Preserving DRUGS: Their Effects On You, an Apple II program by Marshware

Preservation work performed and documented
by Matthew D'Asaro, 4/2022

I found this title on eBay, previously unpreserved in the Apple II community, with an asking price of $200. This seemed high to me, especially once I looked closely at the auction images and realized that someone had applied Scotch tape directly to the media! After pointing this out to the seller and explaining to him that this meant that the data on the disk was likely lost, he agreed to accept a $50 offer and the disk was soon in the mail to me.

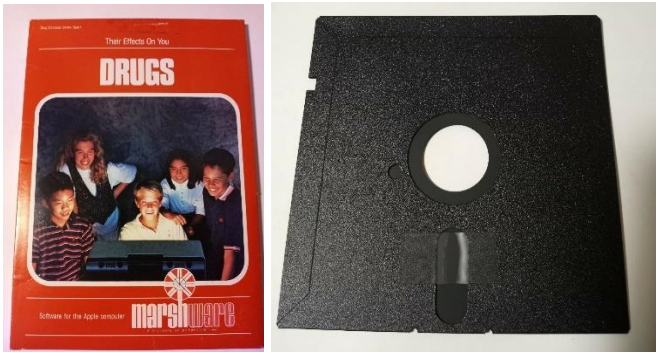

Figure 1: Left: Front cover of software title from eBay auction; Right: Auction photo showing tape on Disk

After I received the software, I inspected it carefully, and indeed, the disk did have scotch tape on the media surface. This tape was very old and well adhered.

Figure 2: Left: Disk as received with tape on media surface; Right: Closeup of tape on media

I picked at the tape a bit (not on the media but on the disk jacket) and confirmed that it was old and hard as a rock. Since I know that isopropyl alcohol is generally safe on disks, and it will (slowly) dissolve tape adhesive, I decided to try that. I liberally applied the alcohol to the tape and then ever-so-gently tried to peel it, a tiny bit at a time. Unfortunately, the result was that the celluloid backing on the tape peeled away and left the adhesive layer on the disk. It was relatively easy to peel this adhesive layer off of the jacket, but it was really stuck to the media surface.

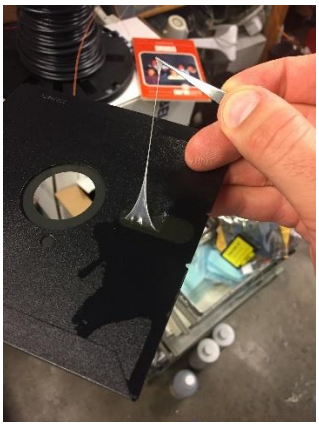
Figure 3: Adhesive removed from jacket but still adhered to media surface

Left with no other choice, I continued to keep the media wet with alcohol and hoped it would dissolve the adhesive. Unfortunately, the adhesive got gooey but did not really dissolve - it was really stuck to the disk surface. Finally, I was forced to use an alcohol wetted Q-Tip to dab very carefully at the adhesive and try to remove it one layer at a time.


Figure 4: Removing stuck adhesive from media surface with isopropyl alcohol and Q-Tip.

Using this technique, most of the adhesive residue was relatively easily removed, but near the edges of the tape, it was rock hard and despite my best efforts, I was not able to remove it all without a little bit of media damage. I feel terrible about this, but am not sure what I should / could have done differently. If you have ever tried to remove 40-year-old scotch tape, that stuff is tenacious. Held up to a light, I can see that a little bit of the oxide layer has come off with the adhesive at the damaged locations.


Figure 5: Media damage from removing adhesive

Despite the damage, I used my applesauce system to do a flux image of the disk three times, twice with my Apple 5.25 drive and once with my Disk II. The very first image taken seems to be the best with additional surface damage occurring on subsequent imaging attempts as I see more damage on Track 12 and 13. This first and best image attempt is in the file called “DRUGS- Their Effects On You - Disk 1, Side A.a2r”. Because of the copy-protection used, the disk analyzer in applesauce was not able to recognize the file system or the checksum format for the sectors, so it was of only limited utility. However, it was able to identify damage on tracks 12 and 13. The rest of this document describes my attempt to recover data from this file.

Before jumping in, a quick note on terminology. Data on a floppy disk cannot be stored in the same format it is stored in memory. This is because on a floppy disk long runs of zeros or long runs of ones can’t be stored directly – think of it a bit like sending data through a transformer or capacitor – DC (constant) voltage can’t pass through. To get around this, Steve Wozniak invented a clever mechanism where a sequence of 8-bit bytes in memory would be written as a sequence of 5-bit "nibbles" on the disk. Later this was improved to be a 6-bit "nibble". In this document “nibble” refers exclusively to this encoded data.

Repairing Track 12

The damage in track 12 was overall much easier to fix than that in track 13 because the data that was damaged (which turned out to be part of an image) contained a lot of repetition, so by searching the entire disk for strings of nibbles identical to those found before and after the damaged area, the correct sequence of fluxes could be identified to fix the damaged area. Figure 6 shows track 12 before any repairs, with the damage highlighted.

