Project #3 – Fuzzing with AFL

In this project, your task is to set up an AFL fuzzer and to use it to find bugs in our PNG parsing library.

Credit Statement

- Fuzzing Applications with American Fuzzy Lop (AFL) Medium article

Task

You are required to find and fix at least 3 bugs in *pngparser.c*, which all will be reported in your write-up, and answer the following questions.

Q1. Why did you need to change *is_png_chunk_valid()*?

- This function reads the crc value of each png chunk in the png image we're parsing and checks whether this chunk is a valid chunk or not. Therefore, this function prevents corrupted data from being parsed by our pngparser program. However, since this function is called on at the very beginning of pngparser.c, if we kept the return value of this function the way it is and not changed it to 1, the fuzzer wouldn't have been able to explore as many paths deep in the program as we wish to, because it would have always been blocked by is_png_chunk_valid() when the data passed was corrupted, and wouldn't have been able to find as many bugs as we want to in the program. Therefore, we needed to change the return value of is_png_chunk_valid() to be always 1.

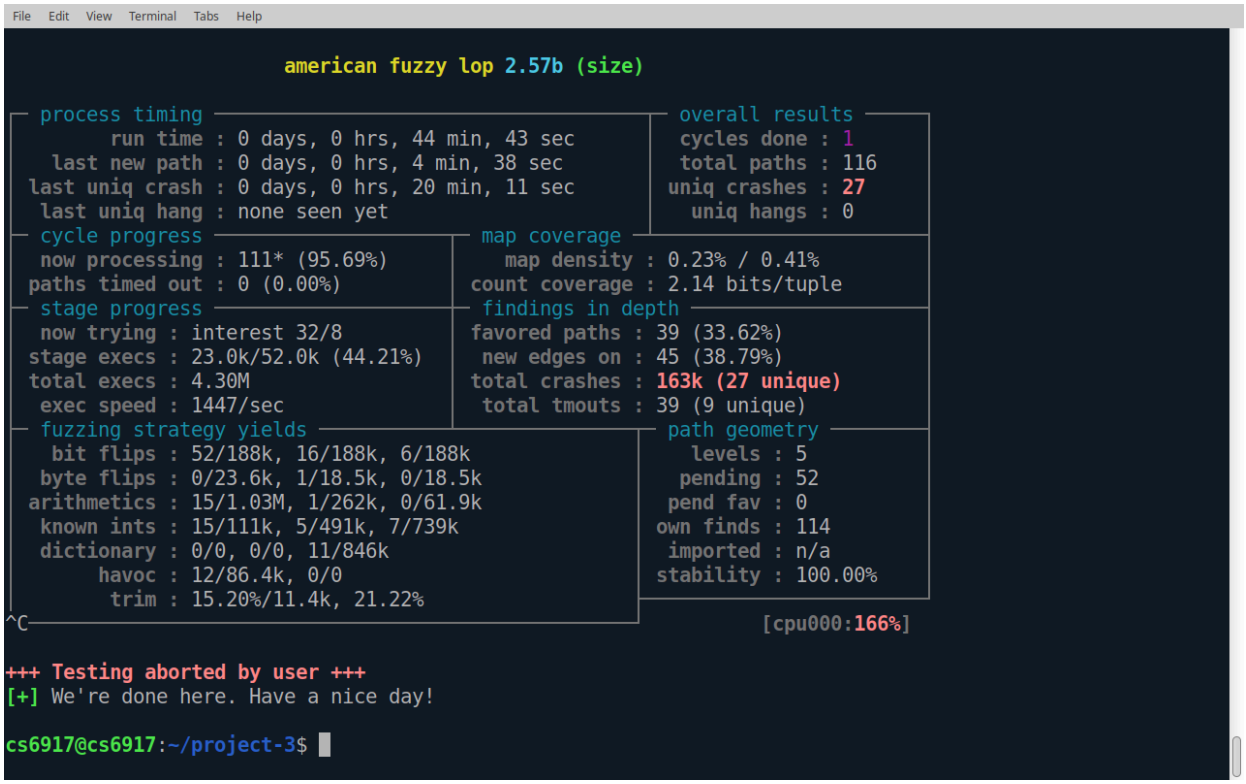
Q2. Why did you have to use *afl-gcc* to compile the source (not *gcc*)?

