Reversing Awave Audio 6.0. and Awave Studio 6.0

Written by hobgoblin

Greetings all reversers,
A couple of days ago I came across a couple of programs that I think can be interestings targets for newbies. It certainly was for me. They proved to be a little bit harder than I anticipated. And that's cool.
I was primarely looking for a program that could batch convert audio files into different formats, and after a search I found Awave Audio 6.0. At the same time I found Awave Studio 6.0, which in fact is a bigger suite that includes Awave Audio 6.0. But their shareware protection is somewhat different.After using Awave Audio for some time, I decided to try to reverse it. (It is a very cool program for the purpose of reformatting audio files, it covers a variety of different formats you seldom see covered by other programs. If you like it, I will strongly suggest that you buy it. If you read this tutorial for the purpose of ripping of the program/programmer, please do something else. Go visit a warez site or whatever. People who makes cool programs (and especially programs that's hard to crack) deserves respect.).

Now, this tutorials was written for newbies. That's why it's quite extensive and detailed.
(l guess Lord Soth will call it a recepie. I can already imagine hearing him laughing.:-). So if some of you more experienced reversers read this, please bear over with me,. Or go do something else. I'm quite sure you're busy…:-)). The first part deals with how I reversed Awave Audio 6.0, the second part is about Awave Studio 6.0.

The purpose of this tutorial is to share some knowledge about packed targets, filesize protection and how using several tools together can be very useful.

Tools: Softice, W32dasm, Procdump, Hexworkshop, Getype, Exescope, patience…

The targets: Awave Audio 6.0 and Awave Studio 6.0 – which can be found at www.softseek.com

The purpose of Awave Audio is to convert audio files into various other formats. It can batch format files. It's a small, but powerful program. When you run the shareware version, you first see a huge nag screen. It offers several different options (as buttons): Registration, online information, entering of a valid code/key, and to continue. The time to get the whole nag screen operational increases as the trial days goes by. So you can only image how long you will have to wait after 365 days. By pushing continue, you'll see the main dialogbox/screen. When you enter several files, and try to batch convert them, you will see a shareware nag stating that only one file can be converted at a time in the shareware version. These two things is the shareware limitations (as far as I could see).

I first tried to sniff out the code/code checking routine, but without success. I found the routine after some time, and by trying some creative patching, I was able to get the program to come up with the Thank you for registering.. message. But it wasn't permanent. Besides, this program doesn't require just an ordinary serial, it requires a key/reg file. So instead of pursuing a key/reg file reversing solution, I decided to try to

patch the program so it worked as a registered program. And frankly, I most often find this just as challenging as figuring out the code.

Now, let's take a closer look at the program. I usually opens up the program in hexworkshop as the first thing, just to take a look at the header information. And by scrolling down a few lines I saw among others (data, text, rsrc) the word aspack. Okey, so it's packed. Exit hewxworkshop, and open up Gettype. No respons there, either by using the GUI or checking the program in DOS. So, I decided to try using Procdump(I have just written in the new script for Aspack 2000), but it failed too. Well, then I had to unpack it manually. First I used Procdump's PE editor to change the section characteristics from C0000040 to E0000020. (If you don't change this, Softice will not be able to break at program entry). Exit procdump and open up the Softice loader. After entering the program, and after Softice broke, I traced my way down to the first "popad"-instruction I saw. When you trace down in the code of a program packed with Aspack, just go on until you reach the popad instruction. When it comes to Aspack 2000, you will have to pass a few ret and jump instructions and a lot of code before you get to this code:

```
:004A64F1  61                popad
:004A64F2  7508              jnz  004A64FC
:004A64F4  B801000000        mov eax, 00000001
:004A64F9  C20C00            ret 000C
:004A64FC  68CA5A4300        push 00435ACA
:004A6501  C3                ret
```

This is the end of the unpacking routine. First of all, write down the value in the push instruction at 004A64FC. This is the adress where the program jumps to after the unpacking is done. This is also the original program entry point. For me the OEP was 435ACA.Place the cursor at the last ret instruction and type a eip, enter and then jmp eip, enter, enter again and then F5. You will by now by out of the Softice screen. Open up Procdump again, and look in the task window. Scroll down until you see Awave listed. Rightclick on the file, and choose full dump. After storing the file (preferably in the same folder as Awave was installed, but with a different name), kill the task. Then open up the PE editor again. Enter the dumped file, and when you see the header infos, change the entry point listed to the OEP you just wrote down.Save the file and exit Procdump. You now have an unpacked file with corrected header information.