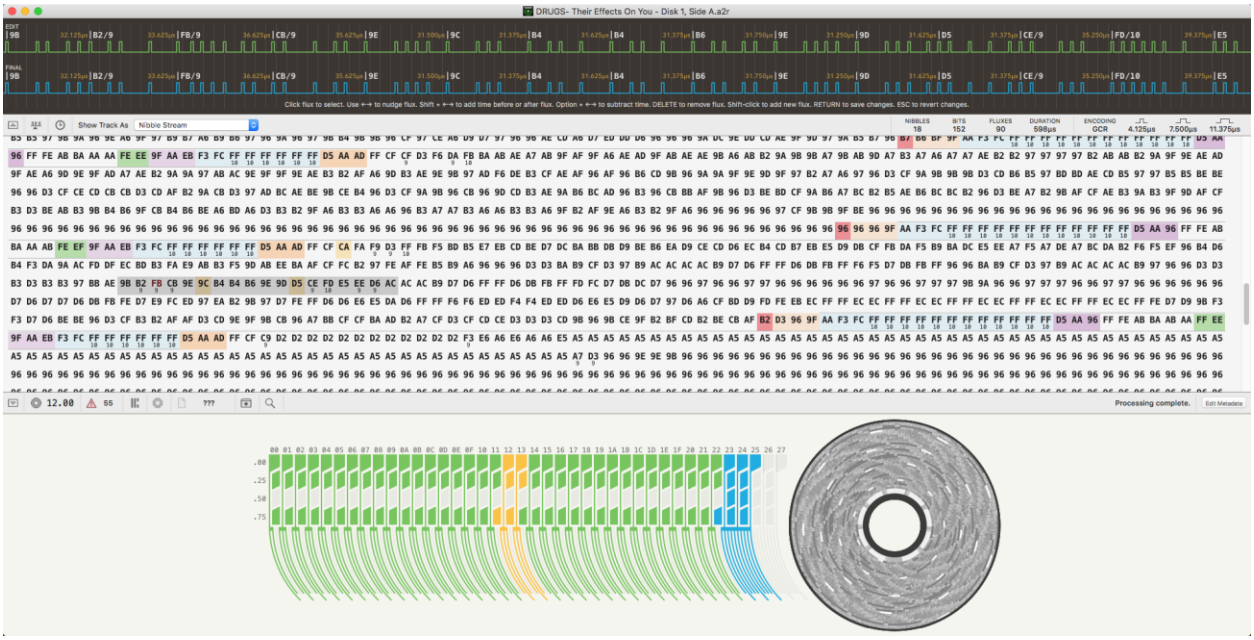

Figure 6: Track 12 in the disk analyzer with the damaged section highlighted.

By searching the entire disk for the stream of nibbles immediately before the damaged area starts (97 BB AE 9B B2), it becomes clear that the next damaged nibble, which was read as FB9,

should most likely actually be a BD nibble as it is in every other occurrence of the pattern before the damage. See Figure 7 below.

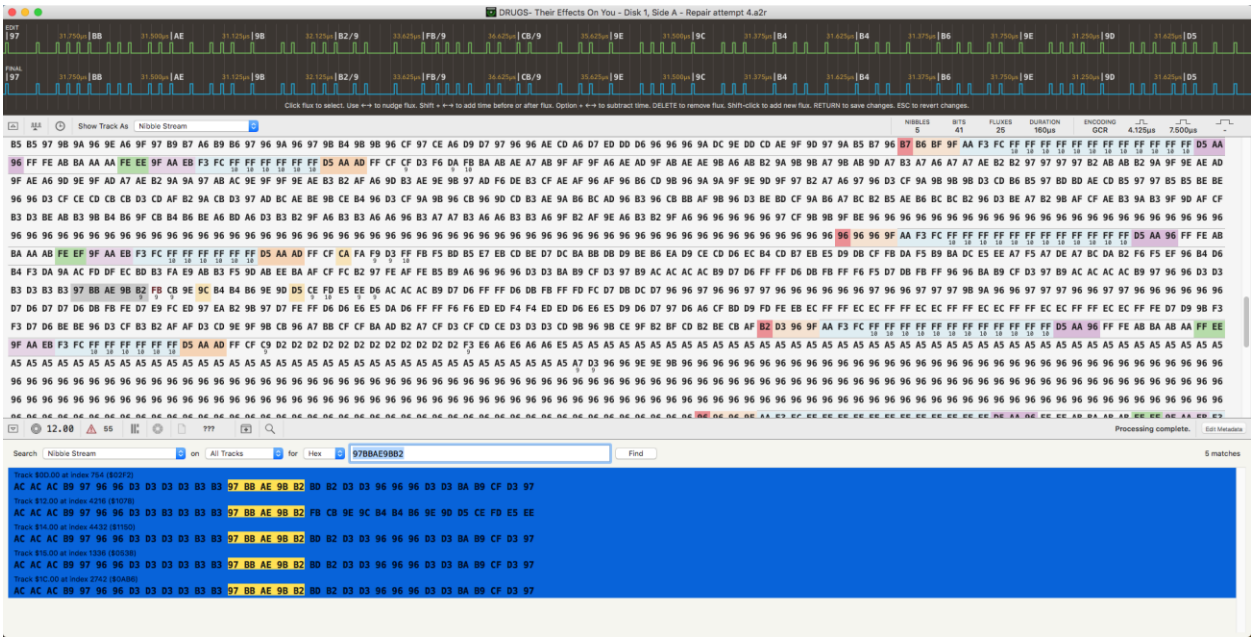

Figure 7: Searching for the stream of nibbles before the damaged area and finding that the first damaged nibble (FB9) should actually be BD.

To change the FB9 nibble to BD, only one flux needs to be moved in the flux editor. When this is done, the nibble stream cleans up a lot and appears as shown in Figure 8 below.

