Fuzzing Crash Report

Students Ernesto Ortiz, Brittany Boles, Garrett Perkins, Zach Wadhams
Professor Dr. Matthew Revelle
Date October 10, 2024

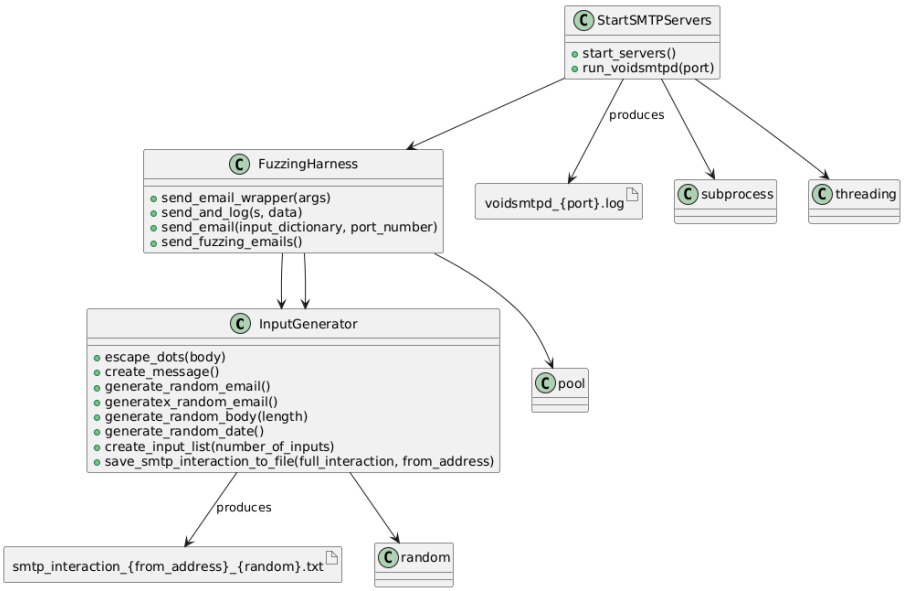

Figure 1: Do you guys like this? We don't have to use it..

Introduction

In this assignment, we fuzz an SMTP server by sending inputs of variable symbols and lengths in the different fields that the protocol requires. The fuzzer is written in Python and has three classes: an InputGenerator class, a StartSMTPServers class, and a FuzzingHarness class.

The StartSMTPServers class is responsible for starting the SMTP server processes. Right now, it is set to start ten servers.

The InputGenerator class has the structure required for SMTP communication. Based on that structure, it varies the values that are sent with the different protocol commands, and the message body itself.

The FuzzingHarness class does the actual fuzzing. It first creates an InputGenerator object and right now it is set up to create a hundred unique inputs, but it could generate as many as are needed. Each input is matched with a port number. The matching is done in a way that assures that each server receives about the same amount of requests. Once the inputs are ready, several threads are started (we use 10). These threads execute the send_email_wrapper method which is in charge of executing the communication with the servers. The threads keep executing this method until all inputs have been sent.

During execution, a log file will be generated for each server. These files are named voidsmtpd_portNumber.log. They contain the message that the server sanitizer produces when the server crashes. Additionally, each individual input is saved to a smtp_interaction_fromAddress_randomInt.txt. The contents of any of these files can be sent to a server to reproduce a fuzzing run.

Crash 1 - SEGV from too large message body

The first discovered crash originates from the size of the message body passed to the `voidsmtp` server. In our fuzzer's `InputGenerator` class, the `create_message()` function assembles each part of the message. For the message body however, the `generate_random_body()` function is as follows:

```
def generate_random_body(self, length):
```

```
characters = string.ascii_letters + string.digits + string.punctuation
return ''.join(random.choice(characters) for _ in range(length))
```

It takes in the desired length of the created string as an integer. A random string consisting of ASCII letters, digits, and punctuation is then returned to be inserted as the message body. Initially, we tried a random length between 1000 and 5000 characters. Upon running this, our log indicated that the first message was accepted and placed into the ring buffer regardless of the body size. The second large body was then accepted and placed into the ring buffer, but when the `voidsmtp` server attempts to print the ring buffer, the server would crash midway through the second large message body and provide the address sanitizer output. This output indicated a SEGV signal, or a segmentation fault, caused by an out of bounds read attempt. Following the provided stack trace, the problem originated in the `print_list()` function within `voidsmtp`. Without access to the source code, we cannot determine the cause with certainty. Our best guess however, is that this large body, overflowed outside of the program's allocated memory. When the program tried to read the entire message, it stepped outside of its allocated memory and triggered a segmentation fault.

