**Part 1**

So the thing is that all the CRC Checks do return an error, but for some reason at byte 24, the coder\_status in ClamAV is toggled from CODER\_STATUS\_NOT\_SPECIFIED to CODER\_STATUS\_FINISHED\_WITH\_MARK, and even if the XzDec.c file (decoding file) does return an error (which it does due to the CRC checks), the main file prioritizes the fact that the status of the decoder is set to CODER\_STATUS\_FINISHED\_WITH\_MARK.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

This never changes once it's set at byte 24, and thus forcefully returns a XZ\_STREAM\_END to CUSTOM\_xz\_parse, which it interprets as there being no problem and returns the default return code of 0. However, this does not make sense to me as there would be no need for the program to continue running if it knew that it’s status was finished, especially considering that it can correctly identify that parts of the data it is decoding is corrupted, but still returns the default return code 0.

A screen shot of a computer program

Description automatically generated

I can see 3 potential possibilities with this:

(1) The authors of ClamAV may have made a mistake and didn’t switch the order of the following two checks, which may have cause a misorder of priority (I would think that an error should be the first thing to be checked and should always take precedence)

A screenshot of a computer program

Description automatically generated

(2) Something quite interesting is that there is a similar equivalent return message for XZ\_STREAM\_END (ClamAV return message) in XZ Utils, which is named LZMA\_STREAM\_END. However, when XZ returns LZMA\_STREAM\_END in its decoding function called stream\_decoder.c (equivalent to XzDec.c), it actually will generate an error message, so it is possible that CUSTOM\_XZ\_parse was not well implemented, as correct me if I’m wrong but CUSTOM\_XZ\_parse is newly added after patching in the docker container, and not from the original source code? (Very Possible that I’m wrong)

For XZ:



A screen shot of a computer screen

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen with text and symbols

Description automatically generated

For ClamAV:

A screen shot of a computer program

Description automatically generated

(3) Everything is correctly implemented and there is an actual good reason why ClamAV decides to keep scanning even if its status says it’s finished, and there is also a good reason why it decides to not call an error even though it clearly is a CRC error which indicates data corruption.

**Part 2**

So they actually adapted most of the code to write their CUSTOM\_XZ\_parse from a function called cli\_scanxz from my observations, at is the only function from the vanilla scanners.c that uses the cli\_XzDecode function. You can also see the similarities between the two functions:

**A screenshot of a computer program

Description automatically generated**A screen shot of a computer program

Description automatically generated

However, the code snippet in the red box is an additional check that is not implemented in CUSTOM\_xz\_parse that could potentially cause a return error even if rc == XZ\_STREAM\_END instead of XZ\_DATA\_ERROR.

Before continuing, I’ll first assume that nothing important changed in the vanilla XzDec.c (as in it doesn’t modify the value of avail.out which should hopefully still be the same as CLI\_XZ\_OBUF\_SIZE, which is 1024\*1024 per their documentation, and that it still returns the errors that arise from the CRC Checks), which it shouldn’t as I did a quick comparison between the vanilla XzDec.c and the patched XzDec.c. I also checked the two xz\_iface.c files which contain the function cli\_XzDecode (the function that decides to return XZ\_STREAM\_END instead of XZ\_RESULT\_DATA\_ERROR from the CRC Check), and they were exactly the same.

This means that the in the vanilla ClamAV scanners.c file, rc would still be returned as XZ\_STREAM\_END.

Now the interesting part comes with the cli\_writen function in scanners.c that takes three inputs: *fd, buf and towrite*

We first note that strm->avail.out == 1048576 == 1024\*1024 == CLI\_XZ\_OBUF\_SIZE in the patched scanners.c file in the “artificial” CUSTOM\_XZ\_parse:

A screenshot of a computer

Description automatically generated

This means that the cli\_XzDecode function in xz\_iface.c did not modify it from its initial value that we had set previously in the scanner.c file (this also applies to strm->avail.out in the vanilla scanners.c file as the code structures are essentially the same for the variables that we consider):

A black background with white text

Description automatically generated

Now, we note that *towrite* is:

CLI\_XZ\_OBUF\_SIZE - strm->avail.out,

meaning that towrite == 0.