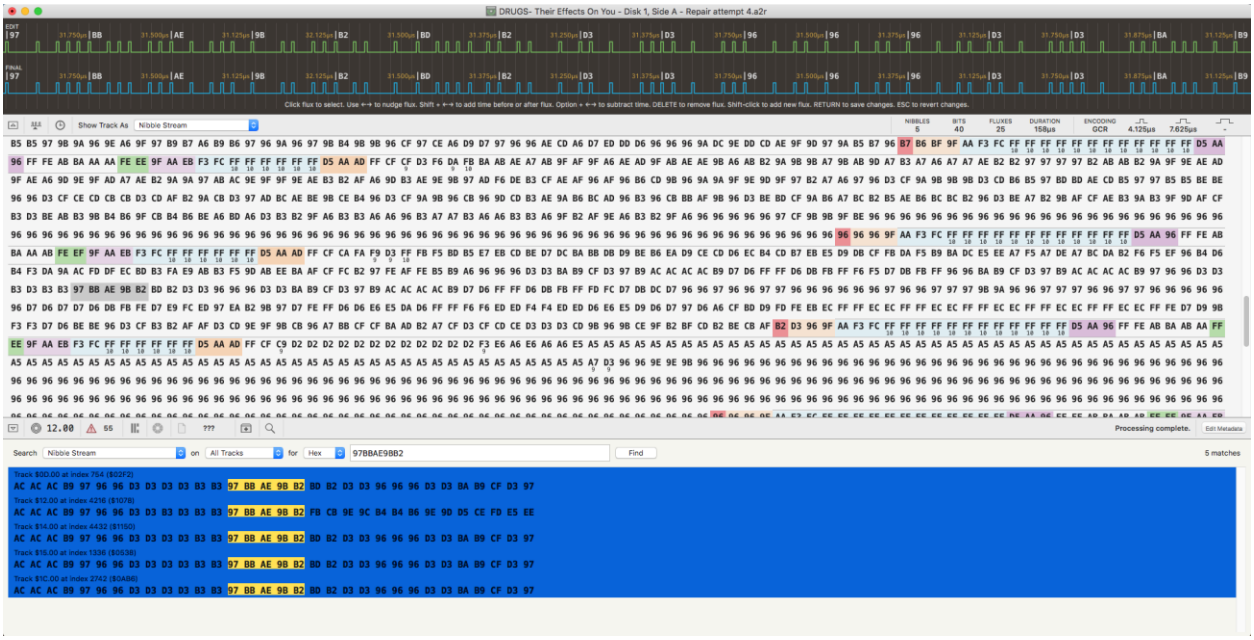

Figure 8: Damaged section on track 12 after a single flux is moved to change the FB9 nibble to be BD.

With this correction made, I initially thought that the track was fixed. However, after getting track 13 done, passport was still reporting an error here. Looking more closely at the search

results shown in Figure 7, there is another error. In the damaged section the third D3 in the sequence 96 D3 D3 D3 D3 B3 has been replaced with a B3. We can be highly confident that this is an error because this sequence is repeated five times in other places on the disk and the damaged section is the only one that doesn't match this pattern. Again, only a single nibble had to be moved to make the correction. With this corrected, the final reconstruction of track 12 is shown below in Figure 9 with the corrected pattern highlighted. Note that other than the repetition, at this point in the project I had no way to confirm that this reconstruction was correct.

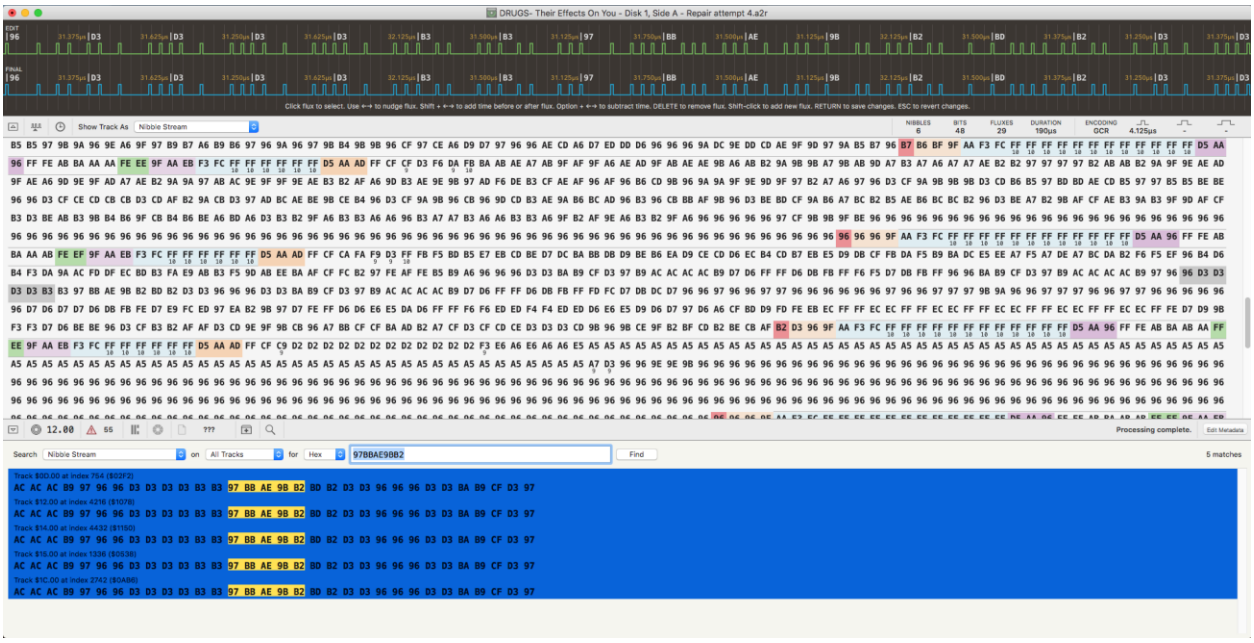

Figure 9: Damaged section on track 12 after a single flux is moved to change the erroneous B3 nibble to be D3. With this correction made, the damaged section is now repaired, although this repair cannot be validated yet.

Repairing Track 13

Track 13 was much harder to repair than track 12 because, first, the damage was a lot more extensive, and second, there was not a lot of repetition of the data around the damaged area from which I could determine the correct reconstruction. Figure 10 below shows track 13 in the disk analyzer with the damaged area highlighted.

