Practic 1

```
/*This is the sample program to notify us for the file creation and file
deletion takes place in “/tmp” directory*/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/inotify.h>
#include <unistd.h>
#include <string.h>

#define EVENT_SIZE (sizeof(struct inotify_event))
#define EVENT_BUF_LEN (1024 * (EVENT_SIZE + 16))
#define MAX_NUMBER_FILES 1024

int find_in_list(char* filename, char** list_of_files, int size_of_list)
{
 for (unsigned int i=0; i<size_of_list; ++i)
 {
   if (strcmp(filename, list_of_files[i]) == 0)
   {
     return i;
   }
 }
 return -1;
}

int main()
{
 int length, i = 0;
 int fd;
 int wd;
 int wd2;
 char buffer[EVENT_BUF_LEN];

 int no_files = 0;
```

```
 char* filenames[MAX_NUMBER_FILES];
 int no_accesses[MAX_NUMBER_FILES];
 int pos;

 /*creating the INOTIFY instance*/
 fd = inotify_init();

 /*checking for error*/
 if (fd < 0)
 {
   perror("inotify_init");
 }

 /*adding the “/tmp” directory into watch list. Here, the suggestion is to
validate the existence of the directory before adding into monitoring
list.*/
 // wd2 = inotify_add_watch(fd, "/tmp", IN_CREATE | IN_DELETE |
IN_ACCESS);
 // wd2 = inotify_add_watch(fd, "/home", IN_CREATE | IN_DELETE |
IN_ACCESS);
 wd2 = inotify_add_watch(fd, "/home/lab3", IN_CREATE | IN_DELETE |
IN_ACCESS);

 /*read to determine the event change happens on “/tmp” directory.
Actually this read blocks until the change event occurs*/
 do
 {
   i = 0;
   length = read(fd, buffer, EVENT_BUF_LEN);

   /*checking for error*/
   if (length < 0)
   {
     perror("read");
   }
```

```
   /*actually read return the list of change events happens. Here, read
the change event one by one and process it accordingly.*/
   while (i < length)
   {
     struct inotify_event *event = (struct inotify_event *)&buffer[i];
     if (event->len)
     {
       if (event->mask & IN_CREATE)
       {
         if (event->mask & IN_ISDIR)
         {
           printf("New directory %s created.\n", event->name);
         }
         else
         {
           printf("New file %s created.\n", event->name);
         }
       }
       else if (event->mask & IN_DELETE)
       {
         if (event->mask & IN_ISDIR)
         {
           printf("Directory %s deleted.\n", event->name);
         }
         else
         {
           printf("File %s deleted.\n", event->name);
         }
       }
       else
       {
         if (event->mask & IN_ACCESS)
         {
           pos = find_in_list(event->name, filenames, no_files);

           if(pos == -1) {
```

```
            filenames[no_files] = (char*) malloc(strlen(event->name) +
1);
            strcpy(filenames[no_files], event->name);
            no_accesses[no_files] = 1;
            no_files += 1;
          }
          else {
            no_accesses[pos] += 1;
          }


          // printf("Accessed %s.\n", event->name);
          for (unsigned int i=0; i<no_files; ++i) {
            printf("%s: %d\n", filenames[i], no_accesses[i]);
          }
          printf("\n\n");
        }
      }
    }
    i += EVENT_SIZE + event->len;
  }
 } while (1);
 /*removing the "/tmp" directory from the watch list.*/
 inotify_rm_watch(fd, wd);
 inotify_rm_watch(fd, wd2);

 /*closing the INOTIFY instance*/
 close(fd);
}
```

With the code from above, I am able to successfully count the number of accesses for files from /home/lab3 folder, but when I access a file from folder lab3-1, it registers an access to the folder lab3-1 and not to the accessed file from it. This happens because the inotify only supervises the top level of the specified folder, and not from the subfolders.

Practic 2

I saw that the file eicar.txt was detected as malicious by 61 out 67 engines, while the file sha-256.c was undetected by all of them.

Practic 3

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/inotify.h>
#include <unistd.h>
#include <string.h>
#include <curl/curl.h>

#include "sha-256.h"

#define EVENT_SIZE (sizeof(struct inotify_event))
#define EVENT_BUF_LEN (1024 * (EVENT_SIZE + 16))
#define MAX_NUMBER_FILES 1024
#define MAXCHAR 10000

int find_in_list(char* filename, char** list_of_files, int size_of_list)
{
 for (unsigned int i=0; i<size_of_list; ++i)
 {
   if (strcmp(filename, list_of_files[i]) == 0)
   {
     return i;
   }
 }
 return -1;
}

size_t write_callback(char *ptr, size_t size, size_t nmemb, void
*userdata)
{

 return fwrite(ptr, size, nmemb, (FILE *) userdata);
```

```
 }

int main()
{
 int length, i = 0;
 int fd;
 int wd;
 int wd2;
 char buffer[EVENT_BUF_LEN];


 int no_files = 0;
 char* filenames[MAX_NUMBER_FILES];
 int no_accesses[MAX_NUMBER_FILES];
 int pos;
 int is_user_produced;
 char working_directory[] = "/path/to/the/supervised/folder";

 const char* base_url =
"https://www.virustotal.com/vtapi/v2/file/report?apikey=";
 const char* api_key = “replace this with the personal apikey”;
 const char* resourse_string = "&resource=";
 const int hash_length = 64;
 char* url;