Looking back up to my initial picture of this thread about the similarities between the two scanners.c functions, we can see that in order to find out whether or not LZMA\_STREAM\_END will return an actual error in the end, we need to check for whether or not cli\_writen(fd, buf, towrite) != towrite)

Case 1\*: cli\_writen(fd, buf, towrite) != 0 🡪 returns an error message; this means that the actual ClamAV library functions as intended as does actually consider failed CRC checks to be an error.

Case 2\*\*:cli\_writen(fd, buf, towrite) == 0 🡪 does not return an error message; this measn that there is probably a misimplementation in the ClamAV library as even though it detects the failed CRC check, it doesn’t consider it as an error even though it clearly is one. (There is a discrepancy.

A screen shot of a computer screen

Description automatically generated



Now when we look at the cli\_writen function, we can see that it is based on the built-in c function *write()*, as that will decide whether or not it will return an error message and -1, which would cause Case 1\*.

A screen shot of a computer program

Description automatically generated



However, remember that we know towrite == 0, and count is simply towrite. todo is also initialized with the value of count, which is 0, and when we use the built-in c function *write()* with its 3rd argument (number of bytes it will process) being 0, *write()* will automatically return 0.

Thus, in the code inside the red box in the picture above, retval == 0 which is not smaller than 0, and cli\_writen will not return -1 with an error message.

Since todo is still 0, it isn’t bigger than 0 and thus we exit the do-while loop. Finally, we return count which still retains a value of 0 to the scanners.c file, indicating that Case 2\*\* is happening. We exit the loop since rc == XZ\_STREAM\_END without ever modifying ret from the initial value of CL\_CLEAN (== 0).

Thus, I believe that the discrepancy actually exists.

**Part 3**

So from what I'm understanding in the code, ClamAV first creates a temporary file, then decompresses the input file, then stores the decompressed data in the temporary file and finally scans the temporary file. However, it only stores the data prior to XZ\_STATE\_BLOCK, which is probably the main contents of the file and corresponds to bytes from 13-24 (I think, could be wrong), out of 52 total bytes. However, this means that ClamAV does not store any of the decompressed data after XZ\_STATE\_BLOCK, namely in cases XZ\_STATE\_BLOCK\_FOOTER, XZ\_STATE\_STREAM\_INDEX, XZ\_STATE\_STREAM\_INDEX\_CRC and XZ\_STATE\_STREAM\_FOOTER, which correspond to bytes 25 and onward up to 52. This is probably why once the data in XZ\_STATE\_BLOCK has finished decompressing and is saved, CODER\_STATUS\_NOT\_SPECIFIED is toggled to CODER\_STATUS\_FINISHED\_WITH\_MARK. Interestingliy enough, CRC checks are only implemented in latter cases I just mentioned, which only happens when scanning bytes 25 and onward. Thus, ClamAV’s developpers maybe intentionally decided that once XZ\_STATE\_BLOCK has finished being decompressed, this holds priority over any potential errors that may occur after the main block (bytes 13-24).

My issues with this are that:

(1) I don’t understand ClamAV’s purpose in continuing to process the bytes once it finishes with XZ\_STATE\_BLOCK if it ultimately doesn't care about what happens after processing XZ\_STATE\_BLOCK, especially considering the fact that in the above cases where byte positions > 24 (XZ\_STATE\_BLOCK\_FOOTER, XZ\_STATE\_STREAM\_INDEX, XZ\_STATE\_STREAM\_INDEX\_CRC and XZ\_STATE\_STREAM\_FOOTER), the only possible error is the CRC Error and there is thus no point in doing anything with these.

(2) I don’t get why it doesn’t scan a fully decompressed file, because from my understanding (googled it so I may very well be wrong), viruses aren’t necessarily only hidden in the main block, so wouldn’t this cause a security issue?

The only possibility that I can think of would be that it scans the different cases one by one and cli\_scanxz only scans for the main block (so there may be other similar functions in the library that scan and decompress XZ\_STATE\_BLOCK\_FOOTER, etc.) but this seems highly unlikely as if this were the case, then ClamAV should definitely return an error found, but I’ve run the newest version of ClamAV (1.1.0) with the CRC files from PathDiff and it returns a “No Virus Detected” instead of either "Virus Detected" or "Error Occured".