DV 1667: Project task #1

Antivirus in Python

Jianguo Ding, Dure Adan Ammara

Department of Computer Science
Blekinge Institute of Technology

2024-01-18

Introduction

The project task in the course **DV1667** is to write your own simplified antivirus program in Python. There is an existing virus database that your program must use. This database contains descriptions of files that are classified as viruses. Your task is to write a program that traverses (goes through) a directory in the file system and all its sub-directories, and its sub-directories, etc. For each file the program finds among these directories, it must check if it matches any of the descriptions in the virus database. Every time the program encounters a file that is included in the virus database, **this must be reported to a log file**.

To make the task more manageable, we break the task down into smaller and more manageable parts ("divide and conquer"). By solving each part separately, the work will hopefully go easier.

The project can be broken down into the following three parts: filter traversal, the virus database, and file identification. Furthermore, you must also submit a written requirements specification together with your program, where you go through the requirements that you worked towards.

Grouping

All parts of this specification must be met. Furthermore, the task must be solved in groups of two (2) students. The students themselves are responsible for coordinating the group division. Note that both members of the group must be able to explain any code written by the group. If someone in the group cannot do this, this person will be tested individually.

System specification and examination

The examination will be performed under Ubuntu 22.04.31[1]. During the examination, your antivirus program will be tested by running it on a test directory where a predetermined number of files must be identified as viruses. It is fine to execute Ubuntu in a virtual environment in your usual operating system (e.g. VirtualBox2)[2].

Part 1 - Filter traversal

To implement the first part of filter traversal, your program should have the capability to accept a starting directory as a command-line argument. Once provided with the starting directory, the program should employ a recursive approach to navigate through all the files and subdirectories within it. This recursive functionality involves the use of a directory traversal function that calls itself for each subdirectory encountered.

In practical terms, the recursive function starts at the specified initial directory, listing all the files and subdirectories within it. For each subdirectory found during this initial scan, the function then calls itself, repeating the process of listing files and subdirectories. This recursive behaviour continues, effectively traversing through multiple levels of subdirectories until all paths have been explored.

By structuring the directory traversal in this recursive manner, the program can efficiently navigate complex directory structures without the need for extensive nested loops or iterative constructs. This recursive design simplifies the code and enhances its readability, making it a common and effective solution for handling file listings within directory hierarchies.

Part 2 – Virus database

The descriptions of viruses are stored in the database. In our case, it's not really a database, just a plain text file, named **signatures.db**. Each line in the text file contains a virus definition. So, you can read character by character on each line until you come across a line end '**\n**' or you use a function to read files line by line. Each virus definition contains two values separated by a sign of '**=**'. The first value is a text string containing the name of the virus. This text string must not exceed 32 characters, including NULL-termination. The name is followed by an '**=**' and then a description of the contents of the virus file, see the example below.

---

[1] URL: https://ubuntu.com/download/desktop

[2] URL: https://www.virtualbox.org

TestVirus.D=255044462d312e340a332030206f62

The second value, i.e. the virus description, has no maximum length limit. Furthermore, the description is stored in hexadecimal notation. This means that each byte in the virus file (one byte) is described with two characters in the virus definition. See example for **TestVirus.D** in the table below. When the program starts, it can assume that the signature database is in the same directory that the program was previously saved in. Note that it is not allowed to hard-code the path to the signature database in the program's source code.

Byte#	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Hex	0x25	0x50	0x44	0x46	0x2d	0x31	0x2e	0x34
Dec	37	80	68	70	45	49	46	52
ASCII	'%'	'P'	'D'	'F'	'-'	'1'	'.'	'4'

The virus description always starts from byte 0 (zero) in the files that are scanned (i.e. the potential virus files). According to the table above, we see that the first byte in the virus description is a text string that contains "%PDF-1.4". We can also conclude that the number of bytes described in the virus description is the length of the description divided by two. So if we have a virus description that is 50 characters long, it describes the first 50 / 2 = 25 bytes in the virus file.

Part 3 – File identification

For each file you encounter during the filter step, you must compare it against the virus definitions in the virus database. Note that you always start by comparing the first byte in the file against the first byte in the virus description. If the file matches one of the virus definitions, you should log this in a log file named **dv1667.log**. In the log file there must be information about which file is infected, as well as the path to the file and the name of the virus that is assumed to have infected the file.

Error Handling

Note that you cannot be sure that the virus database is really in the same directory as the program when it starts. Furthermore, you cannot, for example, be sure that the virus database does not contain syntax errors during the examination. You must therefore include detailed error handling in your program.

Requirements specification

You must document the requirements you used when you developed your AV program. Document the requirements in a report where you briefly explain how you worked with the requirements and include a table of your requirements. The table should include the following columns:

1. ID number
2. Name
3. Brief description
4. How you tested the requirement
5. Dependency on other requirements (list is if the ID number of the requirements)
6. Whether the requirement Shall or Should be implemented (TBI – to be implemented)

Below is an example of such a table including a requirement.

ID	Name	Description	Dependency	Test	TBI
E.1	Printing	Program should print documented results			Yes

Submission

You can find the deadline for the assignment on Canvas. Before you send in your submission data you should check that:

- you meet all the requirements in this specification,
- your project executes correctly under a normal Ubuntu 22.04.3, your program executes when you are in the same directory as the program,
- you have attached your requirements specification as a PDF file, and you have entered your name and email address in all the emails you submit, i.e. your codes,
- you have packed all your submissions as a **tar.gz or zip** archive.

GOOD LUCK!