But what happens when we try to run the program. Nothing. That's odd. I mean, the odd part is that we don't get any error messages. Usually, when there is something wrong with the unpacked program, we will see a messagebox stating that there is an error at this or that address, or an general protection fault occured. Well, that indicates that there might be an anti debug device built into the program code. After disassembling the program in W32dasm, I tried to search for the word SICE. No luck. And trying to put a breakpoint on the function Createfilea in Softice didn't bring up any interesting stuff either. So, I opened the Softice loader again and started to trace through the code from the programs entry point. And after a while I found this interesting code:

```
:00421AC2 A1C4F24400              mov eax, dword ptr [0044F2C4]
:00421AC7 3DDC000000              cmp eax, 000000DC
:00421ACC 7707                    ja 00421AD5
:00421ACE 3DA0000000              cmp eax, 000000A0
:00421AD3 7308                    jnb 00421ADD

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
```

```
|:00421ACC(C)
|
:00421AD5 6A00                      push 00000000
:00421AD7 FF15A8E04300              call dword ptr [0043E0A8] (Exit Process)

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00421AD3(C)
|
:00421ADD E8FE920000                call 0042ADE0
:00421AE2 84C0                      test al, al
:00421AE4 750F                      jne 00421AF5
:00421AE6 6808AF4500                push 0045AF08
:00421AEB 68B8AE4500                push 0045AEB8
:00421AF0 E81BF9FFFF                call 00421410

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00421AE4(C)
|
:00421AF5 E896FBFFFF              call 00421690
:00421AFA 5F                      pop edi
:00421AFB 33C0                    xor eax, eax
:00421AFD 5E                      pop esi
:00421AFE 59                      pop ecx
:00421AFF C21000                  ret 0010
```

The disassembled code doen't show the api calls due to the fact that when I disassembled the unpacked program in W32dasm, the imports didn't show properly. But I'll write in the text where it is needed. Look at address 00421AD7. When I traced through the code this call was executed and the program exited. That's why nothing happened when we tried to run the program. When I traced through the code once more, and forced the program not to jump at 00421ACC, it started as usuall and I could see the nag screen. Now, at 00421AC2 the value stored in [0044F2C4]is moved into eax, and at the next instruction that value is compared with the hex value DC. Wonder what value was stored in [0044F2C4], and where this happened? Put a bmbp 0044F2C4 in Softice and try to run the program and you'll see. Softice breaks two times, but the two first places don't show any interesting code. But the third time Softice breaks we end up here at adress 0042182A:

```
...
...
:00421821 6A00                    push 00000000
:00421823 56                      push esi
:00421824 FF153CE14300            call dword ptr [0043E13C] (GetFileSize)
:0042182A C1E80A                  shr eax, 0A (We end up here..)
:0042182D 56                      push esi
:0042182E A3C4F24400              mov dword ptr [0044F2C4], eax
:00421833 FF1548E14300            call dword ptr [0043E148]
:00421839 FF1584E04300            call dword ptr [0043E084]
:0042183F 50                      push eax
:00421840 E87BFEFFFF              call 004216C0
:00421845 8B3D3CA04500            mov edi, dword ptr [0045A03C]
:0042184B 83C9FF                  or ecx, FFFFFFFF
:0042184E 33C0                    xor eax, eax
:00421850 F2                      repnz
:00421851 AE                      scasb
...
...
```

The function GetFileSize returns the size of a specified file in bytes. When the program returns from the call at 00421824, it has the filesize stored in eax. At the next instruction, shr eax, 0A, that value is divided by 2 ten times.(The instruction SHR eax means that the eax value is divided by 2, and here that instruction is executed 0A times).The result is then stored in [0044F2C4] a couple of instructions later. This stored value is then put back in eax at the address 00421AC2, and that value is then compared to the hex value DC in the following instruction. This means that when the program is packed the ja instruction at 00421ACC will not be executed (the stored value is lower than 000000DC). But if the program is unpacked, it will jump and then exit. Now, just change the

```
:00421ACC 7707     ja 00421AD5
```

to

```
:00421ACC EB00     jmp 00421ACE
```

and the problem is solved, and we can go on reversing the program.

