.o crypt_blowfish
23    ar r pass_mismatch_driver.a pass_mismatch_driver.o crypt_blowfish/*.o
24
25 pass_mismatch_driver.o: pass_mismatch_driver.c
26    $(CC) $(CFLAGS) -c pass_mismatch_driver.c
27
28 crypt_blowfish:
29    $(MAKE) -C crypt_blowfish
30
31 clean:
32    rm -f *.o pass_match_driver pass_match_driver.a pass_mismatch_driver
    pass_mismatch_driver.a *~ core
33    $(MAKE) -C crypt_blowfish clean

```

- 修改libbrcrypt/crypt_blowfish文件夹内的Makefile文件

```

1 CC = $(AFL_HOME)/afl-gcc
2 #cc = gcc
3 AS = $(CC)
4 LD = $(CC)
5 RM = rm -f
6 CFLAGS = -W -Wall -Wbad-function-cast -Wcast-align -Wcast-qual -Wmissing-
    prototypes -Wstrict-prototypes -Wshadow -Wundef -Wpointer-arith -O2 -fomit-
    frame-pointer -funroll-loops
7 ASFLAGS = -c
8 LDFLAGS = -s
9
10 BLOWFISH_OBJS = \
11     crypt_blowfish.o x86.o
12
13 CRYPT_OBJS = \
14     $(BLOWFISH_OBJS) crypt_gensalt.o wrapper.o
15
16 TEST_OBJS = \
17     $(BLOWFISH_OBJS) crypt_gensalt.o crypt_test.o
18
19 TEST_THREADS_OBJS = \

```

```

20     $(BLOWFISH_OBJS) crypt_gensalt.o crypt_test_threads.o
21
22 EXTRA_MANS = \
23     crypt_r.3 crypt_rn.3 crypt_ra.3 \
24     crypt_gensalt.3 crypt_gensalt_rn.3 crypt_gensalt_ra.3
25
26 all: $(CRYPT_OBJS)
27
28 man: $(EXTRA_MANS)
29
30 $(EXTRA_MANS):
31     echo '.so man3/crypt.3' > $@
32
33 crypt_blowfish.o: crypt_blowfish.h
34 crypt_gensalt.o: crypt_gensalt.h
35 wrapper.o: crypt.h ow-crypt.h crypt_blowfish.h crypt_gensalt.h
36
37 .c.o:
38     $(CC) -c $(CFLAGS) $*.c
39
40 .S.o:
41     $(AS) $(ASFLAGS) $*.S
42
43 clean:
44     $(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

```

1 make clean
2 make
3 mkdir IN
4 cd IN
5 echo "Hello world" > test1.txt
6 cd ..
7 afl-fuzz -i IN/ -o OUT ./pass_match_driver_test @@

```

- 执行结果如图所示：

```
american fuzzy lop 2.52b (bcrypt_test)

process timing |-----| overall results
  run time : 0 days, 0 hrs, 30 min, 20 sec | cycles done : 159
  last new path : none yet (odd, check syntax!) | total paths : 1
  last uniq crash : none seen yet | uniq crashes : 0
  last uniq hang : none seen yet | uniq hangs : 0
-----|-----|
cycle progress |-----| map coverage
  now processing : 0 (0.00%) | map density : 0.22% / 0.22%
  paths timed out : 0 (0.00%) | count coverage : 1.00 bits/tuple
-----|-----|
stage progress |-----| findings in depth
  now trying : havoc | favored paths : 1 (100.00%)
  stage execs : 19/25 (76.00%) | new edges on : 1 (100.00%)
  total execs : 4559 | total crashes : 0 (0 unique)
  exec speed : 0.00/sec (zzzz...) | total tmouts : 1 (1 unique)
-----|-----|
fuzzing strategy yields |-----| path geometry
  bit flips : 0/32, 0/31, 0/29 | levels : 1
  byte flips : 0/4, 0/3, 0/1 | pending : 0
  arithmetics : 0/224, 0/0, 0/0 | pend fav : 0
  known ints : 0/25, 0/84, 0/44 | own finds : 0
  dictionary : 0/0, 0/0, 0/0 | imported : n/a
  havoc : 0/4052, 0/0 | stability : 100.00%
  trim : 66.67%/2, 0.00%
-----|-----|
[cpu:308%]

+++ Testing aborted by user +++
[+] We're done here. Have a nice day!
```

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

```
1 | 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

```
1 | alloc-testing.c
2 | alloc-testing.h
3 | framework.c
4 | 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

```
1 # 建立 obj 文件夹保存生成的*.o文件
2 mkdir obj
3 make clean
4 make
5 mkdir IN
6 cd IN
7 echo "Hello world" > test1.txt
8 cd ..
9 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

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

- 在 `src` 文件夹内添加简单的测试代码

```
1 # 文件名: logtest.c
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 #include "log.h"
6
7 int main(int argc, char *argv[]) {
8     log_set_level(LOG_INFO);
9     if(argc == 1){
10         log_debug("Hello, world!");
11         log_warn("%d", argc);
12     }
13     int i = 1;
14     while (i < argc){
15         log_fatal(argv[i]);
```

```

16     i += 1;
17 }
18 return 0;
19 }

```

- 编写Makefile文件:

```

1 CFLAGS ?= -Wall -Os -fPIC
2 CC = $(AFL_HOME)/afl-gcc
3
4 all : logtest logtest.o log.o
5
6 logtest : logtest.o log.o
7     $(CC) $(CFLAGS) -o $@ $^
8
9 log.o : log.c log.h
10     $(CC) -c log.c
11
12 logtest.o : logtest.c log.h
13     $(CC) -c logtest.c
14
15 clean:
16     rm -f *.o logtest

```

- 执行如下命令进行编译和模糊测试:

```

1 make clean
2 make
3 mkdir IN
4 cd IN
5 echo "Hello world" > test1.txt
6 cd ..
7 afl-fuzz -i IN/ -o OUT ./logtest @@

```

- 执行结果如图所示:

american fuzzy lop 2.52b (logtest)

process timing		overall results	
run time	: 0 days, 2 hrs, 55 min, 27 sec	cycles done	: 116k
last new path	: none yet (odd, check syntax!)	total paths	: 1
last uniq crash	: none seen yet	uniq crashes	: 0
last uniq hang	: none seen yet	uniq hangs	: 0
cycle progress		map coverage	
now processing	: 0 (0.00%)	map density	: 0.02% / 0.02%
paths timed out	: 0 (0.00%)	count coverage	: 1.00 bits/tuple
stage progress		findings in depth	
now trying	: havoc	favorable paths	: 1 (100.00%)
stage execs	: 236/256 (92.19%)	new edges on	: 1 (100.00%)
total execs	: 29.8M	total crashes	: 0 (0 unique)
exec speed	: 5958/sec	total tmouts	: 2815 (4 unique)
fuzzing strategy yields		path geometry	
bit flips	: 0/32, 0/31, 0/29	levels	: 1
byte flips	: 0/4, 0/3, 0/1	pending	: 0
arithmetics	: 0/224, 0/0, 0/0	pend fav	: 0
known ints	: 0/25, 0/84, 0/44	own finds	: 0
dictionary	: 0/0, 0/0, 0/0	imported	: n/a
havoc	: 0/29.8M, 0/0	stability	: 100.00%
trim	: 66.67%/2, 0.00%		

^C [cpu:308%]