Crash 2 - Global buffer overflow from repeated invalid CCs

(Even though we are saying it is a cc address, in reality the crash is produced when we use the RCPT TO command to insert additional addresses. There is no CC command in the communication.) The second crash seems to originate from repeatedly providing invalid CC emails to the `voidsmtp` program.

Crash 3 - SEGV for random date inputs

The segmentation fault in vvoidsmtpd was caused by an invalid memory access within the `print_list` function. It seems similar to Section 1, but was in a different location using different testing parameters. The code for the randomized date generates a random date and time within a specified range, assigns it a random time zone, formats it in a random date format, and appends a random amount of padding spaces to the formatted date string. The input contained in `crash3_input.txt` caused the program to crash within the print list function.

Crash 4-interceptor strcpy

Per the AddressSanitizer output, The crash occurs in the function `__interceptor_strcpy`, which suggests an issue with string handling. The overflow is happening near a global variable `msgs` defined in the `voidsmtpd` program at line 34. This variable seems to be a buffer of size 2720 bytes, and the overflow happens right after it. AddressSanitizer shadow memory reports that the address at fault is out-of-bounds (0 bytes to the right of global variable `msgs`), meaning the buffer `msgs` was overrun by at least one byte. This could be because `strcpy` might not be checking the size of the destination buffer before copying data. Below the input data and what we got in return. It was reproducing however now we can't quite get it again.

Message 2
MAIL FROM:
< 7*sko*@5.*gov* >
RCPT TO:
< 866*o*4*csx*30*rb*8*yxngh*4*hjx*671*md*35*oozamrwxrduinbpmv*'*yqxml*3*ai*4*ug*626*a*258*pg*0*s*81*zwag*309*ydswkcv*7*p*6*stg*5*pruici*2*bcm*4*mue#edu*; *ep*8*x*@*e.edu*; *ph*8*natddt*@*wvpo*0*f*4*hytudrbe*25*bktst*5*jrtrjk*4*ajctvghjcq*0*vq.org*; *i*@*oc*5*kp*+*org*; *a*8*pbcoqhe*3*e*68*td*3*h*4*as*64*gqkrye.pt*8*xq*210*plyvwyhqzcqz*5*ay* ==========

==========

```
<86bb04csx3urb8yxngh4hjx6/1md35oozamrwxrdu1nbpmv yqxml3a14ug626a258pg0s81zwag309ydswkcv/postg5pru1c12bcm4mue#edu;ep8x@e.edu;ph8natddt@wvp08f4hytudrbe25bkt
==541428==ERROR: AddressSanitizer: global-buffer-overflow on address 0x55555555d5a0 at pc 0x7ffff74544bf bp 0x7fffffffd300 sp 0x7fffffffcaa8
WRITE of size 521 at 0x55555555d5a0 thread T0
    #0 0x7ffff74544be in __interceptor_strcpy ../../../../src/libsanitizer/asan/asan_interceptors.cpp:440
    #1 0x55555555717b in handle_client (/home/brittanyboles/AutoVulnDiscovery/Assignment2/voidsmtpd+0x317b)
    #2 0x55555556cf1 in main (/home/brittanyboles/AutoVulnDiscovery/Assignment2/voidsmtpd+0x2cf1)
    #3 0x7ffff7029d8f in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #4 0x7ffff7029e3f in __libc_start_main_impl ../csu/libc-start.c:392
    #5 0x555555556664 in _start (/home/brittanyboles/AutoVulnDiscovery/Assignment2/voidsmtpd+0x2664)

0x55555555d5a0 is located 0 bytes to the right of global variable 'msgs' defined in 'voidsmtpd.c:34:9' (0x55555555cb00) of size 2720
0x55555555d5a0 is located 32 bytes to the left of global variable 'msg_idx' defined in 'voidsmtpd.c:35:14' (0x55555555d5c0) of size 4
SUMMARY: AddressSanitizer: global-buffer-overflow ../../../../src/libsanitizer/asan/asan_interceptors.cpp:440 in __interceptor_strcpy
Shadow bytes around the buggy address:
  0x0aab2aaa3a60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0aab2aaa3a70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0aab2aaa3a80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0aab2aaa3a90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0aab2aaa3aa0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0aab2aaa3ab0: 00 00 00 00[f9]f9 f9 f9 04 f9 f9 f9 f9 f9 f9 f9
  0x0aab2aaa3ac0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0aab2aaa3ad0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0aab2aaa3ae0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0aab2aaa3af0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0aab2aaa3b00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
```