 CURL *curl;
 CURLcode res;
 FILE * f = NULL;
 char response_buffer[MAXCHAR];

 /*creating the INOTIFY instance*/
 fd = inotify_init();

 /*checking for error*/
 if (fd < 0)
 {
   perror("inotify_init");
```

```
 }

 /*adding the directory into watch list.*/
 wd = inotify_add_watch(fd, working_directory, IN_CREATE | IN_ACCESS);

 /*read to determine the event change happens on the directory. Actually
this read blocks until the change event occurs*/
 do
 {
   i = 0;
   length = read(fd, buffer, EVENT_BUF_LEN);

   /*checking for error*/
   if (length < 0)
   {
     perror("read");
   }

   /*actually read return the list of change events happens. Here, read
the change event one by one and process it accordingly.*/
   while (i < length)
   {
     struct inotify_event *event = (struct inotify_event *)&buffer[i];
     if (event->len) {
       is_user_produced = 1;
       if (event->mask & IN_ACCESS) {
           // For each event we will create an access event (when read
from the file to compute the hash), which will not be considered
           pos = find_in_list(event->name, filenames, no_files);

           if(pos != -1 && no_accesses[pos] > 0) {
               no_accesses[pos] -= 1;
               is_user_produced = 0;
           }
       }
       if(is_user_produced) {
           // Determine the absolute path of the file
```

```
            char * filename = (char*) malloc(strlen(working_directory) +
strlen(event->name) + 2);
            strcpy(filename, working_directory);
            strcat(filename, "/");
            strcat(filename, event->name);

            // Open the file
            FILE * f = NULL;
            unsigned int i = 0;
            unsigned int j = 0;
            char buf[4096];
            uint8_t sha256sum[32];
            if( ! ( f = fopen( filename, "rb" ) ) )
                    {
                        perror( "fopen" );
                        return( 1 );
                    }

            // Mark that the files is being processed.
            // Another event will be triggered for this file when reading
from it to compute its hash.
            // If this step is not made, then the reading made by the
program will also be process throwing the program into a loop.
            pos = find_in_list(event->name, filenames, no_files);

            if(pos == -1) {
              filenames[no_files] = (char*) malloc(strlen(event->name) +
1);
              strcpy(filenames[no_files], event->name);
              no_accesses[no_files] = 1;
              no_files += 1;
            }
            else {
              no_accesses[pos] += 1;
            }
```

```
            // Read from the file and compute the hash function on the
content
            sha256_context ctx;
            sha256_starts( &ctx );

            while( ( i = fread( buf, 1, sizeof( buf ), f ) ) > 0 )
            {
                sha256_update( &ctx, buf, i );
            }

            sha256_finish( &ctx, sha256sum );

            fclose(f);

            // Generate the url to the virus API
            url = (char*) malloc(strlen(base_url) + strlen(api_key) +
strlen(resourse_string) + hash_length + 1);
            strcpy(url, base_url);
            strcat(url, api_key);
            strcat(url, resourse_string);

            for( j = 0; j < 32; j++ )
            {
              sprintf(url, "%s%02x", url, sha256sum[j] );
            }

            // Sent the request to the virus total API
            curl = curl_easy_init();
            if(curl) {
              curl_easy_setopt(curl, CURLOPT_URL, url);

              f = fopen("tmp.txt", "w+");
              curl_easy_setopt(curl, CURLOPT_WRITEDATA, f);
              curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,
write_callback);
              /* Perform the request, res will get the return code */
              res = curl_easy_perform(curl);
```

```
            /* Check for errors */
            if(res != CURLE_OK)
              fprintf(stderr, "curl_easy_perform() failed: %s\n",
                      curl_easy_strerror(res));

            fclose(f);

            /* always cleanup */
            curl_easy_cleanup(curl);
          }

          // Read the output from the API and search it for the number of
positive detections.
          // If this number is bigger than 0, then the file will be
removed.
          if( ! ( f = fopen( "tmp.txt", "r" ) ) )
            {
                perror( "fopen" );
                return( 1 );
            }
          while (fgets(response_buffer, MAXCHAR-1, f) != NULL) {
            int no_positives;
            char* positive_ptr = strstr(response_buffer, "positives");
            if(positive_ptr != NULL) {
              positive_ptr = positive_ptr + strlen("positives\": ");
              positive_ptr = strtok(positive_ptr, ",");
              no_positives = atoi(positive_ptr);
              if(no_positives > 0) {
                if (remove(filename) == 0)
                  printf("Deleted successfully the file %s.\n",
filename);
                else
                  printf("Unable to delete the file %s.\n", filename);
              }
            }
          }
          fclose(f);
```

```
            free(filename);
            free(url);
        }
      }
      i += EVENT_SIZE + event->len;
    }
  } while (1);
  /*removing the directory from the watch list.*/
  inotify_rm_watch(fd, wd);

  /*closing the INOTIFY instance*/
  close(fd);
}
```