The next thing is the shareware nag screen. How can we get rid of this one? Well, it took me some time to figure out what to do. Going through the deadlisted code in W32dasm didn't help too much. I couldn't find any references to the text displayed in the nag. Putting bpx's at DialogBoxParama, destroywindow, LoadBitmapA,LoadImageA and a couple of others didn't help either. I just ended up in an endless tracing. So I had to try something else. I opened Exescope and ran the program in there. That gave me the clue I needed. When I checked the dialogboxes, I found the dialog box that shows the nag screen. The dialogbox ID number was 250 (decimal), which is FA in hexdecimal. Back in W32dasm I did a search for this dialogbox ID and found it here:

```
* Possible Reference to Dialog: DialogID_00FA
                                |
:0042AE50 C744243CFA000000      mov [esp+3C], 000000FA
:0042AE58 C7442448C0024200      mov [esp+48], 004202C0
:0042AE60 E8BB8AFFFF            call 00423920
:0042AE65 F6D8                  neg al
:0042AE67 1BC0                  sbb eax, eax
:0042AE69 B90A000000            mov ecx, 0000000A
:0042AE6E 8D742430              lea esi, dword ptr [esp+30]
:0042AE72 8D7C2458              lea edi, dword ptr [esp+58]
:0042AE76 F7D8                  neg eax
:0042AE78 F3                    repz
:0042AE79 A5                    movsd
:0042AE7A 89442424              mov dword ptr [esp+24], eax
:0042AE7E B90A000000            mov ecx, 0000000A
:0042AE83 8D742430              lea esi, dword ptr [esp+30]
:0042AE87 8DBC2480000000        lea edi, dword ptr [esp+00000080]
:0042AE8E 8D442408              lea eax, dword ptr [esp+08]

* Possible Reference to Dialog: DialogID_00FC
                                |
:0042AE92 C7442464FC000000      mov [esp+64], 000000FC
:0042AE9A C74424700109C4200     mov [esp+70], 00429C10
:0042AEA2 50                    push eax
:0042AEA3 F3                    repz
:0042AEA4 A5                    movsd

* Possible Reference to Dialog: DialogID_00FD
```

```
                                    |
:0042AEA5 C7842490000000FD000000  mov dword ptr [esp+00000090], 000000FD
:0042AEB0 C784249C000000309F4200  mov dword ptr [esp+0000009C], 00429F30
:0042AEBB FF1534E04300            call dword ptr [0043E034](PropertySheet)
:0042AEC1 85C0                    test eax, eax
:0042AEC3 7D0F                    jge 0042AED4

...
...
```

When the call at 0042AEBB is executed the program starts (and in the evaluation version the nagscreen pops up). But look at this code:

```
:0042AE50 C744243CFA000000        mov [esp+3C], 000000FA
```

The dialogbox ID is moved into the stack. Then the program knows what dialogbox to display. Now, change this to:

```
:0042AE50 C744243C00000000        mov [esp+3C], 00000000
```

When you run the program now it starts without showing the nag screen. And we don’t even have to bother about the timedelay that was supposed to be there in the shareware version. By changing FA to 00, the program pushes the dialogbox ID 00, which don’t exist, into the stack. Somehow it continues as it was normal. I don’t exactly know the theories behind this, but it works. I have tried this with success several times before. It’s a nice way to remove nag screens, don’t you think?