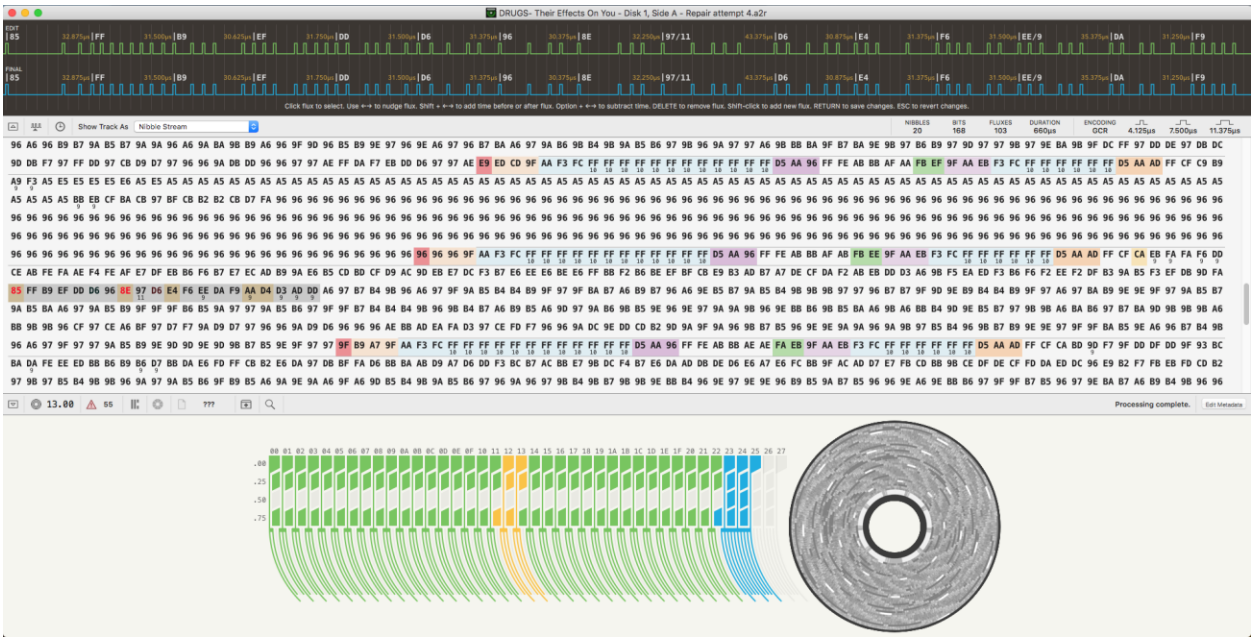

Figure 10: Track 13 in the disk analyzer with the damaged section highlighted.

Searching for the sequence of nibbles immediately before or after the invalid 85 nibble did not yield any other matches elsewhere on the disk. After discovering this, I put aside trying to figure out the damaged “85” nibble and focused on the damage following it. Searching for the sequence EF DD D6 96 yielded five matches elsewhere on the disk and showed that the damaged 8E nibble should most likely be 96 as that is what it is in all the other places on the disk with that same sequence. See Figure 11.

Figure 11: Searching for the stream of nibbles before the damaged “8E” nibble and finding that the damaged nibble should be 96.

To repair this section, I changed the 8E nibble to 96 by moving a single flux, then, again following the search results shown in Figure 11, I changed the 9711 nibble to 97 by moving one

flux and adding one more, and the following D6 nibble to 9B by adding one more flux. With these changes made, all the damage in track 13 except for the unknown “85” nibble appears to be corrected. See Figure 12 and compare with Figure 11.

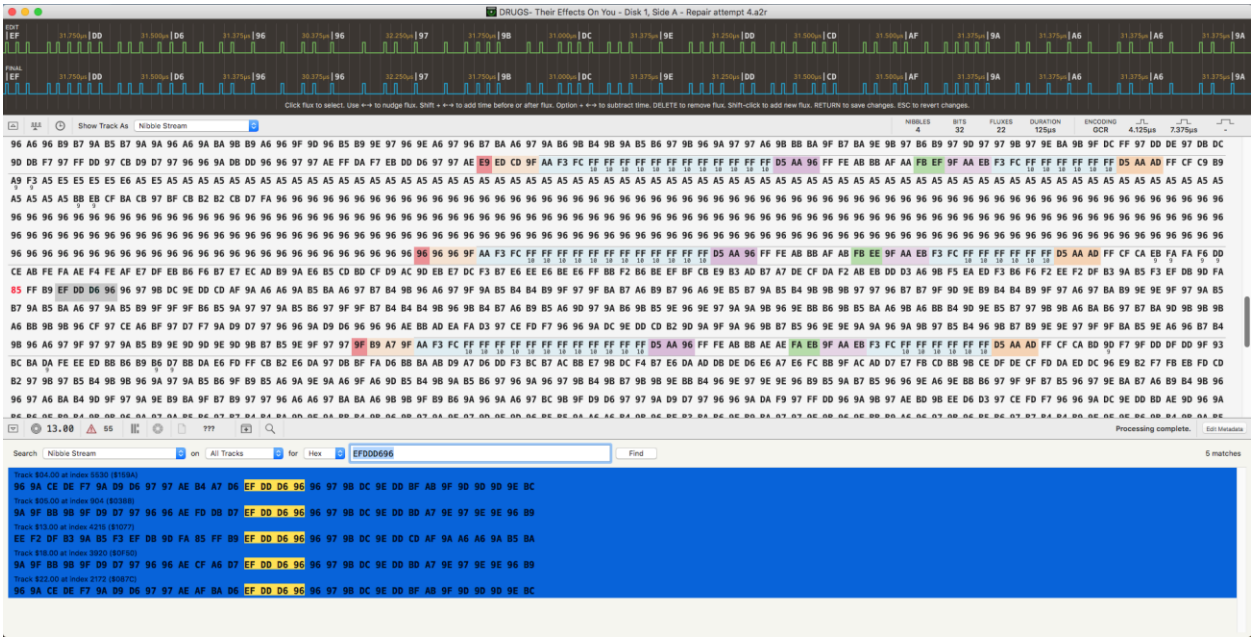

Figure 12: The damaged section of track 13 with all damage other than the damaged “85” nibble apparently corrected.