- When fuzzing using AFL, we had to use afl-gcc instead of the standard gcc compiler because AFL requires instrumentation of the target program (size.c in our case) to effectively monitor its execution paths, crashes, and hangs during the fuzzing process.

- This instrumentation allows AFL to generate inputs that explore new execution paths, improving the coverage and effectiveness of the fuzzing effort. In other words, instrumentation is essential for enabling the sophisticated feedback-driven fuzzing that makes AFL effective.

Q3. How many crashes in total did AFL produce? How many unique crashes?

- AFL produced 116 total crashes
- AFL produced 27 unique crashes
- See screenshot below



Q4. Why are hangs counted as bugs in AFL? Which type of attack can they be used for?

- AFL counts hangs as bugs because they indicate potential problems in the software that could affect its reliability, performance, and security. Such potential problems could lead to security vulnerabilities and attacks such as
    - Denial of service Attacks

        - If a hang is triggered in a service or application (especially those exposed to the internet) by an attacker, they can make it unresponsive, denying legitimate users access to the service, and potentially leading to significant disruptions.
    - Resource exhaustion
        - Hangs often result from infinite loops or excessive consumption of memory resources. An attacker could exploit such a vulnerability and deplete system resources, resulting in slowdowns or crashes, and affecting not just the targeted application but also other applications and processes running on the same system.
    - An indicator of deeper bugs
        - Sometimes, a hang can come up because of underlying issues in the code that might be exploitable in other, more severe ways by attackers.
- While hangs might not directly lead to unauthorized access or data leakage, they pose a significant security concern that can be exploited to attack a software application or system.

Q5. Which interface of *libpngparser* remains untested by AFL (take a look at *pngparser.h*)?

- The following function remains untested by AFL
    - int store_png(const char *filename, struct image *img, struct pixel *palette, uint8_t palette_length);
- size.c calls only on load_png but never calls on store_png and that's why it remains untested by AFL.

Bug Reports

Bug Report #1

You need to report each bug you found in your write-up, and bug descriptions should follow this format.

1. Name:
    a. Segmentation fault (SIGSEGV)
        i. A pointer double free (we're freeing a pointer that has already been freed before)
2. Description:
    a. We finished looping and we didn't find iend_chunk, so we went to error. However, chunk_data has already been freed before.
3. Affected lines:
    a. In pngparser.c:711 (in original program file, ie before editing any lines)
4. Expected vs observed:
    a. The program here ends looping over all the pixels in the image but never finds an IEND png chunk and therefore it should handle this error and exit cleanly. However, while trying to clean up the pointers before exiting with 1, it frees the current_chunk->chunk_data twice resulting in a double free and a segmentation fault.
5. Step to reproduce:
    a. No preconditions are required here to reproduce the bug, just run the command in the next bullet point below.
6. Command:
    a. ./size afl_out/crashes/id:000000,sig:11,src:000000,op:flip1,pos:8
7. Proof-of-Concept (PoC) input (generated by AFL in *afl_out/crashes*)
    a. id:000000,sig:11,src:000000,op:flip1,pos:8
8. Suggested fix description:
    a. After we free any chunk_data, we have to make sure to set the chunk_data pointer to NULL as well. When we free a pointer, it will still be pointing to a valid memory address although this memory address has already been deallocated. Therefore, when we use an if statement like if(chunk_data), exactly like the one pngparser.c:712, the statement would return yes despite that chunk_data has already been before. It returns true because the pointer still points to a valid memory location, and thus, when it tries to free it again, we get a segmentation fault. The way to fix this is that after I free any

chunk_data pointer in the program, I make sure to set it to NULL as well. The lines that I added chunk_data = NULL to were the following
    i. pngparser.c:276
    ii. pngparser.c:660
    iii. pngparser.c:694
    iv. pngparser.c:716
b. NOTE that line numbers differ from the original file as I keep adding new lines to the program.

Bug Report #2

You need to report each bug you found in your write-up, and bug descriptions should follow this format.