The next thing is the shareware limitation on how many files the program can convert. By default the evaluation version only converts the first file of all the files you list. When you try to convert more than one file, a nag box pops up saying something like «The shareware version only converts the first...» or something like that. Put a bpx messageboxa in Softice and try to convert more than one file. When Softice breaks, hit F11, enter and you should be back in Softice where you can see this code:

```
...
...
:004296B5 6A00                    push 00000000
:004296B7 6801040000              push 00000401
:004296BC 687B010000              push 0000017B
:004296C1 50                      push eax
:004296C2 FF15BCE14300            call dword ptr [0043E1BC]
:004296C8 84DB                    test bl, bl
:004296CA 742A                    je 004296F6
:004296CC 83FE01                  cmp esi, 00000001
:004296CF 7E25                    jle 004296F6
:004296D1 8B953E010000            mov edx, dword ptr [ebp+0000013E]
:004296D7 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"Note!"
                                    |
:004296D9 68B4304500              push 004530B4

* Possible StringData Ref from Data Obj ->"This unregistered version of
Awave "
                                        ->"Audio will only convert the
first "
                                        ->"file in the input list..."
                                    |
```

```
:004296DE 6854304500              push 00453054
:004296E3 52                      push edx
...
...
```

When the program comes to address 004296CC, the number of files you have listed in Awave, and are ready to convert, is stored in esi. The program compares that number with 1, and if it's more than one, the next jump instruction is not executed, and you'll get the nag box. Why not change the 00000001 to something else? I changed

```
:004296CC 83FE01                  cmp esi, 00000001
```

to

```
:004296CC 83FE96                  cmp esi, 00000096
```

By doing this I can list up to 150 files for conversion before the nag box pops up and the program halts. Nice, isn't it?

Is it more to do? Well, if you check the main window in Awave when you run it you will see the word Unregistered. Why not remove that one too?

I opened W32dasm, and after a search I found this text string:
«Awave Audio (unreg.)»
When I found the textstring in the disassembled body, I also saw this :

```
...
...
:00421F66 B910A14500              mov ecx, 0045A110
:00421F6B E8B0170000              call 00423720
:00421F70 84C0                    test al, al

* Possible StringData Ref from Data Obj ->"%s - Awave Audio (unreg.)"
                                  |
:00421F72 B818F54400              mov eax, 0044F518
:00421F77 7505                    jne 00421F7E

* Possible StringData Ref from Data Obj ->"%s - Awave Audio v6.0"
                                  |
:00421F79 B800F54400              mov eax, 0044F500
...
...
```

By trying it out in Softice I found out that if the call at 00421F6B returns the value 1 in al (the lower part of the eax register), the unreg. text is displayed. So why not make sure that the call always returns the value 0 in al? Enter the call and scroll down to the bottom and you'll see this code:

```
...
...
:00423756 53                      push ebx
:00423757 E8E4F70000              call 00432F40
:0042375C 85C0                    test eax, eax
:0042375E 0F95C0                  setne al

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042373A(C)
|
:00423761 5F                      pop edi
```

```
:00423762 5E                        pop esi
:00423763 5B                        pop ebx
:00423764 C3                        ret
...
...
```

If you change the 0F95C0 setne al instruction at address 0042375E to 0F94C0 sete al, you will make sure that the returned al value always is set to 0. This routine is called from five different places in the program, and all of them deal with text displayed. (Unregisterd vs. Registered) and so on.

That should be it. We should by now have a fully functional program that says «Registered to:» in the main window.

Let us move over to Awave Studio 6.0, which proved to be more complicated.

I don't know all of what this program is capable of doing. I haven't used it. But it incorporates the conversion functionality from Awave Audio. In fact, it seems to me that Awave Audio is an integrated part of this program. Anyhow, it has the same kind of protection, but it uses it in a different way.
This program is also packed with Aspack. And not only once, but twice. So as a start, we have to unpack it uses the same method I outlined above. You just have to do it two times. After doing that, we have an unpacked file that can be run as normal.
And then the fun begins.
Let us concentrate on the opening nag screen first. This time I tried a bpx dialogboxparama in Softice first. That proved to be a tenstrike this time. Softice broke. After hitting F11 the nag screen appeared. I pushed the Continue button, and Softice broke back. I was then at a ret function. After hitting F10 once, I ended up at 0045714B. Let us take a look:

```
..
..
:00457136 E825230000                call 00459460
:0045713B 84C0                      test al, al
:0045713D 740C                      je 0045714B
:0045713F 8B0D44FA4B00              mov ecx, dword ptr [004BFA44]
:00457145 51                        push ecx
:00457146 E8A5F6FFFF                call 004567F0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045713D(C)
|
:0045714B A140E94B00                mov eax, dword ptr [004BE940]
:00457150 8BF5                      mov esi, ebp
:00457152 3BC5                      cmp eax, ebp
..
..
```