With these corrections made, the next thing I had to do was figure some way to determine what the correct nibble was to replace the “85”. With no other copies of that sequence of nibbles found elsewhere on the disk, I had to get creative for this one. The biggest problem with understanding what kind of data was damaged in this section was that the disk is copy-protected so that applesauce can’t read the filesystem. Attempting to verify it with Passport, however, indicated that it was a modified DOS 3.3 disk and Passport could even verify all tracks higher than the damaged track 13. Furthermore, when trying to run the program in an emulator it would fail with errors like “Error xx in line yy” which seemed to indicate that it was a DOS 3.3 BASIC program underneath the copy protection. Thus, I figured that if I could find some way of getting the damaged sector to pass its checksum, then Passport would likely to able to “crack” it far enough that I could read the files and maybe figure out how to correct the damage. Because the checksums on DOS 3.3 disks are (in the standard format anyway) only one nibble long, by changing a single nibble correctly, the checksum can always be made to pass. Since there are only 64 possible valid nibbles (see Figure 13) I set about trying them one-by-one in place of the damaged “85” nibble. To do this, I brought up another copy of the disk in another disk editor window and then searched this other copy for each possible nibble in the table so that I could see what that nibble looked like and modify the fluxes in the damaged “85” nibble to match it. Once I had the “85” nibble modified for the current attempt, I would export a .woz file and try verifying the new .woz in Passport in AppleWin. On the 28th attempt, the nibble “CB” worked, and track 13 verified in Passport successfully.

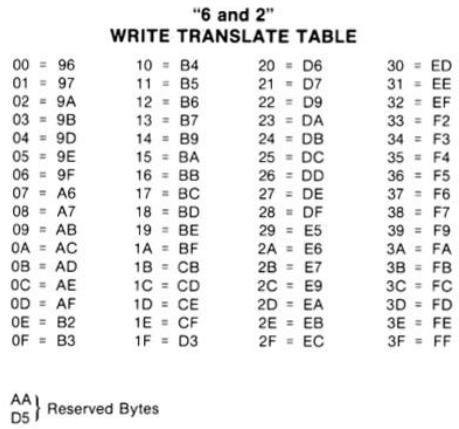

Figure 13: Table of all 64 possible valid nibbles. (Image from https://www.bigmessowires.com/2015/08/27/apple-ii-copy-protection/)

At this point in the process, the damaged section of track 13 appeared as in Figure 14 in the disk editor. I was able to export it to a .woz which is preserved as "DRUGS- Their Effects On You - Disk 1, Side A - repair attempt 3.woz" and which would verify successfully in Passport and would boot and would mostly even work. However, attempting to play through the game quickly revealed that there were still errors. Selecting "2. Choose one topic" from the main menu, then selecting "3. Depressants" and playing through the first three screens, the first quiz (alcohol is not a stimulant in case you are wondering), and then another two screens would result in a crash as shown in Figure 15.

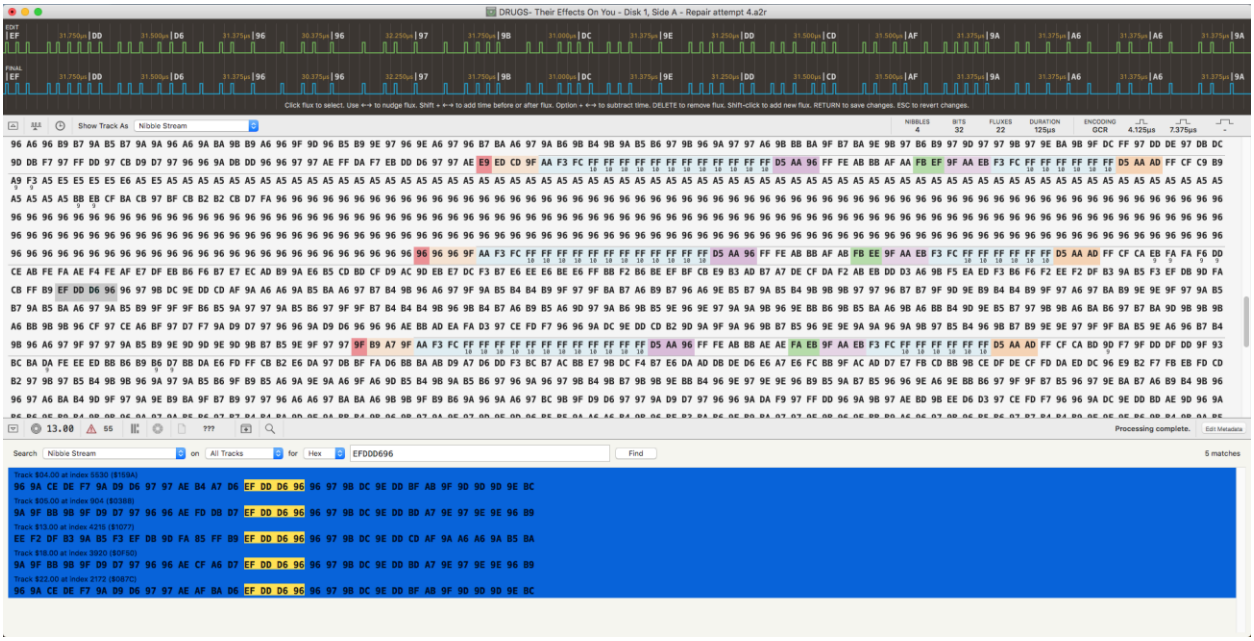

Figure 14: Damaged section of Track 13 with the unknown "85" nibble replaced with "CB" so that the image would pass verification in Passport.

Figure 15: Crash in the program when trying to play though the “Depressants” section.