1. Name:
    a. Segmentation fault (SIGSEGV)
        i. Array out of bound
2. Description:
    a. width and height variables in the convert_rgb_alpha_to_image() function are of type uint32_t, and therefore, they are of size 32 bytes. However, img->size_x and img->size_y are of type uint16_t, and therefore, they are of size 16 bytes. In the function convert_rgb_alpha_to_image() we assign img->size_x the value of width and img->size_y the value of height. However, these two variables are of different types and this can result in undefined behaviors by assigning unexpected values to img->size_x and img->size_y. In particular, in our case, this causes the different arrays being populated in the for loop to go out of bound because the values width and height can be much bigger than img->size_x and img->size_y.
3. Affected lines:
    a. In pngparser.c:462 to 465
        i. ie the two for nested loops in the function convert_rgb_alpha_to_image()

    ii. NOTE that line numbers differ from the original file as I keep adding new lines to the program.
4. Expected vs observed:
    a. we expect that the nested for-loops go over all the pixels in the image by iterating over every row, and every pixel in that row, starting from index 0. However, the for loop conditions are set to have the counters be strictly less than the width and height variables. However, I just discussed above that the values of width and height are not equal to the values of img->size_x and img->size_y because of the different variable types, and thus, we will reach a point when we try to access an array entry in img->px that goes out of bound, resulting in a segmentation fault.
5. Step to reproduce:
    a. I assume here that you have already fixed the bug from the first report and followed the fix suggestions I mentioned there. Then, to reproduce this bug, just run the command mentioned below.
6. Command:
    a. ./size afl_out/crashes/id:000016,sig:11,src:000001,op:flip1,pos:16
7. Proof-of-Concept (PoC) input (generated by AFL in *afl_out/crashes*)
    a. id:000016,sig:11,src:000001,op:flip1,pos:16
8. Suggested fix description:
    a. In the convert_rgb_alpha_to_image() function change the following
        i. for (uint32_t idy = 0; idy < height; idy++) to for (uint32_t idy = 0; idy < img->size_y; idy++)
        ii. if (inflated_buf[idy * (1 + 4 * width)]) to if (inflated_buf[idy * (1 + 4 * img->size_x)])
        iii. for (uint32_t idx = 0; idx < width; idx++) to for (uint32_t idx = 0; idx < img->size_x; idx++)
        iv. pixel_idx = idy * (1 + 4 * width) + 1 + 4 * idx to pixel_idx = idy * (1 + 4 * img->size_x) + 1 + 4 * idx

Bug Report #3

You need to report each bug you found in your write-up, and bug descriptions should follow this format.

1. Name:
    a. Segmentation fault (SIGSEGV)
        i. Array out of bound
2. Description:
    a. width and height variables in the convert_color_palette_to_image() function are of type uint32_t, and therefore, they are of size 32 bytes. However, img->size_x and img->size_y are of type uint16_t, and therefore, they are of size 16 bytes. In the function convert_color_palette_to_image() we assign img->size_x the value of width and img->size_y the value of height. However, these two variables are of different types and this can result in undefined behaviors by assigning unexpected values to img->size_x and img->size_y. In particular, in our case, this causes the different arrays being populated in the for loop to go out of bound because the values width and height can be much bigger than img->size_x and img->size_y.
3. Affected lines:
    a. In pngparser.c:412 to 415
        i. ie the two for nested loops in the function convert_color_palette_to_image()
        ii. NOTE that line numbers differ from the original file as I keep adding new lines to the program.
4. Expected vs observed:
    a. we expect that the nested for-loops go over all the pixels in the image by iterating over every row, and every pixel in that row, starting from index 0. However, the for loop conditions are set to have the counters be strictly less than the width and height variables. However, I just discussed above that the values of width and height are not equal to the values of img->size_x and img->size_y because of the different variable types, and thus, we

will reach a point when we try to access an array entry in img->px that goes out of bound, resulting in a segmentation fault.

5. Step to reproduce:
    a. I assume here that you have already fixed the bugs from the previous reports and followed the fix suggestions I mentioned there. Then, to reproduce this bug, just run the command mentioned below.
6. Command:
    a. ./size afl_out/crashes/id:000002,sig:11,src:000000,op:flip1,pos:16
7. Proof-of-Concept (PoC) input (generated by AFL in *afl_out/crashes*)
    a. id:000002,sig:11,src:000000,op:flip1,pos:16
8. Suggested fix description:
    a. In the convert_rgb_alpha_to_image() function change the following
        i. for (uint32_t idy = 0; idy < height; idy++) to for (uint32_t idy = 0; idy < img->size_y; idy++)
        ii. if (inflated_buf[idy * (1 + width)]) to if (inflated_buf[idy * (1 + img->size_x)])
        iii. for (uint32_t idx = 0; idx < width; idx++) to for (uint32_t idx = 0; idx < img->size_x; idx++)
        iv. palette_idx = inflated_buf[idy * (1 + width) + idx + 1] to palette_idx = inflated_buf[idy * (1 + img->size_x) + idx + 1]

Bug Report #4

You need to report each bug you found in your write-up, and bug descriptions should follow this format.