The call made at 00457146 brings up the opening nag. (You could also use Exescope as described earlier and made a search in a disassembled body afterwards. You will then end up in the call that is made from here). Take a look at the call made from 00457136. If that call returns the value of zero in al, the program jumps and goes on without showing any nag screen. If you enter that call and scroll down to the end, you'll see this:

```
..
..
:00459497 E824390300              call 0048CDC0
:0045949C 85C0                    test eax, eax
:0045949E 0F95C0                  setne al

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045947A(C)
|
:004594A1 5F                      pop edi
:004594A2 5E                      pop esi
:004594A3 5B                      pop ebx
:004594A4 C3                      ret
```

If you change the instruction at 0045949E to 0F94C0 sete al, you will never see the opening nag again. And not only that. This routine are called 14 times (or can be called) during runtime. If you check out all the places that call this code, you will see that it exclusively deals with shareware limitations, including the timelimit. So, a patch here will take care of most of the limitations. But not all. As I checked out the disassembled body for other text references, I came across this:

```
:0045ED4A 56                      push esi
:0045ED4B B910EA4B00              mov ecx, 004BEA10
:0045ED50 E80BA9FFFF              call 00459660
:0045ED55 84C0                    test al, al
:0045ED57 752A                    jne 0045ED83
:0045ED59 8B3514914900            mov esi, dword ptr [00499114]

* Possible StringData Ref from Data Obj ->"EditWave"
                                   |
:0045ED5F 68140A4B00              push 004B0A14
:0045ED64 68CCF74B00              push 004BF7CC
:0045ED69 FFD6                    call esi

* Possible StringData Ref from Data Obj ->"Sorry, this function is only
available "
                                         ->"in the registered version!"
                                   |
:0045ED6B 68D0094B00              push 004B09D0
:0045ED70 681CF84B00              push 004BF81C
:0045ED75 FFD6                    call esi
:0045ED77 32C0                    xor al, al
:0045ED79 5E                      pop esi
:0045ED7A 81C410040000            add esp, 00000410
:0045ED80 C20400                  ret 0004
```

This limitation was not covered by the previous call, so I checked out the call at 0045ED50. If that call returns the value of 1 in al, it jumps over this text. Enter the call, and scroll down as before and you will see this:

```
..
..

:0045969B 53                      push ebx
:0045969C E81F370300              call 0048CDC0
:004596A1 85C0                    test eax, eax
:004596A3 0F94C0                  sete al
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045967F(C)
|
:004596A6 5F                    pop edi
:004596A7 5E                    pop esi
:004596A8 5B                    pop ebx
:004596A9 C3                    ret
```

Change the instruction at 004596A3 to OF95C0 setne al, and you're done. This routine are called five times, and one of them deals with the evaluation text in the about box. The rest is other limitations not covered by the first call.
But beware! One of the calls to this routine has to be modified in order for the program to run properly. I get back to that later.

So what's next? We must also here remove the limitation in the conversion process so we're able to batch convert files. This time you can find it for yourself. It is exactly the same protection routine as in Awave Audio.

So now what? Do we have a fully functional program by now?

No way!

Remember the GetFilesize protection we looked at in Awave Audio?
It's here too.But in a slightly different way.
As you may have noticed by now, the program can be run; it doesn't shut itself down at startup as Awave Audio did. But after using it for a while, you will get a nasty surprise. It shuts itself down without warning after a given period of time. That's not nice, is it? Especially not if you're in the middle of converting 50 audio files.