From this error, it was clear to me that the program was more than likely just BASIC if the copy protection could be stripped away. Luckily, we have Passport which was able to do just that and produce a readable (and bootable) copy of the program with a standard DOS 3.3 file system. Figure 16 shows the passport output as it produced a new file called “DRUGS- Their Effects On You - Disk 1, Side A - repair attempt 3 - cracked.dsk”. With a version of the disk without copy protection now produced, I could easily read the file system in the disk analyzer. The filesystem appears as shown in Figure 17.

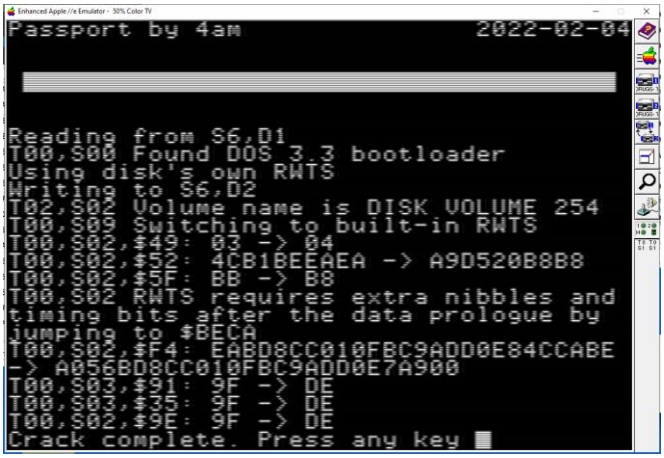

Figure 16: Passport output when producing “DRUGS- Their Effects On You - Disk 1, Side A - repair attempt 3 - cracked.dsk”

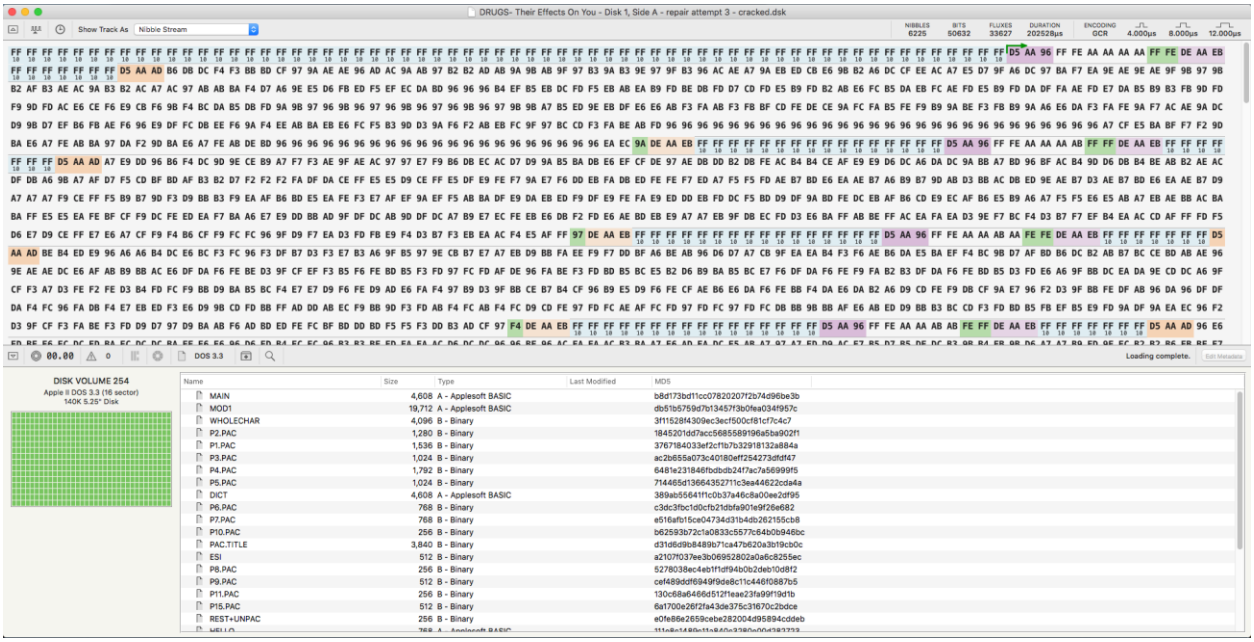

Figure 17: The filesystem on the disk is finally visible once the copy protection is stripped away by Passport.

Knowing that the error is on line 11232, I looked for an Applesoft BASIC program with such a line number and quickly found it in the file called MOD1. What is more this file is written on the damaged area of track 13! Opening this file in the Applesoft BASIC editor included in applesauce and looking at line 11232 I could immediately see one problem. The work “risk” is corrupted to “shsk” in the text “People who consume a pint or more of alcohol over a short period of time *risk* going into a coma…”. The second problem, why it was giving an error, was not as immediately apparent. However, looking closely at the GOSUB statement on that same line shows that it is attempting to GOSUB to a non-existent line 1024. By comparing to similar statements, this was almost certainly supposed to be line 1020 or 1025. See Figure 18.

Figure 18: Listing of the MOD1 Applesoft BASIC file with the damaged line 11232 highlighted.