9. Name:
    a. Segmentation fault (SIGSEGV)
        i. Heap use after free
10. Description:
    a. In the convert_rgb_alpha_to_image() function, the error handling part doesn't return NULL or -1 after freeing all the pointers we have passed to the function, resulting in a heap use after free after program counter leaves the function without realizing that an error

has been encountered and all the pointers in the function have already been freed.

11. Affected lines:
    a. In pngparser.c:482
        i. ie the error part of the convert_rgb_alpha_to_image() function
        ii. NOTE that line numbers differ from the original file as I keep adding new lines to the program.
12. Expected vs observed:
    a. We expect that the function convert_rgb_alpha_to_image() returns a NULL or -1 when encountering an error, but it doesn't, resulting in a heap use after free because the pointers in the function have already been freed but the program doesn't yet realize that we have encountered an error.
13. Step to reproduce:
    a. I assume here that you have already fixed the bugs from the previous reports and followed the fix suggestions I mentioned there. Then, to reproduce this bug, just run the command mentioned below.
14. Command:
    a. ./size afl_out/crashes/id:000020,sig:11,src:000001,op:int8,pos:19,val:+0
15. Proof-of-Concept (PoC) input (generated by AFL in *afl_out/crashes*)
    a. id:000020,sig:11,src:000001,op:int8,pos:19,val:+0
16. Suggested fix description:
    a. In the error section of the convert_rgb_alpha_to_image() function do the following
        i. if (img) {
        ii. if (img->px) {
        iii. free(img->px);
        iv. img->px = NULL;
        v. }
        vi. free(img);
        vii. img = NULL;

viii. }
ix. return NULL;

Bug Report #5

You need to report each bug you found in your write-up, and bug descriptions should follow this format.

17. Name:
    a. Segmentation fault (SIGSEGV)
        i. Trying to access a null pointer
18. Description:
    a. In the convert_color_palette_to_image(), the function doesn't check if the plte_chunk passed as a parameter is NULL or not, but rather, try to access its chunk_data in the following few lines, resulting in a segmentation fault when plte_chunk is NULL.
19. Affected lines:
    a. In pngparser.c:398
        i. ie the error part of the convert_rgb_alpha_to_image() function
        ii. NOTE that line numbers differ from the original file as I keep adding new lines to the program.
20. Expected vs observed:
    a. We expect that when trying to construct plte_entries we'll be able to do that normally because plte_chunk is a valid pointer with valid values, however, when plte_chunk was NULL, it resulted in a segmentation fault.
21. Step to reproduce:
    a. I assume here that you have already fixed the bugs from the previous reports and followed the fix suggestions I mentioned there. Then, to reproduce this bug, just run the command mentioned below.
22. Command:
    a. ./size afl_out/crashes/id:000015,sig:11,src:000000,op:ext_AO,pos:37
23. Proof-of-Concept (PoC) input (generated by AFL in *afl_out/crashes*)

a. id:000020,sig:11,src:000001,op:int8,pos:19,val:+0

24. Suggested fix description:
    a. Add the following if statement before struct plte_entry *plte_entries = (struct plte_entry *)plte_chunk->chunk_data; in the convert_rgb_alpha_to_image() function:
    b. if (!plte_chunk)
    c. return NULL;

Submission

You are required to submit **a zip file** including the following files:

- **a typed written report (a PDF file)**
    - including all the bug reports and the answers to the questions above
- ***PoC input files***
- ***pngparser.c*** - *with your fixes and is_png_chunk_valid patched to return 1*
- ***any files that you changed in the "src" directory (all these should be compiled properly.)***

Ensure that the written portion of the project is a PDF (no doc or docx is allowed). Please submit the zip file to Canvas by the due time. (filename: proj3_[NetID]_[firstname].zip e.g., proj3_f1234xx_john.zip)