Okey. If you put a bpx getfilesize in Softice and try to run the program, you will end up here after the second break:

```
..
..
:00456CCF 55                    push ebp
:00456CD0 56                    push esi
:00456CD1 FF1504914900          call dword ptr [00499104](GetFileSize)
:00456CD7 C1E80A                shr eax, 0A
:00456CDA 56                    push esi
:00456CDB A318E34A00            mov dword ptr [004AE318], eax
:00456CE0 FF15D4914900          call dword ptr [004991D4]
:00456CE6 FF153C914900          call dword ptr [0049913C]
:00456CEC 50                    push eax
..
..
```

It's the same thing as with Awave Audio. After hitting F12, we end up here at the adress 00456F52. After scrolling down a bit we find this code:

```
..
..

:004571D6 FFD6                  call esi
:004571D8 A118E34A00            mov eax, dword ptr [004AE318]
:004571DD 3DC2010000            cmp eax, 000001C2
:004571E2 7707                  ja 004571EB
:004571E4 3D45010000            cmp eax, 00000145
:004571E9 7313                  jnb 004571FE
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004571E2(C)
|
:004571EB 68F06E4500              push 00456EF0
:004571F0 6830F20000              push 0000F230
:004571F5 689A020000              push 0000029A
:004571FA 6A00                    push 00000000
:004571FC FFD6                    call esi
..
..
```

If we change the instruction at the address 004571E2 to EB00, we get rid of the annoying exit process. At least that's what I thought. After a while it exited again. So I did a search W32dasm for [004AE318]. And quite right, I found another occurrence of the code you see from 004571D8 through 004571E9. After patching that one, it stayed stabil. I'll leave it to you to find the exact place.

Now.There is just one more thing to take care of. When I checked the program by trying out menu bars, menu commands and so on just to see how the program ran after the patching, I discovered something. When you try to open the Program setup option in the menu, the program exited. Do the programmer never give up?

After some thinking, trial and error I found the weak spot. Take a look at this:

```
..
..

:00459E58 B910EA4B00              mov ecx, 004BEA10
:00459E5D E8FEF7FFFF              call 00459660
:00459E62 84C0                    test al, al
:00459E64 7430                    je 00459E96
:00459E66 8B3DC0924900            mov edi, dword ptr [004992C0]
:00459E6C 68D8F64B00              push 004BF6D8
:00459E71 68E8030000              push 000003E8
:00459E76 56                      push esi
:00459E77 FFD7                    call edi

* Possible StringData Ref from Data Obj ->"Registered!"
                                  |
:00459E79 6888F44A00              push 004AF488

* Possible Reference to Dialog: DialogID_0027, CONTROL_ID:03E9,
"Standby..."
                                  |
:00459E7E 68E9030000              push 000003E9
:00459E83 56                      push esi
:00459E84 FFD7                    call edi
:00459E86 A0D8F64B00              mov al, byte ptr [004BF6D8]
:00459E8B 84C0                    test al, al
:00459E8D 7507                    jne 00459E96
:00459E8F 6A00                    push 00000000
:00459E91 E8BF580300              call 0048F755

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00459E64(C), :00459E8D(C)
|
```

```
:00459E96 8A153CEE4B00            mov dl, byte ptr [004BEE3C]
:00459E9C 33C0                    xor eax, eax
:00459E9E 8B3DB8924900            mov edi, dword ptr [004992B8]
..
..
```

Take a look at
```
:00459E5D E8FEF7FFFF              call 00459660
```

This is a call to the routine we just recently patched to always return the value 1 in al. Well, if it does that here, the program eventually ends up at 00459E91 where a call to 0048F755 is executed. When that happens, the program exits. So to get around this, just change the conditional jump at 00459E8D to an unconditional jump. Problem fixed.

How did I find this? By repatching my patching of the program step by step until I saw the menu worked as it supposed to. Then I knew where to start my checking.

Well, that should be it. When I try to run the program now, it seems like we have a fully functional program. But as I said, I'm no expert on this program. So if anybody who read this find some error in my patching, please let me know.

Greetings goes out to all the regulars at +Sandmans board, you know who you are.☺

And remember, if you want to keep your knowledge of reversing, just pass it on..

Januar 2000.

Hobgoblin

I can be reached at: the.hobgoblin@moss.online.no

Or you can just post a message at+Sandmans messageboard for newbies.