To correct these two errors (the corruption of “risk” and the line 1024 instead of 1020/1025) in the original disk image was going to require knowing how these changes affected the actual nibbles on the disk. Thus, I modified the file and saved it back to the disk so that I could compare the corrected version with the original and thus know how to manually fix the original disk image. Since Applesauce does not have a feature to edit the file directly, I used AppleWin the do this. First, I started to boot the “DRUGS- Their Effects On You - Disk 1, Side A - repair attempt 3 - cracked.dsk” but then did a control-reset (control-F2 in AppleWin) as soon as it showed a “]“ prompt in the lower left corner. The timing was critical. If I waited too long it would have already started to load the main program and the protection code would kick in and reset the emulator instead of giving me a BASIC prompt. If I did it too soon, for some reason it would freeze when I tried to execute “LOAD MOD1”. However, I eventually did it and was able to load the program and edit the offending line 11232. See Figure 19. This first attempt at a manually corrected cracked version is preserved as “DRUGS- Their Effects On You - Disk 1, Side A - repair attempt 3 - cracked and fixed.dsk”

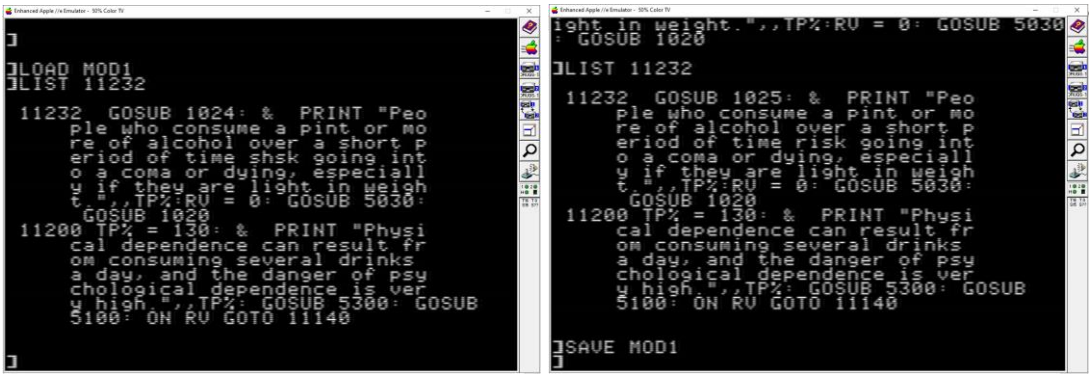

Figure 19: Correcting the damaged line 11232 in the MOD1 program. Left: Listing before corrections; Right: Listing after first round of corrections.

With these corrections made, the program would run and would not crash as in Figure 15, but something still wasn’t right. The program would not wait for the user to “Press RETURN” after displaying the message about alcohol as it should. Fixing this was going to be harder than I thought. Looking closely at the offending line and comparing it to other sections of the code, I could see that two other things didn’t look right. First, the line number was obviously wrong. It is line 11232, but it is coming before line 11200 which is not correct. Second, the GOSUB 5030 appears no where else in the code and it being wrong would explain why the program isn’t pausing after displaying the message about alcohol. To fix the first problem, I looked at the line numbering scheme used and it was pretty clear that that line was supposed to be 11160 (ten more than the line before it, 11150). To fix the second problem I noted that elsewhere in the program when it was supposed to pause for user input, the call was GOSUB 5300, not GOSUB 5030. With these changes made, the program worked much better, pausing after displaying the message, but the graphics were corrupted on that screen with part of the previous screen still being shown. Still not sure if the first GOTO was supposed to be 1025 or 1020, I tried using 1020 instead of 1025 and, woot, it all seemed to work. Figure 20 shows the final corrected version of the damaged line of code. This version of the disk is preserved as “DRUGS- Their Effects On You - Disk 1, Side A - repair attempt 3 - cracked - Copy.dsk”

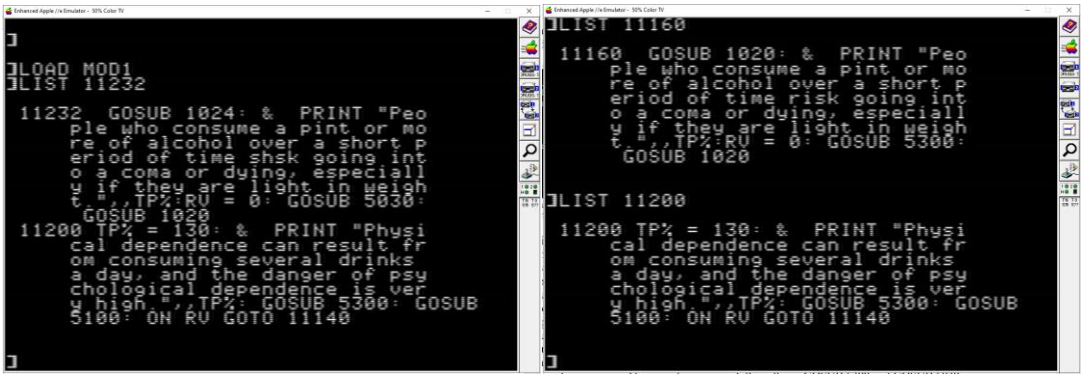

Figure 19: Correcting the damaged line 11232 in the MOD1 program. Left: Listing before corrections; Right: Listing after second and final round of corrections.

At this point I had a working and cracked version of the program. However, my goal was to repair the damage to the original image to produce an archival version of the original software. But, before these corrections could be made to the original uncracked image, another mystery had to be solved. After loading a program (ex. MOD2) from this disk in AppleWin, editing it, and saving it back, I found that the binary representation of the resaved file did not match the original as compared in applesauce. After a bit of investigation of how Applesoft Basic programs are stored in binary, I determined that the memory addresses stored in the file were different between the original and resaved versions of a program. See Figure 21. Comparing these addresses shows a constant offset of 0x3804. For example, the first address for the original program (right side in Figure 21) after correcting for little endian is 0x4011. The first address for the resaved program is 0x080D (left side in Figure 20). The difference between 0x4011 and 0x080D is 0x3804 or 14340 in decimal. Per https://retrocomputing.stackexchange.com/questions/1919/apple-ii-applesoft-basic-memory-management the address in memory that an Applesoft program is stored at can be changed by modifying the values in memory at locations decimal 103 and 104. The default location for Applesoft to save a program at is 0x0801, so if this is added to the offset of 0x3804 we get a final location of 0x4005. To make sure the program is saved at this new location we can execute POKE 104,64: POKE 103,5: POKE 16389,0 where that last POKE assures that the first address in the block of memory that will store the program is zero as is apparently required. With these steps completed, I was able to verify that I could load and resave MOD2 with no changes to the resulting disk image (except for one stray byte… I am not sure what that is about but it is in another sector away from the damaged area, so is of no concern to me for the purpose of comparing the repaired sector.)

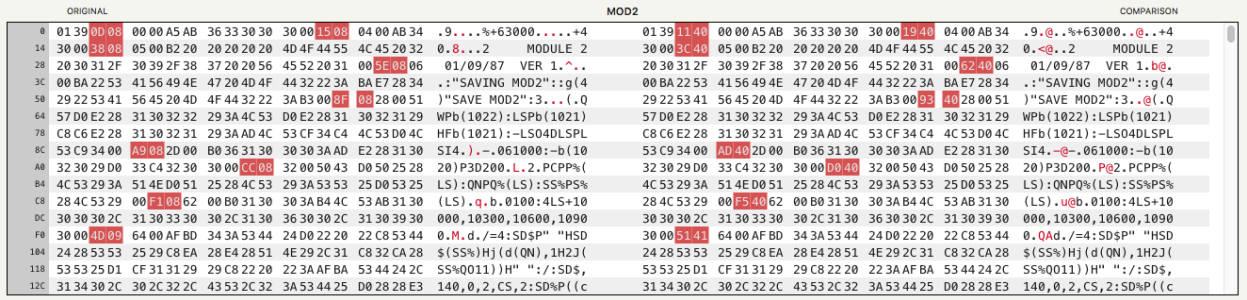

Figure 21: Memory addresses in BASIC program have changed after loading and resaving it. The left file is the resaved version, and the right file is the original one from the disk.

With a process for loading and resaving programs figured out, I could now attempt to correct the damage to MOD1 and then see if the resulting changes on the disk "made sense" in terms of where the original damage on the disk was. Note that the location where MOD1 is stored is slightly different then where MOD2 is stored – 0x4001 instead of 0x4005 so the POKE statement needs to be POKE 104,64: POKE 103,1: POKE 16385,0. With these pokes done, and MOD1 resaved, I could finally compare the flux streams for the original and the repaired images. After a bit of manual work to locate the changes on the disk and to manually compare the flux streams for the damaged and repaired sector, much to my delight all the repairs resulted in changes to only a few nibbles, and all were right on top of the damaged section! This strongly indicated that my repairs were correct as incorrect repairs would have likely resulted in changes far away from the damaged section. See Figure 22.

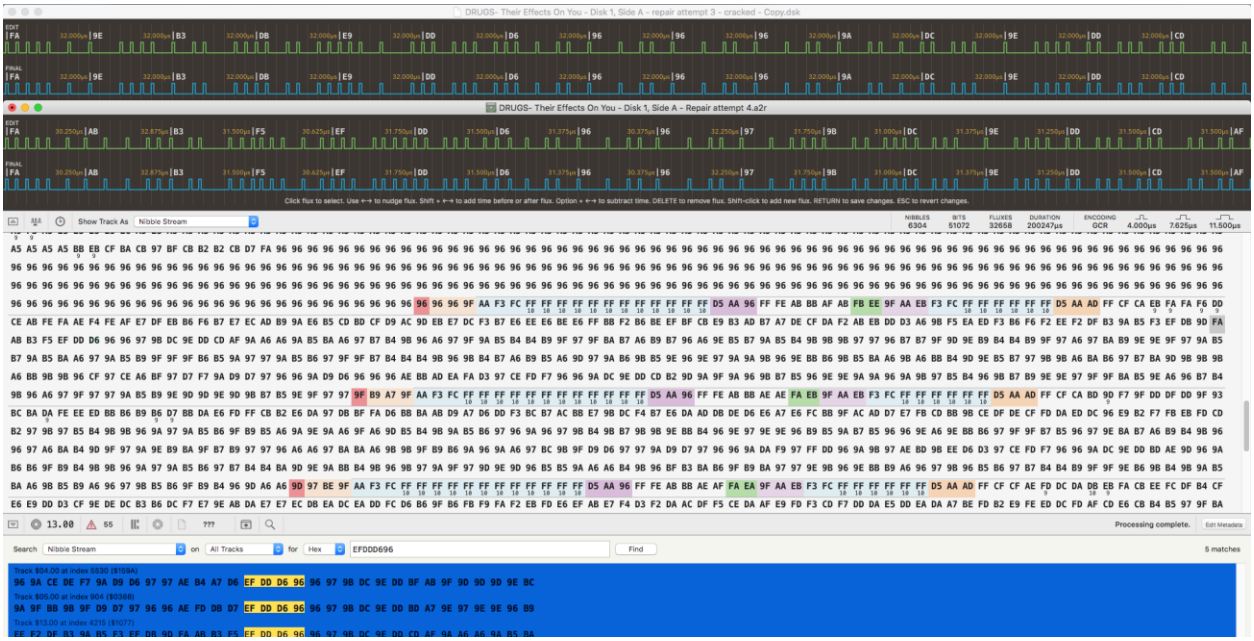

Figure 22: Comparison of the corrected and original versions of the damaged area of the disk. The upper flux stream diagram is for the repaired version and the lower is for the original version. Note that the "original version" shown here already has had the "shsk / risk" corruption repaired. Note that the 5th nibble "EF" here is the first nibble shown in Figure 14.

With the repairs made in the flux editor, I saved the .a2r and exported a .woz. Nervously, I loaded up Passport in AppleWin and did a verification. Watching the track indicator in AppleWin, I let out a cheer when it passed Track 13 without any errors! My repairs had resulted in a valid checksum – another indication that they are correct! Finally, I restarted the emulator to boot the game. Woot again! I could play though the "Depressants" module with no errors at all.

The final repaired image files are "DRUGS- Their Effects On You - Disk 1, Side A – Fixed.a2r" and "DRUGS- Their Effects On You - Disk 1, Side A – Fixed.woz". Enjoy!